The QR Transformation—Part 2

By J. G. F. Francis

The QR transformation is an analogue to the LR transformation (Rutishauser, 1958) based on unitary transformations. Both these transformations are global iterative methods for finding the eigenvalues of a matrix, the matrix converging in general to triangular form. In Par t1 of this paper the QR transformation was briefly described and we were then principally concerned with proving convergence, the main result being expressed in theorem 3. We also showed that if the matrix is first reduced to almost triangular form important advantages are gained (further advantages will become apparent) and we gave in outline a way in which convergence could be improved. In this part of the paper we consider the practical application of the QR transformation. Two versions of the algorithm have been programmed for the Pegasus computer; these are described and an attempt is made to evaluate the method. Some results and detailed algorithms are given in appendices. Part 1 was published on pp. 265–71 of this volume (Oct. 61).

9. The Basic Iteration, Conjugate Complex Eigenvalues

The QR transformation consists of forming a sequence of matrices $A^{(k)}$, where

$$A^{(1)} = A, \text{ the given matrix,}$$

and $$A^{(k+1)} = Q^{(k)*}A^{(k)}Q^{(k)}.$$

The matrix $Q^{(k)}$ is unitary and is chosen so that $Q^{(k)*}A^{(k)} = R^{(k)}$ where $R^{(k)}$ is an upper triangular matrix.

In fact we shall use the generalized transformation (12) discussed in Section 8 in which $Q^{(k)}$ is chosen so that $Q^{(k)*}(A^{(k)} - \zeta^{(k)}I) = R^{(k)}$ because a suitable choice of the origin shift $\zeta^{(k)}$ accelerates the convergence of the last diagonal element. We call the transformation of $A^{(k)}$ into $A^{(k+1)}$ an *iteration*.

We are mainly concerned in this paper with the use of the QR transformation in finding the eigenvalues of non-hermitian matrices which we shall consider to have been reduced to almost triangular form. If the matrix is real we must expect conjugate complex pairs of eigenvalues to be present. If these pairs are the only eigenvalues of equal modulus—which is likely—we know by theorem 9 that under the transformation the pairs will become associated with separate 2×2 submatrices of which we can find the eigenvalues by solving quadratic equations.

The method of accelerating the convergence of the last diagonal element by using the generalized transformation is virtually useless when we are concerned with a pair of complex eigenvalues if the origin is shifted only by real quantities. This is clearly because we cannot move arbitrarily close to either of the roots by such a shift. We must therefore use a complex shift in this case, and this would appear to involve us in generating complex matrices although the matrix is initially real.

To avoid this we develop a more sophisticated technique. Intuitively we would expect that a scheme of iteration using pairs of conjugate origin shifts could be devised that would accelerate the convergence of a "quadratic factor," preserving the real nature of the matrix. This would be analogous to the Bairstow process for finding pairs of roots of polynomials.

First we extend theorem 2 to cover the generalized QR transformation employing shifts of origin. Suppose that the origin shifts $\zeta^{(r)}$ are distinct from the eigenvalues: then in the sequel we write $\mathbf{\Pi}^{(k)} = \prod_{1}^{k}(A - \zeta^{(r)}I)$, and $\mathbf{\Pi}^{(k)}$ is non-singular.

THEOREM 10. *If the generalized QR transformation is performed on the matrix $A^{(1)} = A$ with origin shifts $\zeta^{(r)}$ then the unitary matrix $P^{(k)} = Q^{(1)} \ldots Q^{(k)}$ such that $A^{(k+1)} = P^{(k)*}AP^{(k)}$ can be derived from the unitary-triangular decomposition of $\mathbf{\Pi}^{(k)}$.*

PROOF. We have $A^{(k)} - \zeta^{(k)}I = Q^{(k)}R^{(k)}$ and, substituting this in $P^{(k)}S^{(k)} = Q^{(1)} \ldots Q^{(k)}R^{(k)} \ldots R^{(1)}$,

we obtain

$$P^{(k)}S^{(k)} = Q^{(1)} \ldots Q^{(k-1)}(A^{(k)} - \zeta^{(k)}I)R^{(k-1)} \ldots R^{(1)}.$$

As $$Q^{(1)} \ldots Q^{(k-1)}A^{(k)} = AQ^{(1)} \ldots Q^{(k-1)}$$

we get

$$P^{(k)}S^{(k)} = (A - \zeta^{(k)}I)Q^{(1)} \ldots Q^{(k-1)}R^{(k-1)} \ldots R^{(1)}.$$

Clearly by repeated substitution

$$P^{(k)}S^{(k)} = (A - \zeta^{(k)}I)(A - \zeta^{(k-1)}I) \ldots (A - \zeta^{(1)}I) = \mathbf{\Pi}^{(k)}$$

and as the right-hand side is non-singular it uniquely determines the decomposition on the left-hand side (theorem 1).

Now if the matrix is real and any iteration with a complex shift $\zeta^{(r)}$ is followed by one using the conjugate shift so that $\zeta^{(r+1)} = \overline{\zeta}^{(r)}$, then the product $\mathbf{\Pi}^{(k)}$ is also real. Thus, since the unitary-triangular decomposition of a real matrix is also real, the matrix $P^{(k)}$ and hence $A^{(k+1)} = P^{(k)*}AP^{(k)}$ is real too. It is thus clear that, at the expense of doing two complex iterations instead of one, when we are concerned with a complex pair of eigenvalues we can return to the real form of the matrix

and still retain accelerated convergence. We base the two origin shifts on the roots of the submatrix

$$\begin{bmatrix} a_{n-1,n-1}^{(k)} & a_{n-1,n}^{(k)} \\ a_{n,n-1}^{(k)} & a_{nn}^{(k)} \end{bmatrix};$$

it can be shown that the roots of this submatrix then tend to λ_n and $\lambda_{n-1} = \overline{\lambda}_n$ when $\zeta^{(k)}$ and $\zeta^{(k-1)}$ are sufficiently close to these eigenvalues.

This technique unfortunately involves us in complex arithmetic in the intermediate stages, and appreciably more work than that required by two iterations with real shifts. Since the matrix is real before and after a "double iteration," we are led to look for a purely real process which can be uniformly employed for origin shifts which are either both real or else conjugate complex.

