RDVLIB

A library of Algol procedures
on the system common file RDVLIB

R. J. DeVogelaere
Department of Mathematics
Mathematics Laboratory
University of California Berkeley

May 15, 1970

CONTENTS

I. Access to the procedures.

by Ellen Sherman
Mathematics Laboratory

The system common file rdvlib consists of one file divided into 72 logical records, or subfiles. The first record is an index, each of the remaining records contains a group of procedures written in the CDC hardware representation of Algol. These procedures are listed in Part II and documented in Parts III and IV.

The subfiles are indexed and named to facilitate remote access to the library by users of the Algol pre-processors algol8 and editor, but it is, of course, possible to make a punched card copy of the procedures on one or more records for insertion into an Algol card deck. (Because the file rdvlib consists of several thousand card images, the user is warned not to attempt to copy the whole of it to cards, however.) For example, to extract the five procedures on sle1 (record number 14) for solving systems of linear equations and inverting matrices and the procedures SUM and INPROD (subfile sum, record number 13) used by the procedures on sle1, submit the following j-job:

```
jxxxx,7,10,10000,,200.name
common(rdvlib)
copybr(rdvlib,null,12)
copybr(rdvlib,lfn,2)
rewind(lfn)
copybf(lfn,punch)
rewind(lfn)
copysbf(lfn,output)
<end-of-transmission>
```

Of course, if a listing of the procedures is not required, the desired record, or records, can be copied directly to punch.

The Algol pre-processors algol8 and editor (see the Guide to the Mathematics Department Laboratory and its Supplements) both can insert procedure declarations from rdvlib into the transliterated text of the user's program. To use this feature, the program call for algol8 or editor must be preceded by the line common(rdvlib) and the file name rdvlib must be given as the third parameter of algol8 or the fourth of editor:

```
common(rdvlib)
algol8(o,d,rdvlib)
```

or

```
common(rdvlib)
lgo,editor(zz00,o,d,rdvlib)
```

The requests to the pre-processor for named subfiles are made by means of appropriate comment's, but any such request must be preceded by comment: library; . This comment is then followed by one or more comments of the form comment: <subfile name>; given at that point in the program at which the procedure declarations on the subfile are to be inserted. Thus, for example, to use the procedures DET1 and SOL1 in a program to solve a system of linear equations, the input to algol8 or editor might begin:

```
begin array a[1:5,1:5], b[1:5]; integer array pivot[1:5];
      comment: library; comment: sle1;
      comment: sum;
```

The output to the teletype from algol8 or editor will not include the actual text of the procedure declarations inserted but the line numbers, if given, will correspond to the actual source text written for the Algol compiler. The listing from algol (requested with the l-option in the algol control card) would, of course, include the source text of the procedures inserted. If comment: library; does not precede the requests for the named subfiles, the requests are ignored, i.e., treated as ordinary comment's. Likewise, a request for a subfile name not given in the index is treated as an ordinary comment.

The subfiles t0, t1, t2, and t3 (the second, third, fourth, and fifth records) have been left empty so that it is possible for the user to create a library file which is a copy of the file rdvlib except that one or more of the empty records t0, t1, t2, and t3 have been replaced by records containing his own procedures. This is a very convenient method for inserting already debugged procedures into programs being pre-processed by algol8 or editor.

Such a library file is usually a user common file and can have any logical file name, say zzzlib. Creation of this file is discussed below. To use the library file zzzlib, rather than rdvlib, the line common(rdvlib) must be replaced by common(zzzlib,sc) and zzzlib must replace rdvlib as a parameter of algol8 or editor. All the procedures on rdvlib can then be inserted from zzzlib by the pre-processor and, in addition, the user's own procedures can be inserted from subfiles t0, t1, t2, and t3. The two comments comment: library; and, say, comment: t0; replace the corresponding procedure declarations, thereby reducing the input to algol8 or the size of the file to be processed by editor.

The following job to be submitted from a remote terminal will create a user common file named zzzlib, or alter a pre-existing file zzzlib, copy the first record of rdvlib (the index) to zzzlib, insert the CDC-transliteration of the procedure declarations input between the ←1 and the δ1 in place of the empty record t0, and copy the remainder of rdvlib to zzzlib:

```
<r-job card>
common(rdvlib)
common(zzzlib,wr)
algol8(o)
copybr(rdvlib,zzzlib,1)
copybr(rdvlib,null)
copybr(o,zzzlib)
copybf(rdvlib,zzzlib)
 β00
 ←1
<procedure declarations>
 δ1
 δ
```

The β and δ represent control-b and control-d, respectively. The procedures to be transliterated by algol8 are to be typed according to the same conventions described in the Guide to the Mathematics Laboratory. The output to the teletype will be a reference language copy of the declarations with line numbers, unless the line numbers are suppressed by typing ←112 in place of ←1 or the entire output is suppressed by using ←110. To place procedures on subfiles t1, t2, or t3, the 1 in the first copybr-control card must be replaced by 2, 3, or 4, respectively.

Because user common files are somewhat temporary (no file lasts more than 24 hours, files are purged if they go unused for seven hours, or four hours during peak periods), the user must be prepared to re-create his library file from time to time. Of course, the above procedure can be repeated using paper tape input. However, it would be somewhat more efficient to have a card deck of the transliterated version of the user's procedures. The user's subfile can be copied to punch from his library file by the same method used above to extract sum and sle1 from rdvlib. Because this library file is at least as large as the file rdvlib it is, again, completely impractical to copy the whole of it to punched cards.

To create, or alter, a user common file zzzlib from rdvlib and a card deck of procedure declarations written in CDC-Algol source language, submit the following job over the input counter:

```
JXXXX,7,10,10000.NAME
COMMON(RDVLIB)
COMMON(ZZZLIB,WR)
COPYBR(RDVLIB,ZZZLIB,1)
COPYBR(RDVLIB,NULL)
COPYBR(INPUT,ZZZLIB)
COPYBF(RDVLIB,ZZZLIB)
<7-8-9 card>
<procedure declarations in CDC-Algol source text>
<7-8-9 card>
<6-7-8-9 card>
```

Here the additional procedure declarations have been placed on subfile t0 of zzzlib.

