Printed at the Mathematical Centre, Kruislaan 413, Amsterdam, The Netherlands.

The Mathematical Centre, founded 11th February 1946, is a non-profit institution for the promotion of pure and applied mathematics and computer science. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

MATHEMATICAL CENTRE TRACTS

23

ALGOL 60 PROCEDURES IN NUMERICAL ALGEBRA

PART 2

BY

T.J. DEKKER
W. HOFFMANN

2nd. edition

MATHEMATISCH CENTRUM AMSTERDAM

1971

1st edition: 1968

PREFACE

We here present a set of ALGOL 60 procedures for calculating eigenvalues and/or eigenvectors of real matrices. This set uses some vector procedures published in part 1, which, moreover, contains a set of procedures for solving systems of linear equations, for inverting matrices and for solving linear least-squares problems [3].

The procedures have been tested on an Electrologica X8 computer by means of the " MC ALGOL 60 system for the X8 " of the Mathematical Centre, Amsterdam, written by F.E.J. Kruseman Aretz.
The texts of the procedures have been edited by the program "ALGOL editor", written and published by H.L. Oudshoorn, H.N. Glorie and G.C.J.M. Nogarede (MR 98, Mathematical Centre, Amsterdam, 1968).

In the second edition some minor changes have been introduced.
The texts of the following four procedures have been changed for the following reasons:
zeroin (mca 2310), for better handling a function which vanishes on a part of the given interval;
eqilbr (mca 2405), for reducing the possibility of overflow;
baklbr (mca 2406), for avoiding superfluous row interchanges;
comvalqri (mca 2420), for avoiding division (by a possibly vanishing quantity) in the second (weak) partitioning test.
The other changes are minor corrections outside the procedure texts.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to Prof. Dr. Ir. A. van Wijngaarden and Prof. Dr. F.E.J. Kruseman Aretz for invaluable suggestions and criticisms, and similarly to Dr. B.J. Mailloux, who, moreover, carefully read the manuscript and suggested several improvements.
The authors wish to thank also F.J. van den Bosch, R. van der Horst, D.P. de Jong, J. van der Velden and Miss C.M.L. Smit for their help in writing the ALGOL texts and in testing the procedures,
to Mrs. E. Binnenmarsch and Miss T. Collast for typing the manuscript, and to D. Zwarst and J. Suiker for the printing and binding.

CONTENTS

NOTATIONS

References are given between the square brackets "[" and "]".
" : " denotes the integer division symbol " ÷ " [1, 3.3.4.2.].
"goto" denotes the same symbol as "go to", "boolean" the same as "Boolean" [1, 2.3.].
"I" denotes the identity matrix; the order will be clear from the context.
"'" denotes transposition of a matrix.
Unless stated otherwise, "M" denotes a real matrix, "H" a real upper-Hessenberg matrix, "T" a tridiagonal real symmetric matrix, and "n" the order of the matrix considered.
"||x||" ("||M||") denotes some norm of the vector x (matrix M).

DEFINITIONS

The "dimension" of an array is the number of its subscripts (see also [1, 5.2.3.2.]). Thus, we speak about "one-dimensional" and "two-dimensional" arrays.
For two-dimensional arrays as well as for matrices, we use the following terminology:
the first subscript is called the "row index" and the second the "column index"; the i-th "row" ("column") is the subset for which the row (column)index equals i;
the i-th "subdiagonal" ("superdiagonal") is the subset for which the column index minus the row index equals i (equals -i); the 0-th subdiagonal (or superdiagonal) is called the "main diagonal", whereas the other subdiagonals and superdiagonals are called "codiagonals";
the first codiagonals are said to be "adjacent" to the main diagonal;
the "upper triangle" ("lower triangle", "strict-lower triangle") is the subset for which the column index is $\leq$ (is $\geq$, is >) the row index.

A "lower-triangular" ("upper-triangular") matrix, is a matrix whose elements outside its lower (upper) triangle are zero;
a "unit lower-triangular" matrix is a lower-triangular matrix whose main diagonal elements are equal to 1.
An "upper-Hessenberg" matrix is a matrix whose elements outside its upper triangle and first subdiagonal are zero; a tridiagonal matrix is a matrix whose elements outside its main diagonal and adjacent codiagonals are zero.
The first subdiagonal of an upper-Hessenberg matrix (symmetric tridiagonal matrix) is simply called its "subdiagonal" ("codiagonal").

We use the following vector norms [2, p. 55]: the "one-norm", i.e. the sum of the absolute values of the elements of the vector; the "Euclidean norm", i.e. the square root of the sum of their squares; and the "infinity norm", i.e. their maximum absolute value.
The "infinity norm" of a matrix is defined as the maximum one-norm of its rows.

A set of vectors is said to be "numerically independent" if it is not approximately equal to a set of linearly dependent vectors.

The "machine precision" is the largest number, p, for which 1 + p = 1 on the computer (about $_{10}$-12 for the X8); the "working precision" roughly equals the machine precision.
A "relative tolerance" is a tolerance relative either to a calculated eigenvalue or to some matrix norm.
Relative tolerances must be chosen smaller than 1, and should be chosen not smaller than the machine precision.

INTRODUCTION

Chapter 23 contains a set of procedures for calculating the eigenvalues and/or eigenvectors of real symmetric matrices.
Chapter 24 contains a set of procedures for calculating the real or complex eigenvalues and/or eigenvectors of real matrices.
Of symmetric matrices, only the upper triangles need be given; for these upper triangles, the procedures of chapter 23 use either the upper triangle of a two-dimensional array (in which case its remaining part is not used) or a one-dimensional array(in which case no space is wasted).

In each chapter, we give a survey of its contents and some numerical considerations and comparisons. The chapters are subdivided into sections in each of which we give a more detailed survey of its contents and explain the numerical methods used.
Each section contains some procedures for each of which we give, besides the ALGOL text, a description mentioning the data required, the results delivered and the nonlocal procedures (except the standard functions [1, 3.2.4.]) directly or indirectly used.

The data are given, and the results are delivered, via the parameters of the procedures, moreover, some procedures deliver a result as the value assigned to their identifier.
Some parameters are used as well for data as for results.
For each formal parameter specified array or integer array, we give the "minimal declaration", i.e. a declaration with the appropriate number of bound pairs, where each pair indicates the range which is actually used by the procedure. Sometimes not all elements of the array indicated by the minimal declaration are used. In the descriptions, we always mention which part is used for the data and which part for the results, so that it is always clear which elements are not used at all and which elements are left unchanged.

As to the nonlocal procedures used, these are either procedures published in this volume (and preceding the procedure which uses them), or procedures published in part 1 [3, chapter 20], or standard functions [1, 3.2.4.] (which are not mentioned in the descriptions), or the X8-code procedures TODRUM and FROMDRUM (which are used only by reaeig2 (mca 2415) and comeig2 (mca 2425)).
TODRUM and FROMDRUM perform a transport to or from backing storage. Let a be a real array with k elements, and p an expression whose value is a nonnegative integer; then the call "TODRUM(a, p)" transports array a to the backing storage from locations p through p + 2 × k − 1 (the X8 uses two storage locations per real number), and the call "FROMDRUM(a, p)" transports the contents of the backing storage from locations p through p + 2 × k − 1 to array a.
It is assumed that two-dimensional arrays are stored columnwise, i.e. the elements of a column are stored consecutively.

CHAPTER 23

EIGENSYSTEMS OF REAL SYMMETRIC MATRICES

This chapter contains procedures for calculating eigenvalues and/or eigenvectors of real symmetric matrices, and a procedure (mca 2310) for finding a zero of a function by means of linear interpolation.
For solving the symmetric eigenproblem, two methods have been chosen both starting with Householder's transformation (section 230).
In the first method (section 231), the eigenvalues are calculated by means of iterated linear interpolation, and the eigenvectors by means of inverse iteration; both processes necessarily converge.
The eigenvalues are obtained in monotonically nonincreasing order.
This method is particularly suitable, if only some consecutive eigenvalues and/or the corresponding eigenvectors are required. The second method (section 232) uses QR iteration for calculating all eigenvalues and eigenvectors. Using a suitable "shift" (see p. 32), this process is convergent [29]. Either method yields eigenvalues and eigenvectors in reasonable precision, and the eigenvectors obtained are orthogonal within working precision.
Note, however, that calculated eigenvectors corresponding to closely clustered eigenvalues may deviate substantially from the true eigenvectors [2, chapter 2, in particular formula 10.2].
The computation time for either method is roughly proportional to n cubed, but obviously depends on the number of iterations required (see Appendix).
These methods are mutually competitive as to accuracy and computation time; both are considerably faster than and as reliable (with respect to accuracy obtained) as Jacobi's method [2, p. 266-282 and 343] [23].

Several procedures of this chapter exist in two versions, distinguished by a "2" or "1" at the end of the procedure identifier.
The procedures whose identifiers end with "2" use the upper triangle of a two-dimensional array, declared by array a[1:n, 1:n], for the upper triangle of the given symmetric matrix M (or for the data for Householder's back transformation) [7].
The (i, j)-th element of M is a[i, j] only for $1 \leq i \leq j \leq n$.
The other elements of a are neither used nor changed.
The procedures whose identifiers end with "1" use a one-dimensional array, declared by array a[1:(n + 1) × n : 2], for the upper triangle of M (or for the data for Householder's back transformation).
The (i, j)-th element of M is a[(j - 1) × j : 2 + i] for $1 \leq i \leq j \leq n$.
Thus, the space required for the matrix is cut nearly in half [10], and the time is not greatly different.
The procedures "eigsym2" and "eigsym1" (section 231) use a separate two-dimensional array for the eigenvectors of M; on the other hand, the procedure qrisym (mca 2323) uses the same two-dimensional array for the upper triangle of M as well as for the matrix of eigenvectors; thus, qrisym is the most economic one with respect to storage space.

```
comment mca 2300;
procedure tfmsymtri2(a, n, d, b, bb, em); value n; integer n;
array a, b, bb, d, em;
begin integer i, j, r, r1;
    real w, x, a1, b0, bb0, d0, machtol, norm;
    norm:= 0;
    for j:= 1 step 1 until n do
    begin w:= 0;
        for i:= 1 step 1 until j do w:= abs(a[i,j]) + w;
        for i:= j + 1 step 1 until n do w:= abs(a[j,i]) + w;
        if w > norm then norm:= w
    end;
    machtol:= em[0] × norm; em[1]:= norm; r:= n;
    for r1:= n - 1 step - 1 until 1 do
    begin d[r]:= a[r,r]; x:= tammat(1, r - 2, r, r, a, a);
        a1:= a[r1,r]; if sqrt(x) < machtol then
        begin b0:= b[r1]:= a1; bb[r1]:= b0 × b0; a[r,r]:= 1 end
        else
        begin bb0:= bb[r1]:= a1 × a1 + x;
            b0:= if a1 > 0 then - sqrt(bb0) else sqrt(bb0);
            a1:= a[r1,r]:= a1 - b0; w:= a[r,r]:= 1 / (a1 × b0);
            for j:= 1 step 1 until r1 do b[j]:= (tammat(1, j, j, r,
            a, a) + matmat(j + 1, r1, j, r, a, a)) × w;
            elmveccol(1, r1, r, b, a, tamvec(1, r1, r, a, b) × w ×
            .5);
            for j:= 1 step 1 until r1 do
            begin elmcol(1, j, j, r, a, a, b[j]);
                elmcolvec(1, j, j, a, b, a[j,r])
            end;
            b[r1]:= b0
        end;
        r:= r1
    end;
    d[1]:= a[1,1]; a[1,1]:= 1; b[n]:= bb[n]:= 0
end tfmsymtri2;
```

Section 230 Householder's transformation

This section contains procedures for transforming a real symmetric matrix into a similar tridiagonal one:
tfmsymtri2 and tfmsymtri1 perform Householder's transformation;
baksymtri2 and baksymtri1 perform the corresponding back transformation;
tfmprevec is to be used in combination with tfmsymtri2 for calculating the transforming matrix.

Householder's transformation is an orthogonal similarity transformation which transforms a symmetric matrix into a tridiagonal one [2, p. 290 - 299] [4, AP 210 and AP 231] [7] [10].
Let M be a given symmetric matrix of order n, S the transforming matrix and T the resulting tridiagonal matrix. Since S is orthogonal (i.e. its inverse equals S'), we then have T = S'MS. Matrix S is a product of n - 1 Householder matrices, these being orthogonal symmetric matrices of the form I + suu', where s is a scalar, and u a column vector. The p-th Householder matrix, p = 1,..., n - 1, is chosen in such a way that the last p elements of u vanish, and the desired zeroes are introduced in the (n - p + 1)-th column and row of the matrix. However, if, in this column and row, all elements outside the main diagonal and the adjacent codiagonals are smaller in absolute value than the infinity norm of M times the machine precision, then the p-th transformation is skipped (i.e. the p-th Householder matrix is replaced by I).

The data for the back transformation, viz. the vectors u and scalars s of the Householder matrices, are overwritten on the upper triangle of M, the scalars s being delivered on the main diagonal.
For each Householder matrix ≠ I, the scalar s is negative; if the transformation is skipped, then +1 is delivered instead of s.

The back transformation transforms a vector x into the vector Sx; if x is an eigenvector of T, then Sx is the corresponding eigenvector of M. Starting from the vector v = x, the vector Sx is obtained by successively replacing v by the p-th Householder matrix times v, for p = n - 1, ..., 2, 1; the resulting vector v then equals Sx.
Similarly, tfmprevec calculates the transforming matrix S starting from I.

Description mca 2300
tfmsymtri2 transforms the n-th order symmetric matrix M whose upper triangle is given in array a[1:n, 1:n], into a similar symmetric tridiagonal matrix T.
In array em[0:1], one must give the machine precision, em[0].
tfmsymtri2 delivers the main diagonal, the codiagonal and the squares of the codiagonal elements of T in array d, b, bb[1:n], the remaining elements b[n] and bb[n] obtaining the value 0.
Moreover, the data for the back transformation are delivered in the upper triangle of a, and em[1]:= the infinity norm of M.
tfmsymtri2 uses tamvec, matmat, tammat, elmveccol, elmcolvec and elmcol [3, chapter 20].

```
comment mca 2301;
procedure baksymtri2(a, n, n1, n2, vec); value n, n1, n2;
integer n, n1, n2; array a, vec;
begin integer i, j, k;
    real w;
    for j:= 2 step 1 until n do
    begin w:= a[j,j]; if w < 0 then
        for k:= n1 step 1 until n2 do elmcol(1, j - 1, k, j, vec, a,
        tammat(1, j - 1, j, k, a, vec) × w)
    end
end baksymtri2;


comment mca 2302;
procedure tfmprevec(a, n); value n; integer n; array a;
begin integer i, j, j1, k;
    real ab;
    j1:= 1;
    for j:= 2 step 1 until n do
    begin for i:= 1 step 1 until j1 - 1, j step 1 until n do
        a[i,j1]:= 0; a[j1,j1]:= 1; ab:= a[j,j]; if ab < 0 then
        for k:= 1 step 1 until j1 do elmcol(1, j1, k, j, a, a,
        tammat(1, j1, j, k, a, a) × ab); j1:= j
    end;
    for i:= n - 1 step - 1 until 1 do a[i,n]:= 0; a[n,n]:= 1
end tfmprevec;
```

Description mca 2301
baksymtri2 should be called after tfmsymtri2, and performs the corresponding back transformation on the columns of
array vec[1:n, n1:n2].
The data for the back transformation, as produced by tfmsymtri2, must be given in the upper triangle of array a[1:n, 1:n].
The resulting vectors of the back transformation are overwritten on the corresponding columns of vec.
baksymtri2 uses tammat and elmcol [3, chapter 20].

