

RA

**stichting
mathematisch
centrum**



REKENAFDELING

MR 82/70

NOVEMBER

RA

T.J. DEKKER
NEWTON-LAGUERRE ITERATION

2nd (unrevised) edition

2e boerhaavestraat 49 amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
 AMSTERDAM

Contents

	page
1. Introduction	1
2. Laguerre's formula	2
3. Newton's formula	3
4. Formula for the multiplicity	4
5. Choice of p and q in Laguerre's formula	5
6. Upper bounds for the error of zeros of polynomials	6
7. Numerical experiments	7
References	10
ALGOL 60 program for calculating complex eigenvalues of real matrices	11
Data nr. 1	16
Output nr. 1	17
Data nr. 2	18
Output nr. 2	19
Data nr. 3	20
Output nr. 3	21

1. Introduction

Laguerre's formula is cubically convergent, provided one uses the right guess of the multiplicity of the zero. B. Parlett [2] assumes in his procedure for calculating the eigenvalues of a Hessenberg matrix that the multiplicity of the eigenvalue is either one or two, the latter in the case that convergence is slow.

The purpose of this paper is to show how the multiplicity may be estimated by means of a Newton formula. If this multiplicity, or rather the integer nearest to it, is used in Laguerre's formula, we get cubic convergence for multiple zeros also. The author has done some experiments with this formula and has the impression that the method is favourable if many zeros are multiple or in clusters. In sections 2, 3 and 4, some formulas are derived and some theorems about the order of convergence are proved. It is assumed that the given function f is sufficiently many times differentiable in a neighbourhood of a zero r of f , so that it makes sense to speak about the multiplicity of that zero and about Taylor expansion. As to convergence, only the local convergence, i.e. convergence in a sufficiently small neighbourhood of r is considered.

Section 5 discusses the choice of the two parameters, p and q , in Laguerre's formula and section 6 gives some practical upper bounds for the error in the case that the function is a polynomial.

Section 7 describes some features and results of an ALGOL 60 program for calculating the eigenvalues of a real matrix.

Acknowledgements

The author is indebted to the staff members of the Computation department of the Mathematical Centre for their valuable suggestions and to R.J. De Vogelaere for his suggestions on iteration and iteration control.

2. Laguerre's formula

To derive this formula, we start from the interpolating function

$$2.1 \quad f^*(z) = c(z-a)^p(z-b)^q$$

and choose, for given p and q , the parameters a , b and c such that the values of f^* and its first and second derivative are equal to those of the given function f . Let

$$2.2 \quad s_1 = f'(z)/f(z), \quad s_2 = s_1^2 - f''(z)/f(z).$$

We then find

$$2.3 \quad s_1 = \frac{p}{z-a} + \frac{q}{z-b}, \quad s_2 = -\frac{ds_1}{dz} = \frac{p}{(z-a)^2} + \frac{q}{(z-b)^2}.$$

Putting $s_0 = p + q$ and eliminating b , we obtain Laguerre's formula for the next iterate a , which we call $L(z)$:

$$2.4 \quad a = L(z) = z - s_0 / (s_1 \pm \sqrt{(q/p)(s_0 s_2 - s_1^2)}).$$

Herein, we choose the sign of the square root such that the absolute value of $L(z) - z$ is minimal.

We then have for the Laguerre iteration L , i.e. the iteration which in each step replaces z by $L(z)$, the following

2.5 Theorem. The Laguerre iteration L converges cubically in a neighbourhood of a zero r of f having multiplicity m , if p and q satisfy

$$2.5.1 \quad p = m + O(z-r)^2, \quad q > \text{constant} > 0.$$

Proof. For convenience, we shift the root r to the origin. Then $s_1 = \frac{m}{z} + \sigma$, where σ tends to a finite value and $s_2 = (m + O(z^2))/z^2$. The condition on p now reads $p = m + O(z^2)$ and we have

$$\begin{aligned}
z\sqrt{(q/p)(s_0s_2 - s_1^2)} &= \sqrt{(q/m)(s_0m - m^2 - 2m\sigma z) + O(z^2)} \\
&= \sqrt{q(q - 2\sigma z) + O(z^2)} = q - \sigma z + O(z^2).
\end{aligned}$$

So we find for the Laguerre iterate (we have to take the + sign, since m and q are both positive):

$$\begin{aligned}
L(z) &= z - s_0z/(m + \sigma z + (q - \sigma z) + O(z^2)) \\
&= z - s_0z/(s_0 + O(z^2)) = O(z^3).
\end{aligned}$$

If the sign of the square root is chosen the other way, the same theorem holds but for the condition on q which has to be replaced by " $q < \text{constant} < 0$ ".

If, however, the condition on p is replaced by $p = m + O(z - r)$ the convergence is only quadratic.

3. Newton's formula

Let now

$$3.1 \quad f^*(z) = c(z - a)^p,$$

in which a and c are chosen such that f^* and f have equal function value and derivative at z . Then we have

$$3.2 \quad s_1 = f'(z)/f(z) = p/(z - a)$$

and we obtain Newton's formula for the next iterate a , which we call $N_p(z)$:

$$3.3 \quad a = N_p(z) = z - p/s_1.$$

3.4 Theorem. The iteration N_p converges quadratically in a neighbourhood of an m -fold zero r of f if $p = m + O(z - r)$. Moreover, if the Taylor series of f is

$$3.4.1 \quad f(z) = a_m (z - r)^m + a_{m+1} (z - r)^{m+1} + O(z - r)^{m+2}$$

and p satisfies

$$3.4.2 \quad p = m + \frac{a_{m+1}}{a_m} (z - r) + O(z - r)^2,$$

then the convergence is cubic.

