Raabo.

A/S REGNECENTRALEN
SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

SYSTEM LIBRARY

RCSL NO: 53-M7
EDITION: August 1970
AUTHOR : Helge Elbrønd Jensen

RC 4000 SOFTWARE

MATHEMATICAL PROCEDURE LIBRARY

householder

KEY-WORDS: RC 4000 Software, householder, Eigenvalues, Algol Procedure, ISO Tape.

ABSTRACT: The procedure, householder, calculates eigenvalues and, if wanted, the corresponding eigenvectors for a real symmetric matrix.

INFORMATION DEPARTMENT
DK-2500 VALBY · BJERREGAARDSVEJ 5 · PHONE: (01) 46 08 88 · TELEX: 6464 rcinf dk · CABLES: INFOCENTRALEN

1. Function and parameters

Let a denote a real symmetric matrix of order n, and let ev(1), ev(2), ... , ev(n) denote the eigenvalues for this matrix arranged in an increasing sequence, that is ev(i) $\leq$ ev(j) whenever i $\leq$ j.
Let m1 and m2 be prescribed integers so that 1 $\leq$ m1 $\leq$ m2 $\leq$ n.
The procedure householder calculates the eigenvalues ev(m1), ev(m1+1), ... , ev(m2) and, if wanted the corresponding eigenvectors.

Procedure head:

```
householder(n, m1, m2, a, ev, x, eps1);
value n, m1, m2, eps1;
real eps1;
array a, ev, x;
integer m1, m2, n;
```

Call parameters:

- n : the order of the given matrix;
- m1 : an integer, 1 $\leq$ m1 $\leq$ n, denoting the number of the smallest eigenvalue to be calculated.
- m2 : an integer, m1 $\leq$ m2 $\leq$ n, denoting the number of the greatest eigenvalue to be calculated.
- a : a real array a(1:n×(n+1)/2);
  a must contain the lower triangular part of the given symmetric matrix in the following way:
  the diagonal element number i is stored in a(i×(i+1)/2)
  i = 1, 2, ... , n;
  the element in the i'th row and j'th column where j < i is stored in a((i-1)×i/2+j).

Call/Return parameters:

- eps1 : at entry eps1 is positive or negative.
  if eps1 is positive the eigenvectors are calculated. The absolute value of eps1 is a quantity affecting the precision to which the eigenvectors are computed (See part 2.2);
  at exit eps1 denotes an upper bound for the error in any of the calculated eigenvalues.

Return parameters:

ev : a real array ev(m1:m2) containing the calculated eigenvalues.

x : a real array x(m1:m2, 1:n+2); if the eigenvectors are calculated, they are stored in x in such a way that x(k,1), ... x(k,n) denotes the eigenvector corresponding to ev(k); (for each k x(k,n+1) = x(k,n+2) = 0; these quantities are only introduced for ease of programming).

2. Method

The method consists of four parts, tridiagonalisation, calculation of eigenvalues, calculation of eigenvectors, and backtransformation.

2.1. Tridiagonalisation

A matrix is said to be on tridiagonal form, if all elements that are not in the diagonal or just over or under the diagonal, are zero.
Let A_1 be the given symmetric matrix of order n.
A_1 is transformed - by n-2 orthogonal transformations - to a matrix on triangular form.
Each transformation $P_i (i = 1, 2, \ldots, n-2)$ is of the form

$$P_i = I - 2w_i w_i^T$$

where I is the identity-matrix and w_i^T is the row:

$$w_i^T = (w_{i,1}, w_{i,2}, \ldots, w_{i,n-i}, 0, \ldots, 0).$$

and w_i the corresponding column.
Let $A_{i+1} = P_i A_i P_i \qquad i = 1, 2, \ldots, n-2$
For each i the terms $w_{i,1}, w_{i,2}, \ldots, w_{i,n-i}$ are chosen in such a way that

1^0. $w_i^T w_i = 1$

2^0. In A_{i+1} the elements in the rows number n, n-1, ... , n-i+2 are the same as in A_i. The row number n-i+1 is put on 'triangular' form.

Let the elements of A_i be denoted a_{ij}. Put

$$t = n - i.$$

$$\text{sigma} = a_{t+1,1}^2 + a_{t+2,2}^2 + \dots + a_{t+1,t}^2.$$

$$h_i = \text{sigma} \pm a_{t+1,t}\ \text{sqrt(sigma)}.$$

(+ is used if $a_{t+1,t} \geq 0$ else - is used.)

