Laguerre's Method Applied to the Matrix Eigenvalue Problem

By Beresford Parlett

1. Introduction. We present a new algorithm for the calculation of the eigenvalues of real square matrices of orders up to 100. The basic method is directly applicable to complex matrices as well and, in both cases, with each eigenvalue λ of A a vector v is produced for which $(A - \lambda I)v$ is null except for a small last element. This vector is not always an approximation to the eigenvector for λ and this algorithm claims only to find eigenvalues.

The main concern has been to use only single precision arithmetic although the effect of using an accumulated inner product procedure in one part of the program is shown in the results in Section 15.

The method consists of two parts. Firstly the given matrix A is reduced to almost triangular (Hessenberg) form H by elementary similarity transformations. Direct reduction of H to sparser forms requires extra precision in practice and even then is not without difficulties. So the second stage is the iterative search for the eigenvalues of H. A natural extension of Hyman's method [13] may be used to evaluate $p(z) = \det(H - zI)$ and any number of derivatives in an accurate and stable way. However each evaluation requires approximately n^2 real multiplications and n^2 real additions for an $n \times n$ matrix and complex z. Thus the viability of this approach depends on finding each eigenvalue with a small number of evaluations. Results so far with the method developed here indicate an average of just less than 9 evaluations (3 iterations) per eigenvalue on a wide variety of matrices of orders from 8 to 100.

Now the iterations of Muller, Newton, and Bairstow converge quickly once a fair approximation to an eigenvalue has been found. They do not seem so satisfactory at the beginning of a search. Laguerre's method [2], [5], [6] was designed for polynomials with real zeros and when these are distinct it gives strong convergence right from any starting value. The method can be extended to the complex plane. No longer is convergence certain for any starting value but, in practice, the complex iteration seems as powerful as the real one on all examples so far considered. One Laguerre step requires more calculation than one step of any of the methods mentioned above, but when there are no a priori approximation to the zeros available the reduction in the number of iterations with Laguerre more than compensates for the extra calculation for each step. In addition when an eigenvalue has been found there is enough information available to take one Newton step towards the next eigenvalue.

This paper is mainly a detailed discussion of the practical application of the method and techniques for keeping the number of iterations to a minimum. A description of the program is given in ALGOL 60 [1], together with some results ob-

Received January 21, 1963. The work presented in this paper is supported by the AEC Computing and Applied Mathematics Center, Courant Institute of Mathematical Sciences, New York University, under Contract AT(30-1)-1480 with the U. S. Atomic Energy Commission, and also under Contract Nonr-225(37)(NR-044-211) with the Office of Naval Research.

tained with a FORTRAN version on an IBM 7090 which has been submitted to SHARE as NU EIG3, Ref. 1373.

The work was begun at Stanford University under Professor G. E. Forsythe and continued at the Courant Institute of Mathematical Sciences, New York University, New York.

The author acknowledges the help of George E. Forsythe in debugging the Algol program, and in testing it on the Burroughs B 5000 at Stanford University. See the certification in this issue.

2. The Condition of an Eigenvalue. The accurate location of an eigenvalue of a non-hermitian matrix can be made difficult by the possible sensitivity of the eigenvalue to small perturbations of the matrix elements.

A definition and discussion of the condition of an eigenvalue is given in [10, p. 110]. It suffices for our purposes to say that an eigenvalue is ill-conditioned if a small change in any of the matrix elements causes a large change in the eigenvalue. The effect of ill-conditioning on the numerical computation of eigenvalues is explained in the following way. See [10, p. 109] and [13, p. 32] for more details.

When we evaluate the characteristic polynomial of a Hessenberg matrix with finite precision we obtain the exact value of the characteristic polynomial of a Hessenberg matrix whose elements never differ from those of the given matrix by more than small quantities. Every time we repeat the evaluation with a new argument we are effectively evaluating exactly the characteristic polynomial of a different, though close, matrix. Thus at each step of our procedure the current iterate behaves as though it were approaching an eigenvalue of a different, though close, matrix. For this reason ill-conditioned eigenvalues may be difficult to find.

We conclude that it is fruitless to expect a program to work well on all matrices. We should rather attempt to give with each eigenvalue some information about its condition. If the successive iterates are printed out they will give valuable information about the condition and accuracy of a computed eigenvalue without any extra calculation.

3. Reduction to Almost Triangular Form. For a detailed discussion we refer to Wilkinson [14, 15]. For completeness we include our Crout-like reduction in the Algol 60 program of this article.

The reduction could also be done with orthogonal transformations. One advantage would be that no eigenvalue would have its condition worsened (increased) in the process. On the other hand it would take about twice as long (using Householder's method) as the method we use which is quite accurate enough.

We record any superdiagonal elements $a_{i,i+1}$ which are zero or small enough to be put to zero. Hyman's method for determinant evaluation requires that $a_{i,i+1} \neq 0$. Yet Wilkinson [13] has shown that small values of $a_{i,i+1}$ do not impair accuracy. So two courses can be followed. Either zero pivots can be replaced by some small quantity or advantage can be taken of the zeros to consider a number of smaller matrices.

We have chosen the latter course for a program which computes eigenvalues only. The method we use is easily coded for any principal submatrix of a (lower) Hessenberg matrix and it clearly saves time to consider a number of smaller matrices rather than one large one.

4. Laguerre's Iteration. Let $P(z)$ be a complex polynomial in z with roots $r_1, \cdots, r_n$. Given an approximation z to one of the roots, say r_n, then Laguerre's method uses $P(z)$, $P'(z)$, and $P''(z)$ to obtain a better approximation. We define derivatives of $\log P(z)$ as follows

$$S_1(z) = \frac{P'(z)}{P(z)} = \sum_{i=1}^{n} \frac{1}{z - r_i},$$

$$S_2(z) = \frac{P'(z)^2 - P(z)P''(z)}{P(z)^2} = \sum_{i=1}^{n} \frac{1}{(z - r_i)^2}.$$

We write

$$\frac{1}{z - r_n} = \alpha(z), \qquad \frac{1}{z - r_i} = \beta(z) + \delta_i(z) \qquad (i = 1, 2, \cdots, n - 1),$$

where β is the mean of the $1/(z - r_i)$ $(i = 1, 2, \cdots, n - 1)$, and so

$$0 = \sum_{i=1}^{n-1} \delta_i .$$

Now we define $\delta^2 = \sum_{i=1}^{n-1} \delta_i^2$ and express S_1 and S_2 in terms of α, β, δ, obtaining

$$S_1 = \alpha + (n - 1)\beta, \qquad S_2 = \alpha^2 + (n - 1)\beta^2 + \delta^2.$$

Eliminating β and solving the resulting quadratic gives

$$\alpha = \frac{1}{n} \{S_1 \pm \sqrt{(n - 1)(nS_2 - n\delta^2 - S_1^2)}\}$$

and

$$\text{(A)} \quad r_n = z - \frac{n}{S_1 \pm \sqrt{(n - 1)(nS_2 - n\delta^2 - S_1^2)}} = z - \frac{n}{S_1 \pm W^*}, \quad \text{say.}$$

The next approximation z' is obtained by ignoring the unknown quantity δ^2 in equation (A) above. Thus

$$\text{(B)} \quad z' = z - \frac{n}{S_1 \pm \sqrt{(n - 1)(nS_2 - S_1^2)}} = z - \frac{n}{S_1 \pm W}, \quad \text{say.}$$

We choose that square root of W^2 which maximises $| S_1 \pm W |$. Now $| S_1 \pm W |^2 = | S_1 |^2 + | W |^2 \pm 2 \operatorname{Re} (\bar{S}_1 W)$. So we choose W to make $\operatorname{Re} (\bar{S}_1 W)$ non-negative and when it is zero we arbitrarily take $0 \leqq \arg W < \pi$. We make a similar rule for W^*.

