Raabo.

A/S REGNECENTRALEN
SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

SYSTEM LIBRARY

RCSL NO: 55-D57
TYPE : Algol 5 procedure
AUTHOR : N. Schreiner Andersen,
P. Frost Larsen
EDITION: July 1969 (E)

RC 4000 SOFTWARE

MATHEMATICAL PROCEDURE LIBRARY

eberlein

ABSTRACT:

The procedure, eberlein, solves the eigenproblem for a real matrix by means of a sequence of Jacobi-like transformations.

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

Solution of the eigenproblem for real matrices

eberlein(n, a, t, tmx, first, result).

1. Function and parameters.

It is possible to chose one of several forms of solution according to the rules given in the parameter list.

If the iteration process does not converge within the given number of iterations or converges to a matrix that is not of block diagonal form, no solution is found. This situation is indicated by the boolean parameter, first.

Input parameters:

n : the order of matrix a.

result : if result is true then in case of convergence the eigenvalues will be placed in the two first columns of matrix a.

Input/Output parameters:

a[1:n,1:n] : at entry the matrix for with the eigenproblem is to be solved.

At exit one of the following three situations can occur:

1) if convergence occurs and result is false :

the real eigenvalues occupy diagonal elements while real and imaginary parts of complex conjugate eigenvalues occupy diagonal and off diagonal corners of 2×2 blocks on the main diagonal.

2) if convergence occurs and result is true :

the eigenvalues will be placed in the two first columns according to the following rules

a real eigenvalue x = a[j,j] makes

a[j,1] = x
and a[j,2] = 0

a complex conjugate pair of eigenvalues

x + i×y = a[j,j] + i×a[j,j+1]

and x - i×y makes

a[j,1] = x
a[j,2] = y
a[j+1,1] = x
and a[j+1,2] = -y

3) if convergence fails no eigenvalues can be calculated as a result of the procedure call. The matrix, a, is equal to the transformed matrix. During a new call of ,eberlein, it is possible to try whether more iterations will result in convergence or not. (first is set to false).

t[1:n,1:n] : if first is false at entry and tmx > 0 then t given at entry is multiplied by the transformation matrix calculated in the procedure.

Eigenvectors of real eigenvalues occupy columns of the transformation matrix. Eigenvectors corresponding to complex conjugate eigenvalues given by

a[j,j] + i×a[j,j+1]
and a[j,j] - i×a[j,j+1]

are formed as

t[k,j] + i×t[k,j+1]
and t[k,j] - i×t[k,j+1]

where k = 1,2,....,n .

tmx : at entry:

the maximum number of transformations performed is abs(tmx). If tmx < 0 then t is unaltered.

at exit tmx records the number of transformations performed.

first : at entry tells whether t is a result of a foregoing transformation or not. (see under t[1:n,1:n]).

at exit first is true if convergence occurs in less than tmx iterations otherwise first is false.

2. Method.

The procedure is based on a modification of a generalized Jacobi-method [1]. There exists no proof of convergence for this special modification, but numerical experiments have shown the worth of the method.

A transformation matrix T transforms the matrix A into a matrix of block diagonal form $A' = T^{-1}AT$.

The transformation matrix T is generated from a sequence of two-dimensional transformations $T_1(k, m)$, where (k, m) is the pivot pair.

Each T_i is of the form RS where R is a rotation and S a shear.

Let a_{ij}, r_{ij} and s_{ij}, i = 1, 2, ..., n, j = 1, 2, ... , n be the elements of A, R and S respectively.

Then the rotation is determined as

$$r_{kk} = r_{mm} = \cos x$$
$$r_{km} = -r_{mk} = -\sin x$$
$$r_{ij} = \delta_{ij} \text{ (kronecker-delta) } i \neq k, m \text{ and } j \neq k, m$$

where x are given by

$$\tan 2x = (a_{km} + a_{mk})/(a_{kk} - a_{mm})$$

x being chosen so that after the transformation the norm of the k-th column is greather than or equal to the norm of the m-th column.