II. Contents of file rdvlib 03/29/1968

This library file has been prepared by Professor Rene De Vogelaere, originally for the laboratory of Math 128B of spring 1966. The page initials refer to the author or source of the procedures:

ALG refers to an ALGORITHM in the Communications of the ACM.
AP refers to a procedure written at the Mathematisch Centrum of Amsterdam.
M refers to a procedure written by miscellaneous authors.
RC refers to a procedure written at the Regnecentralen of Copenhagen.
RDV refers to a procedure written by R. De Vogelaere.
RKZ refers to a procedure written by Dr. Zonneveld for his variant of Runge-Kutta methods.

When the page number contains a decimal point, the original algorithm has been modified.

no.	subfile	page content	date	procedure content
2.	t0	reserved for the user		
3.	t1	reserved for the user		
4.	t2	reserved for the user		
5.	t3	reserved for the user		
6.	rdv6801	reserved for the instructor		
7.	rdv6802	reserved for the instructor		
8.	rdv6803	reserved for the instructor		
9.	rdv6804	reserved for the instructor		
10.	rme 1	AP 230	03/09/1966	<u>real</u> <u>procedure</u> ZERO
		AP 236	03/09/1966	<u>real</u> <u>procedure</u> ZEREX
		AP 237	03/09/1966	<u>real</u> <u>procedure</u> POL
11.	rdvio	RDV 00	04/01/1966	<u>integer</u> <u>procedure</u> readi <u>real</u> <u>procedure</u> readr <u>Boolean</u> <u>procedure</u> readb <u>integer</u> <u>procedure</u> ioi <u>real</u> <u>procedure</u> ior <u>Boolean</u> <u>procedure</u> iob <u>procedure</u> ioa
		RDV 01	06/13/1966	<u>procedure</u> outputi <u>procedure</u> outputr <u>procedure</u> outputb <u>procedure</u> outputa

	RDV 03	04/01/1966	procedure oti procedure otr procedure otb procedure ota procedure outi procedure outr procedure outb procedure outa
12. outputm	RDV 02	06/13/1966	procedure outputm
13. sum	AP 119	03/09/1966	real procedure SUM
	AP 120	03/09/1966	real procedure INPROD
14. sle 1	AP 204.1	03/09/1966	real procedure DET 1
	AP 205.1	03/09/1966	procedure SOL 1
	AP 206.1	03/09/1966	procedure INV 1
	AP 207.1	03/09/1966	real procedure DETSOL 1
	AP 208.1	03/09/1966	real procedure DETINV 1
15. ssle 1	AP 224	03/09/1966	real procedure SYMDET 1
	AP 225	03/09/1966	procedure SYMSOL 1
	AP 226	03/09/1966	procedure SYMINV 1
16. evvsle 1	AP 231	04/01/1966	procedure SPAP
	AP 232	04/01/1966	real procedure SEIGENVA
	AP 233	04/01/1966	procedure SEIGENVEC
	AP 234	04/01/1966	procedure STRASF
	AP 235	04/01/1966	procedure SEVAVEC
17. evvle 1	AP 238	04/01/1966	procedure APAP
	AP 239	04/01/1966	real procedure REIGENVA
	AP 240	04/01/1966	procedure REIGENVEC

	AP 241	04/01/1966	<u>procedure</u> ATRASF
	AP 242	04/01/1966	<u>procedure</u> REVAVEC
18. evvsle2	ALG 254	01/09/1968	<u>procedure</u> symmetric QR2
19. iter4	RDV 67 - 24	12/06/1967	<u>procedure</u> ITER4
	RDV 67 - 25	12/06/1967 12/06/1967	<u>procedure</u> in parameters <u>procedure</u> results of iteration
	RDV 67 - 18	10/31/1967	<u>real</u> <u>procedure</u> DIST
20. acciter	RDV 68 - 1	01/09/1968	<u>Boolean</u> <u>procedure</u> accelerated iteration
	RDV 68 - 3	01/09/1968	<u>Boolean</u> <u>procedure</u> accelerate
21. rdv682	RDV 68 - 2	01/09/1968	<u>Boolean</u> <u>procedure</u> FUNCT
22. ssle 2	AP 228	03/09/1966	<u>real</u> <u>procedure</u> SYMDET 2
	AP 229	03/09/1966	<u>procedure</u> SYMSOL 2
23. fv 1	AP 201.1	12/06/1967	<u>real</u> <u>procedure</u> MAX
	AP 202	04/07/1966	<u>real</u> <u>procedure</u> PROD
24. rne3	RDV 67 - 21	12/06/67	<u>Boolean</u> <u>procedure</u> deflation for nle
25. rne4	RDV 67 - 28	12/06/67	<u>Boolean</u> <u>procedure</u> deflation for nle
26. pi 1	RDV 65 - 1	03/09/1966	<u>procedure</u> Grunert analysis
	RDV 65 - 2	03/09/1966	<u>procedure</u> Newton analysis
	RDV 65 - 3	03/09/1966	<u>procedure</u> Newton harmonics
	RDV 65 - 4	03/09/1966	<u>real</u> <u>procedure</u> EVAL 2
	RDV 65 - 7	03/09/1966	<u>procedure</u> Grunert to Newton
	RDV 65 - 9	03/09/1966	<u>procedure</u> Newton to Grunert