Description mca 2302
tfmprevec should be called after tfmsymtri2 and calculates the corresponding transforming matrix.
The data for the back transformation, as produced by tfmsymtri2, must be given in the upper triangle of array a[1:n, 1:n].
The transforming matrix is delivered in the whole of array a[1:n, 1:n].
tfmprevec uses tammat and elmcol [3, chapter 20].

```
comment mca 2305;
procedure tfmsymtri1(a, n, d, b, bb, em); value n; integer n;
array a, b, bb, d, em;
begin integer i, j, r, r1, p, q, ti, tj;
    real s, w, x, a1, b0, bb0, d0, norm, machtol;
    norm:= 0; tj:= 0;
    for j:= 1 step 1 until n do
    begin w:= 0;
        for i:= 1 step 1 until j do w:= abs(a[i + tj]) + w;
        tj:= tj + j; ti:= tj + j;
        for i:= j + 1 step 1 until n do
        begin w:= abs(a[ti]) + w; ti:= ti + i end;
        if w > norm then norm:= w
    end;
    machtol:= em[0] × norm; em[1]:= norm; q:= (n + 1) × n : 2; r:= n;
    for r1:= n - 1 step - 1 until 1 do
    begin p:= q - r; d[r]:= a[q]; x:= vecvec(p + 1, q - 2, 0, a, a);
        a1:= a[q - 1]; if sqrt(x) ≤ machtol then
        begin b0:= b[r1]:= a1; bb[r1]:= b0 × b0; a[q]:= 1 end
        else
        begin bb0:= bb[r1]:= a1 × a1 + x;
            b0:= if a1 > 0 then - sqrt(bb0) else sqrt(bb0);
            a1:= a[q - 1]:= a1 - b0; w:= a[q]:= 1 / (a1 × b0);
            tj:= 0;
            for j:= 1 step 1 until r1 do
            begin ti:= tj + j; s:= vecvec(tj + 1, ti, p - tj, a, a);
                tj:= ti + j;
                b[j]:= (seqvec(j + 1, r1, tj, p, a, a) + s) × w;
                tj:= ti
            end;
            elmvec(1, r1, p, b, a, vecvec(1, r1, p, b, a) × w × .5);
            tj:= 0;
            for j:= 1 step 1 until r1 do
            begin ti:= tj + j; elmvec(tj + 1, ti, p - tj, a, a, b[j]);
                elmvec(tj + 1, ti, - tj, a, b, a[j + p]); tj:= ti
            end;
            b[r1]:= b0
        end;
        q:= p; r:= r1
    end;
    d[1]:= a[1]; a[1]:= 1; b[n]:= bb[n]:= 0
end tfmsymtri1;
```

Description mca 2305

tfmsymtri1 transforms the n–th order symmetric matrix M whose upper triangle is given in array a[1:(n + 1) × n : 2] in such a way that the (i, j)–th element of M is a[(j − 1) × j : 2 + i] for $1 \leq i \leq j \leq n$.
In array em[0:1], one must give the machine precision, em[0].
Matrix M is transformed into a similar symmetric tridiagonal matrix T.
The main diagonal, the codiagonal and the squares of the codiagonal elements of T are delivered in array d, b, bb[1:n], the remaining elements b[n] and bb[n] obtaining the value 0.
Moreover, the data for the back transformation are delivered in a, and em[1]:= the infinity norm of M.
tfmsymtri1 uses vecvec, seqvec and elmvec [3, chapter 20].

```
comment mca 2306;
procedure baksymtri1(a, n, n1, n2, vec); value n, n1, n2;
integer n, n1, n2; array a, vec;
begin integer j, j1, k, ti, tj;
    real w;
    array auxvec[1:n];
    for k:= n1 step 1 until n2 do
    begin for j:= 1 step 1 until n do auxvec[j]:= vec[j,k];
        tj:= j1:= 1;
        for j:= 2 step 1 until n do
        begin ti:= tj + j; w:= a[ti];
            if w < 0 then elmvec(1, j1, tj, auxvec, a, vecvec(1,
            j1, tj, auxvec, a) × w); j1:= j; tj:= ti
        end;
        for j:= 1 step 1 until n do vec[j,k]:= auxvec[j]
    end
end baksymtri1;
```

Description mca 2306
baksymtri1 should be called after tfmsymtri1 and performs the corresponding back transformation on the columns of
array vec[1:n, n1:n2].
The data for the back transformation, as produced by tfmsymtri1, must be given in array a[1:(n + 1) × n : 2].
The resulting vectors of the back transformation are overwritten on the corresponding columns of vec.
baksymtri1 uses vecvec and elmvec [3, chapter 20].

Section 231 Linear interpolation and inverse iteration

This section contains procedures for calculating eigenvalues and/or eigenvectors of real symmetric matrices, and a procedure for calculating a zero of a function:
eigvalsym2 and eigvalsym1 calculate all eigenvalues, or some consecutive eigenvalues including the largest, of a symmetric matrix;
eigsym2 and eigsym1 calculate the corresponding eigenvectors as well;
valsymtri calculates all, or some consecutive, eigenvalues of a symmetric tridiagonal matrix;
vecsymtri calculates the corresponding eigenvectors;
zeroin searches for a zero of a function in a given interval.

The method used in zeroin is a mixture of linear interpolation and extrapolation, and bisection [4, AP 200 and AP 230] [12] [13].
If the given function has different sign at the endpoints of the given interval, then this interval is successively reduced to smaller intervals in whose endpoints the function still has different sign.
In each step, three points, a, b, c, are involved, where, apart from possible interchanges, (see below), b is the most recent iterate, a the previous one, and the "contrapoint" c is the last iterate at which the function does not have the same sign as at b.
If the absolute value of the function at b is greater than at c, then b and c are interchanged (in order to simplify the convergence test).
In each step, linear interpolation or extrapolation is performed between a and b, yielding a point i; if i is not between b and the middle, m, of b and c, then i is replaced by m; moreover, if abs(b − i) < tol, where tol is a given tolerance, i is replaced by sign(c − b) × tol + b. This ensures that any two iterates have a difference not smaller than tol, and that the length of the successive intervals is reduced by at least tol in each step, so that convergence is guaranteed, provided the tolerance is not smaller than the machine precision; the process ends as soon as abs(c − b) $\leq$ 2 × tol.
Since the interchanges mentioned above occur relatively seldom, the process has a completely satisfactory asymptotic behaviour: for a simple zero of a function having a continuous second derivative, the order of convergence is (1 + sqrt(5)) / 2, i. e. about 1.6.
If the given function has the same sign at the endpoints of the given interval, then the interval is reduced by means of bisection (or by taking sign(c − b) × tol + b as new iterate, if the function (nearly) vanishes at b). If sign change is detected, then the process continues as above; otherwise, the process ends as soon as abs(c − b) $\leq$ 2 × tol.

The procedure valsymtri calculates eigenvalues of a symmetric tridiagonal matrix T by means of the method of Sturm-Givens [2, p. 299 − 315] [6] [8] [10] [11] with linear interpolation [4, AP 212 and 232].
Let p(i, x), for i = 0, 1,...,n, denote the i-th principle minor of T(x) = T − xI, i.e. the determinant of the submatrix consisting of the first i rows and columns of T(x) (in particular p(0, x) = 1 for all x). If none of the codiagonal elements of T vanishes, then the sequence of these principle minors is a Sturm sequence; i.e. for each x, the number of agreements in sign of consecutive members of this

sequence equals the number of eigenvalues of T which are greater than x, and for each i > 1 the zeroes of p(i, x) are separated by the zeroes of p(i − 1, x).
Using this property, the eigenvalues of T (which are the zeroes of p(n, x)) are located not by means of bisection [6] [8] [10] [11], but by means of linear interpolation (which converges faster) [4, AP 212 and 232] [18]; in valsymtri, the procedure zeroin is used for this purpose. Moreover, we have incorporated the following idea from [11] and [18] which simplifies the calculations and avoids overflow of the real number capacity; instead of the principle minors, the ratios of successive principle minors, f(i, x) = p(i, x) / p(i − 1, x), are calculated; these ratios are obtained by the recurrence formula (d[i] is the i-th main diagonal element and bb[i] the square of the i-th codiagonal element of T): f(1, x) = d[1] − x,
f(i, x) = d[i] − x − bb[i − 1] / (if abs(f(i − 1, x)) > machtol then f(i − 1, x) else machtol), i = 2,...,n; where machtol is a given norm, ||T||, of T times the machine precision; thus, the number of sign agreements equals the number of positive ratios f(i, x).
The tolerance for each calculated eigenvalue, lambda, is abs(lambda) × given relative tolerance + machtol.

In vecsymtri, an eigenvector of a symmetric tridiagonal matrix T, corresponding to an approximate eigenvalue, lambda, is calculated by means of inverse iteration [2, p. 321 − 330] [9] [10].
Starting from some initial vector, x, the linear system (T − lambda × I)y = x is solved iteratively (by means of Gaussian elimination with row interchanges), the solution y divided by its Euclidean norm replacing x each time. If the distance between some approximate eigenvalues is smaller than machtol, then they are slightly modified such that the distance between them equals machtol [2, p. 328] [9]. This device, invented by Wilkinson, has the effect that a numerically independent set of eigenvectors is obtained, since inverse iteration is very sensitive to small changes in the approximate values of clustered eigenvalues. If the distance between some eigenvalues is smaller than ||T|| times a given "orthogonalisation parameter" (which should be not smaller than the machine precision divided by the tolerance for the eigenvectors), then in each iteration step, Gram-Schmidt orthogonalisation [2, p. 242 and 606] is carried out, so that the eigenvectors obtained are orthogonal within working precision. Note, however, that the calculated eigenvectors corresponding to a cluster of eigenvalues may deviate substantially from the true eigenvectors. If Gram-Schmidt orthogonalisation yields a null vector, then another initial vector, viz. one of the unit vectors, is chosen and the iteration is started again; since the unit vectors span the whole space, at least one of them is not perpendicular to the desired eigenvector, and, thus, is a suitable initial vector.
The iteration ends, as soon as either the Euclidean norm of the residue (T − lambda × I)x (this norm is calculated as the reciprocal of the Euclidean norm of y) is smaller than ||T|| times the tolerance for the eigenvectors, or the maximum allowed number of iterations has been performed. If the tolerance for the eigenvectors is not too small (it should not be smaller than the relative tolerance for the eigenvalues), then one or two iterations suffice in most cases.

To find eigenvalues of a symmetric matrix M, first Householder's transformation (section 230) is performed and then valsymtri is called. Furthermore, to find the corresponding eigenvectors, vecsymtri is used and then the back transformation (section 230) is carried out. The Euclidean norm of the eigenvectors delivered equals 1 (within working precision).

The procedures of this section, except zeroin, use an auxiliary array em[0:9], or a part of it, in which some data for controlling the iterations must be given and some by-products are delivered.
A survey of these data and by-products follows:
1) general
em[0] is the machine precision; must be given for all procedures.
em[1] is some norm of M or T; must be given for valsymtri and vecsymtri; the other procedures deliver the infinity norm of M, produced by tfmsymtri2 or tfmsymtri1.
2) for calculating eigenvalues
em[2] must be given for, and em[3] is delivered by all procedures, except vecsymtri;
em[2] is the relative tolerance for the eigenvalues; more precisely: the tolerance for each calculated eigenvalue, lambda, is abs(lambda) × em[2] + em[1] × em[0];
em[3] is the number of iterations performed for the calculation of the eigenvalues.
3) for calculating eigenvectors
em[4], em[6] and em[8] must be given for, and em[5], em[7] and em[9] are delivered by vecsymtri, eigsym2 and eigsym1; moreover, em[5] must be given for vecsymtri only if n1 > 1.
em[4] is the orthogonalisation parameter;
em[5] is the number of eigenvectors involved in the last Gram-Schmidt orthogonalisation; if, in the calculation of the last eigenvector, no Gram-Schmidt orthogonalisation is carried out, then this number equals 1;
em[6] is the tolerance for the eigenvectors; more precisely: the inverse iteration ends if the Euclidean norm of the residue is smaller than em[1] × em[6];
em[7] is the maximum Euclidean norm of the residues of the calculated eigenvectors of T;
em[8] is the maximum number of inverse iterations allowed for the calculation of each eigenvector;
em[9] is the largest number of iterations performed for the calculation of some eigenvector; the value em[8] + 1 is delivered if the Euclidean norm of the residue for one or more eigenvectors does not become smaller than or equal to em[1] × em[6] within em[8] iterations; nevertheless the eigenvectors may then very well be useful, this should be judged from the value delivered in em[7] or from some other test.
The tolerances should satisfy em[0] < em[2] < em[6];
the orthogonalisation parameter should satisfy em[4] ≥ em[0] / em[6];
For the X8, suitable values of the data to be given in em are:
em[0] = $_{10}$-12, em[2] = $_{10}$-10, em[4] = 0.01, em[6] = $_{10}$-8, em[8] = 5.

```
comment mca 2310;
boolean procedure zeroin(x, y, fx, tolx); real x, y, fx, tolx;
begin real a, fa, b, fb, c, fc, tol, m, p, q;
    a:= x; fa:= fx; b:= x:= y; fb:= fx;
interpolate: c:= a; fc:= fa;
extrapolate: if abs(fc) < abs(fb) then
    begin a:= b; fa:= fb; x:= b:= c; fb:= fc; c:= a; fc:= fa
    end interchange;
    tol:= tolx; m:= (c + b) × .5; if abs(m - b) > tol then
    begin p:= (b - a) × fb; if p > 0 then q:= fa - fb else
        begin q:= fb - fa; p:= - p end;
        a:= b; fa:= fb;
        x:= b:= if p < abs(q) × tol then (if c > b then b + tol
        else b - tol) else if p < (m - b) × q then p / q + b
        else m; fb:= fx;
        goto if (if fc > 0 then fb ≥ 0 else fb ≤ 0) then interpolate
        else extrapolate
    end;
    y:= c; zeroin:= if fc ≥ 0 then fb ≤ 0 else fb ≥ 0
end zeroin;
```

Description mca 2310

zeroin searches for a zero of a function between the given values of x and y within a certain tolerance. The function and the tolerance are, in this order, given by the actual parameters for fx and tolx, which are expressions depending on the Jensen variable x.

zeroin:= true, if either the function values at the given point x and y have different sign, or the procedure finds some point in between at which the sign of the function value differs from that at x and y. Then zeroin calculates and delivers two values x and y lying within the given interval, having function values of different sign and satisfying abs(x − y) $\leq$ 2 × tolx.

Moreover, the absolute function value is not greater at x than at y, so that the delivered value of x is the best value for the zero.

If the function has a continuous second derivative, the order of convergence is about 1.6.

zeroin:= false, if the procedure fails to find points at which the function values have different sign. Then the delivered values of x and y satisfy all the above conditions, except the sign change condition.

One has to take care that tolx is never smaller than the machine precision. Then in either case the process is completed after a finite number of steps, an upper bound for the required number of steps being the length of the given interval divided by the minimum of the tolerance.

```
comment mca 2311;
procedure valsymtri(d, bb, n, n1, n2, val, em); value n, n1, n2;
integer n, n1, n2; array d, bb, val, em;
begin integer k, count;
    real max, x, y, macheps, norm, re, machtol, ub, lb, lambda;

    real procedure quot;
    begin integer p, i;
        real f;
        count:= count + 1; p:= k; i:= 1; f:= d[1] - x; goto test;
    high: i:= i + 1;
        f:= d[i] - x - (if abs(f) > machtol then bb[i - 1] / f else
        bb[i - 1] / machtol);
    test: if f ≤ 0 then p:= p + 1; if p < i then
        begin quot:= max; lb:= x end
        else
        begin if p > n then quot:= if i = n ∧ f ≤ 0 then f else - max
            else
            begin if i < n then goto high;
                quot:= if f > 0 then f else max;
                if x < ub then ub:= x
            end
        end
    end quot;

    macheps:= em[0]; norm:= em[1]; re:= em[2];
    machtol:= norm × macheps; max:= norm / macheps; count:= 0;
    ub:= 1.1 × norm; lb:= - ub; lambda:= ub;
    for k:= n1 step 1 until n2 do
    begin x:= lb; y:= ub; lb:= - 1.1 × norm;
        zeroin(x, y, quot, abs(x) × re + machtol);
        val[k]:= lambda:= if x > lambda then lambda else x;
        if ub > x then ub:= if x > y then x else y
    end;
    em[3]:= count
end valsymtri;
```

Description mca 2311
valsymtri calculates the n1–th to n2–th eigenvalues of the n–th order symmetric tridiagonal matrix T whose main diagonal is given in array d[1:n], and whose squared codiagonal elements are given in array bb[1:n – 1].
The following elements of array em[0:3] must be given (see also p.19).
em[0]: the machine precision;
em[1]: a norm of T;
em[2]: the relative tolerance for the eigenvalues.
The eigenvalues are calculated in monotonically nonincreasing order and delivered in array val[n1:n2]; in particular, if n1 = 1 and n2 = n, then all eigenvalues are calculated.
Moreover em[3]:= the number of iterations performed.
valsymtri uses zeroin (mca 2310).

```
comment mca 2312;
procedure vecsymtri(d, b, n, n1, n2, val, vec, em); value n, n1, n2;
integer n, n1, n2; array d, b, val, vec, em;
begin integer i, j, k, count, maxcount, countlim, orth, ind;
    real bi, bi1, u, w, y, mi1, lambda, oldlambda, ortheps, valspread,
    spr, res, maxres, oldres, norm, newnorm, oldnorm, machtol, vectol;
    array m, p, q, r, x[1:n];
    boolean array int[1:n];
    norm:= em[1]; machtol:= em[0] × norm; valspread:= em[4] × norm;
    vectol:= em[6] × norm; countlim:= em[8]; ortheps:= sqrt(em[0]);
    maxcount:= ind:= 0; maxres:= 0;
    if n1 > 1 then
    begin orth:= em[5]; oldlambda:= val[n1 - orth];
        for k:= n1 - orth + 1 step 1 until n1 - 1 do
        begin lambda:= val[k]; spr:= oldlambda - lambda;
            if spr < machtol then lambda:= oldlambda - machtol;
            oldlambda:= lambda
        end
    end
    else orth:= 1;
    for k:= n1 step 1 until n2 do
    begin lambda:= val[k]; if k > 1 then
        begin spr:= oldlambda - lambda; if spr < valspread then
            begin if spr < machtol then lambda:= oldlambda - machtol;
                orth:= orth + 1
            end
            else orth:= 1
        end;
        count:= 0; u:= d[1] - lambda; bi:= w:= b[1];
        if abs(bi) < machtol then bi:= machtol;
        for i:= 1 step 1 until n - 1 do
        begin bi1:= b[i + 1];
            if abs(bi1) < machtol then bi1:= machtol;
            if abs(bi) ≥ abs(u) then
            begin mi1:= m[i + 1]:= u / bi; p[i]:= bi;
                y:= q[i]:= d[i + 1] - lambda; r[i]:= bi1;
                u:= w - mi1 × y; w:= - mi1 × bi1; int[i]:= true
            end
            else
            begin mi1:= m[i + 1]:= bi / u; p[i]:= u; q[i]:= w;
                r[i]:= 0; u:= d[i + 1] - lambda - mi1 × w; w:= bi1;
                int[i]:= false
            end;
            x[i]:= 1; bi:= bi1
        end transform;
        p[n]:= if abs(u) < machtol then machtol else u;
        q[n]:= r[n]:= 0; x[n]:= 1; goto entry;
    iterate: w:= x[1];
        for i:= 2 step 1 until n do
        begin if int[i - 1] then
            begin u:= w; w:= x[i - 1]:= x[i] end
            else u:= x[i]; w:= x[i]:= u - m[i] × w
        end alternate;
```

Description mca 2312
vecsymtri calculates the eigenvectors corresponding to the n1–th to n2–th eigenvalues of the n–th order symmetric tridiagonal matrix T, whose main diagonal and codiagonal, the latter followed by an additional element 0, are given in array d, b[1:n].
In array em[0:9] one must give the following data (see also p. 19).
em[0]: the machine precision;
em[1]: a norm of T;
em[4]: the orthogonalisation parameter;
em[6]: the tolerance for the eigenvectors;
em[8]: the maximum number of iterations allowed for the calculation of each eigenvector.