The shear is determined by

$$s_{kk} = s_{mm} = \cosh y$$
$$s_{km} = s_{mk} = -\sinh y$$
$$s_{ij} = \delta_{ij} \quad \text{otherwise}$$

y is chosen to reduce the Euclidean norm of

$$A_{i+1} = (T_1 T_2 \dots T_i)^{-1} A_i (T_1 T_2 \dots T_i).$$

In particular

$$\tanh y = (ED - H/2)/(G + 2(E^2 + D^2))$$

where

$$E = a_{km} - a_{mk}$$

$$D = \cos 2x(a_{kk} - a_{mm}) + \sin 2x(a_{km} + a_{mk})$$

$$G = \sum_{i \neq k,m} (a_{ki}^2 + a_{ik}^2 + a_{im}^2 + a_{mi}^2)$$

$$H = \cos 2x\left(2 \sum_{i \neq k,m} (a_{ki}a_{mi} - a_{ik}a_{im})\right)$$

$$-\sin 2x\left(\sum_{i \neq k,m} (a_{ki}^2 + a_{im}^2 - a_{ik}^2 - a_{mi}^2)\right)$$

The process will normally result in a matrix A' with real eigenvalues on the diagonal and complex conjugate eigenvalues in 2×2 blocks on the main diagonal (eigenvalues $a_{jj} \pm ia_{jj+1}$). 2×2 blocks because the process theoretically results in

$$||a^1|| \geq ||a^2|| \geq \ldots \geq ||a^n||$$

where $||a^i||$ is the norm of the i-th column of A'.

If however more than two eigenvalues are of the same norm the above picture does not hold. This also happens if the matrix A_i at some step is near a more general solution of block diagonal form, because the convergence criteria then may stop the process.

The procedure however results in a form with 2×2 blocks obtained by interchanging rows and columns if necessary.

A matrix of the form aI + S where I is the identity matrix and S is skew symmetric, and cases with blocks of this form cannot be handled by the procedure. (If the form not it self is a solution). There is no quarantee that the transformation method used will not result in convergence to a form embedding blocks from which the procedure cannot calculate the eigenvalues. In testexamples this did happen, but only in special cases chosen to examine the stability of such solutions. Notice that if the number of iterations performed is less than the maximum number of iterations allowed and ,first, is false then the resulting matrix is of the above mentioned form.

The eigenvectors are calculated from the rows of the transformation matrix, as described in the data list.

Numerical examples have shown that eigenvectors corresponding to multiple eigenvalues are normally linear dependent. (Except for numerical errors).

The algol procedure is based on an algorithm developped by Eberlein and Boothroyd [2]. The following changes are however made:

1. A program part ensures that the eigenvalues are placed in 1×1 and 2×2 blocks on the main diagonal. Matrices obtained as a result of convergence for which 1×1 and 2×2 blocks can not be constructed results in alarm message through the parameters first and tmx.

2. the parameter list is changed.

3. a new dynamical form of convergence criterion is introduced.

The convergence criterion is based on the four reals ep, eps, eps1, and eps2.

At the start of the procedure ep, eps and eps1 are calculated as

$$ep = max \times 10^{-14}$$
$$eps = max \times 1.001_{10}-9$$
$$eps1 = max \times 10^{-3}$$

where

$$max = maximum(|a_{ij}|)\ i, j = 1, 2, \ldots, n.$$

If for a single value of eps1

$$|a_{ij} - a_{ji}| \leq eps1 \text{ or } (|a_{ij} + a_{ji}| \leq eps1 \text{ and } |a_{ii} - a_{jj}| \leq eps1)$$

no more transformation are carried out before after a change of eps1. If eps1 < eps convergence has occured, otherwise a new value of eps1 is calculated as eps1 = eps1/10. The above value of eps makes sure that the resulting eps1 is near to and less than eps. eps = max $\times 10^{-9}$ could cause an extra serie of iterations with eps1 = max $\times 10^{-10}$ because of rounding errors.

As pivot pairs are only chosen pairs of elements for which

$$(|a_{ij} - a_{ji}| > eps\ 2 \text{ and } |a_{ii} - a_{jj}| > eps\ 2)$$
$$\text{or } |a_{ij} + a_{ji}| > eps\ 2$$

If only identity transformations occurs as a result of this rule then a new value of eps2 is calculated as eps2 = eps2/10. Numerical experiments have shown that this extra mechanism is necessary to ensure convergence of some ill conditioned numerical examples.