It comes out, that $w_{i,1}$, $w_{i,2}$, ... , $w_{i,t}$ must be chosen as follows

$$w_{i,t} = (a_{t+1,t} \pm \text{sqrt(sigma)})/\text{sqrt}(2h_i).$$

$$w_{i,j} = a_{t+1,j}/\text{sqrt}(2h_i) \qquad j = 1, 2, \dots , t-1.$$

By introducing

$$u_i^T = (a_{t+1,1}, a_{t+2,2}, \dots , a_{t+1,t-1}, a_{t+1,t}$$
$$\pm \text{sqrt(sigma)}, 0, \dots , 0).$$

one will obtain

$$P_i = I - \frac{u_i u_i^T}{h_i}$$

and by introducing the vectors p_i, q_i and the scalar k_i as follows

$$p_i = A_i u_i / h_i$$

$$k_i = u_i^T p_i / (2h_i)$$

$$q_i = p_i - k_i u_i$$

a rather simple calculation will show that

$$A_{i+1} = A_i - u_i q_i^T - q_i u_i^T$$

since A_{i+1} is symmetric one is only calculating the lower triangular part of the matrix.

The above equation is used for the calculation of the first t rows ($t = n-i$) in A_{i+1}. The row number t+1 is on triangular form with the diagonal element unchanged from A_i and the element $(t+1,t) = a_{t+1,t} \pm \text{sqrt(sigma)}$. The rows number t+1, ... , n are according to 2^0 - unchanged from A_i.

At entry the lower triangular part of the given matrix is stored in the array a. For each i the array a is used only to store the lower triangular part of the first t rows of A_{i+1}. The other rows are on triangular form, and the diagonal and subdiagonal elements from these rows are stored in to arrays c and b.
The row number t+1 of the array a is used to store information enough to determine the transformation P_i. Now, P_i is determined by the vector u_i^T and the scalar h_i. By replacing in the array a the element $a_{t+1,t}$ by $a_{t+1,t} \pm \text{sqrt}(\text{sigma})$ one obtain that the non-zero elements of the t+1 row in a are exactly the vector u_i^T. Furthermore from these elements h_i can be determined. Recalling that

$$h_i = \text{sigma} \pm a_{t+1,t}\ \text{sqrt}(\text{sigma}).$$

and denoting by sigma_1 the square-sum of the elements in u_i^T one will obtain

$$\text{sigma}_1 = a_{t+1,1}^2 + \ldots + a_{t+1,t-1}^2 + (a_{t+1} \pm \text{sqrt}(\text{sigma}))^2 =$$
$$2\text{sigma} \pm 2a_{t+1}\text{sqrt}(\text{sigma}) = 2h_i.$$

So $h_i = \text{sigma}_1/2$.
For further information about this part see [4], [6].

2.2. Calculation of eigenvalues

This is based on the following theorem:
let $c_1, \ldots, c_n$ denote the diagonal element and $b_2, \ldots, b_n$ the subdiagonal elements of a symmetric triangulator matrix. For each real number x0 let the sequence $t_1(x0), t_2(x0), \ldots, t_n(x0)$ be defined - if possible - as follows

$$t_1(x0) = c_1 - x0$$

$$t_i(x0) = (c_i - x0) - b_i^2/t_{i-1}(x0). \qquad i = 2, \ldots, n.$$

Let h(x0) denote the number of negative $t_i(x0)$.
Then h(x0) is equal to the number of eigenvalues less than or equal to x0.

Assume that the eigenvalues are arranged in an increasing sequence and that the k'th eigenvalue, ev(k), is to be calculated. Let x1 and x2 be real numbers satisfying $x1 \leq ev(k) < x2$. Such numbers exist, e.g. if norm is denoting the infinity norm of the matrix then x1 = -norm and x2 = norm will do.

Let x0 = (x1 + x2)/2.
h(x0) is calculated by using the above mentioned formular for $t_i(x0)$ i = 1, 2, ... , n.
A new pair (x1, x2) is defined in the following way:
if h(x0) >= k then x1:= x1 and x2:= x0 else x1:= x0, x2:= x2.
For the new pair the procedure is repeated. This is done as long as x2 - x1 > 2×x_m-10×(abs(x1) + abs(x2)) + eps1 where eps1 is a prescribed quantity.
At the end one puts ev(k):= (x1+x2)/2.

