DETERMINATION OF EIGENSOLUTIONS OF ARBITRARY MATRICES USING SIMULTANEOUS ITERATION ACCELERATED BY JACOBI-LIKE METHODS.

A thesis submitted at the
University of London for the
degree of Doctor of Philosophy
(Computer Science)

September 1974 B.C. Gudgin

ABSTRACT

ABSTRACT

The direct methods of obtaining the eigenvalues and eigenvectors of a matrix, such as the QR algorithm of Francis, are certainly to be recommended when it is necessary to compute all the eigenvalues and corresponding eigenvectors of relatively small matrices. However, for larger matrices iterative techniques may be the only feasible methods. Iterative methods particularly come into their own when:

1) the required number of eigensolutions is substantially smaller than the dimension of the matrix,

2) initial estimates of the eigenvectors are available,

3) the matrix is sparse.

It is often the case that many technical problems give rise to very large sparse matrices. The author has been involved in marine engine vibration problems and this gave rise to an interest in the methods of obtaining eigensolutions of the matrices involved. It is also usually the case that only a few eigenvalues and eigenvectors need to be determined accurately and that experience with similar problems enables good initial approximations to the eigenvectors to be made. Hence we see that these are ideal conditions in which to use iterative methods.

The best known iterative method is the power method in which a trial vector is continually premultiplied by the matrix until the iterates become proportional to each other. This process can often yield an eigenvector in a very short time but this cannot be guaranteed even with improvements such as shift of origin and acceleration techniques. To overcome possible poor convergence the computation is applied to general iteration vectors between which an orthogonality or biorthogonality relation is maintained. Such methods and the developments thereof are the subject of this thesis.

CONTENTS

CONTENTS

ACKNOWLEDGEMENT

ACKNOWLEGEMENT

The author gratefully acknowledges the help and guidance so freely given in the writing of this thesis by Dr. K.E. Pitman of the Institute of Computer Science and of the Imperial College of Science and Technology and also by Mr. D.G. Davies of the Polytechnic of Central London. He acknowledged also his indebtedness to the staff of the Computer Centre, Queen Mary College for their courteous and generous assistance.

In quoting from a number of papers in French and German the author lays no claim to any ability to read them and frankly confesses that they were translated for him by his father, to whom he is grateful.

For such imperfections as the reader may find the author alone takes responsibility; he only hopes that they will not be too numerous.

CHAPTER 1

PRELIMINARIES

1. INTRODUCTION

In this brief introduction we give many of the definitions and fundamental results upon which the following chapters are based. In general, proofs are given only if they are pertinent to later results.

The (i,j) element of a matrix A will be denoted by a_{ij}. Vectors will be represented by lower case letters; we shall very frequently be concerned with systems of vectors which will be denoted by $x_1, x_2, \ldots, x_n$. We shall refer to the matrix having x_i as its i-th column as X. Matrices of eigenvectors will usually be denoted by the letters U,V,W,X or Y. In particular, matrices of right-hand eigenvectors will be denoted by X and matrices of left-hand eigenvectors by Y.

The notation $|A|$ is reserved exclusively to denote the matrix the elements of which are $|a_{ij}|$. The determinant of a matrix is represented by det(A) and the norm of a matrix by $\|A\|$.

The conjugate of the matrix $A=(a_{ij})$ is represented by $\bar{A}=(\bar{a}_{ij})$. The transpose of A is denoted by A^T and is such that its (i,j) element is equal to a_{ji}. Similarly, the Hermitian transpose of A is denoted by A^H and is such that its (i,j) element is equal to $\bar{a}_{ji}$.

A diagonal matrix with the (i,i) element equal to λ_i will be denoted by $\mathrm{diag}(\lambda_i)$ or, if no confusion can arise, Λ or D may be used to represent diagonal

matrices.

2. DEFINITIONS

The fundamental algebraic eigenproblem with which we are concerned is determining some or all of those values λ for which the set of n homogeneous linear equations in n unknowns

$$(A - \lambda I)x = 0 \tag{2.1}$$

has a non-trivial solution. Equation (2.1) may be rewritten as

$$Ax = \lambda x. \tag{2.2}$$

The theory of simultaneous linear algebraic equations shows that there is a solution, other than the trivial $x=0$, if, and only if, the matrix $(A-\lambda I)$ is singular. That is

$$\det(A - \lambda I) = 0. \tag{2.3}$$

The polynomial $f(\lambda)=\det(A-\lambda I)$ is called the characteristic polynomial and the equation $f(\lambda)=0$ is called the characteristic equation of A.

Theorem 2.1 The characteristic polynomial of a matrix of order n is a polynomial of degree n with leading coefficient $(-1)^n$; i.e.,

$$f(\lambda) = a_0 + a_1\lambda + \ldots. + a_{n-1}\lambda^{n-1} + (-1)^n\lambda^n. \tag{2.4}$$

If the n solutions of

$$f(\lambda) = 0 \tag{2.5}$$

are $\lambda_1,\lambda_2,\ldots.,\lambda_n$, then

$$\lambda_1\lambda_2\ldots.\lambda_n = \det(A) \tag{2.6}$$

Proof: The proof follows by expanding $\det(A-\lambda I)$ in terms of elements in the first row. A rigorous treatment is to be found in Noble (1969).

Since the coefficient of λ^n is non-zero and we are working in the field of complex numbers the equation always has n roots. In general the roots will be complex and of any multiplicity up to n. These n roots are called the eigenvalues of the matrix A.

Corresponding to any eigenvalue λ the equation (2.2) has at least one non-trivial solution. This solution is called the eigenvector of A corresponding to the given value of λ. We refer to the pair (λ,x) as an eigensolution of the matrix A.

Theorem 2.2 (i) There exists at least one eigenvector corresponding to each eigenvalue.

(ii) The eigenvectors corresponding to a given eigenvalue constitute a vector space.

Proof: To find an eigenvector corresponding to λ_i we solve

$$(A - \lambda_i I)x = 0 \ . \qquad (2.7)$$

Since $\det(A-\lambda_i I)=0$, this is a set of n homogeneous equations in n unknowns, the coefficient matrix having rank less than n. Hence a non-zero solution exists, which gives an eigenvector. This proves (i). To prove (ii) suppose that u and v are two eigenvectors corresponding to λ_i, then

$$Au = \lambda_i u, \qquad Av = \lambda_i v \qquad (2.8)$$

so that

$$A(\alpha u + \beta v) = \alpha\lambda_i u + \beta\lambda_i v$$

$$= \lambda_i(\alpha u + \beta v) \qquad (2.9)$$

Hence, $\alpha u+\beta v$ is also an eigenvector and this proves (ii).

We note in particular that any eigenvector is arbitrary to the extent of a constant multiplier, for if

$$Ax = \lambda x \qquad (2.10)$$

then

$$A(kx) = \lambda(kx) \qquad (2.11)$$

for some scalar k. It is often convenient to choose k such that the eigenvector has a particular numerical property. We refer to such eigenvectors as normalised. The most convenient forms of normalisation are those for which

$$\text{(i)}\ x^H x = \sum_{i=1}^{n} |x_i|^2 = 1 \qquad (2.12)$$

$$\text{(ii) if } |x_i| \geq |x_j|, \quad j=1,\ldots,n$$

$$\text{then } kx_i = 1. \qquad (2.13)$$

3. EIGENSOLUTIONS OF THE TRANSPOSED MATRIX

We now consider the eigenvalues and eigenvectors of the transpose of a matrix A. By our previous definitions the eigenequation for the transpose A^T is

$$A^T z = \lambda z \qquad (3.1)$$

where we seek those values of λ for which (3.1) has a non-trivial solution. Following (2.3) these are the values for which

$$\det(A^T - \lambda I) = 0 \tag{3.2}$$

and since the determinant of a matrix is equal to that of its transpose the eigenvalues of A are the same as those of A^T. We denote the eigenvector of A^T corresponding to λ_i as z_i, so that we have

$$A^T z_i = \lambda_i z_i. \tag{3.3}$$

Note that in general $x_i \neq z_i$. Equation (3.3) may be written as

$$z_i^T A = \lambda_i z_i^T. \tag{3.4}$$

To distinguish these vectors from the vectors x_i, where

$$Ax_i = \lambda_i x_i \ , \tag{3.5}$$

the z_i^T are called the left-eigenvectors of A and the x_i the right-eigenvectors of A. If we speak of just the eigenvectors of A the meaning will be apparent from the context. These are the classical definitions as given in, for example, Wilkinson (1965) but for a lot of the work that follows it is convenient to adopt the following slightly modified definition of a left-hand eigenvector.

Instead of (3.4) we consider the equation

$$y_i^H A = \lambda_i y_i^H \tag{3.6}$$

which may be rewritten as

$$A^H y_i = \bar{\lambda}_i y_i. \tag{3.7}$$

Note that $y_i = \bar{z}_i$. We prefer to use y^H and A^H rather than y^T and A^T because we shall make extensive use of inner-products and in this case

$$(x,y) \equiv x^H y = \overline{y^H x} = \overline{(y,x)} \tag{3.8}$$

and

$$x^H x > 0 \tag{3.9}$$

for all non-zero x.

We illustrate this further by using both the classical and modified definitions to prove the following theorem.

Theorem 3.1 A right eigenvector x_i and a left eigenvector y_j corresponding to distinct eigenvalues λ_i and λ_j respectively are orthogonal.

Proof (i): We may rewrite equation (3.5) as

$$x_i^T A^T = \lambda_i x_i^T \tag{3.10}$$

and from (3.3)

$$A^T z_j = \lambda_j z_j. \tag{3.11}$$

Hence by postmultiplying (3.10) by z_j and premultiplying (3.11) by x_i^T we obtain

$$x_i^T A^T z_j = \lambda_i x_i^T z_j \tag{3.12}$$

and

$$x_i^T A^T z_j = \lambda_j x_i^T z_j. \tag{3.13}$$

Subtracting gives

$$0 = x_i^T z_j (\lambda_i - \lambda_j) \tag{3.14}$$

which proves that

$$x_i^T z_j = 0, \quad \text{if } \lambda_i \neq \lambda_j. \tag{3.15}$$

We note that as x_i and z_j are, in general, complex vectors, $x_i^T z_j$ is <u>not</u> an inner-product as is usually understood; for, in this case, we have

$$x_i^T z_j = z_j^T x_i \tag{3.16}$$

and not

$$x_i^T z_j = \overline{z_j^T x_i}. \tag{3.17}$$

We note also that if x is complex we may have

$$x^T x \leq 0. \tag{3.18}$$

In fact $x^T x$ may even be a complex number!

<u>Proof (ii):</u> We may rewrite equation (3.5) as

$$x_i^H A^H = \bar{\lambda}_i x_i^H \tag{3.19}$$

and from (3.7)

$$A^H y_j = \bar{\lambda}_j y_j. \tag{3.20}$$

Hence by postmultiplying (3.19) by y_j and premultiplying (3.20) by x_i^H we obtain

$$x_i^H A^H y_j = \bar{\lambda}_i x_i^H y_j \tag{3.21}$$

and

$$x_i^H A^H y_j = \bar{\lambda}_j x_i^H y_j \ . \tag{3.22}$$

Subtracting gives

$$0 = x_i^H y_j (\bar{\lambda}_i - \bar{\lambda}_j) \tag{3.23}$$

which proves that

$$x_i^H y_j = 0, \quad \text{if } \lambda_i \neq \lambda_j. \tag{3.24}$$

This seems to us to be rather neater than the previous result as $x_i^H y_j$ is an inner-product as usually understood. In particular

$$x_i^H y_j = \overline{y_j^H x_i} \tag{3.25}$$

and

$$x^H x > 0 \tag{3.26}$$

for all non-zero x.

4. EIGENSYSTEMS WITH DISTINCT EIGENVALUES

Firstly we consider the theory of the system of eigenvectors in the case of distinct eigenvalues. The equation

$$(A - \lambda_i I)x_i = 0 \tag{4.1}$$

certainly has at least one solution for each value of λ_i and therefore we are justified in assuming the existence of a set of eigenvectors $x_1, x_2, \ldots, x_n$.

Theorem 4.1 The eigenvectors x_i corresponding to distinct eigenvalues λ_i are linearly independent.

Proof: We assume that they are not independent and let s be the smallest number of linearly dependent vectors such that $x_1, x_2, \ldots, x_s$ are eigenvectors corresponding to distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_s$ of A. Then

$$\sum_{i=1}^{s} a_i x_i = 0 \tag{4.2}$$

and

$$a_i \neq 0, \quad i = 1,2,\ldots,s \tag{4.3}$$

Premultiplying (4.2) by A gives

$$\sum_{i=1}^{s} \alpha_i \lambda_i x_i = 0. \tag{4.4}$$

Multiplying (4.2) by λ_s and subtracting (4.4) gives

$$\sum_{i=1}^{s-1} \alpha_i (\lambda_s - \lambda_i) x_i = 0 \tag{4.5}$$

and

$$\alpha_i \neq 0 \quad \text{and} \quad \lambda_i \neq \lambda_s, \quad i = 1,2,\ldots,s-1. \tag{4.6}$$

Equation (4.5) implies that $x_1, x_2, \ldots, x_{s-1}$ are linearly dependent which is contrary to our hypothesis. Therefore there is no $s \leq n$ and hence the n eigenvectors are linearly independent and span the whole n-dimensional space. From this result we may easily prove the following theorem.

Theorem 4.2 Each of the vectors x_i is unique, apart from an arbitrary multiplier.

Proof: Suppose that corresponding to λ_1 there is an eigenvector x_1 and a second x_1'. Then we may write

$$x_1' = \sum_{i=1}^{n} \alpha_i x_i, \tag{4.7}$$

where at least one of the α_i is non-zero.

Multiplying (4.7) by A gives

$$\lambda_1 x_1' = \sum_{i=1}^{n} \alpha_i \lambda_i x_i. \tag{4.8}$$

Multiplying (4.7) by λ_1 and subtracting from (4.8) gives

$$0 = \sum_{i=2}^{n} \alpha_i (\lambda_i - \lambda_1) x_i. \tag{4.9}$$

But as the x_i are independent we must have

$$\alpha_i (\lambda_i - \lambda_1) = 0, \qquad i = 2,3,\ldots,n; \tag{4.10}$$

$$\Longrightarrow \alpha_i = 0, \qquad i = 2,3,\ldots,n \tag{4.11}$$

as the eigenvectors are distinct. However, as at least one of the α_i was non-zero it must have been α_1 showing that x_1' is a multiple of x_1. Similar results hold for the left-hand eigenvectors.

We showed earlier that

$$x_i^H y_j = 0, \qquad \lambda_i \neq \lambda_j \tag{4.12}$$

and it follows that

$$x_i^H y_i \neq 0, \qquad i = 1,2,\ldots,n. \tag{4.13}$$

If this were not so and x_i was orthogonal to y_i it would be orthogonal to $y_1, y_2, \ldots, y_n$ and hence to the whole n-dimensional space. This is not possible as we demand that x_i is not the null vector.

5. SIMILARITY TRANSFORMATIONS

If we choose the arbitrary multipliers associated with each x_i and y_j so that

$$y_i^H x_i = 1, \qquad i = 1,2,\ldots,n \tag{5.1}$$

this, together with (4.12), implies that the matrix Y^H, which has y_i^H as its i-th row, is the inverse of the matrix X, which has x_i as its i-th column. The n equations

$$Ax_i = \lambda_i x_i, \qquad i = 1,2,\ldots,n \tag{5.2}$$

may be written as

$$AX = X\mathrm{diag}(\lambda_i). \tag{5.3}$$

We have just seen that the inverse of the matrix X exists and is equal to Y^H. Hence we have

$$X^{-1}AX = Y^HAX = \text{diag}(\lambda_i) = \Lambda \tag{5.4}$$

where

$$Y^HX = X^{-1}X = I. \tag{5.5}$$

A transformation of the type $X^{-1}AX$ where X is non-singular is known as a similarity transformation. Equations (5.4) and (5.5) will be of the utmost importance in the work that follows.

6. MULTIPLE EIGENVALUES

We have just considered the case of distinct eigenvalues and we now look at the situation that arises if one or more of the eigenvalues is repeated. Unfortunately the position with respect to the eigenvectors is usually much more complicated than that outlined in sections 4 and 5. However it may still be the case that for a particular matrix A there does indeed exist a similarity transformation which reduces A to diagonal form. That is, there exists an X, implicitly non-singular, such that

$$X^{-1}AX = \text{diag}(\lambda_i) = \Lambda. \tag{6.1}$$

Lemma 6.1 The determinant of the product of two square matrices is equal to the product of the determinants, thus

$$\det(AB) = \det(A)\det(B). \tag{6.2}$$

A complete proof is to be found in Noble (1969).

Theorem 6.2 If equation (6.1) is true the λ_i are the eigenvalues of A and each λ_i occurs with the appropriate multiplicity. In addition the columns of X are the eigenvectors of A.

Proof: $X^{-1}(A - \lambda I)X = X^{-1}AX - \lambda X^{-1}IX$

$$= \text{diag}(\lambda_i - \lambda). \tag{6.3}$$

Taking determinants of both sides and using lemma 6.1 we obtain

$$\det(X^{-1})\det(A - \lambda I)\det(X) = \prod_{i=1}^{n}(\lambda_i - \lambda) \tag{6.4}$$

giving

$$\det(A - \lambda I) = \prod_{i=1}^{n}(\lambda_i - \lambda). \tag{6.5}$$

Hence, from theorem 2.1, the λ_i are the roots of the characteristic equation of A. Writing (6.1) as

$$AX = X\Lambda \tag{6.6}$$

we see that the columns of X are eigenvectors of A. Since X is non-singular, its columns are independent. Note that if λ_1 is, say, a double root then we have

$$Ax_1 = \lambda_1 x_1 \quad \text{and} \quad Ax_2 = \lambda_1 x_2 \tag{6.7}$$

where x_1 and x_2 are independent. Equations (6.7) imply that any vector in the two-dimensional sub-space spanned by x_1 and x_2 is also an eigenvector. For

$$A(\beta_1 x_1 + \beta_2 x_2) = \beta_1\lambda_1 x_1 + \beta_2\lambda_1 x_2$$

$$= \lambda_1(\beta_1 x_1 + \beta_2 x_2). \tag{6.8}$$

It is the case that for any matrix which can be reduced to diagonal form by a similarity transformation and which has multiple eigenvalues of multiplicity m, say, that there is a certain amount of indeterminacy associated with the corresponding m eigenvectors. However, it is always possible to select m vectors which span the m-dimensional subspace and thus it is always possible to choose the complete set of eigenvectors to span the whole n-dimensional space.

We showed in theorem 2.2 that there exists at least one eigenvector corresponding to each eigenvalue and thus far we have only considered the case of m eigenvectors corresponding to an eigenvalue of multiplicity m. We now consider when this is not the case.

7. SIMPLE JORDAN SUBMATRICES

We consider the following very simple example. Let

$$A(a,b) = \begin{pmatrix} a & \varepsilon \\ 0 & b \end{pmatrix}, \qquad \text{where } b \neq a. \tag{7.1}$$

This matrix has the two eigenvalues a and b and the corresponding eigenvectors are

$$\begin{bmatrix} \varepsilon \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \varepsilon \\ b - a \end{bmatrix}.$$

As $b \to a$ the eigenvalues become closer and the eigenvectors more and more parallel. In the limit b=a and we have an eigenvalue a of multiplicity two and

corresponding to it only one eigenvector. The matrix $A(a,a)$ is usually denoted by $C_2(a)$ and in general we define

$$C_1(a) = (a)$$

and

$$C_s(a) = \begin{bmatrix} a & \varepsilon & & & & \\ & a & \varepsilon & & & \\ & & a & \varepsilon & & \\ & & & \cdots & & \\ & & & & a & \varepsilon \\ & & & & & a \end{bmatrix}, \quad \text{for } s > 1 \tag{7.2}$$

where C_s is of order s.

The matrix $C_s(a)$ is normally defined to have $\varepsilon=1$ (see, for example, Wilkinson (1965)) but we feel this to be too specific a choice for our application as will be shown later.

Theorem 7.1 The matrix $C_s(a)$ has an eigenvalue a of multiplicity s but corresponding to these eigenvalues there is only one eigenvector, namely $x=e_1$ where we use e_i to denote the i-th column of the identity matrix.

Proof: Consider the set of equations

$$(C_s(a) - aI)x = 0, \tag{7.3}$$

i.e.,
$$\left.\begin{aligned} 0x_1 + \varepsilon x_2 &= 0 \\ 0x_2 + \varepsilon x_3 &= 0 \\ 0x_{s-1} + \varepsilon x_s &= 0 \\ 0x_s &= 0 \end{aligned}\right\} \tag{7.4}$$

These equations have only the one solution $x_2=x_3=\ldots=x_s=0$ with x_1 arbitrary. Hence e_1 is the

only eigenvector of $C_s(a)$.

Theorem 7.2 The matrix $C_s(a)$ $(s>1)$, cannot be reduced to diagonal form by a similarity transformation.

Proof: Suppose there exists a non-singular X for which

$$X^{-1}C_sX = \text{diag}(\lambda_i), \tag{7.5}$$

that is

$$C_sX = X\Lambda, \tag{7.6}$$

then as we have shown in section 6, the λ_i must be equal to the eigenvalues of $C_s(a)$ and therefore we must have

$$\lambda_i = a, \quad i = 1,2,\ldots,s. \tag{7.7}$$

Equation (7.6) then shows that the columns of X are all eigenvectors of $C_s(a)$ and these columns must be independent. The hypothesis that such an X exists is therefore false and the theorem is proved.

The matrix $C_s(a)$ is of a special type which plays a major role in the theory of the eigenproblem.

8. JORDAN CANONICAL FORM

The matrix $C_s(a)$ of the previous section is called a simple Jordan Submatrix of order s. A block diagonal matrix consisting of only simple Jordan submatrices such as

$$C = \begin{bmatrix} C_3(\lambda_1) & & & & \\ & C_3(\lambda_2) & & & \\ & & C_2(\lambda_2) & & \\ & & & C_2(\lambda_2) & \\ & & & & C_1(\lambda_3) \end{bmatrix} \qquad (8.1)$$

is referred to as a Jordan canonical form and C is said to be the direct sum of the simple Jordan submatrices. The importance of the Jordan canonical form is shown by the following fundamental theorem.

Theorem 8.1 Let A be a matrix of order n with r distinct eigenvalues $\lambda_1,\lambda_2,\ldots,\lambda_r$ of multiplicities $m_1,m_2,\ldots,m_r$ where, obviously,

$$\sum_{i=1}^{r} m_i = n. \qquad (8.2)$$

Then there exists a similarity transformation such that

$$X^{-1}AX = C. \qquad (8.3)$$

The sum of the order of the submatrices associated with λ_i is equal to m_i and, apart from the ordering of the submatrices along the diagonal, the transformed matrix C is unique. Although this theorem is of fundamental importance the proof makes little use of the techniques with which we shall be later concerned but full details are to be found in Noble (1969).

From this theorem and the results of section 7 we see that the total number of eigenvectors of a matrix A is equal to the number of simple submatrices

in the Jordan canonical form. Thus the matrix C defined in equation (8.1) has five eigenvectors, namely e_1, e_4, e_7, e_9 and e_{11}. The eigenvectors of A are given by Xe_1, Xe_4, Xe_7, Xe_9 and Xe_{11}. Note that in this example λ_1 is an eigenvalue of multiplicity three and has one eigenvector; λ_2 is of multiplicity seven and has three eigenvectors and finally λ_3 is an isolated eigenvalue.

We see also that although we defined the elements of the super diagonals of simple Jordan submatrices to be 'ε' it is possible by a suitable similarity transformation to give these elements any non-zero value. Furthermore the matrices of the similarity transformation are diagonal. For example, take

$$C_4 = \begin{bmatrix} a & \varepsilon & & \\ & a & \varepsilon & \\ & & a & \varepsilon \\ & & & a \end{bmatrix} \quad \text{and} \quad X = \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & x_3 & \\ & & & x_4 \end{bmatrix}. \tag{8.4}$$

Hence

$$X^{-1}C_4X = \begin{bmatrix} 1/x_1 & & & \\ & 1/x_2 & & \\ & & 1/x_3 & \\ & & & 1/x_4 \end{bmatrix} \begin{bmatrix} a & \varepsilon & & \\ & a & \varepsilon & \\ & & a & \varepsilon \\ & & & a \end{bmatrix} \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & x_3 & \\ & & & x_4 \end{bmatrix}$$

$$= \begin{bmatrix} a & \varepsilon x_2/x_1 & & \\ & a & \varepsilon x_3/x_2 & \\ & & a & \varepsilon x_4/x_3 \\ & & & a \end{bmatrix} \tag{8.5}$$

Notice that by a suitable choice of the x_i we can

theoretically make $X^{-1}CX$ arbitrarily close to (but never equal to as shown in theorem 7.2) a diagonal matrix.

Finally we note that if the Jordan canonical form of a matrix consists only of submatrices of order one then the matrix can be diagonalised by a similarity transformation.

9. ELEMENTARY DIVISORS

Let C be the Jordan canonical form corresponding to A and consider the matrix $C-\lambda I$. Defining C as in (8.1), for example, we see that

$$(C - \lambda I) = \begin{bmatrix} C_3(\lambda_1-\lambda) & & & & & \\ & C_3(\lambda_2-\lambda) & & & & \\ & & C_2(\lambda_2-\lambda) & & & \\ & & & C_2(\lambda_2-\lambda) & & \\ & & & & \ddots & \\ & & & & & C_1(\lambda_3-\lambda) \end{bmatrix}. \tag{9.1}$$

The determinants of these submatrices of the matrix $(C-\lambda I)$ are called the elementary divisors of A. Thus, in the example of (8.1), the elementary divisors of any matrix A similar to C are

$$(\lambda_1-\lambda)^3,\ (\lambda_2-\lambda)^3,\ (\lambda_2-\lambda)^2,\ (\lambda_2-\lambda)^2 \text{ and } (\lambda_3-\lambda).$$

Clearly, the characteristic polynomial of a matrix is the product of the elementary divisors. If the Jordan canonical form is diagonal we see that the elementary divisors must be linear. We have already seen that a matrix with distinct eigenvalues must

have linear elementary divisors but if it has one or more multiple eigenvalues it may or may not have linear elementary divisors. If a matrix A has one or more non-linear elementary divisors then one or more of the simple Jordan submatrices is of order two or more and hence A has less than n independent eigenvectors. A matrix with fewer than n independent eigenvectors is said to be defective.

10. DEROGATORY MATRICES

A matrix is said to be derogatory if there is more than one Jordan submatrix (and therefore more than one eigenvector) associated with λ_i for some i. Conversely a matrix is said to be non-derogatory if there is only one Jordan submatrix (and hence only one eigenvector) associated with each distinct λ_i for some i. A very thorough and readable treatment of derogatory matrices is to be found in Wilkinson (1965).

11. DEFECTIVE AND DEROGATORY MATRICES

In order to illustrate the four cases of matrices classified according to their defective and derogatory nature we give examples taken from Gregory (1960), to which reference may be made for further details on classification of matrices.

(i) Non-defective and non-derogatory,

$$A = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \lambda_3 & \\ & & & \lambda_4 \end{bmatrix} . \qquad (11.1)$$

This matrix has four distinct eigenvalues λ_1, λ_2, λ_3 and λ_4 and four linearly independent eigenvectors

$$\begin{bmatrix} x_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ x_2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ x_3 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 \\ 0 \\ 0 \\ x_4 \end{bmatrix} .$$

(ii) Non-defective and derogatory,

$$A = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_1 & & \\ & & \lambda_3 & \\ & & & \lambda_4 \end{bmatrix} .$$

This matrix has three distinct eigenvalues λ_1, λ_3 and λ_4 with λ_1 of multiplicity two. There are four linearly independent eigenvectors where the two eigenvectors associated with λ_1 may be any two linearly independent vectors lying in the two dimensional subspace spanned by

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} .$$

The other two eigenvectors are

$$\begin{bmatrix} 0 \\ 0 \\ x_3 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 \\ 0 \\ 0 \\ x_4 \end{bmatrix}.$$

(iii) Defective and non-derogatory,

$$A = \begin{bmatrix} \lambda_1 & 1 & & \\ & \lambda_1 & & \\ & & \lambda_3 & \\ & & & \lambda_4 \end{bmatrix}.$$

Again the matrix has three distinct eigenvalues λ_1, λ_3 and λ_4 with λ_1 of multiplicity two. However there are only three linearly independent eigenvectors

$$\begin{bmatrix} x_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ x_3 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 \\ 0 \\ 0 \\ x_4 \end{bmatrix}.$$

(iv) Defective and derogatory,

$$A = \begin{bmatrix} \lambda_1 & 1 & & \\ & \lambda_1 & & \\ & & \lambda_1 & \\ & & & \lambda_4 \end{bmatrix}.$$

This matrix has only two distinct eigenvalues λ_1 and λ_4 with λ_1 having multiplicity three. Again there are only three linearly independent eigenvectors. The two eigenvectors corresponding to λ_1 may be any two linearly independent vectors

lying in the two dimensional subspace spanned by

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} .$$

The third eigenvector is

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ x_4 \end{bmatrix} .$$

12. SOME PROPERTIES OF HERMITIAN MATRICES

We have seen already the fundamental importance of similarity transformations and we now look at a special type of these transformations which plays a vital role in both the theoretical and practical aspects of the eigenproblem. Our motivation for considering these transformations comes from considering some properties of Hermitian matrices.

Lemma 12.1 The eigenvalues of a Hermitian matrix are real.

Proof: If

$$Ax = \lambda x \tag{12.1}$$

then

$$x^H Ax = \lambda x^H x. \tag{12.2}$$

Now $x^H x$ is real and positive for $x \neq 0$. Further

$$(x^H Ax)^H = x^H A^H x = x^H Ax \tag{12.3}$$

and since $x^H Ax$ is a scalar it must be real. Hence from (12.2) λ must be real. We note however that,

in general, the eigenvectors are complex. If A is real the eigenvectors are always real.

Consider now the left-hand eigenvectors of a Hermitian matrix. From (3.7) we have

$$Ay_i = A^H y_i = \bar{\lambda}_i y_i \tag{12.4}$$

and from (12.1)

$$Ax_i = \lambda_i x_i. \tag{12.5}$$

As the λ_i are real we see that $y_i = x_i$ for all i. Thus the quantities $x_i^H y_i$ which we saw to be of importance in section 3 become $x_i^H x_i$ for Hermitian matrices and it follows immediately that if a Hermitian matrix has distinct eigenvalues then its eigenvectors satisfy

$$x_i^H x_j = 0, \quad i \neq j. \tag{12.6}$$

If we normalise the x_i so that

$$x_i^H x_i = 1 \tag{12.7}$$

we see that in the equation

$$Y^H AX = \Lambda \tag{12.8}$$

we may write

$$X^H AX = \Lambda \tag{12.9}$$

where, from equations (5.4) and (5.5),

$$X^H X = XX^H = I \text{ and } X^H = X^{-1}. \tag{12.10}$$

A matrix which satisfies equation (12.10) is called a unitary matrix. A real unitary matrix is called an orthogonal matrix. We shall see in the next section that equation (12.9) in fact holds for

any Hermitian matrix A irrespective of the multiplicity of its eigenvalues. Unitary transformations have desirable numerical properties and we now consider the effect of applying a unitary transformation to a general matrix.

13. REDUCTION OF A GENERAL SQUARE MATRIX TO TRIANGULAR FORM

Before the main theorem of this section we prove two lemmas.

Lemma 13.1 If $u_1,\ldots,u_s$ is an orthonormal set of vectors of order n ($s<n$), then vectors $v_1,\ldots,v_{n-s}$ exist such that

$$Q = (u_1,\ldots,u_s,v_1,\ldots,v_{n-s}) \qquad (13.1)$$

is a unitary matrix.

Proof: Suppose $w_1,\ldots,w_n$ are any linearly independent vectors of order n. Consider the set of n+s vectors

$$(u_1,\ldots,u_s,w_1,\ldots,w_n).$$

We reduce this to a linearly independent set by an accept or reject procedure. The vectors $u_1,\ldots,u_s$ we know are linearly independent; w_1 may or may not be independent of them. If it is we shall include it but let us assume that it is dependent and is not therefore included. We shall now look to see whether w_2 is or is not independent of $u_1,\ldots,u_s$; again let us assume that it is

dependent and not included. Proceeding in this way we arrive at the "worst possible case" where we have found $(w_1,w_2,\ldots,w_s)$ to be linearly dependent upon $(u_1,u_2,\ldots,u_s)$. It now remains to show that $(w_{s+1},\ldots,w_n)$ cannot be linearly dependent upon $(u_1,\ldots,u_s)$. By hypothesis $(w_1,\ldots,w_n)$ are linearly independent and hence $(w_{s+1},\ldots,w_n)$ are also. Suppose w_{s+1} is linearly dependent upon $(u_1,\ldots,u_s)$. This means that corresponding to the s-dimensional subspace spanned by $(u_1,\ldots,u_s)$ we are able to choose from $(w_1,\ldots,w_n)$ s+1 independent vectors that span this space. This is a contradiction and hence w_{s+1} cannot be linearly dependent upon $(u_1,\ldots,u_s)$. Continuing in this manner we find that $(w_{s+1},\ldots,w_n)$ are all linearly independent of $(u_1,\ldots,u_s)$. Hence we have obtained a set of n vectors, say

$$(u_1,\ldots,u_s,z_1,\ldots,z_{n-s})$$

where

$$(z_1,\ldots,z_{n-s}) \subseteq (w_1,\ldots,w_n) \qquad (13.2)$$

such that any vector is linearly independent of the preceding vectors in the set. By means of the Gram-Schmidt orthogonalisation procedure we may ortho-normalise this set of vectors.

Lemma 13.2 The product of two unitary matrices is itself unitary.

Proof: Let U and V be two unitary matrices such that

$$U^H U = V^H V = I. \tag{13.3}$$

Then

$$(VU)^H(VU) = U^H V^H VU = I. \tag{13.4}$$

We are now in a position to prove the following theorem.

Theorem 13.3 Any square matrix A can be reduced by a unitary transformation to an upper triangular matrix with the eigenvalues of A on the diagonal.

Proof: Let A have an eigenvalue λ_1 with a corresponding eigenvector x_1 which is normalised such that $\|x_1\|_2=1$. We have shown that vectors $w_2,\ldots,w_n$ exist such that

$$Q = (x_1, w_2, \ldots, w_n) = [x_1, W] \tag{13.5}$$

is a unitary matrix. Hence

$$\begin{aligned} Q^H Q &= \begin{bmatrix} x_1^H \\ W^H \end{bmatrix} \begin{bmatrix} x_1, & W \end{bmatrix} \\ &= \begin{bmatrix} x_1^H x_1 & x_1^H W \\ W^H x_1 & W^H W \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & I_{n-1} \end{bmatrix} \end{aligned} \tag{13.6}$$

so that $W^H x_1 = 0$. Thus, since

$$Ax_1 = \lambda_1 x_1, \tag{13.7}$$

$$Q^H A Q = \begin{bmatrix} x_1^H \\ W^H \end{bmatrix} A \, (x_1, \; W)$$

$$= \begin{bmatrix} x_1^H \\ W^H \end{bmatrix} (\lambda_1 x_1, \; AW)$$

$$= \begin{pmatrix} \lambda_1 & x_1^H AW \\ 0 & W^H AW \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_1 & B \\ 0 & C \end{pmatrix}, \text{ say.} \qquad (13.8)$$

We now proceed by induction. If n=2 the theorem is true as (13.8) is already in upper triangular form. Now assume that A is n*n and the theorem is true for n-1. Then $C=W^H AW$ of (13.8) is of order n-1, and a unitary matrix V exists such that $V^H CV$ is upper triangular. The matrix

$$U = \begin{pmatrix} 1 & 0 \\ 0 & V \end{pmatrix} \qquad (13.9)$$

is unitary and

$$U^H Q^H A Q U = \begin{bmatrix} 1 & 0 \\ 0 & V^H \end{bmatrix} \begin{bmatrix} \lambda_1 & B \\ 0 & C \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 & BV \\ 0 & V^H CV \end{bmatrix} . \qquad (13.10)$$

Hence $(QU)^H A (QU)$ is upper triangular. Since, from lemma 13.2, QU is unitary, A has been reduced to

upper triangular form by a unitary transformation. We have shown that if the result is true for a matrix of order (n-1) it is true for one of order n. However, it is true for 2*2 matrices and the result is proved by induction.

14. SOME SPECIAL CASES

We have just proved that for any matrix A there exists a unitary matrix R such that

$$R^H AR = T \qquad (14.1)$$

where T is upper triangular.

Suppose A is Hermitian. Then, as $R^H AR$ is Hermitian, T must be Hermitian and hence T must be diagonal. This proves the result that we stated at the end of section 12, that whatever the multiplicity of the eigenvalues a Hermitian matrix can always be reduced to diagonal form by a unitary transformation. Note also that if A is real symmetric its eigenvalues and eigenvectors are real and hence it may be reduced to diagonal form by an orthogonal transformation.

It also follows immediately that the elementary divisors of a Hermitian matrix are all linear and hence it cannot be defective. If a Hermitian matrix has any multiple eigenvalues then it is derogatory.

15. NORMAL MATRICES

We now ask if there is a more general class

of matrix, other than a Hermitian, which can be reduced to diagonal form by a unitary transformation. Thus far we have considered

$$R^H A R = D \tag{15.1}$$

where D has always been real. We now consider (15.1) in the case of D complex. From equation (15.1) we obtain

$$A = RDR^H \tag{15.2}$$

and hence

$$A^H = RD^H R^H. \tag{15.3}$$

Consider

$$\begin{aligned} AA^H &= RDR^H RD^H R^H \\ &= RDD^H R^H \\ &= RD^H DR^H \quad \text{(as diagonal matrices commute)} \\ &= RD^H R^H RDR^H \\ &= A^H A. \end{aligned} \tag{15.4}$$

We show conversely that if $AA^H = A^H A$, then A may be factorised as in (15.2). From (14.1) any matrix A may be expressed in the form

$$A = RTR^H \tag{15.5}$$

where R is unitary and T upper triangular. Hence we have

$$AA^H = RTR^H RT^H R^H = RT^H R^H RTR^H = A^H A \tag{15.6}$$

giving

$$RTT^H R^H = RT^H TR^H \tag{15.7}$$

hence

$$TT^H = T^HT. \tag{15.8}$$

Equating the elements in equation (15.8) we find that all the off-diagonal elements of T are zero, so that T is diagonal. Hence the most general class of matrices which can be factorised as in (15.2) is the same as that class of matrices for which

$$AA^H = A^HA. \tag{15.9}$$

Such matrices are said to be normal. Obvious examples of normal matrices are Hermitian, skew-Hermitian and unitary matrices; also all diagonal matrices. We now prove the following theorem which gives an alternative definition of a normal matrix.

Theorem 15.1 A matrix A is normal if and only if

$$A = B + C \tag{15.10}$$

where B is Hermitian, C is skew-Hermitian and

$$BC = CB. \tag{15.11}$$

Proof: (i) If equations (15.10) and (15.11) hold then

$$\begin{aligned} AA^H &= (B+C)(B^H+C^H) \\ &= BB^H + BC^H + CB^H + CC^H \\ &= B^HB - BC + CB - C.C \\ &= B^HB - CB + BC + C^HC \\ &= B^HB + C^HB + B^HC + C^HC \\ &= (B^H+C^H)(B+C) \\ &= A^HA. \end{aligned} \tag{15.12}$$

(ii) If $AA^H=A^HA$ then, for any matrix A, we may write

$$A = B + C \tag{15.13}$$

where B is Hermitian and C is a skew-Hermitian matrix. This is seen by considering the (i,j) and (j,i) elements of A.

$$a_{ij} = b_{ij} + c_{ij} \tag{15.14}$$

and

$$a_{ji} = \overline{b}_{ij} - \overline{c}_{ij}. \tag{15.15}$$

These equations always have a solution, namely

$$b_{ij} = \tfrac{1}{2}(a_{ij} + \overline{a}_{ji}) \tag{15.16}$$

and

$$c_{ij} = \tfrac{1}{2}(a_{ij} - \overline{a}_{ji}). \tag{15.17}$$

Obviously we take $b_{ii}=\mathrm{Re}(a_{ii})$ and $c_{ii}=i*\mathrm{Im}(a_{ii})$.

From (15.13) we obtain

$$\begin{aligned} AA^H &= (B+C)(B^H+C^H) \\ &= B^HB - BC + CB + C^HC \end{aligned} \tag{15.18}$$

and

$$\begin{aligned} A^HA &= (B^H+C^H)(B+C) \\ &= B^HB - CB + BC + C^HC. \end{aligned} \tag{15.19}$$

Hence we must have

$$-BC + CB = -CB + BC \tag{15.20}$$

or

$$BC = CB \tag{15.21}$$

which proves sufficiency.

16. PRINCIPAL VECTORS

Finally in this introductory section we mention, mainly for the sake of completeness, the idea of principal vectors. We saw that a matrix A with linear elementary divisors has n eigenvectors spanning the whole n-space. If A has non-linear divisors this is not true as there are fewer than n independent eigenvectors. It is often convenient however to have a set of vectors which spans the whole n-space and which reduces to the eigenvectors of A when A has linear elementary divisors. We saw that if a matrix can be diagonalised then

$$AX = X\mathrm{diag}(\lambda_i). \tag{16.1}$$

If it cannot be we take as a basis the n columns of a matrix X which is such that

$$AX = XC \tag{16.2}$$

where C is the Jordan canonical form of A. To illustrate the importance of these vectors we give an example used by Wilkinson (1965), where a fuller treatment of the subject is to be found.

Suppose A is such that

$$AX = X\begin{bmatrix} C_3(\lambda_1) & & & \\ & C_2(\lambda_1) & & \\ & & C_2(\lambda_2) & \\ & & & C_1(\lambda_3) \end{bmatrix}, \tag{16.3}$$

then in the usual notation for the columns of X, and letting $\varepsilon=1$ for simplicity in equation (7.2),

we obtain

$$Ax_1=\lambda_1 x_1 \qquad Ax_4=\lambda_1 x_4 \qquad Ax_6=\lambda_2 x_6 \qquad Ax_8=\lambda_3 x_8$$

$$Ax_2=\lambda_1 x_2+x_1 \qquad Ax_5=\lambda_1 x_5+x_4 \qquad Ax_7=\lambda_2 x_7+x_6$$

$$Ax_3=\lambda_1 x_3+x_2, \tag{16.4}$$

from which we see that

$$(A-\lambda_1 I)x_1=0 \quad (A-\lambda_1 I)x_4=0 \quad (A-\lambda_2 I)x_6=0 \quad (A-\lambda_3 I)x_8=0$$

$$(A-\lambda_1 I)^2 x_2=0 \quad (A-\lambda_1 I)^2 x_5=0 \quad (A-\lambda_2 I)^2 x_7=0$$

$$(A-\lambda_1 I)^3 x_3=0. \tag{16.5}$$

Each of the vectors therefore satisfies a relation of the form

$$(A - \lambda_i I)^j x_k = 0. \tag{16.6}$$

A vector which satisfies equation (16.6) for a given value of j, but does not satisfy it for any lower value of j, is called a principal vector of grade j corresponding to λ_i. Eigenvectors are principal vectors of grade one. Note that, although there exists a set of principal vectors which spans the whole n-space, in general principal vectors are not unique. If x is a principal vector of grade j corresponding to λ_i then the same is true of any other vector obtained by adding to x multiples of any principal vectors of grade j or less corresponding to λ_i.

CHAPTER 2

JACOBI-LIKE METHODS FOR REDUCING MATRICES TO DIAGONAL FORM

1. INTRODUCTION

In this chapter we describe techniques for solving the eigenproblem for general complex matrices by methods related to the Jacobi method for real symmetric matrices. These techniques are applicable to any form of matrix but are of particular relevance in the case of small dense diagonally dominant matrices. Determining the eigensolutions of such matrices forms a vital part in the acceleration of simultaneous iteration and hence their relevance to our work.

Historically, the method for real symmetric matrices discovered by Jacobi (1846) is the oldest process for determining eigensolutions which is applicable for use on an electronic computer. The extension of this method to deal with Hermitian matrices is simple and it is then a relatively easy step to consider normal matrices.

We saw earlier that for a normal matrix A there is always a unitary matrix V such that

$$V^H A V = \text{diag}\ (\lambda_i). \qquad (1.1)$$

However for an arbitrary matrix A this is not in general true but having considered normal matrices we are well placed to tackle the problem of general matrices.

All of the methods we consider depend upon the application of a series of similarity transformations to convert the matrix A into a

special form. In the simplest cases the similarity transformations are unitary matrices which reduce A directly to diagonal or, in practice, to nearly diagonal form. In the general case we shall use similarity transformations to reduce the matrix to normal form before applying unitary transformations to further reduce A to diagonal form.

Finally in this chapter we give brief details of an abortive attempt to use an algorithm proposed by Rutishauser.

2. THE CLASSICAL JACOBI METHOD FOR REAL SYMMETRIC MATRICES

In the method of Jacobi the original matrix is transformed to diagonal form by a sequence of plane rotations. In fact, to complete the diagonalisation would require an infinite number of such rotations but in practice the process is terminated when the off-diagonal elements are negligible to working accuracy. As we are considering real symmetric matrices, which we know have real eigenvalues and eigenvectors, we use real plane rotations.

Let the matrix V of order n be such that

$$\begin{aligned} v_{pp} &= v_{qq} = \cos\phi \\ v_{pq} &= -v_{qp} = \sin\phi \\ v_{ii} &= 1, \quad i \neq p,q \\ v_{ij} &= 0 \text{ otherwise.} \end{aligned} \tag{2.1}$$

We note that

$$VV^T = V^TV = I. \tag{2.2}$$

If we denote the original matrix by A_0 then we may describe the Jacobi process as follows. A sequence of matrices A_k is produced satisfying the relations

$$A_k = V_k^T A_{k-1} V_k , \qquad k = 1,2,\ldots . \tag{2.3}$$

We note that as A_{k-1} is symmetric, so too is A_k. The matrix V_k is determined by the rules which follow. If no confusion can arise we shall denote the elements of A_{k-1} by a_{ij} and those of A_k by a'_{ij}. Using this notation we consider the details of equation (2.3). The only elements of A_{k-1} that are altered are those in rows p and q and in columns p and q. Hence the modified elements are given by

$$a'_{ij} = a_{ij} , \qquad i,j \neq p,q \tag{2.4}$$

$$\left.\begin{aligned} a'_{pj} &= a_{pj}\cos\phi - a_{qj}\sin\phi \\ a'_{qj} &= a_{qj}\cos\phi + a_{pj}\sin\phi \end{aligned}\right\} \quad j \neq p,q \tag{2.5}$$

$$\left.\begin{aligned} a'_{pp} &= a_{pp}\cos^2\phi + a_{qq}\sin^2\phi - 2a_{pq}\sin\phi\cos\phi \\ a'_{qq} &= a_{qq}\cos^2\phi + a_{pp}\sin^2\phi + 2a_{pq}\sin\phi\cos\phi \end{aligned}\right\} \tag{2.6}$$

$$\begin{aligned} a'_{pq} &= a_{pq}(\cos^2\phi - \sin^2\phi) + (a_{pp} - a_{qq})\sin\phi\cos\phi \\ &= a'_{qp} . \end{aligned} \tag{2.7}$$

We do not explicitly state the form of a'_{jp}, a'_{jq} or

a'_{qp} since these follow by symmetry.

We now introduce two functions which we shall use extensively in what follows. Firstly the function defined by

$$N^2(A_k) = \sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}^{(k)}|^2 \qquad (2.8)$$

and secondly

$$t^2(A_k) = \sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}^{(k)}|^2. \qquad (2.9)$$

We note that $t^2(A_k) = N^2(A_k) - \sum_i |a_{ii}^{(k)}|^2$.

We notice that the transformations given by (2.3) leave $N^2(A_{k-1})$ unaltered and that A_k will tend to diagonal form if and only if $t^2(A_k) \to 0$. We therefore need only consider the effect of the off-diagonal elements and noting from equation (2.5) that

$$a'^2_{pj} + a'^2_{qj} = a^2_{pj} + a^2_{qj}, \qquad j \neq p,q \qquad (2.10)$$

we see that the only reduction in $t^2(A_k)$ that can take place is through the reduction of $|a'_{pq}|$ (and $|a'_{qp}|$).

Hence it is clear that the maximum reduction in $t^2(A_k)$ will occur if we use (2.3) to reduce $|a_{pq}|$ as much as possible. In fact it is possible to make a_{pq} zero at any stage. For if

$$a'_{pq} = a_{pq}(\cos^2\phi - \sin^2\phi) + (a_{pp} - a_{qq})\sin\phi\cos\phi$$

$$= a_{pq}\cos 2\phi + (a_{pp} - a_{qq})\tfrac{1}{2}\sin 2\phi$$

$$= 0 \qquad (2.11)$$

then

$$\tan 2\phi = \frac{2a_{pq}}{(a_{qq} - a_{pp})} . \qquad (2.12)$$

We shall always restrict ϕ such that

$$-\pi/4 \leq \phi \leq \pi/4 \qquad (2.13)$$

and if $a_{pp}=a_{qq}$ we take ϕ to be $\pm\pi/4$ according to the sign of a_{pq}.

It is clear that in order to obtain the maximum reduction of $t^2(A)$ at every stage we should choose p and q such that

$$a_{pq} = \max(a_{ij}) . \qquad (2.14)$$

With the choice of ϕ given by (2.12) we reduce a'_{pq} to zero and hence at the k-th stage

$$t^2(A_k) = t^2(A_{k-1}) - 2(a_{pq}^{(k-1)})^2. \qquad (2.15)$$

Since a_{pq} was the absolutely largest of all the n(n-1) off-diagonal elements of A we have

$$t^2(A_k) \leq (1-2/(n^2 - n))t^2(A_{k-1})). \qquad (2.16)$$

Hence, at the k-th stage

$$t^2(A_k) \leq (1-2/(n^2-n))^k t^2(A_0). \qquad (2.17)$$

If we let $N=\frac{1}{2}(n^2-n)$ we obtain

$$t^2(A_k) < e^{-k/N}t^2(A_0). \qquad (2.18)$$

This shows that the process is convergent although

in practice (2.18) is a very crude bound as we shall see later. Suppose we define r by

$$k = rN \tag{2.19}$$

then

$$t^2(A_{rN}) < e^{-r}t^2(A_0) \tag{2.20}$$

$$< \varepsilon^2 t^2(A_0) \tag{2.21}$$

if $$r > 2\ln(1/\varepsilon). \tag{2.22}$$

If, for example, $\varepsilon=2^{-t}$, then

$$r > 2\ln 2^t \simeq 1.39t. \tag{2.23}$$

3. CONVERGENCE TO A FIXED DIAGONAL MATRIX

We still have to show that A_k tends to a fixed diagonal matrix with the eigenvalues of A on the diagonal. Suppose the iteration is at a stage such that

$$t(A_k) < \varepsilon \tag{3.1}$$

and suppose the eigenvalues λ_i of A_k (and hence of A) are arranged in non-increasing order. If we similarly arrange the diagonal elements of A_k to be in non-increasing order it may be seen that the elements in the two sequences differ by less than ε. This follows from the minimax characterisation of the eigenvalues of the sum of two symmetric matrices as given by, for example, Wilkinson (1965). Thus the $a_{ii}^{(k)}$, arranged in some order, lie in intervals of width 2ε centred on the λ_i. Since we may make

ε arbitrarily small, the eigenvalues can be located to any desired accuracy.

It remains to show that each $a_{ii}^{(k)}$ does indeed converge to a specified λ_i. We firstly consider the case of distinct eigenvalues and define ε by the relation

$$0 < 4\varepsilon = \min_{i \neq j} |\lambda_i - \lambda_j|. \tag{3.2}$$

Let k be chosen so that (3.1) is true, then from (2.16) it is satisfied for all subsequent values of k. With this choice of ε the intervals centred on the λ_i are clearly disjoint and hence there is just one of the $a_{ii}^{(k)}$ in each interval. We assume the λ_i and the $a_{ii}^{(k)}$ have been ordered as before and now show that each $a_{ii}^{(k)}$ lies in the same interval at all later stages of the iteration.

Suppose the next iteration is in the (p,q) plane. The only diagonal elements to be altered are $a_{pp}^{(k-1)}$ and $a_{qq}^{(k-1)}$; hence $a_{pp}^{(k)}$ and $a_{qq}^{(k)}$ must still be the two diagonal elements in the intervals centred on λ_p and λ_q. We show that it must be $a_{pp}^{(k)}$ which lies in the interval containing λ_p. For, reverting to our earlier prime notation

$$a'_{qq} - \lambda_p = a_{qq}\cos^2\phi + a_{pp}\sin^2\phi + 2a_{pq}\sin\phi\cos\phi - \lambda_p$$

$$= (a_{qq} - \lambda_q + \lambda_q - \lambda_p)\cos^2\phi$$

$$+ (a_{pp} - \lambda_p)\sin^2\phi + 2a_{pq}\sin\phi\cos\phi. \qquad (3.3)$$

Hence

$$|a'_{qq} - \lambda_p| \geq |\lambda_q - \lambda_p|\cos^2\phi - |a_{pp} - \lambda_p|\sin^2\phi$$

$$-|a_{qq} - \lambda_q|\cos^2\phi - |a_{pq}|$$

$$\geq 4\varepsilon\cos^2\phi - \varepsilon\sin^2\phi - \varepsilon\cos^2\phi - \varepsilon$$

$$= 2\varepsilon\cos 2\phi. \qquad (3.4)$$

However,

$$|\tan 2\phi| = |2a_{pq}/(a_{pp} - a_{qq})|$$

$$\leq 2\varepsilon/2\varepsilon$$

$$= 1. \qquad (3.5)$$

Hence for ϕ in the range given by (2.13) we have $|2\phi| \leq \pi/4$ and hence

$$|a'_{qq} - \lambda_p| \geq 2^{\frac{1}{2}}\varepsilon \qquad (3.6)$$

showing that a'_{qq} is not in the interval centred on λ_p.

The result is not substantially altered if there are multiple eigenvalues present. We define ε such that (3.2) is true for all distinct λ_i and λ_j. If λ_i is an eigenvalue of multiplicity m, then precisely m of the $a_{ii}^{(k)}$ lie in the interval containing λ_i and the proof that none of the a_{pp} and a_{qq} lying in different intervals centred on distinct values of λ can move out of those intervals remains valid.

Hence, whatever the multiplicity of the eigenvalues, after a certain stage of the iteration each diagonal element of A_k remains in a fixed interval centred on λ_i as k is increased and the width of that interval tends to zero as k tends to infinity.

4. THE GERSCHGORIN DISCS

Suppose the eigenvalues λ_i of A are distinct and that

$$\min_{i \neq j} |\lambda_i - \lambda_j| = 2\delta \qquad (4.1)$$

and we have reached, at the k-th stage, a matrix A_k for which

$$|a_{ii}^{(k)} - \lambda_i| < \delta/4 , \qquad \max_{i \neq j} |a_{ij}| = \varepsilon \qquad (4.2)$$

together with

$$(n - 1)\varepsilon < \delta/4. \qquad (4.3)$$

The Gerschgorin discs are definitely disjoint, since

$$\begin{aligned} |a_{ii}^{(k)} - a_{jj}^{(k)}| &\geq |\lambda_i - \lambda_j| - |a_{ii}^{(k)} - \lambda_i| \\ &\quad - |a_{jj}^{(k)} - \lambda_j| \\ &> 3\delta/2 . \end{aligned} \qquad (4.4)$$

Hence we have

$$|a_{ii}^{(k)} - \lambda_i| < (n - 1)\varepsilon . \qquad (4.5)$$

If we perform upon A the similarity transformation corresponding to multiplying the i-th row by ε/δ and the i-th column by δ/ε, then the i-th Gerschgorin

disc is certainly contained in the circle

$$|a_{ii}^{(k)} - \lambda| < (n-1)\varepsilon^2/\delta \qquad (4.6)$$

while the remainder are contained in the discs

$$|a_{jj}^{(k)} - \lambda| < (n-2)\varepsilon + \delta , \quad j \neq i. \qquad (4.7)$$

The i-th disc is certainly isolated from the others, so from (4.6) we see that when the absolute values of the off-diagonal elements have become less than ε then, for sufficiently small ε, the diagonal elements differ from the eigenvalues by quantities of order ε^2. This result holds for the simple eigenvalues even if A has some multiple eigenvalues.

We see that we can obtain very good estimates of the eigenvalues when the off-diagonal elements have become small. It is therefore important that the convergence of the Jacobi method should become more rapid in the later stages of the iteration. Some results on quadratic convergence are known but before considering these we consider different forms of the Jacobi method.

5. VARIANTS OF THE JACOBI METHOD

The process we have outlined so far depends upon determining at each step the absolutely largest off-diagonal element. Searching for this element is fairly time consuming on an automatic computer and so it is usual to select the elements to be eliminated in some simpler manner.

Any method in which each of the off-diagonal elements is eliminated just once until all $n(n-1)/2$ elements have been chosen as pivot is called a generalised serial Jacobi method. Each sequence of $n(n-1)/2$ rotations is called a sweep. The special serial Jacobi method is the particular case of a generalised serial method in which the order of elimination of elements in each sweep is given by $(1,2), (1,3), \ldots, (1,n); (2,3), (2,4), \ldots, (2,n); \ldots ; (n-1,n)$. The special serial method is particularly suited to implementation on an electronic computer. Forsythe and Henrici (1960) have shown that the special serial Jacobi method does in fact converge but the proof is quite involved. We do not consider it here as the algorithm we have actually used is slightly different and is referred to as the threshold Jacobi method.

The maximum reduction that can be achieved in $t^2(A_k)$ at any stage by a rotation in the (p,q) plane is given by $2(a_{pq}^{(k-1)})^2$. If $a_{pq}^{(k-1)}$ is much smaller than the average value of the other off-diagonal elements then there is little point in performing the (p,q) rotation. This idea leads to the threshold Jacobi method. With each sweep there is an associated threshold value and any rotation based on an off-diagonal element which is below the threshold value is omitted. We may

assume without loss of generality that the off-diagonal elements are of order unity and typically we might then choose the set of threshold values 2^{-2}, 2^{-4}, 2^{-8}, 2^{-12}. Assuming that 2^{-12} is the smallest number m on the machine for which

$$1.0 + m \neq 1.0 \qquad (5.1)$$

we select all subsequent threshold values to be 2^{-12}. This means that only zeros are skipped in the fourth and later sweeps. A suitable criterion for terminating the process is to stop when n(n-1)/2 successive elements have been skipped. We give later the details of the implementation of the algorithm that we have used. It is important to notice that, with the threshold Jacobi method, termination is guaranteed as there are only a finite number of iterations corresponding to any given threshold value. From the analysis of section 4 we see that the diagonal elements of A then differ from the eigenvalues by less than $(n-1)\varepsilon^2/\delta$.

6. ULTIMATE QUADRATIC CONVERGENCE OF JACOBI METHODS

We showed in section 4 that we would expect a Jacobi method to be ultimately quadratically convergent. It has been shown that several variants of the Jacobi method do indeed have ultimate quadratic convergence when the eigenvalues are distinct. The first result in this field was obtained by Henrici (1958). For completeness we

summarise, using the notation of sections 2 and 4, the best results obtained so far.

For the classical Jacobi method, Schönage (1961) has shown that if

$$|\lambda_i - \lambda_j| \geq 2\delta , \quad i \neq j \tag{6.1}$$

and if we have reached a stage at which

$$t(A_r) < \tfrac{1}{2}\delta \tag{6.2}$$

then

$$t(A_{r+N}) \leq (\tfrac{1}{2}n - 1)^{\frac{1}{2}} t^2(A_r)/\delta , \tag{6.3}$$

where $N=\frac{1}{2}n(n-1)$.

For the general serial Jacobi method Wilkinson (1962) has shown that, subject to (6.1) and (6.2)

$$t(A_{r+N}) \leq \tfrac{1}{2}(n^2 - n)^{\frac{1}{2}} t^2(A_r)/\delta . \tag{6.4}$$

For the special serial Jacobi method Wilkinson (1962) has shown that

$$t(A_{r+N}) \leq t^2(A_r)/(2^{\frac{1}{2}}\delta). \tag{6.5}$$

We discuss only (6.5) in detail.

7. BOUND FOR THE SPECIAL SERIAL JACOBI METHOD

As before we assume that the eigenvalues λ_i of A_0 satisfy

$$|\lambda_i - \lambda_j| \geq 2\delta , \quad i \neq j. \tag{7.1}$$

Following Wilkinson (1962) we let

$$S_k = 2^{-\frac{1}{2}} t(A_k). \tag{7.2}$$

Suppose we have reached the stage when

$$S_k < \delta/4. \tag{7.3}$$

Then from the preceding sections we have that

$$\|A_k - \mathrm{diag}(a_{ii}^{(k)})\|_E < \delta/2 \tag{7.4}$$

and hence for some ordering of the λ_i

$$\begin{aligned} |a_{ii}^{(k)} - a_{jj}^{(k)}| &= |(a_{ii}^{(k)} - \lambda_i) - (a_{jj}^{(k)} - \lambda_j) \\ &\qquad + (\lambda_i - \lambda_j)| \\ &\geq |\lambda_i - \lambda_j| - |a_{ii}^{(k)} - \lambda_i| \\ &\qquad - |a_{jj}^{(k)} - \lambda_j| \\ &> 2\delta - \tfrac{1}{2}\delta - \tfrac{1}{2}\delta \\ &= \delta . \end{aligned} \tag{7.5}$$

We recall that the effect of a plane rotation is to reduce $(S_k)^2$ by the square of the element which is reduced to zero and hence, if (7.3) holds for any k, it holds for all subsequent k. For convenience we assume that equation (7.3) holds when k=0.

In the special serial Jacobi method the elements are annihilated in the row order given in section 5. We denote the N off-diagonal elements in their correct order by $\alpha_1, \alpha_2, \ldots, \alpha_N$ and the angles of rotation corresponding to them by $\phi_1, \phi_2, \ldots, \phi_N$. If α_i is annihilated by a rotation in the (p,q) plane we have

$$|\tan 2\phi_i| = \left| \frac{2\alpha_i}{a_{pp}^{(i)} - a_{qq}^{(i)}} \right|$$

$$\leq \frac{2|\alpha_i|}{\delta} \tag{7.6}$$

and hence

$$\begin{aligned} |\sin\phi_i| &\leq |\phi_i| \\ &= \tfrac{1}{2}|2\phi_i| \\ &\leq \tfrac{1}{2}|\tan 2\phi_i| \\ &\leq |\alpha_i|/\delta \ . \end{aligned} \tag{7.7}$$

Since

$$S_{i+1}^2 - S_i^2 = -\alpha_i^2 \tag{7.8}$$

we have

$$0 \leq S_N^2 = S_0^2 - \sum_{i=1}^{N} \alpha_i^2 \ . \tag{7.9}$$

Hence, from (7.7)

$$\sum_{i=1}^{N} \sin^2\phi_i \leq \frac{1}{\delta^2}\sum_{i=1}^{N}\alpha_i^2 \leq S_0^2/\delta^2. \tag{7.10}$$

We now consider the effect of the rotations on the element α_i after it has been annihilated. It is only affected by a subset of the later rotations - namely those in the (p,q) plane where α_i lies in the p or q-th row or column. In order to simplify the resulting analysis we consider firstly the following particular case. We let p=1 and consider what happens to a_{pq} as q=2,3,....,n. From equations (2.5) and (2.7) we see that

$$a_{12}^{(1)} = 0 \ , \qquad q = 2$$

$$a_{12}^{(2)} = 0.\cos\phi_2 - a_{32}^{(1)}\sin\phi_2 \ , \quad q = 3$$

$$a_{12}^{(3)} = a_{12}^{(2)}\cos\phi_3 - a_{42}^{(1)}\sin\phi_3 , \quad q = 4$$

. . . . . .

$$a_{12}^{(n-1)} = a_{12}^{(n-2)}\cos\phi_{n-1} - a_{n2}^{(1)}\sin\phi_{n-1} ,$$

$$q = n, \qquad (7.11)$$

where we use ϕ_i to denote in the correct order each of the N angles needed in one complete sweep. Since $|\cos\phi_i| \leq 1$ we have

$$|a_{12}^{(n-1)}| \leq |a_{32}^{(1)}||\sin\phi_2| + |a_{42}^{(1)}||\sin\phi_3| + \ldots$$

$$\ldots + |a_{n2}^{(1)}||\sin\phi_{n-1}| . \qquad (7.12)$$

Similarly we may consider the history of the other elements in the first row. In particular

$$a_{13}^{(1)} = a_{13}^{(0)}\cos\phi_1 - a_{23}^{(0)}\sin\phi_1 , \qquad q = 2$$

$$a_{13}^{(2)} = 0 , \qquad q = 3$$

$$a_{13}^{(3)} = 0.\cos\phi_3 - a_{43}^{(1)}\sin\phi_3 , \qquad q = 4$$

$$a_{13}^{(4)} = a_{13}^{(3)}\cos\phi_4 - a_{53}^{(1)}\sin\phi_4 , \qquad q = 5$$

. . . . . .

$$a_{13}^{(n-1)} = a_{13}^{(n-2)}\cos\phi_{n-1} - a_{n3}^{(1)}\sin\phi_{n-1} , \quad q = n. \qquad (7.13)$$

Hence we have

$$|a_{13}^{(n-1)}| \leq |a_{43}^{(1)}||\sin\phi_3| + |a_{53}^{(1)}||\sin\phi_4| + \ldots$$

$$\ldots + |a_{n3}^{(1)}||\sin\phi_{n-1}| . \qquad (7.14)$$

In general

$$|a_{1j}^{(n-1)}| \leq \sum_{k=j+1}^{n} |a_{kj}^{(1)}||\sin\phi_{k-1}| \ ,$$

$$j = 2,3,\ldots,n-1 \qquad (7.15)$$

$$a_{1n}^{(n-1)} = 0. \qquad (7.16)$$

Hence

$$\sum_{j=2}^{n-1}(a_{1j}^{(n-1)})^2 \leq \left(\sum_{j=2}^{n-1}\sum_{k=j+1}^{n}(a_{kj}^{(1)})^2\right)\left(\sum_{j=2}^{n-1}\sin^2\phi_j\right) . \qquad (7.17)$$

Now, as each rotation alters only two elements in each of the relevant rows and leaves the sum of their squares unaltered we have

$$(a_{k1}^{(k-2)})^2 + \sum_{j=2}^{k-1}(a_{kj}^{(1)})^2 = \sum_{j=1}^{k-1}(a_{kj}^{(0)})^2 \ ,$$

$$k = 3,4,\ldots,n \qquad (7.18)$$

Hence, certainly

$$\sum_{j=2}^{n-1}(a_{1j}^{(n-1)})^2 \leq \left(\sum_{k=3}^{n}\sum_{j=1}^{k-1}(a_{kj}^{(0)})^2\right)\left(\sum_{j=2}^{n-1}\sin^2\phi_j\right)$$

$$\leq S_0^2 \cdot \sum_{j=2}^{n-1}\sin^2\phi_j \ . \qquad (7.19)$$

We note that this sum of squares of the first row is unaltered by subsequent transformations in this sweep.

