De serie AP 200

Inhoud dd. 1-9-1962

AP 200	ZERO
AP 201	MAX
AP 202	PROD
AP 203	INT
AP 204	DET
AP 205	SOL
AP 206	INV
AP 207	DETSOL
AP 208	DETINV
AP 209	DSBAND
AP 210	PSP1
AP 211	PREP
AP 212	SEIVA
AP 213	SEIVEC
AP 214	TRASF1
AP 215	SYMEVE
AP 216	CSQRT
AP 217	CZERO
AP 218	CPROD
AP 219	CPOL

De serie AP 200

Inhoud dd. 1-12-1962

AP 200 - AP 220 zie inhoud dd. 1-9-1962

AP 220 SYMDET

AP 221 SYMSOL

AP 222 SYMINV

AP 223 syminv

AP 200 Series
==========
(Contents d.d. august 1963)
(For AP 200 - 223 see Contents d.d. december 1962)

AP 224 SYMDET1
225 SYMSOL1
226 SYMINV1
227 syminv1
228 SYMDET2
229 SYMSOL2
230 ZERO
231 SPAP
232 SEIGENVA
233 SEIGENVEC
234 STRASF
235 SEVAVEC
236 ZEREX
237 POL
238 APAP
239 REIGENVA
240 REIGENVEC
241 ATRASF
242 REVAVEC

AP 200 Series

(Contents d.d. January 1964)

(For AP 200 - 242 see Contents d.d. August 1963)

AP 243 – 244 – 245 — Complete elliptic integrals of the first and second kind

AP 246 – 247 – 248 — Incomplete elliptic integrals of the first and second kind

AP 249 – 250 — Jacobian elliptic functions

Procedures in Algol 60

Inleiding

In de serie AP met nummering beginnend bij 200 worden enige in Algol-60 geschreven procedures gepubliceerd. Zij kunnen in het MC-Algol-systeem gebruikt worden. Men bedenke echter, dat deze procedures niet zonder declaratie ter beschikking staan, maar op normale wijze gedeclareerd moeten worden.

De tekst van elke procedure-declaratie wordt voorafgegaan door een commentaar omtrent effect, mee te geven parameters en eventueel methode. Bovendien wordt in dit commentaar vermeld welke non-locale procedures in de betreffende procedure-declaratie voorkomen, behalve de in het Algol-60-rapport [1] section 3.2.4 en 3.2.5 genoemde standaard-functies. De gebruikte non-locale procedures zijn ofwel procedures uit de serie AP 200 ofwel standaard procedures, die tot het complex behoren of als MCP beschikbaar zijn.

Acknowledgements

I wish to express my gratitude to the Staff members of the Computation Department of the Mathematical Centre, especially to Prof. Dr. Ir. A. van Wijngaarden, J.A. Zonneveld and the former collaborator Dr. E.W. Dijkstra. In fact, they are the authors of both versions of ZERO, AP 200 and AP 230, which have been completed after ample discussions about the organization, resulting (with AP 230) in a system-independent formulation of the process.

Moreover, their suggestions have been of great influence on the organization and formulation of the other procedures, especially of those that calculate the eigenvectors and eigenvalues of matrices.

I am also indebted to Dr. J.H. Wilkinson for his valuable suggestions and criticism.

Finally, I thank J.A. Zonneveld for reading the comments of the procedures and for suggesting various improvements.

Amsterdam, August 1963

The Mathematical Centre

T.J. Dekker

```
comment          AP 200

        ZERO:= x:= a zero of fx between a and b. The real fx must be an expression depending on x with
different sign for x = a and x = b. In the array e[1:2] must be given the relative error e[1] and the
absolute error e[2]. The method is a combination of linear interpolation or extrapolation and bisection.
The process ends if a zero x has been found within a tolerance ≤ abs (x × e[1]) + e[2]. For a simple
zero the process is of the order 1.6;

real procedure ZERO (x,a,b,fx,e);

value a,b; real x,a,b,fx; array e;

begin   real c,fa,fb,fc,re,ae;

        re:= e[1]; ae:= e[2];

        x:= a; fa:= fx; x:= b; fb:= fx; goto entry;

begin:  if abs (a - b) ≤ fa then a:= b + sign (c - b) × fa;

        if sign (a - x) = sign (b - a) then x:= a;

        a:= b; fa:= fb; b:= x; fb:= fx;

        if sign (fc) = sign (fb) then

entry:  begin c:= a; fc:= fa end;

        if abs (fb) > abs (fc) then

        begin a:= b; fa:= fb; b:= c; fb:= fc; c:= a; fc:= fa end;

        a:= (a × fb - b × fa) / (fb - fa);

        x:= (c + b) / 2; fa:= abs (b × re) + ae;

        if abs (x - b) > fa then goto begin;

        ZERO:= x

end ZERO;
```

The Series AP 200 of Procedures in ALGOL 60

Introduction

This series has been written primarily for the users of the ALGOL 60 system, developed for the Electrologica X1 computer. The procedures published have been tested on this system. They can be used in ALGOL programs by inserting the procedure declarations in the ordinary way.

The text of each procedure declaration is preceded by a comment, giving details about its effect and parameters and describing the method. Moreover, in these comments the non-local procedures used are mentioned, except the standard functions listed in the (Revised) Report on ALGOL 60, sections 3.2.4 and 3.2.5. The non-local procedures are either procedures written in machine code, in which case they can be called without being declared, or they are procedures published in the present series AP 200.

Summary of AP 200 – 242

At this moment the procedures AP 200 – 242 have been completed. Some of them are new versions, written mainly for the sake of embellishment and elucidation, of procedures that did already exist. The existing versions, however, have not appeared to be incorrect: they are still useful.

In more detail:

Beside AP 200 ZERO there is now a new version AP 230 ZERO, bearing the same name because in any existing program these procedures are interchangeable.

The procedures AP 224 – 229, however, being new versions of AP 220 – 223, bear new names because they cannot replace the old versions in an existing program. AP 224 – 227 contain only slight modifications, but AP 228 – 229 require the parameters to be given in another way; the latter have been written so that they are useful for matrices given in an integer array.

(AP 204 – 208 and 224 – 229 have been published also in Report MR 63, where the methods used and the organization are dealt with; in that report, moreover, some details of the ALGOL 60 system for the Electrologica X1 computer are mentioned.)

AP 231 – 235, being new versions of AP 210 – 215, bear new names because these, too, cannot replace the old versions in an existing program. They contain several improvements and enable the user to obtain more information during the execution of the processes involved, with the possibility of influencing the process if so desired.

As to the nomenclature, the initial letter of the procedure identifier gives some information about the kind of matrices for which the procedure is intended, as follows:

S usually denotes symmetric matrices
C denotes complex arithmetic
A denotes asymmetric matrices having real or complex eigenvalues
R denotes asymmetric matrices having real eigenvalues only.

comment AP 201

MAX:= the maximal value of the expression fk, where fk is computed for k:= a step 1 until b. Moreover k:= the index value for which this maximum has been found. if a > b then k:= a and the value of MAX is undefined;

```
real procedure MAX(k,a,b,fk);

value a,b; integer k,a,b; real fk;

begin real r,s;

MA    : k:= a; if k ≤ b then MAX:= s:= fk; goto MC;

MB    : k:= k + 1; r:= fk;

        if r > s then begin MAX:= s:= r; a:= k end;

MC    : if k < b then goto MB; k:= a

end MAX;
```