Proof. We again shift r to the origin. Then we have

$$1/s_1 = f(z)/f'(z) = \frac{z}{m} \left(1 - \frac{a_{m+1}}{a_m} \frac{z}{m} + O(z^2) \right).$$

Let $p = m + \mu z + O(z^2)$, then

$$\begin{aligned} N_p(z) &= z - z(1 + \mu \frac{z}{m}) \left(1 - \frac{a_{m+1}}{a_m} \frac{z}{m} \right) + O(z^3) \\ &= \left(\frac{a_{m+1}}{a_m} - \mu \right) \frac{z^2}{m} + O(z^3). \end{aligned}$$

So the convergence is cubic, if $\mu = \frac{a_{m+1}}{a_m}$, and otherwise quadratic.

4. Formula for the multiplicity

We again start from the interpolating function (3.1), but we now choose the parameters a , c and p such that the values of f^* and its first and second derivative are equal to those of f . Then

$$4.1 \quad s_1 = p/(z - a), \quad s_2 = p/(z - a)^2.$$

Hence, we have for p and the next iterate $a = M(z)$ (say):

$$4.2 \quad p = s_1^2/s_2,$$

$$4.3 \quad a = M(z) = z - p/s_1 = z - s_1/s_2.$$

(This is, in fact, the ordinary Newton formula $N_1(z)$ for the function $f(z)/f'(z)$.)

4.4 Theorem. The iteration M converges quadratically in a neighbourhood of a zero r of f and the values p converge linearly to the multiplicity of r .

Proof. Let f have the Taylor expansion (3.4.1). Then it easily follows that

$$4.4.1 \quad s_1^2/s_2 = m + 2 \frac{a_{m+1}}{a_m} (z - r) + O(z - r)^2.$$

Thus, p converges linearly to m and, according to theorem (3.4), the convergence of the iteration M is quadratic.

5. Choice of p and q in Laguerre's formula

We may choose p according to (4.2) and use this value in Laguerre's formula (2.4). Since condition (2.5.1) is not satisfied (cf. 4.4.1), we obtain not cubic, but only quadratic, convergence. In fact, the formula thus obtained is equivalent to Newton's formula (4.3).

In order to obtain a cubically convergent process, we must therefore use a better estimate for the multiplicity. As the multiplicity is an integer, we may simply choose p as the integer which is nearest to s_1^2/s_2 . If we are sufficiently near the limit, this yields the correct value and we get cubic convergence.

If, on the other hand, we are not near the limit, it may very well happen that this value for p is useless, either because it is negative or 0 (which happens often if f has non-real zeros) or because, in the case that f is a polynomial, it exceeds the degree.

In these cases, the most obvious choices for p seem to be $p = 1$ or $p = \text{degree}$, respectively.

In this way, we may obtain a negative argument for the square root and thus a real start leads in a natural way to non-real iterates.

Of course, then s_1^2/s_2 will become non-real also, in which case we simply disregard its imaginary part for the determination of p .

As to the choice of q , reasonable values seem p and ∞ or, in case of polynomials, degree- p . In the latter case we should avoid the situation $p = \text{degree}$, $q = 0$, since this formula would not converge cubically; so it is better to take $p \leq \text{degree} - 1$ and thus $q \geq 1$.

Summarizing, we obtain the following choices for p and q :

5.1 if f is not a polynomial:

$p = \text{the positive integer nearest to } s_1^2/s_2,$

$q = \infty \quad \text{or} \quad q = p;$

5.2 if f is a polynomial of degree $n \geq 2$:

$p = \text{the positive integer smaller than } n \text{ nearest to } s_1^2/s_2,$

$q = n - p \quad \text{or} \quad q = \min(n-p, p).$

6. Upper bounds for the error of zeros of polynomials

If the given function f is a polynomial of degree n , we have the following upper bounds for the error $z-r$, where r is the nearest zero of f .

a) expressed in the Newton step $\Delta_1 z = N_1(z) - z$ (cf. 3.3):

$$6.1 \quad |z - r| \leq n/|s_1| = n|\Delta_1 z|;$$

b) expressed in the Laguerre step $\Delta z = L(z) - z$ (cf. 2.4), where p satisfies $1 \leq p < n$:

$$6.2 \quad q = n-p \rightarrow |z - r| \leq (1 + \sqrt{2(n-1)})|\Delta z|,$$

$$6.3 \quad q = \infty \rightarrow |z - r| \leq \sqrt{n} |\Delta z|,$$

$$6.4 \quad q = p \rightarrow |z - r| \leq n |\Delta z|.$$

We prove only (6.2).

$$\begin{aligned}
 |z - r| &= \frac{|\Delta z|}{n} |s_1 + \sqrt{\frac{q}{p} (ns_2^2 - s_1^2)}| |z - r| \\
 &\leq \frac{|\Delta z|}{n} (n + \sqrt{(n-1)(n^2 + n^2)}) = (1 + \sqrt{2(n-1)}) |\Delta z|.
 \end{aligned}$$

These upper bounds are certainly not all best possible.