	RDV 65 - 10	03/09/1966	procedure Chebyshev Gram synth and anal
27. pi 2	RDV 65 - 5	03/09/1966	procedure Lagrange synthesis diagonal
	RDV 65 - 6	03/09/1966	procedure Lagrange analysis diagonal
	RDV 65 - 8	03/09/1966	Boolean procedure AND
28. pi 3	RDV 66 - 1	04/07/1966	real procedure Exact Fit Interpolation
29. os 1	RDV 65 - 11	04/01/1966	procedure mult MS procedure one over MS procedure quot MS procedure square MS
	RDV 65 - 12	04/01/1966	procedure int 1 MS procedure int 2 MS
30. rdv 65 16	RDV 65 - 16	03/09/1966	integer procedure next prime
31. has 1		10/30/1967	Boolean b first
	RDV 63 - 3	04/07/1966	procedure HARSUM
	RDV 63 - 4	04/01/1966	procedure HAS
	RDV 67 - 15	10/30/1967	procedure otha
32. pol 1	RDV 66 - 4	03/09/1966	procedure mapping Bairstow
	RDV 66 - 5	03/14/1966	real procedure POL m
33. me 2	AP 216	03/19/1966	real procedure CSQRT
	AP 217	03/19/1966	real procedure CZERO
	AP 218	03/30/1966	real procedure CPROD
	AP 219	03/30/1966	real procedure CPOL
34. romberg	RC 157	05/01/1966	real procedure Romberg

35.	rkz 1n 5	RKZ 1n 5	03/25/1966	procedure RK1n
36.	rkz 2n 5	RKZ 2n 5	03/25/1966	procedure RK2n
37.	rkz 3n 5	RKZ 3n 5	03/25/1966	procedure RK3n
38.	rkz 4n 5	RKZ 4n 5	03/25/1966	procedure RK4n
39.	rdv 63 05	RDV 63 - 05	03/19/1966	procedure DUFFING 2
40.	rdv 65 13	RDV 65 - 13	04/01/1966	procedure Duffing 1
41.	rdv 65 14	RDV 65 - 14	04/01/1966	procedure Duffing 1
42.	strat1	RDV 65 - 17	01/08/1966	procedure strategy
43.	strat2	RDV 65 - 18	01/08/1966	procedure strategy
44.	temp 1			
45.	rkz 1n 4	RKZ 1n 4	04/18/1966	procedure RK 1n
46.	bessel	RC 173	09/07/1966	real procedure Bessel Jhalf
		RC 177	09/07/1965	real procedure Bessel K
		RC 178	09/07/1966	real procedure Bessel I
47.	elliptic	RC 180	09/07/1966	real procedure incompl ellip 1
		RC 181	09/07/1966	real procedure incompl ellip 2
		RC 182	09/07/1966	real procedure compl ellip 1
		RC 183	09/07/1966	real procedure compl ellip 2
48.	has 2	RC 174	09/07/1966	real procedure Fourier synthesis
		RC 175	09/07/1966	real procedure Fourier analysis

49. rkz 5nx	RC 303	09/07/1966	procedure RK fifth order x
50. rkz 5na	RC 304	09/07/1966	procedure RK fifth order arc
51. rkz 1	RC 308	09/07/1966	procedure rkz 1
	ALG 38	12/10/1967	procedure Telescope 2
52. rkz 2	RC 309	09/07/1966	procedure rkz 2
53. am io			
54. z8096	z8095	12/06/67	integer insymbol,outsymbol; procedure initialise procedure error
	z8096	09/05/1966	integer z8096j; integer array z8096t; procedure initialise in table
	z8097	09/05/1966	procedure in symbol integer z8097j; integer array z8097t;
	z8098	09/05/1966	procedure initialise out table
	z8099	09/05/1966	procedure out symbol procedure out string
55. z8100	z8100	09/05/1966	
	z8101	09/05/1966	
	z8102	09/05/1966	
	z8103	09/05/1966	
56. z8104	z8104	09/05/1966	
	z8105	09/05/1966	
57. z8106	z8106	09/05/1966	
	z8107	09/05/1966	
	z8108	09/05/1966	
	z8109	09/05/1966	
58. z8110	z8110	09/05/1966	
	z8111	09/05/1966	
59. z8112	z8112	09/05/1966	
	z8113	09/05/1966	
	z8114	09/05/1966	
	z8115	09/05/1966	

	z8116	09/05/1966	
	z8117	09/05/1966	
	z8118	09/05/1966	
60. z8119	z8119	09/05/1966	
	z8120	09/05/1966	
	z8121	09/05/1966	
	z8122	09/05/1966	
	z8123	09/05/1966	
	z8124	09/05/1966	
	z8125	09/05/1966	
	z8126	09/05/1966	
61. iter1			
62. slee1	ALG 290.1	12/06/1967	integer procedure exact 1 e
63. gtoortho	RDV 67 - 14	12/06/1967	procedure Grunert to orthogonal
64. diophantine1	ALG 287	12/10/1967	integer procedure INTRANK
	ALG 288	12/10/1967	Boolean procedure SOLVE INTEGER
65. expfit	ALG 295	12/10/1967	procedure expfit
66. economise1	ALG 37	12/10/1967	procedure Telescope 1
	ALG 38	12/10/1967	procedure Telescope 2
67. pol4	ALG 29	12/10/1967	procedure POLYX
68. integration1	ALG 198	12/31/1967	real procedure Integral
69. chebfit	ALG 318	01/09/1968	procedure chebfit
70. b			
71. b1			
72. iter3			

III. Brief descriptions of the records or procedures.

Part III contains a brief description of each record of the rdvlib. The procedures themselves may be more general, so that the user should refer to the complete write-up of a procedure he plans to use. In particular, he is warned to read at least the comment preceding the procedure because these comments give the names of any non-local procedures called. These non-local procedures may be found on other subfiles of rdvlib, such as the frequently used procedures SUM and INPROD which are on the subfile sum of rdvlib; or, they may be found on the system common file bclib , such as the basic input/output procedures used by the procedures on subfile rdvio ; or, the user may have to declare them himself.