Secondly we consider the sum of the squares of the super-diagonal elements of the second row on completion of their succesive annihilation. From a discussion similar to that for the first row we obtain

$$\sum_{j=3}^{n-1}(a_{2j}^{(n)})^2 \leq S_{n-1}^2 \sum_{j=n+1}^{2n-3}\sin^2\phi_j \ .$$

$$\leq S_0^2 \sum_{j=n+1}^{2n-3} \sin^2\phi_j \ , \qquad (7.20)$$

and this sum then remains constant.

In general, for the k-th row we have

$$\sum_{j=k+1}^{n-1} (a_{kj}^{(n+k-2)})^2 \leq S_{f(n)}^2 \sum_{j=0}^{n-k-2} \sin^2\phi_{g(j)}$$

$$\leq S_0^2 \sum_{j=0}^{n-k-2} \sin^2\phi_{g(j)} \ ,$$

$$k = 1,2,\ldots,n-2 \qquad (7.21)$$

where the exact form of f(n) and g(j) are defined by

$$f(n) = \sum_{i=1}^{k-1} (n - i) \qquad (7.22)$$

and

$$g(j) = 2 + \sum_{i=1}^{k-1} (n - i) + j \ . \qquad (7.23)$$

Adding equations (7.21) for k = 1,2,....,n-2 we obtain

$$S_N^2 \leq S_0^2 \cdot \sum_{j=1}^{N} \sin^2\phi_j$$

$$\leq S_0^2 \cdot \frac{S_0^2}{\delta^2}$$

$$= \frac{S_0^4}{\delta^2} \ . \qquad (7.24)$$

Clearly a similar result must hold for each of the rows in every sweep and hence in general we have

$$S_N \leq S_0^2/\delta \ . \qquad (7.25)$$

Thus, from (7.2)

$$\frac{t(A_N)}{2^{\frac{1}{2}}} \leq \frac{t^2(A_0)}{2\delta} \qquad (7.26)$$

or

$$t(A_N) \leq \frac{t^2(A_0)}{2^{\frac{1}{2}}\delta} \qquad (7.27)$$

as stated in (6.5).

In fact (7.27) is often a bad estimate in reality, for we have replaced

$$|a_{ii}^{(k)} - a_{jj}^{(k)}| , \quad i \neq j$$

by δ whereas some may be much larger. We also replaced all S_k by S_0 and finally it is usually the case that some of the contributions made to an element after annihilation are positive and some negative.

8. MULTIPLE EIGENVALUES

In all the foregoing we have assumed that

$$|\lambda_i - \lambda_j| \neq 0 , \quad i \neq j \qquad (8.1)$$

and all the convergence proofs have contained a factor $1/\delta$. Obviously all the proofs break down if δ is zero and even if δ is small we would expect the rate of convergence to be slow. In fact it has been shown, see for example Schönage (1961), that if none of the eigenvalues is of multiplicity greater than two then quadratic convergence can be guaranteed for the classical Jacobi method. Little theoretical progress seems to have been made in the case of eigenvalues of multiplicity greater than two. However, our experience suggests that convergence is usually just as fast if there are

repeated eigenvalues. Wilkinson (1965) reports similar findings and justifies them on the following grounds.

In the first place if all the eigenvalues are equal to λ_1 then the matrix is diagonal and no iterations are required. Now suppose the eigenvalues are all very close so that

$$\lambda_i = a + \delta_i \tag{8.2}$$

where the δ_i are small compared with a. Then we have

$$A = V\text{diag}(a + \delta_i)V^T \tag{8.3}$$

for some orthogonal V, and hence

$$A = aI + V\text{diag}(\delta_i)V^T. \tag{8.4}$$

Now

$$t(V\text{diag}(\delta_i)V^T) \leq (\sum \delta_i^2)^{\frac{1}{2}} \tag{8.5}$$

which shows that the off-diagonal elements must be small initially. Computation with A is identical to computation with (A-aI), and this matrix is no longer special in the original sense. However, as all the elements of (A-aI) are small to start with we may expect that the reduction of the off-diagonal elements below a prescribed level will require less operations for this matrix than for a general symmetric matrix. In fact if $\max|\delta_i| = O(10^{-r})$, then we would expect to reduce the off-diagonal elements of A below 10^{-t-r} in the time it usually takes to

reduce them to 10^{-t}.

9. CALCULATION OF THE EIGENVECTORS

If the last rotation to be performed is V_s then we have

$$(V_s^T \ldots V_2^T V_1^T) A_0 (V_1 V_2 \ldots V_s) = \mathrm{diag}(\lambda_i) \qquad (9.1)$$

to working accuracy. The eigenvectors of A_0 are therefore the columns of the matrix X defined by

$$X = V_1 V_2 \ldots V_s \, . \qquad (9.2)$$

If the eigenvectors are required, then at each stage of the process we may store the current product $V_1 V_2 \ldots V_i$. This requires n^2 storage locations in addition to the $\frac{1}{2}(n^2+n)$ needed for the upper triangles of the A_i. This scheme automatically gives all n eigenvectors.

10. COMPUTATIONAL ASPECTS OF THE ALGORITHM

We are now in a position to discuss in detail the implementation of the algorithm that we have used. Various schemes have been proposed in the past; Jacobi (1846) inspected the absolutely largest off-diagonal element a_{pq} and then chose the angle of rotation ϕ such that in the matrix A_{k+1} the (p,q) element was zero. After the rediscovery of the method by Gregory (1953), when for the first time it was used in automatic computing, it was

applied in such a way that p and q ran row-wise through all super-diagonal elements of the matrix and again the rotation angle ϕ was chosen every time to annihilate the (p,q) element of the matrix A_{k+1}. Later Pope and Tompkins (1957) suggested a strategy which tended to avoid inefficient rotations and thus achieved diagonalisation with less effort. In practice we have found the proposals of Pope and Tompkins to be of little application and our basic algorithm is due to Rutishauser (1966).

The pivots of the Jacobi rotations are chosen by row-wise scanning of the upper triangle of A. Before each sweep we calculate

$$\sigma = \sum_{p=1}^{n-1} \sum_{q=p+1}^{n} |a_{pq}| \qquad (10.1)$$

and during the first three sweeps it performs only those rotations for which

$$|a_{pq}| > 0.2\sigma/n^2 = h \; . \qquad (10.2)$$

In the later sweeps h is set to zero. If σ is less than or equal to some preset tolerance the process is terminated. The tolerance may be altered by the user but we have taken a typical value to be $\frac{1}{2}n(n-1).m$ where m is the smallest number for which

$$1.0 + m \neq 1.0 \qquad (10.3)$$

on the machine being used.

If before the (p,q) rotation the element a_{pq} is small compared to a_{pp} and small compared to a_{qq},

then a'_{pq} is set to zero and the transformation is skipped. By small we mean that the addition of $100.|a_{pq}|$ to both a_{pp} and a_{qq} does not alter the two diagonal elements on the machine being used. This is certainly meaningful as it produces an error no larger than would be produced if the rotation had been performed. However, in order that the procedure can be used on perturbed diagonal matrices this device is suppressed during the first four sweeps.

In order to illustrate the difference between our stopping criteria and Rutishauser's consider the following example of order 2. Let

$$A = \begin{pmatrix} a & \varepsilon \\ \varepsilon & b \end{pmatrix} \tag{10.4}$$

where

$$\varepsilon = m/2 \tag{10.5}$$

and a and b are approximately of order unity. Based on our stopping criteria the matrix A is diagonal and we should accept the eigenvalues

$$\lambda_1 = a \ , \quad \lambda_2 = b \tag{10.6}$$

and the corresponding matrix of eigenvectors

$$V = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{10.7}$$

Rutishauser's algorithm would however perform a rotation to give, to machine precision, the eigen-

values

$$\lambda_1 = a \ , \quad \lambda_2 = b$$

and the corresponding matrix of eigenvectors

$$V = \begin{pmatrix} 2^{-\frac{1}{2}} & 2^{-\frac{1}{2}} \\ 2^{-\frac{1}{2}} & -2^{-\frac{1}{2}} \end{pmatrix}. \qquad (10.9)$$

We think it unlikely that there would be many instances in which it was advantageous to perform the extra rotation and in our particular application to simultaneous iteration it is often disastrous.

In order to annihilate the (p,q) element the rotation parameters c (=cosϕ) and s (=sinϕ) are computed as follows. Firstly, instead of taking

$$\tan 2\phi = \frac{2a_{pq}}{a_{qq} - a_{pp}} \qquad (10.10)$$

we compute

$$\theta = \cot 2\phi = (a_{qq} - a_{pp})/2a_{pq} \qquad (10.11)$$

giving

$$\frac{1}{\theta} = \tan 2\phi = \frac{2t}{1-t^2} \quad \text{where } t = \tan\phi. \qquad (10.12)$$

From (10.12) we have

$$1 - t^2 = 2\theta t \qquad (10.13)$$

or

$$t^2 + 2\theta t - 1 = 0. \qquad (10.14)$$

We take

$$t = -\theta \pm (1 + \theta^2)^{\frac{1}{2}} \qquad (10.15)$$

where we require the smaller (in modulus) root.

If $|\theta|$ is large we do not use the above but instead we take

$$t = 1/2\theta. \tag{10.16}$$

To find whether $|\theta|$ is large we form

$$h = a_{qq} - a_{pp} \tag{10.17}$$

and test if

$$h + 100.|a_{pq}| = h \tag{10.18}$$

to machine precision.

The solution of (10.15) which we require is given by

$$t = -\theta + (1 + \theta^2)^{\frac{1}{2}} = 1/(\theta + (1 + \theta^2)^{\frac{1}{2}})$$
$$\text{if } \theta \geq 0 \tag{10.19}$$

and

$$t = |\theta| - (1 + \theta^2)^{\frac{1}{2}} = -1/(|\theta| + (1 + \theta^2)^{\frac{1}{2}})$$
$$\text{if } \theta < 0. \tag{10.20}$$

From t we easily calculate

$$c = \cos\phi = 1/(1 + t^2)^{\frac{1}{2}} \tag{10.21}$$

whence

$$s = \sin\phi = t.\cos\phi \tag{10.22}$$

and

$$\tau = \tan(\phi/2) = \sin\phi/(1 + \cos\phi). \tag{10.22}$$

Finally we mention that the program does not operate directly on the diagonal elements of the matrix A but transfers them to a one-dimensional array d and then acts upon this. In order to ensure maximum accuracy of the eigenvalues, as well as updating d at each rotation (for the purposes of

calculations performed during any particular sweep), the alterations to d are also accumulated separately and at the end of each sweep these totalled increments of all the diagonal elements during the sweep are used to compute new and better values of the latter. We give more details in chapter 5.

11. NUMERICAL DETAILS

In an attempt to diminish the accumulation of round-off errors the following computational formulae are used. To calculate c,s and τ we proceed as follows. Let

$$h = a_{qq} - a_{pp} \tag{11.1}$$

$$\theta = \tfrac{1}{2}h/a_{pq} \tag{11.2}$$

$$t = 1/(|\theta| + (1 + \theta^2)^{\frac{1}{2}}) \quad \text{if } \theta \geq 0 \text{ or}$$

$$t = -1/(|\theta| + (1 + \theta^2)^{\frac{1}{2}}) \quad \text{if } \theta < 0 \tag{11.3}$$

$$c = 1/(1 + t^2)^{\frac{1}{2}} \tag{11.4}$$

$$s = t.c \tag{11.5}$$

$$\tau = s/(1 + c) \; . \tag{11.6}$$

If θ is large we replace (11.3) by $t=1/2\theta$. We modify the formulae (2.6) and (2.7) as follows. We know

$$\tan 2\phi = 2a_{pq}/(a_{qq} - a_{pp}) \tag{11.7}$$

therefore

$$\frac{\tan\phi}{1 - \tan^2\phi} = \frac{a_{pq}}{a_{qq} - a_{pp}} \; . \tag{11.8}$$

Hence, from

$$a'_{pp} = a_{pp}\cos^2\phi + a_{qq}\sin^2\phi - 2a_{pq}\sin\phi\cos\phi \qquad (11.9)$$

we have

$$\begin{aligned} a'_{pp} &= a_{pp} + (a_{qq} - a_{pp})\sin^2\phi - 2a_{pq}\sin\phi\cos\phi \\ &= a_{pp} + a_{pq}(1 - \tan^2\phi)\frac{\sin^2\phi}{\tan\phi} - 2a_{pq}\sin\phi\cos\phi \\ &= a_{pp} - a_{pq}\sin\phi\cos\phi - a_{pq}\tan^2\phi\sin\phi\cos\phi \\ &= a_{pp} - a_{pq}\sin\phi\cos\phi\sec^2\phi \\ &= a_{pp} - a_{pq}\tan\phi \ . \end{aligned} \qquad (11.10)$$

Hence, we use the computational formulae

$$a'_{pp} = a_{pp} - t.a_{pq} \qquad (11.11)$$

$$a'_{qq} = a_{qq} + t.a_{pq} \qquad (11.12)$$

$$a'_{pq} = 0 \ . \qquad (11.13)$$

Similarly for the off-diagonal elements of A, and also for the components of V, we do not use equations (2.5) but instead modify them in a similar manner. For if

$$a'_{pj} = a_{pj}\cos\phi - a_{qj}\sin\phi \qquad (11.14)$$

we have

$$\begin{aligned} a'_{pj} &= a_{pj}(1 - 2\sin^2\tfrac{1}{2}\phi) - a_{qj}\sin\phi \\ &= a_{pj}\left(1 - \frac{\sin\phi}{\cos\tfrac{1}{2}\phi}.\sin\tfrac{1}{2}\phi\right) - a_{qj}\sin\phi \\ &= a_{pj} - \sin\phi(a_{qj} + a_{pj}\tan\tfrac{1}{2}\phi). \end{aligned} \qquad (11.15)$$

It might be thought that analagously to (11.11) we would use

$$a'_{pj} = \cos\phi(a_{pj} - a_{qj}\tan\phi) \qquad (11.16)$$

offers no advantage since its rounding error
but this ~~suffers from the disadvantage that~~ $\tan\phi$
is similar to that of (11.14). When ϕ is small,
~~can take all values in the range $[-\infty,\infty]$.~~ ~~Equation~~
smaller than
(11.15) has a ~~maximum~~ rounding error ~~half~~ that of (11.14). Hence, we use the computational formulae

$$\begin{aligned} a'_{pj} &= a_{pj} - s.(a_{qj} + \tau.a_{pj}), \\ a'_{qj} &= a_{qj} + s.(a_{pj} - \tau.a_{qj}) . \end{aligned} \qquad (11.17)$$

In the program we work only with the upper triangle of elements of A and hence equations (11.17) are only used if the appropriate a_{pj} and a_{qj} are in this upper triangle. In fact we use three pairs of equations to modify the elements of the p- and q-th row and column as follows. For $1 \leqslant j \leqslant p-1$ we take

$$\begin{aligned} a'_{jp} &= a_{jp} - s.(a_{jq} + a_{jp}.\tau) \\ a'_{jq} &= a_{jq} + s.(a_{jp} - a_{jq}.\tau) , \end{aligned} \qquad (11.18)$$

for $p+1 \leqslant j \leqslant q-1$

$$\begin{aligned} a'_{pj} &= a_{pj} - s.(a_{jq} + a_{pj}.\tau) \\ a'_{jq} &= a_{jq} + s.(a_{pj} - a_{jq}.\tau) , \end{aligned} \qquad (11.19)$$

and for $q+1 \leqslant j \leqslant n$

$$\begin{aligned} a'_{pj} &= a_{pj} - s.(a_{qj} + a_{pj}.\tau) \\ a'_{qj} &= a_{qj} + s.(a_{pj} - a_{qj}.\tau) . \end{aligned} \qquad (11.20)$$

This completes the details of the algorithm; we discuss the programming aspects in chapter 5 and a listing of the program is given in appendix 1.

12. THE JACOBI METHOD FOR HERMITIAN MATRICES

Nearly all of the results we obtained in the previous sections for real symmetric matrices carry over to the case when A is Hermitian. We define the rotation matrix V of order n such that

$$\begin{aligned} v_{pp} &= v_{qq} = \cos\phi \\ v_{pq} &= e^{i\theta}\sin\phi \\ v_{qp} &= -e^{-i\theta}\sin\phi \\ v_{ii} &= 1 , \quad i \neq p,q \\ v_{ij} &= 0 \quad \text{otherwise.} \end{aligned} \tag{12.1}$$

We note that

$$VV^H = V^HV = I . \tag{12.2}$$

Denoting the original matrix by A_0 we may describe the process by the sequence of matrices

$$A_k = V_k^H A_{k-1} V_k . \tag{12.3}$$

As A_{k-1} is Hermitian we see that A_k is also. In a similar notation to section 2 the modified elements are given by

$$a'_{ij} = a_{ij} , \quad i,j \neq p,q \tag{12.4}$$

$$\left.\begin{aligned} a'_{pj} &= a_{pj}\cos\phi - a_{qj}e^{i\theta}\sin\phi \\ a'_{qj} &= a_{qj}\cos\phi + a_{pj}e^{-i\theta}\sin\phi \end{aligned}\right\} \quad j \neq p,q \tag{12.5}$$

$$\begin{aligned} a'_{pp} &= a_{pp}\cos^2\phi + a_{qq}\sin^2\phi \\ &\quad - \sin\phi\cos\phi(a_{pq}e^{-i\theta} + a_{qp}e^{i\theta}) \end{aligned}$$

$$a'_{qq} = a_{qq}\cos^2\phi + a_{pp}\sin^2\phi + \sin\phi\cos\phi(a_{pq}e^{-i\theta} + a_{qp}e^{i\theta}) \qquad (12.6)$$

$$a'_{pq} = a_{pq}\cos^2\phi - a_{qp}e^{2i\theta}\sin^2\phi + e^{i\theta}\sin\phi\cos\phi(a_{pp} - a_{qq}) \; . \qquad (12.7)$$

Again we notice that the transformation given by (12.3) leaves $N^2(A)$ invariant and that A tends to diagonal form if and only if $t^2(A)\to 0$. Again it is possible to reduce a_{pq} to zero at any stage. For if

$$a'_{pq} = a_{pq}\cos^2\phi - a_{qp}e^{2i\theta}\sin^2\phi + e^{i\theta}\sin\phi\cos\phi(a_{pp} - a_{qq}) = 0 \; . \qquad (12.8)$$

Then

$$a_{pq}e^{-i\theta}\cos^2\phi - a_{qp}e^{i\theta}\sin^2\phi = (a_{qq} - a_{pp})\sin\phi\cos\phi \; , \qquad (12.9)$$

thus

$$a_{pq}(\cos\theta - i\sin\theta)\cos^2\phi - a_{qp}(\cos\theta + i\sin\theta)\sin^2\phi = \tfrac{1}{2}(a_{qq} - a_{pp})\sin 2\phi \; . \qquad (12.10)$$

Therefore

$$(\cos^2\phi - \sin^2\phi)(\mathrm{Re}(a_{pq})\cos\theta + \mathrm{Im}(a_{pq})\sin\theta) + i[(\cos^2\phi + \sin^2\phi)(\mathrm{Im}(a_{pq})\cos\theta - \mathrm{Re}(a_{pq})\sin\theta)] = \tfrac{1}{2}(a_{qq} - a_{pp})\sin 2\phi \; . \qquad (12.11)$$

Hence, taking

$$\mathrm{Im}(a_{pq})\cos\theta = \mathrm{Re}(a_{pq})\sin\theta \qquad (12.12)$$

gives

$$\tan\theta = \mathrm{Im}(a_{pq})/\mathrm{Re}(a_{pq}) \qquad (12.13)$$

and

$$\tan 2\phi = \frac{2(\mathrm{Re}(a_{pq})\cos\theta + \mathrm{Im}(a_{pq})\sin\theta)}{(a_{qq} - a_{pp})} . \qquad (12.14)$$

Therefore

$$\frac{\tan\phi}{1 - \tan^2\phi} = \frac{\Omega}{a_{qq} - a_{pp}} \qquad (12.15)$$

where

$$\Omega = \mathrm{Re}(a_{pq})\cos\theta + \mathrm{Im}(a_{pq})\sin\theta . \qquad (12.16)$$

As before we restrict ourselves to implementing the threshold special serial method and rearrange equations (12.5), (12.6) and (12.7) into a more convenient computational form. These then become

$$a'_{pp} = a_{pp} - \Omega\tan\phi$$

$$a'_{qq} = a_{qq} + \Omega\tan\phi \qquad (12.17)$$

$$a'_{pq} = 0 \qquad (12.18)$$

where Ω is defined as in (12.16). Analogously to equations (11.17) we obtain

$$a'_{pj} = a_{pj} - \sin\phi(a_{qj}e^{i\theta} + \tau.a_{pj})$$

and

$$a'_{qj} = a_{qj} + \sin\phi(a_{pj}e^{-i\theta} - \tau.a_{qj}) . \qquad (12.19)$$

As before the program works only with the elements

in the upper triangle of A and hence we again use three pairs of equations for computational purposes. These are similar to those employed in the symmetric case. For example, if $1 \leqslant j \leqslant p-1$ we use

$$a'_{jp} = a_{jp} - s(a_{jq}e^{i\theta} + a_{jp}.\tau)$$

$$a'_{jq} = a_{jq} + s(a_{jp}e^{i\theta} - a_{jq}.\tau) \quad . \qquad (12.20)$$

The complex arithmetic is performed by breaking all equations down into their real and imaginary parts.

The only other feature of the program which does not occur in the symmetric case is the computation of $\cos\theta$ and $\sin\theta$. From (12.13) we form

$$t = \tan\theta \quad . \qquad (12.21)$$

If $|t| \leqslant 1$ then form

$$\cos\theta = 1/(1 + t^2)^{\frac{1}{2}} \qquad (12.22)$$

and

$$\sin\theta = t.\cos\theta \quad . \qquad (12.23)$$

If, however, $|t| > 1$ we take

$$t_1 = \frac{1}{t} = \frac{\mathrm{Re}(a_{pq})}{\mathrm{Im}(a_{pq})} \qquad (12.24)$$

and form

$$\sin\theta = 1/(1 + t_1^2)^{\frac{1}{2}} \qquad (12.25)$$

$$\cos\theta = t_1.\sin\theta \quad . \qquad (12.26)$$

The other computational details are identical with the symmetric case. Programming details are given in chapter 5 and a listing of the program is included in appendix 2.

All the convergence results given earlier carry

over to the Hermitian case with little or no additional analysis. The results of sections 3 and 4 require no modification as the eigenvalues and diagonal elements of a Hermitian matrix are real. The proof given in section 7 also holds, for we note that all the equations and inequalities involving the off-diagonal elements rely not on these elements but on their moduli. In addition, the proof nowhere depends on any of the off-diagonal elements being real. In particular, $|\exp(i\theta)|=1$.

13. THE JACOBI METHOD FOR NORMAL MATRICES

We have shown in chapter 1 that a normal matrix is the most general form for which there exists a unitary transformation such that

$$V^H A V = \mathrm{diag}(\lambda_i) \ . \tag{13.1}$$

We saw also that if A is normal we may write

$$A = B + C' \tag{13.2}$$

where B is Hermitian and C' is skew-Hermitian and

$$BC' = C'B \ . \tag{13.3}$$

If we now write

$$C' = iC \tag{13.4}$$

C is a Hermitian matrix and we may replace equations (13.2) and (13.3) by

$$A = B + iC \tag{13.5}$$

and

$$BC = CB \tag{13.6}$$

respectively, where B and C are now both Hermitian

matrices. Hence in order to diagonalise a normal matrix it is only necessary to diagonalise two Hermitian matrices. We now consider some relations between commuting matrices and their eigenvectors.

Lemma 13.1 If A is normal then A and A^H have a common complete system of eigenvectors.

Proof: From the relations

$$V^H A V = D \quad \text{and} \quad V^H A^H V = D^H \tag{13.7}$$

where $D=\text{diag}(\lambda_i)$, we have

$$AV = VD$$

and

$$A^H V = VD^H . \tag{13.8}$$

The matrices A and A^H therefore have a common complete system of eigenvectors, namely that formed by the columns of V.

Lemma 13.2 If any two matrices B and C have a common complete system of eigenvectors then they commute.

Proof: Suppose the common system of eigenvectors forms the columns of the matrix V. Then we have

$$B = VD_1V^{-1} \quad \text{and} \quad C = VD_2V^{-1} . \tag{13.9}$$

Hence

$$BC = VD_1V^{-1}VD_2V^{-1} = VD_1D_2V^{-1} \tag{13.10}$$

and

$$CB = VD_2V^{-1}VD_1V^{-1} = VD_2D_1V^{-1} . \tag{13.11}$$

As diagonal matrices commute so do B and C.

Theorem 13.3 If B and C commute and have linear elementary divisors then they share a common system

of eigenvectors.

Proof: Suppose the eigenvectors of B are $v_1, v_2, \ldots, v_n$ and that these vectors form the columns of the matrix V. Then

$$V^{-1}BV = \text{diag}(\lambda_i) \tag{13.12}$$

where the λ_i are not necessarily distinct. The manner of proof is dependent on the number of repeated eigenvalues and their respective multiplicities but an example will suffice to show the method. Let us suppose that

$$\lambda_1 = \lambda_2 = \ldots = \lambda_l \tag{13.13}$$

$$\lambda_{l+1} = \lambda_{l+2} = \ldots = \lambda_m \tag{13.14}$$

and that $\lambda_{m+1}, \lambda_{m+2}, \ldots, \lambda_n$ are all distinct. Equation (13.12) may then be written as

$$V^{-1}BV = \begin{bmatrix} \lambda_1 I_l & & & & \\ & \lambda_{l+1} I_{m-l-1} & & & \\ & & \lambda_{m+1} & & \\ & & & \ldots & \\ & & & & \lambda_n \end{bmatrix}. \tag{13.15}$$

Since

$$Bv_i = \lambda_i v_i \tag{13.16}$$

we have

$$CBv_i = \lambda_i Cv_i \tag{13.17}$$

which gives

$$B(Cv_i) = \lambda_i (Cv_i) . \tag{13.18}$$

Thus if v_i is an eigenvector of B corresponding to λ_i then (Cv_i) lies in the subspace of eigenvectors corresponding to λ_i. Hence

$$
\begin{aligned}
Cv_1 &= p_{11}v_1 + p_{21}v_2 + \cdots + p_{l1}v_l \\
Cv_2 &= p_{12}v_1 + p_{22}v_2 + \cdots + p_{l2}v_l \\
&\cdots\cdots\cdots\cdots\cdots \\
Cv_l &= p_{11}v_1 + p_{21}v_2 + \cdots + p_{l1}v_l \\
Cv_{l+1} &= q_{11}v_1 + q_{21}v_2 + \cdots + q_{m1}v_m \\
&\cdots\cdots\cdots\cdots\cdots \\
Cv_m &= q_{1m}v_1 + q_{2m}v_2 + \cdots + q_{mm}v_m \\
Cv_{m+1} &= \mu_{m+1}v_{m+1} \\
&\cdots\cdots \\
Cv_n &= \mu_n v_n \,. \qquad (13.19)
\end{aligned}
$$

These equations may be written in matrix form as

$$
V^{-1}CV = \begin{bmatrix} P & & & & \\ & Q & & & \\ & & \mu_{m+1} & & \\ & & & \cdots & \\ & & & & \mu_n \end{bmatrix}. \qquad (13.20)
$$

Since C has linear elementary divisors it follows that so too must P and Q. Hence there exist matrices T and U such that

$$
T^{-1}PT = \begin{bmatrix} \mu_1 & & \\ & \cdots & \\ & & \mu_l \end{bmatrix} \text{ and } U^{-1}QU = \begin{bmatrix} \mu_{l+1} & & \\ & \cdots & \\ & & \mu_m \end{bmatrix}. \qquad (13.21)
$$

Writing

$$W = \begin{bmatrix} T & & \\ & U & \\ & & I_{n-m} \end{bmatrix} \tag{13.22}$$

we have

$$W^{-1}V^{-1}CVW = \text{diag}(\mu_i) \ . \tag{13.23}$$

However, from (13.15) we obtain

$$W^{-1}V^{-1}BVW = \text{diag}(\lambda_i) \ . \tag{13.24}$$

Hence B and C share a common system of eigenvectors, namely the vectors which are the columns of VW.

We may make use of this theorem to enable us to diagonalise a normal matrix. In equations (13.5) and (13.6) the matrices B and C commute and, as they are Hermitian, both have linear elementary divisors. From (13.5) we may determine a matrix V such that

$$V^{H}BV = \text{diag}(\lambda_i(B)). \tag{13.25}$$

Thus

$$V^{H}AV = \text{diag}(\lambda_i(B)) + iV^{H}CV \tag{13.26}$$

From theorem (13.3) we see that $V^{H}CV$ will be diagonal only if there are no repeated eigenvalues. If it is not diagonal however it follows that we can certainly find a matrix W such that

$$W^{H}V^{H}AVW = \text{diag}(\lambda_i(B)) + i.\text{diag}(\lambda_i(C)) \ . \tag{13.27}$$

A similar algorithm could be devised which was based upon first diagonalising the matrix C. In our algorithm we use, as before, the threshold

special serial Jacobi method on the matrix A but base the transformation on B or C depending upon which one gives the greater reduction in $t^2(A)$.

For a normal matrix A it is necessary to operate on the whole of the matrix A or on the two upper triangles of B and C. We opt for the former as it is extremely easy to calculate the elements of B and C when they are required and much more economical to perform one set of rotations on A.

Computationally the program follows that for Hermitian matrices with only two exceptions. Firstly it is necessary to calculate b_{pq} and c_{pq}; these follow directly from (15.16) and (15.17) of chapter 1. Secondly it is necessary to calculate a_{pq} as we are not necessarily reducing this to zero. A full listing of the program is given in appendix 3.

Convergence of our method for normal matrices relies only on the convergence of Jacobi's method for Hermitian matrices and this we have discussed in the previous section. We note that although a normal matrix may have complex eigenvalues these are dealt with as an ordered pair of reals thus enabling us to use the results from the Hermitian case.

Finally we must mention the work of Goldstine and Horwitz (1959) who developed an optimal algorithm for diagonalising normal matrices. Ruhe (1967) proved that for row cyclic choice of pivots

this algorithm was ultimately quadratically convergent; Loizou (1972) established the same result for maximal pivot choice. Unfortunately the Golstine and Horwitz algorithm is difficult to implement and we believe our algorithm to be very competitive. This is because it is easy to implement, runs nearly as quickly as the algorithm for Hermitian matrices and is very close to being optimal.

14. GENERALISED JACOBI METHODS

Many attempts have been made to generalise the well-known Jacobi algorithm for the diagonalisation of a real symmetric matrix to arbitrary matrices. We have just discussed an extension for normal matrices but for a non-normal matrix it is no longer possible to use unitary transformations alone although, as we have shown in chapter 1, it is possible to use unitary transformations to triangularise an arbitrary matrix. Attempts to formulate computational methods for doing this have not been very successful as shown by Causey (1958), Greenstadt (1955), and Lotkin (1956). Another way of generalising the Jacobi method is to use both unitary and non-unitary transformations and attempt to diagonalise the matrix. It is essential to restrict the non-unitary transformations and use them only to bring the matrix closer to normal.

Eberlein (1962) and Rutishauser (1964) have both proposed such methods, the former using non-unitary transformations and the latter using diagonal similarity transformations. Eberlein has proved the global convergence of her method and Ruhe (1968) has shown a slight modification of the algorithm to be ultimately quadratically convergent for non-defective matrices. No convergence proofs of Rutishauser's algorithm appear to exist and such attempts as we have made lead us to suspect that the method is not of general applicability.

15. MINIMISATION OF MATRIX NORMS

We prove firstly a theorem due to Mirsky (1958) upon which the attempts to diagonalise a matrix are based.

Lemma 15.1 Schur states that if A is a complex (n*n) matrix with eigenvalues $\lambda_1,\lambda_2,\ldots,\lambda_n$ then

$$\|A\|_E^2 \geq \sum_{i=1}^{n} |\lambda_i|^2 . \qquad (15.1)$$

There is equality if and only if A is normal.

Proof: For any matrix A there exists a unitary matrix R such that

$$R^H A R = T \qquad (15.2)$$

where T is a triangular matrix whose diagonal elements are the eigenvalues of A. We know that if A is normal T is diagonal.

From (15.2) we have

$$A = RTR^H \tag{15.3}$$

which gives us

$$\|A\|_E = \|RTR^H\|_E$$

$$= \|T\|_E \ . \tag{15.4}$$

This follows because the Euclidean length of each column of RT is equal to that of the corresponding column of T. Since

$$\|T\|_E \geq \left[\sum_{i=1}^{n} |\lambda_i|^2\right]^{\frac{1}{2}} \tag{15.5}$$

with equality if and only if T is diagonal the lemma is proved.

Theorem 15.2 Let A be an (n*n) arbitrary complex matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ then

$$\inf\|S^{-1}AS\|_E^2 = \sum_{i=1}^{n} |\lambda_i|^2 \tag{15.6}$$

where the lower bound is taken with respect to all non-singular matrices S. Furthermore the lower bound is attained if and only if A is normal.

Proof: Denote by T_1 a matrix such that $T_1^{-1}AT_1$ is triangular, say

$$T_1^{-1}AT_1 = \begin{bmatrix} \lambda_1 & b_{12} & \cdots & b_{1n} \\ & \lambda_2 & \cdots & b_{2n} \\ & & \cdots & \\ & & & \lambda_n \end{bmatrix} . \tag{15.7}$$

Let $|\delta|$ be not greater than unity and define

$$T_2 = \mathrm{diag}(1, \delta, \delta^2, \ldots, \delta^{n-1}) \ . \qquad (15.8)$$

If $S_0 = T_1 T_2$ we have

$$\|S_0^{-1} A S_0\|_E^2 = \|T_2^{-1} T_1^{-1} A T_1 T_2\|_E^2$$

$$= \sum_{i=1}^{n} |\lambda_i|^2 + \sum_{1 \leq i < k \leq n} |b_{ik}|^2 \delta^{2(k-i)}$$

$$\leq \sum_{i=1}^{n} |\lambda_i|^2 + \tfrac{1}{2} n(n-1) b^2 \delta^2 \qquad (15.9)$$

where $b = \max_{i,k} |b_{ik}|$. The relation (15.6) now follows since by (15.1)

$$\|S^{-1} A S\|_E^2 \geqslant \sum_{i=1}^{n} |\lambda_i|^2 \qquad (15.10)$$

for all non-singular S. Further, if A is diagonalisable then the lower bound of $\|S^{-1} A S\|_E$ is clearly attained. If, on the other hand, the bound is attained so that, for some S,

$$\|S^{-1} A S\|_E^2 = \sum_{i=1}^{n} |\lambda_i|^2 \qquad (15.11)$$

then $S^{-1} A S$ is normal and A is diagonalisable.

16. EBERLEIN'S ALGORITHM

Mirsky's theorem on the minimisation of matrix norms gave Eberlein (1962) the idea for a method of making complex matrices arbitrarily close to normal form. She proved global convergence, with some restriction on the choice of pivot elements, and found experimentally that an extension of the procedure produced a nearly diagonal matrix.

Ruhe (1968) has proved that slight modification of Eberlein's original 1962 algorithm is ultimately quadratically convergent for row cyclic choice of pivot. Using the 1962 paper Eberlein (1970) has published a program to reduce arbitrary complex matrices to nearly diagonal form. It is this algorithm which forms the basis of our program.

17. THE TRANSFORMATION MATRICES

As we have already mentioned the basic philosophy of Eberlein's method is to use shear matrices to reduce a matrix A arbitrarily close to normal form. It then may be made almost diagonal as we have shown previously. In practice the shears and rotations are performed together to minimise the arithmetic operations required but for theoretical purposes it is much more convenient to consider them separately. We shall concentrate firstly on the effects of shearing. All the results that follow have been proved by Eberlein but we consider our proofs to be shorter and simpler to understand.

Define the shear matrix S by

$$\begin{aligned} s_{pp} &= s_{qq} = \cosh y \\ s_{pq} &= \bar{s}_{qp} = ie^{i\beta}\sinh y \ , \quad y \text{ and } \beta \text{ real} \\ s_{ij} &= \delta_{ij} \ , \qquad i,j \neq p,q. \end{aligned} \tag{17.1}$$

For arbitrary A the elements of the transformed

matrix A' defined by

$$A' = S^{-1}AS \tag{17.2}$$

are given by

$$a'_{ij} = a_{ij} \qquad i,j \neq p,q$$

$$\left.\begin{aligned} a'_{pi} &= a_{pi}\cosh y - ia_{qi}e^{i\beta}\sinh y \\ a'_{ip} &= a_{ip}\cosh y - ia_{iq}e^{-i\beta}\sinh y \\ a'_{qi} &= a_{qi}\cosh y + ia_{pi}e^{-i\beta}\sinh y \\ a'_{iq} &= a_{iq}\cosh y + ia_{ip}e^{i\beta}\sinh y \end{aligned}\right\} \quad i \neq p,q \tag{17.3}$$

$$a'_{pp} = a_{pp}\cosh^2 y - a_{qq}\sinh^2 y - i\sinh y\cosh y(a_{pq}e^{-i\beta} + a_{qp}e^{i\beta}) \tag{17.4}$$

$$a'_{pq} = a_{pq}\cosh^2 y + a_{qp}e^{2i\beta}\sinh^2 y + ie^{i\beta}\sinh y\cosh y(a_{pp} - a_{qq}) \tag{17.5}$$

$$a'_{qp} = a_{qp}\cosh^2 y + a_{pq}e^{-2i\beta}\sinh^2 y + ie^{-i\beta}\sinh y\cosh y(a_{pp} - a_{qq}) \tag{17.6}$$

$$a'_{qq} = a_{qq}\cosh^2 y - a_{pp}\sinh^2 y + i\sinh y\cosh y(a_{pq}e^{-i\beta} + a_{qp}e^{i\beta}) . \tag{17.7}$$

Equations (17.4) to (17.7) may be written more simply if we use the following substitutions. Let

$$D = a_{pp} - a_{qq} \tag{17.8}$$

$$E = a_{pq} + a_{qp} \tag{17.9}$$

$$F = a_{qp} - a_{pq} \tag{17.10}$$

$$B = a_{pq}e^{-i\beta} + a_{qp}e^{i\beta} \qquad (17.11)$$

$$C = a_{pq}e^{-i\beta} - a_{qp}e^{i\beta} \ . \qquad (17.12)$$

From (17.4) we have

$$\begin{aligned} a'_{pp} &= a_{qq} - a_{qq}\cosh^2 y + a_{pp}\cosh^2 y - \tfrac{1}{2}i\sinh 2y.B \\ &= a_{qq} + D\cosh^2 y - \tfrac{1}{2}i\sinh 2y.B \\ &= a_{qq} + \tfrac{1}{2}D(1+\cosh 2y) - \tfrac{1}{2}iB\sinh 2y \\ &= \tfrac{1}{2}\left[(a_{pp}+a_{qq}) + D\cosh 2y - iB\sinh 2y\right] \ . \end{aligned} \qquad (17.13)$$

From (17.7) we have

$$\begin{aligned} a'_{qq} &= a_{qq} + a_{qq}\sinh^2 y - a_{pp}\sinh^2 y + \tfrac{1}{2}i\sinh 2y.B \\ &= a_{qq} - D\sinh^2 y + \tfrac{1}{2}iB\sinh 2y \\ &= a_{qq} - \tfrac{1}{2}D(\cosh 2y - 1) + \tfrac{1}{2}iB\sinh 2y \\ &= \tfrac{1}{2}\left[(a_{pp}+a_{qq}) - D\cosh 2y + iB\sinh 2y\right] \ . \end{aligned} \qquad (17.14)$$

From (17.5)

$$\begin{aligned} a'_{pq} &= e^{i\beta}(a_{pq}e^{-i\beta}\cosh^2 y + a_{qp}e^{i\beta}\sinh^2 y + \tfrac{1}{2}iD\sinh 2y) \\ &= e^{i\beta}\left[\tfrac{1}{2}a_{pq}e^{-i\beta} - \tfrac{1}{2}a_{qp}e^{i\beta}\right. \\ &\quad \left. + \tfrac{1}{2}(a_{pq}e^{-i\beta} + a_{qp}e^{i\beta})\cosh 2y + \tfrac{1}{2}iD\sinh 2y\right] \\ &= \tfrac{1}{2}e^{i\beta}(a_{pq}e^{-i\beta} - a_{qp}e^{i\beta} + B\cosh 2y + iD\sinh 2y) \\ &= \tfrac{1}{2}e^{i\beta}(C + B\cosh 2y + iD\sinh 2y) \ . \end{aligned} \qquad (17.15)$$

From (17.6)

$$a'_{qp} = e^{-i\beta}(a_{qp}e^{i\beta}\cosh^2 y + a_{pq}e^{-i\beta}\sinh^2 y + \tfrac{1}{2}iD\sinh 2y)$$

$$= \tfrac{1}{2}e^{-i\beta}(a_{qp}e^{i\beta} - a_{pq}e^{-i\beta} + B\cosh 2y + iD\sinh 2y)$$

$$= \tfrac{1}{2}e^{-i\beta}(-C + B\cosh 2y + iD\sinh 2y)\ . \qquad (17.16)$$

Hence we write equations (17.4) to (17.7) in the form

$$a'_{pp} = \tfrac{1}{2}[(a_{pp}+a_{qq}) + D\cosh 2y - iB\sinh 2y] \qquad (17.17)$$

$$a'_{pq} = \tfrac{1}{2}e^{i\beta}(C + B\cosh 2y + iD\sinh 2y) \qquad (17.18)$$

$$a'_{qp} = \tfrac{1}{2}e^{-i\beta}(-C + B\cosh 2y + iD\sinh 2y) \qquad (17.19)$$

$$a'_{qq} = \tfrac{1}{2}[(a_{pp}+a_{qq}) - D\cosh 2y + iB\sinh 2y]\ . \qquad (17.20)$$

We define the commutator matrix C such that

$$C = AA^H - A^HA \qquad (17.21)$$

and note that C is Hermitian. An arbitrary matrix A is normal if and only if

$$C \equiv 0. \qquad (17.22)$$

We also introduce Δ defined by

$$\Delta \equiv N^2(A) - N^2(A'). \qquad (17.23)$$

It is obvious that, to bring a matrix closer to normal form, we shall be interested in positive Δ at each step and ideally in maximising Δ at each step.

We now compute Δ for the single transformation of (17.2). Let

$$\Delta_1 \equiv \sum_{i\neq p,q}(|a_{pi}|^2+|a_{ip}|^2+|a_{qi}|^2+|a_{iq}|^2)$$

$$- \sum_{i\neq p,q}(|a'_{pi}|^2+|a'_{ip}|^2+|a'_{qi}|^2+|a'_{iq}|^2)\ . \qquad (17.24)$$

From equations (17.3) we obtain

$$|a'_{pi}|^2 = (a_{pi}\cosh y - ia_{qi}e^{i\beta}\sinh y)$$
$$*(\bar{a}_{pi}\cosh y + i\bar{a}_{qi}e^{-i\beta}\sinh y)$$
$$= a_{pi}\bar{a}_{pi}\cosh^2 y + a_{qi}\bar{a}_{qi}\sinh^2 y$$
$$+ i\sinh y\cosh y(a_{pi}\bar{a}_{qi}e^{-i\beta} - \bar{a}_{pi}a_{qi}e^{i\beta}) \qquad (17.25)$$

$$|a'_{ip}|^2 = a_{ip}\bar{a}_{ip}\cosh^2 y + a_{iq}\bar{a}_{iq}\sinh^2 y$$
$$+ i\sinh y\cosh y(a_{ip}\bar{a}_{iq}e^{i\beta} - \bar{a}_{ip}a_{iq}e^{-i\beta}) \qquad (17.26)$$

$$|a'_{qi}|^2 = a_{qi}\bar{a}_{qi}\cosh^2 y + a_{pi}\bar{a}_{pi}\sinh^2 y$$
$$+ i\sinh y\cosh y(a_{pi}\bar{a}_{qi}e^{-i\beta} - \bar{a}_{pi}a_{qi}e^{i\beta}) \qquad (17.27)$$

and

$$|a'_{iq}|^2 = a_{iq}\bar{a}_{iq}\cosh^2 y + a_{ip}\bar{a}_{ip}\sinh^2 y$$
$$+ i\sinh y\cosh y(a_{ip}\bar{a}_{iq}e^{i\beta} - \bar{a}_{ip}a_{iq}e^{-i\beta}) . \qquad (17.28)$$

Hence

$$\Delta_1 = \sum_{i\neq p,q} [a_{pi}\bar{a}_{pi}(1-\cosh^2 y-\sinh^2 y)$$
$$+ a_{ip}\bar{a}_{ip}(1-\cosh^2 y-\sinh^2 y)$$
$$+ a_{qi}\bar{a}_{qi}(1-\cosh^2 y-\sinh^2 y)$$
$$+ a_{iq}\bar{a}_{iq}(1-\cosh^2 y-\sinh^2 y)$$

$$-2i\sinh y\cosh y(a_{pi}\bar{a}_{qi}e^{-i\beta}-\bar{a}_{pi}a_{qi}e^{i\beta}$$

$$+a_{ip}\bar{a}_{iq}e^{i\beta}-\bar{a}_{ip}a_{iq}e^{-i\beta})] \quad . \qquad (17.29)$$

Some simple manipulation then gives us

$$\Delta_1 = \sum_{i\neq p,q}\{(a_{pi}\bar{a}_{pi}+a_{ip}\bar{a}_{ip}+a_{qi}\bar{a}_{qi}+a_{iq}\bar{a}_{iq})(1-\cosh 2y)$$

$$- i\sinh 2y[e^{i\beta}(a_{ip}\bar{a}_{iq}-\bar{a}_{pi}a_{qi})$$

$$-e^{-i\beta}(\bar{a}_{ip}a_{iq}-a_{pi}\bar{a}_{qi})]\}$$

$$= \sum_{i\neq p,q}(|a_{pi}|^2+|a_{ip}|^2+|a_{qi}|^2+|a_{iq}|^2)(1-\cosh 2y)$$

$$- i\sinh 2y(e^{i\beta}K - e^{-i\beta}\bar{K})$$

$$= G(1-\cosh 2y) - H\sinh 2y \qquad (17.30)$$

where we define

$$K = \sum_{i\neq p,q}(a_{ip}\bar{a}_{iq} - \bar{a}_{pi}a_{qi}) \qquad (17.31)$$

$$H = i(e^{i\beta}K - e^{-i\beta}\bar{K})$$

$$= -2\mathrm{Im}(e^{i\beta}K) \qquad (17.32)$$

$$G = \sum_{i\neq p,q}(|a_{pi}|^2+|a_{ip}|^2+|a_{qi}|^2+|a_{iq}|^2) \quad . \qquad (17.33)$$

Now let

$$\Delta_2 \equiv |a_{pp}|^2 + |a_{qq}|^2 - |a'_{pp}|^2 - |a'_{qq}|^2 \quad . \qquad (17.34)$$

Then

$$\Delta_2 = a_{pp}\bar{a}_{pp}+a_{qq}\bar{a}_{qq}-\tfrac{1}{2}[(a_{pp}+a_{qq})(\bar{a}_{pp}+\bar{a}_{qq})+D\bar{D}\cosh^2 2y$$

$$-iB\bar{D}\sinh 2y\cosh 2y+i\bar{B}D\sinh 2y\cosh 2y+B\bar{B}\sinh^2 2y]$$

$$= \tfrac{1}{2}\big[a_{pp}\bar{a}_{pp}+a_{qq}\bar{a}_{qq}-a_{pp}\bar{a}_{qq}-\bar{a}_{pp}a_{qq}$$

$$- |D|^2\cosh^2 2y - |B|^2\sinh^2 2y$$

$$+ i(B\bar{D}-\bar{B}D)\sinh 2y\cosh 2y\big]$$

$$= \tfrac{1}{2}\big[(a_{pp}-a_{qq})(\bar{a}_{pp}-\bar{a}_{qq}) - |D|^2\cosh^2 2y$$

$$- |B|^2\sinh^2 2y + \tfrac{1}{2}i(B\bar{D}-\bar{B}D)\sinh 4y\big]$$

$$= \tfrac{1}{2}\big[-|D|^2\sinh^2 2y - |B|^2\sinh^2 2y + \tfrac{1}{2}i(B\bar{D}-\bar{B}D)\sinh 4y\big]$$

$$= -\tfrac{1}{2}(|B|^2+|D|^2)\sinh^2 2y + \frac{i}{4}(B\bar{D}-\bar{B}D)\sinh 4y\ . \qquad (17.35)$$

Finally we define

$$\Delta_3 \equiv |a_{pq}|^2 + |a_{qp}|^2 - |a'_{pq}|^2 - |a'_{qp}|^2. \qquad (17.36)$$

Then

$$\Delta_3 = |a_{pq}|^2 + |a_{qp}|^2 - \tfrac{1}{2}\big[C\bar{C} + B\bar{B}\cosh^2 2y$$

$$-iB\bar{D}\sinh 2y\cosh 2y + i\bar{B}D\sinh 2y\cosh 2y + D\bar{D}\sinh^2 2y\big]$$

$$= |a_{pq}|^2 + |a_{qp}|^2 - \tfrac{1}{2}\big[|a_{pq}|^2+|a_{qp}|^2$$

$$-a_{pq}\bar{a}_{qp}e^{-2i\beta} - a_{qp}\bar{a}_{pq}e^{2i\beta} + |B|^2\cosh^2 2y$$

$$+ |D|^2\sinh^2 2y + i(\bar{B}D-B\bar{D})\sinh 2y\cosh 2y\big]$$

$$= \tfrac{1}{2}\big[|a_{pq}|^2 + |a_{qp}|^2 + 2\mathrm{Re}(\bar{a}_{pq}a_{qp})e^{2i\beta}$$

$$- |B|^2\cosh^2 2y - |D|^2\sinh^2 2y$$

$$- i(\bar{B}D-B\bar{D})\sinh 2y\cosh 2y\big]$$

$$= \tfrac{1}{2}(|B|^2(1-\cosh^2 2y) - |D|^2\sinh^2 2y$$

$$- \tfrac{1}{2}i(\bar{B}D-B\bar{D})\sinh 4y)$$

$$= \tfrac{1}{2}[\,-(|B|^2+|D|^2)\sinh^2 2y - \tfrac{1}{2}i(\bar{B}D-B\bar{D})\sinh 4y] \ . \tag{17.38}$$

Combining the results for Δ_2 and Δ_3 we have

$$\Delta_2 + \Delta_3 = -(|B|^2+|D|^2)\sinh^2 2y - \tfrac{1}{2}i(\bar{B}D-B\bar{D})\sinh 4y$$
$$= \tfrac{1}{2}(|B|^2+|D|^2)(1-\cosh 4y) + \mathrm{Im}(\bar{B}D)\sinh 4y. \tag{17.39}$$

Hence we finally obtain

$$\Delta = \Delta_1 + \Delta_2 + \Delta_3$$
$$= G(1-\cosh 2y) - H\sinh 2y + \tfrac{1}{2}(|B|^2+|D|^2)(1-\cosh 4y)$$
$$+ \mathrm{Im}(\bar{B}D)\sinh 4y \ . \tag{17.40}$$

18. OPTIMAL SHEAR PARAMETERS

With the definition of the commutator matrix C given by equation (17.21) and the expression for Δ we obtained in (17.40) we are now in a position to prove the following.

Theorem 18.1 Setting the two first derivatives of Δ to zero for the maximum is equivalent to annihilating c'_{pq}. In fact we have the identity:

$$c'_{pq} = \frac{e^{i\beta}}{2}\left[\frac{i}{2}\cdot\frac{\partial\Delta}{\partial y} - \frac{1}{\sinh 2y}\cdot\frac{\partial\Delta}{\partial\beta}\right] \tag{18.1}$$

Proof: We recall the formulae of the last section

$$\Delta = G(1-\cosh 2y) - H\sinh 2y + \tfrac{1}{2}(|B|^2+|D|^2)(1-\cosh 4y)$$
$$+ \mathrm{Im}(\bar{B}D)\sinh 4y \tag{18.2}$$

$$G = \sum_{i\neq p,q}(|a_{pi}|^2+|a_{ip}|^2+|a_{qi}|^2+|a_{iq}|^2) \tag{18.3}$$

$$H = i(e^{i\beta}K - e^{-i\beta}\bar{K}) \tag{18.4}$$

$$K = \sum_{i\neq p,q} (a_{ip}\bar{a}_{iq} - \bar{a}_{pi}a_{qi}) \tag{18.5}$$

$$B = a_{pq}e^{-i\beta} + a_{qp}e^{i\beta} \tag{18.6}$$

$$C = a_{pq}e^{-i\beta} - a_{qp}e^{i\beta} \tag{18.7}$$

$$D = a_{pp} - a_{qq} \tag{18.8}$$

$$E = a_{pq} + a_{qp} \tag{18.9}$$

$$F = a_{qp} - a_{pq} \cdot \tag{18.10}$$

We note also that

$$c_{pq} = (AA^H - A^HA)_{pq} = \sum_{i=1}^{n} (a_{pi}\bar{a}_{qi} - \bar{a}_{ip}a_{iq}) \cdot \tag{18.11}$$

From (18.2) we have

$$\frac{\partial \Delta}{\partial y} = -2G\sinh 2y - 2H\cosh 2y - 2(|B|^2+|D|^2)\sinh 4y + 4\mathrm{Im}(\bar{B}D)\cosh 4y \cdot \tag{18.12}$$

Also, we have

$$\frac{\partial H}{\partial \beta} = -e^{i\beta}K - e^{-i\beta}\bar{K} \cdot \tag{18.13}$$

Now

$$|B|^2 = |a_{pq}|^2 + |a_{qp}|^2 + 2\mathrm{Re}(\bar{a}_{pq}a_{qp}e^{2i\beta}) \cdot \tag{18.14}$$

Thus

$$\frac{\partial |B|^2}{\partial \beta} = 4\mathrm{Re}(\bar{a}_{pq}a_{qp}ie^{2i\beta}) = -4\mathrm{Im}(\bar{a}_{pq}a_{qp}e^{2i\beta}). \tag{18.15}$$

Also

$$\mathrm{Im}(\bar{B}D) = \mathrm{Im}\left[(\bar{a}_{pq}e^{i\beta} + \bar{a}_{qp}e^{-i\beta})D\right] \tag{18.16}$$

$$\frac{\partial}{\partial\beta}[\mathrm{Im}(\bar{B}D)] = \mathrm{Im}[iD(\bar{a}_{pq}e^{i\beta} - \bar{a}_{qp}e^{-i\beta})]$$

$$= \mathrm{Re}[D(\bar{a}_{pq}e^{i\beta} - \bar{a}_{qp}e^{-i\beta})] \ . \qquad (18.17)$$

Hence

$$\frac{\partial\Delta}{\partial\beta} = (e^{i\beta}K+e^{-i\beta}\bar{K})\sinh 2y + 4\mathrm{Im}(\bar{a}_{pq}a_{qp}e^{2i\beta})\sinh^2 2y$$

$$+ 2\sinh 2y\cosh 2y\,\mathrm{Re}[D(\bar{a}_{pq}e^{i\beta} - \bar{a}_{qp}e^{-i\beta})] . \qquad (18.18)$$

We now consider c'_{pq}. From (18.11) we have

$$c'_{pq} = \sum_{i=1}^{n}(a'_{pi}\bar{a}'_{qi} - \bar{a}'_{ip}a'_{iq})$$

$$= \sum_{\substack{i=1\\ i\neq p,q}}^{n}(a'_{pi}\bar{a}'_{qi} - \bar{a}'_{ip}a'_{iq})$$

$$+ a'_{pp}\bar{a}'_{qp} - \bar{a}'_{qp}a'_{qq} + a'_{pq}\bar{a}'_{qq} - \bar{a}'_{pp}a'_{pq}$$

$$= \sum_{i\neq p,q}[(a_{pi}\cosh y - ia_{qi}e^{i\beta}\sinh y)$$

$$*(\bar{a}_{qi}\cosh y - i\bar{a}_{pi}e^{i\beta}\sinh y)$$

$$- (\bar{a}_{ip}\cosh y + i\bar{a}_{iq}e^{i\beta}\sinh y)$$

$$*(a_{iq}\cosh y + ia_{ip}e^{i\beta}\sinh y)]$$

$$+ \tfrac{1}{2}e^{i\beta}(-\bar{C}+\bar{B}\cosh 2y - i\bar{D}\sinh 2y)$$

$$*(D\cosh 2y - iB\sinh 2y)$$

$$+ \tfrac{1}{2}e^{i\beta}(C+B\cosh 2y + iD\sinh 2y)$$

$$*(-\bar{D}\cosh 2y - i\bar{B}\sinh 2y)$$

$$= \sum_{i\neq p,q}[a_{pi}\bar{a}_{qi}\cosh^2 y - a_{qi}\bar{a}_{pi}e^{2i\beta}\sinh^2 y$$

$$-\bar{a}_{ip}a_{iq}\cosh^2 y + \bar{a}_{iq}a_{ip}e^{2i\beta}\sinh^2 y]$$

$$-i\sum_{i\neq p,q}[a_{pi}\bar{a}_{pi}e^{i\beta}\sinh y\cosh y + a_{qi}\bar{a}_{qi}e^{i\beta}\sinh y\cosh y$$

$$+a_{ip}\bar{a}_{ip}e^{i\beta}\sinh y\cosh y+a_{iq}\bar{a}_{iq}e^{i\beta}\sinh y\cosh y]$$

$$+\tfrac{1}{2}e^{i\beta}[-\bar{C}D\cosh 2y+\bar{B}D\cosh^2 2y-iD\bar{D}\sinh 2y\cosh 2y$$

$$+i\bar{C}B\sinh 2y-iB\bar{B}\sinh 2y\cosh 2y-\bar{D}B\sinh^2 2y$$

$$-C\bar{D}\cosh 2y-B\bar{D}\cosh^2 2y-iD\bar{D}\sinh 2y\cosh 2y$$

$$-iC\bar{B}\sinh 2y-iB\bar{B}\sinh 2y\cosh 2y+D\bar{B}\sinh^2 2y]$$

$$= \sum_{i\neq p,q}[\cosh^2 y(a_{pi}\bar{a}_{qi}-\bar{a}_{ip}a_{iq})$$

$$+ e^{2i\beta}\sinh^2 y(a_{ip}\bar{a}_{iq}-\bar{a}_{pi}a_{qi})]$$

$$-ie^{i\beta}\sinh y\cosh y \sum_{i\neq p,q}(|a_{pi}|^2+|a_{ip}|^2+|a_{qi}|^2+|a_{iq}|^2)$$

$$+\tfrac{1}{2}e^{i\beta}[\text{"""}]$$

$$= f_1 + f_2 \quad \text{say.} \tag{18.19}$$

$$f_1 = (-\bar{K}\cosh^2 y+e^{2i\beta}\sinh^2 y.K) - \tfrac{1}{2}ie^{i\beta}\sinh 2y.G$$

$$= e^{i\beta}[-\bar{K}\tfrac{1}{2}e^{-i\beta}(\cosh 2y+1) + \tfrac{1}{2}e^{i\beta}K(\cosh 2y-1)]$$

$$- \tfrac{1}{2}ie^{i\beta}\sinh 2y.G$$

$$= \tfrac{1}{2}e^{i\beta}[\cosh 2y(e^{i\beta}K-e^{-i\beta}\bar{K}) - (e^{i\beta}K+e^{-i\beta}\bar{K})$$

$$- iG\sinh 2y]$$

$$= \tfrac{1}{2}e^{i\beta}[-iH\cosh 2y - (e^{i\beta}K+e^{-i\beta}\bar{K}) - iG\sinh 2y] \ .$$

(18.20)

$$f_2 = \tfrac{1}{2}e^{i\beta}[-\cosh 2y(\bar{C}D+C\bar{D}) + \cosh^2 2y(\bar{B}D-B\bar{D})$$

$$-2iD\bar{D}\sinh 2y\cosh 2y + i\sinh 2y(\bar{C}B-C\bar{B})$$

$$-2iB\bar{B}\sinh 2y\cosh 2y + \sinh^2 2y(D\bar{B}-\bar{D}B)]$$

$$= \tfrac{1}{2}e^{i\beta}\left[-2\cosh 2y.\mathrm{Re}(\overline{C}D) + 2\sinh 2y.\mathrm{Im}(C\overline{B})\right.$$
$$+ \tfrac{1}{2}(1+\cosh 4y)(\overline{B}D-B\overline{D}) - i\sinh 4y(D\overline{D}+\overline{B}B)$$
$$\left.+ \tfrac{1}{2}(\cosh 4y-1)(D\overline{B}-\overline{D}B)\right]$$

$$= \tfrac{1}{2}e^{i\beta}\left[-2\cosh 2y.\mathrm{Re}(\overline{C}D) + 2\sinh 2y.\mathrm{Im}(C\overline{B})\right.$$
$$\left.+ \cosh 4y(\overline{B}D-B\overline{D}) - i\sinh 4y(|D|^2+|B|^2)\right]$$

$$= \tfrac{1}{2}e^{i\beta}\left[-2\cosh 2y.\mathrm{Re}(\overline{C}D) + 2\sinh 2y.\mathrm{Im}(C\overline{B})\right.$$
$$\left.+ 2i\cosh 4y.\mathrm{Im}(\overline{B}D) - i\sinh 4y(|D|^2+|B|^2)\right] . \qquad (18.21)$$

However

$$\mathrm{Re}(\overline{C}D) = \mathrm{Re}(D(\overline{a}_{pq}e^{i\beta} - \overline{a}_{qp}e^{-i\beta}))$$

and

$$\mathrm{Im}(C\overline{B}) = \mathrm{Im}\left[(a_{pq}e^{-i\beta}-a_{qp}e^{i\beta})(\overline{a}_{pq}e^{i\beta}+\overline{a}_{qp}e^{-i\beta})\right]$$
$$= \mathrm{Im}(a_{pq}\overline{a}_{qp}e^{-2i\beta} - \overline{a}_{pq}a_{qp}e^{2i\beta})$$
$$= -2\mathrm{Im}(\overline{a}_{pq}a_{qp}e^{2i\beta}) . \qquad (18.23)$$

Hence we obtain

$$c'_{pq} = \tfrac{1}{2}e^{i\beta}\left[-iG\sinh 2y - iH\cosh 2y\right.$$
$$- i\sinh 4y(|B|^2+|D|^2) + 2i\cosh 4y\,\mathrm{Im}(\overline{B}D)$$
$$- (e^{i\beta}K + e^{-i\beta}\overline{K})$$
$$- 4\sinh 2y.\mathrm{Im}(\overline{a}_{pq}a_{qp}e^{2i\beta})$$
$$\left.- 2\cosh 2y.\mathrm{Re}\left[D(\overline{a}_{pq}e^{i\beta} - \overline{a}_{qp}e^{-i\beta})\right]\right] . \qquad (18.24)$$

Comparing equations (18.12), (18.18) and (18.24) we see that

$$c'_{pq} = \frac{e^{i\beta}}{2}\left[\frac{i}{2}.\frac{\partial\Delta}{\partial y} - \frac{1}{\sinh 2y}.\frac{\partial\Delta}{\partial\beta}\right] \qquad (18.25)$$

and the theorem is proved.