If n1 = 1, then the largest n2 eigenvalues must be given in array val[1:n2] in monotonically nonincreasing order. Then vecsymtri delivers the corresponding eigenvectors in the columns of array vec[1:n, 1:n2].
Moreover, in em the following results are delivered:
em[5]:= the number of eigenvectors involved in the last Gram–Schmidt orthogonalisation;
em[7]:= the maximum Euclidean norm of the residues;
em[9]:= the largest number of iterations performed for the calculation of some eigenvector;
if, however, for some calculated eigenvector, the Euclidean norm of the residue remains greater than em[1] × em[6], then em[9]:= em[8] + 1.

If n1 > 1, then vecsymtri should be preceded by one or more calls of vecsymtri producing the eigenvectors corresponding to the largest n1 − 1 eigenvalues. Then, in addition to the data mentioned above, one must give em[5], as produced by the last call of vecsymtri;
the k–th to n2–th eigenvalues, where k = n1 − em[5], must be given in array val[k:n2] in monotonically nonincreasing order (the k–th to (n1 − 1)–th eigenvalues being needed for Wilkinson's device), and the corresponding eigenvectors up to the (n1 − 1)–th (which are needed for the Gram–Schmidt orthogonalisation) in the corresponding columns of array vec[1:n, k:n2].
Then vecsymtri calculates the eigenvectors corresponding to the n1–th to n2–th eigenvalues and delivers them in the corresponding columns of vec; moreover, results as mentioned above are delivered in em, but they now concern the calculation of the n1–th to n2–th eigenvectors only.

Summarising, we have: two subsequent calls "vecsymtri (d, b, n, 1, n1 − 1, val, vec, em)" and "vecsymtri (d, b, n, n1, n2, val, vec, em)" are equivalent to one call "vecsymtri (d, b, n, 1, n2, val, vec, em)", except for the results delivered in em[7] and em[9].
vecsymtri uses vecvec, tamvec and elmveccol [3, chapter 20].

```
    entry: u:= w:= 0;
        for i:= n step - 1 until 1 do
        begin y:= u; u:= x[i]:= (x[i] - q[i] × u - r[i] × w) / p[i];
            w:= y
        end next iteration;
        newnorm:= sqrt(vecvec(1, n, 0, x, x)); if orth > 1 then
        begin oldnorm:= newnorm;
            for j:= k - orth + 1 step 1 until k - 1 do
            elmveccol(1, n, j, x, vec, - tamvec(1, n, j, vec, x));
            newnorm:= sqrt(vecvec(1, n, 0, x, x));
            if newnorm < ortheps × oldnorm then
            begin ind:= ind + 1; count:= 1;
                for i:= 1 step 1 until ind - 1, ind + 1 step 1 until
                n do x[i]:= 0; x[ind]:= 1; if ind = n then ind:= 0;
                goto iterate
            end new start
        end orthogonalisation;
        res:= 1 / newnorm; if res > vectol ∨ count = 0 then
        begin count:= count + 1; if count ≤ countlim then
            begin for i:= 1 step 1 until n do x[i]:= x[i] × res;
                goto iterate
            end
        end;
        for i:= 1 step 1 until n do vec[i,k]:= x[i] × res;
        if count > maxcount then maxcount:= count;
        if res > maxres then maxres:= res; oldlambda:= lambda
    end;
    em[5]:= orth; em[7]:= maxres; em[9]:= maxcount
end vecsymtri;
```

```
comment mca 2313;
procedure eigvalsym2(a, n, numval, val, em); value n, numval;
integer n ,numval; array a, val, em;
begin array b, bb, d[1:n];
    tfmsymtri2(a, n, d, b, bb, em);
    valsymtri(d, bb, n, 1, numval, val, em)
end eigvalsym2;


comment mca 2314;
procedure eigsym2(a, n, numval, val, vec, em); value n, numval;
integer n, numval; array a, val, vec, em;
begin array b, bb, d[1:n];
    tfmsymtri2(a, n, d, b, bb, em);
    valsymtri(d, bb, n, 1, numval, val, em);
    vecsymtri(d, b, n, 1, numval, val, vec, em);
    baksymtri2(a, n, 1, numval, vec)
end eigsym2;
```

Description mca 2313
eigvalsym2 calculates the largest numval eigenvalues of the n-th order symmetric matrix M whose upper triangle is given in array a[1:n, 1:n]. In array em[0:3] the elements with even subscript must be given (see also p. 19), viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the eigenvalues.
eigvalsym2 delivers the first to numval-th eigenvalues of M in monotonically nonincreasing order in array val[1:numval], and the data for Householder's back transformation in the upper triangle of a. Moreover,
em[1]:= the infinity norm of M;
em[3]:= the number of iterations performed.
eigvalsym2 uses tfmsymtri2 (mca 2300), valsymtri and, indirectly, also zeroin (this section) and tamvec, matmat, tammat, elmveccol, elmcolvec and elmcol [3, chapter 20].

Description mca 2314
eigsym2 calculates the largest numval eigenvalues and corresponding eigenvectors of the n-th order symmetric matrix M whose upper triangle is given in array a[1:n, 1:n]. In array em[0:9], the elements with even subscript must be given (see also p. 19), viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the eigenvalues;
em[4]: the orthogonalisation parameter;
em[6]: the tolerance for the eigenvectors;
em[8]: the maximum number of inverse iterations allowed for the calculation of each eigenvector.
eigsym2 delivers the first to numval-th eigenvalues in monotonically nonincreasing order in array val[1:numval], the corresponding eigenvectors in the columns of array vec[1:n, 1:numval], and the data for Householder's back transformation in the upper triangle of a. Moreover,
em[1]:= the infinity norm of M;
em[3]:= the number of iterations performed for the calculation of the eigenvalues;
em[5]:= the number of eigenvectors involved in the last Gram-Schmidt orthogonalisation;
em[7]:= the maximum Euclidean norm of the residues of the calculated eigenvectors (of the transformed matrix);
em[9]:= the largest number of inverse iterations performed for the calculation of some eigenvector; if, however, for some calculated eigenvector, the Euclidean norm of the residue remains greater than em[1] × em[6], then em[9]:= em[8] + 1.
eigsym2 uses tfmsymtri2 and baksymtri2 (section 230), valsymtri, vecsymtri and, indirectly, also zeroin (this section) and vecvec, tamvec, matmat, tammat, elmveccol, elmcolvec and elmcol [3, chapter 20].

```
comment mca 2318;
procedure eigvalsym1(a, n, numval, val, em); value n, numval;
integer n, numval; array a, val, em;
begin array b, bb, d[1:n];
    tfmsymtri1(a, n, d, b, bb, em);
    valsymtri(d, bb, n, 1, numval, val, em)
end eigvalsym1;


comment mca 2319;
procedure eigsym1(a, n, numval, val, vec, em); value n, numval;
integer n, numval; array a, val, vec, em;
begin array b, bb, d[1:n];
    tfmsymtri1(a, n, d, b, bb, em);
    valsymtri(d, bb, n, 1, numval, val, em);
    vecsymtri(d, b, n, 1, numval, val, vec, em);
    baksymtri1(a, n, 1, numval, vec)
end eigsym1;
```

Description mca 2318
eigvalsym1 calculates the largest numval eigenvalues of the n-th order symmetric matrix M whose upper triangle is given in array a[1:(n + 1) × n : 2] in such a way that the (i, j)-th element of M is a[(j - 1) × j : 2 + i] for $1 \leq i \leq j \leq n$.
In array em[0:3] the elements with even subscript must be given (see also p. 19), viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the eigenvalues.
eigvalsym1 delivers the 1-st to numval-th eigenvalues of M in monotonically nonincreasing order in array val[1:numval], and the data for Householder's back transformation in a.
Moreover,
em[1]:= the infinity norm of M;
em[3]:= the number of iterations performed.
eigvalsym1 uses tfmsymtri1 (mca 2305), valsymtri and, indirectly, also zeroin (this section) and vecvec, seqvec and elmvec [3, chapter 20].

Description mca 2319
eigsym1 calculates the largest numval eigenvalues and corresponding eigenvectors of the n-th order symmetric matrix M whose upper triangle is given in array a[1:(n + 1) × n : 2] in such a way that the (i, j)-th element of M is a[(j - 1) × j : 2 + i] for $1 \leq i \leq j \leq n$.
In array em[0:9], the elements with even subscript must be given as for eigsym2.
eigsym1 delivers the 1-st to numval-th eigenvalues in monotonically nonincreasing order in array val[1:numval], the corresponding eigenvectors in the columns of array vec[1:n, 1:numval], and the data for Householder's back transformation in a.
Moreover, in the elements of em with odd subscript, the same results are delivered as by eigsym2.
eigsym1 uses tfmsymtri1 and baksymtri1 (section 230), valsymtri, vecsymtri and, indirectly, also zeroin (this section) and vecvec, tamvec, seqvec, elmvec and elmveccol [3, chapter 20].

Section 232 QR iteration

This section contains procedures for calculating the eigenvalues or the eigenvalues and eigenvectors of real symmetric matrices: qrivalsym2 and qrivalsym1 calculate all eigenvalues of a symmetric matrix; qrisym calculates the eigenvectors as well; qrivalsymtri calculates all eigenvalues of a symmetric tridiagonal matrix; qrisymtri calculates the eigenvectors as well.

The method used in qrivalsymtri and qrisymtri for calculating the eigenvalues of a symmetric tridiagonal matrix T is QR iteration [14] [15] [16] [17] [18]; our procedures being most similar to those published in [18] and [16].
In each step, matrix T is replaced by the similar matrix Q'TQ, where Q is an orthogonal matrix chosen in such a way that, for some suitable "shift" s, Q'(T - sI) is an upper-triangular matrix, R (thus, T - sI = QR, which explains the name of the method).
The matrices Q and R could be calculated by means of Gram-Schmidt orthogonalisation [2, p. 242]; hence, it is obvious that Q has upper-Hessenberg form.

The similar matrix Q'TQ is again symmetric (because Q is orthogonal) and tridiagonal (because R is upper-triangular and Q has upper-Hessenberg form).
The sequence of iterates T converges to a (nearly) diagonal matrix, D, similar to the given matrix, so that the main-diagonal elements of D are (approximately) equal to the required eigenvalues.
As soon as, for some k, the k-th element of the codiagonal of an iterate T has an absolute value smaller than some tolerance, then this element is neglected, and matrix T is subdivided into a submatrix of order k consisting of the first k rows and columns and a submatrix of order n - k consisting of the last n - k rows and columns of T; these "principle" submatrices are then considered and handled separately. Eigenvalues and eigenvectors of submatrices of order 1 or 2 are calculated directly, so that the process is completed if T is subdivided into principle submatrices of order 1 or 2 only. For k = n - 1, moreover, a weaker criterion, due to Kahan [17] [18], for subdividing the matrix is applied, which criterion is especially effective if both the (n - 1)-th and (n - 2)-th elements of the codiagonal become small.

In each step, the shift s is chosen as follows: Let B denote the lower right 2 by 2 submatrix of the considered principle submatrix of T; then s equals that eigenvalue of B closest to its last main-diagonal element. With this choice of s the codiagonal element of B converges to 0. The convergence is cubic for a simple eigenvalue of T and linear for a multiple one. However, the iteration is discontinued if a given maximum allowed number of iterations has been performed.

The procedure qrisymtri also calculates the eigenvectors of the symmetric matrix M = STS', where T is a symmetric tridiagonal matrix and S an orthogonal transformation matrix (cf. section 230) as follows [16]: in each QR iteration step, a matrix X whose initial value is S is replaced by XQ in each step; thus on completion of the process, X is the matrix of eigenvectors of M. In particular, if S = I then the eigenvectors produced are those of T. The procedure qrivalsymtri is a square-root-free version of the QR method due to Ortega and Kaiser [14] [15] [18] in contrast to qrisymtri; hence these procedures are not numerically equivalent as to the calculation of the eigenvalues.

The procedures of this section use an auxiliary array em[0:5], or a part of it, in which some data for controlling the iterations must be given and some by-products are delivered. A survey of these data and by-products follows.

1) general
em[0] is the machine precision; must be given for qrivalsymtri, qrivalsym2, qrisym and qrivalsym1.
em[1] is some norm of M or T; must be given for qrivalsymtri and qrisymtri; the other procedures deliver the infinity norm of M, produced by tfmsymtri2 or tfsymtri1.

2) for the QR iteration
em[2] and em[4] must be given for, and em[3] and em[5] are delivered by, all procedures.
em[2] is the relative tolerance for the QR iteration; if the absolute value of some codiagonal element of the iterated matrix is smaller than em[1] × em[2], then this element is neglected and the matrix is subdivided.
em[3] is the maximum absolute value of the codiagonal elements neglected.
em[4] is the maximum allowed number of QR iterations;
em[5] is the number of QR iterations performed; the value em[4] + 1 is delivered, if the QR iteration process is not completed within em[4] iterations.
The tolerance em[2] should be chosen not smaller than em[0].
For the X8, suitable values of the data to be given in em are
em[0] = $_{10}$-12, em[2] = $_{10}$-12, em[4] = 5 × n.

```
comment mca 2320;
integer procedure qrivalsymtri(d, bb, n, em); value n; integer n;
array d, bb, em;
begin integer count, max, n1, k, k1;
    real tol, tol2, macheps2, bbmax, t, r, w, c, s, shift, u, g, t2,
    w2, p2, dk, cos2, sin2, oldcos2;
    tol:= em[2] × em[1]; tol2:= tol × tol; macheps2:= em[0] ↑ 2;
    count:= 0; bbmax:= 0; max:= em[4];
in: k:= n; n1:= n - 1;
next: k:= k - 1; if k > 0 then
    begin if bb[k] ≥ tol2 then goto next;
        if bb[k] > bbmax then bbmax:= bb[k]
    end;
    if k = n1 then n:= n1 else
    begin
    twoby2: t:= d[n] - d[n1]; r:= bb[n1]; if k < n - 2 then
        begin w:= bb[n - 2]; c:= t × t; s:= r / (c + w);
            g:= (w + s × c) × s; if g < tol2 then
            begin n:= n - 1; n1:= n - 1; if g > bbmax then bbmax:= g;
                goto twoby2
            end
        end negligible bb;
        if abs(t) < tol then s:= sqrt(r) else
        begin w:= 2 / t; s:= w × r / (sqrt(w × w × r + 1) + 1) end;
        if k = n - 2 then
        begin d[n]:= d[n] + s; d[n1]:= d[n1] - s; n:= n - 2 end
        else
        begin count:= count + 1; if count > max then goto end;
            shift:= d[n] + s; if abs(t) < tol then
            begin w:= d[n1] - s;
                if abs(w) < abs(shift) then shift:= w
            end;
            k:= k + 1; g:= d[k] - shift; t2:= g × g; w2:= bb[k];
            p2:= t2 + w2; oldcos2:= 1;
            for k:= k + 1 step 1 until n do
            begin k1:= k - 1; sin2:= w2 / p2; cos2:= t2 / p2;
                dk:= d[k] - shift; u:= (g + dk) × sin2;
                d[k1]:= g + u + shift; g:= dk - u;
                t2:= if cos2 < macheps2 then w2 × oldcos2 else g × g
                / cos2; w2:= bb[k]; p2:= w2 + t2; bb[k1]:= sin2 × p2;
                oldcos2:= cos2
            end;
            d[n]:= g + shift
        end sweep
    end;
    if n > 0 then goto in;
end: em[3]:= sqrt(bbmax); em[5]:= count; qrivalsymtri:= n
end qrivalsymtri;
```

Description mca 2320

qrivalsymtri calculates all eigenvalues of the n-th order symmetric tridiagonal matrix T whose main diagonal is given in array d[1:n] and the squares of whose codiagonal elements with an additional last element 0 are given in array bb[1:n].

In array em[0:5] the following data must be given (see also p. 33):

em[0]: the machine precision;
em[1]: a norm of T;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of iterations.

The eigenvalues of T are delivered in d and the squared codiagonal elements of the tridiagonal matrix resulting from the QR iteration are delivered in bb. Moreover,

em[3]:= the maximum absolute value of the codiagonal elements neglected;
em[5]:= the number of iterations performed.