In practice, however, they are good enough, especially (6.2 & 3). So the criterion " $|\Delta z|$ smaller than a desired tolerance" seems to be a good acceptance test for zeros of polynomials. If z is near the limit, we have $|z - r| \approx |\Delta z|$ and otherwise the error is at worst only a modest factor times $|\Delta z|$.

It should be borne in mind, however, that no rounding errors are considered here. Cancellation of figures may cause a much smaller $|\Delta z|$ and thus a far too optimistic error estimate. This may especially happen near already accepted, and removed, zeros. Therefore, it is important to avoid circles around the accepted zeros during the iteration. This difficulty around the already accepted zeros is, in fact, the most serious drawback of any non-deflating method.

7. Numerical experiments

The author did some experiments with an ALGOL 60 program for calculating the eigenvalues of a real matrix. The main features of the program are

- a) The matrix is first transformed to Hessenberg form by means of Householder's transformation.
- b) For calculating f , f' and f'' Hyman's method is used.
- c) The iteration formula used is Laguerre's formula (2.4), where p and q are chosen according to (5.2).
- d) The iteration is continued until either $|\Delta z| < \text{norm} \times \text{eps}$, where eps is a given parameter and norm is the infinity norm of the matrix, or the number of iterations exceeds a given number.

Here, for the number of iterations, the program allows a higher maximum in case of convergence, i.e. in case $|\Delta z|$ is decreasing, then otherwise. (This strategy is inspired by J.W. Garwick's procedure "converge" [3] and an unpublished procedure for iteration control by R.J. De Vogelaere.)

- e) Iterates outside the circle around the origin with radius the infinity norm of the matrix are rejected and replaced by a suitable number on the edge of the circle. (This often saves an iteration from divergence.)
- f) Accepted zeros r_i are removed by subtracting $\sum (z - r_i)^{-k}$ from s_k ($k = 1, 2$). Moreover circles around these zeros with radius norm times given parameter, ϵ , are avoided during the iteration, as long as $|\Delta z|$ is greater than 4 times this radius.

- g) After accepting a zero (in fact either a real zero or a pair of complex conjugates), the program calculates a start from the next iteration by means of a Newton step (cf. Parlett [2] p.473). The first start is $L(\infty)$ with $p = 1$ or, in an earlier version, $L(0)$.

- h) After accepting a non-real zero, the program accepts its complex conjugate without any checking.

The problem here is how to define non-reality. On the one hand, one has to prevent an accepted conjugate pair from causing a too-high multiplicity in a cluster of zeros, and on the other hand, one wants to deliver complex conjugates pair-wise. In this program the non-reality criterion is " $\text{Im}(z) > \text{norm} \times \epsilon$ ".

- i) In an earlier version not only the multiplicity p (cf. 5.2) was used in Laguerre's formula, but also the limit of z was accepted as a p -tuple zero, where p is the last value used. This, however, yields difficulties, because, especially in the last step, cancellation of figures may cause a useless value of s_1^2/s_2 and thus yield a wrong multiplicity. If this multiplicity turns out too high, it ruins the whole subsequent calculation. A more fundamental objection is the following. Because of the cubic convergence one may use a rather modest tolerance and expect a much higher precision for the last

iterate. Thus one may accept the zero as a cluster of multiplicity p in the prescribed tolerance, but not in the higher precision expected. The newer version accepts each zero as a single one. In case of a multiple zero r the Newton step $\Delta_1 z$ (cf. (g)) usually vanishes and thus the next iterate starts outside a circle around r with radius $\text{norm} \times \text{eta}$ (cf. (f)). In case r is multiple, the next iteration will again enter the circle and converge to r (this may be considered a numerical definition of multiple zeros).

The program was run on the Electrologica computers X1 and X8 by means of the ALGOL systems of the Mathematical Centre, Amsterdam, the X1-system written by Dijkstra and Zonneveld and the X8-system by Kruseman Aretz.

The test matrices used were, among others, Rosser's matrix of order 8, Frank's matrix of order 12, Eberlein's matrix of order 16 (see Parlett [2]), and 5 tenth order matrices in Frobenius canonical form. The results were correct and the number of iterations was about the same as with Parlett's procedure Eig 3. A first impression is that the program is favourable (as to the number of iterations required) for matrices with clusters of eigenvalues. For matrices with well separated eigenvalues, Muller's method, with its high efficiency rate of 1.84, will presumably do better than Laguerre, which has an efficiency rate of $\sqrt[3]{3} \approx 1.44$. (Efficiency rate is the order of convergence for steps consisting of one function evaluation, more precisely: if the order of the process is m and the number of function evaluations per step is k , then the efficiency rate is defined as $\sqrt[k]{m}$.)

Mathematical Centre,
2e Boerhaavestraat 49,
Amsterdam.