comment AP 202

PROD:= the product of the values of fk, where the expression fk is computed for k:= a step 1 until b. if a > b then PROD:= 1;

```
real procedure PROD(k,a,b,fk);

value a,b; integer k,a,b; real fk;

begin   real p; p:= 1;

        for k:= a step 1 until b do p:= fk × p;

        PROD:= p

end PROD;
```

```
comment          AP 203

        INT:= integral from a to b of y.dx. The parameter eps determines the absolute precision of the answer.
The method is Simpson's rule, which is carried out each time over an interval 2 × h and over two intervals h.
If the difference exceeds eps × h then the interval is halved and the integration is repeated.
If the difference is smaller than eps × h / 32 then h is doubled. From the difference a correction term
is computed and added, so that the error is not of the order h ↑ 4 but of the order h ↑ 6.
The total amount of corrections does not exceed eps × abs (b - a) / 180;

real procedure INT (x, a, b, y, eps); value a, b, eps; real x, a, b, y, eps;

begin   boolean ls; real h, F, V, f0, f4, I;

        I:= 0; x:= a; f0:= y;

   m1:  h:= (b - a)/ 4; ls:= true;

   m2:  x:= a + h; F:= f0 + 4 × y;

        x:= x + h; V:= y; F:= F + 2 × V;

        x:= x + h; F:= F + 4 × y;

        x:= x + h; f4:= y; F:= F + f4;

        V:= 8 × V + 2 × (f0 + f4) - F;

        if abs (V) ≥ eps then begin h:= h/2; ls:= false end else

        begin   a:= x; f0:= f4; I:= h × (F - V/15) + I;

                if ls then goto m3;

                if abs (V) ≤ eps/32 then h:= 2 × h;

                if sign (b - a - 4 × h) ≠ sign (h) then goto m1

        end;

        goto m2;

   m3:  INT:= I/3

end INT;
```

```
comment          AP 204

        DET:= determinant of the n-th order matrix A which is given as array A[1: n,1: n]. The method is Crout with
row interchanges: A is replaced by its triangular decomposition L × U with all U[k,k] = 1. The integer array p[1: n]
is an output vector giving the pivotal row indices. The k-th pivot is chosen in the k-th column of L such that
abs (L[i,k]) / row norm is maximal.  DET uses the non-local real procedure INPROD;

real procedure DET (A,n,p); value n; integer n; array A; integer array p;

begin   integer i,j,k; real d,r,s; array v[1: n];

        for i:= 1 step 1 until n do v[i]:= sqrt (INPROD (j,1,n,A[i,j],A[i,j]));

        d:= 1;

        for k:= 1 step 1 until n do

        begin   r:= - 1;

                for i:= k step 1 until n do

                begin   A[i,k]:= A[i,k] - INPROD (j,1,k - 1,A[i,j],A[j,k]);

                        s:= abs (A[i,k]) / v[i];

                        if s > r then begin r:= s; p[k]:= i end

                end LOWER;

                v[p[k]]:= v[k];

                for j:= 1 step 1 until n do

                begin   r:= A[k,j]; A[k,j]:= if j ≤ k then A[p[k],j] else

                        (A[p[k],j] - INPROD (i,1,k - 1,A[k,i],A[i,j])) / A[k,k];

                        if p[k] ≠ k then A[p[k],j]:= - r

                end UPPER;

                d:= A[k,k] × d

        end LU;

        DET:= d

end DET;
```

```
comment          AP 205

        SOL should be preceded by a call of DET,which yields the array LU[1: n,1: n] in triangularly decomposed
form and the integer array p[1: n] of pivotal row indices. SOL replaces the vector b which is given as
array b[1: n] by the solution x of the linear system L × U × x = b. SOL leaves the elements of LU and p
unaltered, hence after one call of DET several calls of SOL are allowed.
SOL uses the non-local real procedure INPROD;

procedure SOL (LU,b,n,p); value n; integer n; array LU,b; integer array p;

begin   integer i,k; real r;

        for k:= 1 step 1 until n do
        begin   r:= b[k];
                b[k]:= (b[p[k]] - INPROD (i,1,k - 1,LU[k,i],b[i])) / LU[k,k];
                if p[k] ≠ k then b[p[k]]:= - r
        end;
        for k:= n step - 1 until 1 do
        b[k]:= b[k] - INPROD (i,k + 1,n,LU[k,i],b[i])
end SOL;
```

```
comment          AP 206

        INV should be preceded by a call of DET, which yields the array LU[1: n,1: n] in triangularly decomposed form
and the integer array p[1: n] of pivotal row indices. INV replaces LU by the inverse matrix of L × U.
INV uses the non-local real procedure INPROD;

procedure INV (LU,n,p); value n; integer n; array LU; integer array p;

begin   integer i,j,k; real r; array v[1: n];

        for k:= n step - 1 until 1 do

        begin   for j:= k + 1 step 1 until n do

                begin   v[j]:= LU[k,j]; LU[k,j]:= 0 end;

                LU[k,k]:= 1 / LU[k,k];

                for j:= k - 1 step - 1 until 1 do

                LU[k,j]:= - INPROD (i,j + 1,k,LU[k,i],LU[i,j]) / LU[j,j];

                for j:= 1 step 1 until n do

                LU[k,j]:= LU[k,j] - INPROD (i,k + 1,n,v[i],LU[i,j])

        end;

        for k:= n step - 1 until 1 do

        begin   if p[k] ≠ k then for i:= 1 step 1 until n do

                begin   r:= LU[i,k]; LU[i,k]:= - LU[i,p[k]]; LU[i,p[k]]:= r end

        end

end INV;
```

```
comment          AP 207

        DETSOL:= determinant of the n-th order matrix A
which is given as array A[1:n,1:n]. Moreover the
vector b which is given as array b[1:n] is replaced by
the solution x of the linear system A × x = b.
DETSOL uses DET and SOL;

real procedure DETSOL (A,b,n);

value n; integer n; array A,b;

begin   integer array p[1:n];
        DETSOL:= DET (A,n,p); SOL (A,b,n,p)

end DETSOL;
```

```
comment          AP 208

        DETINV:= determinant of the n-th order matrix A
which is given as array A[1:n,1:n]. Moreover the
matrix A is replaced by its inverse.
DETINV uses DET and INV;

real procedure DETINV (A,n);

value n; integer n; array A;

begin   integer array p[1:n];

        DETINV:= DET (A,n,p); INV (A,n,p)

end DETINV;
```

```
comment          AP 209

        DSBAND:= determinant van de bandmatrix M, als n > b bovendien N:= M ↑ (- 1) × N, dat is de oplossing X van
het lineaire stelsel M × X = N. De mee te geven integer parameters zijn: m = orde, l = aantal diagonalen links van
de hoofddiagonaal, b = bandbreedte, n = b + aantal rechterleden. Deze parameters moeten voldoen aan: b ≤ m en
0 ≤ l < b ≤ n. De bandmatrix M en de matrix N der rechterleden worden als volgt in het array A[1:m,1:n] meegegeven.
In de eerste b kolommen van A zijn gegeven de elementen van M volgens de formule:
A[i,j] = M[i, if i ≤ l + 1 then j else if i ≤ m - b + l then j + i - l - 1 else j + m - b] voor j ≤ b,
de andere elementen van M zijn 0. De laatste n - b kolommen van A zijn de rechterleden, in
formule: A[i,j] = N[i,j - b] voor j > b.
DSBAND gebruikt SUM, PROD en MAX;

real procedure DSBAND (A,m,l,b,n); value m,l,b,n; integer m,l,b,n; array A;

begin   integer i,j,k; real w;

ELIM:   for k:= 1 step 1 until m do

        begin   l:= if k + b ≤ m then l + 1 else m; MAX (i,k,l,abs (A[i,1]));

                if i ≠ k then for j:= 1 step 1 until n do

                begin w:= - A[i,j]; A[i,j]:= A[k,j]; A[k,j]:= w end;

                for i:= k + 1 step 1 until l do

                begin   w:= A[i,1] / A[k,1];

                        for j:= 2 step 1 until n do

                        A[i, if j ≤ b then j - 1 else j]:= A[i,j] - w × A[k,j];

                        A[i,b]:= 0

                end

        end ELIM;

        DSBAND:= PROD (k,1,m,A[k,1]);

BACK:   for j:= b + 1 step 1 until n do for k:= m step - 1 until 1 do

        A[k,j]:= (A[k,j] - SUM (i,2, if k + b ≤ m then b else m - k + 1,A[k,i] × A[k + i - 1,j]))/A[k,1]

end DSBAND;
```

```
comment          AP 210

        PSP1 transforms an n-th order symmetric matrix into triple diagonal form
according to a method of A.S.Householder(litt. J.H.Wilkinson, Comp.J.3(1960)23-27).
The matrix is defined by means of the actual parameters for A, i and j as follows:
The actual parameter for A must be a subscripted real variabele which for each i and j
satisfying 1 ≤ i ≤ j ≤ n is the (i,j)-th element of the matrix.(So only the upper
triangle of the matrix need be given.)
        The results are delivered in A and in the output-arrays B,BB,D[1:n] as follows:
The main diagonal is delivered in A and in D. The codiagonal elements are delivered in B
and the squares of these elements in BB. Moreover B[n]:= BB[n]:= 0. The vectors defining
the subsequent transformations are delivered in A.
PSP1 uses SUM ;

procedure PSP1 (A,i,j,n,B,BB,D); value n; integer i,j,n; real A; array B,BB D;

begin   integer p,r; real w,x;

HA:     for r:= 1 step 1 until n do

        begin   j:= i:= r; D[r]:= A; BB[r]:= SUM (j,r+1,n,A ↑ 2);

                B[r]:= sqrt (BB[r]); if B[r] < ₁₀-200 then goto HB;

                j:= r+1; if A > 0 then B[r]:= -B[r]; A:= A-B[r]; w:= A × B[r];

                for j:= r+1 step 1 until n do D[j]:= A;

                for p:= r+1 step 1 until n do

                begin   j:= p;

                        B[p]:= (SUM (i,r+1,p-1,A × D[i]) + SUM (j,p,n,A × D[j]))/w

                end;

                x:= SUM (p,r+1,n,D[p] × B[p])/(2 × w);

                for j:= r+1 step 1 until n do B[j]:= D[j] × x + B[j];

                for i:= r+1 step 1 until n do for j:= i step 1 until n do

                A:= D[i] × B[j] + B[i] × D[j] + A;

        HB:

        end

end PSP1;
```

```
comment          AP 211

        PREP , which should be involved before SEIVA, carries out some
prepatory substitutions in the auxiliary array E[1:5]. In detail :
E[3] := 1.1 × an upper bound s  of the absolute value of the eigenvalues
of the n-th order symmetric triple diagonal matrix with main diagonal D
and codiagonal B, the relative error E[1] := 0 and
the absolute error E[2] := eps × s ;

procedure PREP (D,B,n,eps,E); value n; integer n; real eps; array D,B,E;

begin   integer r; real s,x;

        s := 0;

        for r := 1 step 1 until n do
        begin   x := (if r = 1 then 0 else abs (B[r-1]))
                + abs (D[r]) + abs (B[r]);

                if x > s then s := x

        end;

        E[1] := 0; E[2] := eps × s; E[3] := 1.1 × s

end PREP;
```

```
comment          AP 212

        SEIVA := next eigenvalue of the n-th order symmetric triple diagonal matrix
with main diagonal D and the squares of the codiagonal elements in BB.
For the method used see W.Givens, NBS.-AMS.29(1953)117-122.
If there occur vanishing or sufficiently small elements on the codiagonal, the
matrix is subdivided into submatrices with non-vanishing codiagonal elements.
The parameter E is an auxiliary array E[1:5] , the parameters n1,n2 and k are
integer variables which in these order indicate the lowest and the highest
index of the subsequent submatrices and the number of eigenvalues computed.
One gives E[1], E[2] and E[3] suitable starting values by means of PREP, and k
must have the starting value 0. The values of n1,n2,E[4],E[5] need not be given.
SEIVA uses EVEN and ZERO ;

real procedure SEIVA (D,BB,n,n1,n2,k,E); value n; integer n,n1,n2,k; array D,BB,E;
begin   integer r,t; real x;
        real procedure SDET (q,q2); value q,q2; integer q,q2;