Now we know that

$$A^{(k-2)} = Q^{(k-1)*}Q^{(k)*}A^{(k)}Q^{(k)}Q^{(k-1)}$$

and from theorem 10 that

$$(A^{(k)} - \zeta^{(k)}I)(A^{(k)} - \zeta^{(k-1)}I) = Q^{(k)}Q^{(k-1)}R^{(k-1)}R^{(k)}.$$

Thus one method would be to form the real matrix $\Gamma = (A^{(k)} - \zeta^{(k)}I)(A^{(k)} - \zeta^{(k-1)}I)$, compute its unitary-triangular decomposition to obtain $Q^{(k)}Q^{(k-1)}$, and transform $A^{(k)}$ by means of this, giving $A^{(k-2)}$. This process requires a prohibitive amount of work but we show below that when the matrix is almost triangular it is unnecessary to compute more than the first column of Γ, and that this immediately gives the transformation to be applied to $A^{(k)}$.

Suppose an almost triangular matrix with real non-negative subdiagonal elements is similar, through a unitary transformation, to another matrix (whether almost triangular or not). Then the first column of the transformation in general uniquely determines the almost triangular matrix. If we write $F = W^*AW$, where F is almost triangular and W unitary, we can derive:

$$\left.\begin{aligned} &f_{ij} = w_i^*Aw_j \quad \text{for } i = 1, \ldots, j \\ &\text{and, if} \quad h = Aw_j - \sum_{i=1}^{j} f_{ij}w_i, \\ &\text{then } f_{j+1,j} = ||h|| \text{ and } w_{j+1} = \frac{1}{f_{j+1,j}}.h \end{aligned}\right\} \text{for } j = 1, \ldots \qquad (13)$$

We can thus see that the transformation is uniquely characterized by w_1 up to the first zero subdiagonal element $f_{j+1,j}$. If w_1 can be analysed into a linear combination of all the right eigenvectors, and there are no repeated eigenvalues, then h will not vanish till $j = n$.

We can now prove an important theorem:

THEOREM 11. *Suppose that A is a diagonalizable almost triangular matrix with real, positive (non-zero) subdiagonal elements, and suppose that it is transformed by a unitary matrix W into another almost triangular matrix W^*AW which has real, non-negative subdiagonal elements. Then if the first column of W is given by $w_1 = \frac{1}{||\pi_1^{(k)}||}.\pi_1^{(k)}$ where $\pi_1^{(k)}$ is the first column of $\Pi^{(k)}$, the resulting matrix is identical to the kth matrix generated by the generalized QR transformation, so that $A^{(k-1)} = W^*AW$.*

PROOF. The vector e_1 is a combination of all the right eigenvectors v_i of A because the proportion in it of each is given by $u_i^*e_1 = \bar{u}_{1i}$ and by theorem 8 none of the elements u_{1i} can be zero. Hence w_1 contains all the right eigenvectors because none of the factors $A - \zeta^{(r)}I$ of $\Pi^{(k)}$ can eliminate them from e_1 in $\pi_1^{(k)} = \Pi^{(k)}e_1$, the $\zeta^{(r)}$ being distinct from the eigenvalues. Thus we see that w_1 uniquely determines W^*AW, the subdiagonal elements being non-zero. Now from theorem 10 we have

$$\Pi^{(k)}e_1 = P^{(k)}S^{(k)}e_1 = s_{11}^{(k)}p_1^{(k)} = \pi_1^{(k)}$$

and clearly $s_{11}^{(k)} = ||\pi_1^{(k)}||$ because it is real and positive and $||p_1^{(k)}|| = 1$. Hence $w_1 = p_1^{(k)}$ and $A^{(k-1)} = P^{(k)*}AP^{(k)} = W^*AW$, being unique.

This theorem now forms the basis of the technique we use for performing a double iteration. At the kth stage of the QR transformation we devise a real transformation of $A^{(k)}$ such that $A^{(k-2)} = W^*A^{(k)}W$ with the sole requirement that

$$\kappa w_1 = (A^{(k)} - \zeta^{(k)}I)(A^{(k)} - \zeta^{(k-1)}I)e_1 = \gamma_1$$

where κ is a normalizing factor and $\zeta^{(k)}$ and $\zeta^{(k-1)}$ are either both real or conjugate complex origin shifts. Thus we do not have to find the transformation by the unitary-triangular decomposition of

$$(A - \zeta^{(k)}I)(A - \zeta^{(k-1)}I)$$

but instead we base it on only the first column of this matrix, γ_1, which is trivial to compute. The equations (13) above provide a possible way of performing the transformation, but for this method to be stable in practice each new column w_{j+1} has to be "reorthogonalized" against all the previous w_i. In fact we shall use an elimination procedure and this is much more economical; it is described in detail later.

10. Implications of Theoretical Conditions

Before going into the details of a QR algorithm it is necessary to remark on some of the conditions imposed in the theoretical treatment. Iteration on singular matrices and on matrices with zero subdiagonal elements α_i has been excluded. Yet it would appear that our whole objective is to reduce the elements α_i effectively to zero and to hasten this process by origin shifts as close to the eigenvalues as possible.

The shift of origin will certainly tend to make the matrix $A^{(k)} - \zeta^{(k)}I$ singular, and when singular (or, in practice, nearly singular) the iteration (however computed) will not be fully determined. In theory, if the $\alpha_i^{(k)}$ are non-zero, the matrix after the iteration will be fully defined excepting only the nth column. This will not matter because the nth diagonal element will then

be the eigenvalue, and the last subdiagonal element $a_n^{(k+1)}$ will be zero, allowing the matrix to be deflated by the omission of its nth row and column. (The double iteration technique in theory will behave similarly, the last two rows and columns being relevant in this case.) Wilkinson (1959) has demonstrated that a process of deflation like this due to Givens (1958) is quite unreliable in practice. With this process the origin is shifted to an arbitrary eigenvalue, and because this shift and the arithmetic cannot be *exact* it can happen that the element of the transformed matrix which should be equal to the eigenvalue is grossly inaccurate, and its "coupling" off-diagonal element is not small. In these circumstances, reducing the order of the matrix will have disastrous consequences.