Since abs(x1) and abs(x2) always are bounded by norm, it follows that the error in any eigenvalue is bounded by 4×x_m-10×norm + eps1. This number is calculated and stored in eps1.

When calculating the k'th eigenvalue, h(x0) is determined for some x0. The value of h(x0) gives information not only about the k'th eigenvalue, but in general about the eigenvalues of the matrix. By introducing an array p(i) satisfying for each i p(i) $\leq$ ev(i) this information is stored as follows:
if p(h(x_0) + 1) < x0 then p(h(x0) + 1):= x0;
when calculating the k'th eigenvalue one is at the start putting

x1:= max p(1), ... , p(k) ; x2:= ev(k+1);

For further information about this part see [2], [5], [6].

2.3. Calculation of eigenvectors

The matrix is as m 2.2 a symmetric matrix on triangular form with diagonal elements c_1, c_2, ... c_n and subdiagonal elements b_2, ... b_n.
Let ev denote a calculated eigenvalue.
Finding an eigenvector corresponding to ev is equivalent to solve the system

$$\begin{aligned}
&(c_1 - ev)x_1 + b_2x_2 = 0\\
&b_2x_1 + (c_1 - ev)x_2 + b_3x_3 = 0\\
&\vdots\\
&b_{n-1}x_{n-2} + (c_{n-1} - ev)x_{n-1} + b_nx_n = 0\\
&b_nx_{n-1} + (c_n - ev)x_n = 0
\end{aligned} \qquad \text{(I)}$$

where (x_1, ... x_n) denote the wanted eigenvector.

A natural way to solve this system would consist in putting $x_1 = 1$ finding x_2 from the first equation, x_3 from the next and so on; but, as shown in [4], a method like this will often - for several reasons - give hopeless, inaccurate results.

Using a method developed by J.H. Wilkensan ([4]), one is instead solving a system derived from (I) by replacing the zeros on the right side by suitable quantities $d_1, \ldots, d_n$.

These equations are solved by successive elimination of the variables $x_1, x_2, \ldots, x_{n-1}$, but some kind of pivoting is necessary; for each i, x_i is illiminated from the equation, which has the numerical largest coefficient in x_i; more precisely, at the first step we are considering the two first equations

$$(c_1 - cv)x_1 + b_2x_2 = d_1$$

$$b_2x_2 + (c_2 - ev)x_2 + b_3x_3 = d_2.$$

The equation which has the numerical largest coefficient is x_1 is denoted

$$p_1x_1 + q_1x_2 + r_1x_3 = d_1''$$

from this equation x_1 is calculated and the expression inserted in the other equation. The so obtained equation in x_2 and x_3 is denoted

$$u_2x_2 + v_2x_3 = d'$$

At the i'th step we are considering the two equations

$$u_ix_i + v_ix_{i+1} = d_i'$$

$$b_{i+1}x_i + (c_{i+1} - cv)x_{i+1} + b_{i+1}x_{i+2} = d_{i+1}.$$

again the equation which has the numerical largest coefficient in x_i is denoted

$$p_ix_i + q_ix_{i+1} + r_ix_{i+2} = d_i''$$

from this equation x_i is calculated and the expression inserted in the other equation.

In this way we obtain the following system:

$$p_1x_1 + q_1x_2 + r_1x_3 = d_1''$$

$$p_2x_2 + q_2x_3 + r_2x_4 = d_2''$$

$$\vdots$$

$$p_{i-2}x_{n-2} + q_{n-2}x_{n-1} + v_{n-2}x_n = d_{n-2}''$$

$$p_{n-1}x_{n-1} + q_{n-1}x_n = d_{n-1}''$$

$$p_nx_n = d_n''.$$

We now assume, that $d_1, d_2, \ldots, d_n$ were chosen in such a way, that $d_1'', d_2'', \ldots, d_n''$ are all equal to one.
This system is solved in the natural way and the obtained vector normed. (and again denoted $x_1, \ldots, x_n$). It can be proved ([4]) that this vector will usually be a good approximation, at least it will never be hopeless inaccurate.
A vector with sufficient accuracy is obtained by solving the above system once again, but replacing the terms $d_1'', \ldots, d_n''$ by the coordinates in the first approximation $x_1, \ldots, x_n$.
For further information about this part see [3], [4], [6].