Furthermore, qrivalsymtri:= 0, provided the process is completed within em[4] iterations; otherwise, qrivalsymtri:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n - k elements of d are approximate eigenvalues of T.

```
comment mca 2321;
integer procedure qrisymtri(a, n, d, b, bb, em); value n; integer n;
array a, d, b, bb, em;
begin integer i, j, j1, k, m, m1, count, max;
    real bbmax, r, s, sin, t, c, cos, oldcos, g, p, w, tol, tol2,
    lambda, dk1, a0, a1;
    tol:= em[2] × em[1]; tol2:= tol × tol; count:= 0; bbmax:= 0;
    max:= em[4]; m:= n;
in: k:= m; m1:= m - 1;
next: k:= k - 1; if k > 0 then
    begin if bb[k] ≥ tol2 then goto next;
        if bb[k] > bbmax then bbmax:= bb[k]
    end;
    if k = m1 then m:= m1 else
    begin
    twoby2: t:= d[m] - d[m1]; r:= bb[m1];
        if k < m - 2 then
        begin w:= bb[m - 2]; c:= t × t;
            s:= r / (c + w); g:= (s × c + w) × s; if g < tol2 then
            begin m:= m - 1; m1:= m - 1; if g > bbmax then
                bbmax:= g; goto twoby2
            end
        end negligible bb;
        if abs(t) < tol then s:= sqrt(r) else
        begin w:= 2 / t; s:= w × r / (sqrt(w × w × r + 1) + 1) end;
        if k = m - 2 then
        begin d[m]:= d[m] + s; d[m1]:= d[m1] - s;
            t:= - s / b[m1]; r:= sqrt(t × t + 1); cos:= 1 / r;
            sin:= t / r; rotcol(1, n, m1, m, a, cos, sin); m:= m - 2
        end
        else
        begin count:= count + 1; if count > max then goto end;
            lambda:= d[m] + s; if abs(t) < tol then
            begin w:= d[m1] - s;
                if abs(w) < abs(lambda) then lambda:= w
            end;
            k:= k + 1; t:= d[k] - lambda; cos:= 1; w:= b[k];
            p:= sqrt(t × t + w × w); j1:= k;
            for j:= k + 1 step 1 until m do
            begin oldcos:= cos; cos:= t / p; sin:= w / p;
                dk1:= d[j] - lambda; t:= oldcos × t;
                d[j1]:= (t + dk1) × sin × sin + lambda + t;
                t:= cos × dk1 - sin × w × oldcos; w:= b[j];
                p:= sqrt(t × t + w × w); g:= b[j1]:= sin × p;
                bb[j1]:= g × g; rotcol(1, n, j1, j, a, cos, sin);
                j1:= j
            end;
            d[m]:= cos × t + lambda; if t < 0 then b[m1]:= - g
        end qrstep
    end;
    if m > 0 then goto in;
end: em[3]:= sqrt(bbmax); em[5]:= count; qrisymtri:= m
end qrisymtri;
```

Description mca 2321

qrisymtri calculates all eigenvalues of the n–th order symmetric tridiagonal matrix T whose main diagonal is given in array d[1:n], and whose codiagonal elements with an additional last element 0 are given in array b[1:n] with the squares thereof in array bb[1:n].
Moreover, qrisymtri calculates the eigenvectors of M = STS', where S is the orthogonal transformation matrix given in array a[1:n, 1:n] (e.g. as produced by tfmsymtri2 and tfmprevec, section 230).
In array em[1:5], the following data must be given (see also p. 33):
em[1]: a norm of T;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of iterations.
The eigenvalues of T (and thus also of M) are delivered in d and the eigenvectors of M in the corresponding columns of a. The codiagonal of the tridiagonal matrix resulting from the QR iteration is delivered in b, and the squares of these codiagonal elements in bb.
Moreover,
em[3]:= the maximum absolute value of the codiagonal elements neglected;
em[5]:= the number of QR iterations performed.
Furthermore, qrisymtri:= 0, provided the process is completed within em[4] iterations; otherwise, qrisymtri:= the number, k, of eigenvalues, not calculated, em[5]:= em[4] + 1, and only the last n – k elements of d and columns of a are approximate eigenvalues and eigenvectors of M.
qrisymtri uses rotcol [3, mca 2031].

```
comment mca 2322;
integer procedure qrivalsym2(a, n, val, em); value n; integer n;
array a, val, em;
begin array b, bb[1:n];
    tfmsymtri2(a, n, val, b, bb, em);
    qrivalsym2:= qrivalsymtri(val, bb, n, em)
end qrivalsym2;
```

Description mca 2322

qrivalsym2 calculates all eigenvalues of the n-th order symmetric matrix M whose upper triangle is given in array a[1:n, 1:n].
In array em[0:5], the elements with even subscript must be given (see also p. 33) viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of iterations.
The calculated eigenvalues of M are delivered, in array val[1:n], and the data for Householder's back transformation in the upper triangle of a.
em[1]:= the infinity norm of M;
em[3]:= the maximum absolute value of the codiagonal elements neglected;
em[5]:= the number of iterations performed.
Furthermore, qrivalsym2:= 0, provided the QR iteration is completed within em[4] iterations; otherwise, qrivalsym2:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n - k elements of val are approximate eigenvalues of M.
qrivalsym2 uses tfmsymtri2 (mca 2300), qrivalsymtri (mca 2320) and, indirectly, also tamvec, matmat, tammat, elmveccol, elmcolvec and elmcol [3, chapter 20].

```
comment mca 2323;
integer procedure qrisym(a, n, val, em); value n; integer n;
array a, val, em;
begin array b, bb[1:n];
    tfmsymtri2(a, n, val, b, bb, em); tfmprevec(a, n);
    qrisym:= qrisymtri(a, n, val, b, bb, em)
end qrisym;


comment mca 2327;
integer procedure qrivalsym1(a, n, val, em); value n;
integer n; array a, val, em;
begin   array b, bb[1 : n];
    tfmsymtri1(a, n, val, b, bb, em);
    qrivalsym1:= qrivalsymtri(val, bb, n, em)
end qrivalsym1;
```

Description mca 2323

qrisym calculates all eigenvalues and eigenvectors of the n-th order symmetric matrix M whose upper triangle is given in array a[1:n, 1:n]. In array em[0:5], the elements with even subscript must be given (see also p. 33), viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of iterations.
The calculated eigenvalues of M are delivered in array val[1:n] and the eigenvectors of M in the corresponding columns of array a[1:n, 1:n]. Moreover,
em[1]:= the infinity norm of M;
em[3]:= the maximum absolute value of the codiagonal elements neglected;
em[5]:= the number of iterations performed.
Furthermore, qrisym:= 0, provided the process is completed within em[4] iterations; otherwise, qrisym:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n - k elements of val and columns of a are approximate eigenvalues and eigenvectors of M.
qrisym uses tfmsymtri2, tfmprevec (section 230), qrisymtri (mca 2321) and, indirectly, also tamvec, matmat, tammat, elmveccol, elmcolvec, elmcol and rotcol [3, chapter 20].

Description mca 2327

qrivalsym1 calculates all eigenvalues of the n-th order symmetric matrix M whose upper triangle is given in array a[1:(n + 1) × n : 2] in such a way that the (i,j)-th element of M is a[(j - 1) × j : 2 + i] for $1 \leq i \leq j \leq n$.
In array em[0:5], the elements with even subscript must be given as for qrivalsym2.
The calculated eigenvalues of M are delivered in array val[1:n] and the data for Householder's back transformation in a. Moreover, in the elements of em with odd subscript, the same results are delivered as by qrivalsym2.
Furthermore, qrivalsym1:= 0, provided the QR iteration is completed within em[4] iterations; otherwise, qrivalsym1:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n - k elements of val are approximate eigenvalues of M.
qrivalsym1 uses tfmsymtri1 (mca 2305), qrivalsymtri (mca 2320) and, indirectly, also vecvec, seqvec and elmvec [3, chapter 20].

CHAPTER 24

EIGENSYSTEMS OF REAL MATRICES

This chapter contains procedures for calculating eigenvalues and / or eigenvectors of real matrices. To solve the eigenproblem the matrix is first equilibrated by means of a diagonal similarity transformation and the equilibrated matrix is transformed into an upper-Hessenberg matrix by means of a stabilized triangular transformation (section 240).

The eigenvalues of an upper-Hessenberg matrix are calculated by means of QR iteration in one of two variants, viz. single QR iteration, to be used if all eigenvalues are real (section 241), and double QR iteration for finding real and complex eigenvalues (section 242). Unfortunately, the QR-iteration process is not always convergent; counterexamples of matrices having real eigenvalues only are not known to the authors; for counterexamples of matrices having complex eigenvalues, see p. 74.

For the calculation of the eigenvectors, two methods have been chosen, viz. inverse iteration and a more direct method. The inverse iteration converges rapidly in nearly all cases, whereas the direct method does not perform any iteration process after the QR iteration. We have programmed the inverse iteration in a version for real eigenvectors (section 241) and one for complex eigenvectors (section 242), but the direct method only for real eigenvectors (section 241); the version of the latter for complex eigenvectors would also be useful, and is not included because it is not yet completed.

When the processes converge, they yield eigenvalues and eigenvectors in reasonable precision, provided that the matrix is not too ill-conditioned with respect to its eigenvalue and eigenvector problems [2, chapter 2].
A measure for the sensitivity of an eigenvalue of a matrix M to small changes in its elements is the quantity $k = ||x||\ ||y||\ /\ |y'x|$, where x and y are the corresponding (suitably chosen if the eigenvalue is multiple) eigenvectors of M and M'. If M is (approximately) equal to a nondiagonalisable matrix, then k is much larger than 1 for some of its eigenvalues, and M is ill-conditioned with respect to both its eigenvalue and eigenvector problems [2, p. 69].
If, on the other hand, the quantities k are not much larger than 1 for all eigenvalues of M (in particular, the quantities k are equal to 1 for normal matrices, i.e. matrices having a unitary matrix of eigenvectors), then M is well-conditioned with respect to its eigenvalue problem, the eigenvalues obtained by our procedures have a reasonable precision, and the eigenvectors obtained are almost always numerically independent.

Note, however, that a matrix which is well-conditioned with respect to its eigenvalue problem, may be ill-conditioned with respect to its eigenvector problem, this being the case if the matrix has closely clustered eigenvalues; the calculated eigenvectors corresponding to such eigenvalues may very well deviate substantially from the true eigenvectors.

The computation time is roughly proportional to n cubed, but obviously depends on the number of iterations required (see Appendix). Inverse iteration and the direct method for calculating eigenvectors are competitive as to accuracy and computation time.
It is worth remarking, however, that, in our limited experience, the eigenvectors obtained by the direct method are numerically not worse, and sometimes better, independent than those obtained by inverse iteration.

As to the calculation of eigenvalues, the QR iteration appears to be much faster than the nondeflating methods using Hyman's formula for evaluating the characteristic polynomial [2, p. 426] and a standard iteration process (e.g. linear interpolation [4, AP 239], or Laguerre's method [20] [24] [25]) for locating the eigenvalues. A quite different type of method for calculating eigenvalues and eigenvectors is Eberlein's method [26] [2, p. 568] [27], which seems to be quite satisfactory.

As to the memory space required for our procedures, we consider only the n by n arrays used, since the size of the one-dimensional arrays is negligible. The procedures reaeigval (mca 2412) and comeigval (mca 2422), which calculate only eigenvalues, use only one n by n array for the given matrix and as working space. The procedures reaeig1 (mca 2414) and comeig1 (mca 2424), which moreover, calculate the eigenvectors by means of inverse iteration, use three n by n arrays, one for the given matrix, one for the eigenvectors, and a local one as working space; the procedures reaeig2 (mca 2415) and comeig2 (mca 2425), which are numerically equivalent to reaeig1 and comeig1, use only one n by n array for the given matrix, as working space and for the eigenvectors, but they use also $2 \times n \uparrow 2$ real number locations on backing storage as working space. The procedure reaeig3 (mca 2417), which calculates the eigenvalues and eigenvectors by means of QR iteration followed by the direct method, uses two n by n arrays, one for the given matrix and as working space, and one for the eigenvectors.

Section 240 Wilkinson's transformation and equilibration

This section contains a procedure for transforming a matrix into a similar upper-Hessenberg matrix, a procedure for equilibrating a matrix, and procedures for performing the corresponding back transformations:
tfmreahes transforms a matrix into a similar upper-Hessenberg matrix by means of Wilkinson's transformation;
bakreahes1 (bakreahes2) performs the corresponding back transformation on a vector (on the columns of a matrix);
eqilbr equilibrates a matrix by means of a diagonal similarity transformation;
baklbr performs the corresponding back transformation on the columns of a matrix.

Wilkinson's transformation is a triangular similarity transformation with stabilizing row and column interchanges [2, p. 353 - 368] transforming a matrix into an upper-Hessenberg matrix [20] [22].
Let M be a given real matrix and H the resulting upper-Hessenberg matrix. The transforming matrix is the product of a permutation matrix, P, and a unit lower-triangular matrix, L; thus, H satisfies PLH = MPL.
The nondiagonal elements in the first column of L are 0, and the row and column interchanges (which determine P) are chosen in such a way that the absolute value of each element of L is at most 1.
These conditions permit a direct calculation of the matrices H, L and P in n - 1 steps.
Let U denote the upper triangle of H, and Q the strict-lower triangle of the matrix ML - LU. Apart from the stabilizing interchanges, the r-th step (r = 1,...,n - 1) is as follows.
The data for the r-th step are the first r columns of U, the second to r-th columns of the strict-lower triangle of L (the first column of L is trivial), the r-th column of Q (which column equals the (r + 1)-th column of the lower triangle of L times the r-th element of the subdiagonal of H), and the last n - r columns of M.
(For the first step, these data are equal to M, since the first main-diagonal element of U equals that of M, and the elements of the first column of Q equal the corresponding elements of M.)
Using these data, the r-th step produces the r-th element of the sub-diagonal of H and the (r + 1)-th column of the strict-lower triangle of L (which are written over the corresponding elements of the r-th column of Q) and the (r + 1)-th columns of U and Q (which are written over the corresponding elements of the (r + 1)-th column of M).
As to the stabilizing row and column interchanges, the r-th step starts with the selection of the r-th "pivot", i.e. an element of the r-th column of Q having maximum absolute value (and which is going to be the r-th element of the subdiagonal of H); then the r-th "pivotal index", pr (i.e. the row index of the r-th pivot), is delivered in an auxiliary array (for the sake of the back transformation); subsequently, the r-th row and column of the data array are inter-changed with the pr-th ones, and then the results of the r-th step sketched above are produced. If, however, all elements in the r-th column of Q below the first subdiagonal have absolute value smaller than the infinity norm of M times the machine precision,

then these elements are replaced by 0 (so, the (r + 1)–th column of L – I is the null vector) and, as an indication, the r–th pivotal index is replaced by 0. The corresponding back transformation transforms a vector x into the vector PLx; if x is an eigenvector of H, then PLx is the corresponding eigenvector of M.

In eqilbr, the matrix M is equilibrated by means of Osborne's diagonal similarity transformation possibly with interchanges [19]. This transformation should be performed before the transformation to Hessenberg form.
A matrix M is said to be "equilibrated", if each column of M has (nearly) the same Euclidean norm as the corresponding row of M.
The transforming diagonal matrix, D, and the equilibrated matrix are calculated iteratively:
in each step a certain column of the matrix is multiplied by, and the corresponding row divided by, a factor which is chosen in such a way that the considered column and row obtain (roughly) the same Euclidean norm (in fact, the factor is rounded to the nearest integral power of 2, in order to prevent rounding errors in the equilibrated matrix, assuming binary floating point arithmetic); the columns and rows are handled in cyclic order. If the matrix does not contain columns or rows whose off–diagonal elements are (nearly) 0, then the process (with unrounded factors) converges, and in practice a few steps are needed to obtain a reasonably equilibrated matrix [19].
If all off–diagonal elements of some considered column (row) are 0 or nearly 0, then this column (row) is interchanged with the first nonzero column (last nonzero row) of the matrix, and, in order to have a similarity transformation, the corresponding rows (columns) are also interchanged; then for the further equilibration, the submatrix is considered which does not contain such zero columns and rows and the corresponding rows and columns. The equilibration process is continued until, in a whole cycle no factor > 2 or < 0.5 and no zero column or row is found, or until $(n + 1) \times n \div 2$ rows and columns have been considered.

The corresponding back transformation transforms a vector x into the vector Dx and performs the corresponding interchanges; if x is an eigenvector of the equilibrated matrix, then the resulting vector is the corresponding eigenvector of the original matrix.

```
comment mca 2400;
procedure tfmreahes(a, n, em, int); value n; integer n; array a, em;
integer array int;
begin integer i, j, j1, k, l;
    real s, t, machtol, macheps, norm;
    array b[0:n - 1];
    macheps:= em[0]; norm:= 0;
    for i:= 1 step 1 until n do
    begin s:= 0;
        for j:= 1 step 1 until n do s:= s + abs(a[i,j]);
        if s > norm then norm:= s
    end;
    em[1]:= norm; machtol:= norm × macheps; int[1]:= 0;
    for j:= 2 step 1 until n do
    begin j1:= j - 1; l:= 0; s:= machtol;
        for k:= j + 1 step 1 until n do
        begin t:= abs(a[k,j1]); if t > s then
            begin l:= k; s:= t end
        end;
        if l ≠ 0 then
        begin if abs(a[j,j1]) < s then
            begin ichrow(1, n, j, l, a); ichcol(1, n, j, l, a) end
            else l:= j; t:= a[j,j1];
            for k:= j + 1 step 1 until n do a[k,j1]:= a[k,j1] / t
        end
        else
        for k:= j + 1 step 1 until n do a[k,j1]:= 0;
        for i:= 1 step 1 until n do b[i - 1]:= a[i,j]:= a[i,j] + (if
        l = 0 then 0 else matmat(j + 1, n, i, j1, a, a)) -
        matvec(1, if j1 < i - 2 then j1 else i - 2, i, a, b);
        int[j]:= l
    end
end tfmreahes;
```

Description mca 2400
tfmreahes transforms the n-th order matrix M given in
array a[1:n, 1:n] into a similar upper-Hessenberg matrix H.
In array em[0:1], one must give the machine precision, em[0].
Matrix H is delivered in the upper triangle and the first subdiagonal
of a, the (nontrivial elements of the) transforming matrix in the
remaining part of a, and the pivotal indices in
integer array int[1:n].
Moreover, em[1]:= the infinity norm of M.
tfmreahes uses matvec, matmat, ichcol and ichrow [3, chapter 20].

```
comment mca 2401;
procedure bakreahes1(a, n, int, v); value n; integer n; array a, v;
integer array int;
begin integer i, l;
    real w;
    array x[1:n];
    for i:= 2 step 1 until n do x[i - 1]:= v[i];
    for i:= n step - 1 until 2 do
    begin v[i]:= v[i] + matvec(1, i - 2, i, a, x); l:= int[i];
        if l > i then
        begin w:= v[i]; v[i]:= v[l]; v[l]:= w end
    end
end bakreahes1;


comment mca 2402;
procedure bakreahes2(a, n, n1, n2, int, vec); value n, n1, n2;
integer n, n1, n2; array a, vec; integer array int;
begin integer i, l, k;
    array u[1:n];
    for i:= n step - 1 until 2 do
    begin for k:= i - 2 step - 1 until 1 do u[k + 1]:= a[i,k];
        for k:= n1 step 1 until n2 do vec[i,k]:= vec[i,k] +
        tamvec(2, i - 1, k, vec, u); l:= int[i];
        if l > i then ichrow(n1, n2, i, l, vec)
    end
end bakreahes2;
```

Description mca 2401
bakreahes1 should be called after tfmreahes, and performs the corresponding back transformation on the vector given as
array v[1:n].
The transforming matrix and the pivotal indices, as produced by tfmreahes, must be given in the part below the first subdiagonal of array a[1:n, 1:n] and in integer array int[1:n].
The resulting vector of the back transformation is overwritten on v.
bakreahes1 uses matvec [3, mca 2001].