We call z' the *Laguerre iterate* of z for the polynomial P, i.e. $L_P(z) = z'$. The transition from z to z' is called one *Laguerre iteration.*

We observe that $z' = \infty$ if, and only if,

$$S_1(z) = S_2(z) = 0, \qquad \text{i.e.} \qquad P'(z) = P''(z) = 0.$$

Also $z' = r_n$ if and only if $\delta^2 = 0$.

For polynomials with real roots the following results are known, see [2], [5], and [6].

(1) The real line is divided into as many abutting intervals as there are distinct

roots and from any initial point in such an interval the successive Laguerre iterates converge monotonically to the root therein.

(2) Locally, if the root is simple, convergence is cubic. Otherwise it is linear.

(3) Laguerre iterations are invariant under all Möbius transformations

$$Tz = \frac{az+b}{cz+d}, \qquad a,\, b,\, c,\, d \text{ real}, \qquad \begin{vmatrix} a & b \\ c & d \end{vmatrix} \neq 0.$$

I.e. if z' is the iterate of z for a polynomial P (roots r_i), then Tz' is the iterate of Tz for the transformed polynomial (roots Tr_i).

5. The Complex Iteration.

Here we are interested in Laguerre iterations in the complex plane. We can say the following.

Property (1) does not extend to the complex case as it stands, i.e., the complex plane is not covered by abutting regions such that from an initial value in a region successive iterates converge to the zero contained therein. As an example consider $P(z) = z(3z^2 + a^2)$, $a > 0$. Then $L_P(a) = -a$ and $L_P(-a) = a$ yielding no convergence. The question of what can be said about regions of convergence will not be pursued here.

Property (2) does extend to complex Laguerre iterations, being algebraic in nature. See [2, p. 269].

Property (3), also being algebraic, extends to the complex plane. This invariance may be verified at the cost of considerable calculation. We prefer the following brief argument which we present because we have never seen an explicit proof of Property (3).

Let P be a polynomial with complex zeros α, β, $\cdots$ and let summations be over the n zeros of P. Take z as a fixed number, not a zero of P, and consider, after Laguerre, for $v \neq z$,

$$\begin{aligned} f(v, u) &\equiv \sum \left(\frac{u-\alpha}{z-\alpha}\right)^2 - \left(\frac{u-v}{z-v}\right)^2 \\ &= \sum \left(\frac{u-z}{z-\alpha} + 1\right)^2 - \left(\frac{u-z}{z-v} + 1\right)^2 \\ &= \left[S_2(z) - \frac{1}{(z-v)}\right](u-z)^2 + 2\left[S_1(z) - \frac{1}{(z-v)}\right](u-z) + n - 1. \end{aligned}$$

Thus $f = 0$ is a quadratic in $u - z$ (and also in u since z is fixed) with discriminant

$$D(v) = n(v-z)^{-2} + 2S_1(v-z)^{-1} + S_1^2 - (n-1)S_2\,.$$

Since $v \neq z$, we see that $D(v) = 0$ has the same solutions as the Laguerre equation

$$0 = \bar{D}(v) = n + 2S_1(v-z) + (S_1^2 - (n-1)S_2)(v-z)^2.$$

For invariance we must show that $D(v) = 0$ is an invariant equation under the transformation group given in the previous section. This can be shown by rewriting $D(v)$ so that it is a rational function of well-known covariants, including the Hessian of P. However we need only remark that $D(v)$ is the discriminant of an invariant quadratic and so must be a covariant of P.

The fact that $f = 0$ is an invariant equation is seen by writing it in homogeneous form. Rewriting each quantity w as the ratio $w:w'$ the transformation group takes the form

$$W = aw + b, \qquad W' = cw' + d$$

and f becomes

$$f\left(\frac{v}{v'}, \frac{u}{u'}\right) = \sum \left(\frac{u\alpha' - \alpha u'}{z\alpha' - \alpha z'}\right)^2 - \left(\frac{uv' - vu'}{zv' - vz'}\right)^2$$

which is a sum of absolutely invariant terms. On putting all terms $w' = 1$ the original f is recovered.

Practical Application of the Iteration

6. Cubic Convergence for Double Roots. If r_n is a k-fold root of P then the formula (see [2, p. 73])

$$z' = z - \frac{n}{S_1 \pm \sqrt{\left(\frac{n-k}{k}\right)(nS_2 - S_1^2)}}$$

will yield local cubic convergence to successive iterates. In general k is unknown so we do the following. Suppose that we can test to distinguish linear from cubic convergence. For simplicity we assume that linear increments are due to *two* close roots and thus reject the current increment in favour of one computed from the formula above with $k = 2$. This will need little extra calculation.

7. A Modification of the Formula for Polynomials with Real Coefficients. Let z be an approximation to a complex root r_n. We can make use of the fact that $\bar{r}_n \equiv r_{n-1}$ is also a root. Let

$$z = x + iy, \qquad \alpha_1 = \frac{1}{z - r_n}, \qquad \alpha_2 = \frac{1}{z - \bar{r}_n},$$

$$\beta + \delta_i = \frac{1}{z - r_i}, \qquad i = 1, 2, \cdots, (n-2).$$

Then

$$\hat{S}_1 = S_1 - \alpha_2 = \alpha_1 + (n-2)\beta,$$
$$\hat{S}_2 = S_2 - \alpha_2^2 = \alpha_1^2 + (n-2)\beta^2 + \delta^2.$$

But

$$\alpha_2 = \frac{1}{z - \bar{z} + \bar{z} - \bar{r}_n} = \frac{-i}{2y}\left\{1 + \frac{1}{2y}(\bar{z} - \bar{r}_n) - \frac{1}{4y^2}(\bar{z} - \bar{r}_n)^2 + \cdots\right\}.$$

Provided that $|z - r_n|$ is small compared with $|y|$ we may use

$$\alpha_2 \doteqdot \frac{-i}{2y} \quad \text{and} \quad \alpha_2^2 = \frac{-1}{4y^2}$$

to modify S_1 and S_2 and obtain z' from

$$z' = z - \frac{(n-1)}{\hat{S}_1 + \sqrt{(n-2)[(n-1)\hat{S}_2 - \hat{S}_1^2]}}.$$

As a test we require

$$| z_i - z_{i-1} | < | y_i |$$

in order to use the modification. It is also possible to postpone the application until 2 or 3 iterations have improved the approximation. However in practice we have found it preferable to use the modification after 1 iteration. Actually the modification is not used for iterates in the lower half plane since the program will always have found the conjugate root previously.