2.4. Backtransformation

The problem is to transform the calculated eigenvectors (for the triangular matrix) to eigenvectors corresponding to the original matrix. Recalling that the original matrix was transformed to a matrix on tridiagonal form by n-2 orthogonal transformations $P_1, P_2, \ldots, P_{n-2}$, it easily follows, that if z_{n-1} is an eigenvector for the triangular matrix then

$P_1 P_2 \ldots P_{n-2} z_{n-1}$ is an eigenvector for the original matrix.

Putting $P_i P_{i+1} \ldots P_{n-2} z_{n-1} = z_i$

one will obtain $P_i z_{i+1} = z_i$

and the wanted vector z_1, is calculated in n-2 steps. Using the notation from 2, 1 (tridiagonalisation) one will get

$$z_i = z_{i+1} - \frac{u_i u_i^T}{h_i} z_{i+1} \quad \left(\text{because } P = I - \frac{u_i u_i^T}{h_i}\right).$$

The non-zero elements of u_i are stored in the t + 1 row (t = n - i) of the array a and h_i = sigma/2, where sigma denotes the square-sum of the elements in u_i (see 2.1).

3. Accuracy, Time and Storage Requirements

Accuracy: The accuracy in the eigenvalues depends on the value of the call parameter eps1.

It easily follows from the description of the method part 2.2, that the error in any eigenvalue is bounded by $4 * B\text{-}10 \times$ norm + eps1 where norm denotes the infitity norm of the triangular matrix.
For further information on this part see 4. Test and Discussion.

Time : This depends on the wanted accuracy, that is the term eps1, and first of all on the order n of the matrix equation. Generally the execution time will be proportional to n**2.
Using eps1 = B-10 and denoted by

I : The execution time when all eigenvalues and all eigenvectors are calculated

II : The execution time when all eigenvalues but no eigenvectors are calculated.

III: The execution time when only the greatest eigenvalue and the corresponding eigenvector are calculated.

the greatest execution times (in sec.) obtained were as follows:

Order of the matrix	I	II	III
5	0.32	0.25	0.09
10	1.32	0.89	0.28
15	3.22	1.99	0.68
20	6.30	3.63	1.39
25	10.75	5.91	2.46

The following example illustrates the connection between the execution time and the value of eps1, where all eigenvalues and eigenvectors for a matrix of order 20 are calculated:

eps1 =	B-4	B-5	B-6	B-7	B-8	B-9	B-10
Time =	4.88	5.15	5.44	5.74	5.94	6.16	6.30

Storage requirements: 9 segments of program
Typografical length : 149 lines

4. Test and Discussion

The procedure has been tested by several matrices, essentially the following four types (denoting by a(ij) the element in the i'th row and the j'th column and by n the order of the matrix in question):

Type I : a(i,j) = a(j,i) = n - i + 1. This matrix has well-separated eigenvalues given by

$$ev(i) = \frac{1}{2(1-\cos(\frac{2i-1}{2n+1}\,pi)} \qquad i = 1, 2, \ldots, n$$

Type II : a(i,j) = a(j,i) = 1 for all i, j.
All eigenvalues are 0 except one which is n

Type III : a(i,j) = a(j,i) = 0 for i = j else 1.
All eigenvalues are -1 except one which is n-1.

Type IV : a(i,j) = 0 for j < i-1 and j > i+1.
a(i,i-1) = a(i,i+1) = 1.
$a(i,i) = abs(\frac{n+1}{2} - i)$ i = 1, 2, ... , n.
The matrix has a number of extremely close, but not coincident eigenvalues.

When all eigenvalues and all eigenvectors are calculated, a measure for the error for the whole procedure is obtained by checking the identity $Ax_k = ev(k)x_k$ for each k.
Finding the largest deviation in any coordinate and using as testnorm the mean of these k numbers, the following results are obtained:

Matrix	Value of eps1			
	$_{10}$-4	$_{10}$-6	$_{10}$-8	$_{10}$-10
Type I order 10	3.1$_{10}$-5	9.0$_{10}$-7	1.5$_{10}$-8	3.2$_{10}$-9
Type I order 20	1.4$_{10}$-5	1.5$_{10}$-6	1.6$_{10}$-8	3.9$_{10}$-8
Type I order 25	2.0$_{10}$-4	2.1$_{10}$-6	3.2$_{10}$-8	2.1$_{10}$-8
Type II order 10	1.2$_{10}$-5	3.3$_{10}$-7	1.9$_{10}$-9	3.8$_{10}$-10
Type II order 20	3.5$_{10}$-5	7.6$_{10}$-8	3.1$_{10}$-9	2.0$_{10}$-10
Type II order 25	4.5$_{10}$-5	3.5$_{10}$-7	4.6$_{10}$-10	6.4$_{10}$-10
Type III order 10	1.2$_{10}$-5	9.3$_{10}$-8	1.4$_{10}$-9	1.7$_{10}$-10
Type III order 20	1.2$_{10}$-5	6.7$_{10}$-8	6.3$_{10}$-10	4.3$_{10}$-10
Type III order 25	5.4$_{10}$-6	7.0$_{10}$-8	1.5$_{10}$-9	6.0$_{10}$-10
Type IV order 11	2.1$_{10}$-3	9.8$_{10}$-6	2.1$_{10}$-7	9.1$_{10}$-9
Type IV order 15	6.7$_{10}$-3	6.8$_{10}$-5	6.0$_{10}$-7	1.7$_{10}$-7
Type IV order 21	1.5$_{10}$-2	1.9$_{10}$-3	6.3$_{10}$-4	4.1$_{10}$-6

The jacobi algorithm solves almost the same problem as householder; The only difference is, that the jacobi procedure necessarily calculates all the eigenvalues (and eigenvectors), while it is possible with the householder procedure only to calculate some of the eigenvalues (and eigenvectors). Calculating all eigenvalues and all eigenvectors and using in householder eps1 = $_{10}$-10 a comparison between the two procedures gave the following results:

Matrix	Testnorm for householder	Testnorm for jacobi	Time for householder	Time for jacobi
Type I order 5	1.4_{10}-9	0.8_{10}-9	0.35	0.27
Type I order 10	3.2_{10}-9	4.0_{10}-9	1.33	2.02
Type I order 15	2.2_{10}-8	1.0_{10}-8	3.29	6.61
Type I order 20	3.5_{10}-8	2.2_{10}-8	6.30	14.92
Type I order 25	2.1_{10}-8	3.2_{10}-8	11.12	29.08
Type II order 5	7.0_{10}-10	5.0_{10}-10	0.20	0.07
Type II order 10	1.6_{10}-10	0.1_{10}-10	0.53	0.22
Type II order 15	4.0_{10}-10	1.0_{10}-10	1.13	0.55
Type II order 20	4.5_{10}-10	1.6_{10}-10	1.98	0.97
Type II order 25	7.4_{10}-10	1.2_{10}-10	3.46	1.38

Remembering that the matrices of type I have well-separated eigenvalues, and that the matrices of type II have all but one eigenvalue equal to zero, one might draw the following conclusion:
The procedure householder is to be preferred in case of matrices with separated eigenvalues, because of higher speed, or in cases, where only one or a few eigenvalues are wanted.
The procedure jacobi is to be preferred in case of matrices with coincident eigenvalues.

Example

We consider a symmetric matrix of order n. The term m1 denotes the number of the smallest, m2 the number of the greatest eigenvalue to be calculated. The eigenvectors are calculated only if the term eps1 is positive. Input is the value of the quantities n, m1, m2, eps1 and the lower triangular part of the matrix.

Testprogram

```
begin
  integer n, m1, m2, i, k;
  real eps1;
  boolean vect;
  read(in, n,m1,m2,eps1); vect:= eps > 0;
  begin
    array a(1:n×(n+1)/2), x(m1:m2, 1:n+2), ev(m1:m2);
    for i:= 1 step 1 until n×(n+1)/2 do read(in, a(i));
    householder(n, m1,m2,a,ev,x,eps1);
    write(out, <:Eigenvalues <10><10>:>);
    for i:= m1 step 1 until m2 do
    write(out, <<dd>, i, <<    -dddd.ddddddddd>, ev(i), <:<10>:>);
    if vect then
    begin
      write(out, <:<10> Eigenvectors<10>:>);
      for k:= m1 step 1 until m2 do
      begin
        write(out, <:<10>:>, <<dd>, k, <:<10>:>);
        for i:= 1 step 1 until n do
        write(out, <<    -dddd.ddddddddd>, x(k, i), <:<10>:>);
      end k;
    end vect;
  end;
end;
```