Description mca 2402
bakreahes2 should be called after tfmreahes, and performs the corresponding back transformation on the columns of
array vec[1:n, n1:n2].
The transforming matrix and the pivotal indices, as produced by tfmreahes, must be given in the part below the first subdiagonal of array a[1:n, 1:n] and in integer array int[1:n].
The resulting vectors of the back transformation are overwritten on the corresponding columns of vec.
bakreahes2 uses tamvec and ichrow [3, chapter 20].

```
comment mca 2405;
procedure eqilbr(a, n, em, d, int); value n; integer n; array a, em, d;
integer array int;
begin integer i, im, i1, p, q, j, t, count, exponent, ni;
    real c, r, eps, omega, factor;

    procedure move(k); value k; integer k;
    begin real di;
        ni:= q - p; t:= t + 1; if k ≠ i then
        begin ichcol(1, n, k, i, a); ichrow(1, n, k, i, a); di:= d[i];
            d[i]:= d[k]; d[k]:= di
        end
    end move;

    factor:= 1 / (2 × ln(2)); comment more generally: ln(base);
    eps:= em[0]; omega:= 1 / eps; t:= p:= 1; q:= ni:= i:= n;
    count:= (n + 1) × n : 2;
    for j:= 1 step 1 until n do
    begin d[j]:= 1; int[j]:= 0 end;
    for i:= if i < q then i + 1 else p while count > 0 ∧ ni > 0 do
    begin count:= count - 1; im:= i - 1; i1:= i + 1;
        c:= sqrt(tammat(p, im, i, i, a, a)+tammat(i1, q, i, i, a, a));
        r:= sqrt(mattam(p, im, i, i, a, a)+mattam(i1, q, i, i, a, a));
        if c × omega < r × eps then
        begin int[t]:= i; move(p); p:= p + 1 end
        else if r × omega < c × eps then
        begin int[t]:= - i; move(q); q:= q - 1 end
        else
        begin exponent:= ln(r / c) × factor;
            if abs(exponent) > 1 then
            begin ni:= q - p; c:= 2 ↑ exponent; r:= 1 / c;
                d[i]:= d[i] × c;
                for j:= 1 step 1 until im, i1 step 1 until n do
                begin a[j,i]:= a[j,i] × c; a[i,j]:= a[i,j] × r end
            end
            else ni:= ni - 1
        end
    end
end eqilbr;
```

Description mca 2405
eqilbr transforms the n-th order matrix given in array a[1:n, 1:n] into a similar equilibrated matrix. In array em[0:0], one must give the machine precision.
The equilibrated matrix is delivered in a, the main diagonal of the transforming diagonal matrix in array d[1:n], and information defining the possible interchanging of some rows and the corresponding columns in integer array int[1:n].
eqilbr uses tammat, mattam, ichcol and ichrow [3, chapter 20].

```
comment mca 2406;
procedure baklbr(n, n1, n2, d, int, vec); value n, n1, n2;
integer n, n1, n2; array d, vec; integer array int;
begin integer i, j, k, p, q;
    real di;
    p:= 1; q:= n;
    for i:= 1 step 1 until n do
    begin di:= d[i]; if di ≠ 1 then
        for j:= n1 step 1 until n2 do vec[i,j]:= vec[i,j] × di;
        k:= int[i];
        if k > 0 then p:= p + 1 else if k < 0 then q:= q - 1
    end;
    for i:= p - 1 + n - q step - 1 until 1 do
    begin k:= int[i]; if k > 0 then
        begin p:= p - 1; if k ≠ p then ichrow(n1, n2, k, p, vec) end
        else
        begin q:= q + 1; if -k ≠ q then ichrow(n1, n2, - k, q, vec)
        end
    end
end baklbr;
```

Description mca 2406
baklbr should be called after eqilbr and performs the corresponding back transformation on the columns of array vec[1:n, n1:n2].
The main diagonal of the transforming diagonal matrix and the information defining the possible interchanging of some rows and columns, as produced by eqilbr, must be given in array d[1:n] and integer array int[1:n].
The resulting vectors of the back transformation are written over the corresponding columns of vec.
baklbr uses ichrow [3, mca 2022].

Section 241 Single QR iteration

This section contains procedures for calculating eigenvalues and/or eigenvectors of real matrices having real eigenvalues only:
reaeigval calculates the eigenvalues, and reaeig1, reaeig2 and reaeig3 the eigenvalues and eigenvectors of a real matrix;
reavalqri calculates the eigenvalues of a real upper-Hessenberg matrix, and reaqri the eigenvectors as well;
reaveches calculates an eigenvector corresponding to a given real eigenvalue of a real upper-Hessenberg matrix;
reascl normalizes a given matrix of real eigenvectors.

The method used (in reavalqri and reaqri) for calculating the real eigenvalues of an upper-Hessenberg matrix H, is Francis' "single" QR iteration [21] [2, p 515 - 543].
In each step, the matrix H is replaced by the similar matrix Q'HQ, where Q is an orthogonal matrix chosen in such a way that, for some suitable "shift" s, Q'(H - sI) is an upper-triangular matrix, R (thus, H - sI = QR, which explains the name of the method).
The matrices Q and R could be calculated by means of Gram-Schmidt orthogonalisation [2, p. 242]; hence, it is obvious that Q has upper-Hessenberg form.
The similar matrix Q'HQ is again an upper-Hessenberg matrix (because R is upper-triangular and Q has upper-Hessenberg form).
If the given matrix has real eigenvalues only, then, in most cases, the sequence of iterates H converges to a (nearly) upper-triangular matrix, U, similar to the given matrix, so that the diagonal elements of U are (approximately) the required eigenvalues.
As soon as, for some k, the k-th element of the subdiagonal of an iterate H has an absolute value smaller than some tolerance, then this element is neglected and H is partitioned into 4 submatrices, H11, H12, H21, H22, consisting of the first k (H11, H12) or last n - k (H21, H22) rows of H and the first k (H11, H21) or last n - k (H12, H22) columns of H; H21 is the null matrix and H12 plays a role only for the calculation of eigenvectors (see p. 55); subsequently, the "principle" submatrices H11 and H22 are considered and handled separately; eigenvalues of submatrices of order 1 or 2 are calculated directly, so that the process is completed if successive partitionings have led to principle submatrices all of order 1 or 2.

In each step, the shift s is chosen as follows: let B denote the lower right 2 by 2 submatrix of the considered principle submatrix of H; then s equals that eigenvalue of B closest to its last main-diagonal element if the eigenvalues of B are real, and otherwise s equals the real part of the eigenvalues of B.
If the subdiagonal element of B converges to 0, then the convergence is quadratic for a simple eigenvalue of H and linear for a multiple one. However, convergence cannot be guaranteed for all cases, although no counter-example of a real matrix having only real eigenvalues is known to the authors; the iteration is therefore discontinued if a given maximum allowed number of iterations has been performed.

For the calculation of eigenvectors of an upper-Hessenberg matrix H, we have chosen alternative methods, viz. inverse iteration, and a "direct" method linking up with the QR iteration.
The procedure reaveches calculates an eigenvector of H corresponding to a given approximate real eigenvalue, lambda, by means of inverse iteration [2, p. 619 - 628] [4, AP 240] [22]. Starting from the initial vector, x, all of whose elements are 1, the linear system (H - lambda × I)y = x is solved iteratively (by means of Gaussian elimination with row interchanges), the solution y divided by its Euclidean norm replacing x each time. The iteration ends either if the Euclidean norm of the residue (H - lambda × I)x (this norm is calculated as the reciprocal of the Euclidean norm of y) is not larger than a given norm of H times the tolerance for the eigenvectors, or if the maximum allowed number of iterations has been performed.
If the tolerance for the eigenvectors is not too small then one or two iterations suffice in most cases.

To find in reaeigval, reaeig1 and reaeig2 the eigenvalues of a matrix M having real eigenvalues only, M is first equilibrated and transformed to a similar upper-Hessenberg matrix H (section 240), the eigenvalues are then calculated by calling reavalqri, and finally the calculated eigenvalues are sorted into monotonically nonincreasing order.
Furthermore, to find in reaeig1 and reaeig2 the eigenvectors, Wilkinson's device [2, p. 328 and 628] [9] [22] is first applied; i.e. approximate eigenvalues having a distance smaller than "machtol" (= infinity norm of M equilibrated times the machine precision) are slightly modified such that the distance between them equals machtol. (This device has the effect that, for reasonably conditioned matrices, a numerically independent set of eigenvectors is almost always obtained, since inverse iteration is very sensitive to small changes in the approximate values of closely clustered eigenvalues.)
Subsequently, the eigenvectors of H are calculated by calling reaveches; these vectors are then back-transformed to the corresponding eigenvectors of M (section 240), and normalized (by calling reascl) such that, in each eigenvector, an element of maximum absolute value equals 1.

The other, "direct", method for calculating the eigenvectors of an upper-Hessenberg matrix H is used by reaqri, and works as follows.
In each QR iteration step, the corresponding rotation is performed on some matrix X whose initial value is I; i.e. X is replaced by XQ in each step (in this process the submatrices H12 produced by the partitionings of H are also involved); thus, on completion of the QR iteration, X is the product of orthogonal matrices Q transforming the given matrix H into the similar upper-triangular matrix U produced by the QR iteration, i.e. HX = XU.
Subsequently, the eigenvectors, v, of U are calculated directly by solving the corresponding triangular system of linear equations.
If the distance between any two diagonal elements of U (which are approximate eigenvalues of H) is smaller than machtol, then they are slightly modified such that the distance between them equals machtol. This modification is necessary for preventing division by zero.

Finally, the eigenvectors v of U are replaced by the vectors Xv, which are the corresponding eigenvectors of H.
If H is not too ill-conditioned with respect to its eigenvalue problem, then this method yields numerically independent eigenvectors (sometimes better than inverse iteration; and is competitive with inverse iteration as to accuracy and computation time (see p. 43)).

To find (in reaeig3) the eigenvalues and eigenvectors of a matrix M having only real eigenvalues, M is first equilibrated and transformed to a similar upper-Hessenberg matrix H (section 240); the eigenvalues and eigenvectors of H are then calculated by calling reaqri, and finally the eigenvectors of H are back-transformed to the corresponding eigenvectors of M (section 240), and normalized (by calling reascl) such that, in each eigenvector, an element of maximum absolute value equals 1. The procedure reaeig3 does not sort the eigenvalues.

The procedures of this section, except reascl, and those of the next section, except comscl, use an auxiliary array em[0:9] (or a part of it), in which some data for controlling the iterations must be given and some by-products are delivered.
A survey of these data and by-products follows.

1) general
em[0] is the machine precision; must be given for all procedures;
em[1] is some norm of M or H; must be given for reavalqri, reaveches, reaqri, comvalqri and comveches; the other procedures deliver the infinity norm of M equilibrated.

2) for the QR iteration
em[2] and em[4] must be given for, and em[3] and em[5] are delivered by, all procedures, except reaveches and comveches.
em[2] is the relative tolerance for the QR iteration;
if the absolute value of some subdiagonal element is smaller than em[1] × em[2], then this element is neglected and the matrix is partitioned;
em[3] is the maximum absolute value of the subdiagonal elements neglected;
em[4] is the maximum allowed number of QR iterations;
em[5] is the number of QR iterations performed; the value em[4] + 1 is delivered if the QR iteration process is not completed within em[4] iterations.

3) for the inverse iteration
em[6] and em[8] must be given for, and em[7] and em[9] are delivered by, reaveches, reaeig1, reaeig2, comveches, comeig1 and comeig2.
em[6] is the tolerance for the eigenvectors; more precisely, the inverse iteration ends if the Euclidean norm of the residue vector is smaller than em[1] × em[6];
em[7] is the maximum Euclidean norm of the residue vectors of the calculated eigenvectors of H;
em[8] is the maximum number of inverse iterations allowed for the calculation of each eigenvector;
em[9] is the largest number of inverse iterations performed for the calculation of some eigenvector; the value em[8] + 1 is delivered, if the Euclidean norm of the residue for one or more eigenvectors remains larger than em[1] × em[6] during em[8] iterations; nevertheless the eigenvectors may then very well be useful – this should be judged from the value delivered in em[7] or from some other test.

The tolerances should satisfy em[0] $\leq$ em[2] < em[6].
For the X8, suitable values of the data to be given in em are
em[0] = $_{10}-12$, em[2] = $_{10}-12$, em[4] = 10 × n, em[6] = $_{10}-8$, em[8] = 5.

```
comment mca 2410;
integer procedure reavalqri(a, n, em, val); value n; integer n;
array a, em, val;
begin integer n1, i, i1, j, q, max, count;
    real det, w, shift, kappa, nu, mu, r, tol, delta, machtol, s;
    machtol:= em[0] × em[1]; tol:= em[1] × em[2]; max:= em[4];
    count:= 0; r:= 0;
in: n1:= n - 1;
    for i:= n, i - 1 while (if i ≥ 1 then abs(a[i + 1,i]) > tol else
    false) do q:= i; if q > 1 then
    begin if abs(a[q,q - 1]) > r then r:= abs(a[q,q - 1]) end;
    if q = n then
    begin val[n]:= a[n,n]; n:= n1 end
    else
    begin delta:= a[n,n] - a[n1,n1]; det:= a[n,n1] × a[n1,n];
        if abs(delta) < machtol then s:= sqrt(det) else
        begin w:= 2 / delta; s:= w × w × det + 1;
            s:= if s ≤ 0 then - delta × .5 else w × det / (sqrt(s)
            + 1)
        end;
        if q = n1 then
        begin val[n]:= a[n,n] + s; val[n1]:= a[n1,n1] - s; n:= n - 2
        end
        else
        begin count:= count + 1; if count > max then goto out;
            shift:= a[n,n] + s; if abs(delta) < tol then
            begin w:= a[n1,n1] - s;
                if abs(w) < abs(shift) then shift:= w
            end;
            a[q,q]:= a[q,q] - shift;
            for i:= q step 1 until n - 1 do
            begin i1:= i + 1; a[i1,i1]:= a[i1,i1] - shift;
                kappa:= sqrt(a[i,i] ↑ 2 + a[i1,i] ↑ 2);
                if i > q then
                begin a[i,i - 1]:= kappa × nu; w:= kappa × mu end
                else w:= kappa; mu:= a[i,i] / kappa;
                nu:= a[i1,i] / kappa; a[i,i]:= w;
                rotrow(i1, n, i, i1, a, mu, nu);
                rotcol(q, i, i, i1, a, mu, nu);
                a[i,i]:= a[i,i] + shift
            end;
            a[n,n - 1]:= a[n,n] × nu; a[n,n]:= a[n,n] × mu + shift
        end
    end;
    if n > 0 then goto in;
out: em[3]:= r; em[5]:= count; reavalqri:= n
end reavalqri;
```