19. CONDITIONS FOR CONVERGENCE

Before the main theorem we prove some lemmata.

Lemma 19.1

$$H - 2\mathrm{Im}(\bar{B}D) = -2\mathrm{Im}(c_{pq}e^{-i\beta}) \qquad (19.1)$$

Proof: From (18.4) we have

$$H = i(e^{i\beta}K - e^{-i\beta}\bar{K})$$

$$= 2\mathrm{Im}(\bar{K}e^{-i\beta}) \ . \qquad (19.2)$$

From (18.6) and (18.8) we have

$$2\mathrm{Im}(\bar{B}D) = 2\mathrm{Im}\left[(\bar{a}_{pq}e^{i\beta}+\bar{a}_{qp}e^{-i\beta})(a_{pp}-a_{qq})\right]$$

$$= 2\mathrm{Im}(a_{pp}\bar{a}_{pq}e^{i\beta} - \bar{a}_{pq}a_{qq}e^{i\beta}$$

$$+ a_{pp}\bar{a}_{qp}e^{-i\beta} - \bar{a}_{qp}a_{qq}e^{-i\beta})$$

$$= 2\mathrm{Im}\left[(a_{pp}\bar{a}_{qp} - \bar{a}_{pp}a_{pq})e^{-i\beta}\right.$$

$$\left. + (a_{pq}\bar{a}_{qq} - \bar{a}_{qp}a_{qq})e^{-i\beta}\right] \ , \qquad (19.3)$$

using $\mathrm{Im}(ze^{i\theta}) = -\mathrm{Im}(\bar{z}e^{-i\theta})$. From (18.5) and (18.11) we note that

$$\bar{K} + (\bar{a}_{pp}a_{pq} - a_{pp}\bar{a}_{qp}) + (\bar{a}_{qp}a_{qq} - a_{pq}\bar{a}_{qq})$$

$$= -c_{pq} \ . \qquad (19.4)$$

Hence

$$H - 2\mathrm{Im}(\bar{B}D) = -2\mathrm{Im}(c_{pq}e^{-i\beta}) \ . \qquad (19.5)$$

Lemma 19.2

$$|H - 2\mathrm{Im}(\bar{B}D)| \leq G + |B|^2 + |D|^2 . \qquad (19.6)$$

Proof:

$$|H - 2\mathrm{Im}(\bar{B}D)| \leq |H| + 2|\mathrm{Im}(\bar{B}D)|. \qquad (19.7)$$

Now

$$2|\mathrm{Im}(\bar{B}D)| \leq 2|\bar{B}D|$$

$$\leq |B|^2 + |D|^2 . \qquad (19.8)$$

We therefore wish to show

$$|H| \leq G. \qquad (19.9)$$

From (18.4)

$$H = i(e^{i\beta}K - e^{-i\beta}\bar{K}) = -2\mathrm{Im}(e^{i\beta}K) . \qquad (19.10)$$

As

$$K = \sum_{i\neq p,q} (a_{ip}\bar{a}_{iq} - \bar{a}_{pi}a_{qi}) \qquad (19.11)$$

we obtain

$$|H| \leq 2\Big|\sum_{i\neq p,q} (a_{ip}\bar{a}_{iq} - \bar{a}_{pi}a_{qi})\Big| \qquad (19.12)$$

$$\leq 2 \sum_{i\neq p,q} |a_{ip}\bar{a}_{iq} - \bar{a}_{pi}a_{qi}|. \qquad (19.13)$$

Now, for any complex numbers a,b,c,d we have

$$2|a\bar{b} - \bar{c}d| \leq 2|a\bar{b}| + 2|\bar{c}d|$$

$$\leq |a|^2 + |b|^2 + |c|^2 + |d|^2. \qquad (19.14)$$

Hence

$$2 \sum_{i\neq p,q} |a_{ip}\bar{a}_{iq} - \bar{a}_{pi}a_{qi}|$$

$$\leq \sum_{i\neq p,q} [|a_{pi}|^2 + |a_{ip}|^2 + |a_{qi}|^2 + |a_{iq}|^2]$$

$$= G. \qquad (19.15)$$

Therefore

$$|H| \leqslant G \tag{19.16}$$

and we obtain

$$|H - 2\mathrm{Im}(\bar{B}D)| \leqslant G + |B|^2 + |D|^2 . \tag{19.17}$$

It follows that, certainly

$$\begin{aligned} |H - 2\mathrm{Im}(\bar{B}D)| &\leqslant G + |B|^2 + |D|^2 \\ &\leqslant G + 2(|B|^2+|D|^2). \end{aligned} \tag{19.18}$$

Lemma 19.3

$$\begin{aligned} G + 2(|B|^2+|D|^2) &\leqslant 4\|A\|_E^2 \\ &= 4\sum_i \sum_j |a_{ij}|^2. \end{aligned} \tag{19.19}$$

Now from the definition of G in (18.3) it remains to show

$$|B|^2+|D|^2 \leqslant 2(|a_{pp}|^2+|a_{qq}|^2+|a_{pq}|^2+|a_{qp}|^2). \tag{19.20}$$

Now

$$\begin{aligned} |B|^2 &= |a_{pq}|^2 + |a_{qp}|^2 + 2\mathrm{Re}(\bar{a}_{pq}a_{qp}e^{2i\beta}) \\ &\leqslant |a_{pq}|^2 + |a_{qp}|^2 + 2|\bar{a}_{pq}a_{qp}| \\ &\leqslant |a_{pq}|^2 + |a_{qp}|^2 + |a_{pq}|^2 + |a_{qp}|^2 \\ &= 2(|a_{pq}|^2 + |a_{qp}|^2) . \end{aligned} \tag{19.21}$$

Also

$$\begin{aligned} |D|^2 &= (a_{pp}-a_{qq})(\bar{a}_{pp}-\bar{a}_{qq}) \\ &= |a_{pp}|^2 + |a_{qq}|^2 - a_{pp}\bar{a}_{qq} - \bar{a}_{pp}a_{qq} \\ &\leqslant |a_{pp}|^2 + |a_{qq}|^2 - 2\mathrm{Re}(a_{pp}\bar{a}_{qq}) \end{aligned}$$

$$\leq |a_{pp}|^2 + |a_{qq}|^2 + 2|a_{pp}\bar{a}_{qq}|$$

$$\leq 2(|a_{pp}|^2 + |a_{qq}|^2) \ . \qquad (19.22)$$

From (19.21) and (19.22) we have

$$|B|^2 + |D|^2 \leq 2(|a_{pp}|^2+|a_{qq}|^2+|a_{pq}|^2+|a_{qp}|^2) \qquad (19.23)$$

and hence

$$G + 2(|B|^2 + |D|^2) \leq 4\|A\|_E^2 \qquad (19.24)$$

which proves the lemma.

Lemma 19.4 If

$$\tan\beta = -\frac{\mathrm{Re}(c_{pq})}{\mathrm{Im}(c_{pq})} \qquad (19.25)$$

then

$$|\mathrm{Im}(c_{pq}e^{-i\beta})|^2 - |c_{pq}|^2 = 0 \ . \qquad (19.26)$$

Proof: If

$$\mathrm{Im}(c_{pq})\sin\beta + \mathrm{Re}(c_{pq})\cos\beta = 0 \qquad (19.27)$$

then

$$\mathrm{Re}(c_{pq}e^{-i\beta}) = 0. \qquad (19.28)$$

This implies $c_{pq}e^{-i\beta}$ is purely imaginary. Therefore

$$c_{pq}e^{-i\beta} = \pm i|c_{pq}e^{-i\beta}|$$

$$= \pm i|c_{pq}| \ . \qquad (19.29)$$

Thus

$$|\mathrm{Im}(c_{pq}e^{-i\beta})|^2 = |c_{pq}|^2 \qquad (19.30)$$

and the result is proved.

We now prove

Theorem 19.1 Let Δ denote $\Delta(y)$ where y is defined by

$$-\tanh y = \frac{H-2\mathrm{Im}(\bar{B}D)}{2\left[G+2(|B|^2+|D|^2)\right]}$$

$$= \frac{-\mathrm{Im}(c_{pq}e^{-i\beta})}{G+2(|B|^2+|D|^2)} \tag{19.31}$$

then

$$\Delta \geq \frac{1}{3}\cdot\frac{|c_{pq}|^2}{\|A\|_E^2} . \tag{19.32}$$

Proof: Recalling (17.40)

$$\Delta = G(1-\cosh 2y) - H\sinh 2y + \tfrac{1}{2}(|B|^2+|D|^2)(1-\cosh 4y) + \mathrm{Im}(\bar{B}D)\sinh 4y . \tag{19.33}$$

Thus

$$\begin{aligned}\Delta &= -2G\sinh^2 y - 2H\sinh y\cosh y \\ &\quad + \tfrac{1}{2}(|B|^2+|D|^2)(-2\sinh^2 2y) \\ &\quad + 4\mathrm{Im}(\bar{B}D)\sinh y\cosh y\cosh 2y \\ &= -2G\sinh^2 y - 4\sinh^2 y\cosh^2 y(|B|^2+|D|^2) \\ &\quad -2\sinh y\cosh y\left[H-2\mathrm{Im}(\bar{B}D)\cosh 2y\right] \\ &= -2\sinh^2 y\left[G + 2\cosh^2 y(|B|^2+|D|^2)\right] \\ &\quad -2\sinh y\cosh y\left[H - 2\mathrm{Im}(\bar{B}D)(1+2\sinh^2 y)\right] \\ &= -2\sinh^2 y\left[G + 2(|B|^2+|D|^2) + 2\sinh^2 y(|B|^2+|D|^2)\right. \\ &\quad \left. -4\sinh y\cosh y\,\mathrm{Im}(\bar{B}D)\right] \\ &\quad -2\sinh y\cosh y\left[H - 2\mathrm{Im}(\bar{B}D)\right]\end{aligned}$$

$$= -2\sinh^2 y[G + 2(|B|^2+|D|^2) + 2\sinh^2 y(|B|^2+|D|^2)$$
$$-4\sinh y\cosh y\,\mathrm{Im}(\bar{B}D)]$$
$$+ 4\sinh y\cosh y\,\mathrm{Im}(c_{pq}e^{-i\beta}) \quad . \qquad (19.34)$$

Therefore

$$\tfrac{1}{2}\Delta = 2\cosh^2 y\tanh y.\mathrm{Im}(c_{pq}e^{i\beta})$$
$$- \cosh^2 y\tanh^2 y[G + 2(|B|^2+|D|^2)$$
$$+ 2\sinh^2 y(|B|^2+|D|^2)$$
$$- 4\sinh y\cosh y\,\mathrm{Im}(\bar{B}D)] \quad . \qquad (19.35)$$

From (19.1) and (19.31) we obtain

$$\tfrac{1}{2}\Delta = \frac{\cosh^2 y}{G+2(|B|^2+|D|^2)} \Big\{ 2|\mathrm{Im}(c_{pq}e^{i\beta})|^2$$
$$- \Big\{[\mathrm{Im}(c_{pq}e^{i\beta})]^2.[G+2(|B|^2+|D|^2)$$
$$+2\sinh^2 y(|B|^2+|D|^2) - 4\sinh y\cosh y\,\mathrm{Im}(\bar{B}D)]\Big\} /$$
$$/[G+2(|B|^2+|D|^2)]\Big\}$$

$$= \frac{\cosh^2 y|c_{pq}|^2}{G+2(|B|^2+|D|^2)} \Big\{ 2 - [G+2(|B|^2+|D|^2)$$
$$+2\sinh^2 y(|B|^2+|D|^2) - 4\sinh y\cosh y\,\mathrm{Im}(BD)]/$$
$$/[G+2(|B|^2+|D|^2)]\Big\}$$

(from lemma 19.4)

$$= \frac{\cosh^2 y|c_{pq}|^2}{G+2(|B|^2+|D|^2)} \left\{ [G+2(|B|^2+|D|^2) \right.$$

$$-2\sinh^2 y(|B|^2+|D|^2) + 4\sinh y\cosh y \mathrm{Im}(\bar{B}D)]/$$

$$\left. /[G+2(|B|^2+|D|^2)] \right\}$$

$$= \frac{\cosh^2 y|c_{pq}|^2}{G+2(|B|^2+|D|^2)} \left\{ 1 - \right.$$

$$\left. \frac{[2\sinh^2 y(|B|^2+|D|^2) - 4\sinh y\cosh y \mathrm{Im}(\bar{B}D)]}{G+2(|B|^2+|D|^2)} \right\} .$$

(19.36)

We now consider the expression inside the brackets

$$\left\{ 1 - \frac{[2\sinh^2 y(|B|^2+|D|^2) - 4\sinh y\cosh y \mathrm{Im}(\bar{B}D)]}{G+2(|B|^2+|D|^2)} \right\}$$

$$= 1 + \sinh^2 y \left[- \frac{2(|B|^2+|D|^2)}{G+2(|B|^2+|D|^2)} \right.$$

$$\left. + \frac{4(\cosh y/\sinh y)\mathrm{Im}(\bar{B}D)}{G+2(|B|^2+|D|^2)} \right]$$

$$\geq 1 + \sinh^2 y \left[-1 + \frac{4\mathrm{Im}(\bar{B}D).\coth y}{G+2(|B|^2+|D|^2)} \right]$$

$$= 1 + \sinh^2 y \left[-1 + \frac{4\mathrm{Im}(\bar{B}D)}{\frac{1}{2}(2\mathrm{Im}(\bar{B}D)-H)} \right]$$

$$= 1 + \sinh^2 y \left[\frac{-\frac{1}{2}(2\mathrm{Im}(\bar{B}D)-H) + 4\mathrm{Im}(\bar{B}D)}{\frac{1}{2}(2\mathrm{Im}(\bar{B}D)-H)} \right]$$

$$= 1 + 2\sinh^2 y \left[\frac{3\mathrm{Im}(\bar{B}D)+\frac{1}{2}H}{2\mathrm{Im}(\bar{B}D)-H}\right]$$

$$\geqslant 1 - 2\sinh^2 y \cdot \left|\frac{3\mathrm{Im}(\bar{B}D)+\frac{1}{2}H}{2\mathrm{Im}(\bar{B}D)-H}\right| \qquad (19.37)$$

Now

$$\sinh^2 y = 1/(\coth^2 y - 1) , \qquad (19.38)$$

therefore

$$\sinh^2 y = \frac{[H-2\mathrm{Im}(\bar{B}D)]^2}{4[G+2(|B|^2+|D|^2)]^2-[H-2\mathrm{Im}(\bar{B}D)]^2}$$

$$\leqslant \frac{[H-2\mathrm{Im}(\bar{B}D)]^2}{3[G+2(|B|^2+|D|^2)]^2} \cdot \qquad (19.39)$$

Therefore

$$\left\{1 - \frac{[2\sinh^2 y(|B|^2+|D|^2) - 4\sinh y\cosh y\mathrm{Im}(\bar{B}D)]}{G+2(|B|^2+|D|^2)}\right\}$$

$$\geqslant 1 - 2\,\frac{1}{3}\,\frac{[2\mathrm{Im}(\bar{B}D)-H]^2}{[G+2(|B|^2+|D|^2)]^2} \cdot \left|\frac{3\mathrm{Im}(\bar{B}D)+\frac{1}{2}H}{2\mathrm{Im}(\bar{B}D)-H}\right|$$

$$= 1 - \frac{2}{3}\left|\frac{[2\mathrm{Im}(\bar{B}D)-H]\,[3\mathrm{Im}(\bar{B}D)+\frac{1}{2}H]}{[G+2(|B|^2+|D|^2)]^2}\right|$$

$$\geqslant 1 - \frac{2[G+|B|^2+|D|^2]\,[\frac{1}{2}G+3(|B|^2+|D|^2)/2]}{3[G+2(|B|^2+|D|^2)]^2}$$

$$= 1 - \frac{1}{3} \cdot \left[\frac{G^2+4G(|B|^2+|D|^2)+3(|B|^2+|D|^2)^2}{[G+2(|B|^2+|D|^2)]^2}\right]$$

$$\geqslant 1 - 1/3$$

$$= 2/3 \ . \tag{19.40}$$

From lemma 19.3 and using

$$\cosh^2 y \geqslant 1 \tag{19.41}$$

we have from (19.36) and (19.40)

$$\tfrac{1}{2}\Delta \geqslant \frac{|c_{pq}|^2}{6\|A\|_E^2} \ . \tag{19.42}$$

Therefore

$$\Delta \geqslant \frac{|c_{pq}|^2}{3\|A\|_E^2} \tag{19.43}$$

which proves the theorem.

We have thus obtained a lower bound for Δ. Eberlein (1962) has proved sufficient conditions for

$$\lim_{i \to \infty} N^2(C_i) = 0. \tag{19.44}$$

It then follows from theorem 19.1 that as $N^2(A_i)$ is a decreasing monotone function bounded below by $\sum |\lambda_i|^2$

$$\|A_i A_i^H - A_i^H A_i\|_E^2 \longrightarrow 0 \tag{19.45}$$

or, equivalently, that

$$\|A_i\|_E^2 \longrightarrow \sum_{i=1}^{n} |\lambda_i|^2 \tag{19.46}$$

as i increases.

20. IMPLEMENTATION

We have closely followed the implementation due to Eberlein (1970) but one major change has been made. Eberlein orders the eigenvalues at the end of each sweep so that

$$|\mathrm{Re}(\lambda_i)|+|\mathrm{Im}(\lambda_i)| \geqslant |\mathrm{Re}(\lambda_{i+1})|+|\mathrm{Im}(\lambda_{i+1})|, \quad i=1,2,\ldots,n-1 . \qquad (20.1)$$

With this Ruhe (1968) has shown that the process is ultimately quadratically convergent. We carried out two further experiments. In the first the ordering was such that

$$|\lambda_1| \geqslant |\lambda_2| \geqslant \ldots \geqslant |\lambda_n| \qquad (20.2)$$

and in the second no ordering was used. Over a wide range of examples, including many with multiple eigenvalues, ordering was found to have a negligible effect. The number of rotations needed to almost diagonalise the matrix was nearly always the same and never differed by more than a few per cent but always the running time was less without ordering as the permuting of the rows and columns of the matrix was avoided. We are further encouraged to omit the ordering as in our particular application to simultaneous iteration the matrices upon which the generalised Jacobi method is applied are often diagonally dominant with the diagonal elements such that

$$|a_{11}| \geqslant |a_{22}| \geqslant \ldots \geqslant |a_{nn}|. \qquad (20.3)$$

We note that there is no question of the process not converging whether it is performed with or without ordering; it is simply that it has not been proved that ultimate quadratic convergence can be guaranteed to take place if there is no ordering. However, our experience suggests that, in general, ordering is not necessary and some time can be saved by omitting it. A listing of the program is given in appendix 4.

21. RUTISHAUSER'S ALGORITHM

Finally in this chapter we mention an algorithm due to Rutishauser (1964). The algorithm is delightfully simple but we show that convergence can never be proved.

Given an arbitrary matrix A Rutishauser forms the commutator matrix C

$$C = AA^H - A^HA. \tag{21.1}$$

The element c_{pq} is annihilated using Jacobi's method for normal matrices. The rotation matrices R needed for this are also used to form

$$A' = R^HAR . \tag{21.2}$$

We then choose the larger of c_{pp} and c_{qq}, say c_{pp}, and perform a shear on A' given by

$$A'' = S^{-1}A'S \tag{21.3}$$

where

$$s_{ij} = \delta_{ij} \tag{21.4}$$

save that

$$s_{pp} = x \ . \tag{21.5}$$

The elements of A" are then given by

$$\begin{aligned} a''_{ij} &= a_{ij} \qquad i,j \neq p \\ a''_{ip} &= x.a'_{ip} \qquad i \neq p \\ a''_{pi} &= a'_{pi}/x \qquad i \neq p \\ a''_{pp} &= a'_{pp} \ . \end{aligned} \tag{21.6}$$

Again we attempt to maximise Δ given by

$$\Delta = \sum_{i,j} |a_{ij}|^2 - \sum_{i,j} |a''_{ij}|^2 \tag{21.7}$$

$$= \sum_{i,j} |a'_{ij}|^2 - \sum_{i,j} |a''_{ij}|^2 \tag{21.8}$$

$$= (1-x^2) \sum_{i \neq p} |a_{ip}|^2 + (1-1/x^2) \sum_{i \neq p} |a_{pi}|^2 . \tag{21.9}$$

From (21.9)

$$\frac{d\Delta}{dx^2} = -\sum_{i \neq p} |a_{ip}|^2 + x^{-4} \sum_{i \neq p} |a_{pi}|^2 \ . \tag{21.10}$$

If

$$\frac{d\Delta}{dx^2} = 0 \tag{21.11}$$

then

$$x^4 = \sum_{i \neq p} |a_{pi}|^2 \Big/ \sum_{i \neq p} |a_{ip}|^2 \ . \tag{21.12}$$

We then form the new commutator matrix and continue annihilating its elements in a serial fashion.

In attempting to prove convergence we looked

at two possible approaches. Firstly after each rotation-shear pair and secondly after a complete sweep. We have counter examples to show that in both cases A is not nearer to diagonal form than before. The overall process is often convergent but is extremely slow and does not appear to be a contender with the Eberlein type algorithm.

CHAPTER 3

ON THE ORTHONORMALISATION AND BIORTHONORMALISATION OF SETS OF VECTORS

1. INTRODUCTION

In this chapter we shall consider firstly the problem of orthogonalising a set of p linearly independent real vectors. The orthogonalised vectors will then be normalised to give an orthonormal set. Thus, if we let

$$X = [x_1, x_2, \ldots, x_p] \tag{1.1}$$

where the x_i are vectors of n components we shall obtain

$$X^T X = I_p \tag{1.2}$$

or, equivalently,

$$(x_i, x_j) = \delta_{ij} . \tag{1.3}$$

This is solved by the Gram-Schmidt process and is well documented by, for example, Rice (1966) and Björck (1967). We then extend the process to permit the x_i to have complex elements and orthonormalise such that, instead of equation (1.2), we have

$$X^H X = I_p . \tag{1.4}$$

Finally we turn to the problem of biorthonormalisation. That is, given two sets of vectors (which may be complex),

$$X = [x_1, x_2, \ldots, x_p]$$

and

$$Y = [y_1, y_2, \ldots, y_p] \tag{1.5}$$

we wish to biorthonormalise such that

$$Y^H X = I_p \tag{1.6}$$

or, equivalently,

$$(y_i, x_j) = \delta_{ij} \ . \qquad (1.7)$$

We note that the Gram-Schmidt process is a particular case of biorthonormalisation given by setting

$$y_i = x_i \ , \qquad i = 1,2,\ldots,p. \qquad (1.8)$$

The problem of biorthonormalisation does not appear to have been studied although a passing reference to it is made in Clint and Jennings (1971).

Having discussed the theoretical aspects of these problems we look at the implementation of them on a computer. It is well known, for example Rice (1966), that the classical Gram-Schmidt process is not suitable for automatic computation but that a slight variant of it, the modified Gram-Schmidt process, is well constructed for an electronic computer. Analogously we introduce a modified biorthonormalisation process. For our application to simultaneous iteration we require great accuracy on two points; firstly that the vectors are orthogonal, that is, for a specified ε,

$$(x_i, x_j) < \varepsilon \qquad i \neq j \qquad (1.9)$$

and secondly the invariance of the original subspace defined by

$$X = [x_1, x_2, \ldots, x_p] \ . \qquad (1.10)$$

In order to ensure the greatest possible accuracy we introduce reinforcement into our algorithm. This is a device to reorthogonalise any vector

which may be thought to be not strictly orthogonal to its predecessors.

2. GRAM-SCHMIDT ORTHONORMALISATION

Suppose we have a set of p linearly independent real vectors, each of n (non-infinite) components, $[f_1, f_2, \ldots, f_p]$. From these we wish to construct an orthonormal set $[\phi_1, \phi_2, \ldots, \phi_p]$ such that

$$(\phi_i, \phi_j) \equiv (\phi_i)^T(\phi_j) = \delta_{ij} \ . \qquad (2.1)$$

From the set $\{f_i \mid i=1,\ldots,p\}$ we first of all construct an orthogonal set $\{\psi_i \mid i=1,\ldots,p\}$ where

$$\begin{aligned} (\psi_i, \psi_j) &= 0 \quad & i \neq j \\ &\neq 0 \quad & i = j \end{aligned} \qquad (2.2)$$

and

$$\begin{aligned} \psi_1 &= f_1 \\ \psi_2 &= a_{21} f_1 + f_2 \\ &\ldots\ldots\ldots \\ \psi_k &= \sum_{i=1}^{k-1} a_{ki} f_i + f_k \qquad k = 2,3,\ldots,p \ , \end{aligned} \qquad (2.3)$$

where the a_{ij} are constants appropriately determined. Consider now the reciprocal system

$$\begin{aligned} f_1 &= \psi_1 \\ f_2 &= c_{21} \psi_1 + \psi_2 \\ &\ldots\ldots\ldots \\ f_k &= \sum_{i=1}^{k-1} c_{ki} \psi_i + \psi_k \qquad k = 2,3,\ldots,p \ . \end{aligned} \qquad (2.4)$$

We require the ψ_i to be orthogonal thus, on taking inner products with ψ_i, we obtain

$$(\psi_i, f_k) = \sum_{j=1}^{k-1} c_{kj}(\psi_i, \psi_j) + (\psi_i, \psi_k)$$
$$= c_{ki}(\psi_i, \psi_i) \quad . \tag{2.5}$$

Equation (2.5) immediately gives

$$c_{ki} = \frac{(\psi_i, f_k)}{(\psi_i, \psi_i)} \quad . \tag{2.6}$$

From equation (2.4) we obtain

$$\psi_k = f_k - \sum_{i=1}^{k-1} c_{ki}\psi_i$$
$$= f_k - \sum_{i=1}^{k-1} \frac{(\psi_i, f_k)}{(\psi_i, \psi_i)} \cdot \psi_i \qquad k=2,\ldots,p \quad . \tag{2.7}$$

The set $\{\psi_k | k=1,\ldots,p\}$ is now orthogonal. If we further demand that it be orthonormal, that is, that

$$(\psi_i, \psi_i) = 1 \tag{2.8}$$

we form

$$\phi_k = \psi_k / \|\psi_k\|$$
$$= \frac{\psi_k}{(\psi_k, \psi_k)^{\frac{1}{2}}} \tag{2.9}$$

and the set $\{\phi_k | k=1,\ldots,p\}$ is now an orthonormal set. (Throughout this chapter $\|.\|$ denotes the 2-norm.) We may modify equation (2.7) to give

$$\psi_k = f_k - \sum_{i=1}^{k-1} (\phi_i, f_k)\phi_i \qquad k=2,\ldots,p$$

$$\phi_k = \psi_k / \|\psi_k\| \qquad k=1,2,\ldots,p \; . \tag{2.10}$$

The above, the classical Gram-Schmidt process, holds for real vectors in which case we note that

$$(f,g) = (g,f) \tag{2.11}$$

and, for vectors of finite dimension, we take

$$(f,g) = f^T g \; . \tag{2.12}$$

With only slight modification the process holds for complex vectors. However, in this case

$$(f,g) = \overline{(g,f)} \tag{2.13}$$

where $\overline{z}$ denotes the complex conjugate of z. Also, analogously to (2.12) we take

$$(f,g) = f^H g \; . \tag{2.14}$$

We may thus summarise the classical Gram-Schmidt process as

$$\begin{aligned} \phi_1 &= f_1 / \|f_1\| \\ \phi_k' &= f_k - \sum_{i=1}^{k-1} (\phi_i, f_k)\phi_i \qquad k=2,\ldots,p \; . \\ \phi_k &= \phi_k' / \|\phi_k'\| \end{aligned} \tag{2.15}$$

3. MODIFIED GRAM-SCHMIDT

It is well known, see for example Rice (1966), that the classical Gram-Schmidt process is often computationally disastrous. This has led to the introduction of the modified Gram-Schmidt process which we explain below.

In the classical Gram-Schmidt process we

remove, at the k-th stage, all components of the now orthonormal $f_1, f_2, \ldots, f_{k-1}$ from f_k. In the modified process we remove the relevant component of the k-th vector from each of the remaining $(p-k)$ vectors. We may summarise the process as follows

$$\left.\begin{aligned} \phi_1 &= f_1/\|f_1\| \\ f_k^{(i)} &= f_k^{(i-1)} - (\phi_i, f_k^{(i-1)})\phi_i \\ & \qquad i = 1,2,\ldots,k-1 \\ \phi_k &= f_k^{(k-1)}/\|f_k^{(k-1)}\| \end{aligned}\right\} \quad k = 2,\ldots,p \tag{3.1}$$

where we define $f_k^{(0)} = f_k$.

By comparing equations (2.15) with (3.1) the similarity of the two processes will be seen. It is also apparent that the modified process is easier to implement than the classical one. We now prove the following theorem.

Theorem 3.1 The classical and modified Gram-Schmidt processes are theoretically identical.

Proof: We rewrite the second equation of (2.15) as

$$f_k^{(k-1)} = f_k^{(0)} - (\phi_1, f_k^{(0)})\phi_1 - (\phi_2, f_k^{(0)})\phi_2 - \\ - \ldots - (\phi_{k-1}, f_k^{(0)})\phi_{k-1} \tag{3.2}$$

and rewrite (3.1) as

$$f_k^{(k-1)} = f_k^{(0)} - (\phi_1, f_k^{(0)})\phi_1 - (\phi_2, f_k^{(1)})\phi_2 - \\ - \ldots - (\phi_{k-1}, f_k^{(k-2)})\phi_{k-1} . \tag{3.3}$$

We see that for p=1 or 2 the computations involved in the two processes are identical but for $p \geq 3$ they are performed in a different order.

To prove that the processes are theoretically the same we need to show that

$$(\phi_i, f_k^{(0)})\phi_i = (\phi_i, f_k^{(i-1)})\phi_i \qquad 1 \leq i \leq k-1 \ . \tag{3.4}$$

Consider

$$(\phi_i, f_k^{(i-1)})\phi_i$$

$$= \left\{ \phi_i, \left[f_k^{(i-2)} - (\phi_{i-1}, f_k^{(i-2)})\phi_{i-1} \right] \right\} \phi_i$$

(by (3.1))

$$= (\phi_i, f_k^{(i-2)})\phi_i \ , \tag{3.5}$$

as $(\phi_i, \phi_{i-1}) = 0$. Similarly we may show

$$(\phi_i, f_k^{(i-2)})\phi_i = (\phi_i, f_k^{(i-3)})\phi_i \ . \tag{3.6}$$

Repeated application gives

$$(\phi_i, f_k^{(i-1)})\phi_i = (\phi_i, f_k^{(0)})\phi_i \ . \tag{3.7}$$

Since equation (3.7) holds for all

$$1 \leq i \leq k-1 \tag{3.8}$$

it follows that equations (3.2) and (3.3) are identical and the theorem is proved.

Rice (1966) has given a simple error analysis to show why the classical Gram-Schmidt process may be computationally poor and this we outline. Suppose that for some value k we have

$$(\phi_i, \phi_j) = \varepsilon_{ij} \qquad i,j \leq k-1 \ . \tag{3.9}$$

Let

$$f_k = \sum_{i=1}^{k-1} \beta_i \phi_i + \eta \qquad (3.10)$$

where

$$(\eta, \phi_i) = 0 \quad \text{for} \quad 1 \leq i \leq k-1 \ . \qquad (3.11)$$

Then

$$\begin{aligned} \phi_k' &= f_k - \sum_{j=1}^{k-1} (\phi_j, f_k)\phi_j \\ &= \sum_{i=1}^{k-1} \beta_i \phi_i + \eta - \sum_{j=1}^{k-1} \Big[\phi_j, \Big(\sum_{i=1}^{k-1} \beta_i \phi_i + \eta\Big)\Big]\phi_j \\ &= \eta - \sum_{j=1}^{k-1} \Big(\sum_{i \neq j} \beta_i \varepsilon_{ij}\Big)\phi_j \ . \end{aligned} \qquad (3.12)$$

From equations (3.9) and (3.12)

$$\begin{aligned} \varepsilon_{kl} &= (\phi_k, \phi_l) \\ &= (\phi_k'/\|\phi_k'\|, \phi_l) \\ &= \frac{-\sum_{j=1}^{k-1} \big(\sum_{i \neq j} \beta_i \varepsilon_{ij}\big)\varepsilon_{jl}}{\|\phi_k'\|} \ . \end{aligned} \qquad (3.13)$$

Hence non-orthogonality effects are magnified by the factor $1/\|\phi_k'\|$ in the classical Gram-Schmidt process. If, as may quite often happen, $\|\eta\|$ becomes small then $\|\phi_k'\|$ becomes small. This increases the ε_{ij} which makes ϕ_k' tend to become a linear combination of the $\{\phi_i \,|\, i=1,\ldots,k-1\}$. Thus $\varepsilon_{k,k-1}$ approaches unity and the vectors, instead of being orthogonal, become parallel!

However, the modified Gram-Schmidt process <u>always</u> has $\varepsilon_{k,k-1}=0$ (to machine accuracy). For we have

$$(\phi_i, f_k^{(i)}) = (\phi_i, f_k^{(i-1)}) - (\phi_i, f_k^{(i-1)})(\phi_i, \phi_i)$$
$$= 0 . \qquad (3.14)$$

Rice (1966) has carried out a large number of experiments and all his results support the theoretical findings given above. It is also apparent from the above analysis that if the classical process once loses orthogonality it then always produces almost identical vectors. However the modified process always generates distinct vectors (even if they are not orthogonal) as we know that ϕ_k is always orthogonal (to machine accuracy) to ϕ_{k-1}.

In addition to the simple analysis given above Björck (1967) has given a complete rounding error analysis of the Gram-Schmidt process. In order to illustrate the difference between the classical and modified processes he gives the following example. Suppose

$$[f_1, f_2, f_3] = \begin{bmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix} \qquad (3.15)$$

where ε is a small number such that due to the machine round-off error $1+\varepsilon^2$ is everywhere put equal to 1. It is easily verified that, if no other rounding errors are made, then the maximum deviation from orthogonality of the

computed $\{\phi_i\}$ is given by

$$\frac{|(\phi_3,\phi_2)|}{\|\phi_3\|_2\|\phi_2\|_2} = \tfrac{1}{2} \qquad (3.16)$$

for the classical process and

$$\frac{|(\phi_3,\phi_1)|}{\|\phi_3\|_2\|\phi_1\|_2} = \sqrt{\frac{2}{3}}\,\tfrac{1}{2}\varepsilon \qquad (3.17)$$

for the modified process.

4. REINFORCEMENT

By reinforcement we mean reorthogonalising one of the (supposedly orthogonal) ϕ_k against $\{\phi_i \mid i=1,\ldots,k-1\}$. In general there should be no need to reinforce but if a particular f_k is almost parallel to one of the preceding ϕ_i or if $f_k^{(k-1)}$ is a vector all of whose components are small it may well be advisable to take the precaution of reorthogonalising. We have no proof of the validity of the strategy we describe, which was first suggested by Rutishauser (1970), but over a large number of experiments it has been seen to be satisfactory.

At the k-th stage of the modified Gram-Schmidt process we form successively the inner-products

$$(\phi_i, f_k^{(i-1)}) \qquad i=1,\ldots,k-1 \qquad (4.1)$$

and finally

$$\|f_k^{(k-1)}\| = (f_k^{(k-1)}, f_k^{(k-1)})^{\frac{1}{2}} . \qquad (4.2)$$

If any of the products (4.1) were large this would be an indication that ϕ_i and $f_k^{(i-1)}$ were nearly parallel and that a large component of f_k existed in the ϕ_i direction. The existence and removal of such a large component might well suggest the desirability of reorthogonalising. We might also expect $(f_k^{(k-1)}, f_k^{(k-1)})$ not to be pathologically small. If this were the case the resulting error in ϕ_k defined by (3.1) might be magnified. With these points in mind we form at the k-th step the total t given by

$$t = \sum_{i=1}^{k-1} (\phi_i, f_k^{(i-1)})^2 \qquad (4.3)$$

and let

$$s = (f_k^{(k-1)}, f_k^{(k-1)}) . \qquad (4.4)$$

The possibly undesirable situations we have outlined above correspond to either

$$t \gg s \qquad (4.5)$$

or

$$s.\varepsilon = 0 \qquad (4.6)$$

where ε is the machine precision. As a typical value for ε is $O(10^{-12})$ if equation (4.6) is true it is clear that every component of $f_k^{(k-1)}$ must be very small. In this case we assume f_k to be linearly dependent upon $\{\phi_i \mid i=1,\ldots,k-1\}$ and set

$$\phi_k \equiv 0 \ . \qquad (4.7)$$

If however relation (4.5) is true but (4.6) is not we reorthogonalise $f_k^{(k-1)}$. It is a relatively arbitrary choice but we replace (4.5) by the test

$$\text{Is } t \geqslant 100.s \ ? \qquad (4.8)$$

If so, we reinforce f_k; otherwise we accept ϕ_k and proceed to f_{k+1}. The factor of 100 appears to err on the side of caution but does not cause many unnecessary reinforcements to be performed. As a high degree of orthogonality is required in our application this is the factor we have used throughout.

5. COMPUTATIONAL FORMULATION

The modified Gram-Schmidt process is delightfully easy to program and follows exactly the theoretical formulation given in equations (3.1). The tests for reinforcement are as given in section 4. Two versions of the program are needed; one is for the case of real vectors, the other for the complex case. The latter is similar to the real case with all the complex arithmetic performed using only real variables. Both routines use a separate subprogram to calculate the inner-products and, for the complex case, three additional subprograms are required. Full details of the programming are to be found in chapter 5 and

listings of the orthonormalisation routines are included in appendix 5 (real case) and appendix 6 (complex case).

6. BIORTHONORMALISATION

We have just discussed the problem of orthonormalisation and we now turn to that of biorthonormalisation. Thus given two sets of p vectors with n components ($p \leq n$), $\{f_i | i=1,....,p\}$ and $\{g_i | i=1,....,p\}$ it is possible, subject to certain conditions (in this case rather more stringent than those of the Gram-Schmidt process), to biorthogonalise the vectors to give two new sets $\{\psi_i' | i=1,....,p\}$ and $\{\phi_i' | i=1,....,p\}$ such that

$$\begin{aligned} (\psi_i', \phi_j') &= 0 \qquad i \neq j \\ &\neq 0 \qquad i = j \, . \end{aligned} \tag{6.1}$$

We may further demand biorthonormalisation of the sets $\{\psi_i\}$ and $\{\phi_i\}$;

$$\begin{aligned} (\psi_i, \phi_j) &= 0 \qquad i \neq j \\ &= 1 \qquad i = j \, . \end{aligned} \tag{6.2}$$

We assume from the outset that $\{f_i\}$ and $\{g_i\}$ may be complex and express ψ_i and ϕ_i in terms of our original sets $\{f\}$ and $\{g\}$. Thus

$$\psi_1 = f_1$$

$$\psi_2 = a_{21} f_1 + f_2$$

. . . . . .

$$\psi_k = \sum_{i=1}^{k-1} a_{ki} f_i + f_k \qquad k=2,\ldots,p \qquad (6.3)$$

and

$$\phi_1 = g_1$$

$$\phi_2 = b_{21} g_1 + g_2$$

$$\cdots\cdots\cdots$$

$$\phi_k = \sum_{i=1}^{k-1} b_{ki} g_i + g_k \qquad k=2,\ldots,p \qquad (6.4)$$

where the a_{ij} and b_{ij} are appropriately determined.

Consider now the reciprocal systems

$$f_1 = \psi_1$$

$$f_2 = c_{21}\psi_1 + \psi_2$$

$$\cdots\cdots\cdots$$

$$f_k = \sum_{i=1}^{k-1} c_{ki}\psi_i + \psi_k \qquad k=2,\ldots,p \qquad (6.5)$$

and

$$g_1 = \phi_1$$

$$g_2 = d_{21}\phi_1 + \phi_2$$

$$\cdots\cdots\cdots$$

$$g_k = \sum_{i=1}^{k-1} d_{ki}\phi_i + \phi_k \qquad k=2,\ldots,p\ . \qquad (6.6)$$

We require the ψ_i and ϕ_i to be biorthogonal thus, on taking inner-products of (6.5) with ϕ_i and of (6.6) with ψ_i, we obtain

$$(\phi_i, f_k) = c_{ki}(\phi_i, \psi_i) \qquad (6.7)$$

and

$$(\psi_i, g_k) = d_{ki}(\psi_i, \phi_i)\ . \qquad (6.8)$$

These equations immediately give

$$c_{ki} = \frac{(\phi_i, f_k)}{(\phi_i, \psi_i)} \tag{6.9}$$

and

$$d_{ki} = \frac{(\psi_i, g_k)}{(\psi_i, \phi_i)} \tag{6.10}$$

where we define the inner-products exactly as before in section 2. From equations (6.5) and (6.6) we obtain

$$\begin{aligned} \psi_k &= f_k - \sum_{i=1}^{k-1} c_{ki}\psi_i \\ &= f_k - \sum_{i=1}^{k-1} \frac{(\phi_i, f_k)}{(\phi_i, \psi_i)} \cdot \psi_i \end{aligned} \tag{6.11}$$

and

$$\begin{aligned} \phi_k &= g_k - \sum_{i=1}^{k-1} d_{ki}\phi_i \\ &= g_k - \sum_{i=1}^{k-1} \frac{(\psi_i, g_k)}{(\psi_i, \phi_i)} \cdot \phi_i \quad . \end{aligned} \tag{6.12}$$

The sets $\{\psi_i | i=1,\ldots,p\}$ and $\{\phi_i | i=1,\ldots,p\}$ are now biorthogonal. If we further demand that they be biorthonormal, that is, that

$$(\psi_i, \phi_i) = 1 = (\phi_i, \psi_i) \tag{6.13}$$

we may modify equations (6.11) and (6.12) to give

$$\psi'_k = f_k - \sum_{i=1}^{k-1} (\phi_i, f_k)\psi_i \tag{6.14}$$

and

$$\phi'_k = g_k - \sum_{i=1}^{k-1} (\psi_i, g_k)\phi_i \quad . \tag{6.15}$$

The question of how to calculate the normalising factor is much more complicated than in the Gram-Schmidt process and we leave a full discussion to section 8. However the obvious approach, by analogy with the previous sections, would be to form

$$(\phi_k,\psi_k) = \overline{(\psi_k,\phi_k)} \tag{6.16}$$

and then to take

$$\psi_k' = \frac{\psi_k}{(\phi_k,\psi_k)^{\frac{1}{2}}} \tag{6.17}$$

$$\phi_k' = \frac{\phi_k}{(\phi_k,\psi_k)^{\frac{1}{2}}} \quad . \tag{6.18}$$

Whilst this is theoretically acceptable in practice it may not be satisfactory as we see in section 8.

7. MODIFIED BIORTHONORMALISATION

We summarise the biorthonormalisation process as:

Biorthonormalise f_1 and g_1 to give ψ_1 and ϕ_1 such that $(\psi_1,\phi_1)=1$,

$$\left.\begin{aligned} \psi_k' &= f_k - \sum_{i=1}^{k-1} (\phi_i,f_k)\psi_i \\ \phi_k' &= g_k - \sum_{i=1}^{k-1} (\psi_i,g_k)\phi_i \\ &\text{biorthonormalise } \psi_k' \text{ and } \phi_k' \text{ to give} \\ &\psi_k \text{ and } \phi_k \text{ such that } (\psi_k,\phi_k)=1 \end{aligned}\right\} \; k=2,\ldots,p \tag{7.1}$$

Following section 3 it seems natural to introduce

the modified biorthonormalisation process in which, at the k-th stage, we remove the relevant components of the k-th vector from each of the remaining (p-k) vectors. This contrasts with equations (7.1) in which we remove all components of the first (k-1) biorthonormal vectors from the k-th one. We may summarise the modified process as follows:

Biorthonormalise f_1 and g_1 to give ψ_1 and ϕ_1 such that $(\psi_1,\phi_1)=1$,

$$\left.\begin{aligned} f_k^{(i)} &= f_k^{(i-1)} - (\phi_i, f_k^{(i-1)})\psi_i \\ g_k^{(i)} &= g_k^{(i-1)} - (\psi_i, g_k^{(i-1)})\phi_i \end{aligned}\right\} \quad i=1,\ldots,k-1$$

biorthonormalise $f_k^{(k-1)}$ and $g_k^{(k-1)}$ to give ψ_k and ϕ_k such that $(\psi_k,\phi_k)=1$

$$k = 2,\ldots,p \qquad (7.2)$$

where we define $f_k^{(0)}=f_k$ and $g_k^{(0)}=g_k$.

Comparison between equations (7.1) and (7.2) shows the similarity of the processes and again reveals that the modified one is easier to implement. We now prove the following theorem.

Theorem 7.1 The classical and modified biorthonormalisation processes are theoretically identical.

Proof: We consider firstly equations (7.1) rewritten as

$$f_k^{(k-1)} = f_k^{(0)} - (\phi_1, f_k^{(0)})\psi_1 - (\phi_2, f_k^{(0)})\psi_2 - \\ - \cdots - (\phi_{k-1}, f_k^{(0)})\psi_{k-1} \qquad (7.3)$$

$$g_k^{(k-1)} = g_k^{(0)} - (\psi_1, g_k^{(0)})\phi_1 - (\psi_2, g_k^{(0)})\phi_2 - \\ - \cdots - (\psi_{k-1}, g_k^{(0)})\phi_{k-1} \qquad (7.4)$$

and equations (7.2) we write as

$$f_k^{(k-1)} = f_k^{(0)} - (\phi_1, f_k^{(0)})\psi_1 - (\phi_2, f_k^{(1)})\psi_2 - \\ - \cdots - (\phi_{k-1}, f_k^{(k-2)})\psi_{k-1} \qquad (7.5)$$

$$g_k^{(k-1)} = g_k^{(0)} - (\psi_1, g_k^{(0)})\phi_1 - (\psi_2, g_k^{(1)})\phi_2 - \\ - \cdots - (\psi_{k-1}, g_k^{(k-2)})\phi_{k-1} \, . \qquad (7.6)$$

Again we can see that for p=1 or 2 the computations involved in the processes are identical but for $p \geq 3$ they are performed in a different order.

To prove the two schemes are identical we need to show that

$$\left.\begin{aligned} (\phi_i, f_k^{(0)})\psi_i &= (\phi_i, f_k^{(i-1)})\psi_i \qquad (7.7) \\ \text{and} \\ (\psi_i, g_k^{(0)})\phi_i &= (\psi_i, g_k^{(i-1)})\phi_i \qquad (7.8) \end{aligned}\right\} \; 1 \leq i \leq k-1 \, .$$

Consider

$$(\phi_i, f_k^{(i-1)})\psi_i$$

$$= \{\phi_i, [f_k^{(i-2)} - (\phi_{i-1}, f_k^{(i-2)})\psi_{i-1}]\}\psi_i$$

(by (7.2))

$$= (\phi_i, f_k^{(i-2)})\psi_i \, , \qquad (7.9)$$

as $(\phi_i,\psi_{i-1})=0$. Similarly we may show

$$(\phi_i,f_k^{(i-2)})\psi_i = (\phi_i,f_k^{(i-3)})\psi_i \qquad (7.10)$$

and, by repeated application of (7.2), that

$$(\phi_i,f_k^{(i-1)})\psi_i = (\phi_i,f_k^{(0)})\psi_i \ . \qquad (7.11)$$

A similar argument shows that

$$(\psi_i,g_k^{(i-1)})\phi_i = (\psi_i,g_k^{(0)})\phi_i \ . \qquad (7.12)$$

Since equations (7.11) and (7.12) hold for all

$$1 \leqslant i \leqslant k-1 \qquad (7.13)$$

it follows that the theorem is proved.

Identical analysis to that used in section 3 shows the inherent instability of the classical biorthonormalisation process and for the modified biorthonormalisation process shows a stability similar to that for the modified Gram-Schmidt process. We have carried out extensive tests of both the classical and modified biorthonormalisation programs and the experimental evidence strongly supports the theoretical analysis. Indeed there would seem to be no reason whatever to implement either the classical Gram-Schmidt or the classical biorthonormalisation as, in both cases, the modified schemes are easier to program and give better results.

We included reinforcement in our biorthonormalisation scheme utilising the technique of section 4 and applying it to both sets of vectors.

Thus we form

$$t_L = \sum_{i=1}^{k-1} |(\phi_i, f_k^{(i-1)})|^2$$

and

$$t_R = \sum_{i=1}^{k-1} |(\psi_i, g_k^{(i-1)})|^2 \qquad (7.14)$$

in place of equation (4.3). We replace equation (4.4) by

$$s = |(g_k^{(k-1)}, f_k^{(k-1)})| \ . \qquad (7.15)$$

We now perform the tests given in section 4 independently to both t_L and t_R and reinforce if either result suggests that it is necessary.

8. NORMALISATION

There is one very important difference between orthogonalisation and biorthogonalisation. In the Gram-Schmidt process the inner-product (ϕ_k, ϕ_k) is always real and positive irrespective of whether the vectors are real or complex. Hence

$$\|\phi_k\| = (\phi_k, \phi_k)^{\frac{1}{2}} \qquad (8.1)$$

is always real. However, in the case of biortho-normalisation, there is no reason why, for real vectors, (ϕ_k, ψ_k) should be positive and hence $(\phi_k, \psi_k)^{\frac{1}{2}}$ may be complex. If it is desired to remain wholly within the real plane then the sign of each of the components of one of the vectors ϕ_k or ψ_k should be changed.

This leads us to consider how the normalisation should be performed. We saw at the end of section 6 that theoretically we may take, for an unnormalised ψ_k and ϕ_k,

$$\psi_k' = \frac{\psi_k}{(\phi_k, \psi_k)^{\frac{1}{2}}} \tag{8.2}$$

$$\phi_k' = \frac{\phi_k}{(\phi_k, \psi_k)^{\frac{1}{2}}} \quad . \tag{8.3}$$

Unfortunately there is no reason why this should give us

$$\|\psi_k'\| = O(\|\phi_k'\|) \tag{8.4}$$

and in practice the two norms are often wildly different. We therefore propose the following. We wish to determine

$$\psi_k' = \psi_k/\lambda \qquad \text{and} \qquad \phi_k' = \phi_k/\mu \quad . \tag{8.5}$$

such that

$$(\psi_k', \phi_k') = \frac{(\psi_k, \phi_k)}{\overline{\lambda}\,\mu} = 1 \tag{8.6}$$

together with

$$\|\psi_k'\| = \|\phi_k'\| \quad . \tag{8.7}$$

Equations (8.5) give

$$\|\psi_k'\| = \frac{\|\psi_k\|}{|\lambda|} \qquad \text{and} \qquad \|\phi_k'\| = \frac{\|\phi_k\|}{|\mu|} \quad . \tag{8.8}$$

Thus, dropping the subscript k,

$$\frac{\|\psi\|}{|\lambda|} = \frac{\|\phi\|}{|\mu|} \tag{8.9}$$

giving

$$\frac{|\lambda|}{|\mu|} = \frac{\|\psi\|}{\|\phi\|} \quad . \tag{8.10}$$

Equation (8.6) gives

$$\overline{\lambda}\mu = (\psi,\phi) \tag{8.11}$$

thus (8.10) yields

$$|\lambda|^2 = \frac{\|\psi\|}{\|\phi\|} \cdot |(\psi,\phi)| \quad . \tag{8.12}$$

Similarly we may obtain

$$|\mu|^2 = \frac{\|\phi\|}{\|\psi\|} \cdot |(\psi,\phi)| \quad . \tag{8.13}$$

~~Hence~~ (8.11), (8.12), (8.13) are satisfied by

$$\overline{\lambda} = \|\psi\| \left[\frac{(\psi,\phi)}{\|\phi\| \cdot \|\psi\|} \right]^{\frac{1}{2}} \tag{8.14}$$

and

$$\mu = \|\phi\| \left[\frac{(\psi,\phi)}{\|\phi\| \cdot \|\psi\|} \right]^{\frac{1}{2}} \quad . \tag{8.15}$$

Hence equations (8.5) and (8.15) give us a method for normalising such that

$$\|\psi'_k\| = \|\phi'_k\| \quad . \tag{8.16}$$

We have implemented the above in our program and found it most satisfactory. The implementation is straightforward, as is the rest of the biortho-normalisation process, and a complete listing is included in appendix 7.

9. APPLICABILITY

We know that in order to be able to orthogonalise a pair of vectors using the Gram-Schmidt process they must be linearly independent. In order to be able to biorthogonalise two sets of vectors we must have that, at each stage,

$$(\phi_i, \psi_i) \neq 0 \qquad i=1,\ldots,p \; . \tag{9.1}$$

Some rather long and tedious manipulation of equations (6.11) and (6.12) shows that this is equivalent to demanding that at the k-th stage

$$\det \begin{bmatrix} (f_1,g_1) & (f_1,g_2) & \cdots & (f_1,g_k) \\ (f_2,g_1) & (f_2,g_2) & \cdots & (f_2,g_k) \\ \cdots & \cdots & \cdots & \cdots \\ (f_k,g_1) & (f_k,g_2) & \cdots & (f_k,g_k) \end{bmatrix} \neq 0 \tag{9.2}$$

$$k = 1,2,\ldots,p \; .$$

Conversely if equation (9.2) is true for all k then it is possible to biorthonormalise the two sets of vectors.

CHAPTER 4

ITERATIVE METHODS

1. INTRODUCTION

In this chapter we give an account of some of the more commonly used iterative methods for solving the eigenvalue problem. Most methods are essentially iterative in nature but here we confine ourselves to those which are concerned with the determination of no more than a few eigensolutions.

We restrict ourselves for the moment to matrices having linear elementary divisors. For any such matrix A we have

$$\begin{aligned} A &= X.\mathrm{diag}(\lambda_i)X^{-1} \\ &= X\Delta Y^H \\ &= \sum_{i=1}^{n} \lambda_i x_i y_i^H \end{aligned} \tag{1.1}$$

where the rows y_i^H of Y^H and the columns x_i of X are the left-hand and right-hand eigenvectors of A normalised such that

$$y_i^H x_i = 1 \ . \tag{1.2}$$

Hence

$$\begin{aligned} A^s &= X.\mathrm{diag}(\lambda_i^s)Y^H \\ &= \sum_{i=1}^{n} \lambda_i^s x_i y_i^H \qquad s=1,2,3,.... \end{aligned} \tag{1.3}$$

and if

$$|\lambda_1| \geq |\lambda_2| \geq .... \geq |\lambda_r| > |\lambda_{r+1}| \geq .... \geq |\lambda_n| \tag{1.4}$$

the expression on the right of (1.3) is ultimately dominated by the terms $\sum_{i=1}^{r} \lambda_i^s x_i y_i^H$. This is the

fundamental result on which the methods of this chapter are based. We note also that it is the basis of both the LR and QR algorithms and we begin by looking briefly at these.

2. THE LR ALGORITHM

Before continuing with the discussion of methods leading up to simultaneous iteration we look at the LR algorithm due to Rutishauser (1958) and the QR algorithm of Francis (1961). We shall show that there is a close theoretical similarity between simultaneous iteration and the LR and QR algorithms.

Rutishauser's algorithm is based on the triangular decomposition of a matrix and we write

$$A = LR \tag{2.1}$$

where L is unit lower triangular and R is upper triangular. We now consider

$$L^{-1}AL = L^{-1}(LR)L = RL \ . \tag{2.2}$$

Hence if we perform a triangular decomposition of A and then multiply the factors in the reverse order we obtain a matrix similar to A. In the LR algorithm this process is repeated indefinitely and, renaming the original matrix A_1, the algorithm is defined by the equations

$$A_{s-1} = L_{s-1}R_{s-1} \ ; \quad R_{s-1}L_{s-1} = A_s \qquad s=2,3,\ldots . \tag{2.3}$$

We know A_s is similar to A_{s-1} and hence, by induction, to A_1. Rutishauser showed that under certain restrictions

$$L_s \longrightarrow I$$

and

$$R_s \longrightarrow A_s \longrightarrow \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & X & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \quad \text{as } s \to \infty . \tag{2.4}$$

We now derive some relations between the successive iterates which we shall find of use later. From equations (2.3) we have

$$A_s = L_{s-1}^{-1} A_{s-1} L_{s-1} \tag{2.5}$$

and, by repeated application, it follows that

$$A_s = (L_{s-1}^{-1} L_{s-2}^{-1} \cdots L_1^{-1}) A_1 (L_1 L_2 \cdots L_{s-1}) \tag{2.6}$$

or

$$L_1 L_2 \cdots L_{s-1} A_s = A_1 L_1 L_2 \cdots L_{s-1} . \tag{2.7}$$

The matrices T_s and U_s defined by

$$T_s = L_1 L_2 \cdots L_s \quad \text{and} \quad U_s = R_s R_{s-1} \cdots R_1 \tag{2.8}$$

are unit lower triangular and upper triangular respectively. We now consider their product.

$$\begin{aligned} T_s U_s &= L_1 L_2 \cdots L_{s-1} (L_s R_s) R_{s-1} \cdots R_2 R_1 \\ &= L_1 L_2 \cdots L_{s-1} A_s R_{s-1} \cdots R_2 R_1 \\ &= A_1 L_1 L_2 \cdots L_{s-1} R_{s-1} \cdots R_2 R_1 \end{aligned}$$

$$= A_1 T_{s-1} U_{s-1} \cdot \qquad (2.9)$$

Repeated application of this result gives us that

$$T_s U_s = A_1^s \cdot \qquad (2.10)$$

Thus $T_s U_s$ is the triangular decomposition of A_1^s.

3. CONVERGENCE OF THE LR ALGORITHM

We assume initially that the eigenvalues of A_1 satisfy

$$|\lambda_1| > |\lambda_2| > .... > |\lambda_n| \cdot \qquad (3.1)$$

Since A_1 must then have linear divisors we may write

$$A_1^s = X \mathrm{diag}(\lambda_i^s) X^{-1} = X D^s Y \cdot \qquad (3.2)$$

We define the matrices L_X, U_X, L_Y and U_Y by the relations

$$X = L_X U_X \quad ; \quad Y = L_Y U_Y \qquad (3.3)$$

where the U's are upper triangular and the L's unit lower triangular. All four matrices are independent of s and the triangular decompositions exist only if all the leading principal minors of X and Y are non-zero. From equations (3.3) we have

$$\begin{aligned} A_1^s &= X D^s Y \\ &= L_X U_X D^s L_Y U_Y \\ &= L_X U_X (D^s L_Y D^{-s}) D^s U_Y \cdot \end{aligned} \qquad (3.4)$$

Clearly $D^s L_Y D^{-s}$ is a unit lower triangular matrix and its (i,j) element is given by

$$l_{ij}(\lambda_i/\lambda_j)^s \quad \text{when } i > j \tag{3.5}$$

and hence we may write

$$D^S L_Y D^{-S} = I+E_s \quad \text{where } E_s \to 0 \text{ as } s \to \infty . \tag{3.6}$$

Equation (3.4) therefore gives

$$\begin{aligned} A_1^s &= L_X U_X (D^S L_Y D^{-S}) D^S U_Y \\ &= L_X U_X (I+E_s) D^S U_Y \\ &= L_X (I+U_X E_s U_X^{-1}) U_X D^S U_Y \\ &= L_X (I+F_s) U_X D^S U_Y \end{aligned}$$

$$\text{where } F_s \to 0 \text{ as } s \to \infty . \tag{3.7}$$

The matrix $(I+F_s)$ has a triangular decomposition for all sufficiently large s and, since $F_s \to 0$, both the factors of the decomposition tend to I. For small values of s this decomposition may not exist corresponding to the case which arises when a principal minor of some A_s is zero. Ignoring this possibility we see that, in the notation of the previous section,

$$T_s \longrightarrow L_X . \tag{3.8}$$

From this it follows that A_s tends to upper triangular form with the λ_i in the correct order on the diagonal.

If one or more of the leading principal minors of Y vanishes Wilkinson (1965) has shown

that there is nevertheless a permutation matrix P such that PY has a triangular decomposition. Denoting this by $L_Y U_Y$ we have

$$A_1^s = XD^sP^TL_YU_Y = (XP^T)(PD^sP^T)L_YU_Y \ . \tag{3.9}$$

If XP^T has a triangular decomposition L_XU_X, that is if all its leading principal minors are non-zero, then we can show as before that

$$T_s \longrightarrow L_X \tag{3.10}$$

and A_s tends to a triangular matrix with DP^T as its diagonal.

4. EIGENVALUES OF EQUAL MODULUS

We now consider the case when A_1 has some eigenvalues of equal modulus but all its elementary divisors are linear. We assume that all the leading principal minors of X and Y are non-zero as we have just considered the case when they are not. We have

$$A_1^s = XD^sL_YU_Y \ . \tag{4.1}$$

Suppose

$$|\lambda_r| = |\lambda_{r+1}| = ... = |\lambda_t| \tag{4.2}$$

and all the other eigenvalues have distinct moduli. The elements of $D^sL_YD^{-s}$ in the (i,j) position below the diagonal therefore tend to zero unless

$$t \geqslant i > j \geqslant r \tag{4.3}$$

in which case they remain equal in modulus to l_{ij}.

When all the eigenvalues of equal modulus are in fact equal we may write

$$D^S L_Y D^{-S} = L+E_S \quad \text{where } E_S \to 0 \text{ as } s\to\infty \tag{4.4}$$

where L is a fixed unit lower triangular matrix which is equal to I, except for the elements in position (i,j) which satisfy (4.3) where they are equal to l_{ij}. If we write

$$XL = L_X U_X \tag{4.5}$$

then we have

$$\begin{aligned} A_1^S &= X(D^S L_Y D^{-S}) D^S U_Y \\ &= X(L+E_S) D^S U_Y \\ &= L_X U_X (I+L^{-1}E_S) D^S U_Y \\ &= L_X (I+U_X L^{-1} E_S U_X^{-1}) U_X D^S U_Y \\ &= L_X (I+F_S) U_X D^S U_Y \quad \text{where } F_S \to 0 \text{ as } s\to\infty . \end{aligned} \tag{4.6}$$

The matrix $(I+F_S)$ has, as before, a triangular decomposition for all sufficiently large s and, since $F_S \to 0$, both the factors of the decomposition tend to I. Hence we see again that

$$T_S \longrightarrow L_X \tag{4.7}$$

where L_X is the matrix obtained by factorising XL. We notice that the columns of XL are a set of linearly independent eigenvectors of A_1 since XL differs from X only in that columns r to t are

replaced by combinations of themselves. Thus multiple eigenvalues corresponding to linear divisors do not prevent convergence.

If the eigenvalues are of equal modulus but are not actually equal then the matrix L is of a similar form as before but the non-zero subdiagonal elements are not now fixed. If

$$\lambda_i = |\lambda_i| \exp(i\theta_i) \tag{4.8}$$

we have

$$l_{ij} = l_{ij} \exp[is(\theta_i - \theta_j)] \ . \tag{4.9}$$

The matrix XL is fixed apart from columns r to t. For each value of s these columns consist of a linear combination of the corresponding columns of X. Thus in the $L_X U_X$ decomposition of XL all columns of L_X are fixed except for columns r to t and hence, apart from these columns, T_s is convergent.

We also mention for the sake of completeness that if A_1 is symmetric and positive definite the process is always convergent, irrespective of the multiplicities of the eigenvalues. Further details may be found in Wilkinson (1965).

5. THE QR ALGORITHM

We now turn to the QR algorithm of Francis (1961). In place of the triangular decomposition used in the LR algorithm Francis uses a

factorisation into the product of a unitary matrix Q and an upper triangular matrix R.

The algorithm is defined by the equations

$$\begin{aligned} A_s &= Q_s R_s \\ A_{s+1} &= Q_s^H A_s Q_s \\ &= Q_s^H Q_s R_s Q_s \\ &= R_s Q_s \ , \end{aligned} \tag{5.1}$$

and at each stage we see that we are now using a unitary transformation in place of the general similarity transformation of the LR algorithm given by equations (2.3). The factorisation of A_s is essentially unique and indeed it is unique if we take the diagonal elements of R_s to be real and positive. The advantage of this factorisation is that the vanishing of a principal minor of A_s does not cause a breakdown of the process as it does in the LR decomposition. We note that if A_s is real both Q_s and R_s are real.