The starting value of eps2 for every new value of eps1 is eps2 = eps1/10.

If eps2 gets less than ep convergence is not obtained by this algorithm and the process is stopped.

The values of ep, eps, eps1, and eps2 are results of experiments reducing the computation time about 30 per cent compared with a program making transformations for all pairs of elements cyclically.

3. Accuracy and storage Requirements.

Accuracy

In case of convergence the following inequalities holds for the elements of A'

$$|a_{ij} - a_{ji}| \leq \text{eps or}$$
$$|a_{ij} + a_{ji}| \leq \text{eps and } |a_{ii} - a_{jj}| \leq \text{eps}$$

for $i = 1, 2, \ldots, n-1$ and $j = i+1, i+2, \ldots, n$ where

$$\text{eps} = \max(|\text{elements of original matrix, a}|) \times 10^{-9}$$

Storage requirements

1) ALGOL 5, index check: 7 tracks of program and 52 local variables
2) ALGOL 5, no index check: 6 tracks of program and 52 local variables.

Typographical length 167 lines of program exclusive the comment after the last end.

4. Test and discussion

The procedure has been tested on the ALGOL 5 system for matrices of order $\leq$ 12.

The testprogram makes besides call of - eberlein - a calculation of

$$\text{testnorm} = ||A \times \underline{x} - \quad \times \underline{x}||/||\quad \times \underline{x}||$$

where A is the matrix for which the eigenproblem is solved, $\underline{x}$ is a calculated eigenvector and the corresponding eigenvalue.

Calculation of - testnorm - is made by a real procedure testnorm(n, A, t, k, complex, x1, x2) in the testprogram.

A list of input parameters, results, and calculated values of - testnorm - is delivered by the testprogram.

The starting values and following calculation of eps1 and eps2 are obtained as results of experiments resulting in 30 per cent decrease in execution time in solving testexamples.

Example no 1

Time:

ALGOL 5, core storage, no index check 0.36 sec.

Input parameters:

n = 3
tmx = 50

matrix a:

1.000	0.000	0.010
0.100	1.000	0.000
0.000	1.000	1.000

Results:

tmx = 15
first = true

Eigenvalues after 15 iterations

1	1.1000000000	
2	0.9499999999	+0.0866025403×i
3	0.9499999999	-0.0866025403×i

Eigenvectors:

1		
	-0.2745741273	
	-0.2745741274	
	-2.7457412704	
2		
	0.3262845267	+0.1836278766×i
	-0.0041158575	-0.3743846272×i
	-3.2216866940	+1.9075675104×i
3		
	0.3262845267	-0.1836278766×i
	-0.0041158575	+0.3743846272×i
	-3.2216866940	-1.9075675104×i

Testnorm for corresponding eigenvalues and eigenvectors

no. of eigenvalue	testnorm
1	1.5_{10}-10
2 and 3	1.8_{10}-10

Example no 4

Time:

ALGOL 5, core storage, no index check 10.1 sec.

Input parameters:

n = 7
tmx = 100

matrix a:

-1	1	0	0	0	0	0
-1	0	1	0	0	0	0
-1	0	0	1	0	0	0
-1	0	0	0	1	0	0
-1	0	0	0	0	1	0
-1	0	0	0	0	0	1
-1	0	0	0	0	0	0

Results:

tmx = 64
first = true

Eigenvalues after 64 iterations

1	-0.9999999982	
2	0.7071067794	+0.7071067790×i
3	0.7071067794	-0.7071067790×i
4	-0.7071067788	+0.7071067790×i
5	-0.7071067788	-0.7071067790×i
6	0.0000000001	-0.9999999956×i
7	0.0000000001	+0.9999999956×i

Eigenvectors:

1

-0.5585067124	
0.0000000001	
-0.5585067124	
0.0000000001	
-0.5585067123	
0.0000000000	
-0.5585067125	

2

-0.1107710832	+0.3310964522×i
-0.4232186137	+0.4868900143×i
-0.7543150660	+0.3761189310×i
-0.9101086283	+0.0636714003×i
-0.7993375446	-0.2674250515×i
-0.4868900143	-0.4232186137×i
-0.1557935621	-0.3124475306×i