        begin   integer p; real d0,d1,d2;
                p:= 0; SDET:= E[5];
                d1:= t; d2:= (x-D[q]) × d1; goto DB;
        DA:     q:= q+1; d0:= d1; d1:= d2; d2:= (x-D[q]) × d1 - BB[q-1] × d0;
        DB:     if d2 > 0 = d1 ≤ 0 then p:= p+1;
                if p ≤ r then
                begin   if q < q2 then goto DA;
                        if x < E[4] then E[4]:= x;
                        if abs (d2) > E[5] then E[5]:= abs (d2);
                        SDET:= d2
                end
        end SDET;
GA:     if k = 0 then n2:= 0;
        if k = n2 then
        begin   E[4]:= E[3]; E[5]:= 0; n1:= n2+1;
        GC:     n2:= n2+1; if 1-BB[n2]/E[3] ↑ 2 ≠ 1 then goto GC
        end;
        k:= k+1; r:= k-n1+1; t:= EVEN (r);
        SEIVA:= ZERO (x,E[4],-E[3],SDET (n1,n2) E)
end SEIVA;
```

```
comment          AP 213

        SEIVEC computes the eigenvector V of the n-th order symmetric triple diagonal matrix with
main diagonal D and codiagonal B belonging to the eigenvalue x of the submatrix with lowest index n1
and highest index n2. The vector V must be declared with two extra elements, scil. array V[0:n+1].
SEIVEC uses SUM ;

procedure SEIVEC (D,B,n,n1,n2,x,V);

        value n,n1,n2,x; integer n,n1,n2; real x; array D,B,V;

begin   integer i,p,q; real x1;

WA:     p:= n1-1; q:= n2+1; V[p]:= V[q]:= 1;

WB:     i:= p:= p + 1; if p = n2 then goto WD;

        V[p]:= (if p = n1 then (x - D[p]) else ((x - D[p]) × V[p - 1] - B[p - 1] × V[p - 2])) / B[p];

        if abs (V[p]) ≥ abs (V[p - 1]) then goto WB;

        if p ≥ q then goto WD;

WC:     i:= q:= q - 1; if q = n1 then goto WD;

        V[q]:= (if q = n2 then (x - D[q]) else ((x - D[q]) × V[q + 1] - B[q] × V[q + 2])) / B[q - 1];

        if abs (V[q]) ≥ abs (V[q + 1]) then goto WC;

        if p < q then goto WB;

WD:     V[i]:= 1/sqrt (SUM (p, n1 - 1, i - 2, V[p] ↑ 2)/V[i - 1] ↑ 2 + 1

                + SUM (p, i + 2, n2 + 1, V[p] ↑ 2)/V[i + 1] ↑ 2);

        x1:= V[i]/V[i - 1]; for p:= i - 1 step -1 until n1 do V[p]:= V[p - 1] × x1;

        x1:= V[i]/V[i + 1]; for p:= i + 1 step  1 until n2 do V[p]:= V[p + 1] × x1;

        for p:= 1 step 1 until n1 - 1, n2 + 1 step 1 until n do V[p]:= 0

end SEIVEC;
```

```
comment          AP 214

        TRASF1 transforms the eigenvector V of the by means of PSP1 transformed
matrix into the corresponding eigenvector of the original matrix.
TRASF1 uses SUM ;

procedure TRASF1 (A,i,j,n,B.V); value n; integer i,j,n; real A; array B.V;

begin   real x1,f1;

        for i:= n-1 step -1 until 1 do
        begin   if abs (B[i]) ≥ ₁₀-200 then
                begin   j:= i+1; x1 := A;
                        f1:= SUM (j,i+1,n,AXV[j])/(x1XB[i]);
                        for j:= i+1 step 1 until n do V[j]:= Axf1+V[j]
                end
        end
end TRASF1;
```

```
comment          AP 215

        SYMEVE computes the eigenvalues and eigenvectors of the symmetric
matrix A of order n. The actual parameter for A must be a subscripted real
variable depending on the actual parameters for i and j in such a way that
for i and j satisfying 1 ≤ i ≤ j ≤ n the variable A is the (i,j)-th element
of the matrix. The eigenvalues are computed within a tolerance eps × s,
where s is an upper bound for the eigenvalues computed in PREP. The proce-
dures OVA and OVEC serve to deliver each time an eigenvalue or eigenvector
respectively.
SYMEVE uses PSP1,PREP.SEIVA,SEIVEC.TRASF1;

procedure SYMEVE (A,i,j,n,eps,OVA.OVEC);
        value n; integer i,j,n; real A,eps; procedure OVA,OVEC;
begin   integer k,n1,n2,p; real x; array B,BB D[1:n],V[0:n+1].E[1:5];
        PSP1 (A i,j.n,B.BB,D); PREP (D,B,n,eps,E); k:= 0;
next:   x:= SEIVA (D,BB,n,n1,n2,k,E); OVA (x);
        SEIVEC (D,B,n,n1.n2,x,V); TRASF1 (A,i,j.n,B,V); OVEC (p,1,n,V[p]);
        if k < n then goto next
end SYMEVE;
```

```
comment          AP 216

        CSQRT:= rp:= real part and ip:= imaginary part of a complex square root
of the complex number with real part a and imaginary part b.
The answer is situated in the halfplane rp ≥ 0;

real procedure CSQRT (a,b,rp,ip); value a,b; real a,b,rp,ip;

begin   rp:= sqrt ((abs (a) + sqrt (a × a + b × b)) / 2.0);
        ip:= b:= b / (2.0 × rp);
        if a < 0 then begin ip:= if b ≥ 0 then rp else -rp; rp:= abs (b) end;
        CSQRT:= rp
end     CSQRT;
```

```
comment         AP 217

        CZERO:= x:= real part  and y:= imaginary part  of a zero of the function with real part r and
imaginary part s. In array  e[1:3]  the desired tolerances must be given. Moreover  one must give the
variables x and y estimated starting values.
The method is due to D.E.Muller (litt. MTAC 10 (1956) 208 - 215).
The process ends if two subsequent iterates agree within sqrt ((x ↑ 2 + y ↑ 2) × e[1] ↑ 2 + e[2] ↑ 2)
and the modulus of the function value is smaller than or equal to e[3].
CZERO uses CSQRT;

real procedure CZERO (x,y,r,s,e); real x,y,r,s; array e;
begin   real a,b,c,d,e1,e2,e3,g,h,r0,r1,r2,s0,s1,s2,t,u;
        e1:= e[1] ↑ 2; e2:= e[2] ↑ 2; e3:= e[3] ↑ 2; a:= x;
        g:= sqrt (a × a + y ↑ 2) × .1 + .1; h:= 0; c:= -.5; d:= 0;
        x:= a + g; r0:= r; s0:= s; x:= a - g; r1:= r; s1:= s;
        r0:= r0 - r1; s0:= s0 - s1; x:= a;
        r2:= r; s2:= s;
LL:     r1:= r1 - r2; s1:= s1 - s2;
        t:= r0 × c - s0 × d - r1; u:= r0 × d + s0 × c - s1;
        a:= (t - r1) × c - (u - s1) × d - r1; b:= (t - r1) × d + (u - s1) × c - s1;
        r0:= t × c - u × d; s0:= t × d + u × c;
        t:= -2.0 × ((1.0 + c) × r2 - d × s2); u:= -2.0 × ((1.0 + c) × s2 + d × r2);
        CSQRT (a × a - b × b + 2.0 × (t × r0 - u × s0),2.0 × (a × b + t × s0 + u × r0),c,d);
        if a × c + b × d < 0 then begin c:= - c; d:= - d end;
        a:= a + c; b:= b + d;
        c:= (a × t + b × u) / (a × a + b × b); d:= (a × u - b × t) / (a × a + b × b);
        a:= sqrt (c × c + d × d) / 10;
        if a > 1 then begin c:= c/a; d:= d/a end;
        a:= g × c - h × d; h:= g × d + h × c; g:= a; x:= x + g; y:= y + h;
        r0:= r1; s0:= s1; r1:= r2; s1:= s2; r2:= r; s2:= s;
        if g × g + h × h > (x ↑ 2 + y ↑ 2) × e1 + e2 ∨ r2 × r2 + s2 × s2 > e3 then goto LL;
        CZERO:= x
end     CZERO;
```

```
comment          AP 218

        CPROD:= rp:= real part and ip:= imaginary part of the product over k from
a until b of the complex numbers with  real part rk and  imaginary part sk, where
rk and sk are expressions depending on k;

real procedure CPROD (k,a,b,rk,sk,rp,ip); value b; integer k,a,b; real rk,sk,rp,ip;

begin   real p,q,r,s; p:= 1; ip:= q:= 0;

        for k:= a step 1 until b do
        begin   r:= rk; s:= sk; ip:= p × s + q × r;
                p:= p × r - q × s; q:= ip
        end;
        CPROD:= rp:= p
end     CPROD;
```

```
comment          AP 219

        CPOL:= rp:= real part and  ip:= imaginary part  of the n-th degree polynomial
defined by: sigma over k from 0 until n of A × z ↑ (n-k), where z denotes the complex
argument with real part x  and imaginary part y. The polynomial has real coefficients
which are the successive values of the expression A depending on k;

real procedure CPOL (A,k,n,x,y,rp,ip); value x,y,n; real A,x,y,rp,ip; integer k,n;

begin   real p,q,b0,b1,b2;

        p:= 2 × x; q:= x × x + y × y; b2:= b1:= 0;

        for k:= 0 step 1 until n - 1 do

        begin b0:= b1; b1:= b2; b2:= p × b1 - q × b0 + A end;

        ip:= y × b2; k:= n; CPOL:= rp:= x × b2 - q × b1 + A

end     CPOL;
```

```
comment          AP 220

        SYMDET:= determinant of the n-th order symmetric positive definite matrix M which
is defined by the actual parameters in the following way:
A is a subscripted real variable which for each i and j satisfying  1 ≤ i ≤ j ≤ n  is the
(i,j)-th element of M. (Thus one needs to give only the upper triangle of M.)
The method used is the square root method of  Cholesky (litt. NBS.-AMS. 39 pag.31), which
yields an upper triangular matrix U with the property: U transpose × U = M.
SYMDET replaces the upper-diagonal elements of M  by the upper-diagonal elements of U and
the diagonal elements of M by the inverses of the diagonal elements of U.
SYMDET uses SUM;

real procedure SYMDET (A,i,j,n); value n; integer i,j,n; real A;

begin   integer k; real d,r; array v[1:n];

        d:= 1;

        for k:= 1 step 1 until n do

        begin   j:= k; for i:= 1 step 1 until k do v[i]:= A;

                r:= v[k] - SUM (i,1,k-1,v[i] ↑ 2);

                d:= r × d; i:= k; A:= r:= 1 / sqrt (r);

                for j:= k+1 step 1 until n do

                begin   i:= k; A:= (A - SUM (i,1,k-1,A × v[i])) × r end

        end     IU;

        SYMDET:= d

end     SYMDET;
```

```
comment          AP 221

        SYMSOL must be preceded by a call of SYMDET, which yields an upper triangular
matrix U satisfying: U transpose × U = M.
SYMSOL replaces vector b which is given as array b[1: n], by the solution vector x of
the linear system M × x = b. For the definition of the matrices  M and U  by means of
the actual parameters for A, i, j and n see comment of SYMDET.
SYMSOL leaves the elements A unaltered, hence after one call of SYMDET  several calls
of SYMSOL are allowed.
SYMSOL uses SUM ;

procedure SYMSOL (A,i,j,n,b); value n; integer i,j,n; real A; array b;

begin   for j:= 1 step 1 until n do b[j]:= (b[j] - SUM (i,1,j-1,A × b[i])) × A;

        for i:= n step -1 until 1 do

        begin   j:= i; b[i]:= A × (b[i] - SUM (j,i+1,n,A × b[j])) end

end     SYMSOL;
```

```
comment          AP 222

        SYMINV must be preceded by a call of SYMDET, which yields an upper triangular
matrix U satisfying: U transpose x U = M.
SYMINV replaces the matrixelements A  by the corresponding elements of the inverse of
the original  matrix M. For the definition  of the matrices  M and U  by means of the
actual parameters see comment of SYMDET.
SYMINV uses SUM ;

procedure SYMINV (A,i,j,n); value n; integer i,j,n; real A;

begin   integer k; array v[1:n];

        for k:= 1 step 1 until n do
        begin   i:= j:= k; v[k]:= A;
                for j:= k+1 step 1 until n do
                begin   i:= k; A:= v[j]:= - SUM (i,k,j-1,A x v[i]) x A end;
                for i:= 1 step 1 until k do
                begin   j:= k; A:= SUM (j,k,n,A x v[j]) end
        end
end     SYMINV;
```

```
comment          AP 223

        syminv should be preceded by a call of SYMDET yielding an upper triangular
matrix U with the property: U transpose × U = M.
syminv replaces the diagonal elements A  by the corresponding diagonal elements of
the inverse of the original matrix M.
For the definition of the matrices M  and U  by means of the actual parameters see
comment of SYMDET.
syminv uses SUM ;

procedure syminv (A,i,j,n); value n; integer i,j,n; real A;
    begin   integer k; array v[1:n];
            for k:= 1 step 1 until n do
            begin   i:= j:= k; v[k]:= A;
                    for j:= k+1 step 1 until n do v[j]:= - SUM (i,k,j-1,A × v[i]) × A;
                    i:= j:= k; A:= SUM (j,k,n,v[j] ↑ 2)
            end
    end     syminv;
```