The successive iterates satisfy relations similar to those for the LR algorithm which we derived in section 2. We have

$$\begin{aligned} A_{s+1} &= Q_s^H A_s Q_s \\ &= Q_s^H (Q_{s-1}^H A_{s-1} Q_{s-1}) Q_s \\ &= (Q_s^H Q_{s-1}^H \ \ldots\ Q_1^H) A_1 (Q_1 Q_2 \ \ldots\ Q_s) \ , \end{aligned} \tag{5.2}$$

which gives us

$$Q_1Q_2 \cdots Q_sA_{s+1} = A_1Q_1Q_2 \cdots Q_s \ . \qquad (5.3)$$

All A_s are therefore unitarily similar to A_1 and if we define P_s and U_s by

$$P_s = Q_1Q_2 \cdots Q_s \ ; \quad U_s = R_sR_{s-1} \cdots R_1 \qquad (5.4)$$

and consider their product we obtain

$$\begin{aligned} P_sU_s &= Q_1Q_2 \cdots Q_{s-1}(Q_sR_s)R_{s-1} \cdots R_1 \\ &= Q_1Q_2 \cdots Q_{s-1}A_sR_{s-1} \cdots R_1 \\ &= A_1Q_1Q_2 \cdots Q_{s-1}R_{s-1} \cdots R_1 \\ &= A_1P_{s-1}U_{s-1} \ . \end{aligned} \qquad (5.5)$$

Hence

$$P_sU_s = A_1^s \ . \qquad (5.6)$$

Thus P_sU_s is the factorisation of A_1^s.

6. CONVERGENCE OF THE QR ALGORITHM

In general the matrix A_s tends to upper triangular form under similar, but less stringent, conditions than were necessary for convergence of the LR algorithm. We note firstly one small difference between the two algorithms.

If A_1 is already an upper triangular matrix the LR algorithm gives us

$$L_1 = I \ ; \quad R_1 = A_1 \qquad (6.1)$$

and hence

$$A_s = A_1 \quad \text{for all } s. \qquad (6.2)$$

This is not true however for the QR algorithm if we insist that R_s should have positive diagonal elements. Writing

$$a_{ii} = |a_{ii}|\exp(i\theta_i) , \qquad D = \mathrm{diag}(\exp(i\theta_i)) \tag{6.3}$$

we have

$$A_1 = D(D^{-1}A_1) , \qquad A_2 = D^{-1}A_1D . \tag{6.4}$$

Hence, although A_2 has the same diagonal elements as A_1 the super diagonal elements are multiplied by complex factors of modulus unity. Obviously we cannot have A_s tending to a strict limit unless all the eigenvalues are real and positive. However, the factors of modulus unity are of little importance and we say that A_s is essentially convergent if

$$A_{s+1} \longrightarrow D^{-1}A_sD \tag{6.5}$$

asymptotically for some unitary diagonal matrix D.

The proof of convergence is very similar to that given for the LR algorithm. We assume initially that the eigenvalues of A_1 satisfy

$$|\lambda_1| > |\lambda_2| > \ldots . > |\lambda_n| . \tag{6.6}$$

As before we may write

$$A_1^s = XD^sY \tag{6.7}$$

but we now define Q,R,L and U by

$$X = QR , \qquad Y = LU \tag{6.8}$$

where R and U are upper triangular, L is unit lower

triangular and Q is unitary. The QR decomposition always exists but, as before, the triangular decomposition of Y exists only if all its leading principal minors are non-zero. The non-singularity of R follows from that of X.

An analysis similar to that in section 3 gives us

$$A_1^s = Q(I+F_s)RD^sU \tag{6.9}$$

where

$$F_s = RE_sR^{-1} \tag{6.10}$$

and

$$F_s \longrightarrow 0 \quad \text{as } s \longrightarrow \infty \ . \tag{6.11}$$

The matrix $(I+F_s)$ may be factorised into the product of a unitary matrix Q_s and an upper triangular matrix R_s and, since $F_s \rightarrow 0$, Q_s and R_s both tend to I. Hence we have

$$A_1^s = (QQ_s)(R_sRD^sU) \ . \tag{6.12}$$

The first factor of (6.12) is unitary and the second is upper triangular. Provided A_1^s is non-singular its factorisation into this product is unique and hence

$$P_s = QQ_s \tag{6.13}$$

except possibly for a multiplying diagonal unitary matrix. Hence P_s converges essentially to Q. If we insist additionally that all R_s have positive

diagonal elements then it is possible to calculate the unitary diagonal factor from equation (6.12).

The proof shows that provided all leading principal minors of Y are non-zero we not only have convergence but that the eigenvalues are correctly ordered on the diagonal. Although Y will not have a triangular decomposition when one of its principal minors vanishes there is always a permutation matrix P so that PY has such a decomposition. The reasoning is the same as that for the LR algorithm given in section 3.

7. EIGENVALUES OF EQUAL MODULUS

We again assume that A_1 has linear elementary divisors but that some of its eigenvalues are of equal modulus. The analysis is similar to that of section 4 except that we replace equation (4.5) by

$$XL = QR \tag{7.1}$$

which gives us

$$\begin{aligned} A_1^s &= QR(I+L^{-1}E_s)D^sU \\ &= Q(I+RL^{-1}E_sR^{-1})RD^sU \\ &= Q(I+F_s)RD^sU \\ &= (QQ_s)(R_sRD^sU) \qquad \text{where } F_s \to 0 \text{ as } s\to\infty \end{aligned} \tag{7.2}$$

where Q_sR_s is the factorisation of $(I+F_s)$. Hence P_s tends to Q which is the matrix obtained by

factorising XL. The rest of the discussion is as for the LR algorithm given in section 4.

8. THE POWER METHOD

The simplest application of the idea of section 1 is to the power method for determining the dominant eigensolution. Let u_o be an arbitrary vector and let the sequences v_s and u_s be defined by the equations

$$\begin{aligned} v_{s+1} &= Au_s \\ u_{s+1} &= v_{s+1}/\max(v_{s+1}) \end{aligned} \qquad (8.1)$$

where we use the notation max(x) to denote the element of maximum modulus of the vector x. Clearly we have

$$u_s = A^s u_o/\max(A^s u_o) \qquad (8.2)$$

and if we write

$$u_o = \sum_1^n a_i x_i \qquad (8.3)$$

then, apart from the normalising factor, u_s is given by

$$\sum_1^n a_i \lambda_i^s x_i = \lambda_1^s [a_1 x_1 + \sum_2^n a_i (\lambda_i/\lambda_1)^s x_i] \; . \qquad (8.4)$$

If $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots . \geq |\lambda_n|$ then, provided $a_1 \neq 0$, we have

$$u_s \longrightarrow x_1/\max(x_1) \text{ and } \max(v_s) \longrightarrow \lambda_1 \; . \qquad (8.5)$$

Hence this process provides simultaneously the

dominant eigenvalue and the corresponding eigenvector. If $|\lambda_1/\lambda_2|$ is close to unity the convergence is very slow.

If there are a number of independent eigenvectors corresponding to the dominant eigenvalue this does not affect the convergence. Thus if

$$\lambda_1 = \lambda_2 = \ldots = \lambda_r$$

and

$$|\lambda_1| > |\lambda_{r+1}| \geqslant \ldots \geqslant |\lambda_n| \tag{8.6}$$

we have

$$A^s u_o = \lambda_1^s \left[\sum_1^r a_i x_i + \sum_{r+1}^n a_i (\lambda_i/\lambda_1)^s x_i\right]$$
$$\longrightarrow \lambda_1^s \sum_1^r a_i x_i \ . \tag{8.7}$$

The iterates therefore tend to some vector lying in the subspace spanned by the eigenvectors $(x_1,\ldots,x_r)$, the limit depending upon the initial vector u_o.

9. COMPLEX CONJUGATE EIGENVALUES

If the dominant eigenvalues of a real matrix are a complex conjugate pair λ_1 and $\bar{\lambda}_1$ the iterated vectors will not converge. In fact if x_1 and $\bar{x}_1$ are the corresponding eigenvectors an arbitrary real vector u_o is expressible in the form

$$u_o = a_1 x_1 + \bar{a}_1 \bar{x}_1 + \sum_{i=3}^n a_i x_i \ . \tag{9.1}$$

Hence we have

$$A^s u_o = r_1^s \left[\rho_1 e^{i(\alpha+s\theta)} x_1 + \rho_1 e^{-i(\alpha+s\theta)} \bar{x}_1 + \sum_3^n a_i (\lambda_i/r_1)^s x_i\right] \tag{9.2}$$

where

$$\lambda_1 = r_1 e^{i\theta} \quad ; \quad a_1 = \rho_1 e^{i\alpha} \, . \tag{9.3}$$

The components of $x_3, \ldots, x_n$ ultimately die out, but if we write

$$v_{s+1} = Au_s \, , \quad \max(v_{s+1}) = k_{s+1} \, , \quad u_{s+1} = v_{s+1}/k_{s+1} \tag{9.4}$$

it is clear from (9.2) that neither k_{s+1} nor u_{s+1} tend to a limit. If we denote the j-th component of x_1 by $\xi_j \exp(i\phi_j)$ equation (9.2) gives

$$(A^s u_o)_j \longrightarrow 2\rho_1 r_1^s \xi_j \cos(\alpha+\phi_j+s\theta) \tag{9.5}$$

and hence the components of u_s oscillate in sign.

If λ_1 and $\bar{\lambda}_1$ are the roots of

$$\lambda^2 - p\lambda - q = 0 \tag{9.6}$$

we have

$$(A^{s+2} - pA^{s+1} - qA^s)u_o \longrightarrow 0, \; s \to \infty \tag{9.7}$$

or

$$k_{s+1}k_{s+2}u_{s+2} - pk_{s+1}u_{s+1} - qu_s \longrightarrow 0 \tag{9.8}$$

and hence ultimately any three successive iterates are linearly dependent. By the method of least squares we can determine successive approximations p_s and q_s to p and q. We have

$$k_{s+1}k_{s+2}\begin{bmatrix} u_{s+1}^T u_{s+2} \\ u_s^T u_{s+2} \end{bmatrix}$$

$$= \begin{bmatrix} u_{s+1}^T u_{s+1} & u_{s+1}^T u_s \\ u_s^T u_{s+1} & u_s^T u_s \end{bmatrix}\begin{bmatrix} p_s k_{s+1} \\ q_s \end{bmatrix}. \qquad (9.9)$$

When p_s and q_s have tended to limits p and q, λ_1 and $\bar{\lambda}_1$ may be computed from the relations

$$\mathrm{Re}(\lambda_1) = \tfrac{1}{2}p\ , \qquad \mathrm{Im}(\lambda_1) = \tfrac{1}{2}(p^2+4q)^{\frac{1}{2}}\ . \qquad (9.10)$$

10. SIMULTANEOUS DETERMINATION OF SEVERAL EIGENVALUES

The essential feature of the previous section is the determination of two eigenvalues from a single sequence of iterates. The fact that the eigenvalues were complex conjugate is really not pertinent and the method may be extended to cover the determination of several real or complex eigenvalues. Suppose for example that

$$|\lambda_1| \geqslant |\lambda_2| \geqslant |\lambda_3| > |\lambda_4| \geqslant \ldots. \geqslant |\lambda_n|. \qquad (10.1)$$

The components of x_4 to x_n will die out rapidly in the iterated vectors and we shall soon reach a stage at which u_s is effectively given by $a_1x_1+a_2x_2+a_3x_3$. If we define quantities p_2, p_1 and p_0 by the equation

$$(\lambda-\lambda_1)(\lambda-\lambda_2)(\lambda-\lambda_3) \equiv \lambda^3+p_2\lambda^2+p_1\lambda+p_0 \qquad (10.2)$$

then

$$(A^3+p_2A^2+p_1A+p_0I)u_s = 0 \qquad (10.3)$$

giving

$$-(A^3u_s) = (A^2u_s \ , \ Au_s \ , \ u_s)\begin{bmatrix} p_2 \\ p_1 \\ p_0 \end{bmatrix} . \qquad (10.4)$$

The coefficients p_i may therefore be obtained by least squares from any four consecutive iterates following u_s.

The use of such a technique would have most to recommend it when $|\lambda_1|$, $|\lambda_2|$ and $|\lambda_3|$ were close, since it is in such circumstances that the iterates are slow to converge to the dominant eigenvector. Unfortunately if λ_1, λ_2 and λ_3 are of the same sign and close together the equations determining the p_i are ill-conditioned. For these reasons the method is of little practical value. In general to determine r eigenvalues in a well-conditioned manner we require r independent sets of iterated vectors. Such methods we discuss later.

11. DEFLATION

If A is real and has real eigenvalues it is possible to employ shifts of origin to give

convergence either to λ_1 or to λ_n. In the case of complex matrices, by using appropriate shifts of origin, we could in principle make a number of the eigenvalues dominant in turn but in practice this device would usually be prohibitively difficult to use.

It is natural to ask whether we can make use of our knowledge of λ_1 and x_1 in such a way that another eigenvector may be found without danger of again converging to x_1. One class of such methods depends essentially on replacing A by a matrix which possesses only the remaining eigenvalues. We shall refer to such methods as methods of deflation.

Probably the simplest is that due to Hotelling (1933) which can be applied when an eigenvalue λ_1 and vector x_1 of a symmetric matrix A_1 are known. If we define A_2 by the relation

$$A_2 = A_1 - \lambda_1 x_1 x_1^T \quad \text{where } x_1^T x_1 = 1 \qquad (11.1)$$

then from the orthogonality of the x_i we have

$$\begin{aligned} A_2 x_i = A_1 x_i - \lambda_1 x_1 x_1^T x_i &= 0 \quad , \quad i=1 \\ &= \lambda_i x_i \quad , \quad i \neq 1 \ . \end{aligned} \qquad (11.2)$$

Hence the eigenvalues of A_2 are $0, \lambda_2, \ldots, \lambda_n$ corresponding to eigenvectors $x_1, x_2, \ldots, x_n$ and the dominant eigenvalue λ_1 has been reduced to

zero.

When A_1 is unsymmetric there is a corresponding deflation technique also due to Hotelling (1933) but it requires the determination of the left-hand eigenvector y_1 as well as x_1. If both are determined and normalised so that $y_1^H x_1 = 1$ then, defining A_2 by

$$A_2 = A_1 - \lambda_1 x_1 y_1^H , \tag{11.3}$$

we have from the biorthogonality of the x_i and y_i

$$\begin{aligned} A_2 x_i = A_1 x_i - \lambda_1 x_1 y_1^H x_i &= 0 \quad , \quad i=1 \\ &= \lambda_i x_i \quad , \quad i \neq 1 \end{aligned} \tag{11.4}$$

We have found in practice that these two methods of deflation have rather poor numerical stability and their use is not to be recommended.

12. TREPPEN-ITERATION

One problem in using iteration and deflation is that in general we do not know at each stage whether the current dominant eigenvalue is real or complex, repeated, or whether it belongs to a non-linear divisor. It is possible to avoid this difficulty by working simultaneously with a complete set of n vectors and ensuring that they are independent by taking these n vectors to be the columns of a unit lower triangular matrix. If at each stage we denote the matrix formed by

the set of n vectors by L_s the process can be summarised as

$$X_{s+1} = AL_s \quad ; \quad X_{s+1} = L_{s+1}R_{s+1} \tag{12.1}$$

where each L_s is unit lower triangular and each R_s is upper triangular. If we take

$$L_0 = I \tag{12.2}$$

then we have

$$L_sR_s = X_s = AL_{s-1} \, . \tag{12.3}$$

Thus

$$\begin{aligned} L_sR_sR_{s-1} &= AL_{s-1}R_{s-1} \\ &= AAL_{s-2} \\ &= A^2L_{s-2} \, , \end{aligned} \tag{12.4}$$

therefore

$$L_s(R_sR_{s-1} \ldots\ldots R_1) = A^sL_0 = A^s. \tag{12.5}$$

Hence L_s and $(R_sR_{s-1} \ldots\ldots R_1)$ are the matrices obtained by the triangular decomposition of A^s and, by comparison with the LR algorithm of section 2, L_s is equal to the product of the first s lower triangular matrices obtained in the LR algorithm while the R_s are identical with the individual upper triangular matrices in the same algorithm. Although the LR and treppen-iteration algorithms are theoretically similar it does not follow that in practice their behaviour will be even approximately alike.

There is no need to use a complete set of n vectors and it is possible to work with a set of p vectors in unit lower trapezoidal form. Denoting these by T_s the process becomes:

$$AT_s = X_{s+1} \quad ; \quad X_{s+1} = T_{s+1}R_{s+1} \tag{12.6}$$

where R_s is a p*p upper triangular matrix. If the p dominant eigenvalues of A have distinct moduli then

$$T_s \longrightarrow T \tag{12.7}$$

where T is obtained by trapezoidal decomposition of the matrix of p eigenvectors. In general if

$$|\lambda_1| \geqslant |\lambda_2| \geqslant \ldots\ldots \geqslant |\lambda_p| > |\lambda_{p+1}| \geqslant \ldots\ldots \geqslant |\lambda_n| \tag{12.8}$$

T_s does not tend to a limit but it does tend to an invariant subspace. This process was first described by Bauer (1957) and was called treppen-iteration.

If

$$|\lambda_1| \gg |\lambda_2| \tag{12.9}$$

the first column of T_s will converge to x_1 in a few iterations and at this stage there is little point in including x_1 in the matrix T_s when computing X_{s+1}. In general if the first k vectors of T_s have converged we need not multiply these vectors by A in subsequent steps. We may write

$$T_s = [T_s^{(1)}, T_s^{(2)}] \qquad (12.10)$$

where $T_s^{(1)}$ consists of the first k vectors, which have converged, and $T_s^{(2)}$ consists of the remaining (p-k) vectors which have not. We now define X_{s+1} by

$$X_{s+1} = [T_s^{(1)}, AT_s^{(2)}] \qquad (12.11)$$

where $T_s^{(1)}$ is already in trapezoidal form but $AT_s^{(2)}$ consists, in general, of (p-k) full vectors. X_{s+1} is then reduced to trapezoidal form in the obvious way.

13. ORTHOGONALISATION TECHNIQUES

We now turn to an alternative technique for suppressing the dominant eigenvector (or eigenvectors). The simplest application is to real symmetric matrices and these we now consider.

Suppose λ_1 and x_1 have been determined so that

$$Ax_1 = \lambda_1 x_1 \qquad (13.1)$$

where A is real and symmetric. We know that the remaining eigenvectors are orthogonal to x_1 and hence we may suppress the component of x_1 in any of the other vectors by orthogonalising them with respect to x_1. This leads to the iterative procedure defined by the equations:

$$v_{s+1} = Au_s \ , \qquad w_{s+1} = v_{s+1} - (v_{s+1}^T x_1)x_1 \ ,$$

$$u_{s+1} = w_{s+1}/\max(w_{s+1}) \tag{13.2}$$

where we assume that

$$\|x_1\|_2 = 1 \ . \tag{13.3}$$

Clearly u_s tends to the subdominant eigenvector or, if A has a second eigenvalue equal to λ_1, it tends to an eigenvector corresponding to λ_1 which is orthogonal to x_1.

Unless

$$|\lambda_1| \gg |\lambda_2| \tag{13.4}$$

it is not strictly necessary to orthogonalise with respect to x_1 at each iteration but if A is of high order the work involved in the orthogonalisation is in any case relatively small.

This process may be generalised to find x_{r+1} when $x_1, x_2, \ldots, x_r$ have already been determined. The corresponding iteration is defined by

$$v_{s+1} = Au_s \ , \qquad w_{s+1} = v_{s+1} - \sum_{i=1}^{r} (v_{s+1}^T x_i)x_i \ ,$$

$$u_{s+1} = w_{s+1}/\max(w_{s+1}) \ . \tag{13.5}$$

We note that, apart from rounding errors, this method gives results identical with those of Hotelling described in section 11, for, from (13.2) we have,

$$w_{s+1} = v_{s+1} - (v_{s+1}^T x_1)x_1$$

$$= Au_s - x_1(u_s^T Ax_1)$$

$$= Au_s - \lambda_1 x_1(u_s^T x_1)$$

$$= Au_s - \lambda_1 x_1(x_1^T u_s)$$

$$= (A - \lambda_1 x_1 x_1^T)u_s \ . \qquad (13.6)$$

The analogous process for an unsymmetric matrix A requires the computation of both the left-hand eigenvector y_1 and the right-hand eigenvector x_1. If these are normalised so that

$$x_1^H y_1 = 1 \qquad (13.7)$$

we may use the iterative procedure

$$v_{s+1} = Au_s \ , \quad w_{s+1} = v_{s+1} - (v_{s+1}^H y_1)x_1 \ ,$$

$$u_{s+1} = w_{s+1}/\max(w_{s+1}) \ . \qquad (13.8)$$

Because of the biorthogonality of the left-hand and right-hand eigenvectors the component of x_1 in w_{s+1} is suppressed. Again, there is the generalisation when r eigenvectors have been determined:

$$v_{s+1} = Au_s \ , \quad w_{s+1} = v_{s+1} - \sum_{i=1}^{r} (v_{s+1}^H y_i)x_i \ ,$$

$$u_{s+1} = w_{s+1}/\max(w_{s+1}) \ . \qquad (13.9)$$

14. TREPPEN-ITERATION USING ORTHOGONALISATION

In treppen-iteration we iterate simultaneously with a number of vectors whose independence is

maintained by reducing them to standard trapezoidal form at each step. There is an analogous procedure in which the independence of the vectors is maintained by using the Gram-Schmidt orthogonalisation process. Consider firstly the case when we iterate simultaneously with n vectors. Denoting the matrix formed by these vectors at each stage by Q_s the process may be summarised as

$$AQ_s = V_{s+1} , \qquad V_{s+1} = Q_{s+1}R_{s+1} \tag{14.1}$$

where the columns of Q_{s+1} are orthogonal and R_{s+1} is upper triangular. Hence

$$AQ_s = Q_{s+1}R_{s+1} \tag{14.2}$$

and if we take

$$Q_0 = I , \tag{14.3}$$

we have

$$A^s = Q_s(R_sR_{s-1} \ldots\ldots R_1) , \tag{14.4}$$

by reasoning similar to that of section 12. Equation (14.4) implies that Q_s and $(R_sR_{s-1} \ldots\ldots R_1)$ are the matrices obtained by orthogonal triangularisation of A^s. Comparison with the QR algorithm of section 5 shows that Q_s is equal to the product of the first s orthogonal matrices determined by the QR process. Thus, if all the $|\lambda_i|$ are different, Q_s essentially tends to a limit, this being that matrix obtained from the

triangular orthogonalisation of X, the matrix of eigenvectors. The diagonal elements of R_s tend to $|\lambda_i|$. In the general case, where some of the eigenvalues are of the same modulus, the corresponding columns of Q_s may not tend to a limit but they ultimately span the appropriate subspaces. We note, as in section 6, that theoretical equivalence of this method with the QR algorithm does not imply that in practice they will produce even similar results.

15. BI-ITERATION

Bauer (1957) suggested a generalisation of the methods which we have so far discussed and this he called bi-iteration. At each stage two systems of n vectors, $\{x_i\}$ and $\{y_i\}$, are used and these comprise the columns of two matrices X_s and Y_s. The two matrices P_{s+1} and Q_{s+1} defined by

$$P_{s+1} = AX_s , \qquad Q_{s+1} = A^T Y_s \tag{15.1}$$

are formed and from them the two matrices X_{s+1} and Y_{s+1} are derived. The columns of these matrices are chosen to be biorthogonal and the equations defining the i-th columns of X_{s+1} and Y_{s+1} are therefore

$$x_i^{(s+1)} = p_i^{(s+1)} - r_{1i}x_1^{(s+1)} - r_{2i}x_2^{(s+1)} -$$
$$- \cdots - r_{i-1,i}x_{i-1}^{(s+1)}$$
$$y_i^{(s+1)} = q_i^{(s+1)} - u_{1i}y_1^{(s+1)} - u_{2i}y_2^{(s+1)} -$$
$$- \cdots - u_{i-1,i}y_{i-1}^{(s+1)} \qquad (15.2)$$

where the r_{ki} and u_{ki} are chosen such that

$$(y_k^{(s+1)})^T x_i^{(s+1)} = 0 , \quad (x_k^{(s+1)})^T y_i^{(s+1)} = 0$$
$$k = 1,\ldots,i-1 . \qquad (15.3)$$

Clearly we have

$$P_{s+1} = X_{s+1}R_{s+1} , \quad Q_{s+1} = Y_{s+1}U_{s+1} , \qquad (15.4)$$

where R_{s+1} and U_{s+1} are the unit upper triangular matrices formed from the r_{ki} and u_{ki}. From the biorthogonality we have

$$Y_{s+1}^T X_{s+1} = D_{s+1} , \qquad (15.5)$$

where D_{s+1} is a diagonal matrix. If we take

$$X_0 = Y_0 = I \qquad (15.6)$$

we have

$$A=X_1R_1 , \quad AX_s=X_{s+1}R_{s+1} ,$$
$$A^T=Y_1U_1 , \quad A^TY_s=Y_{s+1}U_{s+1} , \qquad (15.7)$$

giving

$$A^s = X_sR_sR_{s-1}\cdots R_1 \qquad (A^T)^s = Y_sU_sU_{s-1}\cdots U_1 . \qquad (15.8)$$

Hence

$$A^{2s} = (Y_s U_s U_{s-1} \cdots U_1)^T X_s R_s R_{s-1} \cdots R_1$$

$$= (U_1^T U_2^T \cdots U_s^T) D_s (R_s R_{s-1} \cdots R_1) \qquad (15.9)$$

showing that the unit lower-triangular matrix $(U_1^T U_2^T \cdots U_s^T)$ is that corresponding to the triangular decomposition of A^{2s}. In the LR algorithm $(L_1 L_2 \cdots L_{2s})$ is the matrix obtained by the triangular decomposition of A^{2s} and hence

$$U_s^T = L_{2s-1} L_{2s} \, . \qquad (15.10)$$

Results from the LR algorithm carry over immediately to bi-iteration. If the $|\lambda_i|$ are distinct $(U_1^T U_2^T \cdots U_s^T)$ tends to the matrix given by the triangular decomposition of X_s and hence

$$U_s \longrightarrow I$$

and

$$R_s \longrightarrow I \, . \qquad (15.11)$$

In practice the columns of X_s and Y_s are normalised at each stage, usually so that

$$\max(x_i^{(s)}) = \max(y_i^{(s)}) = 1. \qquad (15.12)$$

Hence in the case of distinct $|\lambda_i|$

$$X_s \longrightarrow X \quad \text{and} \quad Y_s \longrightarrow Y \qquad (15.13)$$

where X and Y are the matrices formed by the right-hand and left-hand eigenvectors respectively. When some of the $|\lambda_i|$ are equal the subspaces formed

by the corresponding columns of X_s and Y_s tend to the relevant invariant subspaces. If A is symmetric the two systems X_s and Y_s are identical and the method becomes essentially that of section 13 as the columns of X_s are made orthogonal at each stage.

In order to facilitate comparison with the LR algorithm we have considered the case when X_s and Y_s consist of complete systems of n vectors but the process can still be used when they have any number of columns, p say, from 1 to n. Exactly the same equations apply but now R_s and U_s are p*p matrices. In general the process will provide the p dominant left-hand and right-hand eigenvectors or invariant subspaces.

It is against this background that we proceed to consider simultaneous iteration and its developments.

16. SIMULTANEOUS ITERATION

The particular case of bi-iteration for symmetric positive definite matrices was discussed by Rutishauser (1969) who later (1970) published an algol program for this method. We have also discussed the symmetric case, Gudgin (1971), and here we simply recall the main points.

The basic idea of bi-iteration was that two

sets of iteration vectors $(x_1,x_2,....,x_p)$ and $(y_1,y_2,....,y_p)$ are iterated simultaneously with A and A^T respectively. The iteration is then combined with linear iterations such that at any time the systems x_i and y_j are biorthogonal. It is assumed that

$$1 \leqslant p \leqslant n \tag{16.1}$$

although usually

$$1 < p \ll n . \tag{16.2}$$

If A is symmetric the two sets of iteration vectors can be chosen to be identical and they then form a system of orthonormal vectors $(x_1,x_2,....,x_p)$ which are the columns of an n*p matrix X such that

$$X^TX = I_p. \tag{16.3}$$

Denoting the matrix X after k iteration steps by X_k the method is given by

$$\begin{aligned} &\text{i) Choose } X_0 \text{ such that } X_0^TX_0=I \\ &\left.\begin{aligned} &\text{ii) } Z_k=AX_{k-1} \\ &\text{iii) } X_k=Z_kR_k^{-1} \end{aligned}\right\} \quad k=1,2,.... \end{aligned} \tag{16.4}$$

where R_k is an upper triangular matrix with positive diagonal elements chosen such that X_k has its columns orthogonal. This implies that

$$Z_k^TZ_k = (X_kR_k)^T(X_kR_k) = R_k^TR_k \tag{16.5}$$

and we compute R_k by Gram-Schmidt orthogonalisation

of the columns of Z_k. Provided A is positive definite it will be shown that with equations (16.4)

$$\lim_{k\to\infty} X_k = V$$

$$\lim_{k\to\infty} R_k = D \text{ , a p*p diagonal matrix} \tag{16.6}$$

where both limits exist and

$$V^T A V = D \text{ .} \tag{16.7}$$

Thus the columns of X_k converge to eigenvectors and the diagonal elements of R_k to eigenvalues of the matrix A.

This result can be established by showing that (16.4) is equivalent to the LRCH transformation (by which we mean the LR transformation with Cholesky decomposition into two transposed factors - see for example Wilkinson (1965).) if the latter is applied to the matrix

$$G_1 = X_0^T A^T A X_0 = X_0^T A^2 X_0 \tag{16.8}$$

where X_0 is an n*n orthogonal matrix obtained by appending (n-p) further columns to X_0. If we use (16.4) with the initial matrix X_0 then the k-th iterate X_k is related to the k-th LRCH transformation G_{k+1} of G_1 by

$$G_{k+1} = X_k^T A^2 X_k \tag{16.9}$$

where X_k is contained in the first p columns of X_k.

This also enables us to estimate the convergence rate, for letting A have eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p > \lambda_{p+1} \geq \ldots \geq \lambda_n > 0 \quad (16.10)$$

and if v_j is the eigenvector corresponding to λ_j then, denoting the j-th column of X_k by $x_j^{(k)}$, we have

$$\|v_j - x_j^{(k)}\| = O(q^k) \quad (16.11)$$

where $q=\max(\lambda_{j+1}/\lambda_j\ ,\ \lambda_j/\lambda_{j-1})$. A proof of this theorem is to be found in Bauer (1957).

The scheme given in (16.4) conceals the fact that we are actually performing an iteration with p-dimensional spaces. Letting E_k denote the linear space spanned by the columns of X_k we have

$$E_k = \left\{x \mid x=Ay,\ y \in E_{k-1}\right\} . \quad (16.12)$$

We know that if A is symmetric and positive definite, then $\lim_{k\to\infty} E_k$ exists and is an invariant subspace of A. We note that the X_k given by (16.4) appear simply as a means for spanning E_k but that, for example,

$$X_k = AX_{k-1} \quad (16.13)$$

also defines the sequence E_k. However the X_k produced by (16.13) are of little practical use since they are not in general numerically stable while those produced by (16.4) are.

In the case of stable convergence the angle $\phi_j^{(k)}$ between the j-th eigenvector v_j and E_k is asymptotically for $k \to \infty$

$$O(q_j^k) \qquad \text{where } q_j = \lambda_{p+1}/\lambda_j \,. \tag{16.14}$$

To establish this we note that (16.4) and (16.13) are both orthogonally invariant. By this we mean that if U is an n*n orthogonal matrix then replacing A by U^TAU and X_0 by U^TX_0 has the effect that all X_k are replaced by U^TX_k and the R_k are unchanged. Thus we can assume without loss of generality that

$$A = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n). \tag{16.15}$$

Now E_0 can be spanned by the p vectors

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \\ x_{p+1,1} & x_{p+1,2} & \cdots & x_{p+1,p} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \tag{16.16}$$

and hence from (16.12) E_k is spanned by the vectors

$$\begin{bmatrix} \lambda_1^k & 0 & \cdots & 0 \\ 0 & \lambda_2^k & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \lambda_p^k \\ \lambda_{p+1}^k x_{p+1,1} & \lambda_{p+1}^k x_{p+1,2} & \cdots & \lambda_{p+1}^k x_{p+1,p} \\ \cdots & \cdots & \cdots & \cdots \\ \lambda_n^k x_{n1} & \lambda_n^k x_{n2} & \cdots & \lambda_n^k x_{np} \end{bmatrix} . \qquad (16.17)$$

From (16.17) it follows immediately that the angle $\phi_j^{(k)}$ is at most $O(\lambda_{p+1}^k/\lambda_j^k)$.

This result also shows the interesting and important fact that there are directions in E_k which are closer to the eigenvectors v_j than the columns of the matrices X_k as generated by (16.4). Improved convergence may be obtained by a modified iteration procedure for which the convergence rate is the same as given in (16.14). In his paper of 1969 Rutishauser gives such a scheme and this we now describe; the algorithm we have implemented is explained in the next section.

Rutishauser defines the "projected eigenvalue equation"

$$Y^T A^{-2} Y = D_k^{-2} \qquad (16.18)$$

where $Y^TY=I_p$ and D_k^{-2} is a p*p diagonal matrix. The matrix Y is the matrix whose columns are the projection of the columns of A^{-2} onto E_k. The solution of (16.18) is

$$Y = AX_{k-1}Q_kD_k^{-1} \tag{16.19}$$

where Q_k is a p*p orthogonal matrix which transforms

$$G_k=X_{k-1}^TA^2X_{k-1} \quad \text{into} \quad Q_k^TG_kQ_k=D_k^2 \tag{16.20}$$

and X_{k-1} is the previous iteration matrix, the columns of which are assumed to be orthogonal. Assuming the diagonal elements of D_k are in decreasing order such that

$$d_{11} \geq d_{22} \geq \ldots\ldots \geq d_{pp} > 0 \tag{16.21}$$

Rutishauser (1969) suggests the following scheme for obtaining X_k from X_{k-1}:

i) $Z_k=AX_{k-1}$

ii) $G_k=Z_k^TZ_k$

iii) Solve the eigenvalue problem for G_k.

That is, compute Q_k and D_k as in (16.20)

iv) $X_k=Z_kQ_kD_k^{-1}$. (16.22)

Using equations (16.22) the convergence is given, not by (16.11), but by

$$\|v_j - x_j^{(k)}\| = O(q_j^k) \tag{16.23}$$

where $q_j=\lambda_{p+1}/\lambda_j$.

This scheme has been tried in practice and found to suffer from the disadvantage that there is no guarantee that the computed X_k will be orthogonal. Additionally, large errors may be introduced in step (iv) of (16.22) if any of the d_{ii} are small.

17. AN ALTERNATIVE APPROACH TO SIMULTANEOUS ITERATION

We now return to the ideas of section 13 for an alternative and simpler approach to simultaneous iteration. This also leads directly to our implementation of the method.

Previously we saw that if we had determined λ_1 and x_1, where we assume the eigenvalues to be in non-increasing order, it was possible to determine λ_2 and x_2 by iterating with an arbitrary vector which had been orthogonalised with respect to x_1. In general when $x_1,x_2,\ldots,x_{r-1}$ had been determined it was possible to obtain λ_r and x_r by iterating with an arbitrary vector which had been orthogonalised with respect to the already determined x_i.

This leads naturally to a scheme in which the iterations are performed, not serially, but in parallel or simultaneously. Thus we let

$$X = [x_1, x_2, \ldots, x_p] \tag{17.1}$$

and ensure that

$$X^T X = I_p \ . \tag{17.2}$$

We then form the product

$$X' = AX \ , \tag{17.3}$$

reorthogonalise X' and repeat (17.3).

This is repeated until the process has converged.

We note that if

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_p| > |\lambda_{p+1}| \geq |\lambda_{p+2}| \geq \ldots$$
$$\ldots \geq |\lambda_n| \tag{17.4}$$

then convergence is guaranteed. Hence we may summarise the basic process as:

$$\left.\begin{array}{l} \text{i) Choose } X_0 \text{ such that } X_0^T X_0 = I \\ \text{ii) } Y_k := AX_{k-1} \\ \text{iii) } X_k := Y_k R_k^{-1} \text{ where } X_k^T X_k = I \end{array}\right\} \ k=1,2,\ldots \ . \tag{17.5}$$

In order to accelerate the convergence of this method we consider the projected eigenvalue equation:

$$B_k = X_k^T A X_k \ . \tag{17.6}$$

We note that B_k is a matrix of order p and we denote the eigensolution by

$$V_k^T B_k V_k = D_k \ . \tag{17.7}$$

If the first r columns of X_k are eigenvectors of A then the first r rows and r columns of B_k will

be zero except for the diagonal elements which will be equal to the first r eigenvalues of A. This leads to the following scheme for obtaining X_{k+1} from X_k:

i) Choose X_0 such that $X_0^T X_0 = I$

ii) $Y := AX_k$

iii) $B_k := X_k^T Y = X_k^T A X_k$

iv) Solve the eigenproblem for B_k. Thus, we compute V_k and D_k such that

$$V_k^T B_k V_k = D_k$$

v) $Y_k := X_k V_k$

vi) $X_{k+1} := Y_k R_k^{-1}$ where $X_{k+1}^T X_{k+1} = I$ (17.8)

and steps ii) to vi) are repeated for k=0,1,2,.... .

18. THE INTERMEDIATE STEPS

The iteration rule defined by (17.8) is extremely powerful but it is important to realise that, as we saw in section 16, the final space E_m produced by m steps of (17.8) is the same (apart from rounding errors) as that produced by m steps of (17.5). Hence there is nothing to be lost by using (17.5) m-1 times followed by one step of (17.8). A further saving in time is possible by replacing (17.5) with the even simpler rule

$$X_k = AX_{k-1} \qquad k = 1,2,\ldots . \qquad (18.1)$$

However, in this case, two further precautions are necessary. We have seen earlier that continued iteration using (18.1) makes the columns of X_k become more and more parallel. Thus we must always orthonormalise X_{k+m-1} after using (18.1) m-1 times and also we must limit m so that the columns of X_k can never start to become parallel.

In order to choose the correct value of m we note that if

$$(|\lambda_1|/|\lambda_p|)^{m-1} < 10 \qquad (18.2)$$

then the parallelisation of the columns of X_{k+m-1} will not have gone further than to cause the loss of at most one decimal digit when the columns are next orthonormalised. Obviously as λ_1 and λ_p are not known we take the current values of d_{11} and d_{pp} as approximations. At the beginning of the iteration d_{11} and d_{pp} are not known and in the first few steps they are still far from λ_1 and λ_p. In order to prevent m being too large we start therefore with m=2 and allow it to increase by one at each stage of the iteration. Thus we have the following computational scheme:

i) Let k=0, m=2

ii) Choose X_0 such that $X_0^T X_0 = I$

iii) Perform m-1 steps of (18.1) on X_k

iv) Orthonormalise the columns of X_{k+m-1}

v) Let $Y=AX_{k+m-1}$

vi) Let $B=X_{k+m-1}^T Y$

vii) Compute V and D such that $V^TBV=D$

viii) Let $X_{k+m}=X_{k+m-1}V$

ix) Orthonormalise the columns of X_{k+m}

x) Let k=k+m

xi) Test for termination

xii) If $(|d_{11}|/|d_{pp}|)^m < 10$ let m=m+1

xiii) Go to (iii) (18.3)

In practice we have found it advantageous to perform step (iii) between steps (viii) and (ix), omitting step (iv). This saves one orthonormalisation at each stage and our experience is that it does not impair accuracy or convergence.

19. TESTS FOR CONVERGENCE

Rutishauser (1969) has stated that "the most efficient computing process becomes doubtful, if it is not possible to determine the proper time for termination automatically".

In this algorithm it is relatively easy to determine the proper time for termination of the process. The diagonal elements of D in (18.3) (vii) will be changing throughout the iteration process.

Thus, as soon as the first diagonal element does not appear to change between successive stages of (18.3) it is determined to machine accuracy. We may then test for the second diagonal element and thence for the later ones.

This test is not sufficient for the eigenvectors. It is true to say however that if the eigenvalues have not converged to a given precision the eigenvectors will not have converged either. Thus we do not start testing for convergence of an eigenvector until after the corresponding eigenvalue has converged. We base our test for convergence of the vectors on examining the ∞-norm of each vector and testing whether this has altered between successive stages of (18.3).

In practice it is unlikely that we shall need to determine the eigensolutions to full machine accuracy and the tests are ammended to take this into account. Thus for the eigenvalues we test whether

$$\left| \frac{d'_{ii} - d_{ii}}{d'_{ii}} \right| \leqslant \varepsilon \quad \text{for } i=1,2,\ldots,p \tag{19.1}$$

where d'_{ii} is the current value of D_{ii}, d_{ii} was the value of D_{ii} at the previous stage and ε is the desired tolerance. Similarly for the eigenvectors we test whether

$$\left| \frac{\|x_i'\|_\infty - \|x_i\|_\infty}{\|x_i'\|_\infty} \right| \leqslant \varepsilon \quad \text{for } i=1,2,\ldots,p \tag{19.2}$$

where x_i is the i-th column of X. The p quantities on the left of inequality (19.2) form the components of the error vector described in chapter 5, section 15.

Once g eigenvectors have been accepted the active powering steps of (18.3) are no longer applied to these columns but they are retained for orthonormalisation of the columns g+1 to p.

20. THE HERMITIAN CASE

The extensions of the algorithm for the symmetric case needed to cover the Hermitian case are straightforward. We have already discussed the generalisations of the Jacobi and orthonormalisation algorithms and these are used in place of the symmetric and real versions. The only other alteration needed is to work with complex eigenvectors and Hermitian transposes. Thus the steps of (18.3) are replaced by

i) Let k=0, m=2

ii) Choose X_0 such that $X_0^H X_0 = I$

iii) Let $Y = AX_k$

iv) Let $B = X_k^H Y$

v) Compute V and D such that $V^H BV = D$

vi) Let $X_{k+1}=X_kV$

vii) Perform m-1 steps of (18.1) on X_k

viii) Orthonormalise the columns of X_{k+m}

ix) Let k=k+m

x) Test for termination

xi) If $(|d_{11}|/|d_{pp}|)^m < 10$ let m=m+1

xii) Go to (iii). (20.1)

21. EIGENVALUES OF EQUAL MODULUS

We saw in section 10 that it is not necessary for convergence that all the eigenvalues must be distinct. In fact, for both the real and Hermitian cases, if we are iterating with p vectors the only condition necessary for convergence is that

$$|\lambda_p| > |\lambda_{p+1}| \ . \tag{21.1}$$

It should be noted however that if $|\lambda_p|$ is close to $|\lambda_{p+1}|$ then convergence may be impracticably slow. In practice p should be chosen as far as possible so that

$$|\lambda_p| \ll |\lambda_k| \tag{21.2}$$

where k is the number of eigensolutions that it is desired to compute.

22. THE GENERAL CASE

We now turn our attention to the case of a

general matrix and as for the symmetric algorithm we return to the ideas of section 13. We shall assume initially that

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_p| > |\lambda_{p+1}| \geq \ldots \geq |\lambda_n| \quad (22.1)$$

and hence that there are p left-hand and p right-hand eigenvectors corresponding to the p dominant eigenvalues.

Having calculated the first r eigenvalues and eigenvectors we saw how to suppress the appropriate components of these in order to calculate the (r+1)-th eigensolution. Applying the process simultaneously to two sets of p vectors we may write the algorithm as

i) Choose Y_0 and X_0, both n*p, such that

$$Y_0^H X_0 = I$$

ii) Form $Y_k = A^H Y_{k-1}$, $X_k = A X_{k-1}$

iii) Biorthonormalise Y_k and X_k

$$\left.\begin{array}{l}\text{ii) Form } Y_k = A^H Y_{k-1},\ X_k = A X_{k-1} \\ \text{iii) Biorthonormalise } Y_k \text{ and } X_k\end{array}\right\} \quad k=1,2,\ldots . \quad (22.2)$$

To accelerate this method we consider the projected eigenvalue equation

$$B = Y_k^H A X_k \quad (22.3)$$

We denote the eigensolution of B by

$$T_L . B . T_R = D \quad (22.4)$$

where T_L and T_R are respectively the left-hand and right-hand eigenvectors of B and D is a diagonal matrix of the eigenvalues of B. Thus we are led

to the following accelerated scheme for determining Y_{k+1} and X_{k+1} from Y_k and X_k.

i) Choose Y_0 and X_0 such that $Y_0^H X_0 = I$

ii) Form $W = A^H Y_k$, $V = AX_k$

iii) Form $B = Y_k^H V$

iv) Solve the eigenproblem for B; that is compute T_L, T_R and D as in (22.4)

v) Form $Y_{k+1} = W\bar{T}_L$, $X_{k+1} = VT_R$

vi) Biorthonormalise such that $Y_{k+1}^H X_{k+1} = I$ (22.5)

and repeat steps ii) to vi) for k=0,1,.... .

Just as it was possible to use powering steps in the symmetric case so in the general case we may use a similar device to speed up the process. We replace steps (ii) and (iii) of (22.2) by the simpler rule:

$$Y_k = A^H Y_{k-1} \; ; \quad X_k = AX_{k-1} \qquad k=1,2,.... \; . \qquad (22.6)$$

As before we use m-1 steps of (22.6) followed by one step of (22.5). For the reasons outlined previously in section 18 it is essential that m is limited in order to prevent the columns of Y_k and X_k becoming less and less biorthogonal with respect to each other. This leads to a scheme very similar to that for the Hermitian case except that we are now working with two sets of

vectors.

i) Let $k=0$, $m=2$

ii) Choose Y_0 and X_0 such that $Y_0^H X_0 = I$

iii) Let $W=A^H Y_k$, $V=AX_k$

iv) Let $B=Y^H V$

v) Compute T_L, T_R and D such that

$T_L B T_R = D$

vi) Let $Y_{k+1}=W\bar{T}_L$, $X_{k+1}=VT_R$

vii) Perform m-1 steps of (22.6) on Y_k and X_k

viii) Biorthonormalise such that $Y_k^H X_k = I$

ix) Let $k=k+m$

x) Test for termination

xi) If $(|d_{g+1}|/|d_{pp}|)^m < 10$ let $m=m+1$

(g is the number of eigenvectors accepted thus far)

xii) Go to (iii). (22.7)

23. EQUAL EIGENVALUES

We know that an arbitrary matrix does not necessarily have a complete set of eigenvectors and hence the case of equal eigenvalues is more complicated than for Hermitian matrices. Suppose firstly that the matrix is non-defective but derogatory; we then have a situation similar to that of the Hermitian case and convergence is

possible if

$$|\lambda_p| > |\lambda_{p+1}| \ . \qquad (23.1)$$

If the matrix is defective (irrespective of whether or not it is derogatory) in as much as one or more of the first p eigenvectors do not exist there is no guarantee that the process will be convergent. However all our experimental results suggest that convergence does normally take place.

CHAPTER 5

COMPUTATIONAL DETAILS

1. INTRODUCTION

In this chapter we describe the implementation of the theory that we have discussed in the preceding chapters. We also explain the operation of the programs and give sufficient details to enable a prospective user to run them.

All the programs have been written in Fortran and developed on an I.C.L. 1904S machine using the XFAT compiler. They will run on all 1900 series machines but small modifications may be needed to enable them to be used with other compilers on different machines. In the descriptions of the programs that follow we list the non-standard functions and explain their purpose so that any necessary alterations may be made easily.

We begin in the next section by describing the general structure of the programs and then we give details of the Jacobi programs. This is followed by a description of the simultaneous iteration programs and finally we give details of some test runs.

2. PROGRAM STRUCTURE

All the programs are written in a similar manner in order to facilitate their use and comparison between them. The basic structure of the programs is as follows:

i) A steering segment.

ii) The main program segment (called the MASTER segment in 1900 Fortran).

iii) The main subroutine (which performs most of the calculation).

iv) Other subroutines

v) A final subroutine, altered by the user, used for the input of data.

The steering segment is used to allocate channel numbers to the peripherals. Following the usual Fortran convention we have used channel 5 as input from the card reader and channel 6 as output to the line printer. Channel 4 is used, if necessary, for additional monitoring information with channel 2 used for both input to and output from an internal array. This enables character conversions from text strings to integers (and vice versa) to be performed. Although machine dependent a facility similar to this is normally provided with most compilers.

The main program segment is used principally to print the headings and output the results; in particular no calculations pertinent to the problem are performed in it. This is because the writer is convinced that a reasonably neat presentation of the results is an important feature of any computer program. Implicit in this is the need to display the name of the program, the date and time when it was run, the example number and a

clear heading to each set of results. The absence of any calculations from the main segment also means that an intending user can see at a glance the form the output will take just from looking at the first segment. We have also assumed that if the line width of the output is 120 characters or more then it is to a line printer and the results are centralised on the page. If however the line width is less than 120 characters the results are all left-justified as this is particularly suitable for output to a teletype if the program is being run on-line.

The first subroutine normally performs the bulk of the calculation and it is here that the theory we have developed is implemented. Often repeated calculations such as the formation of inner-products are not performed here but are left to the following subroutines.

In all the programs the final subroutine is used to input or store information about the matrix. Since this will change from problem to problem it is up to the user to modify this routine for his particular problem.

3. A JACOBI PROGRAM FOR SYMMETRIC MATRICES - JACO

The program takes as data a real symmetric matrix A of order n and calculates its eigenvalues, λ_i, and optionally the eigenvectors, v_i. Also

printed are the number of rotations needed to diagonalise the matrix to a preset tolerance. Should the total number of sweeps exceed fifty the program is terminated and a warning message is printed. Additionally the program will, if requested, form the matrix V^TAV and normalise the vectors such that

$$\|v_i\|_\infty = 1 \quad i=1,2,\ldots,n \; . \qquad (3.1)$$

We now describe each routine.

Master segment

At the beginning of this segment the arrays are dimensioned. If it is desired to alter the size of these arrays this is the only point at which a change to the program has to be made. Five arrays are used and as far as possible their names have been chosen with reference to the theoretical discussion of earlier chapters.

A (n*n) - stores the elements a_{ij} of the original matrix A.

V (n*n) - stores if they are required the elements of the matrix V, the columns of which approximate the eigenvectors of A.

D (n) - stores the approximations to the eigenvalues λ_i.

B (n) - used as workspace.

Z (n) - used as workspace.

The value given to n must be the same for each of

the five arrays.

There is a call to an I.C.L. subroutine TIME(T). This returns in T the time of day as an eight bit character string. All the other subroutine calls are to segments appearing later in the program.

Subroutine Jaco

The coding follows exactly the computational details given in sections 10 and 11 of chapter 2 and we describe only the additional features not mentioned there. Firstly, as the matrix A is symmetric, only the upper half is used in the computation. This means that the information stored in the lower half can be used to recreate the original matrix. Secondly, in order to ensure the maximum possible accuracy in the eigenvalues the following device is adopted. During each sweep, as well as updating the vector D at each rotation, the updates are accumulated in a separate vector Z. At the end of each sweep the value in Z is then used to produce a fresh value of D. This is given by updating the value of D as it was at the end of the previous sweep (stored in B) by the current value of Z. Although this uses an extra 2n storage locations it does ensure great accuracy in the eigenvalues as very small individual increments in the elements of D are accumulated independently.

Having completed the diagonalisation process to the required tolerance the program uses the exchange sort algorithm to order the eigenvalues (and their corresponding vectors) such that their moduli are in non-increasing order.

Subroutine Normalisation

This subroutine, if called, normalises the eigenvectors as in (3.1).

Subroutine Check

As with Normalisation this subroutine is only called if specifically requested by the user. It uses the calculated values of the eigenvectors to form the product V^TAV storing the result in A. All the summations are performed in double precision.

Subroutine Elapse

This subroutine, which is used in all the programs we have written, performs three small calculations needed by other routines. Firstly, it calculates the mill time that the program has used. This involves using the I.C.L. subroutine MTIME(N) which gives N as the number of milli-seconds so far used by the program. This is converted to minutes and seconds and output at the end of the program. Secondly, Elapse uses the machine routine DATE(D) to obtain in D an eight character text string containing the date. In order to print the month as a three letter abbreviation we call DEFBUF. This routine, combined

with the use of channel 2, enables us to access an internal array called BUFFER and perform the necessary character conversion.

Finally the subroutine calculates the machine constant. That is, the smallest number m such that

$$1.0 + m \neq 1.0 \; . \qquad (3.2)$$

This means that there is no need for the user to provide this information for each different machine.

Subroutine Mxop

Mxop has been developed as a general purpose routine for the outputting of matrices. The input parameters for the subroutine are the matrix A, its actual dimensions as defined in the Master segment, the size of matrix it is desired to output, the format required for an individual element, the number of printing positions per line, the logical stream number for the output channel and finally a flag. If the flag has a negative value the output is left-justified; if flag is positive the output is centralised on the page. If flag is zero Mxop is not initiallised and the output is then in the same format as on the previous occasion. It is recommended that flag is set to zero if the same format is being used a number of times. This saves reinitialising Mxop with a consequent saving of time.

The form of the output from Mxop consists of a heading to indicate which columns of the matrix are to be printed followed by those columns. The program automatically prints as many columns as the width of line allows. For example, suppose a (20*15) matrix is to be output centrally on a 120 character line with each element printed under the format F16.8. The output from Mxop would take the form:

COLUMNS 1 TO 7 ARE:

$$\begin{array}{ccccccc} a_{11} & a_{12} & a_{13} & \cdots & a_{17} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{20,1} & a_{20,2} & a_{20,3} & \cdots & a_{20,7} \end{array}$$

COLUMNS 8 TO 14 ARE:

$$\begin{array}{ccccccc} a_{18} & a_{19} & a_{1,10} & \cdots & a_{1,14} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{20,8} & a_{20,9} & a_{20,10} & \cdots & a_{20,14} \end{array}$$

COLUMN 15 IS:

$$\begin{array}{c} a_{1,15} \\ \vdots \\ a_{20,15} \end{array}$$

There are no restrictions on the size of matrix Mxop can handle including (n*1) and (1*n); in the case of (1*n) matrices however the column headings are suppressed. This subroutine has proved invaluable in the programs we have written

and also during the development stage of many other routines.

Subroutine Input

This routine is used to input the data for each example and it is the user's responsibility to write the appropriate sections of code. A computed GO TO statement transfers control on the n-th example to the CONTINUE statement numbered n*100. The user then inserts an appropriate section of code to set the elements of the array A(I,J) equal to the elements (a_{ij}) of the matrix whose eigensolution it is desired to calculate. This may be done either by generating the elements of A or by reading them from data cards. Finally, after inputting A, control must pass to a RETURN statement in order to leave the subroutine.

This concludes the description of each section of the program.

4. DATA REQUIRED BY JACO

The data required by the program is very simple and takes the following form.

i) The first data card (read from the Master segment) must contain three integers punched in the format (2I2,I3). The first of these is the number of examples to be run, the second is the size of the arrays as dimensioned in the Master segment and the third is the width of output line

required.

ii) The second data card contains information about the first example and this is given in the format (I2,I1,A6,2I1). The first number is the dimension of the matrix A for this particular example; this must be less than or equal to the dimension of the array A in the Master segment. The second number is 1 or 0 depending on whether or not it is desired to calculate the eigenvectors. The third is a text string defining the output format required. Typical values might be F16.8 or E20.10. The last two numbers are flags and if nothing is punched these are taken as zero. If however a 1 is punched in the first of these two positions then subroutine Check is called; a 1 in the second position calls subroutine Normalisation.

iii) If the elements of A are to be read from cards these should now follow. The format is obviously dependent upon what the user has specified in subroutine Input.

iv) After the data cards of iii), or if there are none, a second card as in ii) should follow for the second example. A similar pattern now follows for the subsequent examples.

5. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is

to be found in appendix 1 and we give here some comments on the test runs. Examples 1 to 7 are all taken from Gregory and Karney (1969) and example 8 is due to Rutishauser (1966) and quoted in Wilkinson and Reinsch (1971). These two excellent books have provided the author with much inspiration. For convenience we shall refer to Gregory and Karney as GK and list the example number in their book.

Example 1, GK 4.1, is a 4*4 matrix typical of the type that occurs within simultaneous iteration. The eigenvalues and eigenvectors were obtained to 10 significant figures after 10 rotations in 0.042 seconds.

Example 2, GK 4.2, is a 4*4 matrix with a repeated eigenvalue. Again the eigensolutions were obtained to 10 significant figures, this time after 17 rotations in 0.057 seconds.

Example 3, GK 4.15 taking n=10, is a matrix of order 10. The first eight eigenvalues were computed to 10 significant figures and the last two to 9 significant figures with a similar accuracy in the eigenvectors. 120 rotations were needed taking 0.644 seconds.

Example 4, GK 4.13 taking n=10, is the Hilbert matrix of order 10. The computed eigensolutions are all correct to 10 decimal places. The execution time was 1.128 seconds in which 231 rotations were

performed.

Example 5, GK 4.20 taking n=10 and a=3, needed 234 rotations taking 1.125 seconds to produce eigensolutions to 9 decimal places.

Example 6, GK 4.20 taking n=10 and a=0, needed 89 rotations taking 0.512 seconds to produce eigensolutions to 9 decimal places.

Example 7, GK 4.10, is the Rosser matrix. 125 rotations in 0.546 seconds produced the eigensolutions to 7 decimal places.

Example 8, Rutishauser (1966), is a matrix of order 44. Our program used 6280 rotations in 1 minute 39.8 seconds. The eigensolutions are all correct to 9 decimal places and compare favourably with Rutishauser's results.

6. A JACOBI PROGRAM FOR HERMITIAN MATRICES - HMJO

The program takes as data a Hermitian matrix A of order n and calculates its eigenvalues, λ_i, and optionally the eigenvectors, v_i. Also printed are the number of rotations needed to diagonalise the matrix to a preset tolerance. Should the total number of sweeps exceed fifty the program is terminated and a warning message printed. Additionally the program will, if requested, form the matrix $V^H AV$ and normalise the vectors such that

$$\|v_i\|_\infty = 1 \qquad i=1,2,\ldots,n \; . \qquad (6.1)$$

The routines are very similar to those for the real symmetric case and we describe only the differences between the two.

Master segment

Throughout our programs we have used only real arithmetic and since the original matrix A and its matrix of eigenvectors V may be complex two additional arrays are needed now for storing A and V. In our programming we have, without exception, used "R" and "I" as the last letter of an array name to indicate that that array contains either real or imaginary components. Thus, instead of a complex array A, we use two real arrays AR and AI.

AR (n*n) - stores the real components $Re(a_{ij})$ of the original matrix A.

AI (n*n) - stores the imaginary components $Im(a_{ij})$ of the original matrix A.

VR (n*n) - stores the real part of the matrix V, whose columns approximate the eigenvectors of A.

VI (n*n) - stores the imaginary part of the matrix V.

Subroutine Hmjo

The coding follows the computational details given in section 12 of chapter 2. The same device employed in the real symmetric case is used to ensure maximum accuracy in the eigenvalues.

Other subroutines

The other subroutines are identical to those described in section 3 except for the slight modifications needed for complex working in Normalisation, Check and Input.

7. DATA REQUIRED BY HMJO

The data is input in exactly the same form as that described in section 4. It is the user's responsibility to ensure that the real and imaginary parts of A are correctly input.

8. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is to be found in appendix 2.

Example 1, GK 6.6, is a 4*4 matrix typical of the type that occurs within simultaneous iteration. The eigensolutions were obtained to 10 significant figures after 13 rotations in 0.090 seconds.

Example 2, GK 6.7, is a 4*4 matrix with a repeated eigenvalue. Again the eigensolutions were obtained to 10 significant figures after 17 rotations in 0.115 seconds.

Example 3, GK 6.8, is a matrix of order 5 and 46 rotations in 0.327 seconds produced the eigensolutions to 9 decimal places.

Examples 4 to 9, GK 7.10 taking n=6 to 11 in example

B, have only imaginary components. All the solutions were produced to 9 decimal places. The number of rotations needed and the time taken were 101 (0.861), 131 (1.245), 178 (1.843), 220 (2.534), 305 (3.741) and 319 (4.212).

9. A JACOBI PROGRAM FOR NORMAL MATRICES - NMJO

The program takes as data a normal matrix A of order n and calculates its eigenvalues, λ_i, and optionally the eigenvectors, v_i. Also printed are the number of rotations needed to diagonalise the matrix to a preset tolerance. Should the total number of sweeps exceed fifty the program is terminated and a warning message is printed. Additionally the program will, if requested, form the matrix $V^H AV$ and normalise the vectors such that

$$\|v_i\|_\infty = 1 \qquad i=1,2,\ldots,n \ . \tag{9.1}$$

Again we describe only the differences between this program and the previous one.

Master segment

As well as the original matrix A and the eigenvectors V being complex the eigenvalues may be too. This means that the arrays D, B and Z must each be replaced by two arrays. In addition, since the whole matrix is acted upon in the program, it becomes necessary to keep a copy of

the original matrix. Thus we have the following additional arrays.

AAR (n*n), AAI (n*n) - store a copy of the original values of AR and AI.

DR (n), DI (n) - store respectively the approximations to the real and imaginary parts of the eigenvalues.

YR (n), YI (n) - used as workspace.

ZR (n), ZI (n) - used as workspace.

The change of array name from B to Y is to avoid any possible confusion with the theory as given in section 13 of chapter 2.

Other subroutines

The coding for subroutine Nmjo follows exactly the computational details given in section 13 of chapter 2. The other subroutines are unaltered from the Hermitian program.

10. DATA REQUIRED BY NMJO

The data is input exactly as in section 7.

11. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is to be found in appendix 3.

Examples 1 and 2, are both 4*4 matrices constructed for test purposes. 23 rotations in 0.2 seconds were needed in both cases to produce solutions

to 9 decimal places.

12. A JACOBI PROGRAM FOR GENERAL MATRICES - GLJO

The program takes as data an arbitrary complex matrix A of order n and calculates its eigenvalues, λ_i, and the left and right eigenvectors, w_i and v_i respectively. Also printed are the number of rotations needed to diagonalise the matrix to a preset tolerance. Should the total number of sweeps exceed fifty the program is terminated and a warning message is printed. The routines are now described.

Master segment

As with the other Jacobi programs it is at the beginning of this segment that the arrays are dimensioned and it is here that any change to their size must be made. The arrays used are as follows.

AR, AI (n*n) - store respectively the real and imaginary components of the original matrix A.

AAR, AAI (n*n) - store a copy of the original matrix.

WR, WI (n*n) - store the elements of the matrix whose columns approximate the left eigenvectors of A.

VR, VI (n*n) - store the elements of the matrix whose columns approximate the right eigenvectors of A.

DR, DI (n) - store the approximations to the eigenvalues λ_i.

EN, EM (n) - used as workspace.

The value given to n must be the same for each of the ~~five~~ twelve arrays.

Subroutine Gljo

The coding follows that given by Eberlein (1970). The only difference is that described in section 20 of chapter 2. Thus no ordering of the eigenvalues takes place during the execution of this routine until immediately before returning to the Master segment to output the results. The ordering is such that on output,

$$|\lambda_1| \geq |\lambda_2| \geq \ldots\ldots \geq |\lambda_n| \ . \qquad (12.1)$$

Subroutine Check

This routine, if called by the user, uses the calculated values of the left and right eigenvectors to form the product W^TAV, where W and V are the matrices whose columns are given by w_i and v_i respectively. All the summations are performed in double precision and the resulting product is stored in A.

Other subroutines

These are as previously described.

13. DATA REQUIRED BY GLJO

The form of the data is very similar to that

described in section 4. The difference occurs in (ii) where the data card is in the format (I2,A6,I1). The first number is the dimension of the matrix A for the particular example; the next is a text string defining the output format required and the third is a flag to call subroutine check.

The real and imaginary elements of A must be input separately and this is the user's responsibility.

14. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is to be found in appendix 4.

Example 1, GK 5.1, is a 3*3 real non-symmetric matrix with real eigenvalues. 33 rotations in 0.366 seconds produced the eigenvalues and the left-hand and right-hand eigenvectors to 9 decimal places.

Example 2, GK 5.3, is also a real non-symmetric matrix of order 3. The eigensolutions were obtained to 8 decimal places after 21 rotations in 0.237 seconds.

Example 3, GK 5.5, is a real non-symmetric matrix of order 4 with real eigenvalues. The solutions, correct to 8 decimal places, were produced in 0.557 seconds after 42 rotations.

Example 4, GK 5.8, is a real non-symmetric matrix

of order 4 with two real eigenvalues and a complex conjugate pair. 18 rotations in 0.246 seconds produced the eigensolutions correct to 9 decimal places.