Fortunately, however, the objections to the Givens transformation do not apply to our case. Primarily this is because we never deflate the matrix until we know that we can—that is, not until the last subdiagonal element is effectively zero. The worst that can happen is that we continue to iterate and however ill-determined the transformations may be, though they may upset convergence, they cannot radically effect the eigenvalues because we can always ensure that they are unitary. Secondly, the origin shifts we use are essentially of a type which behave stably in the Givens deflation process and hence in our iterations leading up to deflation. Wilkinson shows that if the nth component of the row eigenvector associated with the origin shift is large, then the practical process behaves as it should in theory. This is exactly the situation we have, for our origin shifts are based on the eigenvalue to which the nth diagonal element is converging, and obviously the associated eigenvector tends to e_n^*. Thus we can confidently expect the iterations to be stable, any ill-determinacy not disturbing convergence, and this is confirmed in practice. (The most unstable situation would occur if an origin shift were to be based on the first diagonal element.) If near-singularity should occur which is not associated with the last diagonal element, it is possible to show that a limited number of iterations will always suffice to convert the matrix to a form in which it is.

As far as the subdiagonal elements are concerned they present no difficulty for the first method described below. When they become effectively zero we partition or deflate the matrix, and we do this if some are initially zero. If they are small then the transformations will be sufficiently defined, for ratios of two small quantities will not normally be involved. If such ratios are involved, however, we have a situation closely connected with the effect of unstable near-singularity and, as we have stated, this will correct itself after a small number of iterations. In the second method to be described, essential significance is lost in certain transitional subdiagonal elements unless they are scaled up. If this is done the situation is as for the first method.

Another point to consider is the presence of eigenvalues of equal modulus. The only serious problems that arise are concerned with the conjugate complex eigenvalues of a real matrix and with equal eigenvalues. Other cases are hardly likely to cause any difficulty (none has occurred in practice) because real origin shifting with a real matrix and complex shifting with a complex matrix are almost certain to break up the groups in question. We have already outlined the strategy for dealing with complex conjugate pairs in a real matrix, and will expound it in detail below. In the theoretical treatment we have said (theorem 7) that equal eigenvalues of a diagonalizable almost triangular matrix will necessarily be associated from the start with isolated principal submatrices, and we have ignored the question of matrices that cannot be diagonalized. Analysis indicates that a group of equal eigenvalues, belonging to a non-linear elementary divisor of a non-diagonalizable matrix, will become associated with a principal submatrix, as will most other groups of eigenvalues with equal modulus.

In practice, on account of round-off errors, one is concerned not so much with strict equality and whether the matrix can be diagonalized or not, but with the extent to which very close eigenvalues are well or ill-determined. In the diagonalizable case we shall generally not obtain zero subdiagonal elements in the initial reduction to almost triangular form, and the eigenvalues will appear in the same submatrix (not exactly equal) in the way that those of a non-diagonalizable matrix do. We can certainly say that all such eigenvalues will become separately resolved to an extent depending on how well determined they are. In the worst cases there may be no convergence, but with origin shifting we will usually obtain a linear rate of convergence or better; in the best cases they will be almost completely resolved from the start. (Of course, using a double iteration technique we can always solve a quadratic to find just two equal eigenvalues.)

11. Practical Computation

We describe below two different algorithms in which the QR transformation is applied to almost triangular matrices. Programs using these algorithms have been written for the Pegasus computer. The first is the simplest, but it does not deal very satisfactorily with a real matrix with complex roots; the second is designed for this problem and is based on the method introduced in Section 9.

An iteration can be performed in various ways. Different methods may or may not require the explicit formation of either the matrix $\boldsymbol{R}$ or the matrix $\boldsymbol{Q}$, or $\boldsymbol{Q}$ may exist only as a product of factors. Methods also vary in the number of multiplications and storage locations they require, their ease of computation, and their stability. Almost triangular matrices are certainly best transformed by an elimination procedure, for the number of multiplications will then be proportional to the square, not the cube, of the order n.

First Method

In the first method we shall consider the matrix to be either real or complex. The $n-1$ subdiagonal elements $a_{i+1,i}$ $(i=1, 2, \ldots, n-1)$, which at present we shall assume to be non-zero and real, are eliminated in turn by a series of elementary unitary transformations starting with a_{21}. This is essentially the process described in Section 3 applied to almost triangular matrices. A typical stage in the reduction would be*

$$\begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \bar{\mu} & \nu & & & \\ & & -\nu & \mu & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix} \begin{bmatrix} r & r & r & r & r & r & r \\ & r & r & r & r & r & r \\ & & \beta & y & y & y & y \\ & & \alpha & x & x & x & x \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \end{bmatrix} = \begin{bmatrix} r & r & r & r & r & r & r \\ & r & r & r & r & r & r \\ & & \kappa & y' & y' & y' & y' \\ & & & x' & x' & x' & x' \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \end{bmatrix}$$

where the elements r, y' and κ are those of the final triangular matrix, and the elements a, x and α are in their original state. The elements in the two rows that are changed are given by:

$$\left.\begin{array}{lll} & \kappa = (|\beta|^2 + \alpha^2)^{\frac{1}{2}} & \text{(real)} \\ & x' = \mu x + \nu y \text{ and } y' = \bar{\mu} y - \nu x & \text{(complex)} \\ \text{where} & \mu = \beta/\kappa & \text{(complex)} \\ & \nu = \alpha/\kappa & \text{(real).} \end{array}\right\} \quad (14)$$

In this way all of the elements below the diagonal are eliminated, using the multipliers μ_i and $\nu_i (i=1, \ldots, n-1)$, leaving real elements on the diagonal, except for the last. This last element β becomes $\kappa = |\beta|$ by a further elementary transformation which multiplies the last row by $\bar{\mu}_n = \bar{\beta}/\kappa$. We have then formed the matrix $\boldsymbol{R} = \boldsymbol{Q}^*\boldsymbol{A}$, and $\boldsymbol{Q}^*$ is equal to the product of the elementary unitary transformations used.