References

1. E.N. Laguerre, Oeuvres de Laguerre, Gauthier-Villars, Paris,
vol I, p. 87-103.
2. B. Parlett, Laguerre's Method applied to the Matrix Eigenvalue
Problem, MTAC 18 (1964), 464-485.
3. J.V. Garwick, Algorithm 1, BIT 1 (1961), p. 64.

begin
comment R 1089, TJD 080866. Complex eigenvalues of real matrices.
 The method used is Newton - Laguerre iteration.
 Input: bigeps, smalleps, maxdiv, maxconv, then the matrices each preceded by its order, and finally the end marker 0.
 Output: bigeps, smalleps, maxdiv, maxconv, then for each matrix:
 if details then zreal, zimag, deltaz and parameter p of each iterate,
 moreover the order and the infinity norm of the matrix and the spur of
 the given matrix and the transformed one, subsequently real and imaginary
 part of each eigenvalue together with the error estimate and the
 number of iterations, and finally the sum of real and imaginary part
 of the eigenvalues and the total number of iterations ;

integer n, i, j; real spur; Boolean qeqm, transpose, details;
array es[1:2]; integer array msl[1:2];

comment NLCR causes new line carriage return,
 PRINTTEXT(s) prints the text between the outer quotes of the string s,
 ABSFIXT(n,m,x) prints x as an unsigned fixed point number having at
 most n digits before and m digits behind the decimal point,
 print(x) prints x as a decimal floating number having a 13 - digit
 fraction and an exponent of at most 3 digits ;

real procedure SUM(i,a,b,x); value b; integer i,a,b; real x;
begin real s; s:= 0;
 for i:= a step 1 until b do s:= s + x; SUM:= s
end SUM;

real procedure INPROD(i,a,b,x,y); value b; integer i,a,b; real x,y;
begin real s; s:= 0;
 for i:= a step 1 until b do s:= s + x × y; INPROD:= s
end INPROD;

procedure comeigval(a, n, e, m, re, im, er, c, out, aux);
value n; integer n; array a, e, re, im, er, aux; integer array m, c;
procedure out;
begin integer k; real norm; array b[1:n];
 tfrmreahes(a, n, e[2] \wedge 3, norm, b); aux[17]:= norm;
 for k:= 1 step 1 until n do
 begin comvalhes(a, n, b, e, m, k, re, im, out, aux);
 er[k]:= aux[13]; c[k]:= aux[14]
 end
end comeigval;

```

procedure comvalhes(a, n, b, e, m, k, re, im, out, d);
value n, k; integer n, k; array a, b, e, re, im, d; integer array m;
procedure out;
begin integer j; real norm, spur, w; array u, v[1:n];
  procedure deltaz;
    begin if d[13] > e[1] then
      cregion(d[6], d[7], k, re, im, norm, e[1]×.25);
      chyman(a, n, b, d[6], d[7], d[7] ≠ 0, 2, u, v, d);
      laguerre(n-k+1, k, re, im, d);
      d[13] := sqrt(d[8] 2 + d[9] 2)/norm; out
    end deltaz;
    d[15] := k; norm := d[17];
    if k = 1 then
      begin d[16] := spur := SUM(j, 1, n, a[j,j]);
        w := (n-1) × (n × (SUM(j, 1, n-1, a[j,j] 2
          + 2 × a[j,j+1] × b[j]) + a[n,n] 2) - spur 2);
        if w > 0 then
          begin d[6] := (spur
            + (if spur ≥ 0 then 1 else -1) × sqrt(w))/n;
            d[7] := 0
          end else
            begin d[6] := spur/n; d[7] := sqrt(-w)/n end ;
            go to newroot
        end ;
        if abs(im[k-1]) > e[1] × norm ∧ d[14] > .5 then
          begin re[k] := re[k-1]; im[k] := -im[k-1]; d[14] := 0;
            d[3] := -d[3]; d[5] := -d[5]; d[7] := -d[7]
          end else
            begin d[0] := d[2]×2; d[1] := d[3]×2; d[2] := d[4]; d[3] := d[5];
              laguerre(1, k-1, re, im, d);
            newroot: d[13] := e[1] × 2; d[14] := 0; out;
              iter(deltaz, e[2], m, 13, d); re[k] := d[6]; im[k] := d[7]
            end
          end comvalhes;
end

procedure iter(step, eps, max, n, X); value n, eps;
integer n; real eps; integer array max; array X; procedure step;
begin integer j, count; Boolean conv; array OLDX[0:n];
  count := 0;
next: count := count+1; X[n+1] := count; step;
  conv := if count = 1 then true else X[n] < .99 × OLDX[n];
  if conv then for j := 0 step 1 until n do OLDX[j] := X[j];
  if X[n] > eps ∧ count < max [if conv then 2 else 1] then
    go to next;
  if OLDX[n] < X[n] then
    for j := 0 step 1 until n do X[j] := OLDX[j]
  end iter;

```

```

procedure laguerre(degree,k,RE,IM,F);
value degree,k; integer degree,k; array RE,IM,F;
begin integer j,m; real a,b,d,s1,t1,s2,t2,dx,dy,p,x,y,w;
  x:= F[6]; y:= F[7]; s1:= F[2]; t1:= F[3];
  s2:= s1  $\wedge$  2 - t1  $\wedge$  2 - (F[0]  $\times$  F[4] - F[1]  $\times$  F[5]);
  t2:= 2  $\times$  s1  $\times$  t1 - (F[0]  $\times$  F[5] + F[1]  $\times$  F[4]);
  for j:= 1 step 1 until k-1 do
    begin a:= x-RE[j]; b:= y-IM[j]; d:= a $\wedge$ 2+b $\wedge$ 2;
      w:= (a  $\times$  F[0] + b  $\times$  F[1])/d;
      b:= (a  $\times$  F[1] - b  $\times$  F[0])/d; a:= w;
      s1:= s1-a; t1:= t1-b; s2:= s2-(a $\wedge$ 2-b $\wedge$ 2); t2:= t2-2 $\times$ a $\times$ b
    end ;
    a:= s1  $\wedge$  2 - t1  $\wedge$  2; b:= 2  $\times$  s1  $\times$  t1;
    p:= (a  $\times$  s2 + b  $\times$  t2)/(s2  $\wedge$  2 + t2  $\wedge$  2);
    m:= if p > degree-1 then (if degree > 1 then degree-1 else 1)
      else if p > 1 then p else 1;
    comment temporary amendment;
    if qeqm  $\wedge$  2 $\times$ m < degree then degree:= 2 $\times$ m;
    w:= (degree-m)/m;
    comsqrt(w  $\times$  (degree  $\times$  s2 - a), w  $\times$  (degree  $\times$  t2 - b), a, b);
    if s1  $\times$  a + t1  $\times$  b < 0 then begin a:= -a; b:= -b end ;
    a:= s1+a; b:= t1+b; d:= a  $\wedge$  2 + b  $\wedge$  2;
    dx:= - degree  $\times$  (a $\times$ F[0] + b $\times$ F[1])/d;
    dy:= - degree  $\times$  (a $\times$ F[1] - b $\times$ F[0])/d;
    F[6]:= x+dx; F[7]:= y+dy; F[8]:= dx; F[9]:= dy;
    F[10]:= p; F[11]:= m; F[12]:= degree
  end laguerre;