Example 5, GK 5.2, is a 3*3 real defective matrix. After 60 rotations taking 0.511 seconds the isolated eigensolution was obtained to 9 decimal places with the repeated eigenvalues and their associated vector being obtained to 5 decimal places.

Example 6, GK 6.5, is a complex matrix of order 4 and the results were obtained to 8 decimal places after 54 rotations taking 0.753 seconds.

Example 7, GK 6.4, is a complex matrix of order 3 and the results were obtained to 8 decimal places after 21 rotations taking 0.234 seconds.

Example 8, Eberlein (1970) example I, is a real matrix of order 7 with one real eigenvalue and the rest complex conjugate pairs. 126 rotations in 2.714 seconds produced the results to 10 decimal places and this compares favourably with Eberlein's results.

Example 9, Eberlein (1970) example II, is a real matrix of order 5 which is defective. 240 rotations taking 3.052 seconds produced the isolated real eigenvalue and its corresponding vector to 10 decimal places. The complex conjugate eigenvalues and their corresponding vectors were obtained to 5 decimal places.

15. A RITZ ITERATION PROGRAM FOR SYMMETRIC MATRICES - QKRZ

For a real symmetric matrix A of order n the program calculates the em absolutely largest eigenvalues, λ_i, and their corresponding eigenvectors, x_i. There is no need to store the matrix A; all that is required is to be able to form the vector w where

$$w = Av \tag{15.1}$$

without altering v. The iteration is carried out with p vectors where $p \geq em$ and usually $p > em$. The optimum value of p will depend on the distribution of the eigenvalues but if no knowledge of this is available a reasonably good rule of thumb is to choose p equal to em plus 2 or 3 for smallish values of em. The eigensolutions are obtained to the tolerance requested and optionally the eigenvectors may be normalised such that

$$\|x_i\|_\infty = 1 , \qquad i=1,2,\ldots,p . \tag{15.2}$$

We now describe the program.

Master segment

All the dimensioning of the arrays is performed at the beginning of this segment and this is the only point at which any change may need to be made. The following arrays are used and as far as possible their names have been chosen with reference to the earlier theoretical

discussion.

X (n*p) - stores the matrix X whose columns approximate the eigenvectors.

V (n), W (n) - used as workspace in forming the product.

RV (p*p), B (p*p) - used as workspace.

D (p) - stores the approximations to the eigenvalues λ_i.

F (p) - stores the error vector as described in chapter 4, section 19.

BB (p), Z (p), DOLD (p), LARGE (p) - used as workspace.

The values given to n and p respectively must be the same for all the arrays.

There then follow three READ statements which input the data required by the program. These statements refer to the FORMAT statements labelled 10, 20 and 30.

There is a call to an I.C.L. subroutine TIME(T). This returns in T the time of day as an eight bit character string. All the other subroutine calls are to segments appearing later in the program.

Subroutine Qkrz

The coding follows exactly the computational details given in sections 17 to 19 of chapter 4 and we describe only the additional features not mentioned there. After the tests for convergence

there is a call to subroutine INFO. This outputs, on channel 4, intermediate information concerning the progress of the program. We describe INFO fully below. A test is also made to see whether it is likely that the required number of solutions will be obtained in the next few steps. If this is the case the value of m (the number of pre-multiplications to be performed) is reduced to 2. Finally a check is made to ensure termination in the event of the number of iterations having exceeded some preset value.

Subroutine Jaco

This is exactly as described in section 3.

Subroutine Ortho

This routine uses the modified Gram-Schmidt process with reinforcement to orthogonalise a set of vectors X(n*p) with $n \geq p$. For the sake of generality the routine assumes that the first F columns of X are already orthonormalised but for our purposes we always take F equal to zero. The coding follows equations (3.1) of chapter 3 with the reinforcement as described in section 4 of the same chapter. If reinforcement is found to be necessary the message "Warning 1 in ortho" is output on channel 4; if a column of X is found to be linearly dependent and is thus set identically equal to zero as in equation (4.7) of chapter 3 the message "Warning 2 in ortho" is

output.

Real Function Inner Product

This routine is called by Ortho and calculates the inner-product of columns k and l of the vectors held in X. The summations are performed in double precision arithmetic.

Subroutine Randomisation

Randomisation places pseudo random numbers in the range (-1,1) into the columns of X; these then form the starting vectors for the iteration. If approximations to the eigenvectors are already known a facility exists for using these. This is detailed in the next section. The random numbers are obtained from the I.C.L. routine FPMCRV which generates pseudo random numbers in the range (0,1). Most machines have a similar facility to this and there should be no difficulty in making the necessary alteration.

Subroutine Info

We have briefly mentioned this routine which outputs, after each test for convergence, sufficient information for the user to see the progress, or otherwise, of his program. The subroutine is split into two halves; both output the same information but the first does this in display form suitable for a line printer whilst the second is more compact and suitable for a teletype. Whichever form the output takes

the information displayed is as follows:

Number of steps performed,

Number of eigenvectors accepted,

Number of eigenvalues accepted,

Number of solutions to be computed,

The error vector,

The present approximations to the eigenvalues.

Clearly the user could alter this routine if he felt more or less information was required but we have found the above to be most useful.

Subroutine Normalisation

This routine, if called by the user, normalises the eigenvectors $\{x_i\}$ such that

$$\|x_i\|_\infty = 1 , \qquad i=1,2,\ldots,p . \qquad (15.3)$$

Subroutines Elapse and Mxop

These are exactly as described in section 3.

Subroutine Product

Product performs a similar role to subroutine Input in the Jacobi programs. A computed GO TO statement transfers control on the n-th example to the CONTINUE statement numbered n*100. The user must then insert a section of code which is such that it computes

$$w = Av, \qquad (15.4)$$

where w and v are both vectors of n components, without altering v. Finally, after computing w, control must pass to a RETURN statement in order to leave the subroutine.

This concludes the description of each section of the program.

16. DATA REQUIRED BY QKRZ

The data required by the program takes the following form.

i) The first data card (read from the Master segment at the format statement numbered 10) must contain four integers punched in the format (I2,I5,I2,I3). The first of these is the number of examples to be run, the second and third are the values of N and P respectively used to dimension the arrays in the Master segment and the fourth is the width of output line required.

ii) The second data card contains information about the first example and this is given in the format (I5,I2,I6,I2,E9.2,A6,I1). The first two numbers are the dimensions of the array X for this particular example; these must be less than or equal to the dimensions given to X in the Master segment as input in (i). The third number is the maximum number of iterative steps that it is wished to perform. If there has been no convergence after this number the program is automatically terminated. The fourth parameter is the number of solutions it is desired to compute and the fifth the accuracy to which they are required. The sixth parameter is

a text string defining the output format required, a typical value might be E18.8. Finally we have a flag which, if set to one, normalises the vectors as in (15.1).

iii) If in advance approximations to the eigenvectors are known a considerable saving in computer time can be effected by starting the iteration with these approximations instead of using random numbers. For this reason the following facility is available. If on the second data card described in (ii) the third parameter, the maximum number of steps to be performed, is given a negative value then it is assumed that initial approximations to the eigenvectors are to be input and randomisation is suppressed. The input is controlled by the format statement labelled 30 and the IF statement immediately preceding it. The user is advised to alter the format statement to suit his particular needs. Although the maximum number of steps has been given a negative value the upper limit is set as the absolute value of that read in.

iv) After the data card described in (ii) and any possible cards for (iii) a second data card as described in (ii) should now follow for the second example. A similar pattern is followed for each subsequent example.

17. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is to be found in appendix 5. In all the examples which follow we requested the solutions to an accuracy of 10^{-8}.

Example 1, GK 4.13, is the Hilbert matrix of order 10. The iteration was performed with four trial vectors and requested two eigensolutions. In 6 steps taking 1.921 seconds the program had accepted 3 eigenvalues and 3 eigenvectors.

Example 2, GK 4.15 taking n=10, was described in section 5. We used four trial vectors and requested two eigensolutions. 8 steps taking 1.275 seconds in fact produced 4 eigensolutions.

Example 3, is a diagonal matrix included to check consistency of the program.

Example 4, GK 7.2, is a tridiagonal matrix of order 21. Five trial vectors produced 3 eigensolutions after 75 steps taking 8.275 seconds. It is worth noting that as the eigenvalues are poorly separated the value of m is increasing throughout the program; that is, an increasing number of intermediate premultiplications are performed at each stage.

Example 5, GK 7.3, is again a tridiagonal matrix of order 21. Five trial vectors produced 4 eigenvalues and 2 eigenvectors after 67 steps in 12.289

seconds. Again the value of m is increasing throughout.

Example 6, GK 7.4 taking a=0, b=1 and n=20, is a tri-diagonal matrix of order 20 with very poorly separated eigenvalues. After 132 steps taking 17.911 seconds and iterating with six trial vectors 4 eigenvalues and 3 eigenvectors were obtained. As the eigenvalues are so close the value of m is constantly increasing.

Example 7, GK 7.10 taking n=19 with $x_k=y_k=$ $=[k(n-k+1)]^{\frac{1}{2}}$, is a tridiagonal matrix of order 20. Using six trial vectors 4 eigenvalues and 2 eigenvectors were obtained after 64 steps in 15.022 seconds. In this example the eigenvalues are well separated and m increases only slowly.

Example 8, Rutishauser (1966), is the matrix of order 44 described in section 5. Four trial vectors produced 3 eigenvalues and 2 eigenvectors after 189 steps taking 21.983 seconds. With the very close eigenvalues m increases rapidly.

18. A RITZ ITERATION PROGRAM FOR HERMITIAN MATRICES - HMRZ

For a Hermitian matrix of order n the program calculates the em absolutely largest eigenvalues, λ_i, and their corresponding eigenvectors, x_i. It is not necessary to store the matrix A, all that is required is to be able to form the vector w

where

$$w = Av \qquad (18.1)$$

without altering v. The iteration is carried out with p vectors where $p \geq em$ and usually $p > em$. The eigensolutions are obtained to the requested tolerance and optionally the eigenvectors may be normalised such that

$$\|x_i\|_\infty = 1 , \qquad i=1,2,\ldots,p . \qquad (18.2)$$

The program is similar to the symmetric case and we concentrate our description on the differences.

Master segment

As before the arrays are dimensioned at the beginning of this segment but as the eigenvectors may be complex most arrays now have to be doubled to hold the real and imaginary parts.

XR, XI (n*p) - store the real and imaginary parts of the matrix X whose columns approximate the eigenvectors.

VR (n), VI (n), WR (n), WI (n) - used as workspace in forming the product.

RVR (p*p), RVI (p*p), BR (p*p), BI (p*p) - used as workspace.

D (p) - stores the approximations to the eigenvalues λ_i.

F (p) - stores the error vector.

BB (p), Z (p), DOLD (p), LARGE (p) - used as workspace.

The values given to n and p respectively must be the same for all the arrays.

Subroutine Hmrz

This is identical to Qkrz except for the complex arithmetic which is performed by working separately with the real and imaginary parts of the variables.

Subroutine Hmjo

This is exactly as described in section 6.

Subroutine Cortho

This is a complex arithmetic version of Ortho. It produces exactly the same warning messages.

Subroutine Hinp

This subroutine replaces the function Inner Product in the symmetric case. It calculates, using double precision, the Hermitian inner-product of the columns k and l of the vectors held in X.

Subroutine Cabs

We now describe three routines not needed in the real symmetric case. The first of these is CABS which calculates the absolute value of a complex number, returning this as a real variable with the original number unaltered.

Subroutine Cdiv

This divides one complex number by another returning the result as a third parameter. Both of the original numbers are unaltered.

Subroutine Csqrt

Given a complex number CSQRT calculates its square root with argument lying in $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$ and returns this as a second parameter leaving the original number unaltered.

Subroutine Randomisation

This is as previously described and in the Hermitian case random numbers are only put in the real part of the array X. This ensures that if the vectors are all real no unnecessary imaginary components will be introduced.

Subroutines Info, Normalisation, Elapse and Mxop

These are as previously described save for the adaption of Normalisation to complex arithmetic.

Subroutine Product

Product is as described before but the user must remember that the vectors w and v now are represented each by two arrays WR, WI and VR, VI respectively and that the matrix product

$$W = AV \tag{18.3}$$

must be programmed as

$$\begin{aligned} WR &= Re(A).VR - Im(A).VI \\ WI &= Re(A).VI + Im(A).VR \ . \end{aligned} \tag{18.4}$$

19. DATA REQUIRED BY HMRZ

The form of the data required by HMRZ does not differ from that described in section 16.

20. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is to be found in appendix 6. In all the examples which follow we requested the solutions to an accuracy of 10^{-8}.

Example 1, GK 4.13, is the Hilbert matrix of order 10. The Hermitian program produced identical results to the symmetric program.

Examples 2 and 3, GK 6.6 and 6.7, are both small matrices but three trial vectors produced 3 eigensolutions in 6 steps in 1.3 seconds.

Example 4, GK 7.10 taking n=11 in example B, is a tridiagonal matrix of order 12. Four trial vectors produced 2 eigensolutions after 50 steps taking 10.341 seconds.

Example 5, GK 7.10 taking n=19 in example B, is a tridiagonal matrix of order 20. As in example 4 we used four trial vectors to produce 2 eigensolutions after 101 steps taking 23.660 seconds. In both these examples m increases slowly throughout.

21. A RITZ ITERATION PROGRAM FOR GENERAL MATRICES - RITZ

For a general matrix A of order n the program calculates the em absolutely largest eigenvalues, λ_i, and their corresponding left and right

eigenvectors, y_i and x_i respectively. It is not necessary to store the matrix A; all that is required is to be able to form the vector w where

$$w = Av$$

or

$$w = A^H v \qquad (21.1)$$

without altering v. The iteration is carried out with p vectors where $p \geq em$ and usually $p > em$. The eigensolutions are obtained to the requested tolerance and optionally both sets of eigenvectors may be normalised such that

$$\|y_i\|_\infty = \|x_i\|_\infty = 1 , \quad i=1,2,....,p . \qquad (21.2)$$

The program is related to the Hermitian case but appears more complex by the need to introduce left and right vectors.

Master segment

We list the arrays dimensioned in this segment.

YR (n*p), YI (n*p) - store the real and imaginary parts of the matrix Y, the columns of which approximate the left eigenvectors.

XR (n*p), XI (n*p) - store the real and imaginary parts of the matrix X, the columns of which approximate the right eigenvectors.

VR (n), VI (n), WR (n), WI (n) - used as workspace in forming the product.

LVR, LVI, RVR, RVI, BR, BI (p*p) - used as workspace.

DR (p), DI (p) - store the approximations to the eigenvalues λ_i.

LF (p), RF (p) - store the left and right error vectors.

EN (p), DOLD (p,2), LARGE (p,2) - used as workspace.

The values given to n and p respectively must be the same for all the arrays.

Subroutine Glrz

The coding follows exactly the computational details given in section 22 of chapter 4. The need to test the left and right vectors is the only real difference between this program and the Hermitian case.

Subroutine Gljo

This is exactly as described in section 12.

Subroutine Biortho

This routine uses the modified biortho-normalisation process with reinforcement to biorthonormalise two sets of vectors Y and X. The coding follows sections 7 and 8 of chapter 3. If reinforcement is found to be necessary the message "Warning 1 in biortho" is output on channel 4; if it is found that it is not possible to biorthonormalise at a particular stage the message output is "Warning 2 in biortho".

Subroutine Cinp

This subroutine calculates, in double

precision, the inner-product of two complex vectors y_k and x_l.

Other subroutines except Product

These are as described in Hmrz.

Subroutine Product

This is of a similar pattern to that in Hmrz but as well as being able to form the matrix product

$$w = Av \tag{21.3}$$

it is necessary to be able to form

$$w = A^H v \ . \tag{21.4}$$

These must be programmed as

$$\begin{aligned} WR &= Re(A).VR - Im(A).VI \\ WI &= Re(A).VI + Im(A).VR \end{aligned} \tag{21.5}$$

and

$$\begin{aligned} WR &= Re(A^H).VR + Im(A^H).VI \\ WI &= Re(A^H).VI - Im(A^H).VR \ . \end{aligned} \tag{21.6}$$

An additional parameter, Hermit, is passed to the subroutine to determine which product is required. If Hermit is negative (21.1) should be formed but if it is positive (21.2) is needed.

22. DATA REQUIRED BY RITZ

The form of the data required by Ritz is basically the same as for Qkrz as described in section 16. However, the second data card is in the format (I5,I2,I6,I2,E9.2,A6,2I1). The first six parameters are exactly as described previously

and the eigth is the normalisation flag, previously the seventh parameter. The only addition, the seventh parameter, is an integer used to control the computed GO TO statement in subroutine Product. It has been found advantageous for our purposes to introduce this additional parameter but a user may well wish to revert to the same pattern employed in the symmetric and Hermitian programs.

23. TEST RESULTS

A listing of the program together with a sample of the output from one of the test runs is to be found in appendix 7. In all the examples which follow we requested the solutions to an accuracy of 10^{-8}.

Example 1, GK 5.5, is a small matrix but nevertheless is a useful test. Three trial vectors produced 3 eigensolutions after 36 steps in 7.828 seconds.

Example 2, GK 5.8, has a complex conjugate pair of eigenvalues. Three trial vectors produced 3 eigensolutions after 44 steps in 7.242 seconds.

Example 3, GK 6.5, has four well separated complex eigenvalues. Three trial vectors produced 3 eigenvalues and 2 eigenvectors after 68 steps in 10.695 seconds.

Example 4, GK 6.10, is a Hessenberg matrix of order 10. We used four trial vectors to produce

4 eigenvalues and 3 eigenvectors in 1 minute 40.3 seconds after 128 steps.

Example 5, Clint and Jennings (1971) Table 4, is a Hessenberg matrix of order 7. We used four trial vectors and in 41 steps computed 3 eigenvalues and 2 eigenvectors. The execution time was 23.376 seconds. Our results compare well with those given by Clint and Jennings.

Example 6, GK 5.26, has as its dominant values a repeated eigenvalue with linear elementary divisors and a pair of complex conjugate eigenvalues. Iteration with four trial vectors determined the 4 eigenvalues and 2 eigenvectors after 29 steps taking 14.790 seconds.

Example 7, GK 5.23 taking $\varepsilon=10^{-10}$, is a sparse matrix of order 20. Using six trial vectors we obtained 5 eigenvalues and 2 eigenvectors after 123 steps taking 1 minute 45.5 seconds.

Example 8, GK 7.10 taking n=19 in example A, is a sparse matrix of the type particularly suited to simultaneous iteration. We used six trial vectors and obtained 4 eigenvalues and 2 eigenvectors in 1 minute 3.3 seconds after 70 steps.

CONCLUSION

CONCLUSION

We have written a program to find the dominant eigenvalues and the corresponding left-hand and right-hand eigenvectors of an arbitrary complex matrix using simultaneous iteration.

In general simultaneous iteration produces results accurately and with great reliability, although running time can be high. The method really comes into its own for large sparse matrices having no particular pattern.

In his program for symmetric matrices Rutishauser (1970) uses Chebyshev iteration in the premultiplication steps. The inclusion of some device similar to this would probably enhance our program for general matrices, particularly in the case of poorly separated eigenvalues.

It certainly appears that there is a need for a simultaneous iteration program for arbitrary matrices and the author hopes that his program may prove its worth (or otherwise) through practical application.

BIBLIOGRAPHY

BIBLIOGRAPHY

Bauer F.L. 1957, "Das Verfahren der Treppeniteration und verwandte Verfahren zur Losung algebraischer Eigenwertprobleme", Z. Angew. Math. Phys., 8, 214-235.

Björck A. 1967, "Solving linear least squares problems by Gram-Schmidt orthogonalisation", BIT, 7, 1-21.

Causey R.L. 1958, "Computing eigenvalues of non-Hermitian matrices by methods of Jacobi type", J. SIAM, 6, 172-181.

Clint M. & Jennings A. 1971, "A simultaneous iteration method for the unsymmetric eigenvalue problem", J. Inst. Maths. Applics., 8, 111-121.

Eberlein P.J. 1962, "A Jacobi-like method for the automatic computation of eigenvalues and eigenvectors of an arbitrary matrix", J. SIAM, 10, 74-88.

Eberlein P.J. 1970, "Solution to the complex eigenproblem by a norm reducing Jacobi type method", Numer. Math., 14, 232-245.

Francis J.G.F. 1961, 1962, "The QR transformation, Parts I and II", Comp. J., 4, 265-271, 332-345.

Goldstine H.H. & Horwitz L.P. 1959, "A procedure for the diagonalisation of normal matrices", J. ACM, 6, 176-195.

Greenstadt J. 1955, "A method for finding roots

of arbitrary matrices", Math. Tab. Aid. Comp., 9, 47-52.

Gregory R.T. 1953, "Computing eigenvalues and eigenvectors", Math. Tab. Aid. Comp., 7, 215-220.

Gregory R.T. 1960, "Defective and derogatory matrices", SIAM Rev., 2, 134-139.

Gregory R.T. & Karney D.L. 1969, "A collection of matrices for testing computational algorithms", Wiley-Interscience, New York.

Gudgin B.C. 1971, "Simultaneous iteration techniques for large sparse matrices", University of London M.Sc. dissertation.

Hotelling H. 1933, "Analysis of a complex of statistical variables into principal components", J. educ. Psychol., 24, 417-441, 498-520.

Jacobi C.G.J. 1846, "Uber ein leichtes Verfahren die in der Theorie der Sacularstorungen vorkommenden Gleichungen numerisch aufzulosen", Crelle's J., 30, 51-94.

Loizou G. 1972, "On the quadratic convergence of the Jacobi method for normal matrices", Comp. J., 15, 274-276.

Lotkin M. 1956, "Characteristic values of arbitrary matrices", Quart. Appl. Math., 14, 267-275.

Mirsky L. 1958, "On the minimisation of matrix norms", Amer. Math. Mon., 65, 106-107.

Noble B. 1969, "Applied linear algebra", Prentice-Hall, New Jersey.

Pope D.A. & Tompkins C. 1957, "Maximising functions of rotations - Experiments concerning speed of diagonalisation of symmetric matrices using Jacobi's method", J. Ass. Comp. Mach., 4, 459-466.

Rice J.R. 1966, "Experiments on Gram-Schmidt orthogonalisation", Maths. of Comp., 20, 325-328.

Ruhe A. 1967, "On the quadratic convergence of the Jacobi method for normal matrices", BIT, 7, 305-313.

Ruhe A. 1968, "On the quadratic convergence of a generalisation of the Jacobi method to arbitrary matrices", BIT, 8, 210-231.

Rutishauser H. 1958, "Solution of eigenvalue problems with the LR-transformation", Appl. Math. Ser. Nat. Bur. Stand., 49, 47-81.

Rutishauser H. 1964, "Une method pour le calcul des valeurs propres des matrices non symetriques", C. R. Acad. Sc. Paris, 259, 2758.

Rutishauser H. 1966, "The Jacobi method for real symmetric matrices", Numer. Math., 9, 1-10.

Rutishauser H. 1969, "Computational aspects of F.L. Bauer's simultaneous iteration method", Numer. Math., 13, 4-13.

Rutishauser H. 1970, "Simultaneous iteration

method for symmetric matrices", Numer. Math., 16, 205-223.

Wilkinson J.H. 1962, "Note on the quadratic convergence of the cyclic Jacobi process", Numer. Math., 4, 296-300.

Wilkinson J.H. 1965, "The algebraic eigenvalue problem", Clarendon Press, Oxford.

Wilkinson J.H. & Reinsch C. 1971, "Handbook for automatic computation, Volume II", Springer-Verlag, Berlin.

APPENDIX 1

A JACOBI PROGRAM FOR SYMMETRIC MATRICES

```
C021                MASTER
C022                INTEGER DIMN,EIVEC,ROT,RES,LINE(73),DATE(3)
C023                REAL A(44,44), V(44,44), D(44), B(44), Z(44), MC
C024                DATA LINE/73*1H*/
C025                K=1
C026                CALL ELAPSE (E,DATE,2,K)
C027                MC=E
C028                READ (5,10) NUMBER,DIMN,LENGTH
C029             10 FORMAT (2I2,I3)
C030                IF (LENGTH .LT. 120) K=-1
C031                DO 500 NO=1,NUMBER
C032                CALL INPUT (A,DIMN,N,NO,EIVEC,RES,NORM,FIELD)
C033                CALL TIME (T)
C034                IF (K) 120,999,100
C035            100 WRITE (6,110) (LINE(I), I=1,73), (DATE(I), I=1,3), T, NO
C036            110 FORMAT (1H1,//,24X,'THE METHOD OF JACOBI FOR THE EIGENSOLUTION OF
C037               1A REAL NXN SYMMETRIC MATRIX',/,24X,73A1,///,45X,'THIS PROGRAM WAS
C038               2RUN ON',I3,1X,A3,1X,I2,/,45X,'AND EXECUTION STARTED AT',A9,//,54X,
C039               3'EXAMPLE NUMBER',I3)
C040                GO TO 140
C041            120 WRITE (6,130) (LINE(I), I=1,34), (DATE(I), I=1,3), T, NO
C042            130 FORMAT (1H ,//////,' JACOBI FOR REAL SYMMETRIC MATRICES',/,1X,34A1
C043               1,///,' DATE:',I3,1X,A3,1X,I2,5X,'TIME:',A9,5X,'EXAMPLE NUMBER',I3)
C044            140 WRITE (6,150) N,N
C045            150 FORMAT (1H ,////,' THE ORIGINAL',I3,'X',I2,' MATRIX A IS GIVEN BEL
C046               1OW:')
C047                CALL MXOP (A,DIMN,DIMN,N,N,FIELD,LENGTH,6,K)
C048                CALL ELAPSE (E,DATE,1,K)
C049                CALL JACO (A,D,V,B,Z,DIMN,N,ROT,EIVEC,MC)
C050                CALL ELAPSE (E,DATE,1,K)
C051                IF (RES .EQ. 1) CALL CHECK (A,V,B,DIMN,N)
C052                IF (NORM .EQ. 1) CALL NORMALISATION (V,DIMN,N)
C053                IF (ROT) 200,250,250
C054            200 ROT=-ROT
C055                IF (K) 230,999,210
C056            210 WRITE (6,220)
C057            220 FORMAT (1H ,////,32X,'** WARNING - A MAXIMUM OF 50 SWEEPS HAVE BEE
C058               1N COMPLETED **')
C059                GO TO 230
C060                WRITE (6,240)
C061            240 FORMAT (1H ,////,' ** WARNING - A MAXIMUM OF 50 SWEEPS HAVE BEEN C
C062               1OMPLETED **')
C063            250 IF (K) 280,999,260
C064            260 WRITE (6,270) ROT
C065            270 FORMAT (1H ,////,43X,I4,' JACOBI ROTATIONS HAVE BEEN USED')
C066                GO TO 300
C067            280 WRITE (6,290) ROT
C068            290 FORMAT (1H ,////,1X,I4,' JACOBI ROTATIONS HAVE BEEN USED')
C069            300 WRITE (6,310)
C070            310 FORMAT (1H ,////,' THE EIGENVALUES OF A ARE:')
C071                CALL MXOP (D,1,DIMN,1,N,FIELD,LENGTH,6,0)
C072                IF (EIVEC .EQ. 0) GO TO 400
C073                WRITE (6,320)
C074            320 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIV
C075               1EN BELOW:')
C076                CALL MXOP (V,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C077                IF (RES .EQ. 0) GO TO 400
C078                WRITE (6,330)
C079            330 FORMAT (1H ,////,' THE RESIDUAL MATRIX V''AV IS GIVEN BELOW:')
C080                CALL MXOP (A,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C081            400 CALL ELAPSE (E,DATE,3,K)
C082                IF (K) 430,999,410
C083            410 WRITE (6,420) (LINE(I), I=1,60)
C084            420 FORMAT (1H ,////,31X,60A1)
C085                GO TO 500
C086            430 WRITE (6,440) (LINE(I), I=1,40)
C087            440 FORMAT (1H ,////,1X,40A1)
C088            500 CONTINUE
C089                STOP OK
C090            999 STOP NONM
C091                END

END OF SEGMENT, LENGTH   317, NAME  NONM
```

```
0092                  SUBROUTINE JACO (A,D,V,B,Z,DIMN,N,ROT,EIVEC,MC)
0093                  INTEGER DIMN,N,EIVEC,ROT,RES,P,Q
0094                  REAL A(DIMN,DIMN), D(DIMN), V(DIMN,DIMN), B(DIMN), Z(DIMN), MC
0095                  DOUBLE PRECISION SUM
0096                  IF (EIVEC .EQ. 0) GO TO 30
0097                  DO 10 I=1,N
0098                  DO 10 J=1,N
0099               10 V(I,J)=0.0
0100                  DO 20 P=1,N
0101               20 V(P,P)=1.0
0102               30 DO 40 P=1,N
0103                  D(P)=A(P,P)
0104                  B(P)=D(P)
0105               40 Z(P)=0.0
0106                  ROT=0
0107                  EPS=N*(N-1)*0.5*MC
0108                  DO 400 I=1,50
0109                  SUM=0.0
0110                  DO 100 P=1,N-1
0111                  DO 100 Q=P+1,N
0112                  TEMP=ABS(A(P,Q))
0113              100 SUM=SUM+DBLE(TEMP)
0114                  SM=SUM
0115                  IF (SM .LE. EPS) GO TO 500
0116                  TRESH=0.0
0117                  IF (I .LT. 4) TRESH=0.2*SM/N**2
0118                  DO 330 P=1,N-1
0119                  DO 330 Q=P+1,N
0120                  G=100.0*ABS(A(P,Q))
0121                  IF (I .GT. 4 .AND. ABS(D(P))+G .EQ. ABS(D(P)) .AND.
0122                 1   ABS(D(Q))+G .EQ. ABS(D(Q))) GO TO 310
0123                  IF (ABS(A(P,Q)) .LE. TRESH) GO TO 320
0124                  H=D(Q)-D(P)
0125                  IF (ABS(H)+G .EQ. ABS(H)) GO TO 110
0126                  THETA=0.5*H/A(P,Q)
0127                  T=1.0/(ABS(THETA)+SQRT(1.0+THETA**2))
0128                  IF (THETA .LT. 0.0) T=-T
0129                  GO TO 120
0130              110 T=A(P,Q)/H
0131              120 C=1.0/SQRT(1.0+T*T)
0132                  S=T*C
0133                  TAU=S/(1.0+C)
0134                  H=T*A(P,Q)
0135                  Z(P)=Z(P)-H
0136                  Z(Q)=Z(Q)+H
0137                  D(P)=D(P)-H
0138                  D(Q)=D(Q)+H
0139                  A(P,Q)=0.0
0140                  IF (P .EQ. 1) GO TO 210
0141                  DO 200 J=1,P-1
0142                  AJP=A(J,P)
0143                  AJQ=A(J,Q)
0144                  A(J,P)=AJP-S*(AJQ+AJP*TAU)
0145              200 A(J,Q)=AJQ+S*(AJP-AJQ*TAU)
0146              210 IF (P+1 .GT. Q-1) GO TO 230
0147                  DO 220 J=P+1,Q-1
0148                  APJ=A(P,J)
0149                  AJQ=A(J,Q)
0150                  A(P,J)=APJ-S*(AJQ+APJ*TAU)
0151              220 A(J,Q)=AJQ+S*(APJ-AJQ*TAU)
0152              230 IF (Q .EQ. N) GO TO 250
0153                  DO 240 J=Q+1,N
0154                  APJ=A(P,J)
0155                  AQJ=A(Q,J)
0156                  A(P,J)=APJ-S*(AQJ+APJ*TAU)
0157              240 A(Q,J)=AQJ+S*(APJ-AQJ*TAU)
0158              250 IF (EIVEC .EQ. 0) GO TO 300
0159                  DO 260 J=1,N
0160                  VJP=V(J,P)
0161                  VJQ=V(J,Q)
0162                  V(J,P)=VJP-S*(VJQ+VJP*TAU)
0163              260 V(J,Q)=VJQ+S*(VJP-VJQ*TAU)
0164              300 ROT=ROT+1
0165                  GO TO 320
0166              310 A(P,Q)=0.0
0167              320 CONTINUE
0168              330 CONTINUE
0169                  DO 340 P=1,N
0170                  B(P)=B(P)+Z(P)
0171                  D(P)=B(P)
0172              340 Z(P)=0.0
0173              400 CONTINUE
0174                  ROT=-ROT
0175              500 DO 540 P=1,N-1
0176                  DO 540 Q=P+1,N
0177                  TEMP=ABS(D(P))-ABS(D(Q))
0178                  IF (TEMP) 510,530,530
0179              510 SM=D(P)
0180                  D(P)=D(Q)
0181                  D(Q)=SM
0182                  DO 520 I=1,N
0183                  SM=V(I,P)
0184                  V(I,P)=V(I,Q)
0185              520 V(I,Q)=SM
0186              530 CONTINUE
0187              540 CONTINUE
0188                  DO 600 I=1,N-1
0189                  DO 600 J=I+1,N
0190              600 A(I,J)=A(J,I)
0191                  RETURN
0192                  END

END OF SEGMENT, LENGTH   821, NAME  JACO
```

```
0193                  SUBROUTINE NORMALISATION (V,DIMN,N)
0194                  INTEGER DIMN
0195                  REAL V(DIMN,DIMN)
0196                  DO 30 J=1,N
0197                  AUX=V(1,J)
0198                  DO 10 I=2,N
0199                  IF (ABS(V(I,J)) .GT. ABS(AUX)) AUX=V(I,J)
0200            10    CONTINUE
0201                  DO 20 I=1,N
0202                  V(I,J)=V(I,J)/AUX
0203            20    CONTINUE
0204            30    CONTINUE
0205                  RETURN
0206                  END

END OF SEGMENT, LENGTH    93, NAME  NORMALISATION


0207                  SUBROUTINE CHECK (A,V,B,DIMN,N)
0208                  REAL A(DIMN,DIMN), V(DIMN,DIMN), B(DIMN)
0209                  DO 40 I=1,N
0210                  DO 30 J=1,N
0211                  SUM=0.0
0212                  DO 20 K=1,N
0213            20    SUM=SUM+A(I,K)*V(K,J)
0214            30    B(J)=SUM
0215                  DO 40 J=1,N
0216            40    A(I,J)=B(J)
0217                  DO 80 K=1,N
0218                  DO 70 J=1,N
0219                  SUM=0.0
0220                  DO 60 I=1,N
0221            60    SUM=SUM+V(I,J)*A(I,K)
0222            70    B(J)=SUM
0223                  DO 80 I=1,N
0224            80    A(I,K)=B(I)
0225                  RETURN
0226                  END

END OF SEGMENT, LENGTH   156, NAME  CHECK


0227                  SUBROUTINE ELAPSE (E,DAT,NO,K)
0228                  INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
0229                  DATA CALL/2/
0230                  DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
0231                 1'OCT','NOV','DEC'/
0232                  GO TO (100,200,300), NO
0233            100   CALL MTIME (N)
0234                  E=N
0235                  CALL=3-CALL
0236                  IF (CALL .EQ. 2) GO TO 110
0237                  EE=E
0238                  RETURN
0239            110   E=E-EE
0240                  E=E+1.0E-3
0241                  RETURN
0242            200   CALL DEFBUF (2,8,BUFFER)
0243                  CALL DATE (E)
0244                  WRITE (2,210) E
0245            210   FORMAT (A8)
0246                  READ (2,220) (DAT(I), I=1,3)
0247            220   FORMAT (I2,1X,I2,1X,I2)
0248                  DAT(2)=MONTH(DAT(2))
0249                  E,EE=1.0E-08
0250            230   IF (1.0+EE .EQ. 1.0) GO TO 240
0251                  E=EE
0252                  EE=EE/2.0
0253                  GO TO 230
0254            240   RETURN
0255            300   IF (K) 370,370,310
0256            310   IF (E .GE. 60.0) GO TO 330
0257                  WRITE (6,320) E
0258            320   FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
0259                  GO TO 390
0260            330   IF (E .GE. 120.0) GO TO 350
0261                  E=E-60.0
0262                  WRITE (6,340) E
0263            340   FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
0264                 1)
0265                  GO TO 390
0266            350   I=E/60.0
0267                  E=E-I*60.0
0268                  WRITE (6,360) I, E
0269            360   FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
0270                 1ONDS')
0271                  GO TO 390
0272            370   WRITE (6,380) E
0273            380   FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
0274            390   RETURN
0275                  END

END OF SEGMENT, LENGTH   182, NAME  ELAPSE
```

```
0276                  SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
0277                  INTEGER DIMN,DIMP,P,WIDTH,STREAM
0278                  REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
0279                  DATA FRMT1/6H(      ,1HN,8HX,1P    ,1HN,1HE,8H)      /
0280                  DATA FRMT2/6H(      ,1HN,8HX,1P    ,1HN,1HE,8H)      /
0281                  IF (IFLAG) 10,200,10
0282                10 CALL DEBUF (2,16,BUFFER)
0283                   FRMT1(5)=BUFFER(1)
0284                   FRMT2(5)=FIELD
0285                   WRITE (2,20) FIELD
0286                20 FORMAT (A6)
0287                   READ (2,30) WIDTH
0288                30 FORMAT (1X,I2)
0289                   IF (P*WIDTH-LENGTH) 100,100,40
0290                40 N1=LENGTH/WIDTH
0291                   N2=P/N1
0292                   LAST=P-N1*N2
0293                   IF (IFLAG) 50,999,60
0294                50 N3=1
0295                   N4=1
0296                   GO TO 70
0297                60 N3=(LENGTH-N1*WIDTH)/2+1
0298                   N4=(LENGTH-LAST*WIDTH)/2+1
0299                70 WRITE (2,80) N3,N1,N4,LAST
0300                80 FORMAT (4I4)
0301                   READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
0302                90 FORMAT (4A4)
0303                   GO TO 300
0304               100 N2=0
0305                   IF (IFLAG) 110,999,120
0306               110 N3=1
0307                   GO TO 130
0308               120 N3=(LENGTH-P*WIDTH)/2+1
0309               130 WRITE (2,140) N3,P
0310               140 FORMAT (2I4)
0311                   READ (2,150) FRMT1(2), FRMT1(4)
0312               150 FORMAT (2A4)
0313                   GO TO 210
0314               200 IF (N2) 999,210,300
0315               210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
0316                   RETURN
0317               300 N2N1=N2*N1
0318                   IF (N-1) 999,400,310
0319               310 DO 330 K=1,N2N1,N1
0320                   KK=K+N1-1
0321                   WRITE (STREAM,320) K, KK
0322               320 FORMAT (1H ,/,' COLUMNS',I3,' TO',I3,' ARE:')
0323               330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
0324                   IF (LAST-1) 370,340,360
0325               340 WRITE (STREAM,350) P
0326               350 FORMAT (1H ,/,' COLUMN',I3,' IS:')
0327                   WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
0328                   GO TO 370
0329               360 KK=N2N1+1
0330                   WRITE (STREAM,320) KK,P
0331                   WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
0332               370 RETURN
0333               400 DO 410 K=1,N2N1,N1
0334                   KK=K+N1-1
0335               410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
0336                   IF (LAST-1) 440,420,430
0337               420 WRITE (STREAM,FRMT2) A(1,P)
0338                   GO TO 440
0339               430 KK=N2N1+1
0340                   WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
0341               440 RETURN
0342               999 STOP MXOP
                       END

END OF SEGMENT, LENGTH   427, NAME  MXOP
```

```
0343                  SUBROUTINE INPUT (A,DIMN,N,NO,EIVEC,RES,NORM,FIELD)
0344                  INTEGER DIMN,RES,EIVEC
0345                  REAL A(DIMN,DIMN), FIELD
0346                  READ (5,10) N, EIVEC, FIELD, RES, NORM
0347                10 FORMAT (I2,I1,A6,2I1)
0348                  GO TO (100,200,300,400,500,600,700,800,900), NO
0349               100 CONTINUE
0350                  READ (5,110) ((A(I,J), J=1,4), I=1,4)
0351               110 FORMAT (4G0.0)
0352                  RETURN
0353               200 CONTINUE
0354                  GO TO 100
0355               300 CONTINUE
0356                  DO 310 I=1,N
0357                  DO 310 J=1,N
0358               310 A(I,J)=0.0
0359                  DO 320 I=1,N
0360                  A(I,1)=1.0
0361                  A(I,N)=I
0362               320 A(N,I)=I
0363                  RETURN
0364               400 CONTINUE
0365                  DO 410 I=1,N
0366                  DO 410 J=1,N
0367               410 A(I,J)=1.0/FLOAT(I+J-1)
0368                  RETURN
0369               500 CONTINUE
0370                  X=3.0
0371               510 X=2.0*X
0372                  DO 520 I=1,N
0373                  DO 520 J=1,N
0374               520 A(I,J)=0.0
0375                  A(1,1)=-1.0
0376                  A(N,N)=-1.0
0377                  DO 530 I=1,N-1
0378                  A(I,I+1)=X
0379               530 A(I+1,I)=X
0380                  DO 540 I=1,N-2
0381                  A(I,I+2)=1.0
0382               540 A(I+2,I)=1.0
0383                  RETURN
0384               600 CONTINUE
0385                  X=0.0
0386                  GO TO 510
0387               700 CONTINUE
0388                  READ (5,710) ((A(I,J), J=1,8), I=1,8)
0389               710 FORMAT (8G0.0)
0390                  RETURN
0391               800 CONTINUE
0392                  DO 810 I=1,N
0393                  DO 810 J=1,N
0394               810 A(I,J)=0.0
0395                  A(1,1),A(N,N)=5.0
0396                  A(1,2),A(2,1),A(N-1,N),A(N,N-1)=2.0
0397                  DO 820 I=2,N-1
0398               820 A(I,I)=4.0
0399                  DO 830 I=2,N-2
0400                  J=I+1
0401               830 A(I,J),A(J,I)=3.0
0402                  DO 840 I=1,N-2
0403                  J=I+2
0404               840 A(I,J),A(J,I)=1.0
0405                  DO 850 I=1,N-3
0406                  J=I+3
0407               850 A(I,J),A(J,I)=1.0
0408                  RETURN
0409               900 CONTINUE
0410                  END

END OF SEGMENT, LENGTH   492, NAME  INPUT
```

THE METHOD OF JACOBI FOR THE EIGENSOLUTION OF A REAL NXN SYMMETRIC MATRIX

THIS PROGRAM WAS RUN ON 13 AUG 74
AND EXECUTION STARTED AT 06/11/41

EXAMPLE NUMBER 2

THE ORIGINAL 4X 4 MATRIX A IS GIVEN BELOW:

4.00000000000E 00	4.00000000000E 00	1.00000000000E 00	4.00000000000E 00
4.00000000000E 00	6.00000000000E 00	1.00000000000E 00	4.00000000000E 00
4.00000000000E 00	1.00000000000E 00	6.00000000000E 00	4.00000000000E 00
1.00000000000E 00	4.00000000000E 00	4.00000000000E 00	8.00000000000E 00

17 JACOBI ROTATIONS HAVE BEEN USED

THE EIGENVALUES OF A ARE:

1.51000000000E 01	5.00000000000E 00	5.00000000000E 00	-1.00000000000E 00

THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIVEN BELOW:

1.00000000000E 00	-8.40436325524E-01	-9.99999999996E-01	-9.99999999999E-01
1.00000000000E 00	1.00000000000E 00	-8.40436325524E-01	9.99999999997E-01
1.00000000000E 00	-9.99999999997E-01	8.40436325524E-01	1.00000000000E 00
9.99999999997E-01	8.40436325523E-01	1.00000000000E 00	-9.99999999999E-01

THE RESIDUAL MATRIX V'AV IS GIVEN BELOW:

1.50100000004E 01	-2.91038304577E-11	1.36967237706E-10	2.18278728437E-11
-2.91038304577E-11	5.00000000002E 00	-2.91038304577E-11	-5.82076609137E-11
1.74622982774E-10	1.45519152283E-11	5.00000000000E 00	-5.09317032997E-11
2.91038304577E-11	-1.45519152283E-11	-8.73114913702E-11	-1.00000000000E 00

EXECUTION TIME WAS 0.057 SECONDS

APPENDIX 2

A JACOBI PROGRAM FOR HERMITIAN MATRICES

```
      MASTER
      INTEGER DIMN,EIVEC,ROT,RES,LINE(79),DATE(3)
      REAL AR(12,12),AI(12,12), VR(12,12),VI(12,12), D(12), B(12), Z(12)
      REAL MC
      DATA LINE/79*1H*/
      K=1
      CALL ELAPSE (E,DATE,2,K)
      MC=E
      READ (5,10) NUMBER,DIMN,LENGTH
   10 FORMAT (2I2,I3)
      IF (LENGTH .LT. 120) K=-1
      DO 500 NO=1,NUMBER
      CALL INPUT (AR,AI,DIMN,N,NO,EIVEC,RES,NORM,FIELD)
      CALL TIME (T)
      IF (K) 120,999,100
  100 WRITE (6,110) (LINE(I), I=1,79), (DATE(I), I=1,3), T, NO
  110 FORMAT (1H1,//,21X,'A JACOBI-LIKE METHOD FOR OBTAINING THE EIGENSO
     1LUTIONS OF A NXN HERMITIAN MATRIX',/,21X,79A1,///,45X,'THIS PROGRA
     2M WAS RUN ON',13,1X,A3,1X,I2,//,45X,'AND EXECUTION STARTED AT',A9,
     3 //,54X,'EXAMPLE NUMBER',I3)
      GO TO 140
  120 WRITE (6,130) (LINE(I), I=1,43), (DATE(I), I=1,3), T, NO
  130 FORMAT (1H1,//////,' A JACOBI-LIKE METHOD FOR HERMITIAN MATRICES',
     1 /,1X,43A1,///,' DATE',I3,1X,A3,1X,I2,5X,'TIME',A9,5X,'EXAMPLE NU
     2MBER',I3)
  140 WRITE (6,150) N,N
  150 FORMAT (1H ,////,' THE ORIGINAL',I3,'X',I2,' MATRIX A IS GIVEN BEL
     1OW:')
      WRITE (6,160)
  160 FORMAT (1H ,/,' REAL PART')
      CALL MXOP (AR,DIMN,DIMN,N,N,FIELD,LENGTH,6,K)
      WRITE (6,161)
  161 FORMAT (1H ,/,' IMAGINARY PART')
      CALL MXOP (AI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
      CALL ELAPSE (E,DATE,1,K)
      CALL HMJO (AR,AI,D,VR,VI,B,Z,DIMN,N,ROT,EIVEC,MC)
      CALL ELAPSE (E,DATE,1,K)
      IF (RES .EQ. 1) CALL CHECK (AR,AI,VR,VI,B,Z,DIMN,N)
      IF (NORM .EQ. 1) CALL NORMALISATION (VR,VI,DIMN,N)
      IF (ROT) 200,250,250
  200 ROT=-ROT
      IF (K) 230,999,210
  210 WRITE (6,220)
  220 FORMAT (1H ,////,32X,'** WARNING - A MAXIMUM OF 50 SWEEPS HAVE BEE
     1N COMPLETED **')
      GO TO 250
  230 WRITE (6,240)
  240 FORMAT (1H ,////,' ** WARNING - A MAXIMUM OF 50 SWEEPS HAVE BEEN C
     1OMPLETED **')
  250 IF (K) 280,999,260
  260 WRITE (6,270) ROT
  270 FORMAT (1H ,////,41X,I4,' JACOBI-LIKE ROTATIONS HAVE BEEN USED')
      GO TO 300
  280 WRITE (6,290) ROT
  290 FORMAT (1H ,////,1X,I4,' JACOBI-LIKE ROTATIONS HAVE BEEN USED')
  300 WRITE (6,310)
  310 FORMAT (1H ,////,' THE EIGENVALUES OF A ARE:')
      CALL MXOP (D,1,DIMN,1,N,FIELD,LENGTH,6,0)
      IF (EIVEC .EQ. 0) GO TO 400
      WRITE (6,320)
  320 FORMAT (1H ,/////,' THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIV
     1EN BELOW:')
      WRITE (6,160)
      CALL MXOP (VR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
      WRITE (6,161)
      CALL MXOP (VI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
      IF (RES .EQ. 0) GO TO 400
      WRITE (6,330)
  330 FORMAT (1H ,/////,' THE RESIDUAL MATRIX V"AV IS GIVEN BELOW:')
      WRITE (6,160)
      CALL MXOP (AR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
      WRITE (6,161)
      CALL MXOP (AI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
  400 CALL ELAPSE (E,DATE,3,K)
      IF (K) 430,999,410
  410 WRITE (6,420) (LINE(I), I=1,60)
  420 FORMAT (1H ,////,31X,60A1)
      GO TO 500
  430 WRITE (6,440) (LINE(I), I=1,40)
  440 FORMAT (1H ,////,1X,40A1)
  500 CONTINUE
      STOP OK
  999 STOP NONM
      END
```

```
C105                SUBROUTINE HMJO (AR,AI,D,VR,VI,B,Z,DIMN,N,ROT,EIVEC,MC)
C106                INTEGER DIMN,N,EIVEC,ROT,P,Q,RES
C107                REAL AR(DIMN,DIMN), AI(DIMN,DIMN), VR(DIMN,DIMN), VI(DIMN,DIMN)
C108                REAL D(DIMN), B(DIMN), Z(DIMN), MC
C109                DOUBLE PRECISION SUM
C110                IF (EIVEC .EQ. 0) GO TO 30
C111                DO 10 I=1,N
C112                DO 10 J=1,N
C113                VR(I,J)=0.0
C114             10 VI(I,J)=0.0
C115                DO 20 P=1,N
C116             20 VR(P,P)=1.0
C117             30 DO 40 P=1,N
C118                D(P)=AR(P,P)
C119                B(P)=D(P)
C120             40 Z(P)=0.0
C121                ROT=0
C122                EPS=N*(N-1)*0.5*MC
C123                DO 600 I=1,50
C124                SUM=0.0
C125                DO 100 P=1,N-1
C126                DO 100 Q=P+1,N
C127                TEMP=SQRT(AR(P,Q)**2+AI(P,Q)**2)
C128            100 SUM=SUM+DBLE(TEMP)
C129                SM=SUM
C130                IF (SM .LE. EPS) GO TO 610
C131                TRESH=0.0
C132                IF (I .LT. 4) TRESH=0.2*SM/N**2
C133                DO 500 P=1,N-1
C134                DO 500 Q=P+1,N
C135                TEMP=SQRT(AR(P,Q)**2+AI(P,Q)**2)
C136                G=100.0*TEMP
C137                IF (I .GT. 4 .AND. ABS(D(P))+G .EQ. ABS(D(P)) .AND.
C138               1 ABS(D(Q))+G .EQ. ABS(D(Q))) GO TO 410
C139                IF (TEMP .LE. TRESH) GO TO 420
C140                E=AR(P,Q)
C141                F=AI(P,Q)
C142                IF (ABS(E)-ABS(F)) 200,210,210
C143            200 T=E/F
C144                ST=1.0/SQRT(1.0+T*T)
C145                CT=T*ST
C146                GO TO 220
C147            210 T=F/E
C148                CT=1.0/SQRT(1.0+T*T)
C149                ST=T*CT
C150            220 OMEGA=E*CT+F*ST
C151                H=D(Q)-D(P)
C152                IF (ABS(H)+G .EQ. ABS(H)) GO TO 230
C153                THETA=0.5*H/OMEGA
C154                T=1.0/(ABS(THETA)+SQRT(1.0+THETA**2))
C155                IF (THETA .LT. 0.0) T=-T
C156                GO TO 240
C157            230 T=OMEGA/H
C158            240 C=1.0/SQRT(1.0+T*T)
C159                S=T*C
C160                TAU=S/(1.0+C)
C161                H=T*OMEGA
C162                Z(P)=Z(P)-H
C163                Z(Q)=Z(Q)+H
C164                D(P)=D(P)-H
C165                D(Q)=D(Q)+H
C166                AR(P,Q)=0.0
C167                AI(P,Q)=0.0
C168                IF (P .EQ. 1) GO TO 310
C169                DO 300 J=1,P-1
```

```
C170                ARJP=AR(J,P)
C171                ARJQ=AR(J,Q)
C172                AIJP=AI(J,P)
C173                AIJQ=AI(J,Q)
C174                AR(J,P)=ARJP-S*(ARJQ*CT+AIJQ*ST+ARJP*TAU)
C175                AR(J,Q)=ARJQ+S*(ARJP*CT-AIJP*ST-ARJQ*TAU)
C176                AI(J,P)=AIJP-S*(AIJQ*CT-ARJQ*ST+AIJP*TAU)
C177          300   AI(J,Q)=AIJQ+S*(AIJP*CT+ARJP*ST-AIJQ*TAU)
C178          310   IF (P+1 .GT. Q-1) GO TO 330
C179                DO 320 J=P+1,Q-1
C180                ARPJ=AR(P,J)
C181                ARJQ=AR(J,Q)
C182                AIPJ=AI(P,J)
C183                AIJQ=AI(J,Q)
C184                AR(P,J)=ARPJ-S*(ARJQ*CT+AIJQ*ST+ARPJ*TAU)
C185                AR(J,Q)=ARJQ+S*(ARPJ*CT+AIPJ*ST-ARJQ*TAU)
C186                AI(P,J)=AIPJ-S*(-AIJQ*CT+ARJQ*ST+AIPJ*TAU)
C187          320   AI(J,Q)=AIJQ+S*(-AIPJ*CT+ARPJ*ST-AIJQ*TAU)
C188          330   IF (Q .EQ. N) GO TO 350
C189                DO 340 J=Q+1,N
C190                ARPJ=AR(P,J)
C191                ARQJ=AR(Q,J)
C192                AIPJ=AI(P,J)
C193                AIQJ=AI(Q,J)
C194                AR(P,J)=ARPJ-S*(ARQJ*CT-AIQJ*ST+ARPJ*TAU)
C195                AR(Q,J)=ARQJ+S*(ARPJ*CT+AIPJ*ST-ARQJ*TAU)
C196                AI(P,J)=AIPJ-S*(AIQJ*CT+ARQJ*ST+AIPJ*TAU)
C197          340   AI(Q,J)=AIQJ+S*(AIPJ*CT-ARPJ*ST-AIQJ*TAU)
C198          350   IF (EIVEC .EQ. 0) GO TO 400
C199                DO 360 J=1,N
C200                VRJP=VR(J,P)
C201                VRJQ=VR(J,Q)
C202                VIJP=VI(J,P)
C203                VIJQ=VI(J,Q)
C204                VR(J,P)=VRJP-S*(VRJQ*CT+VIJQ*ST+VRJP*TAU)
C205                VR(J,Q)=VRJQ+S*(VRJP*CT-VIJP*ST-VRJQ*TAU)
C206                VI(J,P)=VIJP-S*(VIJQ*CT-VRJQ*ST+VIJP*TAU)
C207          360   VI(J,Q)=VIJQ+S*(VIJP*CT+VRJP*ST-VIJQ*TAU)
C208          400   ROT=ROT+1
C209                GO TO 420
C210          410   AR(P,Q)=0.0
C211                AI(P,Q)=0.0
C212          420   CONTINUE
C213          500   CONTINUE
C214                DO 510 P=1,N
C215                B(P)=B(P)+Z(P)
C216                D(P)=B(P)
C217          510   Z(P)=0.0
C218          600   CONTINUE
C219                ROT=-ROT
C220          610   DO 650 P=1,N-1
C221                DO 650 Q=P+1,N
C222                TEMP=ABS(D(P))-ABS(D(Q))
C223                IF (TEMP) 620,640,640
C224          620   SM=D(P)
C225                D(P)=D(Q)
C226                D(Q)=SM
C227                DO 630 I=1,N
C228                SM=VR(I,P)
C229                VR(I,P)=VR(I,Q)
C230                VR(I,Q)=SM
C231                SM=VI(I,P)
C232                VI(I,P)=VI(I,Q)
C233          630   VI(I,Q)=SM
C234          640   CONTINUE
C235          650   CONTINUE
C236                DO 670 I=1,N-1
C237                DO 670 J=I+1,N
C238                AR(I,J)=AR(J,I)
C239          670   AI(I,J)=-AI(J,I)
C240                RETURN
C241                END

END OF SEGMENT, LENGTH  1211, NAME  HHJO
```

```
0242                  SUBROUTINE NORMALISATION (VR,VI,DIMN,N)
0243                  INTEGER DIMN
0244                  REAL VR(DIMN,DIMN),VI(DIMN,DIMN)
0245                  DO 30 J=1,N
0246                  AUX=VR(1,J)**2+VI(1,J)**2
0247                  DO 10 I=2,N
0248                  TEMP=VR(I,J)**2+VI(I,J)**2
0249                  IF (TEMP .GT. AUX) AUX=TEMP
0250               10 CONTINUE
0251                  AUX=SQRT(AUX)
0252                  DO 20 I=1,N
0253                  VR(I,J)=VR(I,J)/AUX
0254               20 VI(I,J)=VI(I,J)/AUX
0255               30 CONTINUE
0256                  RETURN
0257                  END
```

END OF SEGMENT, LENGTH 126, NAME NORMALISATION

```
0258                  SUBROUTINE CHECK (AR,AI,VR,VI,B,Z,DIMN,N)
0259                  REAL AR(DIMN,DIMN),AI(DIMN,DIMN), VR(DIMN,DIMN),VI(DIMN,DIMN)
0260                  REAL B(DIMN), Z(DIMN)
0261                  DOUBLE PRECISION SUMR,SUMI
0262                  DO 40 I=1,N
0263                  DO 30 J=1,N
0264                  SUMR,SUMI=0.0
0265                  DO 20 K=1,N
0266                  SUMR=SUMR+AR(I,K)*VR(K,J)-AI(I,K)*VI(K,J)
0267               20 SUMI=SUMI+AR(I,K)*VI(K,J)+AI(I,K)*VR(K,J)
0268                  B(J)=SUMR
0269               30 Z(J)=SUMI
0270                  DO 40 J=1,N
0271                  AR(I,J)=B(J)
0272               40 AI(I,J)=Z(J)
0273                  DO 80 K=1,N
0274                  DO 70 J=1,N
0275                  SUMR,SUMI=0.0
0276                  DO 60 I=1,N
0277                  SUMR=SUMR+VR(I,J)*AR(I,K)+VI(I,J)*AI(I,K)
0278               60 SUMI=SUMI+VR(I,J)*AI(I,K)-VI(I,J)*AR(I,K)
0279                  B(J)=SUMR
0280               70 Z(J)=SUMI
0281                  DO 80 I=1,N
0282                  AR(I,K)=B(I)
0283               80 AI(I,K)=Z(I)
0284                  RETURN
0285                  END
```

END OF SEGMENT, LENGTH 340, NAME CHECK

```
0286                  SUBROUTINE ELAPSE (E,DAT,NO,K)
0287                  INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
0288                  DATA CALL/2/
0289                  DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
0290                 1 'OCT','NOV','DEC'/
0291                  GO TO (100,200,300), NO
0292              100 CALL MTIME (N)
0293                  E=N
0294                  CALL=3-CALL
0295                  IF (CALL .EQ. 2) GO TO 110
0296                  EE=E
0297                  RETURN
0298              110 E=E-EE
0299                  E=E*1.0E-3
0300                  RETURN
0301              200 CALL DEFBUF (2,8,BUFFER)
0302                  CALL DATE (E)
0303                  WRITE (2,210) E
0304              210 FORMAT (A8)
0305                  READ (2,220) (DAT(I), I=1,3)
0306              220 FORMAT (I2,1X,I2,1X,I2)
0307                  DAT(2)=MONTH(DAT(2))
0308                  E,EE=1.0E-03
0309              230 IF (1.0+EE .EQ. 1.0) GO TO 240
0310                  E=EE
0311                  EE=EE/2.0
0312                  GO TO 230
0313              240 RETURN
0314              300 IF (K) 370,370,310
0315              310 IF (E .GE. 60.0) GO TO 330
0316                  WRITE (6,320) E
0317              320 FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
0318                  GO TO 360
0319              330 IF (E .GE. 120.0) GO To 350
0320                  E=E-60.0
0321                  WRITE (6,340) E
0322              340 FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
0323                 1)
0324                  GO TO 360
0325              350 I=E/60.0
0326                  E=E-I*60.0
0327                  WRITE (6,360) I, E
0328              360 FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
0329                 1ONDS')
0330                  GO TO 390
0331              370 WRITE (6,380) E
0332              380 FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
0333              390 RETURN
0334                  END
```

END OF SEGMENT, LENGTH 182, NAME ELAPSE

```
0335                  SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
0336                  INTEGER DIMN,DIMP,P,WIDTH,STREAM
0337                  REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
0338                  DATA FRMT1/8H(       ,1HN,8HX,1P    ,1HN,1HE,8H)       /
0339                  DATA FRMT2/8H(       ,1HN,8HX,1P    ,1HN,1HE,8H)       /
0340                  IF (IFLAG) 10,200,10
0341               10 CALL DEFBUF (2,16,BUFFER)
0342                  FRMT1(5), FRMT2(5)=FIELD
0343                  WRITE (2,20) FIELD
0344               20 FORMAT (A6)
0345                  READ (2,30) WIDTH
0346               30 FORMAT (1X,I2)
0347                  IF (P*WIDTH-LENGTH) 100,100,40
0348               40 N1=LENGTH/WIDTH
0349                  N2=P/N1
0350                  LAST=P-N1*N2
0351                  IF (IFLAG) 50,999,60
0352               50 N3=1
0353                  N4=1
0354                  GO TO 70
0355               60 N3=(LENGTH-N1*WIDTH)/2+1
0356                  N4=(LENGTH-LAST*WIDTH)/2+1
0357               70 WRITE (2,80) N3,N1,N4,LAST
0358               80 FORMAT (4I4)
0359                  READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
0360               90 FORMAT (4A4)
0361                  GO TO 300
0362              100 N2=0
0363                  IF (IFLAG) 110,999,120
0364              110 N3=1
0365                  GO TO 130
0366              120 N3=(LENGTH-P*WIDTH)/2+1
0367              130 WRITE (2,140) N3,P
0368              140 FORMAT (2I4)
0369                  READ (2,150) FRMT1(2), FRMT1(4)
0370              150 FORMAT (2A4)
0371                  GO TO 210
0372              200 IF (N2) 999,210,300
0373              210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
0374                  RETURN
0375              300 N2N1=N2*N1
0376                  IF (N-1) 999,400,310
0377              310 DO 330 K=1,N2N1,N1
0378                  KK=K+N1-1
0379                  WRITE (STREAM,320) K, KK
0380              320 FORMAT (1H ,7,' COLUMNS',I3,' TO',I3,' ARE:')
0381              330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
0382                  IF (LAST-1) 370,340,360
0383              340 WRITE (STREAM,350) P
0384              350 FORMAT (1H ,7,' COLUMN',I3,' IS:')
0385                  WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
0386                  GO TO 370
0387              360 KK=N2N1+1
0388                  WRITE (STREAM,320) KK,P
0389                  WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
0390              370 RETURN
0391              400 DO 410 K=1,N2N1,N1
0392                  KK=K+N1-1
0393              410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
0394                  IF (LAST-1) 440,420,430
0395              420 WRITE (STREAM,FRMT2) A(1,P)
0396                  GO TO 440
0397              430 KK=N2N1+1
0398                  WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
0399              440 RETURN
0400              999 STOP MXOP
0401                  END

END OF SEGMENT, LENGTH   427, NAME  MXOP
```

```
0402                  SUBROUTINE INPUT (AR,AI,DIMN,N,NO,EIVEC,RES,NORM,FIELD)
0403                  INTEGER DIMN,EIVEC,RES
0404                  REAL AR(DIMN,DIMN),AI(DIMN,DIMN)
0405                  READ (5,10) N,FIELD,EIVEC,RES,NORM
0406               10 FORMAT (I2,A6,3I1)
0407                  GO TO (100,200,300,400,500,600,700,800,900), NO
0408              100 CONTINUE
0409                  READ (5,110) ((AR(I,J),AI(I,J), J=1,N), I=1,N)
0410              110 FORMAT (2G0.0)
0411                  RETURN
0412              200 CONTINUE
0413                  GO TO 100
0414              300 CONTINUE
0415                  GO TO 100
0416              400 CONTINUE
0417                  DO 410 I=1,N
0418                  DO 410 J=1,N
0419              410 AR(I,J), AI(I,J)=0.0
0420                  DO 420 I=1,N-1
0421                  Z=SQRT(FLOAT(I*(N-I)))
0422                  AI(I,I+1)=Z
0423              420 AI(I+1,I)=-Z
0424                  RETURN
0425              500 CONTINUE
0426                  GO TO 400
0427              600 CONTINUE
0428                  GO TO 400
0429              700 CONTINUE
0430                  GO TO 400
0431              800 CONTINUE
0432                  GO TO 400
0433              900 CONTINUE
0434                  GO TO 400
0435                  END

END OF SEGMENT, LENGTH   203, NAME  INPUT
```