For the matrix of order 5:

5	4	3	2	1
4	6	0	4	3
3	0	7	6	5
2	4	6	8	7
1	3	5	7	9

using m1 = 3, m2 = 5 and eps1 = $_{10}$-8 the complete output is:

```
Eigenvalue
3        4.848950119
4        7.513724158
5       22.406875316
```

Eigenvectors

3
-0.547172796
0.312569920
-0.618112076
0.115606593
0.455493746

4
-0.550961958
-0.709440337
0.340179132
0.083410953
0.265435679

5
0.245877938
0.302396039
0.453214523
0.577177152
0.556384584

end

For the matrix of order 10:

10	9	8	7	6	5	4	3	2	1
9	9	8	7	6	5	4	3	2	1
8	8	8	7	6	5	4	3	2	1
7	7	7	7	6	5	4	3	2	1
6	6	6	6	6	5	4	3	2	1
5	5	5	5	5	5	4	3	2	1
4	4	4	4	4	4	4	3	2	1
3	3	3	3	3	3	3	3	2	1
2	2	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	1	1

using m1 = 1, m2 = 10 and eps1 = $-p-10$ the complete output is:

Eigenvalues

1 0.255679563
2 0.273786762
3 0.307978528
4 0.366208875
5 0.465233088
6 0.643104132
7 1.000000000
8 1.873023068
9 5.048917339
10 44.766068656

end

5. References

(1) J.H. Wilkinson: Householders method for symmetric matrices. Numerische Mathematik 4, p. 354-361 (1962)

(2) J.H. Wilkinson: Calculation of the eigenvalues of a symmetric diagonal matrix by the method of besection. Numerische Mathematik 4, p. 362-367 (1962).

(3) J.H. Wilkinson: Calculation of the eigenvectors of a symmetric tridiagonal matrix by inverse iteration. Numerische Mathematik 4, p. 368-376 (1962).

(4) J.H. Wilkinson: Calculation of the eigenvectors of co-diagonal matrices, Computer Journal 1, p. 90-96 (1958).

(5) J.H. Wilkinson, W. Bath, R.S. Martin: Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection Numerisch Mathematik 9, p. 386-393 (1967).

(6) P. Naur: Eigenvalues and eigenvectors of real symmetric matrices, BIT 4, p. 120-130 (1964).