t0 - t3	user can insert procedures
rdv6801-rdv6804	instructor can insert procedures
rne1	for evaluating polynomials and finding one or several zeros
rdvio	input/output procedures
outputm	outputs a two-dimensional matrix given the matrix and its bounds
sum	SUM gives the sum of array elements INPROD gives the inner product of array elements
sle1	for solving systems of linear equations and inverting matrices
ssle1	for solving symmetric systems of equations and finding inverses of symmetric matrices
evvsle1	for finding eigenvalues and eigenvectors of symmetric matrices
evvle1	for finding eigenvalues and eigenvectors of non-symmetric matrices
evvsle2	for finding eigenvalues and eigenvectors of symmetric matrices -- a QR algorithm
iter4	for help in controlling convergence of iteration procedures

acciter	for doing accelerated iteration on vectors
rdv 68 2	an example of a procedure to solve, by iteration, a boundary value problem of ordinary differential equations
ssle2	for solving systems of symmetric linear equations
fv1	MAX gives the maximum component of a vector PROD gives the product of array elements
rne3 rne4	for solving non-linear equations by deflation
pi1	Grunert, Newton, and Chebyschev-Gram interpolation
pi2	LaGrange interpolation AND gives the logical product of the elements of a Boolean array
pi3	general procedures to do exact fit linear interpolation
os1	set of procedures to facilitate the construction of Taylor Series for functions satisfying ordinary differential equations
rdv 65 16	for constructing the first p primes
has1	harmonic analysis and synthesis of periodic functions and a suitable output procedure
pol1	Bairstow mapping for finding quadratic factors of polynomials POLm evaluates a polynomial and its first m derivatives
rne2	for finding the square root of complex numbers, the root of a function of a complex variable, and the product of complex array elements, and for evaluating a polynomial with real coefficients at a complex number
romberg	for integrating a function by Romberg's method
rkz 1n5 rkz 2n5 rkz 3n5 rkz 4n5	for finding solutions of ordinary differential equations by the Runge-Kutta method with automatic adjustment of the interval of integration a la Zonneveld
rdv 63 05	for determining a periodic solution of Duffing's equation by plain integration
rdv 65 13 rdv 65 14	application of the procedures of os1 to Duffing's equation
strat1 strat2	two examples of developing the optimal strategy for a type of two-person game

templ	
rkz 1n4	(see rkz 1n5)
bessel	for computing Bessel functions
elliptic	for computing complete and incomplete elliptic integrals of the first and second kinds
has2	Fourier analysis and synthesis of periodic functions
rkz 5nx rkz 5na	Runge-Kutta Zonneveld procedures (see the description above and the procedure write-ups)
rkz 1 rkz 2	Runge-Kutta Zonneveld procedures
am io	input/output procedures which allow compatibility with programs written at Mathematisch Centrum Amsterdam
z8096	basic input/output procedures for the procedures on z8100 - z8126
z8100 z8104 z8106 z8110 z8112 z8119	input/output procedures written in terms of the basic procedures insymbol, outsymbol, and outstring. Published: R. DeVogelaere, ''Algorithms 335 A Set of Basic Input/Output Procedures,'' Communications of the Association for Computing Machinery, Vol. 11, No. 8, August 1968
iter1	
slee1	for finding an exact solution to linear equations whose coefficients are integers
gtoortho	for obtaining the coefficients of polynomials from the recurrence relations they satisfy
diophantine1	for solving simultaneous linear diophantine equations
expfit	for fitting an exponential curve by least squares
economise1	for reducing the degree of an approximating polynomial
pol4	for finding the coefficients of a transformed polynomial
integration1	for numerical integration by the Newton-Cotes method with automatic adaptation of the step size
chebfit	for fitting a Chebyschev curve
b	
b1	
iter3	

```
comment                       RDV 00;
integer procedure readi;
begin           integer i; input integer(i); readi := i end;

real procedure readr;
begin           real r; input real(r); readr := r end;

Boolean procedure readb;
begin           Boolean b; input boolean(b); readb := b end;

integer procedure ioi(i,s,n); string s; integer i,n;
begin           text(s); text('␣:=␣'); input integer(i); ioi := i;
                integer format(n); output integer(i); text(';␣')
end;

real procedure ior(r,s,B,n,d); real r; string s; Boolean B; integer n,d;
begin           text(s); text('␣:=␣'); input real(r); ior := r;
                real format(B,n,d); output real(r); text(';␣')
end;

Boolean procedure iob(B,s,n); Boolean B; string s; integer n;
begin           text(s); text('␣:=␣'); input boolean(B); iob := B;
                if B then text('true') else text('false');
                text(';␣')
end;

procedure ioa(a,l,u,s,B,n,d); integer l,u,n,d; array a; string s; Boolean B;
begin   integer i;
                if l > u then go to end;
                real format(B,n,d); oti(l,'i',3); text('␣for␣'); text(s); text('[i]␣:=␣');
                for i := l step 1 until u do
                begin input real(a[i]); output real(a[i]); if i < u then text(',␣') else text('␣do␣i␣:=␣i+1;␣') end;
end:
end;
```

```
comment                        RDV 01;

procedure outputi(i); value i; integer i;
begin    integer format(if i = 0 then 2 else entier(ln(abs(i)) × 0.43429449+_10-7) + 3);
         output integer(i); spaces(2)
end;

procedure outputr(r); value r; real r;
begin    integer n,i;
         if r = 0 then begin integer format(2); output integer(0); go to end end;
         n := entier(ln(abs(r)) × 0.43429449+_10-7);
         if n = 8 then begin integer format(11); output integer(entier(r)); go to end end;
         if abs(n) > 8 then real format(false,16,8) else
         if n < - 1 then
         begin    if r ≥ 0 then spaces(2) else text('␣-');
                  real format(true, 2,0); text('0.');
                  for i := n step 1 until - 2 do text ('0');
                  integer format(9); output integer(entier(r × 10.0↑(8-n)));
                  go to end
         end
         else
         real format(true,12-(if n ≥ 0 then 0 else n),8-n);
         output real(r);
end:     spaces(2)
end;

procedure outputb(B); Boolean B;
if B then text ('␣ true ␣ ␣') else text ('␣ false ␣ ␣');

procedure outputa(a,l,u); array a; integer l,u;
begin    integer i; for i := l step 1 until u do outputr(a[i]) end;
```

```
comment                              RDV 02;

procedure outputm(a,l1,u1,l2,u2); array a; integer l1,u1,l2,u2;
begin  integer i,j,gn,k,n; real max,t; Boolean zero; integer  array global n [l2:u2];
       for j := l2 step 1 until u2 do
       begin   max := abs(a[l1,j]);
               for i := l1+1 step 1 until u1 do
               begin t := abs(a[i,j]); if t > max then max := t end;
               global n[j] := if max = 0 then -40 else entier(ln(max) × 0.43429449+⏨-7)
       end; nlcr;
       for i := l1 step 1 until u1 do
       begin   for j := l2 step 1 until u2 do
               begin   gn := global n[j];
                       if gn = -40 then begin integer format(2); outputinteger(0);  go to next end;
                       t := a[i,j];
                       if gn = 8 then begin integer format(11); outputinteger(entier(t));  go to next end;
                       if abs(gn) > 8 then real format(false,16,8) else
                       if gn < -1 then
                       begin      n := entier(ln(abs(t)) × 0.43429449+⏨-7);
                                  if t ≥ 0 then spaces(2)  else text('␣-');
                                  real format(true,2,0); text('0.');
                                  zero := n < gn -9;
                                  for k := if zero then gn-9 else n  step 1 until -2 do text('0');
                                  if ¬ zero then
                                  begin integer format(9 -gn +n);  outputinteger(abs(t × 10.0↑(8-gn)))end;
                                  go to next
                       end
                       else real format(true,12-(if gn ≥0then 0 else gn), 8-gn) ;
                       outputreal(t);
              next:    spaces(2)
              end; nlcr
       end
end outputm;
```

```
comment                 RDV 03;

procedure oti(i,s,n); string s; integer i,n;
begin   text(s); text('_:=_');
        integer format(n); output integer(i); text(';_')
end;
procedure otr(r,s,B,n,d); real r; string s; Boolean B; integer n,d;
begin   text(s); text('_:=_');
        real format(B,n,d); output real(r); text(';_')
end;
procedure otb(B,s,n); Boolean B; string s; integer n;
begin   text(s); text('_:=_');
        boolean format(n); output boolean(B);
        text(';_')
end;
procedure ota(a,l,u,s,B,n,d); integer l,u,n,d; array a; string s; Boolean B;
begin   integer i;
        if l > u then go to end;
        real format(B,n,d); oti(l,'i',3); text('_for_'); text(s); text('[i]_:=_');
        for i := l step 1 until u do
        begin output real(a[i]); if i < u then text(',_') else text('_do_i_:=_i+1;_') end;
end:
end;

procedure outi(i,n); integer i,n;
begin integer format(n); output integer(i) end;

procedure outr(r,B,n,d); real r;  Boolean B; integer n,d;
begin real format(B,n,d); output real(r) end;

procedure outb(B,n); Boolean B; integer n;
begin if B then text('true') else text('false') end;

procedure outa(a,l,u,B,n,d); integer l,u,n,d; array a; Boolean B;
begin   integer i;
        if l > u then go to end;
        real format(B,n,d); for i := l step 1 until u do output real(a[i]);
end:
end;
```