comment AP 224

SYMDET1:= determinant of the n-th order symmetric positive definite matrix M which is defined as follows: the actual parameter for A – being a subscripted real variable whose indices (or index) depend(s) on the actual parameters for i and j – is the (i, j)-th element of M for each i and j satisfying $1 \leq i \leq j \leq n$. Thus one needs to give only the upper triangle of M. In order to avoid waste of space, one may give this triangle in a one-dimensional array. E.g., if the upper triangle of M is given in array C[1 : n × (n + 1) : 2] columnwise, i.e. the columns one after the other, and the successive values (j − 1) × j : 2 have been recorded in an auxiliary integer array J[1 : n], then the appropriate call of SYMDET1 reads:

SYMDET1 (C[i + J[j]], i, j, n).

The method used is the square root method of Cholesky, yielding an upper triangle which, premultiplied by its transpose, gives the original matrix. SYMDET1 replaces the elements of M by the corresponding elements of this upper triangle. It uses the non-local real procedure SUM (= AP 119);

```
real procedure SYMDET1 (A,i,j,n); value n; integer i,j,n; real A;
begin   integer k; real d,r; array v[1:n];

        d:= 1;
        for k:= 1 step 1 until n do

        begin   j:= k; for i:= 1 step 1 until k do v[i]:= A;
                i:= k; A:= r:= sqrt (v[k] - SUM (i,1,k-1,v[i] ↑ 2));
                d:= r × d;
                for j:= k+1 step 1 until n do
                begin i:= k; A:= (A - SUM (i,1,k-1,A × v[i])) / r end
        end LU;

        SYMDET1:= d ↑ 2

end SYMDET1;
```

```
comment         AP 225
        SYMSOL1 replaces the vector given in  array b[1 : n], by the solution vector x of the linear system:
        U transpose × U × x = b,  where U is an upper triangle which is defined by the actual parameters for
A,  i,  j and n  in the same way as the upper triangle of M in SYMDET1 (= AP 224). Consequently,  a call of
SYMSOL1, following a call of SYMDET1  with the same actual parameters for A,  i,  j  and  n,  has the effect
that b is replaced by the solution vector x of the linear system M × x = b.
SYMSOL1 leaves the elements A unaltered. It uses the non-local real procedure SUM (= AP 119);

procedure SYMSOL1(A,i,j,n,b); value n; integer i,j,n; real A; array b;

begin   real r;

        for j:= 1 step 1 until n do
        begin   i:= j; r:= A;
                b[j]:= (b[j] - SUM (i,1,j-1,A × b[i])) / r
        end;
        for i:= n step -1 until 1 do
        begin   j:= i; r:= A;
                b[i]:= (b[i] - SUM (j,i+1,n,A × b[j])) / r
        end
end SYMSOL1;
```

```
comment        AP 226
        SYMINV1 replaces the matrix elements A by the corresponding upper triangular elements of the inverse
        of  U transpose  × U,  where U is an upper triangle which is defined by the actual parameters in the
same way as the upper triangle of  M  in SYMDET1 (= AP 224).  Consequently,  a call of SYMINV1,  following a
call of SYMDET1  with the same actual parameters,  has the effect  that  the upper triangle of the symmetric
positive definite matrix M is replaced by the upper triangle of the inverse of M.
SYMINV1 uses the non-local real procedure SUM (= AP 119);

procedure SYMINV1 (A,i,j,n); value n; integer i,j,n; real A;

begin   integer k; real r; array v[1:n];

        for k:= 1 step 1 until n do

        begin   i:= j:= k; A:= v[k]:= 1 / A;

                for j:= k+1 step 1 until n do

                begin   i:= j; r:= A; i:= k;

                        A:= v[j]:= - SUM (i,k,j-1,A × v[i]) / r

                end;

                for i:= 1 step 1 until k do

                begin   j:= k; A:= SUM (j,k,n,A × v[j]) end

        end

end SYMINV1;
```

```
comment         AP 227
        syminv1 calculates the main diagonal of the inverse of U transpose × U, where U is an upper triangle
        which is defined by the actual parameters for A, i, j and n in the same way as the upper triangle of
M in SYMDET1 (= AP 224). The calculated diagonal elements are delivered in array d[1 : n]. Consequently, a
call of syminv1, following a call of SYMDET1 with the same actual parameters for A, i, j and n, has the
effect that the diagonal elements of the inverse of the symmetric positive definite matrix M are delivered
in array d.
syminv1 leaves the elements A unaltered. It uses the non-local real procedure SUM (= AP 119);

procedure syminv1(A,i,j,n,d); value n; integer i,j,n; real A; array d;

begin   integer k; real r;

        for k:= 1 step 1 until n do
        begin   i:= j:= k; d[k]:= 1 / A;
                for j:= k+1 step 1 until n do
                begin   i:= j; r:= A;
                        d[j]:= - SUM (i,k,j-1,A × d[i]) / r
                end;
                d[k]:= SUM (j,k,n,d[j] ↑ 2)
        end
end syminv1;
```

```
comment        AP 228
        SYMDET2:= determinant of the n-th order symmetric positive definite matrix, given in integer array
        A[1 : n × (n + 1) : 2] in such a way that, for all i and j satisfying 1 ≤ i ≤ j ≤ n, the (i, j)-th
element is A[i + (j - 1) × j : 2]. The method used is the square root method of Cholesky, yielding an upper
triangle U which, premultiplied by its transpose, gives alfa × matrix A. The elements of U are written over
the corresponding elements of A. The scaling factor alfa must be chosen so that the maximal element of U is
just within the integer capacity, in order to obtain a reasonably accurate representation of U. In view of
the definiteness of A this means that alfa must be slightly less (but not too critically, on account of the
inexactness of the arithmetic) than the square of the integer capacity divided by the maximal element of A.
Also, one may use SYMDET2 with real array A, in which case 1.0 is the most obvious value of alfa. If A is
negative definite, one may use SYMDET2 with alfa negative.
SYMDET2 uses the non-local real procedure INPROD (= AP 120);

real procedure SYMDET2 (A,n,alfa);

value n,alfa; integer n; real alfa; integer array A;

begin  integer i,j,k,kk,kj; real d;

       d:= 1; kk:= 0;
       for k:= 1 step 1 until n do

       begin  kk:= kk+k; A[kk]:=
              sqrt(A[kk] × alfa - INPROD(i;1-k,-1,A[kk+i],A[kk+i]));
              d:= A[kk] × d; kj:= kk;
              for j:= k+1 step 1 until n do

              begin  kj:= kj+j-1;
                     A[kj]:= (A[kj] × alfa
                     - INPROD (i,1-k,-1,A[kj+i],A[kk+i])) / A[kk]
              end
       end LU;

       SYMDET2:= d ↑ 2 / alfa ↑ n

end SYMDET2;
```

```
comment          AP 229
        SYMSOL2  replaces the vector given in  real array b[1 : n],  by the solution vector x  of the linear
        system:  U transpose × U × x = alfa × b,  where U is an upper triangle,  given in integer  (or real)
array A[1 : n × (n + 1) : 2]  in such a way that,  for all i and j satisfying  1 ≤ i ≤ j ≤ n,  the (i, j)-th
element is A[i + (j − 1) × j : 2]. The scaling factor alfa is chosen in relation to the scaling of U. Con-
sequently,  the call SYMSOL2 (A, n, alfa, b) following the call SYMDET2 (A, n, alfa)  (viz.: AP 228) has the
effect that b is replaced by the solution vector x of the linear system A × x = b.
SYMSOL2 leaves the elements of A unaltered. It uses the non-local real procedure SUM (= AP 119);

procedure SYMSOL2 (A,n,alfa,b);

value n,alfa; integer n; real alfa; integer array A; real array b;

begin   integer i,j,j0; integer array J[1:n];

        j0:= 0;

        for j:= 1 step 1 until n do

        begin   b[j]:=
                (b[j] × alfa − SUM (i,1,j−1,A[i+j0] × b[i]))/A[j+j0];

                J[j]:= j0; j0:= j0+j
        end;

        for i:= n step −1 until 1 do
        b[i]:= (b[i] − SUM (j,i+1,n,A[i + J[j]] × b[j]))/A[i + J[i]]
end SYMSOL2;
```

comment AP 230

ZERO:= x:= a zero of fx between a and b. The expression fx must depend on x and have different signs for x = a and x = b. In array e[1 : 2] one must give the relative tolerance e[1] and the absolute tolerance e[2], both of which must be positive.

The method is a combination of linear inter- and extrapolation and bisection, proceeding as follows:
Starting from the interval (a, b), ZERO constructs a sequence of shrinking intervals (c, x), each interval having the property that fx has different signs in its end points. If necessary, c and x are interchanged, in order to ensure that fx has the smaller absolute value in x. Subsequently, either interpolation using c and x or extrapolation using x and a point outside (c, x) takes place, yielding a new iterate i.
If abs (i − x) is too small, i is moved slightly towards c. Furthermore, the new iterate is accepted only if it is situated in the x-half of (c,x); otherwise it is replaced by the middle m of the interval. The process ends as soon as the interval (c, x) has a length $\leq 2 \times (\text{abs}(x \times e[1]) + e[2])$. For a simple zero this process is of order 1.6;

```
real procedure ZERO (x,a,b,fx,e); value a,b; real x,a,b,fx; array e;
begin   real c,fa,fb,fc,m,i,tol,re,ae;
        re:= e[1]; ae:= e[2];
        x:= a; fa:= fx; x:= b; fb:= fx; goto entry;
goon:   if abs (i - b) < tol then i:= b + sign (c - b) × tol;
        x:= if sign (i - m) = sign (b - i) then i else m;
        a:= b; fa:= fb; b:= x; fb:= fx;
        if sign (fc) = sign (fb) then
entry:  begin c:= a; fc:= fa end;
        if abs (fb) > abs (fc) then
        begin a:= b; fa:= fb; b:= c; fb:= fc; c:= a; fc:= fa end;
        m:= (b + c) / 2;
        i:= if fb - fa ≠ 0 then (a × fb - b × fa) / (fb - fa) else m;
        tol:= abs (b × re) + ae;
        if abs (m - b) > tol then goto goon;
        ZERO:= x:= b
end     ZERO;
```

comment AP 231

SPAP carries out HOUSEHOLDER's tridiagonalisation (Litt.: J.H. Wilkinson, Comp. J. 3 (1960), 23 – 27, Num. Math. 4 (1962), 354 – 361) on the symmetric matrix M, which in the following way is defined by means of the actual parameters for A, i, j and n:

The actual parameter for A – being a subscripted real variable whose indices (or index) depend(s) on the actual parameters for i and j – is the (i, j)th element of M for each i and j satisfying $1 \leq i \leq j \leq n$. Thus one needs to give the upper triangle of M only. If one wants to avoid waste of space, one may give this triangle in a one–dimensional array. E.g., if the upper triangle of M is given in array C[1 : n × (n + 1) : 2] columnwise, i.e. the columns one after the other, and if the successive values (j – 1) × j : 2 have been recorded in an auxiliary integer array J[1 : n], then the appropriate call of SPAP reads:

SPAP (C[i + J[j]], i, j, n, B, BB, D, E).

The last four parameters are output arrays, to be declared as array B, BB, D[1 : n], E[0 : 3]. However, if SEIGENVA is used after SPAP then the array E must be declared as array E[0 : 7].

SPAP delivers its results as follows:

The main diagonal of the triple diagonal matrix is written over the main diagonal of M and stored in D, the codiagonal elements are delivered in B and the squares of these elements in BB. Moreover, B[n]:= BB[n]:= 0. The vectors defining the subsequent transformations are written over the corresponding rows of the upper triangle of M. Thus enough information is retained for the calculation of eigenvalues and eigenvectors. E[3]:= the maximum of the absolute row sums of M, which matrix norm is an upper bound of the moduli of its eigenvalues. The elements E[0], E[1] and E[2] become zero. (These assignments are carried out for the benefit of SEIGENVA.) At each stage the transformation is skipped if the corresponding codiagonal element B[r] satisfies E[3] – B[r] = E[3]. The arithmetic must be such that this condition is equivalent with abs (B[r]) < E[3] × eps, where eps (nearly) equals the relative machine precision. The matrix norm E[3] must be reasonably large so that at any rate the relation E[3] – B[n] = E[3] holds for the vanishing element B[n].