3

-0.1107710832	-0.3310964521×i
-0.4232186137	-0.4868900143×i
-0.7543150660	-0.3761189310×i
-0.9101086283	-0.0636714003×i
-0.7993375446	+0.2674250515×i
-0.4868900143	+0.4232186137×i
-0.1557935621	+0.3124475306×i

4

0.4530114436	-0.6611576004×i
0.6001930025	+0.1266788858×i
-0.0609645978	-0.3263325577×i
0.7268718886	-0.4735141167×i
0.2738604449	+0.1876434837×i
0.1266788860	-0.6001930026×i
0.7878364866	-0.1471815591×i

5

0.4530114436	+0.6611576004×i
0.6001930025	-0.1266788858×i
-0.0609645978	+0.3263325577×i
0.7268718886	+0.4735141167×i
0.2738604449	-0.1876434837×i
0.1266788860	+0.6001930026×i
0.7878364866	+0.1471815591×i

6

0.5486381922	+0.2372706779×i
0.7859088684	-0.3113675134×i
0.2372706765	-0.5486381898×i
-0.0000000002	+0.0000000002×i
0.5486381923	+0.2372706781×i
0.7859088687	-0.3113675132×i
0.2372706767	-0.5486381901×i

7

0.5486381922	-0.2372706779×i
0.7859088684	+0.3113675134×i
0.2372706765	+0.5486381898×i
-0.0000000002	-0.0000000002×i
0.5486381923	-0.2372706781×i
0.7859088687	+0.3113675132×i
0.2372706767	+0.5486381901×i

Testnorm for corresponding eigenvalues and eigenvectors

no. of eigenvalue	testnorm
1	2.0_{10} -9
2 and 3	3.1_{10} -9
4 and 5	3.0_{10} -9
6 and 7	5.9_{10} -9

Example no 11

Time:

ALGOL 5, core storage, no index check 2.43 sec.

Input parameters:

n = 4
tmx = 100

matrix a:

1	0	0	0
1	1	0	0
0	1	1	0
0	0	1	1

Results:

tmx = 45
first = true

Eigenvalues after 45 iterations

1	1.0014985857	+0.0014992361×i
2	1.0014985857	-0.0014992361×i
3	0.9985014113	+0.0014978791×i
4	0.9985014113	-0.0014978791×i

Eigenvectors:

1		
	-0.0000637318	+0.0000264031×i
	-0.0124657185	+0.0300779129×i
	5.8782571872	+14.1901547856×i
	6694.9278156800	+2771.1941409600×i
2		
	-0.0000637318	-0.0000264031×i
	-0.0124657185	-0.0300779129×i
	5.8782571872	-14.1901547856×i
	6694.9278156800	-2771.1941409600×i
3		
	0.0000637404	+0.0000264010×i
	-0.0124478455	-0.0300712114×i
	-5.8782228542	+14.1910426896×i
	6696.9885651200	-2775.7892716800×i
4		
	0.0000637404	-0.0000264010×i
	-0.0124478455	+0.0300712114×i
	-5.8782228542	-14.1910426896×i
	6696.9885651200	+2775.7892716800×i

Testnorm for corresponding eigenvalues and eigenvectors

no. of eigenvalue			testnorm
1	and	2	1.5_{10} -9
3	and	4	1.8_{10} -9

Example no 12

Input parameters:

n = 3
tmx = 50

matrix a:

1.000	1.000	1.001
-1.000	1.000	0.000
-1.000	0.000	1.000

Results:

tmx = 28
first = false

Limiting matrix after 28 iterations

1.00_{10} +0	-1.17_{10} +0	7.60_{10} -1
1.17_{10} +0	1.00_{10} +0	2.22_{10} -1
-7.60_{10} -1	-2.22_{10} -1	1.00_{10} +0

5. References.

[1] P.I. Eberlein: A Jacobi-like Method for the Automatic Computation of Eigenvalues and Eigenvectors of an Arbitrary Matrix.
I. SIAM, vol. 10, No. 1, 1962.

[2] P.I. Eberlein and John Boothroyd: Solution to the Eigenproblem by a Norm Reducing Jacobi Type Method
Numerische Mathematik 11, 1-12 (1968).