DISTANCE BETWEEN VECTORS

Data: x and x1 are the vectors
l and u are their respective lower and upper bound.

Result: DIST and d is the distance between the two vectors corresponding to the uniform norm or infinity norm;

```
real procedure DIST (x,x1,l,u,d); value l,u; integer l,u; real d; array x,x1;

begin       integer i; real v;

            d := 0; for i := 1 step 1 until u do

            begin v :=abs(x[i]-x1[i]); d := if d < v then v else d end;

            DIST := d

end DIST;
```

comment

AN ITERATION CONTROL PROCEDURE

Data:
- x0[l:u] is the first guess,
- l,u are respectively the subscript bounds of the vectors x0,x, p[1:4] and ip[1:7] are such that
- p[1] defines the domain in which we want the iterates x to remain, i.e., a test is made to insure that the distance between x0 and x remains smaller than p[1],
- p[2] is the maximum error desired,
- p[3] , if larger than p[2], is the error we will accept if the iteration eventually and apparently diverges,
- ip[1] is the maximum number of iterations allowed,
- ip[2] is the maximum number of iterations for which the process appears to be divergent,
- abs(ip[3]) is the order of the method: 1 or 2, if ip[3] is negative, the estimated error is required to be smaller than p[3], otherwise both the distance between the last two iterates and the estimated error are required to be smaller than p[3],
- FUNCT (xi,xf,l,u) is the value of a Boolean procedure which defines the iterate xf of xi and has the value false if that iterate can be evaluated and has the value true otherwise,
- DIST (xi,xf,l,u,d) is the value of a real procedure which defines the distance d or DIST between xi and xf, in both case l and u are the subscript bounds of the arrays xi and xf.

Results:
- x[l:u] is the desired iterate or the best iterate obtained,
- p[4] ,if ip[6] = 1, is an estimate of the error assuming that the computations have been performed with much greater precision than p[2],
- ip[4] is the number of iterates computed,
- ip[5] is the number of iterations which appear to diverge,
- ip[6] is 1 if the iteration succeeds or starts to diverge when a precision p[3] is attained and is 2 if the iteration fails,
- ip[7]
  - is 0 if the iteration succeeds,
  - is 1 if the iteration starts to diverge after the precision p[3] is attained,
  - is 2 if the iteration diverges too often,
  - is 3 if the iterates leave the domain defined by p[1],
  - is 4 if the number of iterations is equal to ip[1],
  - is 5 if the number of iterations is equal to ip[1] and the last step is divergent or very slowly convergent,
  - is 6 if one of the last iterates could not be evaluated.