We now want to find the matrix $\boldsymbol{RQ}$ and this is done by post-multiplying $\boldsymbol{R}$ in turn by the conjugate transpose of the elementary unitary transformations employed in forming it. We thus have as a typical stage:

$$\begin{bmatrix} a & a & x & y & r & r & r \\ a & a & x & y & r & r & r \\ & a & x & y & r & r & r \\ & & & \kappa & r & r & r \\ & & & & r & r & r \\ & & & & & r & r \\ & & & & & & r \end{bmatrix} \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \mu & -\nu & & & \\ & & \nu & \bar{\mu} & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix} = \begin{bmatrix} a & a & x' & y' & r & r & r \\ a & a & x' & y' & r & r & r \\ & & a & x' & y' & r & r & r \\ & & & \alpha & \beta & r & r & r \\ & & & & & r & r & r \\ & & & & & & r & r \\ & & & & & & & r \end{bmatrix}.$$

Here the elements r, y and κ are so far unaltered from the triangular state, and the elements a, x' and α are in their final state. We have in the two columns that are changed:

$$\begin{array}{ll} \alpha = \nu\kappa & \text{(real)} \\ \beta = \bar{\mu}\kappa & \text{(complex)} \\ x' = \mu x + \nu y \text{ and } y' = \bar{\mu} y - \nu x & \text{(complex).} \end{array}$$

The operation completing the iteration consists of the multiplication of the last column by μ_n.

* We will sometimes use a matrix of a fixed rather than general order for illustration.

The iteration is quite easily programmed in fixed-point arithmetic, but in this case care must be taken to preserve the unitary property of the transformation by ensuring that $|\mu_i|^2 + \nu_i^2 = 1$ for each i (when $i = n$, "ν_i" $= 0$). This is done by shifting up the double-length quantity $|\beta|^2 + \alpha^2$ in (14) before taking the square root by the largest possible even number of binary places such that it is still fractional. If this number of places is 2θ we form $\tilde{\kappa} = \{2^{2\theta}(|\beta|^2 + \alpha^2)\}^{\frac{1}{2}}$, $\tilde{\alpha} = 2^{\theta}\alpha$ and $\tilde{\beta} = 2^{\theta}\beta$ and use these shifted quantities in the ratios μ and ν. We obtain $\kappa = 2^{-\theta}\tilde{\kappa}$ as the diagonal element.

If the matrix is real all the arithmetic in an iteration is real; if the elements below the diagonal are real (this will always be the case) but the rest of the matrix is complex we see that the ν_i are real and the subdiagonal elements remain real after each iteration. There are about $4n^2$ multiplications in a real iteration and $12n^2$ in the complex case.

In the Pegasus computer the matrix is stored by columns on a magnetic drum, and efficient access to the rows of the matrix is impossible. It might seem that this would cause the first part of an iteration to be very inefficient, but we deal with this difficulty by a change in the order of the operations. We perform the row operations one column at a time: that is, on eliminating the element $a_{i+1,i}$ we will have correctly performed the first i row operations on those parts of the rows up to and including their ith elements. We then proceed to the next column and perform the appropriate operations on its elements before eliminating $a_{i+2,i+1}$. This is continued until the last column is reached. No difficulty over access to the elements occurs in the second part of an iteration.

The programs which have been written will find the eigenvalues of arbitrary real matrices (initially reducing them to almost triangular form). The convergence of the matrix under repeated application of the above iteration is first order—the pth eigenvalue in order of

size appearing as the pth diagonal element unless there are others of the same modulus. When there are, for example, two eigenvalues of equal modulus, say the pth and the $(p+1)$th, these will become the eigenvalues of the pth 2×2 submatrix on the diagonal (see theorem 9).

We accelerate the convergence of the smallest eigenvalue according to the principle described in Section 8. We subtract a quantity $\zeta^{(k)}$ from the diagonal of the matrix before the kth iteration and add it back again afterwards. Before the kth iteration we find the roots of the last principal 2×2 submatrix and then distinguish two cases. If these roots are real we choose the one that differs least from the last diagonal element $a_{nn}^{(k)}$ and we call this quantity $\lambda^{(k)}$. We then compare this with the previous similar quantity $\lambda^{(k-1)}$ (initially $\lambda^{(0)} = 0$), which we have retained, by testing the size of $\left|\dfrac{\lambda^{(k)} - \lambda^{(k-1)}}{\lambda^{(k)}}\right|$. If this quantity is less than $\frac{1}{2}$ we set $\zeta^{(k)} = \lambda^{(k)}$, but otherwise $\zeta^{(k)} = 0$. Thus the second order process of origin shifting starts when about one binary place of significance has been achieved. (See footnote on page 339.)

The other case to be considered occurs when the roots of the last 2×2 submatrix are complex. If it is known that the matrix has only real eigenvalues we should use the real part of these roots as the potential origin shift; otherwise, with the present algorithm, we may be compelled to do a complex iteration, for we set $\lambda^{(k)}$ to one of the 2×2 roots, compare it with $\lambda^{(k-1)}$ (disregarding the sign of the imaginary part), and make $\zeta^{(k)} = \lambda^{(k)}$ or 0. If $\zeta^{(k)}$ is non-zero the iteration, being complex, is immediately followed by another using the conjugate shift $\zeta^{(k-1)} = \overline{\zeta}^{(k)}$. The matrix then returns to the real form and we set $\lambda^{(k-1)} = \lambda^{(k)}$ for the next comparison. (This procedure is rather unattractive on account of the complex arithmetic necessary, but it was incorporated in the program because the more sophisticated method discussed later had not been developed at the time.)

The arithmetic for a real iteration is performed by a separate section of the program from that used for a complex one, and it is appreciably faster. Unfortunately it is not strictly accurate to say that an iteration employing a complex shift followed by one using the conjugate shift returns the matrix to the real form. It is probably possible for the second of a pair of such iterations to be badly determined, with the effect that some of the imaginary parts will not disappear. This has not yet caused any difficulty in practice, and it is possible that it only can happen in the last two columns in circumstances when the arithmetic will remain complex until deflation occurs, and the situation rights itself. The only purpose of returning each time to a real matrix is to save time when finding real roots. If we give up this objective (for example, for matrices with few real roots) the difficulty can be overcome by using complex arithmetic once a complex shift has occurred for all subsequent iterations, and in this case it is more efficient to omit the alternate conjugate shifts. However, because the danger is mainly hypothetical, it is probably worth making a simple test on the sum of the moduli of the imaginary parts, and to use real arithmetic for a real shift unless this is not sufficiently small.