6. Algorithm

```
eberlein=set 7
eberlein=algol
external

procedure eberlein(n,a,t,tmx,first,result);
value n;
boolean first,result;
integer n,tmx;
array a,t;
comment 1 ;
  begin
  real eps,ep,aii,aij,aji,h,g,hj,aik,aki,aim,ami,tep,tem,d,c,e,akm,amk,cx,sx,
       cot2x,sig,cotx,cos2x,sin2x,te,tee,yh,den,tanhy,chy,shy,c1,c2,s1,s2,
       tki,tmi,tik,tim,eps1,eps2;
  integer i,j,k,m,it,nless1;
  boolean mark,left,right;
    mark := right := false;
    if tmx > 0 then
        begin
            right := true;
            if first then
            for  i := 1 step 1 until n do
                begin
                comment identity matrix is formed in t;
                    t(i,i) := 1;
                    for j := i+1 step 1 until n do t(i,j) := t(j,i) := 0;
                end
        end;
    tmx := abs(tmx);
    comment computation of the maximum absolute element of a;
    ep := 0;
    for i := 1 step 1 until n do
    for j := 1 step 1 until n do
    if abs(a(i,j)) > ep then ep := abs(a(i,j));
    comment 2 ;
    eps    := ep×1.001₁₀-9;
    eps1   := ep×₁₀-3;
    ep     := ep×₁₀-14;
    first  := true;
    nless1 := n-1;
    comment main loop , tmx iterations;
    for it := 1 step 1 until tmx do
        begin
            eps2 := eps1/10;
            comment compute convergence criteria;
            for i := 1 step 1 until n do
                begin
                    aii := a(i,i);
                    for j := i+1 step 1 until n do
                        begin
                            aij := a(i,j);
                            aji := a(j,i);
                            if (abs(aij-aji) > eps1 and abs(aii-a(j,j)) > eps1)
                               or abs(aij+aji) > eps1 then goto cont
                        end
                end convergence test, all i,j;
            goto next_eps1;
```

```
                comment next transformation begins;
cont:           mark := true;
                for k := 1 step 1 until nless1 do
                for m := k + 1 step 1 until n do
                    begin
                        h := g := hj := yh := 0;
                        d   := a(k,k) - a(m,m);
                        akm := a(k,m);
                        amk := a(m,k);
                        c   := akm + amk;
                        e   := akm - amk;
                        if (abs(e) <= eps2 or abs(d) <= eps2)
                              and abs(c) <= eps2 then goto skip;
                        for i := 1 step 1 until n do
                            begin
                                aik := a(i,k);
                                aim := a(i,m);
                                te  := aik × aik;
                                tee := aim × aim;
                                yh  := yh + te - tee;
                                if i<>k and i<>m then
                                    begin
                                        aki := a(k,i);
                                        ami := a(m,i);
                                        h   := h + aki×ami - aik×aim;
                                        tep := te + ami×ami;
                                        tem := tee + aki×aki;
                                        g   := g + tep + tem;
                                        hj  := hj - tep + tem;
                                    end
                            end i;
                        h   := h + h;
                        if abs(c)<=ep then
                            begin
                            comment take R as identity matrix;
                                cx := 1;
                                sx := 0;
                            end else
                            begin
                            comment compute elements of R;
                                cot2x := d/c;
                                sig   := if cot2x<0 then -1 else 1;
                                cotx  := cot2x+(sig×sqrt(1+cot2x↑↑2));
                                sx    := sig/sqrt(1+cotx↑↑2);
                                cx    := sx × cotx;
                            end;
                        if yh<0 then
                            begin
                                tem := cx;
                                cx  := sx;
                                sx  := -tem;
                            end;
                        cos2x := cx↑↑2 - sx↑↑2;
                        sin2x := 2×sx×cx;
                        d     := d×cos2x + c×sin2x;
                        h     := h×cos2x - hj×sin2x;
                        den   := g + 2×(e×e + d×d);
                        tanhy := (e×d - h/2)/den;
```

```
                    comment compute elements of S;
                    chy := 1/sqrt(1 - tanhy×tanhy);
                    shy := chy×tanhy;
                    comment elements of R×S = T;
                    c1 := chy×cx - shy×sx;
                    c2 := chy×cx + shy×sx;
                    s1 := chy×sx + shy×cx;
                    s2 := shy×cx - chy×sx;
                    comment decide whether to apply this transformation;
                    if abs(s1) > ep or abs(s2) > ep then
                        begin
                        comment at least one transformation is made so;
                            mark := false;
                            comment transformation on the left;
                            for i := 1 step 1 until n do
                                begin
                                    aki    := a(k,i);
                                    ami    := a(m,i);
                                    a(k,i) := c1×aki + s1×ami;
                                    a(m,i) := s2×aki + c2×ami;
                                end left transformation;
                            comment transformation on the right;
                            for i := 1 step 1 until n do
                                begin
                                    aik    := a(i,k);
                                    aim    := a(i,m);
                                    a(i,k) := c2×aik - s2×aim;
                                    a(i,m) := c1×aim - s1×aik;
                                    if right then
                                        begin
                                        comment form right vectors;
                                            tik    := t(i,k);
                                            tim    := t(i,m);
                                            t(i,k) := c2×tik - s2×tim;
                                            t(i,m) := c1×tim - s1×tik;
                                        end
                                end right transformation
                        end;
skip:
                end k,m loops;
            if mark then
                begin
                comment 3 ;
                    if eps2 < ep then goto stop;
                    eps2 := eps2/10;
                    goto cont;
                end else goto new_loop;
next_eps1:
            eps1 := eps1/10;
            comment 4 ;
            if eps1 < eps/2 then
                begin
                    tmx := it - 1;
                    goto done;
                end;
new_loop:
        end it loop;
```

```
    stop:
        first := false;
        tmx   := it-1;
    done:
        if first then
            begin
                comment 5 ;
                for i := 1 step 1 until n-2 do
                    begin
                        mark := false;
                        aii  := a(i,i);
                        for j := i+1 step 1 until n do
                        if abs(aii-a(j,j)) <= eps ∧ abs(a(i,j)-a(j,i)) > eps then
                            begin
                                if mark then goto stop;
                                mark := true;
                                if j = i+1 then goto next;
                                for k := 1 step 1 until n do
                                    begin
                                        aik     := a(i+1,k);
                                        a(i+1,k):= a(j,k);
                                        a(j,k)  := aik;
                                    end;
                                for k := 1 step 1 until n do
                                    begin
                                        aki     := a(k,i+1);
                                        a(k,i+1):= a(k,j);
                                        a(k,j)  := aki;
                                        if right then
                                            begin
                                                tki     := t(k,i+1);
                                                t(k,i+1):= t(k,j);
                                                t(k,j)  := tki;
                                            end;
                                    end;
    next:
                            end;
                    end;
                comment the eigenvalues are placed in the first two columns;
                left := right := false;
                if result then
                for i := 1 step 1 until n do
                    begin
                        if -,right and i < n then
                        left := abs(a(i,i+1)-a(i+1,i)) > eps and
                                abs(a(i,i)-a(i+1,i+1)) <= eps;
                        a(i,1) := if right then a(i-1,1) else a(i,i);
                        a(i,2) := if left  then a(i,i+1) else
                                  if right then -a(i-1,2) else 0;
                        right := left;
                        left  := false;
                    end;
            end;
    end eberlein;
```