```

```

procedure cregion(x, y, k, RE, IM, norm, eps);
value k, norm, eps; integer k; real x, y, norm, eps; array RE, IM;
begin integer j; real ratio, neps;
  ratio:=sqrt((x/norm)  $\wedge$  2 + (y/norm)  $\wedge$  2);
  if ratio > 1+4 $\times$ eps then begin x:= x/ratio; y:= y/ratio end ;
  neps:= eps  $\times$  norm;
  again : for j:= 1 step 1 until k-1 do
    begin if ((x-RE[j])  $\wedge$  2 + (y-IM[j])  $\wedge$  2 < neps  $\wedge$  2 then
      begin x:= x + 2  $\times$  neps; go to again end
    end
  end cregion;

```

```

procedure chyman(A,n,B,x,y,complex,m,u,v,F); value n,x,y,complex,m;
Boolean complex; integer n,m; real x,y; array A,B,u,v,F;
begin integer i,k; real f,g;
  procedure element(uv,fg); real fg; array uv;
  begin real a; integer j; a:= if k = 0 then 0 else k × uv[i];
    uv[i]:= if i ≠ n then fg/B[i] else fg;
    fg:= x × uv[i] + a - INPROD(j,i,n,A[i,j],uv[j])
  end element;
  for k:= 0 step 1 until m do
  begin f:= if k = 0 then 1 else 0; g:= 0;
    for i:= n step -1 until 1 do
    begin element(u,f); if complex then
      begin element(v,g); f:= f-y×v[i]; g:= g+y×u[i] end
    end ;
    F[2×k]:= f; F[2×k+1]:= g
  end
end chyman;

procedure comsqrt(a, b, rp, ip); value a, b; real a, b, rp, ip;
begin rp:= sqrt((abs(a) + sqrt(a × a + b × b)) / 2);
  ip:= b:= if rp ≠ 0 then b / rp / 2 else 0;
  if a < 0 then
  begin ip:= if b > 0 then rp else -rp; rp:= abs(b) end
end comsqrt;

procedure tfmreahes(A, n, eps, norm, B);
value n, eps; integer n; real eps, norm; array A, B;
begin integer i, j, k, k1; real w, alfa, tol, tol2; 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; tol2:= tol ↑ 2;
  for k:= 1 step 1 until n-1 do
  begin k1:= k+1; w:= INPROD(i,k+2,n,A[i,k],A[i,k]);
    if w > tol2 then
    begin B[k]:= sqrt(w + A[k1,k] ↑ 2);
      if A[k1,k] > 0 then B[k]:= -B[k];
      A[k1,k]:= A[k1,k] - B[k]; w:= A[k1,k] × B[k];
      for i:= 1 step 1 until n do
      P[i]:= INPROD(j,k1,n,A[i,j],A[j,k])/w;
      alfa:= INPROD(i,k1,n,A[i,k],P[i]);
      for j:= k1 step 1 until n do
      B[j]:= (INPROD(i,k1,n,A[i,k],A[i,j]) + alfa × A[j,k])/w;
      for j:= k1 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:= k1 step 1 until n do
        A[i,j]:= A[i,k] × B[j] + P[i] × A[j,k] + A[i,j]
      end end else
      begin B[k]:= if abs(A[k1,k]) > tol then A[k1,k] else tol×0.5;
        A[k1,k]:= 0
      end end ;
    B[n]:= tol
  end
end tfmreahes;