The procedure uses the non local procedure END JOB which decides what to do if some of the data is unacceptable.

```
procedure ITER4 (x0,x,l,u,p,ip,FUNCT,DIST); value l,u; integer l,u; array x0,x,p;
integer array ip; Boolean procedure FUNCT; real procedure DIST;
begin   integer i,j; real a,eps,error,d,dp,ek,ekp; Boolean stingy; array x1[l:u];
        real procedure MAX1(a,b); real a,b; MAX1:= if a < b then b else a;
        a := p[4] := p[1]; error := eps := p[2]; i := ip[3]; ip[6] := 1; j:= ip[7] := 0;
        ip[4] := 1; stingy := i < 0; ip[3] := i := abs(i);
        if eps ≤ 0 ∨ i = 0 ∨ i > 2 ∨ ip[1] ≤ 0 then END JOB;
        if FUNCT (x0,x1,l,u) then begin ip[7] := 6; go to iteration fails end;
        if DIST (x0,x1,l,u,dp) < eps then
        begin   for i := 1 step 1 until u do x[i] := x1[i]; go to end end;
        if dp > a then
        begin   for i := 1 step 1 until u do x[i] := x1[i]; go to leave domain end;
        p[4] := dp; ekp := 0;

iterate: ip[4] := ip[4] + 1; if FUNCT (x1,x,l,u) then begin ip[7] := 6;  go to iteration fails end;
        ek := DIST (x,x1,l,u,d) / dp;
        if ek < .99 ∧ ( d < eps ∨ stingy) then
        begin   error := d × MAX1(ek,ekp) × ( if ip[3] = 2 then ek else 1) /  (1 - ek);
                if error < eps then go to end
        end;
        if ip[4] ≥ ip[1] then
        begin   if ek ≥ 0.99 then begin j := j+1; ip[7] := 5; go to apparent divergence end;
                error := d × MAX1(ek,ekp) × ( if ip[3] = 2 then ek else 1) /  (1-ek);
                ip[7] := 4; go to iteration fails
        end;

        p[4] := dp := d;
        if DIST (x0,x,l,u,d) > a then go to leave domain;
        if ek < 0.99 then ekp := ek else
        begin   j := j+1;
                if dp ≤ p[3] then
                begin error := dp; ip[7] := 1; for i := 1 step 1 until u do x[i] := x1[i];
                        go to end
                end;
                if j ≥ip[2] then begin ip[7]:=2; goto apparent divergence end
        end;
        for i := 1 step 1 until u do x1[i] := x[i]; go to iterate;
leave domain: ip[7] := 3;
apparent divergence: error := a;
iteration fails: ip[6] := 2;
end:    p[4] := error; ip[5] := j
end ITER4;
```

```
procedure in parameters(p,ip); array p; integer array ip;
comment after starting a new line, this procedure inputs and outputs the parameters required by ITER4,
        the output is on two lines, the declarations must be compatible with p[1:4] and ip[1:7];
begin   nlcr; ior(p[1],'radius␣of␣domain',true,8,3);
        ior(p[2],'maximum␣error␣desired',true,11,8); nlcr;
        ior(p[3],'error␣accepted',true,11,8); nlcr;
        ioi(ip[1],'number␣of␣iterations␣allowed',3);
        ioi(ip[2],'maximum␣number␣of␣divergent␣iterations',2); nlcr;
        ioi(ip[3],'order␣of␣the␣method',1); nlcr;
end in parameters;

procedure results of iteration(x,l,u,p,ip,s,B,n,d); integer l,u,n,d; array x,p; integer array ip; string s;
Boolean B;
comment the success or failure of the iteration is recorded and in the case of success the best iterate is
        printed using the procedure ioa of RDV 00, the name of the result is given through the string s,
        other format information is given through B, n and d;
begin   nlcr;
        if ip[6] = 2 then
        begin   text('The␣iteration␣fails␣after'); outputi(ip[4]); text('␣steps.'); nlcr;
                if ip[7] = 2 then
                begin text('It␣appears␣to␣diverge␣more␣often␣than'); outputi(ip[2]); text('times') end
           else if ip[7] = 3 then
                begin text('The␣iterates␣leave␣the␣domain␣centered␣at␣the␣ first␣guess'); nlcr;
        text('and␣with␣radius'); outputr(p[1])
                end
           else if ip[7] = 5 then text('The␣last␣step␣is␣divergent␣or␣slowly␣convergent.')
           else if ip[7] = 6 then text('The␣last␣iterate␣could␣not␣be␣evaluated.');
                nlcr
        end
   else begin   text('The␣iteration␣succeeds␣after'); outputi(ip[4]);
                text('␣iterations␣with␣an␣error␣which,␣without␣truncation␣ would␣probably␣be␣smaller␣than');
                if ip[7] = 0 then outputr(p[2])
           else begin outputr(p[3]); nlcr; text('␣but␣it␣eventually␣appears␣to␣ diverge') end; nlcr;
                text('␣the␣best␣iterate␣obtained␣corresponds␣to␣the␣ALGOL␣ statements:␣'); nlcr; ota(x,l,u,s,B,n,d)
        end
end results of iteration;
```

comment RDV 68 - 1;

```
Boolean procedure accelerated iteration(x0,x1,x2,x3,x,l,u); value l,u; integer l,u; array x0,x1,x2,x3,x;
comment given the arrays x0,x1,x2,x3[l:u], this procedure determines the array  x[l:u], such that for some a,b,
            x1[i] = a[i] × x0[i] + b[i], x3[i] = a[i] × x2[i] + b[i].
If the result is too large, accelerated iteration is given the value true,  otherwise, the value false;
begin       integer i; real x23,den;
            accelerated iteration :- false;
            for i := l step 1 until u do
            begin   x23 :- x2[i] - x3[i]; den := x0[i] - x1[i] - x23;
                    if abs(den) < ₁₀-20 then
                    begin   if abs(x23) < ₁₀-20 then x[i] := x3[i]
                       else begin accelerated iteration := true; go to end end
                    end
               else x[i] := x3[i] - (x1[i] - x3[i]) × x23 / den
            end;
end:
end accelerated iteration;
```