Description mca 2410
reavalqri calculates the eigenvalues of the n-th order upper-Hessenberg matrix H given in array a[1:n, 1:n], provided that all eigenvalues of H are real.
In array em[0:5], the following data must be given (see also p. 56):
em[0]: the machine precision;
em[1]: norm of H;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of iterations.
The eigenvalues of H are delivered in array val[1:n].
Moreover, the Hessenberg part of a is altered;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of iterations performed.
Furthermore, reavalqri:= 0, provided that the process is completed within em[4] iterations; otherwise, reavalqri:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n - k elements of val are approximate eigenvalues of H.
reavalqri uses rotcol and rotrow [3, section 203].

```
comment mca 2411;
procedure reaveches(a, n, lambda, em, v); value n, lambda; integer n;
real lambda; array a, em, v;
begin integer i, i1, j, count, max;
    real m, r, norm, machtol, tol;
    boolean array p[1:n];
    norm:= em[1]; machtol:= em[0] × norm; tol:= em[6] × norm;
    max:= em[8]; a[1,1]:= a[1,1] - lambda;
gauss: for i:= 1 step 1 until n - 1 do
    begin i1:= i + 1; r:= a[i,i]; m:= a[i1,i];
        if abs(m) < machtol then m:= machtol; p[i]:= abs(m) ≤ abs(r);
        if p[i] then
        begin a[i1,i]:= m:= m / r;
            for j:= i1 step 1 until n do a[i1,j]:= (if j > i1 then
            a[i1,j] else a[i1,j] - lambda) - m × a[i,j]
        end
        else
        begin a[i,i]:= m; a[i1,i]:= m:= r / m;
            for j:= i1 step 1 until n do
            begin r:= (if j > i1 then a[i1,j] else a[i1,j] - lambda);
                a[i1,j]:= a[i,j] - m × r; a[i,j]:= r
            end
        end
    end gauss;
    if abs(a[n,n]) < machtol then a[n,n]:= machtol;
    for j:= 1 step 1 until n do v[j]:= 1; count:= 0;
forward: count:= count + 1; if count > max then goto out;
    for i:= 1 step 1 until n - 1 do
    begin i1:= i + 1;
        if p[i] then v[i1]:= v[i1] - a[i1,i] × v[i] else
        begin r:= v[i1]; v[i1]:= v[i] - a[i1,i] × r; v[i]:= r end
    end forward;
backward: for i:= n step - 1 until 1 do v[i]:= (v[i] - matvec(i
    + 1, n, i, a, v)) / a[i,i]; r:= 1 / sqrt(vecvec(1, n, 0, v, v));
    for j:= 1 step 1 until n do v[j]:= v[j] × r;
    if r > tol then goto forward;
out: em[7]:= r; em[9]:= count
end reaveches;
```

Description mca 2411
reaveches calculates the eigenvector corresponding to the real eigenvalue lambda of the n-th order upper-Hessenberg matrix H given in array a[1:n, 1:n].
In array em[0:9], the following data must be given (see also p. 56,57).
em[0]: the machine precision;
em[1]: a norm of H;
em[6]: the tolerance for the eigenvectors;
em[8]: the maximum allowed number of iterations.
The calculated eigenvector is delivered in array v[1:n].
Moreover, the Hessenberg part of a is altered;
em[7]:= the Euclidean norm, $||r||$, of the residue of the calculated eigenvector;
em[9]:= the number of iterations performed.
If, however, $||r||$ remains larger than em[1] × em[6] during em[8] iterations, then em[9]:= em[8] + 1.
reaveches uses vecvec and matvec [3, section 200].

```
comment mca 2412;
integer procedure reaeigval(a, n, em, val); value n; integer n;
array a, em, val;
begin integer i, j;
    real r;
    integer array int, int0[1:n];
    array d[1:n];
    eqilbr(a, n, em, d, int0); tfmreahes(a, n, em, int);
    j:= reaeigval:= reavalqri(a, n, em, val);
    for i:= j + 1 step 1 until n do
    for j:= i + 1 step 1 until n do
    begin if val[j] > val[i] then
        begin r:= val[i]; val[i]:= val[j]; val[j]:= r end
    end
end reaeigval;
```

Description mca 2412

reaeigval calculates the eigenvalues of the n-th order matrix M given in array a[1:n, 1:n], provided that all eigenvalues of M are real.
In array em[0:5], the elements with even subscript must be given (see also p. 56).
em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of iterations.
The eigenvalues of M are delivered in array val[1:n] in monotonically nonincreasing order.
Moreover, the elements of a are altered;
em[1]:= the infinity norm of M equilibrated;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of iterations performed.
Furthermore, reaeigval:= 0, provided that the process is completed within em[4] iterations; otherwise, reaeigval:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n − k elements of val are approximate eigenvalues of M.
reaeigval uses eqilbr, tfmreahes (section 240) and reavalqri (mca 2410) and, indirectly, also matvec, matmat, tammat, mattam, ichcol, ichrow, rotcol and rotrow [3, chapter 20].

```
comment mca 2413;
procedure reascl(a, n, n1, n2); value n, n1, n2; integer n, n1, n2;
array a;
begin integer i, j;
    real s;
    for j:= n1 step 1 until n2 do
    begin s:= 0;
        for i:= 1 step 1 until n do
        begin if abs(a[i,j]) > abs(s) then s:= a[i,j] end;
        if s ≠ 0 then
        for i:= 1 step 1 until n do a[i,j]:= a[i,j] / s
    end
end reascl;


comment mca 2414;
integer procedure reaeig1(a, n, em, val, vec); value n; integer n;
array a, em, val, vec;
begin integer i, k, max, j, l;
    real residu, r, machtol;
    array d, v[1:n], b[1:n,1:n];
    integer array int, int0[1:n];
    residu:= 0; max:= 0; eqilbr(a, n, em, d, int0);
    tfmreahes(a, n, em, int);
    for i:= 1 step 1 until n do
    for j:= (if i = 1 then 1 else i - 1) step 1 until n do b[i,j]:=
    a[i,j]; k:= reaeig1:= reavalqri(b, n, em, val);
    for i:= k + 1 step 1 until n do
    for j:= i + 1 step 1 until n do
    begin if val[j] > val[i] then
        begin r:= val[i]; val[i]:= val[j]; val[j]:= r end
    end;
    machtol:= em[0] × em[1];
    for l:= k + 1 step 1 until n do
    begin if l > 1 then
        begin if val[l - 1] - val[l] < machtol then val[l]:= val[l -
            1] - machtol
        end;
        for i:= 1 step 1 until n do
        for j:= (if i = 1 then 1 else i - 1) step 1 until n do
        b[i,j]:= a[i,j]; reaveches(b, n, val[l], em, v);
        if em[7] > residu then residu:= em[7];
        if em[9] > max then max:= em[9];
        for j:= 1 step 1 until n do vec[j,l]:= v[j]
    end;
    em[7]:= residu; em[9]:= max; bakreahes2(a, n, k + 1, n, int, vec);
    baklbr(n, k + 1, n, d, int0, vec); reascl(vec, n, k + 1, n)
end reaeig1;
```

Description mca 2413
reascl normalizes the (non-null) columns of array a[1:n, n1:n2] in such a way that, in each column, an element of maximum absolute value equals 1. The normalized vectors are written over the corresponding columns of a.

Description mca 2414
reaeig1 calculates the eigenvalues, provided that they are all real, and the eigenvectors of the n-th order matrix M given in array a[1:n, 1:n].
In array em[0:9], the elements with even subscript must be given (see also p. 56, 57), viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of QR iterations;
em[6]: the tolerance for the eigenvectors;
em[8]: the maximum number of inverse iterations allowed for the calculation of each eigenvector.
The eigenvalues of M are delivered in array val[1:n] in monotonically decreasing order, with the corresponding eigenvectors in the columns of array vec[1:n, 1:n].
Moreover, the elements of a are altered;
em[1]:= the infinity norm of M equilibrated;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of QR iterations performed;
em[7]:= the maximum Euclidean norm of the residues of the calculated eigenvectors (of the transformed matrix);
em[9]:= the largest number of inverse iterations performed for the calculation of some eigenvector.
Furthermore, reaeig1:= 0, provided that the QR iteration process is completed within em[4] iterations; otherwise, reaeig1:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n − k elements of val and columns of vec are approximate eigenvalues and eigenvectors of M; similarly, if, for some calculated eigenvector, the Euclidean norm of the residue remains larger than em[1] × em[6], then em[9]:= em[8] + 1.
reaeig1 uses eqilbr, tfmreahes, bakreahes2, baklbr (section 240), reavalqri, reaveches and reascl (this section), and, indirectly, also vecvec, matvec, tamvec, matmat, tammat, mattam, ichcol, ichrow, rotcol, and rotrow [3, chapter 20].

```
comment mca 2415;
integer procedure reaeig2(a, n, em, val); value n; integer n;
array a, em, val;
begin integer i, k, max, j, l;
    real residu, r, machtol;
    array d, v[1:n];
    integer array int, int0[1:n];
    residu:= 0; max:= 0; eqilbr(a, n, em, d, int0);
    tfmreahes(a, n, em, int); TODRUM(a, 2 × n × n);
    l:= reaeig2:= reavalqri(a, n, em, val);
    for i:= l + 1 step 1 until n do
    for j:= i + 1 step 1 until n do
    begin if val[j] > val[i] then
        begin r:= val[i]; val[i]:= val[j]; val[j]:= r end
    end;
    machtol:= em[0] × em[1];
    for k:= l + 1 step 1 until n do
    begin FROMDRUM(a, 2 × n × n); if k > 1 then
        begin if val[k - 1] - val[k] < machtol then val[k]:= val[k -
            1] - machtol
        end;
        reaveches(a, n, val[k], em, v);
        if em[7] > residu then residu:= em[7];
        if em[9] > max then max:= em[9]; bakreahes1(a, n, int, v);
        TODRUM(v, (k - 1) × n × 2)
    end;
    em[7]:= residu; em[9]:= max; FROMDRUM(a, 0);
    baklbr(n, l + 1, n, d, int0, a); reascl(a, n, l + 1, n)
end reaeig2;
```

Description mca 2415

reaeig2 calculates the eigenvalues, provided that they are all real, and the eigenvectors of the n-th order matrix M given in array a[1:n, 1:n].

In array em[0:9] the elements with even subscript must be given as for reaeig1.

The eigenvalues of M are delivered in array val[1:n] in monotonically decreasing order with the corresponding eigenvectors in the columns of a. The eigenvectors are also delivered in the first n ↑ 2 real number locations of the backing storage, and the next n ↑ 2 real number locations of the backing storage are altered.

Moreover, in the elements of em with odd subscript, the same results are delivered as by reaeig1.

Furthermore, reaeig2:= 0, provided that the QR iteration process is completed within em[4] iterations; otherwise, reaeig2:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n − k elements of val and columns of a are approximate eigenvalues and eigenvectors of M; similarly, if, for some calculated eigenvector, the Euclidean norm of the residue remains larger than em[1] × em[6], then em[9]:= em[8] + 1.

reaeig2 uses eqilbr, tfmreahes, bakreahes1, baklbr (section 240), reavalqri, reaveches, reascl (this section), the X8-code procedures TODRUM and FROMDRUM (see introduction), and, indirectly, also vecvec, matvec, matmat, tammat, mattam, ichcol, ichrow, rotcol and rotrow [3, chapter 20].

```
comment mca 2416;
integer procedure reaqri(a, n, em, val, vec); value n; integer n;
array a, em, val, vec;
begin integer m1, i, i1, m, j, q, max, count;
    real w, shift, kappa, nu, mu, r, tol, s, machtol, elmax, t,
    delta, det;
    array tf[1:n];
    machtol:= em[0] × em[1]; tol:= em[1] × em[2]; max:= em[4];
    count:= 0; elmax:= 0; m:= n;
    for i:= 1 step 1 until n do
    begin vec[i,i]:= 1;
        for j:= i + 1 step 1 until n do vec[i,j]:= vec[j,i]:= 0
    end;
in: m1:= m - 1;
    for i:= m, i - 1 while (if i ≥ 1 then abs(a[i + 1,i]) > tol else
    false) do q:= i; if q > 1 then
    begin if abs(a[q,q - 1]) > elmax then elmax:= abs(a[q,q - 1])
    end;
    if q = m then
    begin val[m]:= a[m,m]; m:= m1 end
    else
    begin delta:= a[m,m] - a[m1,m1]; det:= a[m,m1] × a[m1,m];
        if abs(delta) < machtol then s:= sqrt(det) else
        begin w:= 2 / delta; s:= w × w × det + 1;
            s:= if s ≤ 0 then - delta × .5 else w × det / (sqrt(s)
            + 1)
        end;
        if q = m1 then
        begin a[m,m]:= val[m]:= a[m,m] + s;
            a[q,q]:= val[q]:= a[q,q] - s;
            t:= if abs(s) < machtol then (s + delta) / a[m,q] else
            a[q,m] / s; r:= sqrt(t × t + 1); nu:= 1 / r;
            mu:= - t × nu; a[q,m]:= a[q,m] - a[m,q];
            rotrow(q + 2, n, q, m, a, mu, nu);
            rotcol(1, q - 1, q, m, a, mu, nu);
            rotcol(1, n, q, m, vec, mu, nu); m:= m - 2
        end
        else
        begin count:= count + 1; if count > max then goto end;
            shift:= a[m,m] + s; if abs(delta) < tol then
            begin w:= a[m1,m1] - s;
                if abs(w) < abs(shift) then shift:= w
            end;
            a[q,q]:= a[q,q] - shift;
```

Description mca 2416

reaqri calculates the eigenvalues, provided that they are all real, and the eigenvectors of the n-th order upper-Hessenberg matrix H given in array a[1:n, 1:n].

In array em[0:5], the following data must be given (see also p. 56).

em[0]: the machine precision;
em[1]: a norm of H;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of QR iterations.

The eigenvalues of H are delivered in array val[1:n], with the corresponding eigenvectors in the columns of array vec[1:n, 1:n].

Moreover, the elements of the upper-Hessenberg part of a are altered;

em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of QR iterations performed.

Furthermore, reaqri:= 0, provided that the process is completed within em[4] iterations; otherwise, reaqri:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, only the last n - k elements of val are approximate eigenvalues of H, and no useful eigenvectors are delivered.

reaqri uses matvec, rotcol and rotrow [3, chapter 20].

```
            for i:= q step 1 until m1 do
            begin i1:= i + 1; a[i1,i1]:= a[i1,i1] - shift;
                kappa:= sqrt(a[i,i] ↑ 2 + a[i1,i] ↑ 2);
                if i > q then
                begin a[i,i - 1]:= kappa × nu; w:= kappa × mu end
                else w:= kappa; mu:= a[i,i] / kappa;
                nu:= a[i1,i] / kappa; a[i,i]:= w;
                rotrow(i1, n, i, i1, a, mu, nu);
                rotcol(1, i, i, i1, a, mu, nu);
                a[i,i]:= a[i,i] + shift;
                rotcol(1, n, i, i1, vec, mu, nu)
            end;
            a[m,m1]:= a[m,m] × nu; a[m,m]:= a[m,m] × mu + shift
        end
    end;
    if m > 0 then goto in;
    for j:= n step - 1 until 2 do
    begin tf[j]:= 1; t:= a[j,j];
        for i:= j - 1 step - 1 until 1 do
        begin delta:= t - a[i,i];
            tf[i]:= matvec(i + 1, j, i, a, tf) / (if abs(delta) <
            machtol then machtol else delta)
        end;
        for i:= 1 step 1 until n do vec[i,j]:= matvec(1, j, i, vec,
        tf)
    end;
end: em[3]:= elmax; em[5]:= count; reaqri:= m
end reaqri;
```

```
comment mca 2417;
integer procedure reaeig3(a, n, em, val, vec); value n; integer n;
array a, em, val, vec;
begin integer i;
    real s;
    integer array int, int0[1:n];
    array d[1:n];
    eqilbr(a, n, em, d, int0); tfmreahes(a, n, em, int);
    i:= reaeig3:= reaqri(a, n, em, val, vec); if i = 0 then
    begin bakreahes2(a, n, 1, n, int, vec);
        baklbr(n, 1, n, d, int0, vec); reascl(vec, n, 1, n)
    end
end reaeig3;
```

Description mca 2417
reaeig3 calculates the eigenvalues, provided that they are all real, and the eigenvectors of the n-th order matrix M given in array a[1:n, 1:n].
In array em[0:5], the elements with even subscript must be given (see also p. 56), viz.
em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of QR iterations.
The eigenvalues of M are delivered in array val[1:n], with the corresponding eigenvectors in the columns of array vec[1:n, 1:n].
Moreover, the elements of a are altered;
em[1]:= the infinity norm of M equilibrated;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of QR iterations performed.
Furthermore, reaeig3:= 0, provided that the QR iteration process is completed within em[4] iterations; otherwise, reaeig3:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, only the last n - k elements of val are approximate eigenvalues of M, and no useful eigenvectors are delivered.
reaeig3 uses eqilbr, tfmreahes, bakreahes2, baklbr (section 240), reaqri and reascl (this section), and, indirectly, also matvec, tamvec, matmat, tammat, mattam, ichcol, ichrow, rotcol and rotrow [3, chapter 20].

Section 242 Double QR iteration

This section contains procedures for calculating real or complex eigenvalues and/or eigenvectors of real matrices:
comeigval calculates the eigenvalues, and comeig1 and comeig2 the eigenvalues and eigenvectors of a real matrix;
comvalqri calculates the eigenvalues of a real upper-Hessenberg matrix; comveches calculates the eigenvector corresponding to a given complex eigenvalue of a real upper-Hessenberg matrix;
comscl normalizes a given matrix of real or complex eigenvectors.

The method used in comvalqri for calculating the eigenvalues of a real upper-Hessenberg matrix H, is Francis' "double" QR iteration [21] [2, p. 528-537].
A double QR iteration step is (mathematically) equivalent to two successive single iteration steps (see section 241) in which the shifts are either both real or each others complex conjugate; thus, a double QR iteration step again yields a real upper-Hessenberg matrix as next iterate.
In each double step, the shifts are chosen approximately equal to the eigenvalues of the lower right 2 by 2 submatrix of the considered principle submatrix of H.
The shifts are chosen not exactly equal to these eigenvalues in an attempt to avoid nonconvergence of the iteration. In fact, the shifts are equal to the eigenvalues mentioned plus the square root of the product of the last two subdiagonal elements of H times the machine precision.
Hence, one can influence the process by his choice of the "machine precision" given in em[0].