```

```

start:
qeqm:= 1 true ; comment temporary amendment;
transpose:= 1 true ; details:= 1 true ;
PRINTTEXT(
  ‡      bigeps          smalleps          maxdiv    maxconv‡);
NLCR;
for i:= 1, 2 do begin es[i]:= read; print(es[i]) end ;
for i:= 1, 2 do begin ms[i]:= read; ABSFIXT(8,0, ms[i]) end ; NLCR;
for n:= read while n ≠ 0 do
  begin array matrix[1:n, 1:n], re, im, erval[1:n], aux[0:17];
    integer array cval[1:n];
    procedure outline;
    if details then
      begin integer count;
        count:= aux[14]; print(aux[6]); print(aux[7]);
        if count > 0 then
          begin print(aux[13]); print(aux[10]) end ;
          NLCR
        end outline;
        for i:= 1 step 1 until n do for j:= 1 step 1 until n do
          if transpose then matrix[j,i]:= read else matrix[i,j]:= read;
          if details then
            begin NLCR; PRINTTEXT(
  ‡      z real          z imag          error          p‡
              ); NLCR; NLCR
            end ;
            spur:= SUM(i,1,n, matrix[i,i]);
            comeigval(matrix,n,es,ms,re,im,erval,cval,outline,aux);
            NLCR; PRINTTEXT(
  ‡order      norm          spur          spur transformed‡);
            NLCR; ABSFIXT(4,0,n); print(aux[17]);
            print(spur); print(aux[16]); NLCR; NLCR;
            PRINTTEXT(
  ‡      real part      imaginary part      error          count‡
              ); NLCR; NLCR;
            for j:= 1 step 1 until n do
              begin print(re[j]); print(im[j]); print(erval[j]);
                ABSFIXT(4, 0, cval[j]); NLCR
              end ;
              NLCR; print(SUM(j, 1, n, re[j])); print(SUM(j, 1, n, im[j]));
              PRINTTEXT(‡←—      sums      →‡);
              ABSFIXT(4, 0, SUM(j, 1, n, cval[j])); NLCR
            end
          end
        end
      end
    end
  end

```

```
' bigeps = '      10-3      ' smalleps = '      10-6
' maxdiv = '      10      ' maxconv = '      30
```

```
' matrix of Eberlein of order '      16
```

```
-10+10+10+10+ 2- 2- 2- 2-20+20+20+20+ 4- 4- 4- 4-
-15+15+10+10+ 3- 3- 2- 2-30+30+20+20+ 6- 6- 4- 4-
-10 0+20+10+ 2 0- 4- 2-20 0+40+20+ 4 0- 8- 4-
- 5 0 0+25+ 1 0 0- 5-10 0 0+50+ 2 0 0-10
-10+10+10+10- 2+ 2+ 2+ 2-20+20+20+20- 4+ 4+ 4+ 4-
-15+15+10+10- 3+ 3+ 2+ 2-30+30+20+20- 6+ 6+ 4+ 4-
-10 0+20+10- 2 0+ 4+ 2-20 0+40+20- 4 0+ 8+ 4-
- 5 0 0+25- 1 0 0+ 5-10 0 0+50- 2 0 0+10
-40+40+40+40+ 8- 8- 8- 8-30+30+30+30+ 6- 6- 6- 6-
-60+60+40+40+12-12- 8- 8- 8-45+45+30+30+ 9- 9- 6- 6-
-40 0+80+40+ 8 0-16- 8-30 0+60+30+ 6 0-12- 6-
-20 0 0+102+ 4 0 0-20-15 0 0+75+ 3 0 0-15
-40+40+40+40- 8+ 8+ 8+ 8-30+30+30+30- 6+ 6+ 6+ 6-
-60+60+40+40-12+12+ 8+ 8-45+45+30+30- 9+ 9+ 6+ 6-
-40 0+80+40- 8 0+16+ 8-30 0+60+30- 6 0+12+ 6-
-20 0 0+102- 4 0 0+20-15 0 0+75- 3 0 0+15
```

```
' matrix of Eberlein of order '      5
```

```
+15+11+ 6- 9-15
+ 1+ 3+ 9- 3- 8
+ 7+ 6+ 6- 3-11
+ 7+ 7+ 5- 3-11
+17+12+ 5-10-16
```

```
' end marker '      0
```

bigeps		smalleps		maxdiv	maxconv
+.99999999999994 ₁₀ ⁺ 3		+.99999999999993 ₁₀ ⁺ 6		10	30
order	norm	spur	spur transformed		
16	+420	+240	+.2399999999998 ₁₀ ⁺ 3		

real part	imaginary part	error	count
- .3000000000746 ₁₀ ⁺	1 + .9999999979009 ₁₀ ⁺	0 + .6753525089633 ₁₀ ⁺	6 8
- .3000000000746 ₁₀ ⁺	1 - .9999999979009 ₁₀ ⁺	0 + .6753525089633 ₁₀ ⁺	6 0
- .5999999999105 ₁₀ ⁺	1 - .2000000004009 ₁₀ ⁺	1 + .5650517028877 ₁₀ ⁺	6 4
- .5999999999105 ₁₀ ⁺	1 + .2000000004009 ₁₀ ⁺	1 + .5650517028877 ₁₀ ⁺	6 0
- .9000000000509 ₁₀ ⁺	1 + .2999999998709 ₁₀ ⁺	1 + .2342138034103 ₁₀ ⁺	7 3
- .9000000000509 ₁₀ ⁺	1 - .2999999998709 ₁₀ ⁺	1 + .2342138034103 ₁₀ ⁺	7 0
- .1199999999985 ₁₀ ⁺	2 - .3999999999800 ₁₀ ⁺	1 + .1874531488239 ₁₀ ⁺	13 4
- .1199999999985 ₁₀ ⁺	2 + .3999999999800 ₁₀ ⁺	1 + .1874531488239 ₁₀ ⁺	13 0
+ .3000000000052 ₁₀ ⁺	2 - .1000000000180 ₁₀ ⁺	2 + .9667010237706 ₁₀ ⁺	10 5
+ .3000000000052 ₁₀ ⁺	2 + .1000000000180 ₁₀ ⁺	2 + .9667010237706 ₁₀ ⁺	10 0
+ .4499999999971 ₁₀ ⁺	2 - .1500000000026 ₁₀ ⁺	2 + .1080814464134 ₁₀ ⁺	12 8
+ .4499999999971 ₁₀ ⁺	2 + .1500000000026 ₁₀ ⁺	2 + .1080814464134 ₁₀ ⁺	12 0
+ .6000000000006 ₁₀ ⁺	2 - .1999999999983 ₁₀ ⁺	2 + .2055620514102 ₁₀ ⁺	7 3
+ .6000000000006 ₁₀ ⁺	2 + .1999999999983 ₁₀ ⁺	2 + .2055620514102 ₁₀ ⁺	7 0
+ .1500000000001 ₁₀ ⁺	2 + .4999999998327 ₁₀ ⁺	1 + .7078564979711 ₁₀ ⁺	10 2
+ .1500000000001 ₁₀ ⁺	2 - .4999999998327 ₁₀ ⁺	1 + .7078564979711 ₁₀ ⁺	10 0
+240	-0	<— sums —>	37