comment

1. eberlein solves the eigenproblem for a real matrix by means of a sequence of Jacobi-like transformations.

Input parameters:

n : the order of matrix a.

result : if result is true then in case of convergence the eigenvalues will be placed in the two first columns of matrix a.

Input/Output parameters:

a[1:n,1:n] : at entry the matrix for with the eigenproblem is to be solved.

At exit one of the following three situations can occur:

1) if convergence occurs and result is false :

the real eigenvalues occupy diagonal elements while real and imaginary parts of complex conjugate eigenvalues occupy diagonal and off diagonal corners of 2×2 blocks on the main diagonal.

2) if convergence occurs and result is true :

the eigenvalues will be placed in the two first columns according to the following rules

a real eigenvalue x = a[j,j] makes

```
          a[j,1] = x
and       a[j,2] = 0
```

a complex conjugate pair of eigenvalues

$$x + ixy = a[j,j] + i \times a[j,j+1]$$

and $x - ixy$ makes

```
          a[j,1]   =  x
          a[j,2]   =  y
          a[j+1,1] =  x
and       a[j+1,2] = -y
```

3) if convergence fails no eigenvalues can be calculated as a result of the procedure call. The matrix, a, is equal to the transformed matrix. During a new call of ,eberlein, it is possible to try whether more iterations will result in convergence or not. (first is set to false).

t[1:n,1:n] : if first is false at entry and tmx > 0 then t given at entry is multiplied by the transformation matrix calculated in the procedure.