In the same way as in the single QR iteration, an iterate H is partitioned into 4 submatrices if, for some k, the absolute value of the k-th element of its subdiagonal does not exceed some tolerance; moreover, a weaker criterion due to Francis [21] for partitioning the matrix is applied, which criterion is especially effective if two adjacent subdiagonal elements become small.
Subsequently, the two principle submatrices produced by the partitioning are considered and handled separately; the process is completed when the successive partitionings have led to principle submatrices all of order 1 or 2 (cf. section 241).
In almost all cases, the double QR iteration with the choice of the shifts as mentioned above converges; i.e. the last element of the subdiagonal of the considered principle submatrix of H converges to 0, the convergence being quadratic for simple eigenvalues, and linear for multiple ones.
However, convergence does not always occur; counter-examples are the n-th order permutation matrices ($n > 2$), whose first-subdiagonal elements and (1, n)-th element are 1, and whose other elements are 0; these matrices are invariant with respect to (single or double) QR iteration (the shifts being 0) [2, p. 521].
The iteration (in comvalqri) is discontinued if a given maximum allowed number of iterations has been performed.

In comveches, an eigenvector corresponding to a given approximate complex eigenvalue, kappa (= lambda + i × mu), of a real upper-Hessenberg matrix H is calculated by means of inverse iteration [2, p. 629-633] [22]. Starting from the initial vector, x, having all elements equal to 1, the linear system (H - kappa × I)y = x is solved iteratively (by means of Gaussian elimination with row interchanges), and the solution y divided by its Euclidean norm replaces x each time; the computation is performed using complex numbers where necessary.
The Gaussian elimination yields a complex upper-triangular matrix, U, with real main diagonal; the real parts of the elements are stored in the upper triangle, and the imaginary parts in the strict-lower triangle of the array in which H was given. If the i-th and (i + 1)-th row were interchanged in the i-th Gaussian elimination step, then the i-th row of U is real thereby making this step and the corresponding step of the back substitution twice as fast as otherwise.
The iteration ends either if the Euclidean norm of the residue (H - kappa × I)x (this norm is calculated as the reciprocal of the Euclidean norm of y) is not greater then a given norm of H times the tolerance for the eigenvectors, or if the maximum allowed number of iterations has been performed. If the tolerance for the eigenvectors is not too small, then one or two iterations suffice in most cases.
Our method is the first of the 4 alternatives mentioned in [2, p. 629-630]; the second alternative requires much more time and space, the third does not converge to an eigenvector of H, and the fourth alternative, used in [22], requires about the same computation time and twice as much memory space.

To find (in comeigval, comeig1 and comeig2) the eigenvalues of a real matrix M, M is first equilibrated and transformed to a similar real upper-Hessenberg matrix H (section 240), and then the eigenvalues are calculated by calling comvalqri.
Furthermore, to find (in comeig1 and comeig2) the eigenvectors, Wilkinson's device [2, p. 328 and 628] [9] [22] is first applied; i. e. approximate eigenvalues having a distance smaller than some tolerance are slightly modified such that the distance between them equals that tolerance. (This device has the effect that a numerically independent set of eigenvectors is almost always obtained, provided that the matrix is not too ill-conditioned with respect to its eigenvalue problem, since inverse iteration is very sensitive to small changes in the approximate values of closely clustered eigenvalues.)
Subsequently, the eigenvectors of H are calculated by calling reaveches for the real eigenvalues of H, and comveches for the others; these vectors are then back-transformed to the corresponding eigenvectors of M (section 240; note that the real and imaginary parts of a complex eigenvector are each back-transformed in the same way as a real eigenvector);
finally, the eigenvectors of M are normalized (by calling comscl) such that, in each eigenvector, an element of maximum modulus equals 1.
The procedures of this section, except comscl, use an auxiliary array em[0:9] (or a part of it) in which some data for controlling the iterations must be given and some by-products are delivered.
A survey of these data and by-products is given in section 241(p. 56).

```
comment mca 2420;
integer procedure comvalqri(a, n, em, re, im); value n; integer n;
array a, em, re, im;
begin integer i, j, p, q, max, count, n1, p1, p2, imin1, i1, i2, i3;
    real disc, sigma, rho, g1, g2, g3, psi1, psi2, aa, e, k, s, norm,
    machtol2, tol, w;
    boolean b;
    norm:= em[1]; machtol2:= (em[0] × norm) ↑ 2; tol:= em[2] × norm;
    max:= em[4]; count:= 0; w:= 0;
in: for i:= n, i - 1 while (if i > 1 then abs(a[i + 1,i]) > tol else
    false) do q:= i; if q > 1 then
    begin if abs(a[q,q - 1]) > w then w:= abs(a[q,q - 1]) end;
    if q > n - 1 then
    begin n1:= n - 1; if q = n then
        begin re[n]:= a[n,n]; im[n]:= 0; n:= n1 end
        else
        begin sigma:= a[n,n] - a[n1,n1]; rho:= - a[n,n1] × a[n1,n];
            disc:= sigma ↑ 2 - 4 × rho; if disc > 0 then
            begin disc:= sqrt(disc);
                s:= - 2 × rho / (sigma + (if sigma ≥ 0 then disc
                else - disc)); re[n]:= a[n,n] + s;
                re[n1]:= a[n1,n1] - s; im[n]:= im[n1]:= 0
            end
            else
            begin re[n]:= re[n1]:= (a[n1,n1] + a[n,n]) / 2;
                im[n1]:= sqrt( - disc) / 2; im[n]:= - im[n1]
            end;
            n:= n - 2
        end
    end
    else
    begin count:= count + 1; if count > max then goto out; n1:= n - 1;
        sigma:= a[n,n] + a[n1,n1] + sqrt(abs(a[n1,n - 2] × a[n,n1])
        × em[0]); rho:= a[n,n] × a[n1,n1] - a[n,n1] × a[n1,n];
        for i:= n - 1, i - 1 while (if i - 1 > q then abs(a[i,i - 1]
        × a[i1,i] × (abs(a[i,i] + a[i1,i1] - sigma) + abs(a[i +
        2,i1]))) > abs(a[i,i] × ((a[i,i] - sigma) + a[i,i1] × a[i1,i]
        + rho)) × tol else false) do p1:= i1:= i; p:= p1 - 1;
        p2:= p + 2;
        for i:= p step 1 until n - 1 do
        begin imin1:= i - 1; i1:= i + 1; i2:= i + 2; if i = p then
            begin g1:= a[p,p] × (a[p,p] - sigma) + a[p,p1] × a[p1,p]
                + rho; g2:= a[p1,p] × (a[p,p] + a[p1,p1] - sigma);
                if p1 < n1 then
                begin g3:= a[p1,p] × a[p2,p1]; a[p2,p]:= 0 end
                else g3:= 0
            end
            else
            begin g1:= a[i,imin1]; g2:= a[i1,imin1];
                g3:= if i2 ≤ n then a[i2,imin1] else 0
            end;
```

Description mca 2420

comvalqri calculates the eigenvalues of the n–th order upper–Hessenberg matrix H given in array a[1:n, 1:n].
In array em[0:5], the following data must be given (see also p. 56).
em[0]: the machine precision;
em[1]: a norm of H;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of QR iterations.
The real and imaginary parts of the eigenvalues of H are delivered in array re, im[1:n], the members of each nonreal complex conjugate pair being consecutive.
Moreover, the elements of a are altered;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of iterations performed.
Furthermore, comvalqri:= 0, provided that the process is completed within em[4] iterations; otherwise, comvalqri:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n – k elements of re and im contain approximate eigenvalues of H.

```
            k:= if g1 > 0 then sqrt(g1 ↑ 2 + g2 ↑ 2 + g3 ↑ 2) else -
            sqrt(g1 ↑ 2 + g2 ↑ 2 + g3 ↑ 2); b:= abs(k) > machtol2;
            aa:= if b then g1 / k + 1 else 2;
            psi1:= if b then g2 / (g1 + k) else 0;
            psi2:= if b then g3 / (g1 + k) else 0;
            if i ≠ q then a[i,imin1]:= if i = p then - a[i,imin1]
            else - k;
            for j:= i step 1 until n do
            begin e:= aa × (a[i,j] + psi1 × a[i1,j] + (if i2 ≤ n
                then psi2 × a[i2,j] else 0));
                a[i,j]:= a[i,j] - e; a[i1,j]:= a[i1,j] - psi1 × e;
                if i2 ≤ n then a[i2,j]:= a[i2,j] - psi2 × e
            end;
            for j:= q step 1 until (if i2 ≤ n then i2 else n) do
            begin e:= aa × (a[j,i] + psi1 × a[j,i1] + (if i2 ≤ n
                then psi2 × a[j,i2] else 0));
                a[j,i]:= a[j,i] - e; a[j,i1]:= a[j,i1] - psi1 × e;
                if i2 ≤ n then a[j,i2]:= a[j,i2] - psi2 × e
            end;
            if i2 ≤ n1 then
            begin i3:= i + 3; e:= aa × psi2 × a[i3,i2]; a[i3,i]:= - e;
                a[i3,i1]:= - psi1 × e;
                a[i3,i2]:= a[i3,i2] - psi2 × e
            end
        end
    end;
    if n > 0 then goto in;
out: em[3]:= w; em[5]:= count; comvalqri:= n
end comvalqri;
```

```
comment mca 2421;
procedure comveches(a, n, lambda, mu, em, u, v); value n, lambda, mu;
integer n; real lambda, mu; array a, em, u, v;
begin integer i, i1, j, count, max;
    real aa, bb, d, m, r, s, w, x, y, norm, machtol, tol;
    array g, f[1:n];
    boolean array p[1:n];
    norm:= em[1]; machtol:= em[0] × norm; tol:= em[6] × norm;
    max:= em[8];
    for i:= 2 step 1 until n do
    begin f[i - 1]:= a[i,i - 1]; a[i,1]:= 0 end;
    aa:= a[1,1] - lambda; bb:= - mu;
    for i:= 1 step 1 until n - 1 do
    begin i1:= i + 1; m:= f[i]; if abs(m) < machtol then m:= machtol;
        a[i,i]:= m; d:= aa ↑ 2 + bb ↑ 2; p[i]:= abs(m) < sqrt(d);
        if p[i] then
        begin comment a[i,j]×factor and a[i1,j]←a[i,j];
            f[i]:= r:= m × aa / d; g[i]:= s:= - m × bb / d;
            w:= a[i1,i]; x:= a[i,i1]; a[i1,i]:= y:= x × s + w × r;
            a[i,i1]:= x:= x × r - w × s; aa:= a[i1,i1] - lambda - x;
            bb:= - mu - y;
            for j:= i + 2 step 1 until n do
            begin w:= a[j,i1]; x:= a[i,j]; a[j,i]:= y:= x × s + w × r;
                a[i,j]:= x:= x × r - w × s; a[j,i1]:= - y;
                a[i1,j]:= a[i1,j] - x
            end
        end
        else
        begin comment interchange a[i1,j] and a[i,j]←a[i1,j]×factor;
            f[i]:= r:= aa / m; g[i]:= s:= bb / m;
            w:= a[i1,i1] - lambda; aa:= a[i,i1] - r × w - s × mu;
            a[i,i1]:= w; bb:= a[i1,i] - s × w + r × mu;
            a[i1,i]:= - mu;
            for j:= i + 2 step 1 until n do
            begin w:= a[i1,j]; a[i1,j]:= a[i,j] - r × w; a[i,j]:= w;
                a[j,i1]:= a[j,i] - s × w; a[j,i]:= 0
            end
        end
    end;
```

Description mca 2421
comveches calculates the eigenvector corresponding to the complex eigenvalue lambda + i × mu of the n-th order upper-Hessenberg matrix H given in array a[1:n, 1:n].
In array em[0:9] the following data must be given (see also p. 56,57).
em[0]: the machine precision;
em[1]: a norm of H;
em[6]: the tolerance for the eigenvectors;
em[8]: the maximum allowed number of iterations.
The real and imaginary parts of the calculated eigenvector are delivered in array u, v[1:n].
Moreover, the elements of a are altered;
em[7]:= the Euclidean norm, ||r||, of the residue of the calculated eigenvector;
em[9]:= the number of iterations performed.
If, however, ||r|| remains larger than em[1] × em[6] during em[8] iterations, then em[9]:= em[8] + 1.
comveches uses vecvec, matvec and tamvec [3, section 200].

```
    p[n]:= true; d:= aa ↑ 2 + bb ↑ 2; if d < machtol ↑ 2 then
    begin aa:= machtol; bb:= 0; d:= machtol ↑ 2 end;
    a[n,n]:= d; f[n]:= aa; g[n]:= - bb;
    for i:= 1 step 1 until n do
    begin u[i]:= 1; v[i]:= 0 end;
    count:= 0;
forward: if count > max then goto outm;
    for i:= 1 step 1 until n do
    begin if p[i] then
        begin w:= v[i]; v[i]:= g[i] × u[i] + f[i] × w;
            u[i]:= f[i] × u[i] - g[i] × w; if i < n then
            begin v[i + 1]:= v[i + 1] - v[i];
                u[i + 1]:= u[i + 1] - u[i]
            end
        end
        else
        begin aa:= u[i + 1]; bb:= v[i + 1];
            u[i + 1]:= u[i] - (f[i] × aa - g[i] × bb); u[i]:= aa;
            v[i + 1]:= v[i] - (g[i] × aa + f[i] × bb); v[i]:= bb
        end
    end forward;
backward: for i:= n step - 1 until 1 do
    begin i1:= i + 1;
        u[i]:= (u[i] - matvec(i1, n, i, a, u) + (if p[i] then
        tamvec(i1, n, i, a, v) else a[i1,i] × v[i1])) / a[i,i];
        v[i]:= (v[i] - matvec(i1, n, i, a, v) - (if p[i] then
        tamvec(i1, n, i, a, u) else a[i1,i] × u[i1])) / a[i,i]
    end backward;
normalise: w:= 1 / sqrt(vecvec(1, n, 0, u, u) + vecvec(1, n, 0,
    v, v));
    for j:= 1 step 1 until n do
    begin u[j]:= u[j] × w; v[j]:= v[j] × w end;
    count:= count + 1; if w > tol then goto forward;
outm: em[7]:= w; em[9]:= count
end comveches;
```

```
comment mca 2422;
integer procedure comeigval(a, n, em, re, im); value n; integer n;
array a, em, re, im;
begin integer array int, int0[1:n];
    array d[1:n];
    eqilbr(a, n, em, d, int0); tfmreahes(a, n, em, int);
    comeigval:= comvalqri(a, n, em, re, im)
end comeigval;


comment mca 2423;
procedure comscl(a, n, n1, n2, im); value n, n1, n2; integer n, n1, n2;
array a, im;
begin integer i, j, k;
    real s, u, v, w;
    for j:= n1 step 1 until n2 do
    begin s:= 0; if im[j] ≠ 0 then
        begin for i:= 1 step 1 until n do
            begin u:= a[i,j] ↑ 2 + a[i,j + 1] ↑ 2; if u > s then
                begin s:= u; k:= i end
            end;
            if s ≠ 0 then
            begin v:= a[k,j] / s; w:= - a[k,j + 1] / s;
                for i:= 1 step 1 until n do
                begin u:= a[i,j]; s:= a[i,j + 1];
                    a[i,j]:= u × v - s × w;
                    a[i,j + 1]:= u × w + s × v
                end
            end;
            j:= j + 1
        end
        else
        begin for i:= 1 step 1 until n do
            begin if abs(a[i,j]) > abs(s) then s:= a[i,j] end;
            if s ≠ 0 then
            for i:= 1 step 1 until n do a[i,j]:= a[i,j] / s
        end
    end
end comscl;
```

Description mca 2422
comeigval calculates the eigenvalues of the n-th order matrix M given in array a[1:n, 1:n].
In array em[0:5], the elements with even subscript must be given (see also p. 56).
em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of QR iterations.
The real and imaginary parts of the eigenvalues of M are delivered in array re, im[1:n], the members of each nonreal complex conjugate pair being consecutive.
Moreover, the elements of a are altered;
em[1]:= the infinity norm of M equilibrated;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of QR iterations performed.
Furthermore, comeigval:= 0, provided that the process is completed within em[4] iterations; otherwise, comeigval:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n - k elements of re and im contain approximate eigenvalues of M.
comeigval uses eqilbr, tfmreahes (section 240), comvalqri (mca 2420) and, indirectly, also matvec, matmat, tammat, mattam, ichcol and ichrow [3, chapter 20].

Description mca 2423
comscl normalizes the eigenvectors corresponding to the complex eigenvalues whose imaginary parts are given in array im[n1:n2].
The corresponding eigenvectors must be given in the columns of array a[1:n, n1:n2] as follows:
each real eigenvector must be given in a column whose corresponding element of im equals 0;
the real and imaginary part of each complex eigenvector must be given in consecutive columns whose corresponding elements of im are not equal to 0.
The eigenvectors are normalized in such a way that, in each eigenvector, an element of maximum modulus equals 1.
The normalized eigenvectors are written over the corresponding given eigenvectors.

```
comment mca 2424;
integer procedure comeig1(a, n, em, re, im, vec); value n; integer n;
array a, em, re, im, vec;
begin integer i, j, k, pj, itt;
    real x, y, max, neps;
    array ab[1:n,1:n], d, u, v[1:n];
    integer array int, int0[1:n];

    procedure transfer;
    begin integer i, j;
        for i:= 1 step 1 until n do
        for j:= (if i = 1 then 1 else i - 1) step 1 until n do
        ab[i,j]:= a[i,j]
    end transfer;

    eqilbr(a, n, em, d, int0); tfmreahes(a, n, em, int); transfer;
    k:= comeig1:= comvalqri(ab, n, em, re, im); neps:= em[0] × em[1];
    max:= 0; itt:= 0;
    for i:= k + 1 step 1 until n do
    begin x:= re[i]; y:= im[i]; pj:= 0;
    again: for j:= k + 1 step 1 until i - 1 do
        begin if ((x - re[j]) ↑ 2 + (y - im[j]) ↑ 2 ≤ neps ↑ 2) then
            begin if pj = j then neps:= em[2] × em[1] else pj:= j;
                x:= x + 2 × neps; goto again
            end
        end;
        re[i]:= x; transfer; if y ≠ 0 then
        begin comveches(ab, n, re[i], im[i], em, u, v);
            for j:= 1 step 1 until n do vec[j,i]:= u[j]; i:= i + 1;
            re[i]:= x
        end
        else
        begin reaveches(ab, n, x, em, v) end;
        for j:= 1 step 1 until n do vec[j,i]:= v[j];
        if em[7] > max then max:= em[7];
        itt:= if itt > em[9] then itt else em[9]
    end;
    em[7]:= max; em[9]:= itt; bakreahes2(a, n, k + 1, n, int, vec);
    baklbr(n, k + 1, n, d, int0, vec); comscl(vec, n, k + 1, n, im)
end comeig1;
```

Description mca 2424

comeig1 calculates the eigenvalues and eigenvectors of the n-th order matrix M given in array a[1:n, 1:n].