A JACOBI-LIKE METHOD FOR OBTAINING THE EIGENSOLUTIONS OF A NXN HERMITIAN MATRIX

THIS PROGRAM WAS RUN ON 22 AUG 74
AND EXECUTION STARTED AT 17/26/07

EXAMPLE NUMBER 2

THE ORIGINAL 4X 4 MATRIX A IS GIVEN BELOW:

REAL PART

7.0000000000E 00	3.0000000000E 00	1.0000000000E 00	-1.0000000000E 00
3.0000000000E 00	7.0000000000E 00	1.0000000000E 00	-1.0000000000E 00
1.0000000000E 00	1.0000000000E 00	7.0000000000E 00	-3.0000000000E 00
-1.0000000000E 00	-1.0000000000E 00	-3.0000000000E 00	7.0000000000E 00

IMAGINARY PART

0.0000000000E-01	0.0000000000E-01	2.0000000000E 00	2.0000000000E 00
0.0000000000E-01	0.0000000000E-01	-2.0000000000E 00	-2.0000000000E 00
-2.0000000000E 00	2.0000000000E 00	0.0000000000E-01	0.0000000000E-01
-2.0000000000E 00	2.0000000000E 00	0.0000000000E-01	0.0000000000E-01

17 JACOBI-LIKE ROTATIONS HAVE BEEN USED

THE EIGENVALUES OF A ARE:

1.2000000000E 01	8.0000000000E 00	8.0000000000E 00	2.2178493499E-11

THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIVEN BELOW:

REAL PART

9.9999999998E-01	-7.5344577084E-01	-2.3682526042E-02	9.9999999997E-01
9.9999999999E-01	-2.3635386941E-02	6.5751005347E-01	-1.0000000000E 00
9.9999999990E-01	7.0345434262E-01	7.1626815161E-02	9.5532478115E-13
-1.0000000000E 00	-7.1626815165E-02	7.0545434262E-01	1.1696948540E-12

IMAGINARY PART

3.8184358637E-12	6.5751005347E-01	2.3635386937E-02	1.2589642869E-12
1.2059210404E-12	2.3682526027E-02	-7.5344577084E-01	7.5065995085E-13
4.7408111140E-13	2.4308902187E-02	7.0550148170E-01	9.9999999996E-01
1.2215076004E-12	7.0550148169E-01	-2.4308902194E-02	9.9999999997E-01

THE RESIDUAL MATRIX V"AV IS GIVEN BELOW:

REAL PART

1.2000000000E 01	4.1991872366E-11	-1.4655570826E-10	-8.0678643980E-11
7.5931012360E-11	8.0000000000E 00	5.5933924159E-11	9.2627328522E-12
-1.1913742342E-10	5.5472176809E-11	8.0000000000E 00	-7.9095697697E-11
-7.4676235182E-11	-3.2174115328E-12	-9.3576094988E-11	7.2759576104E-12

IMAGINARY PART

-2.0447904742E-21	-1.1635165175E-10	4.1585646435E-11	-4.4902830586E-11
1.3452043715E-10	7.0401622293E-12	1.1823431123E-11	-3.6440484147E-11
-9.2766424537E-11	-2.9710406865E-12	-2.4073187888E-11	-5.4440284912E-11
3.8149387708E-11	3.6962325777E-11	8.1458943384E-11	-1.2411902714E-22

EXECUTION TIME WAS 0.116 SECONDS

APPENDIX 3

A JACOBI PROGRAM FOR NORMAL MATRICES

```
C021                MASTER
C022                INTEGER DIMN,EIVEC,ROT,RES,LINE(76),DATE(3)
C023                REAL AR(12,12),AI(12,12), VR(12,12),VI(12,12), DR(12),DI(12)
C024                REAL YR(12),YI(12), ZR(12),ZI(12), AAR(12,12),AAI(12,12), MC
C025                DATA LINE/76*1H*/
C026                K=1
C027                CALL ELAPSE (E,DATE,2,K)
C028                MC=E
C029                READ (5,10) NUMBER,DIMN,LENGTH
C030             10 FORMAT (3I2,I3)
C031                IF (LENGTH .LT. 120) K=-1
C032                DO 500 NO=1,NUMBER
C033                CALL INPUT (AR,AI,DIMN,N,NO,EIVEC,RES,NORM,FIELD)
C034                DO 20 I=1,N
C035                DO 20 J=1,N
C036                AAR(I,J)=AR(I,J)
C037             20 AAI(I,J)=AI(I,J)
C038                CALL TIME (T)
C039                IF (K) 120,999,100
C040            100 WRITE (6,110) (LINE(I), I=1,76), (DATE(I), I=1,3), T, NO
C041            110 FORMAT (1H1,///,32X,'A JACOBI-LIKE METHOD FOR OBTAINING THE EIGENSO
C042               1LUTIONS OF A NXN NORMAL MATRIX',/,21X,76A1,///,45X,'THIS PROGRAM W
C043               2AS RUN ON',I3,1X,A3,1X,I2,//,45X,'AND EXECUTION STARTED AT',A9,///,
C044               3 54X,'EXAMPLE NUMBER',I3)
C045                GO TO 140
C046            120 WRITE (6,130) (LINE(I), I=1,40), (DATE(I), I=1,3), T, NO
C047            130 FORMAT (1H ,//////,' A JACOBI-LIKE METHOD FOR NORMAL MATRICES',/,
C048               1 1X,40A1,///,' DATE:',I3,1X,A3,1X,I2,5X,'TIME:',A9,5X,'EXAMPLE NUMB
C049               2ER',I3)
C050            140 WRITE (6,150) N,N
C051            150 FORMAT (1H ,////,' THE ORIGINAL',I3,'X',I2,' MATRIX A IS GIVEN BEL
C052               1OW:')
C053                WRITE (6,160)
C054            160 FORMAT (1H ,/,' REAL PART')
C055                CALL MXOP (AR,DIMN,DIMN,N,N,FIELD,LENGTH,6,K)
C056                WRITE (6,161)
C057            161 FORMAT (1H ,/,' IMAGINARY PART')
C058                CALL MXOP (AI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C059                CALL ELAPSE (E,DATE,1,K)
C060                CALL MNJO (AR,AI,DR,DI,VR,VI,YR,YI,ZR,ZI,DIMN,N,ROT,EIVEC,MC)
C061                CALL ELAPSE (E,DATE,1,K)
C062                IF (RES .EQ. 1) CALL CHECK (AAR,AAI,VR,VI,YR,YI,DIMN,N)
C063                IF (NORM .EQ. 1) CALL NORMALISATION (VR,VI,DIMN,N)
C064                IF (ROT) 200,250,250
C065            200 ROT=-ROT
C066                IF (K) 230,999,210
C067            210 WRITE (6,220)
C068            220 FORMAT (1H ,////,32X,'** WARNING = A MAXIMUM OF 50 SWEEPS HAVE BEE
C069               1N COMPLETED **')
C070                GO TO 250
C071            230 WRITE (6,240)
C072            240 FORMAT (1H ,////,' ** WARNING = A MAXIMUM OF 50 SWEEPS HAVE BEEN C
C073               1OMPLETED **')
C074            250 IF (K) 280,999,260
C075            260 WRITE (6,270) ROT
C076            270 FORMAT (1H ,////,41X,I4,' JACOBI-LIKE ROTATIONS HAVE BEEN USED')
C077                GO TO 300
C078            280 WRITE (6,290) ROT
C079            290 FORMAT (1H ,////,1X,I4,' JACOBI-LIKE ROTATIONS HAVE BEEN USED')
C080            300 WRITE (6,310)
C081            310 FORMAT (1H ,////,' THE EIGENVALUES OF A ARE GIVEN BELOW:')
C082                WRITE (6,160)
C083                CALL MXOP (DR,1,DIMN,1,N,FIELD,LENGTH,6,0)
C084                WRITE (6,161)
C085                CALL MXOP (DI,1,DIMN,1,N,FIELD,LENGTH,6,0)
C086                IF (EIVEC .EQ. 0) GO TO 400
C087                WRITE (6,320)
C088            320 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIV
C089               1EN BELOW:')
C090                WRITE (6,160)
C091                CALL MXOP (VR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C092                WRITE (6,161)
C093                CALL MXOP (VI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C094                IF (RES .EQ. 0) GO TO 400
C095                WRITE (6,330)
C096            330 FORMAT (1H ,////,' THE RESIDUAL MATRIX V"AV IS GIVEN BELOW:')
C097                WRITE (6,160)
C098                CALL MXOP (AAR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C099                WRITE (6,161)
C100                CALL MXOP (AAI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C101            400 CALL ELAPSE (E,DATE,3,K)
C102                IF (K) 430,999,410
C103            410 WRITE (6,420) (LINE(I), I=1,60)
C104            420 FORMAT (1H ,////,31X,60A1)
C105                GO TO 500
C106            430 WRITE (6,440) (LINE(I), I=1,40)
C107            440 FORMAT (1H ,////,1X,40A1)
C108            500 CONTINUE
C109                STOP OK
C110            999 STOP NONM
C111                END

END OF SEGMENT, LENGTH   437, NAME  NONM
```

```
C112                  SUBROUTINE NMJO (AR,AI,DR,DI,VR,VI,YR,YI,ZR,ZI,DIMN,N,ROT,EIVEC,
C113                 1 MC)
C114                  INTEGER DIMN,N,EIVEC,ROT,P,Q
C115                  REAL AR(DIMN,DIMN),AI(DIMN,DIMN), DR(DIMN),DI(DIMN)
C116                  REAL VR(DIMN,DIMN),VI(DIMN,DIMN), YR(DIMN),YI(DIMN)
C117                  REAL ZR(DIMN),ZI(DIMN), MC
C118                  DOUBLE PRECISION SUM
C119                  IF (EIVEC .EQ. 0) GO TO 30
C120                  DO 10 I=1,N
C121                  DO 10 J=1,N
C122                  VR(I,J)=0.0
C123                10 VI(I,J)=0.0
C124                  DO 20 P=1,N
C125                20 VR(P,P)=1.0
C126                30 DO 40 P=1,N
C127                  DR(P)=AR(P,P)
C128                  DI(P)=AI(P,P)
C129                  YR(P)=DR(P)
C130                  YI(P)=DI(P)
C131                40 ZR(P), ZI(P)=0.0
C132                  ROT=0
C133                  EPS=N*N*MC
C134                  DO 600 I=1,50
C135                  SUM=0.0
C136                  DO 100 P=1,N-1
C137                  DO 100 Q=P+1,N
C138                  TEMP=SQRT(AR(P,Q)**2+AI(P,Q)**2)+SQRT(AR(Q,P)**2+AI(Q,P)**2)
C139                100 SUM=SUM+DBLE(TEMP)
C140                  SM=SUM
C141                  IF (SM .LE. EPS) GO TO 610
C142                  TRESH=0.0
C143                  IF (I .LT. 4) TRESH=0.2*SM/N**2
C144                  DO 500 P=1,N-1
C145                  DO 500 Q=P+1,N
C146                  BRPQ=(AR(P,Q)+AR(Q,P))/2.0
C147                  BIPQ=(AI(P,Q)-AI(Q,P))/2.0
C148                  CRPQ=(AI(P,Q)+AI(Q,P))/2.0
C149                  CIPQ=(AR(Q,P)-AR(P,Q))/2.0
C150                  E=BRPQ**2+BIPQ**2
C151                  F=CRPQ**2+CIPQ**2
C152                  G=(DR(Q)-DR(P))**2
C153                  H=(DI(Q)-DI(P))**2
C154                  IF (E+G-F-H) 200,210,210
C155                200 IFLAG=2
C156                  G=100.0*SQRT(F)
C157                  IF (I .GT. 4 .AND. ABS(DI(P))+G .EQ. ABS(DI(P))
C158                 1 .AND. ABS(DI(Q))+G .EQ. ABS(DI(Q))) GO TO 410
C159                  IF (G .LE. TRESH) GO TO 420
C160                  E=CRPQ
C161                  F=CIPQ
C162                  EE=BRPQ
C163                  FF=BIPQ
C164                  H=DI(Q)-DI(P)
C165                  GO TO 220
C166                210 IFLAG=1
C167                  G=100.0*SQRT(E)
C168                  IF (I .GT. 4 .AND. ABS(DR(P))+G .EQ. ABS(DR(P))
C169                 1 .AND. ABS(DR(Q))+G .EQ. ABS(DR(Q))) GO TO 410
C170                  IF (G .LE. TRESH) GO TO 420
C171                  E=BRPQ
C172                  F=BIPQ
C173                  EE=CRPQ
C174                  FF=CIPQ
C175                  H=DR(Q)-DR(P)
C176                220 IF (ABS(E)-ABS(F)) 230,240,240
C177                230 T=E/F
C178                  ST=1.0/SQRT(1.0+T*T)
C179                  CT=T*ST
C180                  GO TO 250
C181                240 T=F/E
C182                  CT=1.0/SQRT(1.0+T*T)
C183                  ST=T*CT
C184                250 OMEGA=E*CT+F*ST
C185                  OMEGADASH=EE*CT+FF*ST
C186                  IF (ABS(H)+G .EQ. ABS(H)) GO TO 260
C187                  THETA=0.5*H/OMEGA
C188                  T=1.0/(ABS(THETA)+SQRT(1.0+THETA**2))
C189                  IF (THETA .LT. 0.0) T=-T
C190                  GO TO 270
C191                260 T=OMEGA/H
C192                270 C=1.0/SQRT(1.0+T*T)
C193                  S=T*C
C194                  TAU=S/(1.0+C)
C195                  H=T*OMEGA
C196                  IF (IFLAG .EQ. 1) GO TO 280
C197                  ZI(P)=ZI(P)-H
C198                  ZI(Q)=ZI(Q)+H
C199                  DI(P)=DI(P)-H
C200                  DI(Q)=DI(Q)+H
C201                  E=(C*C-S*S)*(BRPQ*CT+BIPQ*ST)+(DR(P)-DR(Q))*S*C
C202                  F=BIPQ*CT-BRPQ*ST
C203                  BRPQ=E*CT-F*ST
C204                  BIPQ=F*CT+E*ST
C205                  AR(P,Q)=BRPQ
C206                  AI(P,Q)=BIPQ
C207                  AR(Q,P)=BRPQ
C208                  AI(Q,P)=-BIPQ
C209                  H=(DR(P)-DR(Q))*S*S+2.0*S*C*OMEGADASH
C210                  ZR(P)=ZR(P)-H
C211                  ZR(Q)=ZR(Q)+H
C212                  DR(P)=DR(P)-H
C213                  DR(Q)=DR(Q)+H
C214                  GO TO 290
```

```
C215            280 ZR(P)=ZR(P)-H
C216                ZR(Q)=ZR(Q)+H
C217                DR(P)=DR(P)-H
C218                DR(Q)=DR(Q)+H
C219                E=(C+C-S*S)*(CRPQ*CT+CIPQ*ST)+(DI(P)-DI(Q))*S*C
C220                F=CIPQ*CT-CRPQ*ST
C221                CRPQ=E*CT-F*ST
C222                CIPQ=F*CT+E*ST
C223                AR(P,Q)=-CIPQ
C224                AI(P,Q)=CRPQ
C225                AR(Q,P)=CIPQ
C226                AI(Q,P)=CRPQ
C227                H=(DI(P)-DI(Q))*S*S+2.0*S*C*OMEGADASH
C228                ZI(P)=ZI(P)-H
C229                ZI(Q)=ZI(Q)+H
C230                DI(P)=DI(P)-H
C231                DI(Q)=DI(Q)+H
C232            290 IF (P .EQ. 1) GO TO 310
C233                DO 300 J=1,P-1
C234                ARJP=AR(J,P)
C235                AIJP=AI(J,P)
C236                ARJQ=AR(J,Q)
C237                AIJQ=AI(J,Q)
C238                AR(J,P)=ARJP-S*(ARJQ*CT+AIJQ*ST+ARJP*TAU)
C239                AI(J,P)=AIJP-S*(AIJQ*CT-ARJQ*ST+AIJP*TAU)
C240                AR(J,Q)=ARJQ+S*(ARJP*CT-AIJP*ST-ARJQ*TAU)
C241                AI(J,Q)=AIJQ+S*(AIJP*CT+ARJP*ST-AIJQ*TAU)
C242                ARPJ=AR(P,J)
C243                AIPJ=AI(P,J)
C244                ARQJ=AR(Q,J)
C245                AIQJ=AI(Q,J)
C246                AR(P,J)=ARPJ-S*(ARQJ*CT-AIQJ*ST+ARPJ*TAU)
C247                AI(P,J)=AIPJ-S*(AIQJ*CT+ARQJ*ST+AIPJ*TAU)
C248                AR(Q,J)=ARQJ+S*(ARPJ*CT+AIPJ*ST-ARQJ*TAU)
C249            300 AI(Q,J)=AIQJ+S*(AIPJ*CT-ARPJ*ST-AIQJ*TAU)
C250            310 IF (P+1 .GT. Q-1) GO TO 330
C251                DO 320 J=P+1,Q-1
C252                ARPJ=AR(P,J)
C253                AIPJ=AI(P,J)
C254                ARJQ=AR(J,Q)
C255                AIJQ=AI(J,Q)
C256                ARJP=AR(J,P)
C257                AIJP=AI(J,P)
C258                ARQJ=AR(Q,J)
C259                AIQJ=AI(Q,J)
C260                AR(P,J)=ARPJ-S*(ARQJ*CT-AIQJ*ST+ARPJ*TAU)
C261                AI(P,J)=AIPJ-S*(AIQJ*CT+ARQJ*ST+AIPJ*TAU)
C262                AR(J,Q)=ARJQ+S*(ARJP*CT-AIJP*ST-ARJQ*TAU)
C263                AI(J,Q)=AIJQ+S*(AIJP*CT+ARJP*ST-AIJQ*TAU)
C264                AR(J,P)=ARJP-S*(ARJQ*CT+AIJQ*ST+ARJP*TAU)
C265                AI(J,P)=AIJP-S*(AIJQ*CT-ARJQ*ST+AIJP*TAU)
C266                AR(Q,J)=ARQJ+S*(ARPJ*CT+AIPJ*ST-ARQJ*TAU)
C267            320 AI(Q,J)=AIQJ+S*(AIPJ*CT-ARPJ*ST-AIQJ*TAU)
C268            330 IF (Q .EQ. N) GO TO 350
C269                DO 340 J=Q+1,N
C270                ARPJ=AR(P,J)
C271                AIPJ=AI(P,J)
C272                ARQJ=AR(Q,J)
C273                AIQJ=AI(Q,J)
C274                AR(P,J)=ARPJ-S*(ARQJ*CT-AIQJ*ST+ARPJ*TAU)
C275                AI(P,J)=AIPJ-S*(AIQJ*CT+ARQJ*ST+AIPJ*TAU)
C276                AR(Q,J)=ARQJ+S*(ARPJ*CT+AIPJ*ST-ARQJ*TAU)
C277                AI(Q,J)=AIQJ+S*(AIPJ*CT-ARPJ*ST-AIQJ*TAU)
C278                ARJP=AR(J,P)
C279                AIJP=AI(J,P)
C280                ARJQ=AR(J,Q)
C281                AIJQ=AI(J,Q)
C282                AR(J,P)=ARJP-S*(ARJQ*CT+AIJQ*ST+ARJP*TAU)
C283                AI(J,P)=AIJP-S*(AIJQ*CT-ARJQ*ST+AIJP*TAU)
C284                AR(J,Q)=ARJQ+S*(ARJP*CT-AIJP*ST-ARJQ*TAU)
C285            340 AI(J,Q)=AIJQ+S*(AIJP*CT+ARJP*ST-AIJQ*TAU)
C286            350 IF (EIVEC .EQ. 0) GO TO 400
C287                DO 360 J=1,N
C288                VRJP=VR(J,P)
C289                VIJP=VI(J,P)
C290                VRJQ=VR(J,Q)
C291                VIJQ=VI(J,Q)
C292                VR(J,P)=VRJP-S*(VRJQ*CT+VIJQ*ST+VRJP*TAU)
C293                VI(J,P)=VIJP-S*(VIJQ*CT-VRJQ*ST+VIJP*TAU)
C294                VR(J,Q)=VRJQ+S*(VRJP*CT-VIJP*ST-VRJQ*TAU)
C295            360 VI(J,Q)=VIJQ+S*(VIJP*CT+VRJP*ST-VIJQ*TAU)
C296            400 ROT=ROT+1
C297                GO TO 420
C298            410 AR(P,Q),AI(P,Q),AR(Q,P),AI(Q,P)=0.0
C299            420 CONTINUE
C300            500 CONTINUE
C301                DO 510 P=1,N
C302                YR(P)=YR(P)+ZR(P)
C303                YI(P)=YI(P)+ZI(P)
C304                DR(P)=YR(P)
C305                DI(P)=YI(P)
C306            510 ZR(P),ZI(P)=0.0
C307            600 CONTINUE
C308                ROT=-ROT
C309            610 DO 650 P=1,N-1
C310                DO 650 Q=P+1,N
C311                TEMP=ABS(DR(P))-ABS(DR(Q))
C312                IF (TEMP) 620,640,640
```

```
0313              620 SM=DR(P)
0314                  DR(P)=DR(Q)
0315                  DR(Q)=SM
0316                  SM=DI(P)
0317                  DI(P)=DI(Q)
0318                  DI(Q)=SM
0319                  DO 630 I=1,N
0320                  SM=VR(I,P)
0321                  VR(I,P)=VR(I,Q)
0322                  VR(I,Q)=SM
0323                  SM=VI(I,P)
0324                  VI(I,P)=VI(I,Q)
0325              630 VI(I,Q)=SM
0326              640 CONTINUE
0327              650 CONTINUE
0328                  RETURN
0329                  END
```

END OF SEGMENT, LENGTH 2032, NAME NIIJO

```
0330                  SUBROUTINE NORMALISATION (VR,VI,DIMN,N)
0331                  INTEGER DIMN
0332                  REAL VR(DIMN,DIMN),VI(DIMN,DIMN)
0333                  DO 30 J=1,N
0334                  AUX=VR(1,J)**2+VI(1,J)**2
0335                  DO 10 I=2,N
0336                  TEMP=VR(I,J)**2+VI(I,J)**2
0337                  IF (TEMP .GT. AUX) AUX=TEMP
0338               10 CONTINUE
0339                  AUX=SQRT(AUX)
0340                  DO 20 I=1,N
0341                  VR(I,J)=VR(I,J)/AUX
0342               20 VI(I,J)=VI(I,J)/AUX
0343               30 CONTINUE
0344                  RETURN
0345                  END
```

END OF SEGMENT, LENGTH   126, NAME NORMALISATION

```
0346                  SUBROUTINE CHECK (AR,AI,VR,VI,B,Z,DIMN,N)
0347                  REAL AR(DIMN,DIMN),AI(DIMN,DIMN), VR(DIMN,DIMN),VI(DIMN,DIMN)
0348                  REAL B(DIMN), Z(DIMN)
0349                  DOUBLE PRECISION SUMR,SUMI
0350                  DO 40 I=1,N
0351                  DO 30 J=1,N
0352                  SUMR,SUMI=0.0
0353                  DO 20 K=1,N
0354                  SUMR=SUMR+AR(I,K)*VR(K,J)-AI(I,K)*VI(K,J)
0355               20 SUMI=SUMI+AR(I,K)*VI(K,J)+AI(I,K)*VR(K,J)
0356                  B(J)=SUMR
0357               30 Z(J)=SUMI
0358                  DO 40 J=1,N
0359                  AR(I,J)=B(J)
0360               40 AI(I,J)=Z(J)
0361                  DO 80 K=1,N
0362                  DO 70 J=1,N
0363                  SUMR,SUMI=0.0
0364                  DO 60 I=1,N
0365                  SUMR=SUMR+VR(I,J)*AR(I,K)+VI(I,J)*AI(I,K)
0366               60 SUMI=SUMI+VR(I,J)*AI(I,K)-VI(I,J)*AR(I,K)
0367                  B(J)=SUMR
0368               70 Z(J)=SUMI
0369                  DO 80 I=1,N
0370                  AR(I,K)=B(I)
0371               80 AI(I,K)=Z(I)
0372                  RETURN
0373                  END
```

END OF SEGMENT, LENGTH   340, NAME CHECK

```
0374                  SUBROUTINE ELAPSE (E,DAT,NO,K)
0375                  INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
0376                  DATA CALL/2/
0377                  DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
0378                 1 'OCT','NOV','DEC'/
0379                  GO TO (100,200,300), NO
0380            100   CALL MTIME (N)
0381                  E=N
0382                  CALL=3-CALL
0383                  IF (CALL .EQ. 2) GO TO 110
0384                  EE=E
0385                  RETURN
0386            110   E=E-EE
0387                  E=E*1.0E-3
0388                  RETURN
0389            200   CALL DEFBUF (2,8,BUFFER)
0390                  CALL DATE (E)
0391                  WRITE (2,210) E
0392            210   FORMAT (A8)
0393                  READ (2,220) (DAT(I), I=1,3)
0394            220   FORMAT (I2,1X,I2,1X,I2)
0395                  DAT(2)=MONTH(DAT(2))
0396                  E,EE=1.0E-08
0397            230   IF (1.0+EE .EQ. 1.0) GO TO 240
0398                  E=EE
0399                  EE=EE/2.0
0400                  GO TO 230
0401            240   RETURN
0402            300   IF (K) 370,370,310
0403            310   IF (E .GE. 60.0) GO TO 330
0404                  WRITE (6,320) E
0405            320   FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
0406                  GO TO 300
0407            330   IF (E .GE. 120.0) GO TO 350
0408                  E=E-60.0
0409                  WRITE (6,340) E
0410            340   FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
0411                 1)
0412                  GO TO 300
0413            350   I=E/60.0
0414                  E=E-I*60.0
0415                  WRITE (6,360) I, E
0416            360   FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
0417                 1ONDS')
0418                  GO TO 300
0419            370   WRITE (6,380) E
0420            380   FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
0421            390   RETURN
0422                  END

END OF SEGMENT, LENGTH   182, NAME  ELAPSE
```

```
C423                    SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
C424                    INTEGER DIMN,DIMP,P,WIDTH,STREAM
C425                    REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
C426                    DATA FRMT1/3H(       ,1HN,8HX,1P     ,1HN,1HE,8H)       /
C427                    DATA FRMT2/3H(       ,1HN,8HX,1P     ,1HN,1HE,8H)       /
C428                    IF (IFLAG) 10,200,10
C429                 10 CALL DEBUG (2,16,BUFFER)
C430                    FRMT1(5), FRMT2(5)=FIELD
C431                    WRITE (2,20) FIELD
C432                 20 FORMAT (A6)
C433                    READ (2,30) WIDTH
C434                 30 FORMAT (1X,I2)
C435                    IF (P*WIDTH-LENGTH) 100,100,40
C436                 40 N1=LENGTH/WIDTH
C437                    N2=P/N1
C438                    LAST=P-N1*N2
C439                    IF (IFLAG) 50,999,60
C440                 50 N3=1
C441                    N4=1
C442                    GO TO 70
C443                 60 N3=(LENGTH-N1*WIDTH)/2+1
C444                    N4=(LENGTH-LAST*WIDTH)/2+1
C445                 70 WRITE (2,80) N3,N1,N4,LAST
C446                 80 FORMAT (4I4)
C447                    READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
C448                 90 FORMAT (4A4)
C449                    GO TO 300
C450                100 N2=0
C451                    IF (IFLAG) 110,999,120
C452                110 N3=1
C453                    GO TO 130
C454                120 N3=(LENGTH-P*WIDTH)/2+1
C455                130 WRITE (2,140) N3,P
C456                140 FORMAT (2I4)
C457                    READ (2,150) FRMT1(2), FRMT1(4)
C458                150 FORMAT (2A4)
C459                    GO TO 210
C460                200 IF (N2) 999,210,300
C461                210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
C462                    RETURN
C463                300 N2N1=N2*N1
C464                    IF (N-1) 999,400,310
C465                310 DO 330 K=1,N2N1,N1
C466                    KK=K+N1-1
C467                    WRITE (STREAM,320) K, KK
C468                320 FORMAT (1H ,/,' COLUMNS',I3,' TO',I3,' ARE:')
C469                330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
C470                    IF (LAST-1) 370,340,360
C471                340 WRITE (STREAM,350) P
C472                350 FORMAT (1H ,/,' COLUMN',I3,' IS:')
C473                    WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
C474                    GO TO 370
C475                360 KK=N2N1+1
C476                    WRITE (STREAM,320) KK,P
C477                    WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
C478                370 RETURN
C479                400 DO 410 K=1,N2N1,N1
C480                    KK=K+N1-1
C481                410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
C482                    IF (LAST-1) 440,420,430
C483                420 WRITE (STREAM,FRMT2) A(1,P)
C484                    GO TO 440
C485                430 KK=N2N1+1
C486                    WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
C487                440 RETURN
C488                999 STOP MXOP
C489                    END
```

END OF SEGMENT, LENGTH 427, NAME MXOP

```
C490                    SUBROUTINE INPUT (AR,AI,DIMN,N,NO,EIVEC,RES,NORM,FIELD)
C491                    INTEGER DIMN,EIVEC,RES
C492                    REAL AR(DIMN,DIMN),AI(DIMN,DIMN)
C493                    READ (5,10) N,FIELD,EIVEC,RES,NORM
C494                 10 FORMAT (I2,A6,3I1)
C495                    GO TO (100,200,300,400,500,600,700,800,900), NO
C496                100 CONTINUE
C497                    READ (5,110) ((AR(I,J),AI(I,J), J=1,N), I=1,N)
C498                110 FORMAT (2G0.0)
C499                    RETURN
C500                200 CONTINUE
C501                    GO TO 100
C502                300 CONTINUE
C503                    GO TO 100
C504                400 CONTINUE
C505                    GO TO 100
C506                500 CONTINUE
C507                    GO TO 100
C508                600 CONTINUE
C509                    GO TO 100
C510                700 CONTINUE
C511                    GO TO 100
C512                800 CONTINUE
C513                    GO TO 100
C514                900 CONTINUE
C515                    GO TO 100
C516                    END
```

END OF SEGMENT, LENGTH 128, NAME INPUT

A JACOBI-LIKE METHOD FOR OBTAINING THE EIGENSOLUTIONS OF A NXN NORMAL MATRIX

THIS PROGRAM WAS RUN ON 22 AUG 74
AND EXECUTION STARTED AT 17/32/00

EXAMPLE NUMBER 2

THE ORIGINAL 4X 4 MATRIX A IS GIVEN BELOW;

REAL PART

2.2393667092E-01	-1.4101642897E-01	-1.0050767763E-01	1.6600549880E-01
-6.4359278054E-02	6.2802085617E-01	1.1943912588E-02	-1.3711832447E-01
-4.7358443491E-02	5.4043649252E-02	3.8954842098E-01	-7.3896741436E-03
1.7744905748E-01	-2.1309514388E-01	-1.3259464256E-01	4.3826830401E-01

IMAGINARY PART

4.8038636393E-01	-9.1160043599E-02	2.1039442730E-01	2.3505447361E-02
2.2123232547E-01	4.9183378858E-01	-4.4002797075E-03	1.7532788095E-01
-8.8968702480E-02	1.9443069768E-01	4.9935598560E-01	1.7402323924E-01
1.1275091512E-01	2.7699137816E-02	-5.5888827189E-03	5.2763291202E-01

23 JACOBI-LIKE ROTATIONS HAVE BEEN USED

THE EIGENVALUES OF A ARE GIVEN BELOW;

REAL PART

8.7424234466E-01	4.9051844312E-01	2.9933140445E-01	1.5702039847E-02

IMAGINARY PART

3.9311659081E-01	4.0306134132E-01	7.6137083077E-01	4.4866028722E-01

THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIVEN BELOW;

REAL PART

-3.7531880771E-01	-4.5538921493E-02	7.0829571846E-01	9.9987117327E-01
9.9257106670E-01	-4.2325197420E-01	8.1325553462E-01	-6.3187473469E-02
1.4533296153E-01	9.5296631331E-01	7.1427963564E-01	8.0788787982E-03
-4.4270737623E-01	-3.4011639292E-01	9.9996628872E-01	-4.9632302525E-01

IMAGINARY PART

-3.6447438765E-01	3.7125964408E-01	1.0797693578E-02	1.6051071054E-02
1.1321013838E-01	3.5792143000E-01	-3.3259861376E-01	-2.6693250465E-01
3.4622447423E-01	-3.0307623745E-01	-4.8029459117E-01	5.3387104229E-01
-4.8098138599E-01	-6.5221350999E-01	-8.2122724747E-03	-1.4361310420E-01

THE RESIDUAL MATRIX V"AV IS GIVEN BELOW;

REAL PART

8.7424234464E-01	1.0958365342E-11	-3.2287061913E-11	3.0411229091E-12
-3.0093446185E-12	4.9051844313E-01	-1.3983481040E-11	-1.2807532812E-12
2.5124791136E-11	-2.7206681352E-11	2.9933140444E-01	-5.9969806898E-12
2.4802382370E-12	6.8283156907E-12	-8.1286088971E-12	1.5702039849E-02

IMAGINARY PART

3.9311659081E-01	-7.2759576142E-12	4.0922761380E-11	-1.0089706848E-12
8.2101435105E-12	4.0306134132E-01	-1.1368683772E-12	5.2011728258E-12
4.0671466195E-11	1.0908403043E-11	7.6137083077E-01	-8.3973011554E-12
1.9489831640E-12	8.7680973593E-12	-1.1141310097E-11	4.4866028721E-01

EXECUTION TIME WAS 0.216 SECONDS

APPENDIX 4

A JACOBI PROGRAM FOR GENERAL MATRICES

```
C021                MASTER
C022                INTEGER DIMN,ROT,RES,LINE(84),DATE(3)
C023                REAL AR(12,12),AI(12,12),WR(12,12),WI(12,12),VR(12,12),VI(12,12)
C024                REAL AAR(12,12),AAI(12,12),EN(12),EM(12),DR(12),DI(12),MC
C025                DATA LINE/84*1H*/
C026                K=1
C027                CALL ELAPSE (E,DATE,2,K)
C028                MC=E
C029                READ (5,10) NUMBER,DIMN,LENGTH
C030             10 FORMAT (2I2,I3)
C031                IF (LENGTH .LT. 120) K=-1
C032                DO 500 NO=1,NUMBER
C033                CALL INPUT (AR,AI,DIMN,N,NO,RES,FIELD)
C034                DO 20 I=1,N
C035                DO 20 J=1,N
C036                AAR(I,J)=AR(I,J)
C037             20 AAI(I,J)=AI(I,J)
C038                CALL TIME (T)
C039                IF (K) 120,999,100
C040            100 WRITE (6,110) (LINE(I), I=1,84), (DATE(I), I=1,3), T, NO
C041            110 FORMAT (1H1,//,18X,'A JACOBI-LIKE METHOD FOR OBTAINING THE EIGENSO
C042               1LUTION OF A NXN GENERAL COMPLEX MATRIX',/,18X,84A1,///,45X,'THIS P
C043               2ROGRAM WAS RUN ON',I3,1X,A3,1X,I2,/,45X,'AND EXECUTION STARTED AT'
C044               3,A9,//,54X,'EXAMPLE NUMBER',I3)
C045                GO TO 140
C046            120 WRITE (6,130) (LINE(I), I=1,49), (DATE(I), I=1,3), T, NO
C047            130 FORMAT (1H ,//////,' A JACOBI-LIKE METHOD FOR GENERAL COMPLEX MATR
C048               1ICES',/,1X,49A1,///,' DATE:',I3,1X,A3,1X,I2,5X,'TIME:',A9,5X,'EXAMP
C049               2LE NUMBER',I3)
C050            140 WRITE (6,150) N,N
C051            150 FORMAT (1H ,////,' THE ORIGINAL',I3,'X',I2,' MATRIX A IS GIVEN BEL
C052               1OW:')
C053                WRITE (6,160)
C054            160 FORMAT (1H ,/,' REAL PART')
C055                CALL MXOP (AR,DIMN,DIMN,N,N,FIELD,LENGTH,6,K)
C056                WRITE (6,161)
C057            161 FORMAT (1H ,/,' IMAGINARY PART')
C058                CALL MXOP (AI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C059                CALL ELAPSE (E,DATE,1,K)
C060                CALL GLJO (AR,AI,WR,WI,VR,VI,EN,DR,DI,DIMN,N,MC,ROT)
C061                CALL ELAPSE (E,DATE,1,K)
C062                IF (RES .EQ. 1) CALL CHECK (AAR,AAI,WR,WI,VR,VI,EN,EM,DIMN,N)
C063                IF (ROT) 200,250,250
C064            200 ROT=-ROT
C065                IF (K) 230,999,210
C066            210 WRITE (6,220)
C067            220 FORMAT (1H ,////,32X,'** WARNING - A MAXIMUM OF 50 SWEEPS HAVE BEE
C068               1N COMPLETED **')
C069                GO TO 250
C070            230 WRITE (6,240)
C071            240 FORMAT (1H ,////,' ** WARNING - A MAXIMUM OF 50 SWEEPS HAVE BEEN C
C072               1OMPLETED **')
C073            250 IF (K) 280,999,260
C074            260 WRITE (6,270) ROT
C075            270 FORMAT (1H ,////,36X,I4,' JACOBI-LIKE ROTATIONS AND SHEARS HAVE BE
C076               1EN USED')
C077                GO TO 300
C078            280 WRITE (6,290) ROT
C079            290 FORMAT (1H ,////,1X,I4,' JACOBI-LIKE ROTATIONS AND SHEARS HAVE BEE
C080               1N USED')
C081            300 WRITE (6,310)
C082            310 FORMAT (1H ,////,' THE EIGENVALUES OF A ARE GIVEN BELOW;')
C083                WRITE (6,160)
C084                CALL MXOP (DR,1,DIMN,1,N,FIELD,LENGTH,6,0)
C085                WRITE (6,161)
C086                CALL MXOP (DI,1,DIMN,1,N,FIELD,LENGTH,6,0)
C087                WRITE (6,320)
C088            320 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING RIGHT-HAND EIGENVEC
C089               1TORS IS GIVEN BELOW;')
C090                WRITE (6,160)
C091                CALL MXOP (VR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C092                WRITE (6,161)
C093                CALL MXOP (VI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C094                WRITE (6,330)
C095            330 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING LEFT-HAND EIGENVECT
C096               1ORS IS GIVEN BELOW;')
C097                WRITE (6,160)
C098                CALL MXOP (AR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C099                WRITE (6,161)
C100                CALL MXOP (AI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C101                IF (RES .EQ. 0) GO TO 400
C102                WRITE (6,340)
C103            340 FORMAT (1H ,////,' THE RESIDUAL MATRIX W''AV IS GIVEN BELOW;')
C104                WRITE (6,160)
C105                CALL MXOP (AAR,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C106                WRITE (6,161)
C107                CALL MXOP (AAI,DIMN,DIMN,N,N,FIELD,LENGTH,6,0)
C108            400 CALL ELAPSE (E,DATE,3,K)
C109                IF (K) 430,999,410
C110            410 WRITE (6,420) (LINE(I), I=1,60)
C111            420 FORMAT (1H ,////,31X,60A1)
C112                GO TO 500
C113            430 WRITE (6,440) (LINE(I), I=1,40)
C114            440 FORMAT (1H ,////,1X,40A1)
C115            500 CONTINUE
C116                STOP OK
C117            999 STOP NONM
C118                END

END OF SEGMENT, LENGTH   449, NAME  NONM
```

```
C119                SUBROUTINE GLJO (AR,AI,WR,WI,VR,VI,EN,DDR,DDI,DIMN,N,MC,ROT)
C120                INTEGER DIMN,ROT
C121                REAL AR(DIMN,DIMN),AI(DIMN,DIMN), WR(DIMN,DIMN),WI(DIMN,DIMN)
C122                REAL VR(DIMN,DIMN),VI(DIMN,DIMN), EN(DIMN), DDR(DIMN),DDI(DIMN
C123                REAL MAX,ND,MC,ISW,MC
C124                MARK=0
C125                ROT=0
C126                EPS=10.0*N*(N-1)*MC
C127                DO 10 I=1,N-1
C128                WR(I,I),VR(I,I)=1.0
C129                WI(I,I),VI(I,I)=0.0
C130                DO 10 J=I+1,N
C131             10 WR(I,J),WR(J,I),WI(I,J),WI(J,I),
C132               1 VR(I,J),VR(J,I),VI(I,J),VIKJ,I)=0.0
C133                WR(N,N),VR(N,N)=1.0
C134                WI(N,N),VI(N,N)=0.0
C135                DO 440 IT=1,50
C136                IF (MARK .EQ. 1) GO TO 450
C137                TAU=0.0
C138                DO 110 K=1,N
C139                TEM=0.0
C140                DO 100 I=1,N
C141                IF (I .EQ. K) GO TO 100
C142                ARIK=AR(I,K)
C143                AIIK=AI(I,K)
C144                TEM=ARIK*ARIK+AIIK*AIIK+TEM
C145            100 CONTINUE
C146                TAU=TAU+TEM
C147            110 CONTINUE
C148                IF (TAU .LE. EPS) GO TO 450
C149                MARK=1
C150                DO 430 K=1,N-1
C151                DO 430 M=K+1,N
C152                HJ,HR,HI,G=0.0
C153                DO 310 I=1,N
C154                IF (I .EQ. K .OR. I .EQ. M) GO TO 300
C155                ARKI=AR(K,I)
C156                AIKI=AI(K,I)
C157                ARMI=AR(M,I)
C158                AIMI=AI(M,I)
C159                ARIK=AR(I,K)
C160                AIIK=AI(I,K)
C161                ARIM=AR(I,M)
C162                AIIM=AI(I,M)
C163                HR=HR+ARKI*ARMI+AIKI*AIMI-ARIK*ARIM-AIIK*AIIM
C164                HI=HI+AIKI*ARMI-ARKI*AIMI-ARIK*AIIM+AIIK*ARIM
C165                TE= ARIK*ARIK+AIIK*AIIK+ARMI*ARMI+AIMI*AIMI
C166                TEE=ARIM*ARIM+AIIM*AIIM+ARKI*ARKI+AIKI*AIKI
C167                G=G+TE+TEE
C168                HJ=HJ-TE+TEE
C169            300 CONTINUE
C170            310 CONTINUE
C171                BR=AR(K,M)+AR(M,K)
C172                BI=AI(K,M)+AI(M,K)
C173                ER=AR(K,M)-AR(M,K)
C174                EI=AI(K,M)-AI(M,K)
C175                DR=AR(K,K)-AR(M,M)
C176                DI=AI(K,K)-AI(M,M)
C177                TE =BR*BR+EI*EI+DR*DR
C178                TEE=BI*BI+ER*ER+DI*DI
C179                IF (TE .LT. TEE) GO TO 320
C180                ISW=1.0
C181                C=BR
C182                S=EI
C183                D=DR
C184                DE=DI
C185                ROOT2=SQRT(TE)
C186                GO TO 330
C187            320 ISW=-1.0
C188                C=BI
C189                S=-ER
C190                D=DI
C191                DE=DR
C192                ROOT2=SQRT(TEE)
C193            330 ROOT1=SQRT(S*S+C*C)
C194                SIG=1.0
C195                IF (D .LT. 0.0) SIG=-1.0
C196                SA=0.0
C197                CA=1.0
C198                IF (C .LT. 0.0) CA=-1.0
C199                IF (ROOT1 .GE. EPS) GO TO 350
C200                SX,SA=0.0
C201                CX,CA=1.0
C202                E=ER
C203                B=BI
C204                IF (ISW .GT. 0.0) GO TO 3402
C205                E=EI
C206                B=-BR
C207            340 ND=D*D+DE*DE
C208                GO TO 370
C209            350 IF (ABS(S) .LE. EPS) GO TO 360
C210                CA=C/ROOT1
C211                SA=S/ROOT1
C212            360 COT2X=D/ROOT1
C213                COTX=COT2X+(SIG*SQRT(1.0+COT2X*COT2X))
C214                SX=SIG/SQRT(1.0+COTX*COTX)
C215                CX=SX*COTX
C216                ETA=(ER*BR+BI*EI)/ROOT1
C217                TSE=(BR*BI-ER*EI)/ROOT1
C218                TE=SIG*(-ROOT1*DE+TSE*D)/ROOT2
C219                TEE=(D*DE+ROOT1*TSE)/ROOT2
C220                ND=ROOT2*ROOT2+TEE*TEE
```

```
0221                TEE=HJ*CX*SX
0222                COS2A=CA*CA-SA*SA
0223                SIN2A=2.0*CA*SA
0224                TEM=HR*COS2A+HI*SIN2A
0225                TEP=HI*COS2A-HR*SIN2A
0226                HR=CX*CX*HR-SX*SX*TEM-CA*TEE
0227                HI=CX*CX*HI+SX*CX*TEP-SA*TEE
0228                B=ISH*TE*CA+ETA*SA
0229                E=CA*ETA-ISH*TE*SA
0230          370   S=HR-SIG*ROOT2*E
0231                C=HI-SIG*ROOT2*B
0232                ROOT=SQRT(C*C+S*S)
0233                IF (ROOT .GE. EPS) GO TO 380
0234                CB,CH=1.0
0235                SB,SH=0.0
0236                GO TO 390
0237          380   CB=-C/ROOT
0238                SB=S/ROOT
0239                TEE=CB*B-E*SB
0240                NC=TEE*TEE
0241                TANH=ROOT/(G+2.0*(NC+ND))
0242                CH=1.0/SQRT(1.0-TANH*TANH)
0243                SH=CH*TANH
0244          390   TEM=SX*SH*(SA*CB-SB*CA)
0245                C1R=CX*CH-TEM
0246                C2R=CX*CH+TEM
0247                C1I,C2I=-SX*SH*(CA*CB+SA*SB)
0248                TEP=SX*CH*CA
0249                TEM=CX*SH*SB
0250                S1R=TEP-TEM
0251                S2R=-TEP-TEM
0252                TEP=SX*CH*SA
0253                TEM=CX*SH*CB
0254                S1I=TEP+TEM
0255                S2I=TEP-TEM
0256                TEM=SQRT(S1R*S1R+S1I*S1I)
0257                TEP=SQRT(S2R*S2R+S2I*S2I)
0258                IF (TEM .LT. EPS .AND. TEP .LT. EPS) GO TO 420
0259                MARK=0
0260                DO 400 I=1,N
0261                ARKI=AR(K,I)
0262                ARMI=AR(M,I)
0263                AIKI=AI(K,I)
0264                AIMI=AI(M,I)
0265                AR(K,I)=C1R*ARKI-C1I*AIKI+S1R*ARMI-S1I*AIMI
0266                AI(K,I)=C1R*AIKI+C1I*ARKI+S1R*AIMI+S1I*ARMI
0267                AR(M,I)=S2R*ARKI-S2I*AIKI+C2R*ARMI-C2I*AIMI
0268                AI(M,I)=S2R*AIKI+S2I*ARKI+C2R*AIMI+C2I*ARMI
0269                WRIK=WR(I,K)
0270                WRIM=WR(I,M)
0271                WIIK=WI(I,K)
0272                WIIM=WI(I,M)
0273                WR(I,K)=C1R*WRIK-C1I*WIIK+S1R*WRIM-S1I*WIIM
0274                WI(I,K)=C1R*WIIK+C1I*WRIK+S1R*WIIM+S1I*WRIM
0275                WR(I,M)=S2R*WRIK-S2I*WIIK+C2R*WRIM-C2I*WIIM
0276          400   WI(I,M)=S2R*WIIK+S2I*WRIK+C2R*WIIM+C2I*WRIM
0277                DO 410 I=1,N
0278                ARIK=AR(I,K)
0279                ARIM=AR(I,M)
0280                AIIK=AI(I,K)
0281                AIIM=AI(I,M)
0282                AR(I,K)=C2R*ARIK-C2I*AIIK-S2R*ARIM+S2I*AIIM
0283                AI(I,K)=C2R*AIIK+C2I*ARIK-S2R*AIIM-S2I*ARIM
0284                AR(I,M)=-S1R*ARIK+S1I*AIIK+C1R*ARIM-C1I*AIIM
0285                AI(I,M)=-S1R*AIIK-S1I*ARIK+C1R*AIIM+C1I*ARIM
0286                VRIK=VR(I,K)
0287                VRIM=VR(I,M)
0288                VIIK=VI(I,K)
0289                VIIM=VI(I,M)
0290                VR(I,K)=C2R*VRIK-C2I*VIIK-S2R*VRIM+S2I*VIIM
0291                VI(I,K)=C2R*VIIK+C2I*VRIK-S2R*VIIM-S2I*VRIM
0292                VR(I,M)=-S1R*VRIK+S1I*VIIK+C1R*VRIM-C1I*VIIM
0293          410   VI(I,M)=-S1R*VIIK-S1I*VRIK+C1R*VIIM+C1I*VRIM
0294          420   CONTINUE
0295                ROT=ROT+1
0296          430   CONTINUE
0297          440   CONTINUE
0298                ROT=-ROT
0299          450   DO 460 I=1,N
0300                DDR(I)=AR(I,I)
0301          460   DDI(I)=AI(I,I)
0302                DO 640 I=1,N
0303                DR=DDR(I)
0304                DI=DDI(I)
0305          600   EN(I)=DR*DR+DI*DI
0306                DO 640 K=1,M-1
0307                DO 640 L=K+1,N
0308                TEM=EN(K)-EN(L)
0309                IF (TEM) 610,630,630
0310          610,  TEM=EN(K)
0311                EN(K)=EN(L)
0312                EN(L)=TEM
0313                TEM=DDR(K)
0314                DDR(K)=DDR(L)
0315                DDR(L)=TEM
0316                TEM=DDI(K)
0317                DDI(K)=DDI(L)
0318                DDI(L)=TEM
```

```
0319                    DO 620 I=1,N
0320                    TEM=VR(I,K)
0321                    VR(I,K)=VR(I,L)
0322                    VR(I,L)=TEM
0323                    TEM=VI(I,K)
0324                    VI(I,K)=VI(I,L)
0325                    VI(I,L)=TEM
0326                    TEM=WR(I,K)
0327                    WR(I,K)=WR(I,L)
0328                    WR(I,L)=TEM
0329                    TEM=WI(I,K)
0330                    WI(I,K)=WI(I,L)
0331              620   WI(I,L)=TEM
0332              630   CONTINUE
0333              640   CONTINUE
0334                    RETURN
0335                    END
```

END OF SEGMENT, LENGTH 1707, NAME GLJO

```
0336                    SUBROUTINE CHECK (AR,AI,WR,WI,VR,VI,EN,EM,DIMN,N)
0337                    INTEGER DIMN
0338                    REAL AR(DIMN,DIMN),AI(DIMN,DIMN), WR(DIMN,DIMN),WI(DIMN,DIMN)
0339                    REAL VR(DIMN,DIMN),VI(DIMN,DIMN), EN(DIMN),EM(DIMN)
0340                    DOUBLE PRECISION SUMR,SUMI
0341                    DO 40 I=1,N
0342                    DO 30 J=1,N
0343                    SUMR,SUMI=0.0
0344                    DO 20 K=1,N
0345                    SUMR=SUMR+AR(I,K)*VR(K,J)-AI(I,K)*VI(K,J)
0346               20   SUMI=SUMI+AR(I,K)*VI(K,J)+AI(I,K)*VR(K,J)
0347                    EN(J)=SUMR
0348               30   EM(J)=SUMI
0349                    DO 40 J=1,N
0350                    AR(I,J)=EN(J)
0351               40   AI(I,J)=EM(J)
0352                    DO 80 J=1,N
0353                    DO 70 I=1,N
0354                    SUMR,SUMI=0.0
0355                    DO 60 K=1,N
0356                    SUMR=SUMR+WR(K,I)*AR(K,J)-WI(K,I)*AI(K,J)
0357               60   SUMI=SUMI+WR(K,I)*AI(K,J)+WI(K,I)*AR(K,J)
0358                    EN(I)=SUMR
0359               70   EM(I)=SUMI
0360                    DO 80 I=1,N
0361                    AR(I,J)=EN(I)
0362               80   AI(I,J)=EM(I)
0363                    RETURN
0364                    END
```

END OF SEGMENT, LENGTH 350, NAME CHECK

```
0365                    SUBROUTINE ELAPSE (E,DAT,NO,K)
0366                    INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
0367                    DATA CALL/2/
0368                    DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
0369                   1 'OCT','NOV','DEC'/
0370                    GO TO (100,200,300), NO
0371              100   CALL MTIME (N)
0372                    E=N
0373                    CALL=3-CALL
0374                    IF (CALL .EQ. 2) GO TO 110
0375                    EE=E
0376                    RETURN
0377              110   E=E-EE
0378                    E=E*1.0E-3
0379                    RETURN
0380              200   CALL DEFBUF (2,8,BUFFER)
0381                    CALL DATE (E)
0382                    WRITE (2,210) E
0383              210   FORMAT (A8)
0384                    READ (2,220) (DAT(I), I=1,3)
0385              220   FORMAT (I2,1X,I2,1X,I2)
0386                    DAT(2)=MONTH(DAT(2))
0387                    E,EE=1.0E-10
0388              230   IF (1.0+EE .EQ. 1.0) GO TO 240
0389                    E=EE
0390                    EE=EE/2.0
0391                    GO TO 230
0392              240   RETURN
0393              300   IF (K) 320,320,310
0394              310   IF (E .GE. 60.0) GO TO 330
0395                    WRITE (6,320) E
0396              320   FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
0397                    GO TO 300
0398              330   IF (E .GE. 120.0) GO TO 350
0399                    E=E-60.0
0400                    WRITE (6,340) E
0401              340   FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
0402                   1)
0403                    GO TO 300
0404              350   I=E/60.0
0405                    E=E-I*60.0
0406                    WRITE (6,360) I, E
0407              360   FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
0408                   1ONDS')
0409                    GO TO 300
0410              370   WRITE (6,380) E
0411              380   FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
0412              390   RETURN
0413                    END
```

END OF SEGMENT, LENGTH 182, NAME ELAPSE

```
C414                    SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
C415                    INTEGER DIMN,DIMP,P,WIDTH,STREAM
C416                    REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
C417                    DATA FRMT1/3H(       ,1HN,8HX,1P     ,1HN,1HE,8H)       /
C418                    DATA FRMT2/2H(       ,1HN,8HX,1P     ,1HN,1HE,8H)       /
C419                    IF (IFLAG) 10,200,10
C420                 10 CALL DEFBUF (2,16,BUFFER)
C421                    FRMT1(5), FRMT2(5)=FIELD
C422                    WRITE (2,20) FIELD
C423                 20 FORMAT (A6)
C424                    READ (2,30) WIDTH
C425                 30 FORMAT (1X,I2)
C426                    IF (P*WIDTH-LENGTH) 100,100,40
C427                 40 N1=LENGTH/WIDTH
C428                    N2=P/N1
C429                    LAST=P-N1*N2
C430                    IF (IFLAG) 50,999,60
C431                 50 N3=1
C432                    N4=1
C433                    GO TO 70
C434                 60 N3=(LENGTH-N1*WIDTH)/2+1
C435                    N4=(LENGTH-LAST*WIDTH)/2+1
C436                 70 WRITE (2,80) N3,N1,N4,LAST
C437                 80 FORMAT (4I4)
C438                    READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
C439                 90 FORMAT (4A4)
C440                    GO TO 300
C441                100 N2=0
C442                    IF (IFLAG) 110,999,120
C443                110 N3=1
C444                    GO TO 130
C445                120 N3=(LENGTH-P*WIDTH)/2+1
C446                130 WRITE (2,140) N3,P
C447                140 FORMAT (2I4)
C448                    READ (2,150) FRMT1(2), FRMT1(4)
C449                150 FORMAT (2A4)
C450                    GO TO 210
C451                200 IF (N2) 999,210,300
C452                210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
C453                    RETURN
C454                300 N2N1=N2*N1
C455                    IF (N-1) 999,400,310
C456                310 DO 330 K=1,N2N1,N1
C457                    KK=K+N1-1
C458                    WRITE (STREAM,320) K, KK
C459                320 FORMAT (1H ,/,' COLUMNS',I3,' TO',I3,' ARE:')
C460                330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
C461                    IF (LAST-1) 370,340,360
C462                340 WRITE (STREAM,350) P
C463                350 FORMAT (1H ,/,' COLUMN',I3,' IS:')
C464                    WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
C465                    GO TO 370
C466                360 KK=N2N1+1
C467                    WRITE (STREAM,320) KK,P
C468                    WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
C469                370 RETURN
C470                400 DO 410 K=1,N2N1,N1
C471                    KK=K+N1-1
C472                410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
C473                    IF (LAST-1) 440,420,430
C474                420 WRITE (STREAM,FRMT2) A(1,P)
C475                    GO TO 440
C476                430 KK=N2N1+1
C477                    WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
C478                440 RETURN
C479                999 STOP MXOP
C480                    END

END OF SEGMENT, LENGTH   427, NAME  MXOP
```

```
0481                    SUBROUTINE INPUT (AR,AI,DIMN,N,NO,RES,FIELD)
0482                    INTEGER DIMN,RES
0483                    REAL AR(DIMN,DIMN),AI(DIMN,DIMN)
0484                    READ (5,10) N,FIELD,RES
0485                 10 FORMAT (I2,A6,I1)
0486                    GO TO (100,200,300,400,500,600,700,800,900), NO
0487                100 CONTINUE
0488                    DO 110 J=1,N
0489                    DO 110 I=1,N
0490                110 AI(I,J)=0.0
0491                    READ (5,120) ((AR(I,J), J=1,N), I=1,N)
0492                120 FORMAT (60.0)
0493                    RETURN
0494                200 CONTINUE
0495                    GO TO 100
0496                300 CONTINUE
0497                    GO TO 100
0498                400 CONTINUE
0499                    GO TO 100
0500                500 CONTINUE
0501                    GO TO 100
0502                600 CONTINUE
0503                    READ (5,610) ((AR(I,J),AI(I,J), J=1,N), I=1,N)
0504                610 FORMAT (2G0.0)
0505                    RETURN
0506                700 CONTINUE
0507                    GO TO 600
0508                800 CONTINUE
0509                    DO 810 J=2,N
0510                    DO 810 I=1,N
0511                    AR(I,J),AI(I,J)=0.0
0512                810 AR(J-1,J)=1.0
0513                    DO 820 I=1,N
0514                    AR(I,1)=-1.0
0515                820 AI(I,1)=0.0
0516                    RETURN
0517                900 CONTINUE
0518                    GO TO 100
0519                    END

END OF SEGMENT, LENGTH   219, NAME  INPUT
```

A JACOBI-LIKE METHOD FOR OBTAINING THE EIGENSOLUTION OF A NXN GENERAL COMPLEX MATRIX

THIS PROGRAM WAS RUN ON 22 Aug 74
AND EXECUTION STARTED AT 17/34/22

EXAMPLE NUMBER 6

THE ORIGINAL 4X 4 MATRIX A IS GIVEN BELOW;

REAL PART

5.00000000000E 00	5.00000000000E 00	-6.00000000000E 00	-7.00000000000E 00
3.00000000000E 00	6.00000000000E 00	-5.00000000000E 00	-8.00000000000E 00
2.00000000000E 00	3.00000000000E 00	-1.00000000000E 00	-5.00000000000E 00
1.00000000000E 00	2.00000000000E 00	-3.00000000000E 00	0.00000000000E-01

IMAGINARY PART

9.00000000000E 00	5.00000000000E 00	-6.00000000000E 00	-7.00000000000E 00
3.00000000000E 00	1.00000000000E 01	-5.00000000000E 00	-8.00000000000E 00
2.00000000000E 00	3.00000000000E 00	-3.00000000000E 00	-5.00000000000E 00
1.00000000000E 00	2.00000000000E 00	-3.00000000000E 00	4.00000000000E 00

54 JACOBI-LIKE ROTATIONS AND SHEARS HAVE BEEN USED

THE EIGENVALUES OF A ARE GIVEN BELOW;

REAL PART

4.00000000005E 00	3.00000000007E 00	2.00000000003E 00	1.00000000006E 00

IMAGINARY PART

8.00000000024E 00	7.00000000017E 00	6.00000000017E 00	5.00000000012E 00

THE MATRIX OF CORRESPONDING RIGHT-HAND EIGENVECTORS IS GIVEN BELOW;

REAL PART

1.02273513332E 00	1.08745467566E 00	8.73190213986E-01	1.75023786646E 00
1.02273512976E 00	1.08745467586E 00	1.74638042440E 00	8.75118968506E-01
1.02273513936E 00	3.04860119776E-10	8.73190212656E-01	8.75118968796E-01
-6.65955766846E-08	1.08745467556E 00	8.73190214596E-01	8.75118897136E-01

IMAGINARY PART

1.20840047206E-01	-2.30757877076E-01	3.73503376866E-01	-5.65872536156E-01
1.20840052856E-01	-2.30757876756E-01	7.47006765706E-01	-2.82936285086E-01
1.20840053016E-01	7.24577993096E-11	3.73503380926E-01	-2.82936280706E-01
-5.96066439616E-09	-2.30757876826E-01	3.73503381046E-01	-2.82936259796E-01

THE MATRIX OF CORRESPONDI'IG LEFT-HAND EIGENVECTORS IS GIVEN BELOW:

REAL PART

9.64308085446E-01	8.79955196126E-01	-9.68096674786E-01	6.87358167266E-08
9.64308164226E-01	8.79955196356E-01	-1.14117469816E-09	-1.03455883046E 00
-9.64308164026E-01	-1.73991039266E 00	9.68096676046E-01	1.03455882946E 00
-1.92861624966E 00	-8.79955196296E-01	9.68096675796E-01	1.03455876166E 00

IMAGINARY PART

-1.13936665646E-01	1.86726488406E-01	4.14099219496E-01	1.50823712136E-08
-1.13936672266E-01	1.86726488646E-01	-5.03672043486E-09	-3.34485067456E-01
1.13936672386E-01	-3.73452977086E-01	-4.14099214326E-01	3.34485072036E-01
2.27873338116E-01	-1.86726488496E-01	-4.14099214666E-01	3.34485052336E-01

THE RESIDUAL MATRIX W'AV IS GIVEN BELOW:

REAL PART

4.00000000096E 00	5.97088899946E-10	4.11091605206E-10	2.86159774996E-07
3.94794327476E-10	3.00000000136E 00	1.12777343026E-10	9.02218746166E-10
-2.78333399676E-11	-3.11702685826E-10	1.99999999986E 00	1.32774707276E-09
1.15674804726E-07	4.35810429496E-11	-5.73125881986E-09	1.00000000036E 00

IMAGINARY PART

-8.00000000126E 00	4.70023504396E-10	2.50565790346E-10	1.15476723296E-07
-3.27232404876E-10	7.00000000206E 00	9.57697920976E-10	-6.25732354826E-10
8.66810301496E-10	1.03240218616E-10	6.00000000156E 00	-5.75524935356E-09
2.85829201236E-07	-6.97627194566E-10	1.03443452866E-09	5.00000000116E 00