If the program had been written for finding the eigenvalues of general complex matrices (including normal matrices), we would have none of the above difficulties. We would simply use a series of complex shifts determined in the same way as the real shifts with a real matrix.

The subdiagonal elements a_{nn-1} and $a_{n-1\,n-2}$ are tested after each iteration. If either are zero we deflate the matrix and print out the one or two eigenvalues found. Here we can mention another point: while the iterations proceed, all the subdiagonal elements (or at least those separating certain submatrices) are tending to zero, and some may well become zero. We therefore inspect them before each iteration and only carry out the iteration on the lowest part of the matrix such that they are all non-zero. This saves unnecessary work and also ensures that a condition is fulfilled for some earlier remarks and theorems to be true.

Second Method

The method we have just described is clearly not very satisfactory when the matrix is real and yet has complex eigenvalues, and this has prompted the development of the second technique which combines two iterations in one real operation.

We know that the result of two iterations,

$$\boldsymbol{A}^{(k+2)} = \boldsymbol{Q}^{(k+1)*}\,\boldsymbol{Q}^{(k)*}\,\boldsymbol{A}^{(k)}\,\boldsymbol{Q}^{(k)}\,\boldsymbol{Q}^{(k+1)},$$

is such that if $\boldsymbol{W} = \boldsymbol{Q}^{(k)}\boldsymbol{Q}^{(k+1)}$ and

$$\boldsymbol{\Gamma} = (\boldsymbol{A}^{(k)} - \zeta^{(k)}\boldsymbol{I})(\boldsymbol{A}^{(k)} - \zeta^{(k+1)}\boldsymbol{I}),$$

then $\boldsymbol{W}^*\boldsymbol{\Gamma} = \boldsymbol{\Delta}$ (say) is an upper triangular matrix. However, we showed in Section 9 that it is unnecessary to form the matrix $\boldsymbol{\Gamma}$ to find $\boldsymbol{W}$, because $\boldsymbol{A}^{(k+2)}$ is fully determined by the first column only of $\boldsymbol{\Gamma}$, combined with the almost triangular natures of $\boldsymbol{A}^{(k)}$ and $\boldsymbol{A}^{(k+2)}$. The first column of $\boldsymbol{\Gamma}$, $\boldsymbol{\gamma}_1$, gives the first column of $\boldsymbol{W}$ which defines the transformation.

Now $\boldsymbol{W}^*$ is a unitary matrix which reduces $\boldsymbol{\Gamma}$ to the triangle $\boldsymbol{\Delta}$, and $\boldsymbol{W}$ is therefore composed of n unitary factors (see Section 3) of the form $\boldsymbol{N}_i = \begin{bmatrix} \boldsymbol{I}_{i-1} & 0 \\ 0 & \boldsymbol{M}_i \end{bmatrix}$ so that $\boldsymbol{W} = \boldsymbol{N}_1\,\boldsymbol{N}_2 \ldots \boldsymbol{N}_n$. The pre-multiplication of $\boldsymbol{\Gamma}$ successively by the first $n-1$ $\boldsymbol{N}_i^*$ causes the elements below the diagonal to be eliminated in turn column by column; the last factor $\boldsymbol{N}_n^*$ (in which $\boldsymbol{M}_n^*$ is scalar) merely ensures that δ_{nn} is positive. From the form of each $\boldsymbol{N}_i$ we see that the first column of $\boldsymbol{W}$ is equal to the first column of $\boldsymbol{N}_1 = \boldsymbol{M}_1$, and this is any unitary matrix the transpose of which eliminates the unwanted elements in the first column of $\boldsymbol{\Gamma}$. (There are two such elements to eliminate because $\boldsymbol{\Gamma}$ is the product of two almost triangular matrices and has all $\gamma_{ij} = 0$ for $i > j+2$.)

We wish to transform $\boldsymbol{A}^{(k)}$ by means of $\boldsymbol{W}$. As a first step let us operate on it with $\boldsymbol{N}_1$, which is determined so that $\boldsymbol{N}_1^*\boldsymbol{\gamma}_1 = \boldsymbol{\delta}_1 = \delta_{11}\boldsymbol{e}_1$ by any suitable

unitary elimination procedure. This will change the first three rows and columns of $A^{(k)}$, so we have

$$A^{(k)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & . & . & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & . & . & a_{2n} \\ & a_{32} & a_{33} & a_{34} & . & . & a_{3n} \\ & & a_{43} & a_{44} & . & . & a_{4n} \\ & & & a_{54} & . & . & . \\ & & & & . & . & . \\ & & & & & . & a_{nn} \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} \underline{a}'_{11} & \underline{a}'_{12} & \underline{a}'_{13} & \underline{a}_{14} & . & . & \underline{a}_{1n} \\ \underline{a}'_{21} & \underline{a}'_{22} & \underline{a}'_{23} & \underline{a}_{24} & . & . & \underline{a}_{2n} \\ \underline{a}'_{31} & \underline{a}'_{32} & \underline{a}'_{33} & \underline{a}_{34} & . & . & \underline{a}_{3n} \\ a'_{41} & a'_{42} & a'_{43} & a_{44} & . & . & a_{4n} \\ & & & a_{54} & . & . & . \\ & & & & . & . & . \\ & & & & & . & a_{nn} \end{bmatrix} = N_1^* A^{(k)} N_1$$

where the elements changed by the row and column operations are underlined and primed respectively. The resulting matrix is no longer almost triangular and so, since $A^{(k+2)}$ is, we can say that the matrices $N_2, \ldots N_n$ reduce $N_1 A^{(k)} N_1^*$ to almost triangular form. This reduction is easily accomplished column by column by any unitary elimination procedure, and since such a process will use a series of transformations which have exactly the same form as $N_2, \ldots, N_n$, the product of all the transformations on $A^{(k)}$ will have the required first column—that of N_1. The technique we have derived for a "double iteration" is therefore to perform an initial transformation on $A^{(k)}$, and to reduce the resulting matrix to almost triangular form by a method such as that due to Givens (1958) or Householder (Wilkinson, 1960), obtaining $A^{(k+2)}$. The individual transformations are not uniquely determined but on account of our earlier reasoning their product must be so, provided the subdiagonal elements of $A^{(k)}$ are non-zero.