8. Removal of Accepted Roots. The basic quantities S_1 and S_2 are suitable for the subtraction of the effect of previously accepted roots. Suppose we have found $n - p$ roots $r_{p+1}, \cdots, r_n$; then we want to use the polynomial of degree p with roots $r_1, \cdots, r_p$ and observe that

$$S_1^{(p)} = \sum_{i=1}^{p} \frac{1}{z - r_i} = S_1^{(n)} - \sum_{i=p+1}^{n} \frac{1}{z - r_i},$$

$$S_2^{(p)} = \sum_{i=1}^{p} \frac{1}{(z - r_i)^2} = S_2^{(n)} - \sum_{i=p+1}^{n} \frac{1}{(z - r_i)^2}.$$

This is fortunate because we evaluate P, P', P'', and hence $S_1^{(n)}$ and $S_2^{(n)}$, from the Hessenberg matrix and so avoid the calculation of the coefficients of the characteristic polynomial. Thus we have no possibility of "dividing out" the previous roots by synthetic division. The work required to subtract out the known roots is dominated by the work required to find $S_1^{(n)}$ and $S_2^{(n)}$ and the method is very accurate for well separated roots in whatever order they are found.

For close roots the method is less satisfactory but the author knows of no better one.

9. When Has a Root Been Found? The author knows of no rigorous yet feasible numerical criterion for a given number to be an acceptable approximation to a zero of a polynomial even when "acceptable approximation" is clearly defined. Here are the tests used in the program of this paper which is designed for floating point numbers with 8 decimal mantissas.

Let z be the current iterate, Δz the computed increment, and $| z | = | \mathrm{Re}(z) | + | \mathrm{Im}(z) |$. Only the ratios

$$P(z):P'(z):P''(z)$$

will be available, not their individual values.

Test No. 1. $| P(z) | < 10^{-8} | z | | P'(z) |$. In practice 10^{-6} or 10^{-5} might be more realistic factors than 10^{-8}. Certainly many acceptable zeros might fail this test whose main purpose is to catch possible zero or very small values of $P(z)$ and also values of $S_1 = P'/P$ so large that there will be no change observable in z to 8 decimal places. The main concern is to detect convergence of the sequence of iterates. Now if

$$| \Delta z/z | < 10^{-4}$$

and cubic convergence (the number of correct digits trebles at each step) were in effect we would not expect to see any change in $z + \Delta z$ upon further iterations in an 8 decimal machine. Unfortunately this relative test is inapplicable to zero or very small eigenvalues and for these we compare Δz with c, the modulus of the largest eigenvalue (yet found).

Test No. 2. $|\Delta z| < 10^{-4} \max(|z|, 10^{-3}c)$.

Finally if convergence is slow (linear) then Test No. 2 is not fine enough and

$$|\Delta z| < 10^{-6} \max(|z|, 10^{-2}c)$$

would be more appropriate. For simplicity we take

Test No. 3. $|\Delta z| < 10^{-8}c$.

There remain two possible causes for failure to pass any of these tests. With complex eigenvalues cycles can occur (see Section 5) and as a crude diagnosis and cure for this possibility we use

Test No. 4. If, after 3 iterations, $|\Delta z| > |z|$ then* restart the iteration from infinity. If infinity could provoke a cycle (which is easily decided, see Section 12) then restart the iteration from the mean of the superdiagonal elements.

Finally an eigenvalue may be so ill-conditioned that none of these tests is passed. A weakening of the tests would mean accepting well-conditioned eigenvalues with unnecessary lack of accuracy. The program regards ill conditioning as the cause of failure to converge by the 15th step and the 16th iterate is accepted as an eigenvalue. Matrix No. 3 affords an example of this phenomenon. Another possibility is to switch to double precision evaluation of P, P', P'' after 15 iterations.

Further experience may dictate changes in some or all of these tests.

10. Evaluation of the Characteristic Polynomial and its Derivatives.

We consider a lower Hessenberg matrix H; $h_{ij} = 0$ if $j > i + 1$. The determinant of $H - zI$ may be found by using Gaussian elimination, a sequence of plane rotations, or Hyman's method, see [13]. We begin with a matrix which is real except possibly for the principal diagonal. The first two of the methods mentioned will lead, in general, to wholly complex matrices. All steps in the computation would have to be coded for complex arithmetic. However Hyman's method keeps any complex numbers confined to the diagonal.

It has been claimed, see [9, p. 430], that small numbers in the superdiagonal will cause inaccuracy in the computed determinant. However Wilkinson showed in [13] that Hyman's method in fact produces the exact determinant of a close Hessenberg matrix. Thus the result can only be inaccurate if the determinant is ill-conditioned and if this is the case no method can ameliorate the situation without recourse to higher precision.

In [13] Wilkinson advocates replacing zero elements in the superdiagonal by very small numbers. However, since the program for the method to be described is easily written so as to operate on any block of a Hessenberg matrix between zero superdiagonal elements we can take advantage of them and the consequent reduction of the full matrix.

Essentially Hyman's method consists of subtracting suitable multiples of the other columns of $H - zI$ from the first column so that it becomes null except pos-

* In more recent programs we use $|\Delta z_{i+1} + \Delta z_i| < .3|\Delta z_i|$ in place of $|\Delta z| > |z|$.

sibly for the last element which we shall call $f(z)$. Let $H = (h_{ij})$, $V_i = -h_{i,i+1} \neq 0$ $(i = 1, 2, \cdots, n-1)$, and $V_n = +1$. Then

$$P(z) = \det(H - zI) = \left(\prod_{i=1}^{n-1} V_i\right) f(z).$$

This constant non-zero coefficient of $f(z)$ can be ignored since it will eventually cancel out in the homogeneous expressions

$$S_1(z) = \frac{P'(z)}{P(z)} \quad \text{and} \quad S_2(z) = \frac{P'(z)^2 - P(z)P''(z)}{P(z)^2}.$$

Let the multiple of column i which must be added to column 1 to annihilate the $(i-1)$th element be u_i. Put $u_1 = 1$ and then the u_i $(i = 2, \cdots, n)$ are computed recursively from

$$u_{i+1} = (h_{i1}u_1 + h_{i2}u_2 + \cdots + h_{ii}u_i - zu_i)/V_i.$$

Since $V_n = 1$ we find that $u_{n+1} = f(z)$. The simple form of the relation invites both differentiation with respect to z and the taking of real and imaginary parts. This yields six real valued recursions for the calulation of the real and imaginary parts of $f(z)$, $f'(z)$, and $f''(z)$.

For a compact method of coding these computations in real arithmetic as a sequence of inner products see [7].

The use of an accumulated inner product routine would yield $f(z), f'(z), f''(z)$ with excellent accuracy at the cost of extra time. This extra time varies greatly with different machines. Such accuracy does not seem warranted except perhaps for the final approximation to each eigenvalue and, in practice, the single precision results seem quite adequate. The reader is referred to the examples at the end of the paper.

For complex z the evaluation of f, f', f'' requires approximately $3n(n+1)$ real multiplications and the same number of additions. These evaluations form the major part of the program and this shows the importance of minimising the number of evaluations.

11. The Possibility of Overflow. Using computers which use only numbers with moduli in the range $(10^{-50}, 10^{50})$ the danger of overflow in evaluating determinants is quite high. With the method givn above this danger is lessened because the determinant itself is not computed; rather

$$P(z) = kf(z), \qquad k = (-1)^n \prod_{i=1}^{n-1} h_{i,i+1}.$$

If the matrix H is scaled down by 10 then so will be $f(z)$, whilst $f'(z)$ will be unchanged, and $f''(z)$ will be scaled up by 10. Thus, in practice, we may leave H unaltered.