EXECUTION TIME WAS 0.753 SECONDS

****

APPENDIX 5

A RITZ ITERATION PROGRAM FOR SYMMETRIC MATRICES

```
C023                  MASTER
C024                  INTEGER DIMN,DIMP,P,EM,G,H,LINE(64),DATE(3)
C025                  REAL X(250,8), V(250), W(250)
C026                  REAL RV(8,8), B(8,8), D(8), F(8), BB(8), Z(8), DOLD(8), LARGE(8)
C027                  REAL MC
C028                  COMMON NO,FIELD,LENGTH,K
C029                  DATA LINE/64*1H*/
C030                  K=1
C031                  CALL ELAPSE (E,DATE,2,K)
C032                  MC=E
C033                  READ (5,10) NUMBER,DIMN,DIMP,LENGTH
C034               10 FORMAT (I2,I5,I2,I3)
C035                  IF (LENGTH .LT. 120) K=-1
C036                  DO 500 NO=1,NUMBER
C037                  READ (5,20) N,P,KM,EM,EPS,FIELD,NORM
C038               20 FORMAT (I5,I2,I6,I2,E9.2,A6,I1)
C039                  IF (KM .LT. 0) READ (5,30) ((X(I,J), J=1,P), I=1,N)
C040               30 FORMAT (60.0)
C041                  CALL TIME (T)
C042                  IF (K) 120,999,100
C043              100 WRITE (6,110) (LINE(I), I=1,64), (DATE(I), I=1,3), T, NO
C044              110 FORMAT (1H1,///,29X,'THE METHOD OF "QUICK RITZ" ITERATION FOR A REA
C045                 1L SYMMETRIC MATRIX',//,29X,64A1,///,45X,'THIS PROGRAM WAS RUN ON',I
C046                 23,1X,A3,1X,I2,//,45X,'AND EXECUTION STARTED AT',A9,//,54X,'EXAMPLE
C047                 3NUMBER',I3)
C048                  GO TO 140
C049              120 WRITE (6,130) (LINE(I), I=1,35), (DATE(I), I=1,3), T, NO
C050              130 FORMAT (1H ,///////,' "QUICK RITZ" FOR SYMMETRIC MATRICES',//,1X,35A
C051                 11,///,' DATE:',I3,1X,A3,1X,I2,5X,'TIME:',A9,5X,'EXAMPLE NUMBER',I3)
C052              140 CALL ELAPSE (E,DATE,1,K)
C053                  CALL QKRZ (X,V,W,RV,B,D,F,BB,Z,DOLD,LARGE,DIMN,DIMP,N,P,EM,
C054                 1 KM,KS,H,G,MC,EPS)
C055                  CALL ELAPSE (E,DATE,1,K)
C056                  IF (NORM .EQ. 1) CALL NORMALISATION (X,DIMN,DIMP,N,P)
C057                  IF (K) 220,999,200
C058              200 WRITE (6,210) H,G,EPS
C059              210 FORMAT (1H ,////,21X,I2,' EIGENVALUES AND',I3,' EIGENVECTORS HAVE
C060                 1 BEEN ACCEPTED TO AN ACCURACY OF',1PE9.2)
C061                  GO TO 240
C062              220 WRITE (6,230) H,G,EPS
C063              230 FORMAT (1H ,////,1X,I2,' EIGENVALUES AND',I3,' EIGENVECTORS HAVE B
C064                 1EEN ACCEPTED TO',1PE9.2)
C065              240 WRITE (6,300) P
C066              300 FORMAT (1H ,////,' THE',I3,' APPROXIMATIONS TO THE EIGENVALUES ARE
C067                 1:')
C068                  CALL MXOP (D,1,DIMP,1,P,FIELD,LENGTH,6,K)
C069                  WRITE (6,310)
C070              310 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIV
C071                 1EN BELOW:')
C072                  CALL MXOP (X,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
C073                  WRITE (6,320) P
C074              320 FORMAT (1H ,////,' THE',I3,' SETS OF CORRESPONDING ERRORS ARE:')
C075                  CALL MXOP (F,1,DIMP,1,P,FIELD,LENGTH,6,0)
C076                  CALL ELAPSE (E,DATE,3,K)
C077                  IF (K) 420,999,400
C078              400 WRITE (6,410) (LINE(I), I=1,60)
C079              410 FORMAT (1H ,////,31X,60A1)
C080                  GO TO 500
C081              420 WRITE (6,430) (LINE(I), I=1,40)
C082              430 FORMAT (1H ,////,1X,40A1)
C083              500 CONTINUE
C084                  STOP OK
C085              999 STOP NONM
C086                  END

END OF SEGMENT, LENGTH   332, NAME  NONM
```

```
C087                SUBROUTINE QKRZ (X,V,W,RV,B,D,F,BB,Z,DOLD,LARGE,DIMN,DIMP,N,P,E
C088               1 KM,KS,H,G,MC,EPS)
C089                INTEGER DIMN,DIMP,P,EM,H,G,H1,G1
C090                REAL X(DIMN,DIMP), V(DIMN), W(DIMN), RV(DIMP,DIMP), B(DIMP,DIMP
C091                REAL D(DIMP), F(DIMP), BB(DIMP), Z(DIMP), DOLD(DIMP), LARGE(DIM
C092                REAL INNER PRODUCT, MC
C093                DOUBLE PRECISION SUM
C094                DO 10 I=1,P
C095                DOLD(I)=0.0
C096             10 LARGE(I)=0.0
C097          C 58 IS 77/2 -76 IS LARGEST EXPONENT ON M/C)
C098                CONST=35.0-ALOG10(FLOAT(N))
C099                KS,G,H,G1,H1=0
C100                M=2
C101                ZZ=0.1
C102                EPS2=10.0*EPS
C103                IF (KM .LT. 0) GO TO 30
C104                DO 20 J=1,P
C105             20 CALL RANDOMISATION (X,DIMN,DIMP,N,J,ZZ)
C106             30 CALL ORTHO (X,RV,DIMN,DIMP,N,P,0,MC)
C107                KM=IABS(KM)
C108          C FORM B=X'AX
C109            100 DO 130 K=G+1,P
C110                DO 110 I=1,N
C111            110 V(I)=X(I,K)
C112                CALL PRODUCT (V,W,DIMN,N)
C113                DO 130 J=G+1,P
C114                SUM=0.0
C115                DO 120 I=1,N
C116            120 SUM=SUM+X(I,J)*W(I)
C117            130 B(J-G,K-G)=SUM
C118          C SOLVE E-VALUE PROBLEM FOR B I.E. V'BV=D
C119                CALL JACO (B,D,RV,BB,Z,DIMP,P-G,K,1,MC,EPS2)
C120                IF (G .EQ. 0) GO TO 200
C121                DO 140 I=G+1,P
C122                J=P+1-I
C123            140 D(J+G)=D(J)
C124                DO 150 I=1,G
C125            150 D(I)=DOLD(I)
C126          C FORM X=VV WHERE Y=AX
C127            200 DO 220 J=G+1,P
C128                DO 210 I=1,N
C129            210 V(I)=X(I,J)
C130                CALL PRODUCT (V,W,DIMN,N)
C131                DO 220 I=1,N
C132            220 X(I,J)=W(I)
C133                DO 250 I=1,N
C134                DO 240 J=G+1,P
C135                SUM=0.0
C136                DO 230 K=G+1,P
C137            230 SUM=SUM+X(I,K)*RV(K-G,J-G)
C138            240 Z(J)=SUM
C139                DO 250 J=G+1,P
C140            250 X(I,J)=Z(J)
C141          C PERFORM M PREMULTIPLICATIONS
C142                DO 310 K=1,M-1
C143                DO 310 J=G+1,P
C144                DO 300 I=1,N
C145            300 V(I)=X(I,J)
C146                CALL PRODUCT (V,W,DIMN,N)
C147                DO 310 I=1,N
C148            310 X(I,J)=W(I)
C149                KS=KS+M
C150          C FORM X=YR
C151                CALL ORTHO (X,B,DIMN,DIMP,N,P,G,MC)
C152          C CHECK STEPS AND CONVERGENCE
C153                IF (M .GE. P) GO TO 430
C154                DO 400 I=M+1,P
C155                IF (ABS((DOLD(I)-D(I))/D(I)) .GT. EPS) GO TO 410
C156            400 M1=M1+1
C157            410 M=M+M1
C158                M1=0
C159                DO 420 I=G+1,P
C160            420 DOLD(I)=D(I)
C161            430 DO 450 J=G+1,P
C162                AUX=ABS(X(1,J))
C163                DO 440 I=2,N
C164                IF (ABS(X(I,J)) .GT. AUX) AUX=ABS(X(I,J))
C165            440 CONTINUE
C166                F(J)=(LARGE(J)-AUX)/AUX
C167            450 LARGE(J)=AUX
C168                DO 460 J=G+1,P
C169                IF (ABS(F(J)) .GT. EPS) GO TO 470
C170            460 G1=G1+1
C171            470 G=G+G1
C172                G1=0
C173                CALL INFO (D,F,DIMP,P,KS,G,H,EM)
C174                IF (G .LE. M) GO TO 510
C175                IF (G .LT. EM) GO TO 500
C176                G=M
C177                M=2
C178                GO TO 510
C179            500 G=M
C180            510 IF (G .GE. EM .OR. KS .GE. KM) GO TO 520
C181                CONST1=CONST/ALOG10(ABS(D(G+1)))
C182                CONST2=M*ALOG10(ABS(D(G+1)/D(P)))
C183                IF (FLOAT(M) .LT. CONST1 .AND. CONST2 .LT. 1.0) M=M+1
C184                GO TO 100
C185            520 RETURN
C186                END

END OF SEGMENT, LENGTH   794, NAME  QKRZ
```

```
0187                  SUBROUTINE JACO (A,D,V,B,Z,DIMN,N,ROT,EIVEC,MC,EPS2)
0188                  INTEGER DIMN,EIVEC,ROT,RES,P,Q
0189                  REAL A(DIMN,DIMN), D(DIMN), V(DIMN,DIMN), B(DIMN), Z(DIMN), MC
0190                  DOUBLE PRECISION SUM
0191                  DO 10 I=1,N
0192                  DO 10 J=1,N
0193                10 V(I,J)=0.0
0194                  DO 20 P=1,N
0195                20 V(P,P)=1.0
0196                30 DO 40 P=1,N
0197                  D(P)=A(P,P)
0198                  B(P)=D(P)
0199                40 Z(P)=0.0
0200                  ROT=0
0201                  EPS=5.0*N*(N-1)*MC
0202                  DO 400 I=1,50
0203                  SUM=0.0
0204                  DO 100 P=1,N-1
0205                  DO 100 Q=P+1,N
0206                  TEMP=ABS(A(P,Q))
0207               100 SUM=SUM+DBLE(TEMP)
0208                  SM=SUM
0209                  IF (SM .LE. EPS) GO TO 500
0210                  TRESH=0.0
0211                  IF (I .LT. 4) TRESH=0.2*SM/N**2
0212                  DO 330 P=1,N-1
0213                  DO 330 Q=P+1,N
0214                  G=100.0*ABS(A(P,Q))
0215                  IF (I .GT. 4 .AND. ABS(D(P))+G .EQ. ABS(D(P)) .AND.
0216                 1   ABS(D(Q))+G .EQ. ABS(D(Q))) GO TO 310
0217                  IF (ABS(A(P,Q)) .LE. TRESH) GO TO 320
0218                  H=D(Q)-D(P)
0219                  IF (ABS(H)+G .EQ. ABS(H)) GO TO 110
0220                  THETA=0.5*H/A(P,Q)
0221                  T=1.0/(ABS(THETA)+SQRT(1.0+THETA**2))
0222                  IF (THETA .LT. 0.0) T=-T
0223                  GO TO 120
0224               110 T=A(P,Q)/H
0225               120 C=1.0/SQRT(1.0+T*T)
0226                  S=T*C
0227                  TAU=S/(1.0+C)
0228                  H=T*A(P,Q)
0229                  Z(P)=Z(P)-H
0230                  Z(Q)=Z(Q)+H
0231                  D(P)=D(P)-H
0232                  D(Q)=D(Q)+H
0233                  A(P,Q)=0.0
0234                  IF (P .EQ. 1) GO TO 210
0235                  DO 200 J=1,P-1
0236                  AJP=A(J,P)
0237                  AJQ=A(J,Q)
0238                  A(J,P)=AJP-S*(AJQ+AJP*TAU)
0239               200 A(J,Q)=AJQ+S*(AJP-AJQ*TAU)
0240               210 IF (P+1 .GT. Q-1) GO TO 2302
0241                  DO 220 J=P+1,Q-1
0242                  APJ=A(P,J)
0243                  AJQ=A(J,Q)
0244                  A(P,J)=APJ-S*(AJQ+APJ*TAU)
0245               220 A(J,Q)=AJQ+S*(APJ-AJQ*TAU)
0246               230 IF (Q .EQ. N) GO TO 250
0247                  DO 240 J=Q+1,N
0248                  APJ=A(P,J)
0249                  AQJ=A(Q,J)
0250                  A(P,J)=APJ-S*(AQJ+APJ*TAU)
0251               240 A(Q,J)=AQJ+S*(APJ-AQJ*TAU)
0252               250 IF (EIVEC .EQ. 0) GO TO 300
0253                  DO 260 J=1,N
0254                  VJP=V(J,P)
0255                  VJQ=V(J,Q)
0256                  V(J,P)=VJP-S*(VJQ+VJP*TAU)
0257               260 V(J,Q)=VJQ+S*(VJP-VJQ*TAU)
0258               300 ROT=ROT+1
0259                  GO TO 320
0260               310 A(P,Q)=0.0
0261               320 CONTINUE
0262               330 CONTINUE
0263                  DO 340 P=1,N
0264                  B(P)=B(P)+Z(P)
0265                  D(P)=B(P)
0266               340 Z(P)=0.0
0267               400 CONTINUE
0268                  ROT=-ROT
0269               500 DO 540 P=1,N-1
0270                  DO 540 Q=P+1,N
0271                  TEMP=ABS(D(P))-ABS(D(Q))
0272                  IF (ABS(TEMP) .LE. EPS2) GOQTO 530
0273                  IF (TEMP) 510,530,530
0274               510 SM=D(P)
0275                  D(P)=D(Q)
0276                  D(Q)=SM
0277                  DO 520 I=1,N
0278                  SM=V(I,P)
0279                  V(I,P)=V(I,Q)
0280               520 V(I,Q)=SM
0281               530 CONTINUE
0282               540 CONTINUE
0283                  RETURN
0284                  END

END OF SEGMENT, LENGTH   708, NAME  JACO
```

```
C285                    SUBROUTINE ORTHO (X,B,DIMN,DIMP,N,P,F,MC)
C286                    INTEGER DIMN,DIMP,P,F,ORIG
C287                    REAL X(DIMN,DIMP), B(DIMP,DIMP), INNER PRODUCT, MC
C288                    DOUBLE PRECISION SUM
C289                    DO 80 K=F+1,P
C290                    ORIG=1
C291                 10 SUM=0.0
C292                    IF (K .EQ. 1) GO TO 40
C293                    DO 30 I=1,K-1
C294                    S=INNER PRODUCT (X,DIMN,DIMP,N,I,K)
C295                    IF (ORIG .EQ. 1) B(P-K+1,I)=S
C296                    SUM=SUM+S*S
C297                    DO 20 J=1,N
C298                    X(J,K)=X(J,K)-S*X(J,I)
C299                 20 CONTINUE
C300                 30 CONTINUE
C301                 40 S=INNER PRODUCT (X,DIMN,DIMP,N,K,K)
C302                    SUM=SUM+S
C303                    T=SUM
C304                    IF (S .GT. T/100.0 .AND. T*MC .NE. 0.0) GO TO 50
C305                    ORIG=0
C306                    WRITE (4,100)
C307                100 FORMAT (' WARNING 1 IN ORTHO')
C308                    IF (S*MC .NE. 0.0) GO TO 10
C309                    WRITE (4,110)
C310                110 FORMAT (' WARNING 2 IN ORTHO')
C311                    S=0.0
C312                    B(P-K+1,K)=0.0
C313                 50 IF (S .EQ. 0.0) GO TO 60
C314                    S=SQRT(S)
C315                    B(P-K+1,K)=S
C316                    S=1.0/S
C317                 60 DO 70 J=1,N
C318                 70 X(J,K)=S*X(J,K)
C319                 80 CONTINUE
C320                    RETURN
C321                    END
```

END OF SEGMENT, LENGTH 264, NAME ORTHO

```
C322                    REAL FUNCTION INNER PRODUCT (X,DIMN,DIMP,N,K,L)
C323                    INTEGER DIMN,DIMP,P
C324                    REAL X(DIMN,DIMP)
C325                    DOUBLE PRECISION SUM
C326                    SUM=0.0
C327                    DO 10 I=1,N
C328                 10 SUM=SUM+X(I,K)*X(I,L)
C329                    INNER PRODUCT=SUM
C330                    RETURN
C331                    END
```

END OF SEGMENT, LENGTH 80, NAME INNERPRODUCT

```
C332                    SUBROUTINE RANDOMISATION (X,DIMN,DIMP,N,L,Z)
C333                    INTEGER DIMN,DIMP
C334                    REAL X(DIMN,DIMP)
C335                    DO 10 I=1,N
C336                 10 X(I,L)=2.0*FPMCRV(Z)-1.0
C337                    Z=FPMCRV(Z)
C338                    RETURN
C339                    END
```

END OF SEGMENT, LENGTH 67, NAME RANDOMISATION

```
C340                    SUBROUTINE INFO (D,F,DIMP,P,KS,G,H,EM)
C341                    INTEGER DIMN,DIMP,STREAM,PREV,G,H,EM,P,LINE(63)
C342                    REAL F(DIMP), D(DIMP)
C343                    DATA LINE/63*1H*/, PREV/0/
C344                    COMMON NO,FIELD,LENGTH,K
C345                    IF (K) 200,200,100
C346                 100 IF (PREV .EQ. NO) GO TO 120
C347                    WRITE (4,110) NO, (LINE(I), I=1,63)
C348                 110 FORMAT (1H1,//,29X,'INFORMATIONAL OUTPUT ON LOGICAL STREAM 4 FOR
C349                1'XAMPLE NUMBER',I3,/,29X,63A1)
C350                 120 WRITE (4,130) KS,G,H,EM
C351                 130 FORMAT (1H ,//,43X,'NUMBER OF  STEPS  PERFORMED =',I6,//,
C352                    1 43X,'NUMBER OF EIGENVECTORS ACCEPTED =',I3,/,
C353                    2 43X,'NUMBER OF EIGENVALUESU ACCEPTED =',I3,//,
C354                    3 43X,'NUMBER OF SOLNS. TO BD COMPUTED =',I3)
C355                    GO TO 300
C356                 200 IF (PREV .EQ. NO) GO TO 2202
C357                    WRITE (4,210) NO, (LINE(I), I=1,42)
C358                 210 FORMAT (1H ,///////,' INFORMATIONAL OUTPUT FOR EXAMPLE NUMBER',I3
C359                1,1X,42A1)
C360                 220 WRITE (4,230) KS,G,H,EM
C361                 230 FORMAT (1H ,//,' STEPS PERFORMED =',I6,/,' EIGENVECTORS ACCEPTED
C362                   1',I3,/,' EIGENVALUES  ACCEPTED =',I3,/,' SOLNS. TO BE COMPUTED =
C363                   2I3)
C364                 300 WRITE (4,310)
C365                 310 FORMAT (' THE ERROR VECTOR IS:')
C366                    CALL MXOP (F,1,DIMP,1,P,FIELD,LENGTH,4,K)
C367                    WRITE (4,320)
C368                 320 FORMAT (' THE APPROXIMATIONS TO THE EIGENVALUES ARE:')
C369                    CALL MXOP (D,1,DIMP,1,P,FIELD,LENGTH,4,0)
C370                    PREV=NO
C371                    RETURN
C372                    END
```

END OF SEGMENT, LENGTH 173, NAME INFO

```
C373                  SUBROUTINE NORMALISATION (X,DIMN,DIMP,N,P)
C374                  INTEGER DIMN,DIMP,P
C375                  REAL X(DIMN,DIMP)
C376                  DO 30 J=1,P
C377                  AUX=X(1,J)
C378                  DO 10 I=2,N
C379                  IF (ABS(X(I,J)) .GT. ABS(AUX))  AUX=X(I,J)
C380               10 CONTINUE
C381                  DO 20 I=1,N
C382                  X(I,J)=X(I,J)/AUX
C383               20 CONTINUE
C384               30 CONTINUE
C385                  RETURN
C386                  END
```

END OF SEGMENT, LENGTH 105, NAME NORMALISATIONP

```
C387                  SUBROUTINE ELAPSE (E,DAT,NO,K)
C388                  INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
C389                  DATA CALL/2/
C390                  DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
C391                 1 'OCT','NOV','DEC'/
C392                  GO TO (100,200,300), NO
C393               100 CALL MTIME (N)
C394                  E=N
C395                  CALL=3-CALL
C396                  IF (CALL .EQ. 2) GO TO 110
C397                  EE=E
C398                  RETURN
C399               110 E=E-EE
C400                  E=E+1.0E-3
C401                  RETURN
C402               200 CALL DEFBUF (2,8,BUFFER)
C403                  CALL DATE (E)
C404                  WRITE (2,210) E
C405               210 FORMAT (A8)
C406                  READ (2,220) (DAT(I), I=1,3)
C407               220 FORMAT (I2,1X,I2,1X,I2)
C408                  DAT(2)=MONTH(DAT(2))
C409                  E,EE=1.0E-03
C410               230 IF (1.0+EE .EQ. 1.0) GO TO 240
C411                  E=EE
C412                  EE=EE/2.0
C413                  GO TO 230
C414               240 RETURN
C415               300 IF (K) 370,370,310
C416               310 IF (E .GE. 60.0) GO TO 330
C417                  WRITE (6,320) E
C418               320 FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
C419                  GO TO 390
C420               330 IF (E .GE. 120.0) GO TO 3502
C421                  E=E-60.0
C422                  WRITE (6,340) E
C423               340 FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
C424                 1)
C425                  GO TO 390
C426               350 I=E/60.0
C427                  E=E-I*60.0
C428                  WRITE (6,360) I, E
C429               360 FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
C430                1ONDS')
C431                  GO TO 390
C432               370 WRITE (6,380) E
C433               380 FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
C434               390 RETURN
C435                  END
```

END OF SEGMENT, LENGTH 182, NAME ELAPSE

```
0436                  SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
0437                  INTEGER DIMN,DIMP,P,WIDTH,STREAM
0438                  REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
0439                  DATA FRMT1/3H(        ,1HN,8HX,1P     ,1HN,1HE,8H)       /
0440                  DATA FRMT2/3H(        ,1HN,8HX,1P     ,1HN,1HE,8H)       /
0441                  IF (IFLAG) 10,200,10
0442               10 CALL DEFBUF (2,16,BUFFER)
0443                  FRMT1(5), FRMT2(5)=FIELD
0444                  WRITE (2,20) FIELD
0445               20 FORMAT (2A8)
0446                  READ (2,30) WIDTH
0447               30 FORMAT (1X,I2)
0448                  IF (P*WIDTH-LENGTH) 100,100,40
0449               40 N1=LENGTH/WIDTH
0450                  N2=P/N1
0451                  LAST=P-N1*N2
0452                  IF (IFLAG) 50,999,60
0453               50 N3=1
0454                  N4=1
0455                  GO TO 70
0456               60 N3=(LENGTH-N1*WIDTH)/2+1
0457                  N4=(LENGTH-LAST*WIDTH)/2+1
0458               70 WRITE (2,80) N3,N1,N4,LAST
0459               80 FORMAT (4I4)
0460                  READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
0461               90 FORMAT (4A4)
0462                  GO TO 300
0463              100 N2=0
0464                  IF (IFLAG) 110,999,120
0465              110 N3=1
0466                  GO TO 130
0467              120 N3=(LENGTH-P*WIDTH)/2+1
0468              130 WRITE (2,140) N3,P
0469              140 FORMAT (2I4)
0470                  READ (2,150) FRMT1(2), FRMT1(4)
0471              150 FORMAT (2A4)
0472                  GO TO 210
0473              200 IF (N2) 999,210,300
0474              210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
0475                  RETURN
0476              300 N2N1=N2*N1
0477                  IF (N-1) 999,400,310
0478              310 DO 330 K=1,N2N1,N1
0479                  KK=K+N1-1
0480                  WRITE (STREAM,320) K, KK
0481              320 FORMAT (1H ,7,' COLUMNS',I3,' TO',I3,' ARE:')
0482              330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
0483                  IF (LAST-1) 370,340,360
0484              340 WRITE (STREAM,350) P
0485              350 FORMAT (1H ,7,' COLUMN',I3,' IS:')
0486                  WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
0487                  GO TO 370
0488              360 KK=N2N1+1
0489                  WRITE (STREAM,320) KK,P
0490                  WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
0491              370 RETURN
0492              400 DO 410 K=1,N2N1,N1
0493                  KK=K+N1-1
0494              410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
0495                  IF (LAST-1) 440,420,430
0496              420 WRITE (STREAM,FRMT2) A(1,P)
0497                  GO TO 440
0498              430 KK=N2N1+1
0499                  WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
0500              440 RETURN
0501              999 STOP MXOP
0502                  END

END OF SEGMENT, LENGTH   427, NAME  MXOP
```

```
C503                  SUBROUTINE PRODUCT (V,W,DIMN,N)
C504                  INTEGER DIMN
C505                  REAL V(DIMN), W(DIMN)
C506                  DOUBLE PRECISION SUM
C507                  COMMON NO
C508                  GO TO (100,200,300,400,500,600,700,800,900), NO
C509            100   CONTINUE
C510                  DO 120 I=1,N
C511                  SUM=0.0
C512                  DO 110 J=1,N
C513                  SUM=SUM+(1.0/FLOAT(I+J-1))*V(J)
C514            110   CONTINUE
C515                  W(I)=SNGL(SUM)
C516            120   CONTINUE
C517                  RETURN
C518            200   CONTINUE
C519                  DO 210 I=1,N-1
C520                  W(I)=V(I)+FLOAT(I)*V(N)
C521            210   CONTINUE
C522                  SUM=0.0
C523                  DO 220 I=1,N
C524                  SUM=SUM+FLOAT(I)*V(I)
C525            220   CONTINUE
C526                  W(N)=SNGL(SUM)
C527                  RETURN
C528            300   CONTINUE
C529                  W(1)=4.0*V(1)
C530                  W(2)=3.0*V(2)
C531                  W(3)=2.0*V(3)
C532                  DO 310 I=4,N
C533                  W(I)=V(I)
C534            310   CONTINUE
C535                  RETURN
C536            400   CONTINUE
C537                  K=(N-1)/2
C538                  W(1)=K*V(1)+V(2)
C539                  W(K+1)=V(K)+V(K+2)
C540                  W(N)=V(N-1)+K*V(N)
C541                  L=K
C542                  DO 410 I=2,K
C543                  L=L-1
C544                  J=N-I+1
C545                  W(I)=V(I-1)+L*V(I)+V(I+1)
C546                  W(J)=V(J-1)+L*V(J)+V(J+1)
C547            410   CONTINUE
C548                  RETURN
C549            500   CONTINUE
C550                  K=(N-1)/2
C551                  W(1)=K*V(1)+V(2)
C552                  W(K+1)=V(K)+V(K+2)
C553                  W(N)=V(N-1)-K*V(N)
C554                  L=K
C555                  DO 510 I=2,K
C556                  L=L-1
C557                  J=N-I+1
C558                  W(I)=V(I-1)+L*V(I)+V(I+1)
C559                  W(J)=V(J-1)-L*V(J)+V(J+1)
C560            510   CONTINUE
C561                  RETURN
C562            600   CONTINUE
C563                  W(1)=V(2)
C564                  W(N)=V(N-1)
C565                  DO 610 I=2,N-1
C566                  W(I)=V(I-1)+V(I+1)
C567            610   CONTINUE
C568                  RETURN
C569            700   CONTINUE
C570                  X=SQRT(FLOAT(N-1))
C571                  W(1)=X*V(2)
C572                  DO 710 I=2,N-1
C573                  Y=X
C574                  X=SQRT(FLOAT(I*(N-I)))
C575                  W(I)=Y*V(I-1)+X*V(I+1)
C576            710   CONTINUE
C577                  Y=X
C578                  W(N)=Y*V(N-1)
C579                  RETURN
C580            800   CONTINUE
C581                  W(1)=5.0*V(1)+2.0*V(2)+V(3)+V(4)
C582                  W(2)=2.0*V(1)+6.0*V(2)+3.0*V(3)+V(4)+V(5)
C583                  W(3)=V(1)+3.0*V(2)+6.0*V(3)+3.0*V(4)+V(5)+V(6)
C584                  W(42)=V(39)+V(40)+3.0*V(41)+6.0*V(42)+3.0*V(43)+V(44)
C585                  W(43)=V(40)+V(41)+3.0*V(42)+6.0*V(43)+2.0*V(44)
C586                  W(44)=V(41)+V(42)+2.0*V(43)+5.0*V(44)
C587                  DO 810 I=5,41
C588                  SUM=V(I-3)+V(I-2)+3.0*V(I-1)+6.0*V(I)+3.0*V(I+1)+V(I+2)+V(I+3)
C589                  W(I)=SNGL(SUM)
C590            810   CONTINUE
C591                  RETURN
C592            900   CONTINUE
C593                  RETURN
C594                  END

END OF SEGMENT, LENGTH   805, NAME  PRODUCT
```

THE METHOD OF "QUICK RITZ" ITERATION FOR A REAL SYMMETRIC MATRIX

THIS PROGRAM WAS RUN ON 22 Aug 74
AND EXECUTION STARTED AT 17/39/06

EXAMPLE NUMBER 1

3 EIGENVALUES AND 3 EIGENVECTORS HAVE BEEN ACCEPTED TO AN ACCURACY OF 1.00E-08

THE 4 APPROXIMATIONS TO THE EIGENVALUES ARE:

1.7519196702E 00 3.4292954848E-01 3.5741816272E-02 2.5308907688E-03

THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIVEN BELOW:

1.0000000000E 00	1.0000000000E 00	-5.0604464868E-01	-1.8310074992E-01
6.0399191433E-01	-1.1046517177E-01	1.0000000000E 00	-1.0000000000E 00
4.5213829897E-01	-3.6628257965E-01	6.3141538762E-01	-3.5041395168E-01
3.6528601340E-01	-4.4242591786E-01	2.4089524194E-01	-7.1501577465E-01
3.0775304745E-01	-4.6053694517E-01	-4.5861875156E-02	-6.0964676230E-01
2.6672514426E-01	-4.5634125730E-01	-2.4504058875E-01	-3.3913340171E-01
2.3580130793E-01	-4.4303622451E-01	-3.8217819752E-01	-3.3662948670E-02
2.1156324395E-01	-4.2617131612E-01	-4.7640186975E-01	2.5545435265E-01
1.9200512816E-01	-4.0826080801E-01	-5.4083648067E-01	5.1053849891E-01
1.7586003439E-01	-3.9048378674E-01	-5.8436376351E-01	7.2797972593E-01

THE 4 SETS OF CORRESPONDING ERRORS ARE:

1.0401433487E-11 -1.1410745889E-11 4.2474223130E-10 -3.2801305697E-09

EXECUTION TIME WAS 1.950 SECONDS

INFORMATIONAL OUTPUT ON LOGICAL STREAM 4 FOR EXAMPLE NUMBER 1
****

NUMBER OF STEPS PERFORMED = 2
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS:
-1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00
THE APPROXIMATIONS TO THE EIGENVALUES ARE:
1.0311615175E 00 2.5612776372E-01 5.1845470385E-03 1.5400640715E-04

NUMBER OF STEPS PERFORMED = 4
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS:
-2.8639349088E-03 2.3195772231E-03 9.7824237673E-03 -1.1661225376E-02
THE APPROXIMATIONS TO THE EIGENVALUES ARE:
1.7519196702E 00 3.4292954849E-01 3.5741816253E-02 2.5308784147E-03

NUMBER OF STEPS PERFORMED = 6
NUMBER OF EIGENVECTORS ACCEPTED = 3
NUMBER OF EIGENVALUES ACCEPTED = 3
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS:
1.0401433487E-11 -1.1410745889E-11 4.2474223150E-10 -3.2801365697E-06
THE APPROXIMATIONS TO THE EIGENVALUES ARE:
1.7519196702E 00 3.4292954848E-01 3.5741816272E-02 2.5308907688E-03

APPENDIX 6

A RITZ ITERATION PROGRAM FOR HERMITIAN MATRICES

```
C023                MASTER
C024                INTEGER DIMN,DIMP,P,EM,G,H,LINE(60),DATE(3)
C025                REAL XR(50,8),XI(50,8), VR(50),VI(50), WR(50),WI(50)
C026                REAL RVR(8,8),RVI(8,8), BR(8,8),BI(8,8), D(8), F(8), BB(8)
C027                REAL Z(3), DOLD(8), LARGE(8), MC
C028                COMMON NO,FIELD,LENGTH,K
C029                DATA LINE/60*1H*/
C030                K=1
C031                CALL ELAPSE (E,DATE,2,K)
C032                MC=E
C033                READ (5,10) NUMBER,DIMN,DIMP,LENGTH
C034             10 FORMAT (I2,I5,I2,I3)
C035                IF (LENGTH .LT. 120) K=-1
C036                DO 500 NO=1,NUMBER
C037                READ (5,20) N,P,KM,EM,EPS,FIELD,NORM
C038             20 FORMAT (I5,I2,I6,I2,E9.2,A6,I1)
C039                IF (K*1 .LT. 0) READ (5,30) ((XR(I,J), XI(I,J), J=1,P), I=1,N)
C040             30 FORMAT (2G0.0)
C041                CALL TIME (T)
C042                IF (K) 120,999,100
C043            100 WRITE (6,110) (LINE(I), I=1,60), (DATE(I), I=1,3), T, NO
C044            110 FORMAT (1H1,//,31X,'THE METHOD OF "QUICK RITZ" ITERATION FOR AN HE
C045               1RMITIAN MATRIX',/,31X,60A1,///,45X,'THIS PROGRAM WAS RUN ON',I3,1X
C046               2,A3,1X,I2,),45X,'AND EXECUTION STARTED AT',A9,//,54X,'EXAMPLE NUMB
C047               3ER',I3)
C048                GO TO 140
C049            120 WRITE (7,130) (LINE(I), I=1,35), (DATE(I), I=1,3), T, NO
C050            130 FORMAT (1H ,//////,' "QUICKRITZ" FOR HERMITIAN MATRICES',/,1X,35A
C051               11,//,' DATE:',I3,1X,A3,1X,I4,5X,'TIME:',A9,5X,'EXAMPLE NUMBER',I3)
C052            140 CALL ELAPSE (E,DATE,1,K)
C053                CALL QRQZ (XR,XI,VR,VI,WR,WI,RVR,RVI,BR,BI,D,F,BB,Z,DOLD,
C054               1 LARGE,DIMN,DIMP,N,P,EM,KM,KS,H,G,MC,EPS)
C055                CALL ELAPSE (E,DATE,1,K)
C056                IF (NORM .EQ. 1) CALL NORMALISATION (XR,XI,DIMN,DIMP,N,P)
C057                IF (K) 220,999,200
C058            200 WRITE (7,210) H,G,EPS
C059            210 FORMAT (1H ,////,21X,I2,' EIGENVALUES AND',I3,' EIGENVECTORS HAVE
C060               1BEEN ACCEPTED TO AN ACCURACY OF',1PE9.2)
C061                GO TO 240
C062            220 WRITE (6,230) H,G,EPS
C063            230 FORMAT (1H ,////,1X,I2,' EIGENVALUES AND',I3,' EIGENVECTORS HAVE B
C064               1EEN ACCEPTED TO',1PE9.2)
C065            240 WRITE (6,300) P
C066            300 FORMAT (1H ,////,' THE',I3,' APPROXIMATIONS TO THE EIGENVALUES ARE
C067               1:')
C068                CALL MXOP (D,1,DIMP,1,P,FIELD,LENGTH,6,K)
C069                WRITE (6,310)
C070            310 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIV
C071               1EN BELOW:')
C072                WRITE (6,320)
C073            320 FORMAT (1H ,/,' REAL PART')
C074                CALL MXOP (XR,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
C075                WRITE (6,321)
C076            321 FORMAT (1H ,/,' IMAGINARY PART')
C077                CALL MXOP (XI,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
C078                WRITE (6,330) P
C079            330 FORMAT (1H ////,' THE',I3,' SETS OF CORRESPONDING ERRORS ARE:')
C080                CALL MXOP (F,1,DIMP,1,P,FIELD,LENGTH,6,0)
C081                CALL ELAPSE (E,DATE,3,K)
C082                IF (K) 420,999,400
C083            400 WRITE (6,410) (LINE(I), I=1,60)
C084            410 FORMAT (1H ,////,31X,60A1)
C085                GO TO 500
C086            420 WRITE (6,430) (LINE(I), I=1,40)
C087            430 FORMAT (1H ,////,1X,40A1)
C088            500 CONTINUE
C089                STOP OK
C090            999 STOP NONM
C091                END

END OF SEGMENT, LENGTH   363, NAME  NONM
```

```
C092                SUBROUTINE HHRZ (XR,XI,VR,VI,WR,WI,RVR,RVI,BR,BI,D,F,BB,Z,DOLD,
C093               1 LARGE,DIMN,DIMP,N,P,EM,KM,KS,M,G,MC,EPS)
C094                INTEGER DIMN,DIMP,P,EM,H,G,H1,G1
C095                REAL XR(DIMN,DIMP),XI(DIMN,DIMP), VR(DIMN),VI(DIMN)
C096                REAL WR(DIMN),WI(DIMN), RVR(DIMP,DIMP),RVI(DIMP,DIMP)
C097                REAL BR(DIMP,DIMP),BI(DIMP,DIMP), D(DIMP), F(DIMP), BB(DIMP)
C098                REAL Z(DIMP), DOLD(DIMP), LARGE(DIMP), MC
C099                DOUBLE PRECISION SUMR,SUMI
C100                DO 10 I=1,P
C101                DOLD(I)=0.0
C102             10 LARGE(I)=0.0
C103             C 38 IS 76/2 (76 IS LARGEST EXPONENT ON M/C)
C104                CONST=38.0-ALOG10(FLOAT(N))
C105                KS,G,H,G1,H1=0
C106                M=2
C107                ZZ=0.1
C108                EPS2=10.0*EPS
C109                IF (KM .LT. 0) GO TO 40
C110                DO 20 J=1,P
C111             20 CALL RANDOMISATION (XR,DIMN,DIMP,N,J,ZZ)
C112                DO 30 J=1,P
C113                DO 30 I=1,N
C114             30 XI(I,J)=0.0
C115             40 CALL CORTHO (XR,XI,DIMN,DIMP,N,P,0,MC)
C116                KM=IABS(KM)
C117             C FORM B=X*AX
C118            100 DO 130 K=G+1,P
C119                DO 110 I=1,N
C120                VR(I)=XR(I,K)
C121            110 VI(I)=XI(I,K)
C122                CALL PRODUCT (VR,VI,WR,WI,DIMN,N)
C123                DO 130 J=G+1,P
C124                SUMR,SUMI=0.0
C125                DO 120 I=1,N
C126                SUMR=SUMR+XR(I,J)*WR(I)+XI(I,J)*WI(I)
C127            120 SUMI=SUMI+XR(I,J)*WI(I)-XI(I,J)*WR(I)
C128                BR(J-G,K-G)=SUMR
C129            130 BI(J-G,K-G)=SUMI
C130             C SOLVE E-VALUE PROBLEM FOR B I.E. V*BV=D
C131                CALL HHJA (BR,BI,D,RVR,RVI,BB,Z,DIMP,P-G,K,1,MC,EPS2)
C132                IF (G .EQ. 0) GO TO 200
C133                DO 140 I=G+1,P
C134                J=P+1-I
C135            140 D(J+G)=D(J)
C136                DO 150 I=1,G
C137            150 D(I)=DOLD(I)
C138             C FORM X=YV WHERE Y=AX
C139            200 DO 220 J=G+1,P
C140                DO 210 I=1,N
C141                VR(I)=XR(I,J)
C142            210 VI(I)=XI(I,J)
C143                CALL PRODUCT (VR,VI,WR,WI,DIMN,N)
C144                DO 220 I=1,N
C145                XR(I,J)=WR(I)
C146            220 XI(I,J)=WI(I)
C147                DO 250 I=1,N
C148                DO 240 J=G+1,P
C149                SUMR,SUMI=0.0
C150                DO 230 K=G+1,P
C151                SUMR=SUMR+XR(I,K)*RVR(K-G,J-G)-XI(I,K)*RVI(K-G,J-G)
C152            230 SUMI=SUMI+XR(I,K)*RVI(K-G,J-G)+XI(I,K)*RVR(K-G,J-G)
C153                Z(J)=SUMR
C154            240 BB(J)=SUMI
C155                DO 250 J=G+1,P
C156                XR(I,J)=Z(J)
C157            250 XI(I,J)=BB(J)
C158             C PERFORM M PREMULTIPLICATIONS
C159                DO 310 K=1,M-1
C160                DO 310 J=G+1,P
C161                DO 300 I=1,N
C162                VR(I)=XR(I,J)
C163            300 VI(I)=XI(I,J)
C164                CALL PRODUCT (VR,VI,WR,WI,DIMN,N)
C165                DO 310 I=1,N
C166                XR(I,J)=WR(I)
C167            310 XI(I,J)=WI(I)
C168                KS=KS+M
C169             C FORM X=YR
```

```
C170                  CALL CORTHO (XR,XI,DIMN,DIMP,N,P,G,MC)
C171            C CHECK STEPS AND CONVERGENCE
C172                  IF (H .GE. P) GO TO 430
C173                  DO 400 I=H+1,P
C174                  IF (ABS((DOLD(I)-D(I))/D(I)) .GT. EPS) GO TO 410
C175             400  H1=H1+1
C176             410  H=H+H1
C177                  H1=0
C178                  DO 420 I=G+1,P
C179             420  DOLD(I)=D(I)
C180             430  DO 450 J=G+1,P
C181                  AUX=SQRT(XR(1,J)**2+XI(1,J)**2)
C182                  DO 440 I=2,H
C183                  ZZ=SQRT(XR(I,J)**2+XI(I,J)**2)
C184                  IF (ZZ .GT. AUX) AUX=ZZ
C185             440  CONTINUE
C186                  F(J)=(LARGE(J)-AUX)/AUX
C187             450  LARGE(J)=AUX
C188                  DO 460 J=G+1,P
C189                  IF (ABS(F(J)) .GT. EPS) GO TO 470
C190             460  G1=G1+1
C191             470  G=G+G1
C192                  G1=0
C193                  CALL INFO (D,F,DIMP,P,KS,G,H,EM)
C194                  IF (G .LE. H) GO TO 510
C195                  IF (G .LT. EM) GO TO 500
C196                  G=H
C197                  M=2
C198                  GO TO 510
C199             500  G=H
C200             510  IF (G .GE. EM .OR. KS .GE. KM) GO TO 520
C201                  CONST1=CONST/ALOG10(ABS(D(G+1)))
C202                  CONST2=M*ALOG10(ABS(D(G+1)/D(P)))
C203                  IF (FLOAT(M) .LT. CONST1 .AND. CONST2 .LT. 1.0) M=M+1
C204                  GO TO 100
C205             520  RETURN
C206                  END

END OF SEGMENT. LENGTH  1093. NAME  HMRZ

C207                  SUBROUTINE HMJO (AR,AI,D,VR,VI,B,Z,DIMN,N,ROT,EIVEC,MC,EPS2)
C208                  INTEGER DIMN,EIVEC,ROT,P,H,RES
C209                  REAL AR(DIMN,DIMN), AI(DIMN,DIMN), VR(DIMN,DIMN), VI(DIMN,DIMN)
C210                  REAL D(DIMN), B(DIMN), Z(DIMN), MC
C211                  DOUBLE PRECISION SUM
C212                  DO 10 I=1,N
C213                  DO 10 J=1,N
C214                  VR(1,J)=0.0
C215              10  VI(I,J)=0.0
C216                  DO 20 P=1,N
C217              20  VR(P,P)=1.0
C218              30  DO 40 P=1,N
C219                  D(P)=AR(P,P)
C220                  B(P)=D(P)
C221              40  Z(P)=0.0
C222                  ROT=0
C223                  EPS=5.0*N*(N-1)*MC
C224                  DO 600 I=1,50
C225                  SUM=0.0
C226                  DO 100 P=1,N-1
C227                  DO 100 Q=P+1,N
C228                  TEMP=SQRT(AR(P,Q)**2+AI(P,Q)**2)
C229             100  SUM=SUM+DBLE(TEMP)
C230                  SM=SUM
C231                  IF (SM .LE. EPS) GO TO 610
C232                  TRESH=0.0
C233                  IF (I .LT. 4) TRESH=0.2*SM/N**2
C234                  DO 500 P=1,N-1
C235                  DO 500 Q=P+1,N
C236                  TEMP=SQRT(AR(P,Q)**2+AI(P,Q)**2)
C237                  G=100.0*TEMP
C238                  IF (I .GT. 4 .AND. ABS(D(P))+G .EQ. ABS(D(P)) .AND.
C239                 1 ABS(D(Q))+G .EQ. ABS(D(Q))) GO TO 410
C240                  IF (TEMP .LE. TRESH) GO TO 420
C241                  E=AR(P,Q)
C242                  F=AI(P,Q)
C243                  IF (ABS(E)-ABS(F)) 200,210,210
C244             200  T=E/F
C245                  ST=1.0/SQRT(1.0+T*T)
C246                  CT=T*ST
C247                  GO TO 220
C248             210  T=F/E
C249                  CT=1.0/SQRT(1.0+T*T)
C250                  ST=T*CT
C251             220  OMEGA=E*CT+F*ST
C252                  H=D(Q)-D(P)
C253                  IF (ABS(H)+G .EQ. ABS(H)) GO TO 230
C254                  THETA=0.5*H/OMEGA
C255                  T=1.0/(ABS(THETA)+SQRT(1.0+THETA**2))
C256                  IF (THETA .LT. 0.0) T=-T
C257                  GO TO 240
C258             230  T=OMEGA/H
C259             240  C=1.0/SQRT(1.0+T*T)
C260                  S=T*C
C261                  TAU=S/(1.0+C)
C262                  H=T*OMEGA
C263                  Z(P)=Z(P)-H
C264                  Z(Q)=Z(Q)+H
C265                  D(P)=D(P)-H
C266                  D(Q)=D(Q)+H
C267                  AR(P,Q)=0.0
C268                  AI(P,Q)=0.0
C269                  IF (P .EQ. 1) GO TO 310
C270                  DO 300 J=1,P-1
C271                  ARJP=AR(J,P)
```

```
C272                ARJQ=AR(J,Q)
C273                AIJP=AI(J,P)
C274                AIJQ=AI(J,Q)
C275                AR(J,P)=ARJP-S*(ARJQ*CT+AIJQ*ST+ARJP*TAU)
C276                AR(J,Q)=ARJQ+S*(ARJP*CT-AIJP*ST-ARJQ*TAU)
C277                AI(J,P)=AIJP-S*(AIJQ*CT-ARJQ*ST+AIJP*TAU)
C278          300   AI(J,Q)=AIJQ+S*(AIJP*CT+ARJP*ST-AIJQ*TAU)
C279          310   IF (P+1 .GT. Q-1) GO TO 330
C280                DO 320 J=P+1,Q-1
C281                ARPJ=AR(P,J)
C282                ARJQ=AR(J,Q)
C283                AIPJ=AI(P,J)
C284                AIJQ=AI(J,Q)
C285                AR(P,J)=ARPJ-S*(ARJQ*CT+AIJQ*ST+ARPJ*TAU)
C286                AR(J,Q)=ARJQ+S*(ARPJ*CT+AIPJ*ST-ARJQ*TAU)
C287                AI(P,J)=AIPJ-S*(-AIJQ*CT+ARJQ*ST+AIPJ*TAU)
C288          320   AI(J,Q)=AIJQ+S*(-AIPJ*CT+ARPJ*ST-AIJQ*TAU)
C289          330   IF (Q .EQ. N) GO TO 350
C290                DO 340 J=Q+1,N
C291                ARPJ=AR(P,J)
C292                ARQJ=AR(Q,J)
C293                AIPJ=AI(P,J)
C294                AIQJ=AI(Q,J)
C295                AR(P,J)=ARPJ-S*(ARQJ*CT-AIQJ*ST+ARPJ*TAU)
C296                AR(Q,J)=ARQJ+S*(ARPJ*CT+AIPJ*ST-ARQJ*TAU)
C297                AI(P,J)=AIPJ-S*(AIQJ*CT+ARQJ*ST+AIPJ*TAU)
C298          340   AI(Q,J)=AIQJ+S*(AIPJ*CT-ARPJ*ST-AIQJ*TAU)
C299          350   IF (IVEC .EQ. 0) GO TO 400
C300                DO 360 J=1,N
C301                VRJP=VR(J,P)
C302                VRJQ=VR(J,Q)
C303                VIJP=VI(J,P)
C304                VIJQ=VI(J,Q)
C305                VR(J,P)=VRJP-S*(VRJQ*CT+VIJQ*ST+VRJP*TAU)
C306                VR(J,Q)=VRJQ+S*(VRJP*CT-VIJP*ST-VRJQ*TAU)
C307                VI(J,P)=VIJP-S*(VIJQ*CT-VRJQ*ST+VIJP*TAU)
C308          360   VI(J,Q)=VIJQ+S*(VIJP*CT+VRJP*ST-VIJQ*TAU)
C309          400   ROT=ROT+1
C310                GO TO 420
C311          410   AR(P,Q)=0.0
C312                AI(P,Q)=0.0
C313          420   CONTINUE
C314          500   CONTINUE
C315                DO 510 P=1,N
C316                B(P)=B(P)+Z(P)
C317                D(P)=B(P)
C318          510   Z(P)=0.0
C319          600   CONTINUE
C320                ROT=-ROT
C321          610   DO 650 P=1,N-1
C322                DO 650 Q=P+1,N
C323                TEMP=ABS(D(P))-ABS(D(Q))
C324                IF (ABS(TEMP) .LE. EPS2) GO TO 640
C325                IF (TEMP) 620,640,640
C326          620   SM=D(P)
C327                D(P)=D(Q)
C328                D(Q)=SM
C329                DO 630 I=1,N
C330                SM=VR(I,P)
C331                VR(I,P)=VR(I,Q)
C332                VR(I,Q)=SM
C333                SM=VI(I,P)
C334                VI(I,P)=VI(I,Q)
C335          630   VI(I,Q)=SM
C336          640   CONTINUE
C337          650   CONTINUE
C338                RETURN
C339                END

END OF SEGMENT, LENGTH  1174, NAME  HHJO
```

```
C340                    SUBROUTINE CORTHO (XR,XI,DIMN,DIMP,N,P,F,MC)
C341                    INTEGER DIMN,DIMP,P,F
C342                    REAL XR(DIMN,DIMP),XI(DIMN,DIMP), MC
C343                    DOUBLE PRECISION SUM
C344                    DO 30 K=F+1,P
C345                 10 SUM=0.0
C346                    IF (K .EQ. 1) GO TO 40
C347                    DO 30 I=1,K-1
C348                    CALL HINP (XR,XI,DIMN,DIMP,N,I,K,SR,SI)
C349                    CALL CABS (SR,SI,S)
C350                    SUM=SUM+S*S
C351                    DO 20 J=1,N
C352                    TEMP   =XR(J,K)-SR*XR(J,I)+SI*XI(J,I)
C353                    XI(J,K)=XI(J,K)-SI*XR(J,I)-SR*XI(J,I)
C354                 20 XR(J,K)=TEMP
C355                 30 CONTINUE
C356                 40 CALL HINP (XR,XI,DIMN,DIMP,N,K,K,SR,SI)
C357                    CALL CABS (SR,SI,S)
C358                    SUM=SUM+S
C359                    T=SUM
C360                    IF (S .GT. T/100.0 .AND. T*MC .NE. 0.0) GO TO 50
C361                    WRITE (4,100)
C362                100 FORMAT (' WARNING 1 IN CORTHO')
C363                    IF (S*MC .NE. 0.0) GO TO 10
C364                    WRITE (4,110)
C365                110 FORMAT (' WARNING 2 IN CORTHO')
C366                    SR,SI=0.0
C367                    GO TO 60
C368                 50 CALL CSQRT(SR,SI,TR,TI)
C369                    CALL CDIV (1.0,0.0,TR,TI,SR,SI)
C370                 60 DO 70 J=1,N
C371                    TEMP   =SR*XR(J,K)-SI*XI(J,K)
C372                    XI(J,K)=SI*XR(J,K)+SR*XI(J,K)
C373                 70 XR(J,K)=TEMP
C374                 80 CONTINUE
C375                    RETURN
C376                    END

END OF SEGMENT, LENGTH   307, NAME  CORTHO


C377                    SUBROUTINE HINP (XR,XI,DIMN,DIMP,N,K,L,SR,SI)
C378                    INTEGER DIMN,DIMP
C379                    REAL XR(DIMN,DIMP),XI(DIMN,DIMP)
C380                    DOUBLE PRECISION SUMR,SUMI
C381                    SUMR,SUMI=0.0
C382                    DO 10 I=1,N
C383                    SUMR=SUMR+XR(I,K)*XR(I,L)+XI(I,K)*XI(I,L)
C384                 10 SUMI=SUMI+XR(I,K)*XI(I,L)-XI(I,K)*XR(I,L)
C385                    SR=SUMR
C386                    SI=SUMI
C387                    RETURN
C388                    END

END OF SEGMENT, LENGTH   161, NAME  HINP


C389                    SUBROUTINE CABS (ZR,ZI,R)
C390                    XR=ABS(ZR)
C391                    XI=ABS(ZI)
C392                    IF (XI-XR) 20,20,10
C393                 10 R=XR
C394                    XR=XI
C395                    XI=R
C396                 20 IF (XI) 30,40,30
C397                 30 R=XI/XR
C398                    R=XR*SQRT(1.0+R*R)
C399                    RETURN
C400                 40 R=XR
C401                    RETURN
C402                    END

END OF SEGMENT, LENGTH    66, NAME  CABS


C403                    SUBROUTINE CDIV (XR,XI,WR,WI,ZR,ZI)
C404                    YR=WR
C405                    YI=WI
C406                    IF (ABS(YR)-ABS(YI)) 20,20,10
C407                 10 H=YI/YR
C408                    YR=WI*YI+YR
C409                    ZR=(XR+H*XI)/YR
C410                    ZI=(XI-H*XR)/YR
C411                    RETURN
C412                 20 H=YR/YI
C413                    YI=WI*YR+YI
C414                    ZR=(H*XR+XI)/YI
C415                    ZI=(H*XI-XR)/YI
C416                    RETURN
C417                    END

END OF SEGMENT, LENGTH    97, NAME  CDIV
```

```
0418                    SUBROUTINE CSQRT (ZR,ZI,YR,YI)
0419                    XR=ZR
0420                    XI=ZI
0421                    CALL CABS (XR,XI,H)
0422                    H=SQRT((ABS(XR)+H)/2.0)
0423                    IF (XI) 10,20,10
0424                 10 XI=XI/(2.0*H)
0425                 20 IF (XR) 40,30,30
0426                 30 XR=H
0427                    GO TO 70
0428                 40 IF (XI) 60,50,50
0429                 50 XR=XI
0430                    XI=H
0431                    GO TO 70
0432                 60 XR=-XI
0433                    XI=-H
0434                 70 YR=XR
0435                    YI=XI
0436                    RETURN
0437                    END
```

END OF SEGMENT, LENGTH   88, NAME CSQRT

```
0438                    SUBROUTINE RANDOMISATION (X,DIMN,DIMP,N,L,Z)
0439                    INTEGER DIMN,DIMP
0440                    REAL X(DIMN,DIMP)
0441                    DO 10 I=1,N
0442                 10 X(I,L)=2.0*FPMCRV(Z)-1.0
0443                    Z=FPMCRV(Z)
0444                    RETURN
0445                    END
```

END OF SEGMENT, LENGTH   67, NAME RANDOMISATION

```
0446                    SUBROUTINE INFO (D,E,DIMP,P,KS,G,H,EM)
0447                    INTEGER DIMP,DIMP,STREAM,PREV,G,H,EM,P,LINE(63)
0448                    REAL E(DIMP), D(DIMP)
0449                    DATA LINE/63*1H /, PREV/0/
0450                    COMMON NO,FIELD,LENGTH,K
0451                    IF (K) 200,200,100
0452                 100 IF (PREV .EQ. NO) GO TO 120
0453                    WRITE (4,110) NO, (LINE(I), I=1,63)
0454                 110 FORMAT (1H1,///,20X,'INFORMATIONAL OUTPUT ON LOGICAL STREAM 4 FOR E
0455                    1XAMPLE NUMBER',I3,/,20X,63A1)
0456                 120 WRITE (4,130) KS,G,H,EM
0457                 130 FORMAT (1H ,//,43X,'NUMBER OF  STEPS  PERFORMED =',I6,/,
0458                    1 43X,'NUMBER OF EIGENVECTORS ACCEPTED =',I3,/,
0459                    2 43X,'NUMBER OF EIGENVALUES  ACCEPTED =',I3,/,
0460                    3 43X,'NUMBER OF SOLNS. TO BE COMPUTED =',I3)
0461                    GO TO 300
0462                 200 IF (PREV .EQ. NO) GO TO 220
0463                    WRITE (4,210) NO, (LINE(I), I=1,42)
0464                 210 FORMAT (1H ,///////,' INFORMATIONAL OUTPUT FOR EXAMPLE NUMBER',I3,/
0465                    1,1X,42A1)
0466                 220 WRITE (4,230) KS,G,H,EM
0467                 230 FORMAT (1H ,///,' STEPS PERFORMED =',I4,/,' EIGENVECTORS ACCEPTED =
0468                    1',I3,/,' EIGENVALUES  ACCEPTED =',I3,/,' SOLNS. TO BE COMPUTED =',
0469                    2I3)
0470                 300 WRITE (4,310)
0471                 310 FORMAT (' THE ERROR VECTOR IS:')
0472                    CALL MXOP (E,1,DIMP,1,P,FIELD,LENGTH,4,K)
0473                    WRITE (4,320)
0474                 320 FORMAT (' THE APPROXIMATIONS TO THE EIGENVALUES ARE:')
0475                    CALL MXOP (D,1,DIMP,1,P,FIELD,LENGTH,4,0)
0476                    PREV=NO
0477                    RETURN
0478                    END
```

END OF SEGMENT, LENGTH   173, NAME INFO

```
C479                  SUBROUTINE NORMALISATION (X,Y,DIMN,DIMP,N,P)
C480                  INTEGER DIMN,DIMP,P
C481                  REAL X(DIMN,DIMP), Y(DIMN,DIMP)
C482                  DO 30 J=1,P
C483                  SR=X(1,J)
C484                  SI=Y(1,J)
C485                  AUX=SR*SR+SI*SI
C486                  DO 10 I=2,N
C487                  TR=X(I,J)
C488                  TI=Y(I,J)
C489                  ZZ=TR*TR+TI*TI
C490                  IF (ZZ .LE. AUX) GO TO 10
C491                  SR=TR
C492                  SI=TI
C493                  AUX=ZZ
C494               10 CONTINUE
C495                  CALL CDIV (1.0,0.0,SR,SI,TR,TI)
C496                  DO 20 I=1,N
C497                  SR    =TR*X(I,J)-TI*Y(I,J)
C498                  Y(I,J)=TI*X(I,J)+TR*Y(I,J)
C499               20 X(I,J)=SR
C500               30 CONTINUE
C501                  RETURN
C502                  END
```

END OF SEGMENT, LENGTH 176, NAME NORMALISATION

```
C503                  SUBROUTINE ELAPSE (E,DAT,NO,K)
C504                  INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
C505                  DATA CALL/2/
C506                  DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
C507                 1 'OCT','NOV','DEC'/
C508                  GO TO (100,200,300), NO
C509              100 CALL MTIME (N)
C510                  E=N
C511                  CALL=3-CALL
C512                  IF (CALL .EQ. 2) GO TO 110
C513                  EE=E
C514                  RETURN
C515              110 E=E-EE
C516                  E=E+1.0E-3
C517                  RETURN
C518              200 CALL DEFBUF (2,8,BUFFER)
C519                  CALL DATE (E)
C520                  WRITE (2,210) E
C521              210 FORMAT (A8)
C522                  READ (2,220) (DAT(I), I=1,3)
C523              220 FORMAT (I2,1X,I2,1X,I2)
C524                  DAT(2)=MONTH(DAT(2))
C525                  E,EE=1.0E-08
C526              230 IF (1.0+EE .EQ. 1.0) GO TO 240
C527                  E=EE
C528                  EE=EE/2.0
C529                  GO TO 230
C530              240 RETURN
C531              300 IF (K) 370,370,310
C532              310 IF (E .GE. 60.0) GO TO 330
C533                  WRITE (6,320) E
C534              320 FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
C535                  GO TO 390
C536              330 IF (E .GE. 120.0) GO TO 350
C537                  E=E-60.0
C538                  WRITE (6,340) E
C539              340 FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
C540                 1)
C541                  GO TO 390
C542              350 I=E/60.0
C543                  E=E-I*60.0
C544                  WRITE (6,360) I, E
C545              360 FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
C546                 1ONDS')
C547                  GO TO 390
C548              370 WRITE (6,380) E
C549              380 FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
C550              390 RETURN
C551                  END
```

END OF SEGMENT, LENGTH 182, NAME ELAPSE

```
C552                    SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
C553                    INTEGER DIMN,DIMP,P,WIDTH,STREAM
C554                    REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
C555                    DATA FRMT1/3H(        ,1HN,8HX,1P     ,1HN,1HE,8H)       /
C556                    DATA FRMT2/3H(        ,1HN,8HX,1P     ,1HN,1HE,8H)       /
C557                    IF (IFLAG) 10,200,10
C558                 10 CALL DEFBUF (2,16,BUFFER)
C559                    FRMT1(5), FRMT2(5)=FIELD
C560                    WRITE (2,20) FIELD
C561                 20 FORMAT (A6)
C562                    READ (2,30) WIDTH
C563                 30 FORMAT (1X,I2)
C564                    IF (P*WIDTH-LENGTH) 100,100,40
C565                 40 N1=LENGTH/WIDTH
C566                    N2=P/N1
C567                    LAST=P-N1*N2
C568                    IF (IFLAG) 50,999,60
C569                 50 N3=1
C570                    N4=1
C571                    GO TO 70
C572                 60 N3=(LENGTH-N1*WIDTH)/2+1
C573                    N4=(LENGTH-LAST*WIDTH)/2+1
C574                 70 WRITE (2,80) N3,N1,N4,LAST
C575                 80 FORMAT (4I4)
C576                    READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
C577                 90 FORMAT (4A4)
C578                    GO TO 300
C579                100 N2=0
C580                    IF (IFLAG) 110,999,120
C581                110 N3=1
C582                    GO TO 130
C583                120 N3=(LENGTH-P*WIDTH)/2+1
C584                130 WRITE (2,140) N3,P
C585                140 FORMAT (2I4)
C586                    READ (2,150) FRMT1(2), FRMT1(4)
C587                150 FORMAT (2A4)
C588                    GO TO 210
C589                200 IF (N2) 999,210,300
C590                210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
C591                    RETURN
C592                300 N2N1=N2*N1
C593                    IF (N-1) 999,400,310
C594                310 DO 330 K=1,N2N1,N1
C595                    KK=K+N1-1
C596                    WRITE (STREAM,320) K, KK
C597                320 FORMAT (1H ,/,' COLUMNS',I3,' TO',I3,' ARE:')
C598                330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
C599                    IF (LAST-1) 370,340,360
C600                340 WRITE (STREAM,350) P
C601                350 FORMAT (1H ,/,' COLUMN',I3,' IS:')
C602                    WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
C603                    GO TO 370
C604                360 KK=N2N1+1
C605                    WRITE (STREAM,320) KK,P
C606                    WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
C607                370 RETURN
C608                400 DO 410 K=1,N2N1,N1
C609                    KK=K+N1-1
C610                410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
C611                    IF (LAST-1) 440,420,430
C612                420 WRITE (STREAM,FRMT2) A(1,P)
C613                    GO TO 440
C614                430 KK=N2N1+1
C615                    WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
C616                440 RETURN
C617                999 STOP MXOP
C618                    END

END OF SEGMENT, LENGTH   427, NAME  MXOP
```

```
C619                    SUBROUTINE PRODUCT (VR,VI,WR,WI,DIMN,N)
C620                    INTEGER DIMN,PREV
C621                    REAL VR(DIMN), VI(DIMN), WR(DIMN), WI(DIMN)
C622                    REAL AR(12,12),AI(12,12)
C623                    DOUBLE PRECISION SUMR,SUMI
C624                    DATA PREV/0/
C625                    COMMON NO
C626                    GO TO (100,200,300,400,500,600,700,800,900), NO
C627                100 CONTINUE
C628                    DO 120 I=1,N
C629                    SUMR,SUMI=0.0
C630                    DO 110 J=1,N
C631                    SUMR=SUMR+(1.0/FLOAT(I+J-1))*VR(J)
C632                110 SUMI=SUMI+(1.0/FLOAT(I+J-1))*VI(J)
C633                    WR(I)=SNGL(SUMR)
C634                120 WI(I)=SNGL(SUMI)
C635                    RETURN
C636                200 CONTINUE
C637                    IF (PREV .EQ. NO) GO TO 220
C638                    READ (5,210) ((AR(I,J),J=1,N),I=1,N), ((AI(I,J),J=1,N),I=1,N)
C639                210 FORMAT (4G0.0)
C640                    PREV=NO
C641                220 DO 240 I=1,N
C642                    SUMR,SUMI=0.0
C643                    DO 230 J=1,N
C644                    SUMR=SUMR+AR(I,J)*VR(J)-AI(I,J)*VI(J)
C645                230 SUMI=SUMI+AR(I,J)*VI(J)+AI(I,J)*VR(J)
C646                    WR(I)=SNGL(SUMR)
C647                240 WI(I)=SNGL(SUMI)
C648                    RETURN
C649                300 CONTINUE
C650                    GO TO 200
C651                400 CONTINUE
C652                    X1=SQRT(FLOAT(N-1))
C653                    WR(1)=-X1*VI(2)
C654                    WI(1)=X1*VR(2)
C655                    DO 410 K=2,N-1
C656                    Y1=-X1
C657                    X1=SQRT(FLOAT(K*(N-K)))
C658                    WR(K)=-Y1*VI(K-1)-X1*VI(K+1)
C659                410 WI(K)=Y1*VR(K-1)+X1*VR(K+1)
C660                    Y1=-X1
C661                    WR(N)=-Y1*VI(N-1)
C662                    WI(N)=Y1*VR(N-1)
C663                    RETURN
C664                500 CONTINUE
C665                    GO TO 400
C666                600 CONTINUE
C667                700 CONTINUE
C668                800 CONTINUE
C669                900 CONTINUE
C670                    RETURN
C671                    END

END OF SEGMENT, LENGTH   416, NAME  PRODUCT
```