In order to simplify the computation, at each stage the vector defining the r–th transformation is normalized so that the square of its Euclidean norm equals –2 × B[r] × the (r + 1)th element of the vector. SPAP uses the non–local real procedure SUM, which must have the property that after a call of SUM the summation variable has obtained the rejected value;

```
procedure SPAP (A,i,j,n,B,BB,D,E); value n; integer i,j,n; real A; array B,BB,D,E;
begin   integer p,r; real w,x,s;
        s:= 0; for p:= 1 step 1 until n do
        begin   j:= p; w:= SUM (i,1,p-1,abs (A)) + SUM (j,p,n,abs (A)); if w > s then s:= w end;
HA:     for r:= 1 step 1 until n do
        begin   j:= i:= r; D[r]:= A; BB[r]:= SUM (j,r+1,n,A ↑ 2);
                B[r]:= sqrt (BB[r]); if s - B[r] = s then begin B[r]:= BB[r]:= 0; goto HB end;
                j:= r+1; if A > 0 then B[r]:= -B[r]; A:= A-B[r]; w:= A × B[r];
```

```
comment     AP 231, continued;

            for j:= r+1 step 1 until n do D[j]:= A;
            for p:= r+1 step 1 until n do
            begin   j:= p; B[p]:= (SUM (i,r+1,p-1,A × D[i]) + SUM (j,p,n,A × D[j]))/w end;
            x:= SUM (p,r+1,n,D[p] × B[p])/(2 × w);
            for j:= r+1 step 1 until n do B[j]:= D[j] × x + B[j];
            for i:= r+1 step 1 until n do for j:= i step 1 until n do
            A:= D[i] × B[j] + B[i] × D[j] + A;
        HB:
        end; E[0]:= E[1]:= E[2]:= 0; E[3]:= s
end     SPAP;
```

comment AP 232

SEIGENVA:= E[6]:= next eigenvalue of the n-th order symmetric triple diagonal matrix with main diagonal given in array D[1 : n] and the squares of the codiagonal elements, concluded by 0, in array BB[1 : n]. In array e[1 : 2] one must give the relative tolerance e[1] and the absolute tolerance e[2] for the eigenvalue. In array E[0 : 7] SEIGENVA records some administrative quantities. Before the first call of SEIGENVA only the following elements of E must be given: E[0]:= E[2]:= 0 and E[3]:= a suitable matrix norm, being an upper bound of the moduli of the eigenvalues (with negative sign if so desired, see below).
The method is based on the STURM property of the sequence of principal minors (Litt.: W. Givens, NBS-AMS 29 (1953), 117 - 122). If the codiagonal contains small elements BB[r] satisfying ss - BB[r] = ss, where ss = E[3] ↑ 2, then these elements are neglected and the matrix is subdivided into submatrices which are dealt with separately. The arithmetic must be such that this smallness condition is equivalent with BB[r] < E[3] × eps, where eps (nearly) equals the square root of the relative machine precision. The matrix norm E[3] must be reasonably large so that at any rate the smallness condition holds for the vanishing element BB[n].
The eigenvalue is calculated by means of the non-local real procedure ZERO, which finds a zero of a function having different signs in the end points of a given interval. The r-th eigenvalue of a certain submatrix is located by means of the function: if p = r or r - 1 then (-1) ↑ r × det (lambda × I - matrix) else sign (p-r) × the maximal modulus of the function values already computed. Here p = the number of sign variations in the STURM sequence. The factor (-1) ↑ r is calculated by means of the non-local integer procedure EVEN.
Calling SEIGENVA n times one obtains all eigenvalues of the matrix. The eigenvalues of each submatrix are delivered in order of decreasing magnitude.
In order to obtain the eigenvalues of a symmetric matrix, one may well use SPAP, followed by the calls of SEIGENVA with the same actual parameters for n, D, BB and E. In that case no preparatory assignments in array E are needed, as SPAP carries them out.
The main purpose of the subdivision into submatrices is to facilitate the calculation of mutually orthogonal eigenvectors in the case that some eigenvalues are (nearly) coincident. It should be noted, however, that this is just the case where the error in the eigenvalues may be as large as the largest codiagonal element neglected, which is (at most) E[3] × the square root of the machine precision. If one wants to avoid this inconvenience one may call SEIGENVA with negative E[3] and abs (E[3]) defined as above. In that case only those codiagonal elements are neglected the squares whereof are equal to the vanishing element BB[n]. After a call of SPAP and the assignment E[3]:= -abs (E[3]) this means that just those elements are neglected for which the transformation was skipped by SPAP.
If one is not interested in the remaining eigenvalues of the submatrix considered one performs the assignment E[0]:= E[2] before the next call of SEIGENVA, whereupon SEIGENVA will operate on the next submatrix.
SEIGENVA can also be used for the calculation of eigenvalues of so called "quasi symmetric" triple diagonal matrices, i.e. triple diagonal matrices with the property that the products of the corresponding codiagonal elements are non-negative. In this case these products, concluded by 0, must be given in array BB.
In array E[0 : 7] the following quantities are recorded:

AP 232, continued.

E[0] = number of calculated eigenvalues. SEIGENVA increases this number by 1. The starting value must be 0.
E[1] = lower index and E[2] = upper index of the submatrix considered. They satisfy the relations E[1] $\leq$ E[0] $\leq$ E[2]. If E[0] = E[2] the next submatrix is taken. The starting value of E[2] must be 0.
E[3] = a suitable matrix norm, being an upper bound of the moduli of the eigenvalues or the reversed value. The sign of E[3] rules the subdivision. The value of E[3] must be given. SEIGENVA does not alter it.
E[4] = an upper bound of the next eigenvalue of the submatrix considered.
E[5] = maximum of the calculated absolute values of the characteristic function of the submatrix considered.
E[6] = eigenvalue computed lastly.
E[7] = squared codiagonal element neglected lastly.
These quantities contain sufficient information for subsequent calls of SEIGENVA and subsequent calculations of the eigenvectors of the given symmetric triple diagonal matrix. SEIGENVA leaves the elements of D, BB and e unaltered. It uses the non-local type procedures ZERO (= AP 230) and EVEN (= AP 118);

```
real procedure SEIGENVA (D,BB,n,e,E); value n; integer n; array D,BB,e,E;
begin   integer r,t,k,n1,n2; real x,low,ss;
        real procedure SDET (q,q2); value q,q2; integer q,q2;
        begin   integer p; real d0,d1,d2;
                p:= 0; SDET:= E[5]; d1:= t; d2:= (x-D[q]) × d1; goto DB;
        DA:     q:= q+1; d0:= d1; d1:= d2; d2:= (x-D[q]) × d1 - BB[q-1] × d0;
        DB:     if d2 > 0 = d1 ≤ 0 then p:= p+1; if p ≤ r then
                begin   if q < q2 then goto DA; if x < E[4] then E[4]:= x;
                        if abs (d2) > E[5] then E[5]:= abs (d2);
                        SDET:= if p ≥ r-1 then d2 else - E[5]
                end
        end     SDET;
GA:     k:= E[0]; n2:= E[2]; low:= -2 × abs (E[3]); ss:= E[3] × abs (E[3]);
        if k = n2 then
        begin   E[4]:= -low; E[5]:= 0; n1:= n2+1;
        GC:     n2:= n2+1;
                if if ss > 0 then ss - BB[n2] ≠ ss else BB[n2] ≠ BB[n] then goto GC;
                E[7]:= BB[n2]
        end else n1:= E[1];
        k:= k+1; r:= k-n1+1; t:= EVEN (r); E[0]:= k; E[1]:= n1; E[2]:= n2;
        SEIGENVA:= E[6]:= ZERO (x,E[4],low,SDET (n1,n2),e)
end     SEIGENVA;
```

comment AP 233

SEIGENVEC calculates an eigenvector of the n-th order symmetric triple diagonal matrix with main diagonal given in array D[1 : n] and the codiagonal given in array B[1 : n - 1]. The eigenvector calculated corresponds with the eigenvalue E[6] of the submatrix with lower index E[1] and upper index E[2] and has the Euclidean norm 1.

The eigenvector of the submatrix is computed by means of forward and backward recursion meeting each other at a component, the modulus of which is a relative maximum. This eigenvector of the submatrix is supplied with components 0 in order to obtain an eigenvector of the entire matrix. The eigenvector is delivered in array V [1 : n] which must be declared, however, as containing two extra elements, viz. array V[0 : n + 1].

SEIGENVEC may well be used after SEIGENVA with the same actual parameters for n, D and E. If SEIGENVA is called with E[3] > 0, then (nearly) coincident eigenvalues will usually come out as eigenvalues of different submatrices. In that case SEIGENVEC will find mutually orthogonal corresponding eigenvectors. It may occur, however, that the matrix has very close eigenvalues even if the codiagonal elements are not at all small. In that case SEIGENVEC will not find mutually orthogonal (and possibly not even independent) corresponding eigenvectors.

SEIGENVEC leaves the elements of D, B and E unaltered. It uses the non-local real procedure SUM (= AP 119);

```
procedure SEIGENVEC (D,B,n,E,V); value n; integer n; array D,B,E,V;
begin   integer n1,n2,i,p,q; real x,x1; n1:= E[1]; n2:= E[2]; x:= E[6];
WA:     p:= n1-1; q:= n2+1; V[p]:= V[q]:= 1;
WB:     i:= p:= p + 1; if p = n2 then goto WD;
        V[p]:= (if p = n1 then (x - D[p]) else ((x - D[p]) × V[p - 1] - B[p - 1] × V[p - 2])) / B[p];
        if abs (V[p]) ≥ abs (V[p - 1]) then goto WB; if p ≥ q then goto WD;
WC:     i:= q:= q - 1; if q = n1 then goto WD;
        V[q]:= (if q = n2 then (x - D[q]) else ((x - D[q]) × V[q + 1] - B[q] × V[q + 2])) / B[q - 1];
        if abs (V[q]) > abs (V[q + 1]) then goto WC; if p < q then goto WB;
WD:     V[i]:= 1/sqrt (SUM (p, n1 - 1, i - 2, V[p] ↑ 2)/V[i - 1] ↑ 2 + 1
                + SUM (p, i + 2, n2 + 1, V[p] ↑ 2)/V[i + 1] ↑ 2);
        x1:= V[i]/V[i - 1]; for p:= i - 1 step -1 until n1 do V[p]:= V[p - 1] × x1;
        x1:= V[i]/V[i + 1]; for p:= i + 1 step  1 until n2 do V[p]:= V[p + 1] × x1;
        for p:= 1 step 1 until n1 - 1, n2 + 1 step 1 until n do V[p]:= 0
end     SEIGENVEC;
```

comment AP 234

STRASF carries out the back-transformation of the n-vector given in array V[1 : n], in correspondence with HOUSEHOLDER's tridiagonalisation carried out by SPAP (= AP 231). The codiagonal, concluded by 0, of the symmetric triple diagonal matrix must be given in array B[1 : n] and the vectors of the subsequent transformations must be given in the upper triangle, defined by the actual parameters for A, i, j and n as described for the upper triangle of M in SPAP. Consequently, following a call of SPAP, a call of STRASF with the same actual parameters for A, i, j, n and B and with an eigenvector of the symmetric triple diagonal matrix given in V has the effect that V is replaced by the corresponding eigenvector of the original symmetric matrix M.