If overflow does occur then we can simply change the value of u_1 (see Section 10) and restart the computation. If the smallest possible value of u_1 still provokes an overflow then the value of z will have to be changed. In the notation of Section 10, $f(z) = u_{n+1}$. If overflow occurs in computing u_j then, for simplicity, we take $.9(j-1)z/(n+1)$ as a new initial value.

12. Starting an Iteration. The appropriate choice of starting value is of the utmost importance in the success of this method. Property (3) of Laguerre's iteration (see Section 4) can be used to provide a real starting value of modulus greater than the eigenvalue of greatest modulus provided that all the eigenvalues are real. If the largest eigenvalue is simple then the more it is isolated the closer will this number approximate it. This number is, in fact, the Laguerre iterate of infinity. Its use was suggested by the late H. J. Maehly.*

Let the eigenvalues of a Hessenberg matrix H satisfy $|\lambda_1| \leqq |\lambda_2| \leqq \cdots \leqq |\lambda_n|$. Let $L(z)$ be the iterate of z for the characteristic polynomial det $(H - zI)$ and let $l(z)$ denote the iterate of z for the reciprocal polynomial. Property (3) implies that

$$L(\infty) = \frac{1}{l(0)}.$$

For the reciprocal polynomial,

$$\sigma_1(0) = \sum_{i=1}^{n} \frac{1}{0 - \frac{1}{\lambda_i}} = -\sum_{i=1}^{n} \lambda_i, \qquad \sigma_2(0) = \sum_{i=1}^{n} \frac{1}{\left(0 - \frac{1}{\lambda_i}\right)^2} = \sum_{i=1}^{n} \lambda_i^2.$$

For the matrix H,

$$\sum_{i=1}^{n} \lambda_i = \text{trace } (H) = \sum_{i=1}^{n} h_{ii},$$

$$\sum_{i=1}^{n} \lambda_i^2 = \text{trace } (H^2) = \sum_{i=1}^{n} h_{ii}^2 + 2 \sum_{i=1}^{n-1} h_{i,i+1}h_{i+1,i}.$$

Hence, using the Laguerre formula, we have

$$L(\infty) = -\frac{1}{n}\left[\sigma_1(0) + \sqrt{(n-1)[n\sigma_2(0) - \sigma_1(0)^2}\right].$$

In the complex case we may also compute $L(\infty)$ but no longer is it necessarily an approximation to a maximal eigenvalue. Indeed if the eigenvalues are symmetric about the origin in such a way that

$$\text{trace } (H) = \text{trace } (H^2) = 0$$

then $L(\infty) = 0$ and $L(0) = \infty$ so that we have a cycle of length 2. However this particular danger is easy to detect and avoid.

In practice we have found that a starting value such as

$$z_0 = \left(\frac{1+i}{\sqrt{2}}\right)(\max_i \sum_j h_{ij})$$

is frequently 10 to 100 times greater in modulus than the largest eigenvalue. Consequently we use $z_0 = L(\infty)$ for the first initial value but subsequently switch to using a Newton iteration from the eigenvalue just found. This can be done in the following way.

* Private communication.

Let P be the original polynomial with zeros $r_1, \cdots, r_p$ of which zeros $r_{q+1}, \cdots, r_p$ have been found already. Let Q be the polynomial with zeros $r_1, \cdots, r_q$. We suppose that we have just found a simple zero r_q and wish to seek the zeros of the polynomial

$$R(z) = Q(z)/(z - r_q).$$

By equating Taylor Series about r_q it follows that

$$R'(r_q)/R(r_q) = Q''(r_q)/2Q'(r_q) = \frac{P''(r_q)}{2P'(r_q)} - \sum_{i=q+1}^{p} \frac{1}{r_q - r_i}.$$

A Newton iteration for R from r_q would yield

$$z_0 = r_q - R(r_q)/R'(r_q).$$

In practice we shall not know $R'(r_q)/R(r_q)$ but (ignoring roundoff errors) we can compute easily

$$s = \frac{P''(r_q^*)}{2P'(r_q^*)} - \sum_{i=q+1}^{p} \frac{1}{r_q^* - r_i}$$

where r_q^* (the iterate preceding r_q) is close to r_q. Now

$$\frac{P''(r_q)}{2P'(r_q)} = \frac{P''(r_q^*)}{2P'(r_q^*)}(1 - 2\delta) + \delta\frac{P'''(r_q^*)}{P'(r_q^*)} + 0(\delta^2), \qquad \delta = r_q - r_q^*.$$

So we may expect to use the formula $z = r - 1/s$ with safety provided that r_q is not close to any of the zeros $r_{q+1}, \cdots, r_p$. If r_q is close to a zero that has not yet been found then s will still be an adequate approximation to R'/R. Only when

$$\sum_{i=q+1}^{p} \frac{1}{r_q^* - r_i}$$

is large enough to provoke cancellation in the subtraction do we reject this starting value and use instead $L(\infty)$.

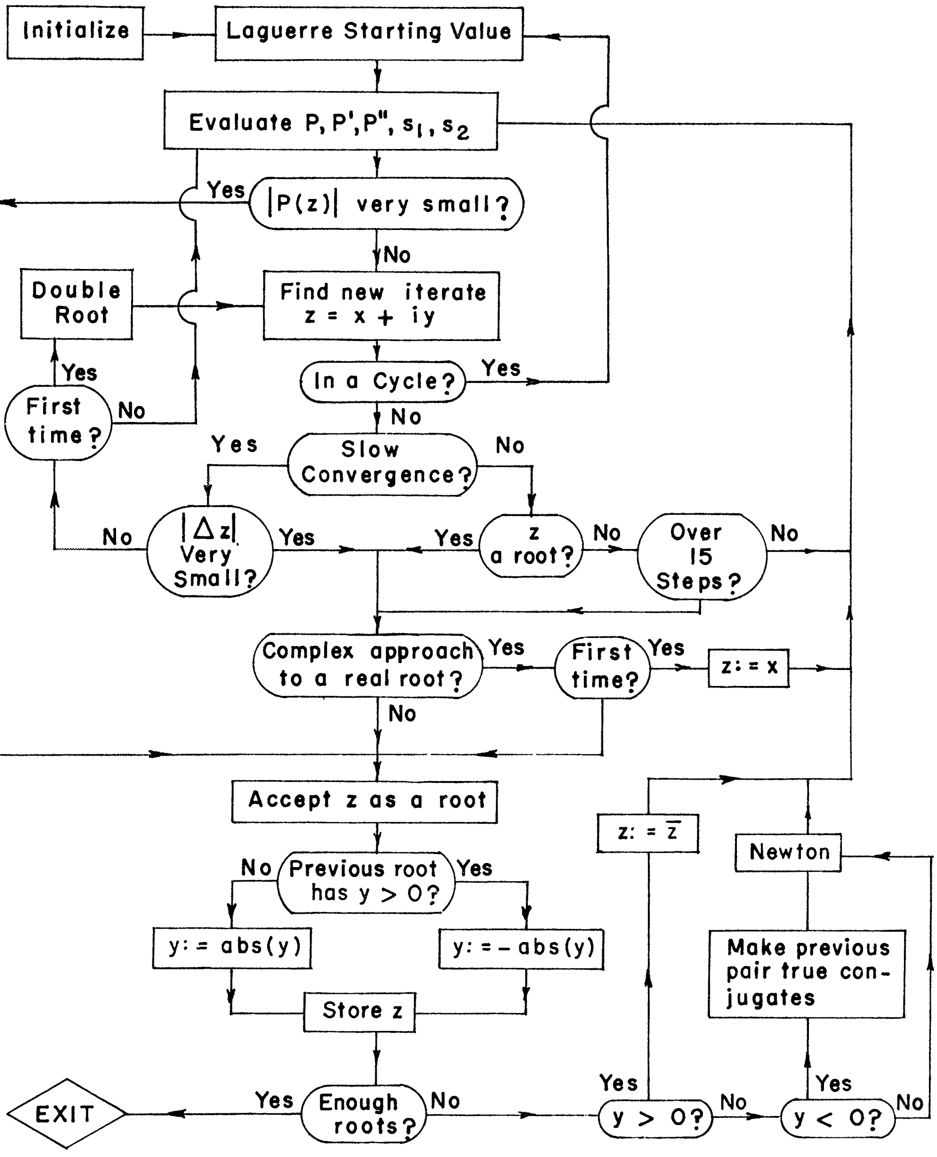
For the same reason we prefer *not* to evaluate R'/R as follows.