6. Algol text

```
householder = set 9
householder = algol
external

 procedure householder(n,m1,m2,a,ev,x,eps1);
 value n,m1,m2,eps1;
 real eps1;
 array a,ev,x;
 integer m1,m2,n;


 begin
  integer i,j,k,i0,j0,i1,t,t0,t1;
  real h,s,k1,sigma,at,bt,eps,bi,bi1,norm,x1,x2,x0,u,v;
  array c(1:n),r(0:n),p,b,q(1:n+1),m(1:n+2);
  boolean vect;
```

```
eps:=0; j:=n×(n+1)/2;
for i:=1 step 1 until j do eps:=eps + abs(a(i));
eps:=(3₁₀-11)×eps/j;
for i:=1 step 1 until n-2 do
begin
 t:=n-i; t0:=t×(t+1)/2; t1:=t0 +t;
 sigma:=0;
 for k:=t0+1 step 1 until t1 do sigma:=sigma+a(k)××2;
 at:=a(t1);
 b(t+1):=bt:= if at>0 then-sqrt(sigma) else sqrt(sigma);
 if abs(bt)>eps then
 begin
  h:=sigma-at×bt; a(t1):=at-bt;
  for j:=1 step 1 until t do
  begin
   comment computation of pi;
   s:=0; j0:= (j-1)×j/2;
   for k:=1 step 1 until j do s:=s+a(j0+k)×a(t0+k);
   for k:=j+1 step 1 until t do s:=s+a(k×(k-1)/2+j)×a(t0+k);
   q(j):=s/h;
  end j;
  k1:=0;
  comment computation of ki;
  for j:=1 step 1 until t do k1:=k1+a(t0+j)×q(j);
  k1:=k1/2/h;
  comment computation of qi;
  for j:=1 step 1 until t do q(j):=q(j)-k1×a(t0+j);
  for j:= 1 step 1 until t do
  begin
   comment computation of the i+1 matrix;
   j0:=(j-1)×j/2;
   for k:=1 step 1 until j do
   a(j0+k):=a(j0+k)-a(t0+j)×q(k)-a(t0+k)×q(j);
  end j;
 end abs(bt)>eps;
end i;
for i:=1 step 1 until n do c(i):=a(i×(i+1)/2);
b(2):=a(2);  b(1):=(n+1):=0;


comment the eigenvalues ev(m1),ev(m1+1), . . ,ev(m2)
are now calculated;

vect:=(if eps1<0 then false else true);
eps1:=abs(eps1);
norm:=0;
for i:=1 step 1 until n do
begin
 h:=abs(b(i))+abs(c(i))+abs(b(i+1));
 if norm<h then norm:=h;
 q(i):=b(i)××2;
end i;
for i:=m1 step 1 until m2 do p(i):= -norm;
for k :=m2 step -1 until m1 do
begin
comment computation of the k eigenvalue;
```

```
 for i:=m1 step 1 until k-1 do if p(i)>p(k) then p(k):= p(i);
 x1:=p(k); x2:= ( if k<n then ev(k+1) else norm);
 for x0:=(x1+x2)/2 while x2-x1>2×10-10×(abs(x1)+abs(x2))+eps1 do
 begin
   h:=0; s:=1;
  for i:=1 step 1 until n do
  begin
   s:=c(i)-x0-(if s<>0 then q(i)/s else abs(b(i))×10);
   if s<0 then h:=h+1;
  end i;
  if h>=k then x2:=x0 else x1:=x0;
 if p(h+1)<x0 then p(h+1):=x0;
  end x0;
 ev(k):=x0;
end k;
eps1:=1/2×eps1+4×10-10×norm;


if vect then
begin
 comment computation of the eigenvectors corresponding
 to the calculated eigenvalues;
 eps:= (3-11)×norm;
 for k:=m2 step -1 until m1 do
 begin
  comment the pivotal equations are calculated;
  u:=c(1)-ev(k); v:=b(2);
  if abs(v)<eps then v:=eps;
  for i:=1 step 1 until n-1 do
  begin
   bi:=b(i+1); if abs(bi)<eps then bi:=eps;
   bi1:=b(i+2); if abs(bi1)<eps then bi1:=eps;
   if abs(u)>abs(bi) then
   begin
     p(i):=u; q(i):=v; r(i):=0;
     m(i+1):=bi/u;
     u:=c(i+1)-ev(k)-m(i+1)×v; v:=bi1;
   end
   else
   begin
     p(i):=bi; q(i):=c(i+1)-ev(k);
     r(i):=bi1; m(i+1):=u/bi;
     u:=v-m(i+1)×(c(i+1)-ev(k));
     v:=-m(i+1)×bi1;
   end;
  end i;
  q(n+1):=q(n):=r(n):=x(k,n+1):=x(k,n+2):=h:=0;
  p(n):=if abs(u)>eps then u else eps;
  for i:=n step -1 until 1 do
  begin
   comment the first approximation;
   x(k,i):=(1-q(i)×x(k,i+1)-r(i)×x(k,i+2))/p(i);
   h:=h+x(k,i)↑2;
  end;
```

```
      h:=sqrt(h);
      for i:=1 step 1 until n do x(k,i):=x(k,i)/h;
      h:=0;
      for i:=n step -1 until 1 do
      begin
       comment the second approximation;
       x(k,i):=(x(k,i)-q(i)×x(k,i+1)-r(i)××x(k,i+2))/p(i);
       h:=h+x(k,i)××2;
      end;
      h:=sqrt(h);
      for i:=1 step 1 until n do x(k,i):=x(k,i)/h;
     end k;

     comment the calculated eigenvectors are now transformed
     to eigenvectors corresponding to the original matrix;

     for k:=m1 step 1 until m2 do
     begin
      for j:=n-2 step -1 until 1 do
      begin
       t:=n-j;  t0:=t×(t+1)/2; sigma:=0;
       for i:=1 step 1 until t do sigma:=sigma+a(t0+i)××2;
       if sigma<>0 then
       begin
        s:=0;
        for i:=1 step 1 until t do s:=s+a(t0+i)××x(k,i);
        s:=-2××s/sigma;
        for i:=1 step 1 until t do
        x(k,i):=x(k,i)+s××a(t0+i);
       end sigma<>0;
      end j;
     end k;
    end vect;
   end;
  end;
```