A typical stage in the iteration can be illustrated thus:

$$\begin{bmatrix} h & h & h & z & z & z & z & z & z \\ h & h & h & z & z & z & z & z & z \\ & h & h & z & z & z & z & z & z \\ & & d & d & d & z & z & z & z \\ & & d & d & d & z & z & z & z \\ & & d & d & d & a & a & a & a \\ & & & & & a & a & a & a \\ & & & & & & a & a & a \\ & & & & & & & a & a \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} h & h & h & h' & z' & z' & z & z & z \\ h & h & h & h' & z' & z' & z & z & z \\ & h & h & h' & z' & z' & z & z & z \\ & & \underline{h} & \underline{h}' & \underline{z}' & \underline{z}' & \underline{z} & \underline{z} & \underline{z} \\ & & \underline{0} & \underline{d}' & \underline{d}' & \underline{d}' & \underline{z} & \underline{z} & \underline{z} \\ & & \underline{0} & \underline{d}' & \underline{d}' & \underline{d}' & \underline{z} & \underline{z} & \underline{z} \\ & & & d' & d' & d' & a & a & a \\ & & & & & & a & a & a \\ & & & & & & & a & a \end{bmatrix}$$

where we indicate the elements changed by the transformation as above, three rows and three columns being affected. The elements a and h are those of the initial and final matrices $A^{(k)}$ and $A^{(k+2)}$ respectively; the elements d may need special treatment and are discussed below.

We observe that the first transformation of $A^{(k)}$, by N_1, is similar to the subsequent transformations, and we can set up the initial conditions for the iteration by finding γ_{11}, γ_{21} and γ_{31}; these are given by

$$\begin{aligned} \gamma_{11} &= a_{11}(a_{11} - \sigma) + a_{12} \cdot a_{21} + \rho \\ \gamma_{21} &= a_{21}(a_{11} + a_{22} - \sigma) \\ \text{and} \qquad \gamma_{31} &= a_{21} \cdot a_{32} \end{aligned}$$

where σ and ρ are the sum and product of the two origin shifts being used. For purposes of computation it is useful to imagine the column $\boldsymbol{\gamma}_1$ adjoined to the matrix on its left. The transformation by N_{n-1} is atypical in that only one element, that in position $(n, n-2)$, is eliminated by it, and it therefore only effects two rows and columns—the last—of the matrix, and the transformation by N_n is nugatory and need not be performed, only affecting signs in the last row and column (since the matrix is real).

As we mentioned earlier, the transformation leads to $A^{(k+2)}$ only when it is unique. This is not true if a subdiagonal element of $A^{(k)}$ is zero, and we must therefore ensure uniqueness by iterating on only a principal submatrix if some $a_{i+1,i} = 0$. However, the transformation is also in danger of being ill-determined on account of the smallness of the subdiagonal elements, for at least some of them are tending to zero and in this case will involve us in ratios of small quantities. This is why the submatrix of elements,

$[d_{rs}] = \begin{bmatrix} d_{10} & d_{11} & d_{12} \\ d_{20} & d_{21} & d_{22} \\ d_{30} & d_{31} & d_{32} \end{bmatrix}$, involved in each stage of the iteration needs special treatment. Even though they are small, these elements are still able to determine the transformations sufficiently well (that is, as well as two steps of the first method would determine them) for the transformations can be identified individually with the parallel set of transformations N_i by which we could reduce $\boldsymbol{\Gamma}$ to $\boldsymbol{\Delta}$, provided that these are defined by the same scheme of elimination. If the N_i are so defined, the three elements d_{10}, d_{20} and d_{30} in the ith stage of the iteration in column $i-1$ are in fact a multiple (perhaps small) of the three elements which determine the ith transformation in the triangularization of $\boldsymbol{\Gamma}$. These are the elements in positions (i, i), $(i+1, i)$ and $(i+2, i)$ of $N_{i-1}^* \ldots N_2^* N_1^* \boldsymbol{\Gamma}$. It is therefore necessary to attach some form of scale factor to the d_{rs} below the diagonal, and in practice it has proved convenient to carry out the operations on all nine elements d_{rs} with floating-point arithmetic.

It should be emphasized that, if these elements were not be to be treated in this or an equivalent way, inaccuracies would not be introduced into the eigenvalues of the matrix, but the convergence of the algorithm

would be jeopardized. On a machine with built-in floating-point arithmetic, it would naturally be best to perform all the operations in an iteration with floating-point arithmetic, and then those on the d_{rs} would not be separated from the rest. On Pegasus, however, floating-point operations have to be specially programmed, so the main part of the iteration is done with fixed-point arithmetic.

Householder's method has been chosen for carrying out the eliminations as it appears to involve least work, and the program is easy to write. With this method the transformation matrices are of the form

$$\boldsymbol{N}_i = \boldsymbol{N}_i^* = \boldsymbol{I} - 2\boldsymbol{t}_i\boldsymbol{t}_i^*$$

where $\boldsymbol{t}_i$ is a vector such that $||\boldsymbol{t}_i|| = 1$, of which the elements t_{ij} are given by:

$$t_{ij} = 0 \text{ for } j < i \text{ and } j > i + 2,$$

and
$$t_{ii} = \left\{\frac{1}{2}\left(1 + \frac{d_{10}}{\kappa}\right)\right\}^{\frac{1}{2}}$$

$$t_{ii+1} = \frac{d_{20}}{2\kappa t_{ii}}$$

$$t_{ii+2} = \frac{d_{30}}{2\kappa t_{ii}}$$

where $\kappa = \text{sign } d_{10} \cdot (d_{10}^2 + d_{20}^2 + d_{30}^2)^{\frac{1}{2}}$. We remove one of the square roots here and reduce the number of multiplications involved in the transformation by writing

$$\boldsymbol{z}_i^* = [0, \ldots, 0, 1, \psi_1, \psi_2, 0, \ldots, 0] = \frac{1}{t_{ii}} \cdot \boldsymbol{t}_i^*$$

obtaining $\boldsymbol{N}_i = \boldsymbol{I} - (1 + \phi)\boldsymbol{z}_i\boldsymbol{z}_i^*$

where $\phi = \dfrac{d_{10}}{\kappa}$, $\psi_1 = \dfrac{d_{20}}{\kappa + d_{10}}$ and $\psi_2 = \dfrac{d_{30}}{\kappa + d_{10}}$. (15)