$$\begin{aligned}\frac{R'(r_q)}{R(r_q)} &= \frac{Q''(r_q)}{2Q'(r_q)} \\ &\doteqdot \frac{Q''(r_q^*)}{Q(r_q^*)} \Big/ \frac{2Q'(r_q^*)}{Q(r_q^*)} \\ &\doteqdot [S_1^2(r_q^*) - S_2(r_q^*)]/2S_1(r_q^*).\end{aligned}$$

If one of the remaining roots is close to r_q then S_1^2 and S_2 will be nearly equal and their computed difference will have few significant figures.

In practice the author has found that this technique seems to ensure that well-conditioned simple eigenvalues are found in 3 iterations (on the average) independent of the order of the matrix for orders up to 100. This is in contrast to a previous practice (see [7]) of always restarting with $L(\infty)$ for which the average number of iterations per eigenvalue rose from 3 up to 6 as the orders rose from 12 to 64.

13. FLOW CHART FOR LAGUERRE METHOD



14. ALGOL 60 Procedures.

begin

comment Before giving the procedure *Eig* 3 and its subsidiary procedures *Tringle* and *Laguerre* we list the supplementary subroutines needed by the method. These may conveniently be machine coded library subroutines if not built-in functions of the system being used;

```
real procedure  max(a,b);  value a, b;  real a, b;
                max := if a ≧ b then a else b;
```

```
real procedure  min(a,b);  value a,b;  real a,b;
                min := if a ≦ b then a else b;
integer procedure  mod(l,m);  value l, m;  integer l,m;
                   mod := l − m × (l ÷ m);
procedure  comsqrt (a,b,x,y);  value a, b;  real a, b, x, y;
  begin comment x + iy := (a+ib) ↑ (1/2), where x ≧ 0;  ⟨code⟩ end;
procedure  scale (V,j,k);  value j,k;  integer j,k;  array V;
begin
comment  the exponents of the elements V[j] through V[k] of V are scaled so that
  the greatest (algebraically) is zero;  ⟨code⟩
end;
Boolean procedure overflow;
  begin comment overflow = false unless machine overflow has occurred since the
    procedure was last invoked in which case overflow = true;  ⟨code⟩
  end;

procedure  Tringle (eps,n,A,INT);  value eps, n;  real eps;
  integer n;  array A;  integer array INT;
comment  Given the n × n matrix A Tringle uses a Crout-like elimination with
  interchanges, pivoting on A[i,i+1], to produce a similar almost lower triangular
  (Hessenberg) matrix written over A. INT[i] is the index of the column which was
  interchanged with column (i+1) at the ith stage. If, after interchange, A[i,i+1]
  is small then INT[i] := 0. A non-local procedure min(a,b) is used. The scalar
  products in statements D1 and D2 may be accumulated in double precision for
  greater accuracy;
begin integer i, j, k, l, m, j1, j2;
  real s, t, u;
for j := 1 step 1 until n − 1 do
begin comment  find max doomed element in row j;
  j1 := j+1;  j2 := j+2;
  l := j1;  s := abs(A[j,j1]);
  for k := j2 step 1 until n do
    begin t := abs(A[j,k]);  if t > s then
      begin l := k;  s := t end
    end;
  comment  Interchange rows and columns j+1, l;
  if l > j1 then
    begin for k := 1 step 1 until n do
      begin t := A[k,j1];  A[k,j1] := A[k,l];
           A[k,l] := t end k;
        for k := 1 step 1 until n do
    begin t := A[j1,k];  A[j1,k] := A[l,k];
         A[l,k] := t end k
  end if;
  comment  Are doomed elements negligible?;*
```