order	norm	spur	spur transformed	
5	+60	+5	+.5000000000015 ₁₀ ⁺ 1	
real part	imaginary part	error	count	
+ .1500019082881 ₁₀ ⁺	1 + .3570720966396 ₁₀ ⁺	1 + .2448380706389 ₁₀ ⁺	6 5	
+ .1500019082881 ₁₀ ⁺	1 - .3570720966396 ₁₀ ⁺	1 + .2448380706389 ₁₀ ⁺	6 0	
+ .1499981967787 ₁₀ ⁺	1 - .3570707634317 ₁₀ ⁺	1 + .1633034000587 ₁₀ ⁺	7 2	
+ .1499981967787 ₁₀ ⁺	1 + .3570707634317 ₁₀ ⁺	1 + .1633034000587 ₁₀ ⁺	7 0	
- .1000000000075 ₁₀ ⁺	1 - .4235164736271 ₁₀ ⁺	21 + .1074449963813 ₁₀ ⁺	11 3	
+ .5000002101267 ₁₀ ⁺	1 - .4235164736271 ₁₀ ⁺	21 <— sums —>	10	

```

' bigeps = '      10-3      ' smalleps = '      10-6
' maxdiv = '      10      ' maxconv = '      30

' matrix of order '      10

0,0,0,0,-120,0,0,0,0,0,
1,0,0,0,-274,0,0,0,0,0,
0,1,0,0,-225,0,0,0,0,0,
0,0,1,0,-85,0,0,0,0,0,
0,0,0,1,-15,0,0,0,0,0,
0,0,0,0,0,-15,-85,-225,-274,-120,
0,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,1,0

' matrix of order '      10

0,0,0,-24,0,0,0,0,0,0,
1,0,0,+50,0,0,0,0,0,0,
0,1,0,-35,0,0,0,0,0,0,
0,0,1,+10,0,0,0,0,0,0,
0,0,0,0,0,0,+6,0,0,0,
0,0,0,0,1,0,-11,0,0,0,
0,0,0,0,0,1,+6,0,0,0,
0,0,0,0,0,0,0,-2,0,0,
0,0,0,0,0,0,0,1,+3,0,
0,0,0,0,0,0,0,0,0,1

' end marker '      0

```

bigeps		smalleps		maxdiv	maxconv		
+.9999999999994 ₁₀		3	+.9999999999993 ₁₀	6	10	30	
order	norm	spur		spur transformed			
10	+719	-30		-30			
real part		imaginary part		error	count		
-.4000000001142 ₁₀		1	-0	+.4410824423796 ₁₀	6	3	
-.299999999276 ₁₀		1	-0	+.3028075981486 ₁₀	10	5	
-.3000000000586 ₁₀		1	-0	+.5682899702787 ₁₀	12	4	
-.2000000000156 ₁₀		1	-0	+.2635621555004 ₁₀	9	4	
-.1999999999194 ₁₀		1	-0	+.4646842353033 ₁₀	6	3	
-.1000000000009 ₁₀		1	-0	+.1621215430338 ₁₀	7	3	
-.999999999873 ₁₀		0	-0	+.1132137445867 ₁₀	7	3	
-.4000000001979 ₁₀		1	+.3853820784243 ₁₀	8	+.5473232283765 ₁₀	11	3
-.4999999809937 ₁₀		1	+.3196600908417 ₁₀	6	+.7111538700240 ₁₀	6	2
-.5000000063090 ₁₀		1	-.1067892507578 ₁₀	6	+.3452325040621 ₁₀	9	2
-.2999999987541 ₁₀		2	+.2167246608681 ₁₀	6	← sums →	32	
order	norm	spur		spur transformed			
10	+51	+20		+20			
real part		imaginary part		error	count		
+.2999999999822 ₁₀		1	-0	+.8851244955744 ₁₀	6	3	
+.2999999999975 ₁₀		1	-0	+.4885304315094 ₁₀	6	2	
+.1999999999889 ₁₀		1	-0	+.1576667007328 ₁₀	6	4	
+.2000000000040 ₁₀		1	-0	+.1453166015508 ₁₀	6	2	
+.20000000000367 ₁₀		1	-0	+.2902244399922 ₁₀	6	2	
+.9999999999918 ₁₀		0	-0	+.4801869767664 ₁₀	13	3	
+.10000000000002 ₁₀		1	-0	+0		2	
+.10000000000029 ₁₀		1	-0	+.2860100071415 ₁₀	12	2	
+.9999999999764 ₁₀		0	-0	+.3597118479324 ₁₀	11	2	
+.3999999999909 ₁₀		1	-0	+.7370651060735 ₁₀	8	2	
+20		-0	← sums →	24			

```
' bigeps = ' 10-4      ' smalleps = ' 10-6
' maxdiv = ' 10      ' maxconv = ' 30
```

```
' matrix with double eigenvalues -5(1)-1, the order is ' 10
```

```
0 0 0 0 -360 0 0 0 0 120
3 0 0 0 -822 -1 0 0 0 274
0 3 0 0 -675 0 -1 0 0 225
0 0 3 0 -255 0 0 -1 0 85
0 0 0 3 -45 0 0 0 -1 15
0 0 0 0 -480 0 0 0 0 120
4 0 0 0 -1096 -1 0 0 0 274
0 4 0 0 -900 0 -1 0 0 225
0 0 4 0 -340 0 0 -1 0 85
0 0 0 4 -60 0 0 0 -1 15
```

```
' matrix from a 5-th order matrix of Eberlein, the order is ' 10
```

```
30 22 12 -18 -30 -15 -11 -6 9 15
2 6 18 -6 -16 -1 -3 -9 3 8
14 12 12 -6 -22 -7 -6 -6 3 11
14 14 10 -6 -22 -7 -7 -5 3 11
34 24 10 -20 -32 -17 -12 -5 10 16
15 11 6 -9 -15 0 0 0 0 0
1 3 9 -3 -8 0 0 0 0 0
7 6 6 -3 -11 0 0 0 0 0
7 7 5 -3 -11 0 0 0 0 0
17 12 5 -10 -16 0 0 0 0 0
```

```
' end marker ' 0
```