These last three quantities cannot be greater than one, and therefore can be stored in fixed-point form for the fixed-point part of the iteration. If they are formed with floating-point arithmetic there is no difficulty, but if they should be formed with fixed-point arithmetic the unitary property of $\boldsymbol{N}_i$ would have to be maintained, either by shifting up their components (as in the first method) or else by computing ϕ as $\dfrac{2}{1 + \psi_1^2 + \psi_2^2} - 1$. (Whatever the inaccuracies in ψ_1 and ψ_2 the second technique makes $\boldsymbol{N}_i$ unitary, and this may even commend it in floating-point working for reducing round-off errors.)

As illustrated above the ith $(i + 1)$th and $(i + 2)$th rows and columns are changed when the matrix is multiplied before and after by $\boldsymbol{N}_i$. For the operation on the columns we have

$$[\underline{\boldsymbol{a}}_i, \underline{\boldsymbol{a}}_{i+1}, \underline{\boldsymbol{a}}_{i+2}] = [\boldsymbol{a}_i, \boldsymbol{a}_{i+1}, \boldsymbol{a}_{i+2}]\left[\boldsymbol{I} - (1 + \phi)\begin{bmatrix}1\\ \psi_1\\ \psi_2\end{bmatrix}[1, \psi_1, \psi_2]\right]$$

so that, if $\boldsymbol{\eta} = (1 + \phi)(\boldsymbol{a}_i + \psi_1\boldsymbol{a}_{i+1} + \psi_2\boldsymbol{a}_{i+2})$,

then
$$\left.\begin{aligned}\underline{\boldsymbol{a}}_i &= \boldsymbol{a}_i - \boldsymbol{\eta}\\ \underline{\boldsymbol{a}}_{i+1} &= \boldsymbol{a}_{i+1} - \psi_1\boldsymbol{\eta}\\ \underline{\boldsymbol{a}}_{i+2} &= \boldsymbol{a}_{i+2} - \psi_2\boldsymbol{\eta}\end{aligned}\right\} \quad (16)$$

and there are similar formulae for the operation on the rows. The column operation makes the elements in positions $(i + 3, i)$ and $(i + 3, i + 1)$, non-zero—they become new d_{30} and new d_{31}—and it can be easily verified that the row operation eliminates d_{20} and d_{30} in positions $(i + 1, i - 1)$ and $(i + 2, i - 1)$ as is required, and replaces d_{10} in position $(i, i - 1)$ by $-\kappa$. This last point implicitly contravenes the restriction imposed in Part 1 on the diagonal elements of the triangular part of the unitary-triangular decomposition. There we required these elements to be positive so that the decomposition should be unique. Waiving this restriction, however, is of no practical consequence, for the diagonal elements, and the size of those off the diagonal, in the successive matrices generated will still be unchanged. The condition is also violated by our not applying the final transformation matrix $\boldsymbol{N}_n$.

On a machine which can hold the whole almost triangular matrix in immediate-access storage, each row operation can be carried out and completed at once, as can each column operation. On Pegasus, storing the matrix by columns, we meet the problem of access to rows in a similar way to that described in the first method.

We have stated above that it is essential to partition the matrix and operate on a submatrix when any element $a_{i+1\,i} = 0$. This is no disadvantage and is in fact the obvious device used in the first method, mainly to save unnecessary work. (If the elements of the matrix are stored in floating-point form they will not normally vanish at any stage, but for the purposes of saving work and testing convergence the subdiagonal elements should be treated as zero when less than a suitable amount.) We thus scan the subdiagonal elements before each iteration to find the smallest q such that all $a_{i+1\,i} \neq 0$ for $q \leqslant i < n$.

The program written for this method also incorporates a further device which often results in a significant saving of work. When the product of the $(i - 1)$th and the ith subdiagonal elements is small enough to be treated as zero, it may be possible to start the iteration at a_{ii} rather than a_{qq}. Consider the situation if we were to do this: we find the initial quantities ϕ, ψ_1 and ψ_2 from a_{ii}, a_{ii-1}, $a_{i+1\,i}$, $a_{i+1\,i+1}$ and $a_{i+2\,i+1}$ and perform (notionally) the first row operation. The previously zero elements in the $(i + 1)$th and $(i + 2)$th positions of column $(i - 1)$ would become, using (16), $-\psi_1\eta$ and $-\psi_2\eta$, respectively, where $\eta = (1 + \phi)a_{ii-1}$. Thus from (15) these two elements become $-\dfrac{a_{ii-1}}{\kappa} \cdot d_{20}$ and $-\dfrac{a_{ii-1}}{\kappa} \cdot d_{30}$. Clearly, if both these are sufficiently small they can be ignored and we can start the iteration in column i. A satisfactory criterion for this proves to be the size of

$$\delta_i = |a_{ii-1}| \cdot \frac{|d_{20}| + |d_{30}|}{|d_{10}|}$$

$$= \left| a_{ii-1}a_{i+1i} \frac{|a_{ii} + a_{i+1i+1} - \sigma| + |a_{i+2i+1}|}{a_{ii}(a_{ii} - \sigma) + a_{ii+1}a_{i+1i} + \rho} \right|$$

which should be less than $\varepsilon/2$ where ε is the smallest effectively non-zero number (on Pegasus $\varepsilon = 2^{-38}$). Thus, at the same time as inspecting the subdiagonal elements for finding q, the program finds the smallest p such that all $\delta_i > \varepsilon/2$ for $p \leqslant i \leqslant n-3$. Clearly $1 \leqslant q \leqslant p < n$.