* In more recent programs we use $s \leqq eps \times sa$ where $sa = max_i\ (abs\ (A\ [j,\ i]))$, $i = 1, \cdots, j$.

```
  if s ≦ eps × min(abs(A[j,j]), abs(A[j1,j1])) then
     begin l := 0;
       for k := j2 step 1 until n do A[j,k] := 0;
       go to L
     end;
  t := A[j,j1];
  for k := j2 step 1 until n do A[j,k] := A[j,k]/t;
  comment update row j+1;
L: for i := 1 step 1 until n do
      begin u := 0;  m := min(j,i−2);
        if l = 0 then go to D2;
D1:     for k := j2 step 1 until n do u := u + A[k,i] × A[j,k];
D2:     for k := 1 step 1 until m do u := u − A[k,i] × A[j1,k+1];
        A[j1,i] := A[j1,i] + u
      end i;
INT[j] := l;
end j;
INT[n] := 0;
end Tringle;

procedure Laguerre (eps,n1,u,v,A,RTR,RTI); value eps, n1, u, v; integer n1,
   u, v; real eps; array A, RTR, RTI;
comment A is an almost triangular real matrix. This procedure uses the principal
   submatrix of A from A[u,u] through A[v,v] and calculates min(v−u+1,n1−u+1)
   of its eigenvalues to be stored in vectors RTR and RTI containing, respectively,
   the real and imaginary parts. It is assumed that no A[i,i+1] = 0. The integer
   mark tells which convergence test was passed by the eigenvalues. eps is a small
   tolerance. Procedures max, min, and comsqrt enter as non-local identifiers. A local
   procedure Evaluate is declared below.;
begin
  integer j, q, mark, tally;
  Boolean slow, once, vnear;
  real s1r, s1i, s2r, s2i, dr, di, er, ei, d1, d2, newdelta, newrate, oldelta, oldrate,
     eigsum1, eigsum2, spur1, spur2, x, y, deltax, deltay, d, cap, cup, g, h, t, zz;

  procedure Evaluate;
  comment Hyman's method is used in evaluating the logarithmic derivatives s1
     and s2 of the characteristic polynomial of the given submatrix of A, removing
     the contribution of eigenvalues already found. Important: eps, u, v, q, A, RTR,
     RTI, x, y, s1r, s1i, s2r, s2i, mark, and vnear enter as non-local entities, as do
     procedures mod, scale, and overflow;
  begin integer i, j, k, m, s;
     real q1r, q1i, q2r, q2i, d, d1, d2, t, t1r, t1i, t2r, t2i, r;
     array P[1:6,u:v+1], B[1:6];
mark := 0;  vnear := false;
  P[1,u] := 1.0;  for j := 2 step 1 until 6 do P[j,u] := 0;
  if abs(y) < abs(x) × eps then begin y := 0;  m := 3 end
  else m := 6;
  E1: for k := u step 1 until v do
         for j := 1 step 1 until m do
           begin s := sign (3.5−j);
             r := − x × P[j,k] + y × s × P[j+3 × s,k] −
               (if j = 1 then 0 else mod(j−1,3) × P[j−1,k]);
             for i := u step 1 until k do
             r := r + P[j,i] × A[k,i];
             if overflow then go to E0;
             P[j,k+1] := if k = v then r else −r/A[k,k+1]
           end j;
for j := 1 step 1 until m do B[j] := P[j,v+1];
for k := m + 1 step 1 until 6 do B[k] := 0;
comment  Scale down B to prevent B[i] × B[j] going out of range;
scale (B,1,m);
comment  Find the contributions q1, q2 to s1, s2 of the eigenvalues already found.
  Then compute t1 := P′/P, t2 := P″/P and hence s1 and s2;
q1r := q1i := q2r := q2i := 0;  t := x ↑ 2 + y ↑ 2;
for j := u step 1 until q do
  begin d1 := RTR[j] − x;  d2 := RTI[j] − y;
    d := d1 ↑ 2 + d2 ↑ 2;
    if d ≦ t × eps ↑ 3 then begin vnear := true;  go to E2 end;
    d1  := d1/d;  d2 := − d2/d;
    q1r := q1r + d1;  q1i := q1i + d2;
    q2r := q2r + d1 ↑ 2 − d2 ↑ 2;
    q2i := q2i + 2 × d1 × d2
  end j;
if (abs(q2r) + abs(q2i)) × t × eps ↑ 2 ≧ 1 then vnear := true;
d1 := B[1] ↑ 2 + B[4] ↑ 2;  d2 := B[2] ↑ 2 + B[5] ↑ 2;
if d1 ≦ d2 × t × 100 × eps ↑ 4 then go to E2;
t1r := B[2]/B[1];  t1i := 0;  t2r := B[3]/B[1];  t2i := 0;
if y ≠ 0 then begin t1r := (B[2] × B[1] + B[5] × B[4])/d1;
                    t1i := (B[5] × B[1] − B[4] × B[2])/d1;
                    t2r := (B[3] × B[1] + B[6] × B[4])/d1;
                    t2i := (B[6] × B[1] − B[4] × B[3])/d1
              end if;
s1r := t1r + q1r;  s1i := t1i + q1i;
s2r := t1r ↑ 2 − t1i ↑ 2 − t2r − q2r;
s2i := 2 × t1r × t1i − t2i − q2i;
go to E3;
  E0: P[1,u] := P[1,u] × 1.0₁₀−10;
      if P[1,u] = 0 then begin t := 0.9 × (k−u)/(v−u+1);  P[1,u] := 1.0;
                               x := t × x;  y := t × y end;
      overflow := false;  go to E1;
  E2: mark := 1;
  E3: end Evaluate;

comment  Start Laguerre;
once := slow := vnear := false;  q := u − 1;
```

```
tally := eigsum1 := eigsum2 := cap := cup := zz := 0;
oldelta := oldrate := 1.0;
for j := u step 1 until v−1 do cup := cup + abs(A[j,j+1]);
cup := cup/(v−u+1);
comment  Find the traces of $A$ and $A \uparrow 2$;
spur1 := A[u,u];  spur2 := spur1 ↑ 2;
for j := u + 1 step 1 until v do
   begin t := A[j,j];
         spur1 := spur1 + t;
         spur2 := spur2 + t ↑ 2 + 2 × A[j−1,j] × A[j,j−1]
   end j;
comment  Use the iterate of infinity in obtaining an initial approximation;
L1: s1r := eigsum1 − spur1;
    s2r := spur2 − eigsum2;
    if abs(s1r) + abs(s2r) ≦ eps × zz then
       begin y := 0; x := cup;  go to Hy end if;
    dr := (v−q−1) × ((v−q) × s2r − s1r ↑ 2);
    er := sqrt(abs(dr));
    if dr < 0 then begin x := −s1r/(v−q); y: = er/(v−q) end
    else begin y := 0;  x := −(s1r + (if s1r ≧ 0 then er else − er))/(v−q)
             end;
    if q = 0 then begin x := 2 × x;  y := 2 × y end;
Hy: Evaluate;
if mark = 1 then go to L5;
tally := tally + 1;  g := v − q;
L2: if abs(y) > newdelta then begin s1i := s1i + 1/(2 × y);
                                    s2r := s2r + 1/(4 × y ↑ 2);
                                    g := g − 1
                              end if;
comment  Compute Laguerre's formula: $new := old - g/(s1+d \uparrow (1/2))$;
if slow then h := 0.5 × (g−2) else h := g − 1;
dr := h × (g × s2r−s1r ↑ 2+s1i ↑ 2);
di := h × (g × s2i−2 × s1r × s1i);
if di = 0 then begin t := sqrt(abs(dr));
                     er := max(0,sign(dr) × t);
                     ei := max(0, −sign(dr) × t)
               end
else comsqrt(dr,di,er,ei);
if (s1r × er + s1i × ei) < 0 then begin er := −er;  ei := −ei end;
d1 := s1r + er;  d2 := s1i + ei;  t := d1 ↑ 2 + d2 ↑ 2;
deltax := −g × d1/t;  x := x + deltax;
deltay := g × d2/t;  y := y + deltay;
newdelta := abs(deltax) + abs(deltay);
newrate := newdelta/oldelta;
d := abs(x) + abs(y);  zz := max(d, eps × cap);
if tally ≦ 3 then go to Lt;
```

```
comment  test for cycle;
if (newdelta ≧ max(3 × oldelta, .5×d)) then
  begin oldelta := cup;  oldrate := 3.0;
        go to if tally ≦ 15 then L1 else L5
  end;
comment  if the iterates seem linear after 3 iterations it is assumed to be due to a
  double root;
if newrate ≧ .8 × oldrate then
  begin mark := 3;
        if newdelta < eps ↑ 2 × 100 × zz then go to L4;
        if ¬ slow then begin x := x − deltax;  y := y − deltay;
                             slow := true;  go to L2
                       end
        else slow := false;  go to L3
  end if;
comment  Test for an eigenvalue and at L4 test for a complex approach to a real
  eigenvalue;
Lt: mark := 2;  if newdelta < eps × zz then go to L5;
L3: oldelta := newdelta;  oldrate := newrate;
go to if tally > 15 then L4 else Hy;
L4: slow := false;
   if (y=0)∨(abs(y)×(abs(s1r)+abs(s1i))≧1)∨ once then go to L5;
   once := true;  y := 0;  go to Hy;
comment  Accept x + y √−1 as an eigenvalue;
L5: q := q + 1;  RTR[q] := x;
   if abs(y) < .01 × eps × zz then y := 0;  if q = u then go to L6;
   if 0 < RTI[q−1] then begin RTI[q] := y := −abs(y);  go to L7 end;
L6: RTI[q] := y := +abs(y);
L7: cap := max(d,cap);  tally := 0;  oldelta := oldrate := 1;
   once := false;
   eigsum1 := eigsum1 + x;
   eigsum2 := eigsum2 + x ↑ 2 − y ↑ 2;
   if q ≧ n1 then go to FINIS;
comment  If RTI[q] > 0 then x − y √−1 is the new initial approximation.
  A rough conjugate complex pair of eigenvalues are made exactly conjugate;
if y > 0 then begin y := − y;  go to Hy end;
if y < 0 then begin RTI[q−1] := .5 × (RTI[q−1] − RTI[q]);
                    RTI[q] := −RTI[q−1];
                    RTR[q] := RTR[q−1] := .5 × (RTR[q−1]+RTR[q])
              end;
comment  A Newton step to start the next search. See Section 12 for an alternate
  formula;
if v − u − q < 3 ∨ vnear then go to L1;
dr := s1r ↑ 2 − s1i ↑ 2 − s2r;  di := 2 × s1r × s1i − s2i;
d2 := dr ↑ 2 + di ↑ 2;
if d2 = 0 then go to L1;
```

```
x := x − 2×(s1r × dr + s1i × di)/d2;  y := abs(y − 2×(s1i×dr−s1r×di)/d2);
go to if abs(x) + abs(y) > 2 × cap then L1 else Hy;
FINIS: end Laguerre;

procedure Eig3 (m,n,A,RTR,RTI,eps);  value m, n, eps;  real eps;  integer m,
  n;  array A,RTR,RTI;
comment  Eig3 finds m(≦n) eigenvalues of the real n × n matrix A and stores
  the real parts in RTR and the imaginary parts in RTI. Non-local procedures
  Tringle, Laguerre, and max are used. If the machine word has a mantissa of s
  bits to the base b then it is recommended that eps := 1/b ↑ (s÷2);
begin integer j, k, u, v;
      real p,q,r,s,d,e,f,spura,spurb,spurc;
      integer array C[1:n];
spura := spurb := spurc := 0;
for j := 1 step 1 until n do spura := spura + A[j,j];
Tringle (eps, n, A, C);
for j := 1 step 1 until n do spurb := spurb + A[j,j];
comment  Search for any zero superdiagonal elements thus finding A in reduced
  form. Dispense with Laguerre for 1 × 1 and 2 × 2 submatrices;
u := v := 0;
L1: if v ≧ n then go to L4 else u := v := v + 1;
L2: if C[v] = 0 then go to L3;
    v := v + 1;  go to L2;
L3: if v > u + 1
      then Laguerre (eps,min(m,v),u,v,A,RTR,RTI)
      else begin
             if v = u then begin RTI[u] := 0;
                                 RTR[u] := A[u,u]
                           end
             else begin p := A[u,u];  q := A[v,v];
                        r := .5 × (p + q);
                        d := .25 × ((p−r)−(q−r)) ↑ 2 + A[u,v]×A[v,u];
                        s := sqrt(abs(d));
                        e := 1 − max(0, − sign(d));
                        f := p × q − A[u,v] × A[v,u];
                        RTR[u] := r + .5 × ((p−r)+(q−r)) + (if r≧0 then e
                          else −e) × s;
                        RTI[u] := − (e−1) × s;
                        d := RTR[u] ↑ 2 + RTI[u] ↑ 2;
                        RTR[v] := f × RTR[u]/d;
                        RTI[v] := −f × RTI[u]/d;
                  end
           end;
      go to L1;
L4: for j := step 1 until m do spurc := spurc + RTR[j];
comment  spura,spurb,spurc may be written out for comparison;
end Eig3;
end
```