STRASF leaves the elements of A and B unaltered. It uses the non-local real procedure SUM (= AP 119);

```
procedure STRASF (A,i,j,n,B,V); value n; integer i,j,n; real A; array B,V;
begin   real x1,f1; for i:= n-1 step -1 until 1 do
        begin   if B[i] ≠ B[n] then
                begin   j:= i+1; x1:= A; f1:= SUM (j,i+1,n,A×V[j])/(x1×B[i]);
                        for j:= i+1 step 1 until n do V[j]:= A×f1+V[j]
                end
        end
end STRASF;
```

comment AP 235

SEVAVEC calculates the eigenvalues and eigenvectors of the n-th order symmetric matrix M defined by the actual parameters for A, i, j and n in the same way as described in SPAP. In array e[1 : 2] one must give the relative tolerance e[1] and the absolute tolerance e[2] for the eigenvalues. In the auxiliary array E[0 : 7] which must be declared only, some administrative quantities are recorded (see SEIGENVA). The procedures OVA (x) with parameter real x and OVEC (V) with parameter array V serve to deliver each time the eigenvalue x, resp. the eigenvector V given in array V[1 : n]. In these procedures one can obtain additional information from array E. Moreover, one may influence the computation by modifying some elements of E. In this connection it is essential that in the body of OVA, if x is non-value, the calculation of the eigenvalue is carried out in just one assignment statement involving x. SEVAVEC uses SPAP (= AP 231), SEIGENVA (= AP 232), SEIGENVEC (= AP 233) and STRASF (= AP 234), which see for further details;

```
procedure SEVAVEC (A,i,j,n,e,E,OVA,OVEC);
value n; integer i,j,n; real A; array e,E; procedure OVA,OVEC;
begin   integer k; array B,BB,D[1:n],V[0:n+1];
        SPAP (A,i,j,n,B,BB,D,E);
next:   OVA (SEIGENVA (D,BB,n,e,E));
        SEIGENVEC (D,B,n,E,V); STRASF (A,i,j,n,B,V); OVEC (V);
        k:= E[0]; if k < n then goto next
end SEVAVEC;
```

comment AP 236

ZEREX:= x:= the largest zero of fx smaller than the given value of x. Moreover, xa:= the previous value of x. One must give starting values to x and xa such that desired zero $\leq$ xa < x. The function, defined by the expression fx depending on x must be convex between the desired zero and the given value of x. Moreover, the desired zero must be well separated from the other zeroes of fx. In array e[1 : 2] one must give the relative tolerance e[1] and the absolute tolerance e[2].
One may also call ZEREX with starting values x and xa such that desired zero < x < xa. In this case, fx must be convex and non-vanishing between desired zero and xa, with a possible exception for a neighbourhood of x, where fx might be badly defined. In this case also, the desired zero must be well separated.
ZEREX has been written mainly for finding the zeroes of a polynomial P (x), having real and well separated zeroes only. The successive calls of ZEREX, with fx = P (x) / PROD (i, 1, k − 1, x − Z[i]), will yield the zeroes Z[k] in order of decreasing magnitude, provided that values of x and xa (with Z[1] $\leq$ xa < x) are defined before ZEREX is called for the first time.
Method: The desired zero is calculated by means of linear extrapolation. The starting values are two points between x and xa. If x < xa then a (possibly dangerous) neighbourhood of x is avoided by successive halving of the interval (x, xa) until an extrapolate safely smaller than x is found. Just then this extrapolate is accepted and the ordinary extrapolation starts. If the difference of two successive iterates is too small, then the later iterate is slightly diminished. As soon as fx changes sign the extrapolation ends and the zero is located by means of the real procedure ZERO, which yields a zero x within a tolerance 2 × (abs (x × e[1]) + e[2]). The function must be convex and the desired zero must be well separated in order to ensure that the extrapolates remain larger than the desired zero and that the sign changing will indeed be stated.
ZEREX uses the non-local real procedure ZERO (= AP 230);

```
real procedure ZEREX (x,fx,xa,e); real x,fx,xa; array e;
begin   real a,b,fa,fb,i,be,re,ae;
        re:= e[1]; ae:= e[2];
        b:= (2 × xa + x) / 3; xa:= x; x:= b; fb:= fx;
reject: x:= (b + xa) / 2; a:= b; fa:= fb; b:= x; fb:= fx;
        i:= (a × fb − b × fa) / (fb − fa);
        goto if (xa − i) × 2 < b − xa then reject else accept;
goon:   i:= (a × fb − b × fa) / (fb − fa);
accept: be:= b − (abs (b × re) + ae); a:= b; fa:= fb;
        x:= b:= if i < be then i else be; fb:= fx;
        if sign (fb) = sign (fa) then goto goon;
        ZEREX:= ZERO (x,a,b, if x = a then fa else if x = b then fb else fx ,e)
end     ZEREX;
```

comment AP 237

POL:= the value in x of the n-th degree polynomial defined by: sigma over k from 0 until n of A × x ↑ (n − k). In other words: the coefficients of the polynomial are the successive values of the expression A depending on k;

```
real procedure POL (A,k,n,x); value n,x; integer k,n; real A,x;
begin   real r; r:= 0;
        for k:= 0 step 1 until n do r:= r × x + A; POL:= r
end POL;
```

comment AP 238

APAP transforms the n–th order matrix given in array A[1 : n, 1 : n] into an upper HESSENBERG matrix H, say, (i.e. H[i, j] = 0 for i > j + 1) according to HOUSEHOLDER's method (Litt.: J. H. Wilkinson, Comp. J. 3 (1960), 23 – 27). A suitable value, e.g. the relative machine precision, must be given to the parameter eps, being the relative tolerance for the transformation. APAP delivers its results as follows:
norm:= the maximum of the absolute row sums of A, which matrix norm is an upper bound of the moduli of the eigenvalues. The upper triangular elements of the resulting HESSENBERG matrix H (i.e. the elements H[i, j] with i ≤ j) are written over the corresponding elements of A.
In array B[1 : n] the codiagonal elements B[k]:= H[k + 1, k] are deliverd, moreover B[n]:= eps × norm. The vectors defining the subsequent transformations are written over the corresponding columns of A, using only the elements below the main diagonal. Thus enough information is retained for the calculation of eigenvalues and eigenvectors.
At each stage the transformation is skipped if the corresponding codiagonal element B[k] satisfies abs (B[k]) ≤ eps × norm, in which case the value eps × norm is assigned to B[k].
In order to simplify the computation, at each stage the vector defining the k–th transformation is normalised so that the square of its Euclidean norm equals –2 × B[k] × the (k + 1)–th element of the vector.
APAP uses the non–local real procedure SUM (= AP 119) and the real procedure INPROD (= AP 120);

```
procedure APAP (A,n,eps,norm,B); value n,eps; integer n; real eps,norm; array A,B;
begin   integer i,j,k; real w,alfa,tol; array P[1:n];
        norm:= 0; for i:= 1 step 1 until n do
        begin   w:= SUM (j,1,n,abs (A[i,j])); if w > norm then norm:= w end;
        tol:= eps × norm;
HA :    for k:= 1 step 1 until n do
        begin   B[k]:= sqrt (INPROD (i,k+1,n,A[i,k],A[i,k]));
                if abs (B[k]) ≤ tol then begin B[k]:= tol; goto HB end;
                if A[k+1,k] > 0 then B[k]:= – B[k]; A[k+1,k]:= A[k+1,k] – B[k]; w:= A[k+1,k] × B[k];
                for i:= 1 step 1 until n do P[i]:= INPROD (j,k+1,n,A[i,j],A[j,k])/w;
                alfa:= INPROD (i,k+1,n,A[i,k],P[i]);
                for j:= k+1 step 1 until n do B[j]:= (INPROD (i,k+1,n,A[i,k],A[i,j]) + alfa × A[j,k])/w;
                for j:= k+1 step 1 until n do
                begin   for i:= 1 step 1 until k do A[i,j]:= P[i] × A[j,k] + A[i,j];
                        for i:= k+1 step 1 until n do A[i,j]:= A[i,k] × B[j] + P[i] × A[j,k] + A[i,j]
                end;
        HB :
end     end APAP;
```

comment AP 239

REIGENVA:= E[2]:= Z[E[0]]:= next eigenvalue of the n–th order upper HESSENBERG matrix whose upper triangle is given in array A[1 : n, 1 : n] (thus, REIGENVA uses only the elements A[i, j] with i ≤ j) and whose codiagonal is given in array B[1 : n − 1]. The eigenvalues of this matrix must be real and well separated.

In array e[1 : 2] one must give the relative tolerance e[1] and the absolute tolerance e[2] for the eigenvalue. In array E[0 : 3] one must give the serial number E[0] of the desired eigenvalue (i.e. 1 + the number of eigenvalues already computed) and a matrix norm E[1] which must be an upper bound of the moduli of the eigenvalues. Moreover, in array Z[1 : E[0]] one must give the eigenvalues already computed.

REIGENVA delivers the next eigenvalue in E[2] and in Z[E[0]], the number of iterations in E[3] and an estimate of the corresponding eigenvector in array V[1 : n] (for the benefit of REIGENVEC (= AP 240). Note that, if REIGENVEC is used, E must be declared array E[0 : 5]). Subsequent calls of REIGENVA yield the eigenvalues in order of decreasing magnitude. Consequently if one wants all eigenvalues of the matrix one declares array Z, V[1 : n] and carries out the assignment E[1]:= matrix norm and the statement for k:= 1 step 1 until n do S, where S stands for a statement involving the assignment E[0]:= k and a call of REIGENVA. Then all eigenvalues are delivered in array Z[1 : n].

The eigenvalues are calculated by means of the non-local real procedure ZEREX, which requires that the eigenvalues are real and well separated. The characteristic function is evaluated according to HYMANS' method (Litt.: J. H. Wilkinson, Num. Math. 2 (1960), p. 327 sqq). This method requires that the codiagonal elements given in array B do not vanish. It is advisable to replace all codiagonal elements whose moduli are smaller than some threshold (e.g. matrix norm × relative machine precision), by this threshold.