comment RDV 68 - 3;

```
Boolean procedure accelerate(x0,x,l,u); value l,u; integer l,u; array x0,x;
comment given the nonlocal Boolean procedure FUNCT(x0,x,l,u) which computes  an iterate x[l:u] of
a given array x0[l:u] and is given the value true if the iterate cannot be  obtained,
given the array x0[l:u], this procedure determines the iterate x1 of x0 and  x2 of x1, both by FUNCT
then the array x[l:u] obtained by component by component accelerated iteration.
It uses the nonlocal Boolean procedure accelerated iteration (RDV 68 - 1).
accelerate is given the value true if some failure occurs in FUNCT or  accelerated iteration;
begin    array x1,x2[l:u];
         accelerate := if FUNCT(x0,x1,l,u) then true
                    else if FUNCT(x1,x2,l,u) then true
                    else accelerated iteration(x0,x1,x1,x2,x,l,u)
end accelerate;
```

```
comment        AP 119

               SUM delivers the sum of the successive values of the parameter  ti  obtained by
replacement of  i  in succession by  h, h+1,...,k.  If  h>k  then SUM:=0;

real procedure SUM (i,h,k,ti); value k; integer i,h,k; real ti;

begin          real s; s:=0; i:=h; go to test;

next:          s:= s+ti; i := i + 1;

test:          if i ≤ k  then go to next;  SUM:=s

end;
```

```
comment        AP 120

INPROD is the sum of the products of  x  and  y  evaluated at  k  where  k  is in the ordered
set  a step 1 until b (a ≤ b);

real procedure INPROD (k,a,b,x,y); value a,b; integer k,a,b; real x,y;

begin real p;

        p := 0;

        for k := a step 1 until b do p := p + x × y;

INPROD := p

 end INPROD;
```

```
comment                AP 201.1        12/06/67

                 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 MAX := 0;

real procedure MAX(k,a,b,fk); value a,b; integer k,a,b; real fk;

begin   real r,s;
        k := a; s := if k ≤ b then fk else 0; go to MC;
MB:     k := k + 1; r := fk; if r > s then begin s := r; a := k end;
MC:     if k < b then  go to MB;
        k := a; MAX := s

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
            DET 1:= determinant of the n-th order matrix A which is given as array A[l:u,l:u].  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[l:u] is an output vector given 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 1 uses the non-local real procedure INPROD;

real procedure DET 1 (A,l,u,p); value l,u; integer l,u; array A; integer array p;

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

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

            d:= 1;

            for k:= l step 1 until u do

            begin   r:= - 1;

                    for i:= k step 1 until u do

                    begin   A[i,k]:= A[i,k] - INPROD (j,l,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 := l step 1 until u do

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

                            (A[p[k],j] - INPROD (i,l,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 1:= d
end DET 1;
```

comment AP 205.1

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

```
procedure SOL 1 (LU,b,l,u,p); value l,u; integer l,u; array LU,b; integer array p;

begin       integer i,k; real r;

            for k:= l step 1 until u do

            begin   r:= b[k];

                    b[k]:= (b[p[k]] - INPROD (i,l,k - 1,LU[k,i],b[i])) / LU[k,k];

                    if p[k] ≠ k then b[p[k]]:= - r

            end;

            for k:= u step - 1 until l do

            b[k]:= b[k] - INPROD (i,k + 1,u,LU[k,i],b[i])

end SOL 1;
```

```
comment      AP 206.1

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

procedure INV 1 (LU,l,u,p); value l,u; integer l,u; array LU; integer array p;

begin        integer i,j,k; real r; array v[l: u];

             for k:= u step - 1 until l do

             begin   for j:= k + 1 step 1 until u 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 l do

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

                     for j:= l step 1 until u do

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

             end;

             for k:= u step - 1 until l do

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

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

             end

end INV 1;
```

```
comment     AP 207.1

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

real procedure DETSOL 1 (A,b,l,u);

value l,u; integer l,u; array A,b;

begin       integer array p[l:u];

            DETSOL 1:= DET 1 (A,l,u,p); SOL 1 (A,b,l,u,p)

end DETSOL 1;
```

```
comment     AP 208.1

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

real procedure DETINV 1 (A,l,u);

value l,u; integer l,u; array A;

begin       integer array p[l:u];

            DETINV 1:= DET 1 (A,l,u,p); INV 1 (A,l,u,p)

end DETINV 1;
```

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 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 $1 \leq i \leq j \leq n$, 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. Consequently, 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
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 ≤ 2 × (abs (x × 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; go to entry;

go on:      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 go to go on;
           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; go to 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:

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] ≤ E[0] ≤ 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; go to 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 go to 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 go to 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 go to 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 go to WB; if p ≥ q then go to WD;

WC:     i:= q:= q - 1; if q = n1 then go to 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 go to WC; if p < q then go to 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 go to 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 ≤ 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] ≤ 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);

            go to if (xa - i) × 2 < b - xa then reject else accept;
```

```
comment            AP 236   continued;