15. Numerical Results. We give below the results of applying the method discussed above, here called Method I, to some test matrices. For a comparison we also used an inner product routine which accumulates the sums in double precision for the reduction to Hessenberg form. This is Method II. On the first 4 test matrices Method II produced results which were negligibly different from those of Method I and only the latter will be given in these cases.

The program was coded in FORTRAN (see SHARE ref. 1373) and run on the IBM 7090 at the AEC Computing Center, New York University, New York. The column headed 0 indicates the order in which the roots were found. The column headed I indicates the number of iterations required.

Matrix No. 1 (Rosser [8]), 8 $\times$ 8.

$$\begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ & & 899 & 196 & 61 & 49 & 8 & 52 \\ & & & 611 & 8 & 44 & 59 & -23 \\ & & & & 411 & -599 & 208 & 208 \\ \text{symmetric} & & & & & 411 & 208 & 208 \\ & & & & & & 99 & -911 \\ & & & & & & & 99 \end{bmatrix}$$

Trace of Hessenberg form = 4039.9999.
Time <1.8 secs.

Correct (8 digits)	Computed, $\epsilon = 10^{-4}$	0	I
1020.0490	1020.0500	1	9
1020.0000	1019.9997	2	1
1019.9020	1019.9019	3	6
1000.	1000.0001	4	5
1000.	999.99999	5	1
.098048640	.098045509	8	1
0.0	−.0000094··	7	3
−1020.0490	−1020.0490	6	1
4040.0	4040.0007		27

Average no. of iterations per root 3.375.
Matrix No. 2 (P. A. White [9]), 10 $\times$ 10.

$$A = \begin{bmatrix} B & 2B \\ 4B & 3B \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 10^{-5} & 0 & 0 & 0 & 0 \end{bmatrix}$$

Eigenvalues of A:
$.5 \exp(2\pi ik/5)$,
$-.1 \exp(2\pi ik/5)$,
$k = 1, 2, 3, 4, 5$.

Trace of Hessenberg form: 0.
Computed eigenvalues were in error by less than 2 units in the last (8th) decimal place.
Time <3 secs. Average no. of iterations per root 3.

Matrix No. 3 (Frank [4], Wilkinson [13]), 12 × 12.

$$\begin{bmatrix} 12 & 11 & & & & & & \\ 11 & 11 & 10 & & & & & \\ 10 & 10 & 10 & 9 & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & & & \\ 1 & 1 & 1 & 1 & 1 & \cdot & \cdot & 1 \end{bmatrix}$$

Trace: 78.0
Time < 9 secs.

Correct (8 digits)	Computed, $\epsilon = 10^{-4}$	0	I
32.228891	32.228895	1	3
20.198988	20.198989	2	3
12.311077	12.311077	3	3
6.9615330	6.9615331	4	3
3.5118559	3.5118560	5	3
1.5539887	1.5539888	6	3
.64350532	.64351736	7	3
.28474972	.28432775	8	16
.14364652	.13838053	9	16
.081227659	.071327899*	12	16
.049507429	.036500148*	11	16
.031028060	.020891586*	10	16
78.0	77.961279		101

Average no. of iterations per root 8.416.

Matrix No. 4 (Eberlein [3]), 16 × 16.

$$A = \begin{bmatrix} B & 2B \\ 4B & 3B \end{bmatrix}, \quad B = \begin{bmatrix} 5C & -C \\ 5C & C \end{bmatrix}, \quad C = \begin{bmatrix} -2 & 2 & 2 & 2 \\ -3 & 3 & 2 & 2 \\ -2 & 0 & 4 & 2 \\ -1 & 0 & 0 & 5 \end{bmatrix}$$

Eigenvalues: $60 \pm 20i, 45 \pm 15i, 30 \pm 10i, 15 \pm 5i, -12 \pm 4i, -9 \pm 3i, -6 \pm 2i, -3 \pm i$.
Trace of Hessenberg form: 239.99999.
Sum of computed eigenvalues: 239.99999.
All eigenvalues had at least six figures correct.
The greatest error (relative and absolute) occurred in computing $-9 \pm 3i$.
$-9.0000061 + 2.9999923i$, $-9.0000055 - 2.9999945i$.
Time < 6 secs.
Average no. of iterations per eigenvalue 2.69.