REIGENVA may well be used after APAP, which delivers codiagonal elements whose moduli are larger than or equal to eps × matrix norm. REIGENVA leaves the elements of A, B and e unaltered. It uses INPROD (= AP 120), PROD (= AP 202) and ZEREX (= AP 236);

```
real procedure REIGENVA (A,n,B,e,E,Z,V); value n; integer n; array A,B,e,E,Z,V;
begin   real x,xa; integer k;
        real procedure RDET (x); value x; real x;
        begin   integer i,j; E[3]:= E[3] + 1; V[n]:= 1;
                for i:= n step -1 until 2 do V[i-1]:= (x × V[i] − INPROD (j,i,n,A[i,j],V[j]))/B[i-1];
                RDET:= (x × V[1] − INPROD (j,1,n,A[1,j],V[j])) /PROD (i,1,k-1,x − Z[i])
        end RDET;
        k:= E[0]; E[3]:= 0; x:= if k > 1 then Z[k-1] else 2 × E[1];
        xa:= if k > 2 then Z[k-2] else if k = 2 then x + E[1] else E[1];
        REIGENVA:= E[2]:= Z[k]:= ZEREX (x,RDET (x),xa,e)
end REIGENVA;
```

comment AP 240

REIGENVEC calculates the eigenvector corresponding with the real eigenvalue E[2] of the n-th order upper HESSENBERG matrix whose upper triangle is given in array A[1 : n, 1 : n] (thus, REIGENVEC uses only the elements A[i, j] with i ≤ j) and whose codiagonal is given in array B[1 : n - 1].
One must give: in array V[1 : n] an estimate of the eigenvector (which needs not be normalised), in array e[1 : 2] the relative tolerance e[1] and the absolute tolerance e[2] for the eigenvalue and in array E[2 : 5] the eigenvalue E[2].
The eigenvector is calculated by means of inverse iteration, each step involving Gaussian elimination with partial pivoting. This process is of order n ↑ 2 per step and requires - beside the given matrix - a temporary storage for n × (n + 3) : 2 real numbers. Each step starts with a normalised estimate of the eigenvector. The iteration ends if the inverse iteration yields a vector whose Euclidean norm is larger than or equal to 1/(4 × (abs (E[2] × e[1]) + e[2])) or if 10 steps have been carried out.
REIGENVEC delivers the eigenvector (normalised so that its Euclidean norm = 1) in array V[1 : n] and, moreover, the number of iterations in E[4] and the normalisation factor, i.e. 1/Euclidean norm of the vector iterated inversely, in E[5]. Thus, the value E[5] is approximately equal to the Euclidean norm of (matrix - E[2] × I) × V.
If the matrix has (nearly) coinciding eigenvalues then REIGENVEC may yield corresponding eigenvectors which are not independent. In this case it may be helpful to call REIGENVEC with E[2] slightly modified, so that the successive values of E[2] do not agree within working accuracy.
REIGENVEC may well be used after REIGENVA, in which case the matrix must have well separated eigenvalues. It leaves the elements of A, B and e unaltered. It uses the non-local real procedure INPROD (= AP 120);

```
procedure REIGENVEC (A,n,B,e,E,V); value n; integer n; array A,B,e,E,V;
begin   integer i,j,i0,i1; real m,r,labda; Boolean array p[1:n]; array C[1:n × (n+3) : 2 - 1];
        labda:= E[2]; i1:= 0; C[1]:= A[1,1] - labda; for j:= 2 step 1 until n do C[j]:= A[1,j];
gauss:  for i:= 1 step 1 until n-1 do
        begin   i0:= i1; i1:= i1+n-i+1; r:= C[i0+i]; m:= B[i]; p[i]:= abs (m) ≤ abs (r);
                if p[i] then
                begin   C[i1+i]:= m:= m/r; for j:= i+1 step 1 until n do
                        C[i1+j]:= (if j > i+1 then A[i+1,j] else A[i+1,j] - labda) - m × C[i0+j]
                end     else
                begin   C[i0+i]:= m; C[i1+i]:= m:= r/m; for j:= i+1 step 1 until n do
                        begin   r:= if j > i+1 then A[i+1,j] else A[i+1,j] - labda;
                                C[i1+j]:= C[i0+j] - m × r; C[i0+j]:= r
        end     end     end gauss;
        r:= 1/sqrt (INPROD (j,1,n,V[j],V[j])); for j:= 1 step 1 until n do V[j]:= V[j] × r; E[4]:= 0;
```

```
        comment          AP 240, continued;

iterat: i0:= 0; E[4]:= E[4] + 1; for i:= 1 step 1 until n-1 do
        begin   i0:= i0+n-i+1; if p[i] then V[i+1]:= V[i+1] - C[i0+i] × V[i] else
                begin r:= V[i+1]; V[i+1]:= V[i] - C[i0+i] × r; V[i]:= r end
                end forward;
                for i:= n step -1 until 1 do
                begin V[i]:= (V[i] - INPROD (j,i+1,n,C[i0+j],V[j]))/C[i0+i]; i0:= i0-n+i-2 end backward;
                r:= 1/sqrt (INPROD (j,1,n,V[j],V[j])); for j:= 1 step 1 until n do V[j]:= V[j] × r;
                if r > 4 × (abs(labda × e[1]) + e[2]) ∧ E[4] < 9.5 then goto iterat; E[5]:= r
end REIGENVEC;
```

comment AP 241

ATRASF carries out the backtransformation of the n–vector, given in array V[1 : n], in correspondence with HOUSEHOLDER's transformation carried out by APAP (= AP 238). The codiagonal, concluded by the threshold eps × norm, of the HESSENBERG matrix must be given in array B[1 : n] and the vectors of the subsequent transformations must be given in the part below the main diagonal of array A[1 : n, 1 : n]. Consequently, a call of ATRASF following a call of APAP, with an eigenvector of the HESSENBERG matrix given in V, has the effect that V is replaced by the corresponding eigenvector of the original matrix. Since HOUSEHOLDER's transformation is orthogonal, the Euclidean norm of V remains invariant.

ATRASF may also be used for the backtransformation of a complex eigenvector. In this case one calls ATRASF twice, once for the real part and once for the imaginary part of the eigenvector.

ATRASF leaves the elements of A and B unaltered. It uses the non–local real procedure INPROD (= AP 120);

```
procedure ATRASF (A,n,B,V); value n; integer n; array A,B,V;
begin   integer i,j; real r; for j:= n–1 step –1 until 1 do
        begin   if B[j] ≠ B[n] then
                begin   r:= INPROD (i,j+1,n,A[i,j],V[i])/(A[j+1,j] × B[j]);
                        for i:= j+1 step 1 until n do V[i]:= A[i,j] × r + V[i]
end     end     end ATRASF;
```

comment AP 242

REVAVEC calculates the eigenvalues and eigenvectors of the n-th order matrix given in array A[1 : n, 1 : n]. The eigenvalues must be real and well separated. In array e[0 : 2] one must give the relative tolerance e[0] for the transformation (relative to matrix norm) and the relative tolerance e[1] and the absolute tolerance e[2] for the eigenvalues. The arrays E and Z need be declared only: array E[0 : 5], Z[1 : n]. In array Z the eigenvalues are delivered and in array E the following quantities:

E[0]:= serial number of the last computed or next eigenvalue
E[1]:= matrix norm: the maximum of the absolute row sums of A
E[2]:= last computed eigenvalue
E[3]:= number of iterations for the calculation of the eigenvalue
E[4]:= number of iterations for the calculation of the eigenvector
E[5]:= (transformed matrix - lambda × I) × eigenvector (approximately).

The procedures OVA (x) with parameter real x and OVEC (V) with parameter array V serve to deliver each time the eigenvalue x or the eigenvector given in array V[1 : n]. In these procedures one can obtain additional information from the actual arrays. In this connection it is essential that in the body of OVA, if x is non-value, the calculation of the eigenvalue is carried out in just one assignment statement involving x.

REVAVEC delivers the eigenvalues in order of decreasing magnitude. The eigenvectors, more precisely: the solutions of the linear systems:

$$\text{Sigma}\ (A[i, j] \times V[j]) = \text{lambda} \times V[i]$$

are normalised so that Euclidean norm = 1. REVAVEC uses the non-local procedures APAP (= AP 238), REIGENVA (= AP 239), REIGENVEC (= AP 240) and ATRASF (= AP 241), which see for further details;

```
procedure REVAVEC (A,n,e,E,Z,OVA,OVEC); value n; integer n; array A,e,E,Z; procedure OVA,OVEC;

begin   integer k; array B,V[1:n]; APAP (A,n,e[0],E[1],B); for k:= 1 step 1 until n do

        begin   E[0]:= k; OVA (REIGENVA (A,n,B,e,E,Z,V));

                REIGENVEC (A,n,B,e,E,V); ATRASF (A,n,B,V); OVEC (V)

end     end REVAVEC;
```

comment AP 243

K calculates the complete elliptic integral of the first kind:

$$K:=\int_0^{\pi/2}(1-(\sin^2 A)(\sin^2 x))^{-1/2}\,dx$$

the relative error is of the order $_{10}-12$.

(literature : D. J. Hofsommer and R. P. van de Riet,
On the numerical integration of Elliptic Integrals of the
first and second kind and the Elliptic Functions of Jacobi.
Numerische Mathematik 5 (1963) pp 291 - 302);

```
real procedure K (A); value A; real A;
begin   real a1, a2, b, k1; integer n; b:= abs (sin (A)); k1:= cos(A); if b ≥ .9539
        then    begin   a1:= (1+b)/2; b:= sqrt(b); a2:= (a1+b)/2;
                        b:=sqrt (a1×b); K:= ln (128×(a2+b)×a2×a1↑2/
                        k1↑4)/(a2+b)/2
                end
        else    begin   b:= abs (cos (A)); a1:=1; for n:=1, 2, 3 do
                        begin a2:= (a1+b)/2; b:= sqrt (a1×b); a1:= a2
                        end; K:= 3.14159265359/(a1+b)
                end
end K;
```

comment AP 244

EE calculates the complete elliptic integral of the second kind:

$$EE:=\int_0^{\pi/2}(1-(\sin^2 A)(\sin^2 x))^{+1/2}\,dx$$

the relative error is of the order $_{10}$-12, for literature reference see the comment of AP 243;

```
real procedure EE (A); value A; real A;
begin   real a1, a2, b, s, k1; integer n; b:= abs (sin (A));
        k1:= abs (cos (A)); if b ≥ .9539
        then    begin   a1:= (1+b)/2; s:= k1×k1/2+a1×a1-b; b:= sqrt (b);
                        a2:= (a1+b)/2; A:= a1×b; s:= s/2+a2×a2-A;
                        b:= sqrt (A); EE:= (a2+b)/2+(s/(a2+b))× ln (128×
                        (a2+b)×a2×a1×a1/(k1↑4))
                end
        else    begin   b:= k1; a1:= 1; s:= 1 + b × b;
                        for n:= 1, 2, 3 do
                        begin a2:= (a1+b)/2; A:= a1×b; a1:= a2;
                              s:= s/2-a1×a1+A; b:= sqrt (A)
                        end; EE:= 12.5663706144 × s /(a1+b)
                end
end EE;
```

```
comment          AP 245

        BB calculates the complete elliptic integral:

        BB:=∫(0 to π/2) cos^2 x(1 - (sin^2 A)(sin^2 x))^-1/2 dx

        the relative error is of the order 10-12, for literature reference
        see the comment of AP 243;

real procedure  BB (A); value A; real A;
begin    real a1, a2, b, b1, s, k1; integer n; b:= abs (sin (A));
         k1:= abs (cos (A)); b1:= b; if b ≥ .9539
         then    begin    a1:= (1+b)/2; s:= a1×a1 - k1×k1 /2 -b;
                          b:= sqrt (b); a2:= (a1+b)/2; A:= a1×b;
                          b:= sqrt (A); s:= s/2+a2×a2-A;
                          BB:= ((a2+b)/2+(s/(a2+b))× ln (128×(a2+b)×a2×a1×
                          a1/(k1↑4)))/(b1×b1)
                  end
              else    begin    b:= k1; a1:= 1; s:= 0; k1:= 1;
                          for n:= 1, 2, 3 do
                          begin k1:= .25 × k1 × (a1 - b)/(a1 + b);
                              a2:= (a1+b)/2; A:= a1×b; a1:= a2;
                              s:= s/2 - k1; b:= sqrt (A)
                          end; BB:= 3.14159265359 × (.5 + s × 4)/(a1 + b)
                  end
end BB;
```