THE METHOD OF "QUICK RITZ" ITERATION FOR AN HERMITIAN MATRIX
****

THIS PROGRAM WAS RUN ON 22 AUG 74
AND EXECUTION STARTED AT 18/26/47

EXAMPLE NUMBER 4

2 EIGENVALUES AND 2 EIGENVECTORS HAVE BEEN ACCEPTED TO AN ACCURACY OF 1.00E-08

THE 4 APPROXIMATIONS TO THE EIGENVALUES ARE:
1.1000000000E 01 -1.1000000000E 01 8.9999997437E 00 -8.9999997438E 00

THE MATRIX OF CORRESPONDING EIGENVECTORS IS GIVEN BELOW:

REAL PART

-8.2091321537E-12	-8.7534865046E-12	-1.7127066929E-01	-1.7127066929E-01
1.5430336922E-01	1.5430335000E-01	-1.5700165022E-05	-1.5675361285E-05
2.2645191134E-10	2.2555468604E-10	8.0829311214E-01	8.0829311215E-01
-5.9761430470E-01	-5.9761430470E-01	1.2535947462E-05	1.2482574675E-05
-7.7123150710E-10	-7.6943251770E-10	-8.4852888188E-01	-8.4852888188E-01
1.0000000000E 00	9.9999999999E-01	1.4088731415E-05	1.4105597074E-05
9.5478842623E-10	9.5496943686E-10	-3.3466577230E-01	-3.3466577231E-01
-8.4515425476E-01	-8.4513425474E-01	-2.9110933610E-06	-2.8658068914E-06
-4.7393724492E-10	-4.7858421313E-10	1.0000000000E 00	9.9999999999E-01
3.4503277972E-01	3.4503277973E-01	-1.8254473616E-05	-1.8297614588E-05
6.5283618523E-11	6.5283618528E-11	-1.6475705012E-01	-1.6475705015E-01
-4.6524210536E-02	-4.6524210537E-02	7.2096290751E-06	7.2187713158E-06

IMAGINARY PART

4.6524210567E-02	-4.6524210567E-02	-4.3720115173E-07	4.3720228859E-07
-1.1059455574E-02	-1.1059360827E-09	4.6474621231E-01	-4.6474620662E-01
-3.4503277972E-01	3.4503277980E-01	1.3264270914E-06	-1.3264434529E-06
-1.2860255083E-09	1.2860255083E-09	-9.9999160902E-01	9.9999159681E-01
8.4515425473E-01	-8.4513425473E-01	-3.6101846490E-07	3.6102937884E-07
0.0000000000E-01	0.0000000000E-01	3.3467381090E-01	-3.3467580683E-01
-9.9999999993E-01	9.9999999996E-01	-8.5429246610E-07	8.5429928731E-07
-6.0026650317E-10	5.9844751377E-10	8.4852719063E-01	-8.4852718026E-01
5.9761430475E-01	-5.9761430477E-01	-9.0949470177E-13	-3.6379788071E-12
1.4761099010E-09	-1.4742909116E-09	-8.0830627629E-01	8.0830626640E-01
-1.5430335007E-01	1.5430335007E-01	4.9137438403E-07	-4.9136847232E-07
-4.9828940973E-10	4.9828940973E-10	1.7127580671E-01	-1.7127580461E-01

THE 4 SETS OF CORRESPONDING ERRORS ARE:
-1.0646796506E-09 -1.0579200848E-09 9.2991897273E-05 9.2647140912E-05

EXECUTION TIME WAS 10.408 SECONDS

****

INFORMATIONAL OUTPUT ON LOGICAL STREAM 4 FOR EXAMPLE NUMBER 4
****

NUMBER OF STEPS PERFORMED = 2
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS;
-1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
5.0620590941E 00 -3.0620590941E 00 -6.2422322079E-01 6.2422322079E-01

NUMBER OF STEPS PERFORMED = 4
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS;
-1.7194316868E-01 5.4624138098E-02 5.3217739960E-02 -2.5910664184E-01
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
9.3136477139E 00 -9.3136477138E 00 4.9117095301E-02 -4.9117095438E-02

NUMBER OF STEPS PERFORMED = 6
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS;
3.7199937631E-02 -2.2558262016E-02 1.2537291467E-02 -2.6984363223E-02
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
1.0507681491E 01 -1.0507681490E 01 2.3345559450E 00 -2.3345559451E 00

NUMBER OF STEPS PERFORMED = 8
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS;
-1.2653731127E-02 -4.2539078661E-03 5.2428331290E-02 6.2035046435E-02
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
1.0781676456E 01 -1.0781676456E 01 4.4663241648E 00 -4.4663241649E 00

NUMBER OF STEPS PERFORMED = 11
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS;
-1.6099934182E-04 1.4608257652E-02 -4.9370839009E-03 1.5076986457E-01
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
1.0914072784E 01 -1.0914072785E 01 6.4054766265E 00 -6.4054766266E 00

NUMBER OF STEPS PERFORMED = 15
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTOR IS;
1.3665648387E-02 9.4095931485E-03 2.3760354099E-02 5.9497108053E-02
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
1.0986372497E 01 -1.0986372497E 01 8.2024609946E 00 -8.2024609946E 00

```
                                    NUMBER OF  STEPS  PERFORMED =    20
                                    NUMBER OF EIGENVECTORS ACCEPTED =  0
                                    NUMBER OF EIGENVALUES  ACCEPTED =  0
                                    NUMBER OF SOLNS. TO BE COMPUTED =  2
THE ERROR VECTOR IS;
                  4.4064604862E-03  3.3430029225E-03  5.8350554635E-02  4.9680002496E-02
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
                  1.0999454808E 01 -1.0999454808E 01  8.8808129508E 00 -8.8808129509E 00

                                    NUMBER OF  STEPS  PERFORMED =    26
                                    NUMBER OF EIGENVECTORS ACCEPTED =  0
                                    NUMBER OF EIGENVALUES  ACCEPTED =  0
                                    NUMBER OF SOLNS. TO BE COMPUTED =  2
THE ERROR VECTOR IS;
                  3.0639562285E-04  3.4992404333E-04 -4.3186286769E-03 -1.9461010748E-03
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
                  1.0999993625E 01 -1.0999993625E 01  8.9901743652E 00 -8.9901743653E 00

                                    NUMBER OF  STEPS  PERFORMED =    33
                                    NUMBER OF EIGENVECTORS ACCEPTED =  0
                                    NUMBER OF EIGENVALUES  ACCEPTED =  0
                                    NUMBER OF SOLNS. TO BE COMPUTED =  2
THE ERROR VECTOR IS;
                  5.5114327047E-06  6.0372618911E-06 -1.3349613392E-03 -1.0989136154E-03
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
                  1.0999999972E 01 -1.0999999972E 01  8.9951177449E 00 -8.9951177449E 00

                                    NUMBER OF  STEPS  PERFORMED =    41
                                    NUMBER OF EIGENVECTORS ACCEPTED =  0
                                    NUMBER OF EIGENVALUES  ACCEPTED =  2
                                    NUMBER OF SOLNS. TO BE COMPUTED =  2
THE ERROR VECTOR IS;
                  4.2793994347E-08  4.0457826766E-08  5.9779306235E-04  5.8691298552E-04
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
                  1.1000000000E 01 -1.1000000000E 01  8.9999857018E 00 -8.9999857020E 00

                                    NUMBER OF  STEPS  PERFORMED =    50
                                    NUMBER OF EIGENVECTORS ACCEPTED =  2
                                    NUMBER OF EIGENVALUES  ACCEPTED =  2
                                    NUMBER OF SOLNS. TO BE COMPUTED =  2
THE ERROR VECTOR IS;
                 -1.0646796306E-09 -1.0370200848E-09  9.2991897273E-05  9.2647140912E-05
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
                  1.1000000000E 01 -1.1000000000E 01  8.9999997437E 00 -8.9999997438E 00
```

APPENDIX 7

A RITZ ITERATION PROGRAM FOR GENERAL MATRICES

```
0023                  MASTER
0024                  INTEGER DIMN,DIMP,P,TYPE,EM,G,H,LINE(60),DATE(3)
0025                  REAL YR(20,6),YI(20,6),XR(20,6),XI(20,6)
0026                  REAL VR(20),VI(20),WR(20),WI(20)
0027                  REAL LVR(6,6),LVI(6,6),RVR(6,6),RVI(6,6),BR(6,6),BI(6,6)
0028                  REAL DR(6),DI(6),LF(6),RF(6),EN(6),DOLD(6,2),LARGE(6,2),MC
0029                  COMMON TYPE,NO,FIELD,LENGTH,K
0030                  DATA LINE/60*1H*/
0031                  K=1
0032                  CALL ELAPSE (E,DATE,2,K)
0033                  MC=E
0034                  READ (5,10) NUMBER,DIMN,DIMP,LENGTH
0035               10 FORMAT (I2,I5,I2,I3)
0036                  IF (LENGTH .LT. 120) K=-1
0037                  DO 500 NO=1,NUMBER
0038                  READ (5,20) N,P,KU,EM,EPS,FIELD,TYPE,NORM
0039               20 FORMAT (I5,I2,I6,I2,E9.2,A6,2I1)
0040                  IF (KU .LT. 0) READ (5,30) ((YR(I,J),YI(I,J), J=1,P), I=1,N),
0041                 1 ((XR(I,J),XI(I,J), J=1,P), I=1,N)
0042               30 FORMAT (2G0.0)
0043                  CALL TIME (T)
0044                  IF (K) 120,999,100
0045              100 WRITE (6,110) (LINE(I), I=1,60), (DATE(I), I=1,3), T, NO
0046              110 FORMAT (1H1,///,31X,'THE METHOD OF "QUICK RITZ" ITERATION FOR AN AR
0047                 1BITRARY MATRIX',//,31X,60A1,///,45X,'THIS PROGRAM WAS RUN ON',I3,1X
0048                 2,A3,1X,I2,//,45X,'AND EXECUTION STARTED AT',A9,//,54X,'EXAMPLE NUMB
0049                 3ER',I3)
0050                  GO TO 140
0051              120 WRITE (6,130) (LINE(I), I=1,35), (DATE(I), I=1,3), T, NO
0052              130 FORMAT (1H ,//////,' "QUICK RITZ" FOR ARBITRARY MATRICES',//,1X,35A
0053                 11,//,' DATE:',I3,1X,A3,1X,I2,5X,'TIME:',A9,5X,'EXAMPLE NUMBER',I3)
0054              140 CALL ELAPSE (E,DATE,1,K)
0055                  CALL GLRZ (YR,YI,XR,XI,VR,VI,WR,WI,LVR,LVI,RVR,RVI,BR,BI,
0056                 1 DR,DI,LF,RF,EN,DOLD,LARGE,DIMN,DIMP,N,P,EM,KM,KS,H,G,MC,EPS)
0057                  CALL ELAPSE (E,DATE,1,K)
0058                  IF (NORM .EQ. 0) GO TO 150
0059                  CALL NORMALISATION (YR,YI,DIMN,DIMP,N,P)
0060                  CALL NORMALISATION (XR,XI,DIMN,DIMP,N,P)
0061              150 IF (K) 220,999,200
0062              200 WRITE (6,210) H,G,EPS
0063              210 FORMAT (1H ,////,21X,I2,' EIGENVALUES AND',I3,' EIGENVECTORS HAVE
0064                 1BEEN ACCEPTED TO AN ACCURACY OF',1PE9.2)
0065                  GO TO 240
0066              220 WRITE (6,230) H,G,EPS
0067              230 FORMAT (1H ,////,1X,I2,' EIGENVALUES AND',I3,' EIGENVECTORS HAVE B
0068                 1EEN ACCEPTED TO',1PE9.2)
0069              240 WRITE (6,300) P
0070              300 FORMAT (1H ,////,' THE',I3,' APPROXIMATIONS TO THE EIGENVALUES ARE
0071                 1:')
0072                  WRITE (6,310)
0073              310 FORMAT (1H ,/,' REAL PART')
0074                  CALL MXOP (DR,1,DIMP,1,P,FIELD,LENGTH,6,K)
0075                  WRITE (6,311)
0076              311 FORMAT (1H ,/,' IMAGINARY PART')
0077                  CALL MXOP (DI,1,DIMP,1,P,FIELD,LENGTH,6,0)
0078                  WRITE (6,320)
0079              320 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING LEFT-HAND EIGENVECT
0080                 1ORS IS GIVEN BELOW:')
0081                  WRITE (6,310)
0082                  CALL MXOP (YR,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
0083                  WRITE (6,311)
0084                  CALL MXOP (YI,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
0085                  WRITE (6,330)
0086              330 FORMAT (1H ,////,' THE MATRIX OF CORRESPONDING RIGHT-HAND EIGENVEC
0087                 1TORS IS GIVEN BELOW:')
0088                  WRITE (6,310)
0089                  CALL MXOP (XR,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
0090                  WRITE (6,311)
0091                  CALL MXOP (XI,DIMN,DIMP,N,P,FIELD,LENGTH,6,0)
0092                  WRITE (6,340) P
0093              340 FORMAT (1H ,////,' THE',I3,' SETS OF CORRESPONDING ERRORS ARE:')
0094                  WRITE (6,350)
0095              350 FORMAT (' LEFT-HAND')
0096                  CALL MXOP (LF,1,DIMP,1,P,FIELD,LENGTH,6,0)
0097                  WRITE (6,351)
0098              351 FORMAT (' RIGHT-HAND')
0099                  CALL MXOP (RF,1,DIMP,1,P,FIELD,LENGTH,6,0)
0100                  CALL ELAPSE (E,DATE,3,K)
0101                  IF (K) 420,999,400
0102              400 WRITE (6,410) (LINE(I), I=1,60)
0103              410 FORMAT (1H ,////,31X,60A1)
0104                  GO TO 500
0105              420 WRITE (6,430) (LINE(I), I=1,40)
0106              430 FORMAT (1H ,////,1X,40A1)
0107              500 CONTINUE
0108                  STOP OK
0109              999 STOP NONM
0110                  END

END OF SEGMENT, LENGTH   475, NAME  NONM
```

```
      SUBROUTINE GLRZ (YR,YI,XR,XI,VR,VI,WR,WI,LVR,LVI,RVR,RVI,BR,BI,
     1 DR,DI,LF,RF,EN,DOLD,LARGE,DIMN,DIMP,N,P,EM,KM,KS,H,G,MC,EPS)
      INTEGER DIMN,DIMP,P,EM,H,G,M1,G1
      REAL YR(DIMN,DIMP),YI(DIMN,DIMP), XR(DIMN,DIMP),XI(DIMN,DIMP)
      REAL VR(DIMN),VI(DIMN), WR(DIMN),WI(DIMN)
      REAL LVR(DIMP,DIMP),LVI(DIMP,DIMP), RVR(DIMP,DIMP),RVI(DIMP,DIMP)
      REAL BR(DIMP,DIMP),BI(DIMP,DIMP), DR(DIMP),DI(DIMP)
      REAL LF(DIMP),RF(DIMP), EN(DIMP), DOLD(DIMP,2), LARGE(DIMP,2), MC
      DOUBLE PRECISION SUMR,SUMI
      DO 10 I=1,P
      DOLD(I,1),DOLD(I,2)=0.0
   10 LARGE(I,1),LARGE(I,2)=0.0
C 38 IS 7672 (76 IS LARGEST EXPONENT ON M/C)
      CONST=38.0-ALOG10(FLOAT(N))
      KS,G,H,G1,M1=0
      M=2
      ZZ=0.1
      EPS1=SQRT(EPS)
      EPS2=10.0*EPS
      IF (KM .LT. 0) GO TO 40
      DO 20 J=1,P
   20 CALL RANDOMISATION (YR,DIMN,DIMP,N,J,ZZ)
      DO 30 J=1,P
      DO 30 I=1,N
      XR(I,J)=YR(I,J)
      YI(I,J)=0.0
   30 XI(I,J)=0.0
   40 CALL BIORTHO (YR,YI,XR,XI,DIMN,DIMP,N,P,0,MC)
      KM=IABS(KM)
C FORM B=Y*AX
  100 DO 140 K=G+1,P
      DO 110 I=1,N
      VR(I)=XR(I,K)
  110 VI(I)=XI(I,K)
      CALL PRODUCT (VR,VI,WR,WI,DIMN,N,-1)
      DO 120 I=1,N
      XR(I,K)=WR(I)
  120 XI(I,K)=WI(I)
      DO 140 J=G+1,P
      SUMR,SUMI=0.0
      DO 130 I=1,N
      SUMR=SUMR+YR(I,J)*WR(I)+YI(I,J)*WI(I)
  130 SUMI=SUMI+YR(I,J)*WI(I)-YI(I,J)*WR(I)
      BR(J-G,K-G)=SUMR
  140 BI(J-G,K-G)=SUMI
      DO 160 K=G+1,P
      DO 150 I=1,N
      VR(I)=YR(I,K)
  150 VI(I)=YI(I,K)
      CALL PRODUCT (VR,VI,WR,WI,DIMN,N,1)
      DO 160 I=1,N
      YR(I,K)=WR(I)
  160 YI(I,K)=WI(I)
C SOLVE E-VALUE PROBLEM FOR B I.E. WAV=D
      CALL GLJO (BR,BI,LVR,LVI,RVR,RVI,EN,DR,DI,DIMP,P-G,MC,EPS2,I)
      IF (G .EQ. 0) GO TO 200
      DO 180 I=G+1,P
      J=P+1-I
      DR(J+G)=DR(J)
  180 DI(J+G)=DI(J)
      DO 190 I=1,G
      DR(I)=DOLD(I,1)
  190 DI(I)=DOLD(I,2)
C FORM Y=YW, X=XV
  200 DO 230 I=1,N
      DO 220 J=G+1,P
      SUMR,SUMI=0.0
      DO 210 K=G+1,P
      SUMR=SUMR+XR(I,K)*RVR(K-G,J-G)-XI(I,K)*RVI(K-G,J-G)
  210 SUMI=SUMI+XR(I,K)*RVI(K-G,J-G)+XI(I,K)*RVR(K-G,J-G)
      VR(J)=SUMR
  220 VI(J)=SUMI
      DO 230 J=G+1,P
      XR(I,J)=VR(J)
  230 XI(I,J)=VI(J)
      DO 260 I=1,N
      DO 250 J=G+1,P
      SUMR,SUMI=0.0
      DO 240 K=G+1,P
      SUMR=SUMR+YR(I,K)*LVR(K-G,J-G)+YI(I,K)*LVI(K-G,J-G)
  240 SUMI=SUMI-YR(I,K)*LVI(K-G,J-G)+YI(I,K)*LVR(K-G,J-G)
      VR(J)=SUMR
  250 VI(J)=SUMI
      DO 260 J=G+1,P
      YR(I,J)=VR(J)
  260 YI(I,J)=VI(J)
```

```
C PERFORM M PREMULTIPLICATIONS.
      DO 310 K=1,M-1
      DO 310 J=G+1,P
      DO 300 I=1,N
      VR(I)=XR(I,J)
  300 VI(I)=XI(I,J)
      CALL PRODUCT (VR,VI,WR,WI,DIMN,N,-1)
      DO 310 I=1,N
      XR(I,J)=WR(I)
  310 XI(I,J)=WI(I)
      DO 330 K=1,M-1
      DO 330 J=G+1,P
      DO 320 I=1,N
      VR(I)=YR(I,J)
  320 VI(I)=YI(I,J)
      CALL PRODUCT (VR,VI,WR,WI,DIMN,N,1)
      DO 330 I=1,N
      YR(I,J)=WR(I)
  330 YI(I,J)=WI(I)
  340 KS=KS+M
C FORM Y"X=I
      CALL BIORTHO (YR,YI,XR,XI,DIMN,DIMP,N,P,G,MC)
C CHECK STEPS AND CONVERGENCE
      IF (M .GE. P) GO TO 430
      DO 400 I=M+1,P
      IF (ABS(DR(I))-EPS1) 401,401,402
  401 ZZ=ABS(DOLD(I,1)-DR(I))
      GO TO 403
  402 ZZ=ABS((DOLD(I,1)-DR(I))/DR(I))
  403 IF (ABS(DI(I))-EPS1) 404,404,405
  404 AUX=ABS(DOLD(I,2)-DI(I))
      GO TO 406
  405 AUX=ABS((DOLD(I,2)-DI(I))/DI(I))
  406 IF (ZZ .GT. EPS .OR. AUX .GT. EPS) GO TO 410
  400 M1=M1+1
  410 M=M+M1
      M1=0
      DO 420 I=G+1,P
      DOLD(I,1)=DR(I)
  420 DOLD(I,2)=DI(I)
      DO 450 J=G+1,P
      AUX=SQRT(YR(1,J)**2+YI(1,J)**2)
      DO 440 I=2,N
      ZZ=SQRT(YR(I,J)**2+YI(I,J)**2)
      IF (ZZ .GT. AUX) AUX=ZZ
  440 CONTINUE
      LF(J)=(LARGE(J,1)-AUX)/AUX
  450 LARGE(J,1)=AUX
      DO 470 J=G+1,P
      AUX=SQRT(XR(1,J)**2+XI(1,J)**2)
      DO 460 I=2,N
      ZZ=SQRT(XR(I,J)**2+XI(I,J)**2)
      IF (ZZ .GT. AUX) AUX=ZZ
  460 CONTINUE
      RF(J)=(LARGE(J,2)-AUX)/AUX
  470 LARGE(J,2)=AUX
      DO 480 J=G+1,P
      IF (ABS(LF(J)) .GT. EPS .OR. ABS(RF(J)) .GT. EPS) GO TO 490
  480 G1=G1+1
  490 G=G+G1
      G1=0
      CALL INFO (DR,DI,LF,RF,DIMP,P,KS,G,M,EM)
      IF (G .LE. M) GO TO 510
      IF (G .LT. EM) GO TO 500
      G=M
      M=2
      GO TO 510
  500 G=M
  510 IF (G .GE. EM .OR. KS .GE. KM) GO TO 520
      CALL CABS (DR(G+1),DI(G+1),ZZ)
      CALL CABS (DR(P),DI(P),AUX)
      CONST1=CONST/ALOG10(ZZ)
      CONST2=M*ALOG10(ZZ/AUX)
      IF (FLOAT(M) .LT. CONST1 .AND. CONST2 .LT. 1.0) M=M+1
      GO TO 100
  520 DO 600 J=1,P
      DO 600 I=1,N
  600 YI(I,J)=-YI(I,J)
      RETURN
      END
```

END OF SEGMENT, LENGTH 1718, NAME GLRZ

```
0277                SUBROUTINE GLJO (AR,AI,WR,WI,VR,VI,EN,DDR,DDI,DIMN,N,MC,EPS2,ROT)
0278                INTEGER DIMN,ROT
0279                REAL AR(DIMN,DIMN),AI(DIMN,DIMN), WR(DIMN,DIMN),WI(DIMN,DIMN)
0280                REAL VR(DIMN,DIMN),VI(DIMN,DIMN), EN(DIMN), DDR(DIMN),DDI(DIMN)
0281                REAL MAX,ND,MC,ISW,MC
0282                MARK=0
0283                ROT=0
0284                EPS=10.0*N*(N-1)*MC
0285                DO 10 I=1,N-1
0286                WR(I,I),VR(I,I)=1.0
0287                WI(I,I),VI(I,I)=0.0
0288                DO 10 J=I+1,N
0289             10 WR(I,J),WR(J,I),WI(I,J),WI(J,I),
0290               1 VR(I,J),VR(J,I),VI(I,J),VI(J,I)=0.0
0291                WR(N,N),VR(N,N)=1.0
0292                WI(N,N),VI(N,N)=0.0
0293                DO 440 IT=1,50
0294                IF (MARK .EQ. 1) GO TO 450
0295                TAU=0.0
0296                DO 110 K=1,N
0297                TEM=0.0
0298                DO 100 I=1,N
0299                IF (I .EQ. K) GO TO 100
0300                ARIK=AR(I,K)
0301                AIIK=AI(I,K)
0302                TEM=ARIK*ARIK+AIIK*AIIK+TEM
0303           100  CONTINUE
0304                TAU=TAU+TEM
0305           110  CONTINUE
0306                IF (TAU .LE. EPS) GO TO 450
0307                MARK=1
0308                DO 430 K=1,N-1
0309                DO 430 M=K+1,N
0310                HJ,HR,HI,G=0.0
0311                DO 310 I=1,N
0312                IF (I .EQ. K .OR. I .EQ. M) GO TO 300
0313                ARKI=AR(K,I)
0314                AIKI=AI(K,I)
0315                ARMI=AR(M,I)
0316                AIMI=AI(M,I)
0317                ARIK=AR(I,K)
0318                AIIK=AI(I,K)
0319                ARIM=AR(I,M)
0320                AIIM=AI(I,M)
0321                HR=HR+ARKI*ARMI+AIKI*AIMI-ARIK*ARIM-AIIK*AIIM
0322                HJ=HJ+AIKI*ARMI-ARKI*AIMI-ARIK*AIIM+AIIK*ARIM
0323                TE= ARIK*ARIK+AIIK*AIIK+ARMI*ARMI+AIMI*AIMI
0324                TEE=ARIM*ARIM+AIIM*AIIM+ARKI*ARKI+AIKI*AIKI
0325                G=G+TE+TEE
0326                HJ=HJ-TE+TEE
0327           300  CONTINUE
0328           310  CONTINUE
0329                BR=AR(K,M)+AR(M,K)
0330                BI=AI(K,M)+AI(M,K)
0331                ER=AR(K,M)-AR(M,K)
0332                EI=AI(K,M)-AI(M,K)
0333                DR=AR(K,K)-AR(M,M)
0334                DI=AI(K,K)-AI(M,M)
0335                TE =BR*BR+EI*EI+DR*DR
0336                TEE=BI*BI+ER*ER+DI*DI
0337                IF (TE .LT. TEE) GO TO 320
0338                ISW=1.0
0339                C=BR
0340                S=EI
0341                D=DR
0342                DE=DI
0343                ROOT2=SQRT(TE)
0344                GO TO 330
0345           320  ISW=-1.0
0346                C=BI
0347                S=-ER
0348                D=DI
0349                DE=DR
0350                ROOT2=SQRT(TEE)
0351           330  ROOT1=SQRT(S*S+C*C)
0352                SIG=1.0
0353                IF (D .LT. 0.0) SIG=-1.0
0354                SA=0.0
0355                CA=1.0
0356                IF (C .LT. 0.0) CA=-1.0
0357                IF (ROOT1 .GE. EPS) GO TO 350
0358                SX,SA=0.0
0359                CX,CA=1.0
0360                E=ER
0361                B=BI
0362                IF (ISW .GT. 0.0) GO TO 340
0363                E=EI
0364                B=-BR
0365           340  ND=D*D+DE*DE
0366                GO TO 370
0367           350  IF (ABS(S) .LE. EPS) GO TO 360
0368                CA=C/ROOT1
0369                SA=S/ROOT1
0370           360  COT2X=D/ROOT1
0371                COTX=COT2X+(SIG*SQRT(1.0+COT2X*COT2X))
0372                SX=SIG/SQRT(1.0+COTX*COTX)
0373                CX=SX*COTX
0374                ETA=(ER*BR+BI*EI)/ROOT1
0375                TSE=(BR*BI-ER*EI)/ROOT1
```

```
C376                TE=SIG*(-ROOT1*DE+TSE*D)/ROOT2
C377                TEE=(D*DE+ROOT1*TSE)/ROOT2
C378                ND=ROOT2*ROOT2+TEE*TEE
C379                TEE=HJ*CX*SX
C380                COS2A=CA*CA-SA*SA
C381                SIN2A=2.0*CA*SA
C382                TEM=HR*COS2A+HI*SIN2A
C383                TEP=HI*COS2A-HR*SIN2A
C384                HR=CX*CX*HR-SX*SX*TEM-CA*TEE
C385                HI=CX*CX*HI+SX*SX*TEP-SA*TEE
C386                B=ISW*TE*CA+ETA*SA
C387                E=CA*ETA-ISW*TE*SA
C388          570   S=HR-SIG*ROOT2*E
C389                C=HI-SIG*ROOT2*B
C390                ROOT=SQRT(C*C+S*S)
C391                IF (ROOT .GE. EPS) GO TO 380
C392                CB,CH=1.0
C393                SB,SH=0.0
C394                GO TO 390
C395          580   CB=-C/ROOT
C396                SB=S/ROOT
C397                TEE=CB*B-E*SB
C398                NC=TEE*TEE
C399                TANH=ROOT/(G+2.0*(NC+ND))
C400                CH=1.0/SQRT(1.0-TANH*TANH)
C401                SH=CH*TANH
C402          590   TEM=SX*SH*(SA*CB-SB*CA)
C403                C1R=CX*CH-TEM
C404                C2R=CX*CH+TEM
C405                C1I,C2I=-SX*SH*(CA*CB+SA*SB)
C406                TEP=SX*CH*CA
C407                TEM=CX*SH*SB
C408                S1R=TEP-TEM
C409                S2R=-TEP-TEM
C410                TEP=SX*CH*SA
C411                TEM=CX*SH*CB
C412                S1I=TEP+TEM
C413                S2I=TEP-TEM
C414                TEM=SQRT(S1R*S1R+S1I*S1I)
C415                TEP=SQRT(S2R*S2R+S2I*S2I)
C416                IF (TEM .LT. EPS .AND. TEP .LT. EPS) GO TO 420
C417                MARK=0
C418                DO 400 I=1,N
C419                ARKI=AR(K,I)
C420                ARMI=AR(M,I)
C421                AIKI=AI(K,I)
C422                AIMI=AI(M,I)
C423                AR(K,I)=C1R*ARKI-C1I*AIKI+S1R*ARMI-S1I*AIMI
C424                AI(K,I)=C1R*AIKI+C1I*ARKI+S1R*AIMI+S1I*ARMI
C425                AR(M,I)=S2R*ARKI-S2I*AIKI+C2R*ARMI-C2I*AIMI
C426                AI(M,I)=S2R*AIKI+S2I*ARKI+C2R*AIMI+C2I*ARMI
C427                WRIK=WR(I,K)
C428                WRIM=WR(I,M)
C429                WIIK=WI(I,K)
C430                WIIM=WI(I,M)
C431                WR(I,K)=C1R*WRIK-C1I*WIIK+S1R*WRIM-S1I*WIIM
C432                WI(I,K)=C1R*WIIK+C1I*WRIK+S1R*WIIM+S1I*WRIM
C433                WR(I,M)=S2R*WRIK-S2I*WIIK+C2R*WRIM-C2I*WIIM
C434          400   WI(I,M)=S2R*WIIK+S2I*WRIK+C2R*WIIM+C2I*WRIM
C435                DO 410 I=1,N
C436                ARIK=AR(I,K)
C437                ARIM=AR(I,M)
C438                AIIK=AI(I,K)
C439                AIIM=AI(I,M)
C440                AR(I,K)=C2R*ARIK-C2I*AIIK-S2R*ARIM+S2I*AIIM
C441                AI(I,K)=C2R*AIIK+C2I*ARIK-S2R*AIIM-S2I*ARIM
C442                AR(I,M)=-S1R*ARIK+S1I*AIIK+C1R*ARIM-C1I*AIIM
C443                AI(I,M)=-S1R*AIIK-S1I*ARIK+C1R*AIIM+C1I*ARIM
C444                VRIK=VR(I,K)
C445                VRIM=VR(I,M)
C446                VIIK=VI(I,K)
C447                VIIM=VI(I,M)
C448                VR(I,K)=C2R*VRIK-C2I*VIIK-S2R*VRIM+S2I*VIIM
C449                VI(I,K)=C2R*VIIK+C2I*VRIK-S2R*VIIM-S2I*VRIM
C450                VR(I,M)=-S1R*VRIK+S1I*VIIK+C1R*VRIM-C1I*VIIM
C451          410   VI(I,M)=-S1R*VIIK-S1I*VRIK+C1R*VIIM+C1I*VRIM
C452          420   CONTINUE
C453                ROT=ROT+1
C454          430   CONTINUE
C455          440   CONTINUE
C456                ROT=-ROT
C457          450   DO 460 I=1,N
C458                DDR(I)=AR(I,I)
C459          460   DDI(I)=AI(I,I)
C460                DO 600 I=1,N
C461                DR=DDR(I)
C462                DI=DDI(I)
C463          600   EN(I)=DR*DR+DI*DI
C464                DO 660 K=1,N-1
C465                DO 660 L=K+1,N
C466                TEM=EN(K)-EN(L)
C467                IF (ABS(TEM) .LE. EPS2) GO TO 610
C468                IF (TEM) 630,650,650
C469          610   TEM=DDR(K)-DDR(L)
C470                IF (ABS(TEM) .LE. EPS2) GO TO 620
C471                IF (TEM) 630,650,650
C472          620   TEM=DDI(K)-DDI(L)
C473                IF (ABS(TEM) .LE. EPS2) GO TO 650
C474                IF (TEM) 630,650,650
```

```
0475              630 TEM=EN(K)
0476                  EN(K)=EN(L)
0477                  EN(L)=TEM
0478                  TEM=DDR(K)
0479                  DDR(K)=DDR(L)
0480                  DDR(L)=TEM
0481                  TEM=DDI(K)
0482                  DDI(K)=DDI(L)
0483                  DDI(L)=TEM
0484                  DO 640 I=1,N
0485                  TEM=WR(I,K)
0486                  WR(I,K)=WR(I,L)
0487                  WR(I,L)=TEM
0488                  TEM=WI(I,K)
0489                  WI(I,K)=WI(I,L)
0490                  WI(I,L)=TEM
0491                  TEM=VR(I,K)
0492                  VR(I,K)=VR(I,L)
0493                  VR(I,L)=TEM
0494                  TEM=VI(I,K)
0495                  VI(I,K)=VI(I,L)
0496              640 VI(I,L)=TEM
0497              650 CONTINUE
0498              660 CONTINUE
0499                  RETURN
0500                  END

END OF SEGMENT, LENGTH  1766, NAME  GLJO


0501                  SUBROUTINE BIORTHO (YR,YI,XR,XI,DIMN,DIMP,N,P,F,MC)
0502                  INTEGER DIMN,DIMP,P,F
0503                  REAL YR(DIMN,DIMP),YI(DIMN,DIMP), XR(DIMN,DIMP),XI(DIMN,DIMP)
0504                  REAL LT,RT,MC
0505                  DOUBLE PRECISION LSUM,RSUM
0506                  DO 110 K=F+1,P
0507               10 LSUM=0.0
0508                  RSUM=0.0
0509                  IF (K .EQ. 1) GO TO 60
0510                  DO 30 I=1,K-1
0511                  CALL CIMP (N,I,K,XR,XI,YR,YI,DIMN,DIMP,SR,SI)
0512                  CALL CABS (SR,SI,S)
0513                  LSUM=LSUM+S*S
0514                  DO 20 J=1,N
0515                  TEMP   =YR(J,K)-SR*YR(J,I)+SI*YI(J,I)
0516                  YI(J,K)=YI(J,K)-SI*YR(J,I)-SR*YI(J,I)
0517               20 YR(J,K)=TEMP
0518               30 CONTINUE
0519                  DO 50 I=1,K-1
0520                  CALL CIMP (N,I,K,YR,YI,XR,XI,DIMN,DIMP,SR,SI)
0521                  CALL CABS (SR,SI,S)
0522                  RSUM=RSUM+S*S
0523                  DO 40 J=1,N
0524                  TEMP   =XR(J,K)-SR*XR(J,I)+SI*XI(J,I)
0525                  XI(J,K)=XI(J,K)-SI*XR(J,I)-SR*XI(J,I)
0526               40 XR(J,K)=TEMP
0527               50 CONTINUE
0528               60 CALL CIMP (N,K,K,XR,XI,YR,YI,DIMN,DIMP,SR,SI)
0529                  CALL CABS (SR,SI,S)
0530                  LSUM=LSUM+S
0531                  RSUM=RSUM+S
0532                  LT=LSUM
0533                  RT=RSUM
0534                  IF ((S .GT. LT/100.0 .AND. LT*MC .NE. 0.0) .AND.
0535                 1    (S .GT. RT/100.0 .AND. RT*MC .NE. 0.0)) GO TO 70
0536                  WRITE (4,910)
0537              910 FORMAT (' WARNING 1 IN BIORTHO')
0538                  IF (S*MC .NE. 0.0) GO TO 10
0539                  WRITE (4,920)
0540              920 FORMAT (' WARNING 2 IN BIORTHO')
0541                  SR=0.0
0542                  SI=0.0
0543                  GO TO 80
0544               70 CALL CIMP (N,K,K,XR,XI,XR,XI,DIMN,DIMP,SR,SI)
0545                  XNORM=SQRT(SR)
0546                  CALL CIMP (N,K,K,YR,YI,YR,YI,DIMN,DIMP,SR,SI)
0547                  YNORM=SQRT(SR)
0548                  CALL CIMP (N,K,K,XR,XI,YR,YI,DIMN,DIMP,SR,SI)
0549                  SR=SR/(XNORM*YNORM)
0550                  SI=SI/(XNORM*YNORM)
0551                  CALL CSQRT (SR,SI,TR,TI)
0552                  D1R=XNORM*TR
0553                  D1I=XNORM*TI
0554                  D2R=YNORM*TR
0555                  D2I=YNORM*TI
0556                  CALL CDIV (1.0,0.0,D1R,D1I,SR,SI)
0557               80 DO 90 J=1,N
0558                  TEMP   = SR*XR(J,K)+SI*XI(J,K)
0559                  XI(J,K)=-SI*XR(J,K)+SR*XI(J,K)
0560               90 XR(J,K)= TEMP
0561                  CALL CDIV (1.0,0.0,D2R,D2I,SR,SI)
0562                  DO 100 J=1,N
0563                  TEMP   = SR*YR(J,K)-SI*YI(J,K)
0564                  YI(J,K)= SI*YR(J,K)+SR*YI(J,K)
0565              100 YR(J,K)= TEMP
0566              110 CONTINUE
0567                  RETURN
0568                  END

END OF SEGMENT, LENGTH   581, NAME  BIORTHO
```

```
0569                  SUBROUTINE CINP (N,K,L,YR,YI,XR,XI,DIMN,DIMP,SR,SI)
0570                  INTEGER DIMN,DIMP
0571                  REAL YR(DIMN,DIMP), YI(DIMN,DIMP), XR(DIMN,DIMP), XI(DIMN,DIMP)
0572                  DOUBLE PRECISION SUMR,SUMI
0573                  SUMR=0.0
0574                  SUMI=0.0
0575                  DO 10 I=1,N
0576                  SUMR=SUMR+YR(I,K)*XR(I,L)+YI(I,K)*XI(I,L)
0577               10 SUMI=SUMI-YI(I,K)*XR(I,L)+YR(I,K)*XI(I,L)
0578                  SR=SNGL(SUMR)
0579                  SI=SNGL(SUMI)
0580                  RETURN
0581                  END

END OF SEGMENT, LENGTH    169, NAME  CINP


0582                  SUBROUTINE CABS (ZR,ZI,R)
0583                  XR=ABS(ZR)
0584                  XI=ABS(ZI)
0585                  IF (XI-XR) 20,20,10
0586               10 R=XR
0587                  XR=XI
0588                  XI=R
0589               20 IF (XI) 30,40,30
0590               30 R=XI/XR
0591                  R=XR*SQRT(1.0+R*R)
0592                  RETURN
0593               40 R=XR
0594                  RETURN
0595                  END

END OF SEGMENT, LENGTH     66, NAME  CABS


0596                  SUBROUTINE CDIV (XR,XI,WR,WI,ZR,ZI)
0597                  YR=WR
0598                  YI=WI
0599                  IF (ABS(YR)-ABS(YI)) 20,20,10
0600               10 H=YI/YR
0601                  YR=H*YI+YR
0602                  ZR=(XR+H*XI)/YR
0603                  ZI=(XI-H*XR)/YR
0604                  RETURN
0605               20 H=YR/YI
0606                  YI=H*YR+YI
0607                  ZR=(H*XR+XI)/YI
0608                  ZI=(H*XI-XR)/YI
0609                  RETURN
0610                  END

END OF SEGMENT, LENGTH     97, NAME  CDIV


0611                  SUBROUTINE CSQRT (ZR,ZI,YR,YI)
0612                  XR=ZR
0613                  XI=ZI
0614                  CALL CABS (XR,XI,H)
0615                  H=SQRT((ABS(XR)+H)/2.0)
0616                  IF (XI) 10,20,10
0617               10 XI=XI/(2.0*H)
0618               20 IF (XR) 40,30,30
0619               30 XR=H
0620                  GO TO 70
0621               40 IF (XI) 60,50,50
0622               50 XR=XI
0623                  XI=H
0624                  GO TO 70
0625               60 XR=-XI
0626                  XI=-H
0627               70 YR=XR
0628                  YI=XI
0629                  RETURN
0630                  END

END OF SEGMENT, LENGTH     88, NAME  CSQRT


0631                  SUBROUTINE RANDOMISATION (X,DIMN,DIMP,N,L,Z)
0632                  INTEGER DIMN,DIMP
0633                  REAL X(DIMN,DIMP)
0634                  DO 10 I=1,N
0635               10 X(I,L)=2.0*FPMCRV(Z)-1.0
0636                  RETURN
0637                  END

END OF SEGMENT, LENGTH     64, NAME  RANDOMISATION
```

```
C638                  SUBROUTINE INFO (DR,DI,LF,RE,DIMP,P,KS,G,H,EM)
C639                  INTEGER DIMP,TYPE,STREAM,PREV,G,H,EM,P,LINE(63)
C640                  REAL LF(DIMP),RE(DIMP), DR(DIMP),DI(DIMP)
C641                  DATA LINE/63*1H*/, PREV/0/
C642                  COMMON TYPE,NO,FIELD,LENGTH,K
C643                  IF (K) 200,200,100
C644            100   IF (PREV .EQ. NO) GO TO 120
C645                  WRITE (4,110) NO, (LINE(I), I=1,63)
C646            110   FORMAT (1H1,//,29X,'INFORMATIONAL OUTPUT ON LOGICAL STREAM 4 FOR E
C647               1XAMPLE NUMBER',I3,/,29X,63A1)
C648            120   WRITE (4,130) KS,G,H,EM
C649            130   FORMAT (1H ,//,43X,'NUMBER OF  STEPS  PERFORMED =',I6,/,
C650               1 43X,'NUMBER OF EIGENVECTORS ACCEPTED =',I3,/,
C651               2 43X,'NUMBER OF EIGENVALUES  ACCEPTED =',I3,/,
C652               3 43X,'NUMBER OF SOLNS. TO BE COMPUTED =',I3)
C653                  GO TO 300
C654            200   IF (PREV .EQ. NO) GO TO 220
C655                  WRITE (4,210) NO, (LINE(I), I=1,42)
C656            210   FORMAT (1H ,///////,' INFORMATIONAL OUTPUT FOR EXAMPLE NUMBER',I3,/
C657               1,1X,42A1)
C658            220   WRITE (4,230) KS,G,H,EM
C659            230   FORMAT (1H ,//,' STEPS PERFORMED =',I4,/,' EIGENVECTORS ACCEPTED =
C660               1',I3,/,' EIGENVALUES  ACCEPTED =',I3,/,' SOLNS. TO BE COMPUTED =',
C661               2I3)
C662            300   WRITE (4,310)
C663            310   FORMAT (' THE ERROR VECTORS ARE:')
C664                  WRITE (4,320)
C665            320   FORMAT (' LEFT-HAND')
C666                  CALL MXOP (LF,1,DIMP,1,P,FIELD,LENGTH,4,K)
C667                  WRITE (4,330)
C668            330   FORMAT (' RIGHT-HAND')
C669                  CALL MXOP (RE,1,DIMP,1,P,FIELD,LENGTH,4,0)
C670                  WRITE (4,340)
C671            340   FORMAT (1H ,/,' THE APPROXIMATIONS TO THE EIGENVALUES ARE:')
C672                  WRITE (4,350)
C673            350   FORMAT (' REAL PART')
C674                  CALL MXOP (DR,1,DIMP,1,P,FIELD,LENGTH,4,0)
C675                  WRITE (4,360)
C676            360   FORMAT (' IMAGINARY PART')
C677                  CALL MXOP (DI,1,DIMP,1,P,FIELD,LENGTH,4,0)
C678                  PREV=NO
C679                  RETURN
C680            999   STOP INFO
C681                  END
```

END OF SEGMENT, LENGTH 222, NAME INFO

```
C682                  SUBROUTINE NORMALISATION (X,Y,DIMN,DIMP,N,P)
C683                  INTEGER DIMN,DIMP,P
C684                  REAL X(DIMN,DIMP), Y(DIMN,DIMP)
C685                  DO 30 J=1,P
C686                  SR=X(1,J)
C687                  SI=Y(1,J)
C688                  AUX=SR*SR+SI*SI
C689                  DO 10 I=2,N
C690                  TR=X(I,J)
C691                  TI=Y(I,J)
C692                  ZZ=TR*TR+TI*TI
C693                  IF (ZZ .LE. AUX) GO TO 10
C694                  SR=TR
C695                  SI=TI
C696                  AUX=ZZ
C697             10   CONTINUE
C698                  CALL CDIV (1.0,0.0,SR,SI,TR,TI)
C699                  DO 20 I=1,N
C700                  SR     =TR*X(I,J)-TI*Y(I,J)
C701                  Y(I,J)=TI*X(I,J)+TR*Y(I,J)
C702             20   X(I,J)=SR
C703             30   CONTINUE
C704                  RETURN
C705                  END
```

END OF SEGMENT, LENGTH 176, NAME NORMALISATION

```
0706                  SUBROUTINE ELAPSE (E,DAT,NO,K)
0707                  INTEGER CALL, DAT(3), MONTH(12), BUFFER(2)
0708                  DATA CALL/2/
0709                  DATA MONTH/'JAN','FEB','MAR','APR','MAY','JUN','JLY','AUG','SEP',
0710                 1 'OCT','NOV','DEC'/
0711                  GO TO (100,200,300), NO
0712            100   CALL MTIME (N)
0713                  E=N
0714                  CALL=3-CALL
0715                  IF (CALL .EQ. 2) GO TO 110
0716                  EE=E
0717                  RETURN
0718            110   E=E-EE
0719                  E=E*1.0E-3
0720                  RETURN
0721            200   CALL DEFBUF (2,8,BUFFER)
0722                  CALL DATE (E)
0723                  WRITE (2,210) E
0724            210   FORMAT (A8)
0725                  READ (2,220) (DAT(I), I=1,3)
0726            220   FORMAT (I2,1X,I2,1X,I2)
0727                  DAT(2)=MONTH(DAT(2))
0728                  E,EE=1.0E-08
0729            230   IF (1.0+EE .EQ. 1.0) GO TO 240
0730                  E=EE
0731                  EE=EE/2.0
0732                  GO TO 230
0733            240   RETURN
0734            300   IF (K) 370,370,310
0735            310   IF (E .GE. 60.0) GO TO 330
0736                  WRITE (6,320) E
0737            320   FORMAT (1H ,////,44X,'EXECUTION TIME WAS',F7.3,' SECONDS')
0738                  GO TO 390
0739            330   IF (E .GE. 120.0) GO TO 350
0740                  E=E-60.0
0741                  WRITE (6,340) E
0742            340   FORMAT (1H ,////,40X,'EXECUTION TIME WAS 1 MINUTE',F7.3,' SECONDS'
0743                 1)
0744                  GO TO 390
0745            350   I=E/60.0
0746                  E=E-I*60.0
0747                  WRITE (6,360) I, E
0748            360   FORMAT (1H ,////,39X,'EXECUTION TIME WAS',I3,' MINUTES',F7.3,' SEC
0749                 1ONDS')
0750                  GO TO 390
0751            370   WRITE (6,380) E
0752            380   FORMAT (1H ,////,' EXECUTION TIME WAS',F9.3,' SECONDS')
0753            390   RETURN
0754                  END

END OF SEGMENT, LENGTH   182, NAME  ELAPSE
```

```
0755                    SUBROUTINE MXOP (A,DIMN,DIMP,N,P,FIELD,LENGTH,STREAM,IFLAG)
0756                    INTEGER DIMN,DIMP,P,WIDTH,STREAM
0757                    REAL A(DIMN,DIMP), FRMT1(6), FRMT2(6), BUFFER(2)
0758                    DATA FRMT1/3H(       ,1HN,8HX,1P    ,1HN,1HE,8H)       /
0759                    DATA FRMT2/3H(       ,1HN,8HX,1P    ,1HN,1HE,8H)       /
0760                    IF (IFLAG) 10,200,10
0761                 10 CALL DEFBUF (2,16,BUFFER)
0762                    FRMT1(5), FRMT2(5)=FIELD
0763                    WRITE (2,20) FIELD
0764                 20 FORMAT (A6)
0765                    READ (2,30) WIDTH
0766                 30 FORMAT (1X,I2)
0767                    IF (P*WIDTH-LENGTH) 100,100,40
0768                 40 N1=LENGTH/WIDTH
0769                    N2=P/N1
0770                    LAST=P-N1*N2
0771                    IF (IFLAG) 50,999,60
0772                 50 N3=1
0773                    N4=1
0774                    GO TO 70
0775                 60 N3=(LENGTH-N1*WIDTH)/2+1
0776                    N4=(LENGTH-LAST*WIDTH)/2+1
0777                 70 WRITE (2,80) N3,N1,N4,LAST
0778                 80 FORMAT (4I4)
0779                    READ (2,90) FRMT1(2), FRMT1(4), FRMT2(2), FRMT2(4)
0780                 90 FORMAT (4A4)
0781                    GO TO 300
0782                100 N2=0
0783                    IF (IFLAG) 110,999,120
0784                110 N3=1
0785                    GO TO 130
0786                120 N3=(LENGTH-P*WIDTH)/2+1
0787                130 WRITE (2,140) N3,P
0788                140 FORMAT (2I4)
0789                    READ (2,150) FRMT1(2), FRMT1(4)
0790                150 FORMAT (2A4)
0791                    GO TO 210
0792                200 IF (N2) 999,210,300
0793                210 WRITE (STREAM,FRMT1) ((A(I,J), J=1,P), I=1,N)
0794                    RETURN
0795                300 N2N1=N2*N1
0796                    IF (N-1) 999,400,310
0797                310 DO 330 K=1,N2N1,N1
0798                    KK=K+N1-1
0799                    WRITE (STREAM,320) K, KK
0800                320 FORMAT (1H ,/,' COLUMNS',I3,' TO',I3,' ARE:')
0801                330 WRITE (STREAM,FRMT1) ((A(I,J), J=K,KK), I=1,N)
0802                    IF (LAST-1) 370,340,360
0803                340 WRITE (STREAM,350) P
0804                350 FORMAT (1H ,/,' COLUMN',I3,' IS:')
0805                    WRITE (STREAM,FRMT2) (A(I,P), I=1,N)
0806                    GO TO 370
0807                360 KK=N2N1+1
0808                    WRITE (STREAM,320) KK,P
0809                    WRITE (STREAM,FRMT2) ((A(I,J), J=KK,P), I=1,N)
0810                370 RETURN
0811                400 DO 410 K=1,N2N1,N1
0812                    KK=K+N1-1
0813                410 WRITE (STREAM,FRMT1) (A(1,J), J=K,KK)
0814                    IF (LAST-1) 440,420,430
0815                420 WRITE (STREAM,FRMT2) A(1,P)
0816                    GO TO 440
0817                430 KK=N2N1+1
0818                    WRITE (STREAM,FRMT2) (A(1,J), J=KK,P)
0819                440 RETURN
0820                999 STOP MXOP
0821                    END

END OF SEGMENT, LENGTH   427, NAME  MXOP
```

```
C822                SUBROUTINE PRODUCT (VR,VI,WR,WI,DIMN,N,HERMIT)
C823                INTEGER DIMN,PREV,HERMIT,TYPE
C824                REAL VR(DIMN),VI(DIMN), WR(DIMN),WI(DIMN)
C825                REAL AR(10,10),AI(10,10)
C826                DOUBLE PRECISION SUMR,SUMI
C827                DATA PREV/0/
C828                COMMON TYPE,NO
C829                GO TO (100,200,300,400,500,600,700,800,900), TYPE
C830          100   CONTINUE
C831                IF (PREV .EQ. NO) GO TO 120
C832                READ (5,110) ((AR(I,J),AI(I,J), J=1,N), I=1,N)
C833          110   FORMAT (2G0.0)
C834                PREV=NO
C835          120   IF (HERMIT) 121,121,150
C836          121   DO 140 I=1,N
C837                SUMR,SUMI=0.0
C838                DO 130 J=1,N
C839                SUMR=SUMR+AR(I,J)*VR(J)-AI(I,J)*VI(J)
C840          130   SUMI=SUMI+AR(I,J)*VI(J)+AI(I,J)*VR(J)
C841                WR(I)=SUMR
C842          140   WI(I)=SUMI
C843                RETURN
C844          150   DO 170 I=1,N
C845                SUMR,SUMI=0.0
C846                DO 160 J=1,N
C847                SUMR=SUMR+AR(J,I)*VR(J)+AI(J,I)*VI(J)
C848          160   SUMI=SUMI+AR(J,I)*VI(J)-AI(J,I)*VR(J)
C849                WR(I)=SUMR
C850          170   WI(I)=SUMI
C851                RETURN
C852          200   CONTINUE
C853                IF (HERMIT) 210,210,230
C854          210   K=N
C855                DO 220 I=1,N-1
C856                WR(I)=K*VR(I)+N*VR(I+1)
C857                WI(I)=K*VI(I)+N*VI(I+1)
C858          220   K=K-1
C859                WR(N)=1.0E-10*VR(1)+VR(N)
C860                WI(N)=1.0E-10*VI(1)+VI(N)
C861                RETURN
C862          230   WR(1)=N*VR(1)+1.0E-10*VR(N)
C863                WI(1)=N*VI(1)+1.0E-10*VI(N)
C864                K=N-1
C865                DO 240 I=2,N
C866                WR(I)=N*VR(I-1)+K*VR(I)
C867                WI(I)=N*VI(I-1)+K*VI(I)
C868          240   K=K-1
C869                RETURN
C870          300   CONTINUE
C871                IF (HERMIT) 310,310,330
C872          310   WR(1)=VR(2)
C873                WI(1)=VI(2)
C874                DO 320 K=2,N-1
C875                NK=N-K+1
C876                WR(K)=NK*VR(K-1)+K*VR(K+1)
C877          320   WI(K)=NK*VI(K-1)+K*VI(K+1)
C878                WR(N)=VR(N-1)
C879                WI(N)=VI(N-1)
C880                RETURN
C881          330   WR(1)=(N-1)*VR(2)
C882                WI(1)=(N-1)*VI(2)
C883                DO 340 K=2,N-1
C884                NK=N-K
C885                K1=K-1
C886                WR(K)=K1*VR(K1)+NK*VR(K+1)
C887          340   WI(K)=K1*VI(K1)+NK*VI(K+1)
C888                WR(N)=(N-1)*VR(N-1)
C889                WI(N)=(N-1)*VI(N-1)
C890                RETURN
C891          400   CONTINUE
C892          500   CONTINUE
C893          600   CONTINUE
C894          700   CONTINUE
C895          800   CONTINUE
C896          900   CONTINUE
C897                END

END OF SEGMENT, LENGTH   709, NAME  PRODUCT
```

THE METHOD OF "QUICK RITZ" ITERATION FOR AN ARBITRARY MATRIX

THIS PROGRAM WAS RUN ON 22 AUG 74
AND EXECUTION STARTED AT 18/38/46

EXAMPLE NUMBER 6

4 EIGENVALUES AND 2 EIGENVECTORS HAVE BEEN ACCEPTED TO AN ACCURACY OF 1.00E-05

THE 4 APPROXIMATIONS TO THE EIGENVALUES ARE;

REAL PART

3.0000000001E 00 3.0000000000E 00 1.9999999985E 00 2.0000000007E 00

IMAGINARY PART

5.7206765570E-11 -1.4122110052E-11 1.0000000002E 00 -9.9999999987E-01

THE MATRIX OF CORRESPONDING LEFT-HAND EIGENVECTORS IS GIVEN BELOW;

REAL PART

-3.6133949243E-10	-2.4957572904E-10	-6.0000000515E-01	-6.0000003291E-01
7.2799504243E-10	4.1078209428E-10	9.9999999999E-01	9.9999999999E-01
-3.7028742735E-10	-1.9618554294E-10	-3.9999999406E-01	-3.9999996631E-01
-3.6493118985E-01	-4.0700210132E-01	-1.4444987223E-09	-1.4929067485E-09
9.9999999997E-01	1.1050315209E-01	6.7962270061E-10	7.2481247854E-10
-3.1114643044E-01	1.0000000000E 00	-2.2074697869E-11	-6.8140789237E-12

IMAGINARY PART

-4.4026459351E-11	-6.4903249247E-11	2.0000008061E-01	-2.0000007581E-01
6.3183257150E-11	1.1029384971E-10	0.0000000000E-01	0.0000000000E-01
-2.6194437082E-11	-2.3619245913E-11	-8.0000038072E-01	2.0000007419E-01
6.6112315924E-06	4.0422630203E-05	8.8332933099E-10	-8.4887722348E-10
0.0000000000E-01	-6.0633918338E-05	-6.4829685088E-10	5.5024657840E-10
-1.9333694731E-05	0.0000000000E-01	4.3848989380E-11	-4.0683799566E-11

THE MATRIX OF CORRESPONDING RIGHT-HAND EIGENVECTORS IS GIVEN BELOW;

REAL PART

1.0000000000E 00	7.1192604471E-01	1.0000000000E 00	1.0000000000E 00
9.9999999998E-01	7.1192604471E-01	9.0163936979E-01	9.0163936081E-01
9.9999999972E-01	7.1192802267E-01	2.2131128599E-01	2.2131148884E-01
9.9999999956E-01	7.1192604465E-01	5.4098362200E-01	5.4098361661E-01
9.9999999935E-01	7.1192604480E-01	3.8065574798E-01	3.8065574437E-01
2.9649894916E-01	9.9999999999E-01	1.8032787399E-01	1.8032787218E-01

IMAGINARY PART

3.6379788071E-12	3.4611568481E-05	3.6379788071E-12	0.0000000000E-01
-1.8189894035E-11	3.4611570300E-05	8.1967233986E-02	-8.1967241509E-02
-1.4551915228E-11	3.4611592127E-05	6.5573786946E-02	-6.5573792894E-02
-2.5465851650E-11	3.4611615774E-05	4.9180340189E-02	-4.9180346617E-02
-2.5465851650E-11	3.4611623050E-05	3.2786893409E-02	-3.2786896378E-02
2.0211290575E-05	0.0000000000E-01	1.6393446703E-02	-1.6393448193E-02

THE 4 SETS OF CORRESPONDING ERRORS ARE;

LEFT-HAND	-2.9033703039E-09	-7.8564464101E-11	-1.4823517329E-07	-1.2648271689E-07
RIGHT-HAND	-2.1184402404E-09	8.5055849555E-11	-8.9006766408E-08	-7.6327127298E-08

EXECUTION TIME WAS 14.841 SECONDS

INFORMATIONAL OUTPUT ON LOGICAL STREAM 4 FOR EXAMPLE NUMBER 6

NUMBER OF STEPS PERFORMED = 2
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTORS ARE:
LEFT-HAND
-1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00
RIGHT-HAND
-1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00 -1.0000000000E 00

THE APPROXIMATIONS TO THE EIGENVALUES ARE:
REAL PART
5.1030217221E 00 2.8471131065E 00 1.5258417791E 00 -6.7895938233E-01
IMAGINARY PART
4.0522212390E-12 -2.5216142542E-11 6.1744475494E-12 -1.7024840258E-12

NUMBER OF STEPS PERFORMED = 4
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTORS ARE:
LEFT-HAND
8.3649328567E-01 2.2472572620E-01 -4.6677818321E-01 -1.8951937028E-01
RIGHT-HAND
8.5259168130E-01 1.2586896225E 00 -4.3980479932E-01 -1.1551952936E-01

THE APPROXIMATIONS TO THE EIGENVALUES ARE:
REAL PART
3.4169276508E 00 2.9996085617E 00 1.9768174780E 00 1.9768174781E 00
IMAGINARY PART
5.6190971262E-11 -1.0972574555E-10 9.4118127468E-01 -9.4118127452E-01

NUMBER OF STEPS PERFORMED = 7
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTORS ARE:
LEFT-HAND
-1.7649621273E-01 1.0907090413E-03 3.0429616867E-02 3.5962321231E-02
RIGHT-HAND
-7.3167630419E-02 4.0245122059E-03 3.2124953410E-02 3.6775143484E-02

THE APPROXIMATIONS TO THE EIGENVALUES ARE:
REAL PART
3.0096010267E 00 2.9999977454E 00 1.9930324185E 00 1.9930324185E 00
IMAGINARY PART
1.4407022584E-11 1.6884420365E-11 1.0058487427E 00 -1.0058487420E 00

NUMBER OF STEPS PERFORMED = 11
NUMBER OF EIGENVECTORS ACCEPTED = 0
NUMBER OF EIGENVALUES ACCEPTED = 0
NUMBER OF SOLNS. TO BE COMPUTED = 2

THE ERROR VECTORS ARE:
LEFT-HAND
-1.0044029550E-02 -2.0153279104E-02 -1.2024044930E-02 -9.8696055766E-03
RIGHT-HAND
-8.6076700690E-03 7.3454814303E-03 -6.6656762707E-03 -5.3365424933E-03

THE APPROXIMATIONS TO THE EIGENVALUES ARE:
REAL PART
3.0000238410E 00 2.9999999917E 00 2.0001392839E 00 2.0001392838E 00
IMAGINARY PART
2.4481000560E-10 -2.3309643971E-11 9.9994382581E-01 -9.9994382656E-01

```
                                NUMBER OF  STEPS  PERFORMED =    16
                                NUMBER OF EIGENVECTORS ACCEPTED =  0
                                NUMBER OF EIGENVALUES  ACCEPTED =  0
                                NUMBER OF SOLNS, TO BE COMPUTED =  2
THE ERROR VECTORS ARE;
LEFT-HAND
                      -1.0845253720E-04 -3.1701056510E-06  2.6123719807E-04  2.7170912317E-04
RIGHT-HAND
                      -1.2492520826E-04  2.2853485750E-06  5.4377088164E-05  6.0673823491E-05
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
REAL PART
                       3.0000000055E 00  2.9999999999E 00  1.9999997704E 00  1.9999997696E 00
IMAGINARY PART
                      -6.0364092299E-12 -1.1365506485E-11  1.0000003340E 00 -1.0000003345E 00

                                NUMBER OF  STEPS  PERFORMED =    22
                                NUMBER OF EIGENVECTORS ACCEPTED =  0
                                NUMBER OF EIGENVALUES  ACCEPTED =  2
                                NUMBER OF SOLNS, TO BE COMPUTED =  2
THE ERROR VECTORS ARE;
LEFT-HAND
                      -6.2117235606E-07 -1.9739321607E-08  6.3885465529E-06  6.3196252129E-06
RIGHT-HAND
                      -7.4157177683E-07  1.5463153448E-08  4.5858375081E-06  4.5454076873E-06
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
REAL PART
                       2.9999999999E 00  2.9999999999E 00  2.0000000004E 00  2.0000000001E 00
IMAGINARY PART
                      -4.9683377998E-11 -2.1073990094E-12  9.9999999968E-01 -9.9999999993E-01

                                NUMBER OF  STEPS  PERFORMED =    29
                                NUMBER OF EIGENVECTORS ACCEPTED =  2
                                NUMBER OF EIGENVALUES  ACCEPTED =  4
                                NUMBER OF SOLNS, TO BE COMPUTED =  2
THE ERROR VECTORS ARE;
LEFT-HAND
                      -2.9033703039E-09 -7.8564464101E-11 -1.4823517529E-07 -1.2648271689E-07
RIGHT-HAND
                      -2.1184402404E-09  8.5055849555E-11 -8.9006766408E-08 -7.6327127298E-08
THE APPROXIMATIONS TO THE EIGENVALUES ARE;
REAL PART
                       3.0000000001E 00  3.0000000000E 00  1.9999999985E 00  2.0000000007E 00
IMAGINARY PART
                       5.7206765570E-11 -1.4122110052E-11  1.0000000002E 00 -9.9999999987E-01
```