In array em[0:9], the elements with even subscript must be given (see also p. 56,57), viz.

em[0]: the machine precision;
em[2]: the relative tolerance for the QR iteration;
em[4]: the maximum allowed number of QR iterations;
em[6]: the tolerance for the eigenvectors;
em[8]: the maximum number of iterations allowed for the calculation of each eigenvector.

The real and imaginary parts of the eigenvalues are delivered in array re, im[1:n], the members of each nonreal complex conjugate pair being consecutive; the eigenvectors are delivered in the columns of array vec[1:n, 1:n], an eigenvector corresponding to a real eigenvalue being in the corresponding column, and the real and imaginary part of an eigenvector corresponding to the first member of a nonreal complex conjugate pair being in the two consecutive columns corresponding to this pair. (The eigenvectors corresponding to the second members of nonreal complex conjugate pairs are not delivered, since they are simply the complex conjugate of those corresponding to the first members of such pairs.)

Moreover, the elements of a are altered;

em[1]:= the infinity norm of M equilibrated;
em[3]:= the maximum absolute value of the subdiagonal elements neglected;
em[5]:= the number of QR iterations performed;
em[7]:= the maximum Euclidean norm of the residues of the calculated eigenvectors (of the transformed matrix);
em[9]:= the largest number of inverse iterations performed for the calculation of some eigenvector.

Furthermore, comeig1:= 0, provided that the QR iteration process is completed within em[4] iterations; otherwise, comeig1:= the number, k, of eigenvalues not calculated, em[5]:= em[4] + 1, and only the last n − k elements of re and im, and the last n − k columns of vec contain approximate eigenvalues and eigenvectors of M; similarly, if, for some calculated eigenvector, the Euclidean norm of the residue remains greater than em[1] × em[6] during em[8] iterations, then em[9]:= em[8] + 1.

comeig1 uses eqilbr, tfmreahes, bakreahes2 and baklbr (section 240), reaveches (mca 2411), comvalqri, comveches and comscl (this section), and, indirectly, also vecvec, matvec, tamvec, matmat, tammat, mattam, ichcol and ichrow [3, chapter 20].

```
comment mca 2425;
integer procedure comeig2(a, n, em, re, im); value n; integer n;
array a, em, re, im;
begin integer i, j, k, pj, s1, itt;
    real x, y, max, neps;
    array d, u, v[1:n];
    integer array int, int0[1:n];
    s1:= 2 × n × n; eqilbr(a, n, em, d, int0);
    tfmreahes(a, n, em, int); TODRUM(a, s1);
    k:= comeig2:= comvalqri(a, n, em, re, im); FROMDRUM(a, s1);
    neps:= em[0] × em[1]; max:= 0; itt:= 0;
    for i:= k + 1 step 1 until n do
    begin x:= re[i]; y:= im[i]; pj:= 0;
    again: for j:= k + 1 step 1 until i - 1 do
        begin if ((x - re[j]) ↑ 2 + (y - im[j]) ↑ 2 ≤ neps ↑ 2) then
            begin if pj = j then neps:= em[2] × em[1] else pj:= j;
                x:= x + 2 × neps; goto again
            end
        end;
        re[i]:= x; if y ≠ 0 then
        begin comveches(a, n, re[i], im[i], em, u, v);
            FROMDRUM(a, s1); bakreahes1(a, n, int, u);
            TODRUM(u, 2 × n × (i - 1)); i:= i + 1; re[i]:= x
        end
        else
        begin reaveches(a, n, x, em, v); FROMDRUM(a, s1) end;
        bakreahes1(a, n, int, v); TODRUM(v, 2 × n × (i - 1));
        if em[7] > max then max:= em[7];
        itt:= if itt > em[9] then itt else em[9]
    end;
    em[7]:= max; em[9]:= itt; FROMDRUM(a, 0);
    baklbr(n, k + 1, n, d, int0, a); comscl(a, n, k + 1, n, im)
end comeig2;
```

Description mca 2425

comeig2 calculates the eigenvalues and eigenvectors of the n–th order matrix M given in array a[1:n, 1:n].

In array em[0:9], the elements with even subscript must be given as for comeig1.

The real and imaginary parts of the eigenvalues are delivered in array re, im[1:n], and the eigenvectors in the columns of a in the same way as by comeig1.

The eigenvectors are also delivered in the first n ↑ 2 real number locations of the backing storage; the next n ↑ 2 real number locations of the backing storage are altered.

Moreover, in the elements of em with odd subscript, the same results are delivered as by comeig1.

Furthermore, comeig2:= 0, provided that the QR iteration process is completed within em[4] iterations; otherwise, comeig2:= the number, k, of eigenvalues not calculated,

em[5]:= em[4] + 1, and only the last n − k elements of re and im, and the last n − k columns of a contain approximate eigenvalues and eigenvectors of M; similarly, if, for some calculated eigenvector, the Euclidean norm of the residue remains greater than em[1] × em[6] during em[8] iterations, then em[9]:= em[8] + 1.

comeig2 uses eqilbr, tfmreahes, bakreahes1 and baklbr (section 240), reaveches (mca 2411), comvalqri, comveches and comscl (this section), the X8–code procedures TODRUM and FROMDRUM (see introduction), and, indirectly, also vecvec, matvec, tamvec, matmat, tammat, mattam, ichcol and ichrow [3, chapter 20].

REFERENCES

1. P. Naur (ed.), Revised report on the algorithmic language ALGOL 60 (1962).

2. J. H. Wilkinson, The algebraic eigenvalue problem (Clarendon Press, Oxford 1965).

3. T. J. Dekker, ALGOL 60 procedures in numerical algebra, part 1 (Mathematical Centre Tracts 22, Amsterdam 1968).

4. T. J. Dekker (ed.), Series AP 200 (Mathematical Centre Amsterdam 1962–1965).

5. A. S. Householder and F. L. Bauer, On certain methods for expanding the characteristic polynomial, Num. Math. 1(1959) 29–37.

6. W. Givens, Numerical computation of the characteristic values of a real symmetric matrix (Oak Ridge National Laboratory, ORNL-1574, 1954).

7. J. H. Wilkinson, Householder's method for symmetric matrices, Num. Math. 4(1962) 354–361.

8. J. H. Wilkinson, Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection, Num. Math. 4(1962) 362–367.

9. J. H. Wilkinson, Calculation of the eigenvectors of a symmetric tridiagonal matrix by inverse iteration, Num. Math. 4(1962) 368–376.

10. P. Naur, Eigenvalues and eigenvectors of real symmetric matrices, ALGOL programming cont. no. 9, BIT4(1964) 120–130.

11. W. Barth, R. S. Martin and J. H. Wilkinson, Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection, Num. Math. 9(1967) 386–393.

12. J. H. Wilkinson, Two algorithms based on successive linear interpolation (Stanford University, Techn. rep. no. CS 60, 1967).

13. T. J. Dekker, Finding a zero by means of successive linear interpolation, to appear in Proc. Symp. on Constructive aspects of the fundamental theorem of algebra, Rüschlikon, Switzerland (1967).

14. J. M. Ortega and H. F. Kaiser, The LL' and QR methods for symmetric tridiagonal matrices, Comp. J. 6(1963) 99–101.

15. P. A. Businger, Algorithm 253, Eigenvalues of a real symmetric matrix by the QR method, Comm. ACM 8(1965) 217–218.

16. P. A. Businger, Algorithm 254, Eigenvalues and eigenvectors of a real symmetric matrix by the QR method, Comm. ACM 8(1965) 218–219.

17. W. Kahan, When to neglect off–diagonal elements of symmetric tridiagonal matrices (Stanford University, Techn. rep. no. CS 42, 1966).

18. W. Kahan and J. Varah, Two working algorithms for the eigenvalues of a symmetric tridiagonal matrix (Stanford University, Techn. rep. no. CS 43, 1966).

19. E. E. Osborne, On preconditioning of matrices, J. ACM 7(1960) 338–354.

20. B. Parlett, Laguerre's method applied to the matrix eigenvalue problem, Math. Comp. 18(1964) 464–485.

21. J. G. Francis, The QR transformation, parts 1 and 2, Comp. J. 4(1961) 265–271 and 332–345.

22. J. M. Varah, Eigenvectors of a real matrix by inverse iteration (Stanford University, Techn. rep. no. CS 34, 1966).

23. H. Rutishauser, The Jacobi method for real symmetric matrices, Num. Math. 9(1966) 1–10.

24. T. J. Dekker, Newton–Laguerre iteration (Mathematical Centre, Amsterdam, MR 82, 1966).

25. T. J. Dekker, Newton–Laguerre iteration, Programmation en Mathematiques mumeriques. (C.N.R.S., Paris, 1966) 189–200.

26. P. J. Eberlein, A Jacobi–like method for the computation of eigenvalues and eigenvectors of an arbitrary matrix, J. SIAM 10(1962) 74–88.

27. P. J. Eberlein and J. Boothroyd, Solution to the eigenproblem by a norm reducing Jacobi type method, Num. Math. 11(1968) 1–12.

28. J. B. Rosser, C. Lanczos, M.R. Hestenes and W. Karush, Separation of close eigenvalues of a real symmetric matrix, J. Res. Nat. Bur. Standards 47(1951) 291–297.

29. J. H. Wilkinson, Global convergence of tridiagonal QR algorithm with origin shifts, to appear in Linear Algebra and its applications, Vol, 1 no. 3.

EPILOGUE 1

EXPERIMENTS USING THE MC ALGOL 60 SYSTEM FOR THE X8.

In this epilogue we give our results for a number of well-known matrices.

a) The matrix W21+ of Wilkinson [2, p. 308].
eigsym2 and qrisym delivered all eigenvalues in at least 9 digits. The eigenvectors were normalized to Euclidean length 1. The first and second eigenvector lied in the subspace spanned by the first and second correct eigenvectors, but were rotated by about 10 degrees; the same holds for the third and fourth eigenvectors. The fifth and sixth eigenvectors corresponded in 2 digits to the correct eigenvectors, the seventh and eighth in 5, the ninth and tenth in 7, and the others in at least 9 digits.
The computation has taken 11.5 seconds with eigsym2 and 13.0 seconds with qrisym.

b) One of Rutishauser's matrices of order 44 [23].
It took eigsym2 109.3 seconds to deliver all eigenvalues and eigenvectors; the eigenvalues numbered 1,15,20,21,...,30,44, and the eigenvectors numbered 1,15,28,29,30,44, (which are given by Rutishauser) were correct in at least 9 digits.

c) Rosser's matrix of order 8 [28].
Both eigsym2 and qrisym delivered the following results: the eigenvalues numbered 1,2,3,5,8 were correct in 11 digits, number 4 in 10, 6 in 8 and 7 in 9 digits. The eigenvectors were correct in 6 to 7 digits. The computation time was 2.1 seconds with eigsym2 and 1.7 seconds with qrisym.

d) A matrix of order 10 [21].
Both reaeig1 and reaeig3 delivered all eigenvalues and eigenvectors in 5.5 seconds. The eigenvalues agreed in 9 digits with those given by Francis. We could not check the eigenvectors, but the maximal component of all residual vectors, devided by the matrix norm, was $.9_{10}-11$ with reaeig1 and $.5_{10}-11$ with reaeig3.

e) A matrix of order 16 [26].
It took comeig1 17 seconds to deliver all eigenvalues and eigenvectors.
The eigenvalues were correct in at least 10 digits.

f) A matrix of order 40 [26].
It took comeig1 198 seconds to deliver all eigenvalues and eigenvectors.
The eigenvalues were correct in at least 9 digits.
(The results given in Eberlein's paper should be multiplied by a factor 10).

EPILOGUE 2

TESTMATRICES.

In this epilogue we will mention the sets of testmatrices we used to obtain the time formulas in the Appendix.
These matrices have the property that all eigenvalues, or all eigenvalues and the set of eigenvectors, can be chosen arbitrarily.
The matrices used for establishing the formulas for the procedures of CHAPTER 23 are matrices M = XDX of the order n = 2 ↑ k, where k is an integer ranging from 1 to 6.
The matrices X = X(k) are defined by the recurrence relation

$$X(k+1) = 1/\mathrm{sqrt}(2) \begin{pmatrix} X(k) & X(k) \\ X(k) & -X(k) \end{pmatrix} \quad \text{for } k \geq 0,$$

X(0) being the identity matrix of order 1. Consequently XX = I.
Moreover, the columns of X are the eigenvectors of M and the elements of the diagonal matrix D are the eigenvalues of M.

The testmatrices for the procedures of CHAPTER 24 were matrices of the form M = XDY or Y'DX' of order n, n ranging from 12 to 30.
D is a diagonal matrix and XY = I. The matrices X depend on a parameter p, and are defined in the following way:
X[i, j] = p − min(n − i, n − j) for i = 1,...,n; j = 1,...,n − 1.
X[i, n] = 1 for i = 1,...,n.
Obviously, the columns of X are the eigenvectors of M; the angles between these eigenvectors can be varied by the choice of p. We choose p = 10 ↑ k, where k is an integer ranging from 1 to 6.
Y, the inverse of X, is of tridiagonal form with diagonal d, sub-diagonal b and super diagonal c.
d[i] = −2 for i = 2,...,n − 1; d[1] = −1, d[n] = 1 − p.
b[i] = 1 for i = 1,...,n − 2; b[n − 1] = p.
c[i] = 1 for i = 1,...,n − 1.
The rows of Y are the eigenvectors of M', which, as can be seen immediately, are all but one independent of p.

APPENDIX

TIMES FOR THE MC ALGOL 60 SYSTEM FOR THE X8.

In this appendix, we give practical formulas for the computation times in milliseconds of those procedures published above which calculate the eigenvalues, or the eigenvalues and eigenvectors, of a general symmetric or asymmetric real matrix.
The computation time for the major procedures of this booklet obviously depends not only on the order n of the matrix but also on the number of iterations required. We give approximate time formulas depending on n only, because the number of iterations required is closely related to the condition of the matrix and the desired precision, and is therefore not known in advance. Calling the respective procedures with a relative precision of 10^{-9}, the number of QR-iterations proved to be about 2×n, and the number in valsymtri about 15×n.
The formulas have been obtained by fitting a third degree polynomial by the method of least squares to the computation times for our sets of testmatrices (see p. 93). The tests were performed on an Electrologica X8 computer using the MC ALGOL 60 system for the X8, in which system the procedures mca 2000 to 2005 are available as machine-code procedures [3].

The following results have been obtained:
CHAPTER 23 EIGENSYSTEMS OF REAL SYMMETRIC MATRICES.

mca 2313	eigvalsym2	$.3n\uparrow 3 + 15n\uparrow 2$	msec.
mca 2314	eigsym2	$.7n\uparrow 3 + 25n\uparrow 2$	"
mca 2318	eigvalsym1	$.3n\uparrow 3 + 12n\uparrow 2$	"
mca 2319	eigsym1	$.7n\uparrow 3 + 20n\uparrow 2$	"
mca 2322	qrivalsym2	$.3n\uparrow 3 + 6n\uparrow 2$	"
mca 2323	qrisym	$2.1n\uparrow 3 + 8.5n\uparrow 2$	"
mca 2327	qrivalsym1	$.3n\uparrow 3 + 5n\uparrow 2$	"

CHAPTER 24 EIGENSYSTEMS OF REAL MATRICES.

mca 2412	reaeigval	$1.4n\uparrow 3$	msec.
mca 2414	reaeig1	$2.7n\uparrow 3$	"
mca 2417	reaeig3	$3.5n\uparrow 3$	"
mca 2422	comeigval	$1.6n\uparrow 3$	"
mca 2424	comeig1	$3.0n\uparrow 3$	"

Notes
a) Comparing mca 2414 and mca 2417, one might say that reaeig1 is much faster than reaeig3. Taking into account the number i of QR iterations, the formulas would be the following:

mca 2414	reaeig1	$1.4n\uparrow 3 + .7in\uparrow 2$	msec.
mca 2417	reaeig3	$.4n\uparrow 3 + 1.6in\uparrow 2$	msec.

Thus, when the number of iterations is small (relative to n), reaeig3 will be faster than reaeig1.

b) For reaeig2 and comeig2, no time formulas are given, because these procedures are the same as reaeig1 and comeig1, apart from the use of a backing storage.

The formulas presented in this appendix, with the exception of those for mca 2422 and mca 2424 can be used for an estimation of the computation time, which, for most matrices, will not deviate from the actual computation time by more than ten percent. Due to the effect that the number i of QR iterations is somewhat more sensitive to the distibution of the eigenvalues if they are complex, the time formulas for mca 2422 and mca 2424 are of limited value only.