comment AP 246

F calculates the incomplete elliptic integral of the first kind:

$$F := \int_0^P (1 - (\sin^2 A)(\sin^2 x))^{-1/2}\, dx$$

with $0 \leq P \leq \pi/2$, the relative error is of the order $_{10}-12$,
for literature reference see the comment of AP 243;

```
real procedure F (A, P); value A, P; real A, P;
begin   real a, a1, b, b1, si; integer n, m; b:= abs (cos (A));
        b1:= abs (sin (A)); if b1 < .9539
        then    begin  a:= 1; si:= cos (P)/sin (P); n:= 0;  A:= b;
                       for m:= 1, 2, 3 do
                       begin si:= (si-A/si)/2; n:= 2×n + (1 - sign (si))
                            /2; a:= (a+b)/2; b:= sqrt (A); A:= a×b
                       end; si:= (si - A/si)/2; n:= 2×n + 1 - sign (si);
                       a:= (a+b)/2; F:= ((2 × arctan (a/si) + n ×
                       3.14159265359)/a)/32
                end
        else    begin  a:= 1; b:= b1; si:= cos (P)/sin (P); a1:= b×b;
                       for m:= 1, 2 do
                       begin A:= a×b; si:= a×si + sqrt (a×a×(si×si + 1)
                            - a1); a:= (a+b)/2; si:= si/(a×2);
                            b:= sqrt (A); a1:= A
                       end; si:= a×si + sqrt (a×a×(si×si + 1) - A);
                       a:= (a+b)/2; si:= si/(a×2); F:= ln ((1 + sqrt
                       (1 + si×si))/si)/a
                end
end F;
```

```
comment        AP 247
        E calculates the incomplete elliptic integral of the second kind:
        E:= ∫0^P (1 - (sin^2 A)(sin^2 x))^+1/2 dx
        with 0 ≤ P ≤ π/2, the relative error is of the order 10-12,
        for literature reference see the comment of AP 243;
real procedure  E (A, P); value A, P; real  A, P;
begin   real U, V;
        procedure  EA;
        begin   real  a, a1, b, S1, S2, si, co; integer n, m;
                b:= U; a:= 1; co:= cos(P)/sin (P); S1:= S2:= 0; a1:= b×b;
                n:= 0; for m:= 1, 2, 3, 4 do
                begin   A:= a×b; co:= (co - A/co)/2; n:= 2×n + (1 - sign
                (co))/2; P:= a×a; a:= (a+b)/2; P:= P - a1;
                S2:= S2 + P × sign (1.5 - n + (n:4)×4)/sqrt
                (a×a +co×co); S1:= S1/2 + P; a1:= A; b:= sqrt (A)
        end; E:= (((2 × arctan (a/co) + (n +(1 - sign (co))/2) ×
        3.14159265359) × (.250 - S1)/a)/2 + S2)/4
    end;
    procedure  EB;
    begin   real a, S1, co, si; real array b [0:3], SIN [0:3];
            integer m; a:= 1; S1:= U × U; b [0]:= V; si:= sin (P);
            co:= cos (P)/si; SIN [0]:= si; b [3]:= 1 + V;
            co:= (co + sqrt (co × co + 1 - V × V))/b [3]; si:= co×co;
```

```
comment continuation of AP 247;
                for m:= 0, 1, 2 do
                begin   SIN [m+1]:= 1/sqrt (1 + si); A:= a × b[m];
                        a:= b [3]/2; b [m+1]:= sqrt (A); P:= a × a;
                        S1:= S1/2 + P - A; b [3]:= a + b [m+1];
                        co:= (a × co + sqrt (P × (si + 1) - A))/b [3];
                        si:= co × co
                        end; si:= sqrt (1 + si); P:= 1/si; U:= 0; b [3]:= a;
                        for m:= 3, 2, 1, 0 do U:= 2 × U + b [m] × (P - SIN [m]);
                        E:= 4 × S1 × ln ((1 + si)/co)/a + P + U
        end;
        U:= abs (cos (A)); V:= abs (sin (A)); if V < .9539 then EA else EB
end E;
```

comment AP 248

B calculates the incomplete elliptic integral:

$$B := \int_0^P \cos^2 x(1 - (\sin^2 A)(\sin^2 x))^{-1/2}\, dx$$

with $0 \leq P \leq \pi/2$, the relative error is of the order $_{10}-12$,
for literature reference see the comment of AP 243;

```
real procedure B (A, P); value A, P; real A, P;
begin   real U, V;
        procedure BA;
        begin   real a,b, S1, S2, co; integer n, m; array p [0:4];
                b:= U; a:= 1; co:= cos(P)/sin (P); S2:= S1:= 0; n:= 0;
                p [0]:= 1; for m:= 1, 2, 3, 4 do
                begin   A:= a×b; co:= (co - A/co)/2; n:= 2×n + (1 - sign
                        (co))/2; p [m]:= .25 × p [m - 1] × (a - b)/(a + b);
                        a:= (a+b)/2;S1:= S1/2 + p [m]; S2:= S2 + p [m - 1]
                        × sign (1.5 - n + (n:4)×4)/sqrt (a×a +co×co);
                        b:= sqrt (A)
                end; B:= (2 × arctan (a/co) + (n +(1 - sign (co))/2) ×
                3.14159265359) × (.015625 - S1/4)/a + S2/4
        end;
        procedure BB;
        begin   real a, S1, co, si; real array b [0:3], SIN [0:3];
                integer m; a:= 1; S1:= - U × U; b [0]:= V; si:= sin (P);
                co:= cos (P)/si; SIN [0]:= si; b [3]:= 1 + V; V:= V × V;
                co:= (co + sqrt (co × co + 1 - V))/b [3]; si:= co × co;
```

```
comment continuation of AP 248;
                    for     m:= 0, 1, 2 do
                    begin   SIN [m+1]:= 1/sqrt (1 + si); A:= a × b[m];
                            a:= b [3]/2; b [m+1]:= sqrt (A); P:= a × a;
                            S1:= S1/2 + P - A; b [3]:= a + b [m+1]; co:= (a × co
                            + sqrt (P × (si + 1) - A))/b [3]; si:= co × co
                    end; si:= sqrt (1 + si); P:= 1/si; U:= 0; b [3]:= a;
                    for m:= 3, 2, 1, 0 do U:= 2 × U + b [m] × (P - SIN [m]);
                    B:= (4 × S1 × ln ((1 + si)/co)/a + P + U)/V
            end;
            U:= abs (cos (A)); V:= abs (sin (A)); if V < .9539 then BA else BB
end B;
```

comment AP 249

If u = F (A, P), where F (A, P) is defined in the comment of AP 246, the inverse function is called the amplitude function P = am (u, A). The Jacobian elliptic functions are then defined by:

sn (u, A) = sin (am (u, A)),

cn (u, A) = cos (am (u, A)) and

dn (u, A) = $(1 - \sin^2 A \, \mathrm{sn}^2 (u, A))^{-1/2}$.

The following procedure calculates sn (u, A), with $0 \leq u \leq K(A)$. The comment of AP 243 contains the literature reference and the definition of K (A). The absolute error is of the order 10^{-12};

```
real procedure sn (u, A); value A, u; real A, u;
begin   real array a [0:3], b [0:3]; real a1, t; integer i;
procedure V1;
begin   a1:= (a [i-1] + b [i-1])/2;
        b [i]:= sqrt (a [i-1] × b [i-1]); a [i]:= a1
end;
procedure V2 (j); integer j; t:= 2 × a [i] × t/(a [i] + b [i] - j × (a [i] - b [i]) × t × t);
a [0]:= 1; b [0]:= abs (sin (A)); if b [0] ≥ .9539
then    begin  for i:= 1, 2 do V1;
               a1:= exp ((a [2] + b [2]) × u); t:= (a1 - 1)/
               (2 × sqrt (a1)); for i:= 2, 1, 0 do V2 (1);
               sn:= t/sqrt (1 + t × t)
        end
else    begin  b [0]:= abs (cos (A)); for i:= 1, 2, 3 do V1;
```

```
comment continuation of AP 249;
                                a1:= (a [3] + b [3]) × u/2; t:= sin (a1);
                                for i:= 3, 2, 1, 0 do V2 (-1); sn:= t
                        end
end sn;
```

```
comment          AP 250
        JEF calculates the Jacobian elliptic functions cn (u, A) and
dn (u, A) and assigns these values to the varibles cn and dn.
See the comment of AP 249;

procedure JEF (u, A, cn, dn); value A, u; real A, u, cn, dn;
begin   real array a [0:3], b [0:3]; real a1, b1, t, k1; integer i;
        procedure V1;
        begin   a1:= (a [i-1] + b [i-1])/2;
                b [i]:= sqrt (a [i-1] × b [i-1]); a [i]:= a1
        end;
        procedure V2 (j); integer j; t:= 2 × a [i] × t/(a [i] + b [i] - j × (a [i] - b [i]) × t × t);
        procedure V3;
        begin t:= (a1 - 1)/(2 × sqrt (a1)); for i:=  2, 1, 0 do V2 (1) end;
        a [0]:= 1; b1:= b [0]:= abs (sin (A)); k1:= abs (cos (A));
        if b [0] ≥ .9539 then
        begin   for i:= 1, 2 do V1;
                a1:= (ln ((a [2] + b [2]) × a [2] × a [1] × a [1])/2
                + 2.42601513194 - 2 × ln (k1))/(a [2] + b [2]);
                if u/a1 > .5 then
                begin   a1:= exp ((a [2] + b [2]) × (a1 - u)); V3;
                        cn:= k1 × t /sqrt (1 + (k1 × t)↑2);
                        dn:= k1 × sqrt ((1 + t × t)/(1 + (k1 × t)↑2))
                end else
```

```
comment continuation of AP 250;
            begin   a1:= exp ((a [2] + b [2]) × u); V3;
                    cn:= 1/sqrt (1 + t × t);
                    dn:= sqrt ((1 + (k1 × t)↑2)/(1 + t × t))
            end
end else
begin   b [0]:= k1; for i:= 1, 2, 3 do V1;
        a1:= (a [3] + b [3]) × u/2; if a1 < .78539816340 then
        begin   t:= sin (a1); for i:= 3, 2, 1, 0 do V2 (-1);
                cn:= sqrt (1 - t × t);
                dn:= sqrt (1 - (b1 × t)↑2)
        end else
        begin   t:= sin (1.57079632679 - a1);
                for i:= 3, 2, 1, 0 do V2 (-1);
                cn:= b [0] × t /sqrt (1 - (b1 × t)↑2);
                dn:= b [0]/sqrt (1 - (b1 × t)↑2)
        end
    end
end JEF;
```