Eigenvectors of real eigenvalues occupy columns of the transformation matrix. Eigenvectors corresponding to complex conjugate eigenvalues given by

a[j,j] + i×a[j,j+1]
and a[j,j] - i×a[j,j+1]

are formed as

t[k,j] + i×t[k,j+1]
and t[k,j] - i×t[k,j+1]

where k = 1,2,....,n .

tmx : at entry:

the maximum number of transformations performed is abs(tmx). If tmx < 0 then t is unaltered.

at exit tmx records the number of transformations performed.

first : at entry tells whether t is a result of a foregoing transformation or not. (see under t[1:n,1:n]).

at exit first is true if convergence occurs in less than tmx iterations otherwise first is false.

. A dynamical form of the convergence criterion is introduced, which are based on the four reals ep, eps, eps1, and eps2.
In case of convergence of the iteration process the resulting matrix, a satisfies

(abs(a(i, j) - a(j, i)) ≤ eps1
(∨ abs(a(i, i) - a(j, j)) ≤ eps1)
∧ abs(a(i, j) + a(j, i)) ≥ eps1

where eps1 < eps/2

3. If convergence is not obtained and the resulting transformation matrix is the identify matrix then if eps2 = eps2/10 < ep the process is stopped (no solution) otherwise a new transformation is made with eps2 = eps2/10.

4. If eps1 < eps/2 the convergence criterion is fulfilled and the iteration process is stopped. If eps1 ≥ eps the calculation is continued with the new value of eps1.

5. A look up for the eigenvalues is made and at the same time it is controllet whether the resulting matrix 1×1 and 2×2 is on block diagonal form or not.

If the matrix does not consist of 1×1 and 2×2 blocks this is (if possibly) obtained by interchange of rows and columns on the a and t matrices.
Special forms of matrices that fulfil the convergence criterion are not of block diagonal form (with at most two -valid- elements in a row or column) and the procedure eberlein can not solve the eigenproblem for these special matrices;