* Imaginary part <.05. The ill-conditioning of the small eigenvalues is revealed by the iteration count reaching its permitted maximum.

Matrix No. 5 (Eberlein [3]), 40 × 40.

$$A = \begin{bmatrix} 8B & 4B \\ -5B & -B \end{bmatrix}, \quad B = \begin{bmatrix} C & 2C \\ 4C & 3C \end{bmatrix},$$

$$C = \begin{bmatrix} 4D & 3D \\ -2D & -D \end{bmatrix}, \quad D = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$j = 1 \pm \sqrt{-3}$

Eigenvalues: 3, 6, −15, −30, $\pm 1.5j$, $\pm 3j$, $\pm 7.5j$, $\pm 15j$, 4, 8, −20, −40, $\pm 2j$, $\pm 10j$, $\pm 20j$.

	Method I	Method II
Trace of Hessenberg form:	−83.999968	−83.999983
Sum of computed eigenvalues:	−83.999982	−84.000007
Greatest errors (absolute and relative) occurred in the small real roots	3.0001076	2.9999915
	3.9999094	4.0000106
	5.9999073	6.0000135
	8.0000845	7.9999872
Time <	.91 minutes	1.1 minutes
Av. no. of iterations per root	3.0	3.0

Matrix No. 6 (Parlett), 64 × 64.

$$A = \begin{bmatrix} B & 2B \\ 4B & 3B \end{bmatrix}, \quad B = \begin{bmatrix} 3C & 3C \\ 5C & C \end{bmatrix},$$

$$C = \begin{bmatrix} 6D & -D & D & 0 \\ 8D & 0 & D & 2D \\ -2D & 0 & D & 2D \\ 5D & -D & -D & D \end{bmatrix}, \quad D = \begin{bmatrix} -2 & 2 & 2 & 2 \\ -3 & 3 & 2 & 2 \\ -2 & 0 & 4 & 2 \\ -1 & 0 & 0 & 5 \end{bmatrix}$$

$j = 3 \pm \sqrt{-1}$, $k = 1 \pm 2\sqrt{-1}$.

Eigenvalues: $120m$, $-40m$, $-24m$, $8m$, $90m$, $-30m$, $-18m$, $6m$, $60m$, $-20m$, $-12m$, $4m$, $30m$, $-10m$, $-6m$, $2m$.

$m = j, k$

		Method I	Method II
Trace of Hessenberg form:		1279.9994	1279.9995
Sum of computed eigenvalues:		1279.9994	1279.9996
Greatest error:		12.000138 + 3.9997210i:	12.000018 + 3.9999665i
		12.000118 − 3.9997633i:	11.999996 − 4.0000147i
No. of conjugate prs. with errors in last p figures	$p = 4$	3	0
	3	11	7
	2	16	12
	1	2	13
Time <		4.75 minutes	5.54 minutes
Av. no. of iterations per root		2.5	2.5

Matrix No. 7 (Parlett), 100×100.
$A = [a_{ij}]$; $a_{ij} = 0$ ($i < j$, with two exceptions);
$a_{ii} = 101 - i$; $a_{ij} = (-1)^{i+j+1}40/(i + j - 2)$ $(i > j)$.
In order to obtain a lower Hessenberg form with a minimal number of zero elements we put $a_{12} = 40/102$, $a_{1,100} = 40$.
Eigenvalues: 100, 99, $\cdots$, 2, 1.

	Method I	Method II
Trace of Hessenberg form:	5049.9985	5049.9973
Sum of computed eigenvalues:	5049.9988	5049.9978
No. of correct digits (at least)		
100, 99, 98, 97, 96	8, 7, 6, 4, 4	8, 7, 7, 7, 6
95, $\cdots$, 89	4	5
88, $\cdots$, 86	5	5
85, $\cdots$, 72	5	6
71, $\cdots$, 55	6	6
54, $\cdots$, 1	6	7
Time	12 minutes	15 minutes
Av. no. of iterations per root	3	3

Tentative conclusion: Method II will give one more digit correct than will Method I but at the cost of a 20% increase in computing time for matrices of order greater than 30.

In many cases fewer iterations are required and results are slightly improved by starting at 0 instead of ∞. However this is not a safe general practice.

New York University
Courant Institute of Mathematical Sciences
AEC Computing and Applied Mathematics Center
New York 3, New York

1. J. W. Backus et al., "Report on the algorithmic language Algol 60," *Numer. Math.* v. 2, 1960, p. 106–136.
2. E. Durand, *Solutions Numériques des Equations Algébriques*, Vol. 2, Masson et Cie, Paris, 1960.
→ P. J. Eberlein, "A Jacobi-like method for the automatic computation of eigenvalues and eigenvectors of an arbitrary matrix," *J. Soc. Indust. Appl. Math.*, v. 10, 1962, p. 74–88.
→ W. L. Frank, "Computing eigenvalues of complex matrices by determinant evaluation and by methods of Danilewski and Wielandt," *J. Soc. Indust. Appl. Math.*, v. 6, 1958, p. 378–392.
5. E. N. Laguerre, *Oeuvres de Laguerre*, Gauthier-Villars, Paris, Vol. 1, p. 87–103.
6. H. J. Maehly, "Zur Iterativen Auflösung Algebraischer Gleichungen," *Z. Angew. Math. Phys.*, v. 5, 1954, p. 260–263.
7. B. N. Parlett, *Applications of Laguerre's Method to the Matrix Eigenvalue Problem*, Tech. Report No. 21, Stanford Univ., Applied Math. and Stat. Labs., Contract Nonr 225(37), 1962.
8. J. B. Rosser et al., "Separation of close eigenvalues of a real symmetric matrix," *J. Res. Nat. Bur. Standards*, v. 47, 1951, p. 291–297.
→ P. A. White, "The computation of eigenvalues and eigenvectors of a matrix," *J. Soc. Indust. Appl. Math.*, v. 6, 1958, p. 393–437.
10. J. H. Wilkinson, *Determination of Characteristic Values and Characteristic Vectors*, Application of Advanced Numer. Anal. to Digital Computers, Summer Session, 1958, Univ. of Michigan, Ann Arbor, Mich., p. 101–154.
11. J. H. Wilkinson, *Notes on Practical Methods of Solving Linear Systems and Calculating the Eigensystems of Matrices*, National Physical Laboratory, Teddington, England, 1959.

12. J. H. WILKINSON, *Advanced Numerical Analysis*, Summer Session, 1960, Univ. of Michigan, Ann Arbor, Mich.
13. J. H. WILKINSON, "Error analysis of floating-point computation," *Numer. Math.*, v. 5, 1960, p. 319–340.
14. J. H. WILKINSON, "Stability of the reduction of a matrix to almost triangular and triangular form by elementary similarity transformations," *J. Assoc. Comput. Mach.*, v. 6, 1959, p. 336–359.
15. J. H. WILKINSON, *The Numerical Eigenvalue Problem,* Oxford Univ. Press, in preparation.