bigeps		smalleps		maxdiv	maxconv
+.999999999999 ₁₀ ⁺		4	+.999999999999 ₁₀ ⁻	6	10
order		norm	spur	spur transformed	
10		+1375	-30	-.299999999999 ₁₀ ⁺	
real part		imaginary part		error	count
-.399989973391 ₁₀ ⁺		1	+0	+.1580279266116 ₁₀ ⁻	6
-.4000005259033 ₁₀ ⁺		1	+0	+.4172092563422 ₁₀ ⁻	6
-.3000027706683 ₁₀ ⁺		1	-.7296685768499 ₁₀ ⁻	4	+.5343925570865 ₁₀ ⁻
-.2999984608075 ₁₀ ⁺		1	+.5491460202955 ₁₀ ⁻	4	+.2939639197688 ₁₀ ⁻
-.2000014592817 ₁₀ ⁺		1	-.2302103213440 ₁₀ ⁻	6	+.8979248888795 ₁₀ ⁻
-.1999983257934 ₁₀ ⁺		1	+.1703961456778 ₁₀ ⁻	6	+.2222638813894 ₁₀ ⁻
-.9999999589027 ₁₀ ⁻		0	-.4762801672581 ₁₀ ⁻	5	+.3444804070472 ₁₀ ⁻
-.9999999957927 ₁₀ ⁻		0	+.7523237179584 ₁₀ ⁻	5	+.2007604636096 ₁₀ ⁻
-.5000000003070 ₁₀ ⁺		1	-.1419928190858 ₁₀ ⁻	3	+.7507035066522 ₁₀ ⁻
-.4999999995700 ₁₀ ⁺		1	+.1345993837996 ₁₀ ⁻	3	+.5403373434417 ₁₀ ⁻
-.2999991511193 ₁₀ ⁺		2	-.2274506961042 ₁₀ ⁻	4	← sums →
order		norm	spur	spur transformed	
10		+180	+10	+.1000000000007 ₁₀ ⁺	
real part		imaginary part		error	count
+.1500002361647 ₁₀ ⁺		1	+.3570709487936 ₁₀ ⁺	1	+.1828237746628 ₁₀ ⁻
+.1500002361647 ₁₀ ⁺		1	-.3570709487936 ₁₀ ⁺	1	+.1828237746628 ₁₀ ⁻
+.1500016387492 ₁₀ ⁺		1	-.3570737931019 ₁₀ ⁺	1	+.5899138871583 ₁₀ ⁻
+.1500016387492 ₁₀ ⁺		1	+.3570737931019 ₁₀ ⁺	1	+.5899138871583 ₁₀ ⁻
+.1499824760253 ₁₀ ⁺		1	+.3570814503051 ₁₀ ⁺	1	+.5708445962160 ₁₀ ⁻
+.1499824760253 ₁₀ ⁺		1	-.3570814503051 ₁₀ ⁺	1	+.5708445962160 ₁₀ ⁻
+.1500013385417 ₁₀ ⁺		1	-.3570508061923 ₁₀ ⁺	1	+.6120317852755 ₁₀ ⁻
+.1500013385417 ₁₀ ⁺		1	+.3570508061923 ₁₀ ⁺	1	+.6120317852755 ₁₀ ⁻
-.9999840087548 ₁₀ ⁻		0	+.6331372925938 ₁₀ ⁻	11	+.9083298989051 ₁₀ ⁻
-.1000014427047 ₁₀ ⁺		1	-.1582765984939 ₁₀ ⁻	12	+.8855455801551 ₁₀ ⁻
+.9999715353828 ₁₀ ⁺		1	+.6173096327445 ₁₀ ⁻	11	← sums →