go on:      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 go to go on;
            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;
```

<u>comment</u>

APAP transforms the n-th order matrix given in <u>array</u> 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 <u>array</u> B[1 : n] the codiagonal elements B[k]:= H[k + 1, k] are delivered, 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 <u>real</u> <u>procedure</u> SUM (= AP 119) and the <u>real</u> <u>procedure</u> 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; go to 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 eigen-
value.  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 eigen-
values 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 eigen-
values 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
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 $i \leq j$) 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 normalized), 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) X 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;
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 go to 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 ASTRASF;
```

```
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 abso-
lute 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:
                              Sigma (A[i, j] × V[j]) = lambda × 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

Eigenvalues and Eigenvectors of a real symmetric matrix by the QR method [F2]
by P. A. Businger, Comm. ACM 8 (April 1965), 218,
modified by John H. Welsch, Comm. ACM 10 (June 1967), 376;

procedure symmetric QR 2(n,g,x); value n; integer n; array g,x;
     comment uses Householder's method and the QR algorithm to find all n eigenvalues
     and eigenvectors of the real symmetric matrix whose lower triangular part is given
     in the array g. The computed eigenvalues are stored as the diagonal elements g[i,i] and
     the eigenvectors as the corresponding columns of the array x. The original contents
     of the lower triangular part of g are lost during the computation whereas the strictly
     upper triangular part of g is left untouched.
     References:
     FRANCIS, J.G.F., The QR transformation - Part 2. Comput. J. 4 (1961), 332-345.
     PARLETT, B., The development and use of methods of LR type. New York U., 1963.
     WILKINSON, J.H., Householder's method for symmetric matrices. Numer. Math. 4 (1962), 354-361;
begin        real procedure sum(i,m,n,a); value m,n; integer i,m,n; real a;
     begin   real s; s := 0; for i := m step 1 until n do s := s + a; sum := s end sum;
     real procedure max(a,b); value a,b; real a,b; max := if a > b then a else b;
```

```
procedure Householder tridiagonalization 2(n,g,a,b,x,norm); value n; integer n; array g,a,b,x; real norm;
comment nonlocal real procedures sum, max;
comment reduces the given real symmetric n by n matrix g to tridiagonal form using n-2 elementary
orthogonal transformations (I-2ww') = (I-gamma uu'). Only the lower triangular part of g need be
given. The computed diagonal and subdiagonal elements of the reduced matrix are stored in a[1:n]
and b[1:n-1] respectively. The transformations on the right are also applied to the n by n matrix x.
The columns of the strictly lower triangular part of g are replaced by the nonzero portion of
the vectors u. norm is set equal to the infinity norm of the reduced matrix;
begin   integer i,j,k; real t,sigma,alpha,beta,gamma,absb; array p[2:n];
        norm := absb := 0;
        for k := 1 step 1 until n-2 do
        begin   a[k] := g[k,k]; sigma := sum(i,k+1,n,g[i,k]↑2); t := absb + abs(a[k]);
                absb := sqrt(sigma); norm := max(norm,t+absb); alpha := g[k+1,k];
                b[k] := beta := if alpha < 0 then absb else - absb;
                if sigma ≠ 0 then
                begin   gamma := 1/(sigma - alpha×beta); g[k+1,k] := alpha - beta;
                        for i := k+1 step 1 until n do
                        p[i] := gamma × (sum(j,k+1,i,g[i,j]×g[j,k]) +
                                sum(j,i+1,n,g[j,i]×g[j,k]));
                        t := .5 × gamma × sum(i,k+1,n,g[i,k] × p[i]);
                        for i := k+1 step 1 until n do p[i] := p[i] - t × g[i,k];
                        for i := k+1 step 1 until n do for j := k+1 step 1 until i do
                           g[i,j] := g[i,j] - g[i,k] × p[j] - p[i] × g[j,k];
                        for i := 2 step 1 until n do
                        p[i] := gamma × sum(j,k+1,n, x[i,j] × g[j,k]);
                        for i := 2 step 1 until n do for j := k+1 step 1 until n do
                           x[i,j] := x[i,j] - p[i] × g[j,k]
                end
        end k;
        a[n-1] := g[n-1,n-1]; a[n] := g[n,n]; b[n-1] := g[n,n-1]; t := abs(b[n-1]);
        norm := max(norm, absb+abs(a[n-1])+t); norm := max(norm, t+abs(a[n]) )
end Householder tridiagonalization 2;
```

```
     integer i,j,k,m,m1; real t,norm,eps,sine,cosine,lambda,mu,a0,a1,b0,beta,x0,x1;
     array a[1:n],b[0:n],c[0:n-1],cs,sn[1:n-1];
     for i := 1 step 1 until n do
     begin   x[i,i] := 1; for j := i+1 step 1 until n do x[i,j] := x[j,i] := 0  end x := identity matrix;
     Householder tridiagonalization 2(n,g,a,b,x,norm); eps := norm × 1.5₁₀-11;
     comment the tolerance used in the QR iteration is set equal to the product of the infinity norm of the
     reduced matrix and the relative machine precision (here assumed  to be 1.5₁₀-11 which is
     appropriate for a machine with a 36-bit mantissa);
     b[0] := mu := 0; m := n;
inspect: if m = 0 then go to return else i := k := m1 := m - 1;
     if abs(b[k]) ≤ eps then begin g[m,m] := a[m]; m := k; go to inspect end;
     for i := i-1 while abs(b[i]) > eps do k := i;
     comment find eigenvalues of lower 2 × 2;
     b0 := b[m1] ↑ 2; a1 := sqrt((a[m1] - a[m])↑2 + 4 × b0);  t := a[m1] × a[m] - b0;
     a0 := a[m1] + a[m]; lambda := .5 × (if a0 ≥ 0 then a0 + a1 else a0 - a1);  t := t/lambda;
     comment compute shift;
     if abs(t-mu) < .5 × abs(t) then mu := lambda := t
     else if abs(lambda-mu) < .5 × abs(lambda) then mu := lambda
     else begin mu := t; lambda := 0 end;
     a[k] := a[k] - lambda; beta := b[k];
     for j := k step 1 until m1 do
     begin   a0 := a[j]; a1 := a[j+1] - lambda; b0 := b[j]; t :=  sqrt(a0↑2 + beta↑2);
             cosine := cs[j] := a0/t; sine := sn[j] := beta/t;  a[j] := cosine × a0 + sine × beta;
             a[j+1] := - sine × b0 + cosine × a1; b[j] := cosine × b0 +  sine × a1; beta := b[j+1];
             b[j+1] := cosine × beta; c[j] := sine × beta
     end transformation on the left;
     b[k-1] := c[k-1] := 0;
     for j := k step 1 until m1 do
     begin   sine := sn[j]; cosine := cs[j]; a0 := a[j]; b0 := b[j];
             b[j-1] := b[j-1] × cosine + c[j-1] × sine;  a[j] := a0 × cosine + b0 × sine + lambda;
             b[j] := - a0 × sine + b0 × cosine; a[j+1] := a[j+1] × cosine;
             for i := 1 step 1 until n do
             begin   x0 := x[i,j]; x1 := x[i,j+1];
                     x[i,j] := x0 × cosine + x1 × sine;  x[i,j+1] := - x0 × sine + x1 × cosine
             end i
     end transformation on the right;
     a[m] := a[m] + lambda; go to inspect;
return:
end symmetric QR 2;
```