7. Testprogram.

```
begin
real procedure testnorm(n,A,t,k,complex,x1,x2);
value n,k,complex,x1,x2;
array A,t;
boolean complex;
integer n,k;
real x1,x2;
comment The procedure performs a test of eigenvalues and corresponding
        eigenvectors calculated by procedure eberlein;
    begin
    integer i,j;
    real sum,sum1,sum2,norm;
        sum := norm := 0;
        for i := 1 step 1 until n do
            begin
                sum1 := sum2 := 0;
                for j := 1 step 1 until n do sum1 := sum1+A(i,j)×t(j,k);
                if complex then
                for j := 1 step 1 until n do sum2 := sum2+A(i,j)×t(j,k+1);
                sum := (if complex then
                        (sum1-x1×t(i,k)+x2×t(i,k+1))××2
                       +(sum2-x2×t(i,k)-x1×t(i,k+1))××2
                       else
                        (sum1-x1×t(i,k))××2 ) + sum;
            end;
        for i := 1 step 1 until n do norm := norm+t(i,k)××2;
        if complex then
        for i := 1 step 1 until n do norm := norm + t(i,k+1)××2;
        x1 := if complex then sqrt(x1××2+x2××2) else abs x1;
        testnorm :=sqrt(sum/norm)/x1;
    end procedure testnorm;

integer i,j,n,no,m,layoutno,tmx,total,res;
boolean b1,b2,complex,first,result;
real im,x1,x2,layout;
read_data:
    total := 0;
    i:=read (in,no,n,m,res);
    comment
    no  = example no,
    n   = order of matrix,
    m   = value of tmx,
    res = integer code for result;
    if i <4 then goto stop;
    tmx:=m;
    result := res = 1;
    write (out,<:<10>Example_no:>,<<ddd>,no,
               <:<10><10>Input_parameters:<10><10>n____=:>,<<dddd>,n,
               <:<10>tmx___=:>,tmx,<:<10><10>matrix_a::>);
```

```
        begin
        array a,t,A(1:n,1:n);
            read (in,layoutno);
            comment layout no serves a choice between three layouts in
                    output of the unaltered matrix a;
            layout := case layoutno of (real<<  -ddd>,
            real<<  -d.ddd>,real<<  dd>,real<<-d.d> );
            write (out,<:<10>:> );
            for j := 1 step 1 until n do
                begin
                    write (out,<:<10>:> );
                    for i := 1 step 1 until n do
                        begin
                        comment input and output of matrix a;
                            read (in,a(j,i));
                            A(j,i) := a(j,i);
                            write (out,string layout,a(j,i));
                        end
                end;
            first := true;
new_eber:
            eberlein(n,a,t,tmx,first,result);
            write (out,<:<10><10><10>Results:<10><10>tmx  =:> ,<<-ddd> ,tmx);
            if first then
            write (out,<:<10>first = true:> ) else
            write (out,<:<10>first = false:> );
            if -,result or -,first then
                begin
                    write (out,<:<10><10><10>Limiting matrix after:>,
                            <<ddd> ,total+tmx,<: iterations<10>:> );
                    for j := 1 step 1 until n do
                        begin
                            write (out,<:<10>:> );
                            for i := 1 step 1 until n do
                            write (out,<<  -d.dd10+dd> ,a(j,i));
                        end
                end else
                begin
                    write (out,<:<10><10><10>Eigenvalues after:>,
                            <<dddd> ,total+tmx,<: iterations<10>:> );
                    for i := 1 step 1 until n do
                        begin
                            write (out,<:<10>:> ,<<dd> ,i,
                                    <<  -dddd.dddddddddd> ,a(i,1));
                            if a(i,2) <> 0 then
                            write (out,<<  +dddd.dddddddddd> ,a(i,2),<:x:> );
                        end;
                    if m = 0 then goto next;
                    write (out,<:<10><10><10>Eigenvectors:<10>:> );
                    b1 := b2 := false;
                    for i := 1 step 1 until n do
                        begin
                            write(out,<:<10>:> ,<<dd> ,i,<:<10>:> );
                            b1 := -,b2 and a(i,2) <> 0;
```

```
                        for j := 1 step 1 until n do
                            begin
                                im := if b2 and m>0 then t(j,i-1) else
                                      if b2 and m<0 then t(i-1,j) else
                                      if m>0 then t(j,i) else t(i,j);
                                write (out,<<____-dddd.dddddddddd>,im);
                                im := if b2 and m>0 then -t(j,i) else
                                      if b2 and m<0 then -t(i,j) else
                                      if b1 and m>0 then t(j,i+1) else
                                      if b1 and m<0 then t(i+1,j) else 0;
                                if b1 or b2 then
                                write (out,<<  +dddd.dddddddddd>,im,<:x:>);
                                write (out,<:<10>:>);
                            end;
                        b2 := b1;
                        b1 := false;
                    end;
                write (out,<:<10><10><10><10>Testnorm for corresponding:>,
                           <: eigenvalues and eigenvectors<10><10><10>:>,
                           <:no. of eigenvalue_____testnorm<10>:>,
                           <:<10>:>);
                b1 := false;
                for i := 1 step 1 until n do
                    begin
                        x1 := a(i,1);
                        x2 := a(i,2);
                        complex := x2 <> 0;
                        b1 := -,b1 and complex;
                        if b1 or -,complex then
                            im := testnorm(n,A,t,i,complex,x1,x2);
                        if b1 then
                            begin
                            write(out,<:<10>:>,<<__-dd>,i,<:__and:>,i+1,
                                      <:__________:>,<<d.d$_{10}$+dd>,im);
                            end else
                        if -, complex then
                            begin
                            write(out,<:<10>:>,<<__-dd>,i,
                                      <:_____:>,
                                      <<d.d$_{10}$+dd>,im);
                            end;
                    end;
            end;
next:
            if-, first and tmx = m then
                begin
                    total := total+tmx;
                    tmx   := m;
                    if total <= 3xm then goto new_cber;
                end;
            write(out,<:<12>:>);
        end;
    goto read_data;
stop:
end testprogram
```