Using this system we thus start an iteration at the pth row and column of the matrix, each column operation beginning at the qth element of each column. Notice that if $p > q$ it is necessary to change the sign of the element a_{pp-1}.

After each iteration the roots of the 2×2 principal submatrix $\begin{bmatrix} a_{n-1n-1} & a_{n-1n} \\ a_{n\,n-1} & a_{n\,n} \end{bmatrix}$ are found, and the subdiagonal elements a_{nn-1} and a_{n-1n-2} are inspected. If either or both of these is zero (or effectively zero) the program prints the appropriate eigenvalue or eigenvalues, and the order of the matrix is reduced by one or two. If the order of the reduced matrix is less than three the remaining eigenvalues are found and printed; otherwise the roots of the new lowest principal 2×2 submatrix are calculated and the new subdiagonal elements a_{nn-1} and a_{n-1n-2} are tested (as after an iteration), and we either return to iterate again, if they are non-zero, or else print the appropriate eigenvalues and deflate as above.

Before each iteration we thus have the roots of the last principal 2×2 submatrix. We call these $\lambda^{(k)}$ and $\lambda^{(k+1)}$, ordered to differ least from a_{n-1n-1} and a_{nn} respectively, and, having retained the two previous similar roots $\lambda^{(k-2)}$ and $\lambda^{(k-1)}$, we calculate $\left|\frac{\lambda^{(k)} - \lambda^{(k-2)}}{\lambda^{(k)}}\right|$ and $\left|\frac{\lambda^{(k+1)} - \lambda^{(k-1)}}{\lambda^{(k+1)}}\right|$. The choice of the origin shifts $\zeta^{(k)}$ and $\zeta^{(k+1)}$ for the iteration depends on these quantities. If they are both greater than $\frac{1}{2}$ we set $\zeta^{(k)} = \zeta^{(k+1)} = 0$; if they are both less than $\frac{1}{2}$ we set $\zeta^{(k)} = \lambda^{(k)}$ and $\zeta^{(k+1)} = \lambda^{(k+1)}$; otherwise we set both $\zeta^{(k)}$ and $\zeta^{(k+1)}$ to be the real part of either $\lambda^{(k)}$ or $\lambda^{(k-1)}$, whichever corresponds to the quantity less than $\frac{1}{2}$.* We then have $\rho = \zeta^{(k)}\zeta^{(k-1)}$ and $\sigma = \zeta^{(k)} + \zeta^{(k+1)}$ and proceed to find p and q for the iteration.

The system of origin shifting used here and in the first method is not the only one possible and may not be the best. For example, there are some advantages in using a "non-restoring" type of shifting, in which any shift, or pair of shifts, which is set zero in the present system is replaced by the shift or pair of shifts used in the previous iteration. This technique, however, would not be suitable in the first method when we perform complex shifts, which are alternately conjugate, with an essentially real matrix.

* The experimental work has been done with a criterion of $\frac{1}{2}$, but this is somewhat arbitrary and there are indications that a smaller amount, say $\frac{1}{8}$, would be better.

12. Conclusion

The principal merits and demerits of the two versions of the QR algorithm will probably be clear to the reader at this stage. The first method has the advantage of being conceptually simple, and the programming would present no difficulties on most machines. Experiments have shown that it is a powerful technique for finding the eigenvalues of real unsymmetric matrices when the eigenvalues are all real. Unfortunately this is often not the case, and there are distinct disadvantages when the method is adapted to finding conjugate complex pairs of eigenvalues of a real matrix, the main one being the increase which becomes necessary in the amount of work and the storage space required by the complex arithmetic. In this instance one would naturally wish to avoid the complex arithmetic—the problem being essentially real—but in the case of the general matrix with complex elements the method would seem to be promising.

The second variation of the algorithm is designed to deal satisfactorily with the general case of a real unsymmetric matrix having both real and complex eigenvalues. The arithmetic is real throughout and there is no significant distinction between finding real and complex roots. No more storage space is required than that needed for the original matrix. It is remarkable how much less work it involves than the first method in finding complex pairs of roots, for two complex iterations of the first method involve about $24n^2$ (real) multiplications as opposed to $5n^2$ multiplications of the double iteration of the second method. (Use of the Householder rather than the Givens type of eliminations would only reduce the factor 24 to 20.) The only disadvantage of the second method is the necessity of using some sort of floating representation for some of the quantities. This causes no difficulty on machines with built-in floating-point arithmetic, and on such machines the algorithm is very simple to program.

The general strategy of the QR transformation as we have presented it (particularly in the second method) has much to commend it. There are only two stages—reduction to almost triangular form and iteration—and neither involves using multiples greater than one. One would therefore expect the round-off errors to build up only gradually, and in practice this is borne out, no stage requiring greater than single-length precision. Unitary similarity transformations cannot change the condition number of a matrix for each eigenvalue (see Fike, 1959), and the eigenvalues which we find will be the exact eigenvalues of a matrix which differs by a relatively small amount from the original matrix. The process is also one which combines guaranteed linear convergence with convergence of a higher order, and therefore the origin shifting technique has a certain sureness and stability as well as efficiency. Deflation is automatic, and criteria for convergence are simple.

The two programs have been tried out in experiments on a number of different matrices, the second technique being used on about 25 matrices of orders from 3 to 32 (up to 49 is acceptable). No difficulties have yet been encountered, in spite of close or repeated roots. The number of iterations required is often remarkably few. This is probably due to the fact that the convergence of the smallest eigenvalue (or pair of eigenvalues) is very rapid once a few figures have been found. Each iteration helps to resolve all the eigenvalues, so that when the stage is reached for shifting the origin towards a particular one we are likely to know it already to several significant figures.

Some typical results are summarized in Appendix A.

At the beginning of this paper I suggested the use of elementary transformations with interchanges in place of unitary transformations, since they are more economical. They do not preserve the normal or hermitian property of a matrix, but they could certainly be employed in algorithms similar to both those described. It may be that convergence cannot be proved in this case, but experience leads one to expect an exactly equivalent effect to that obtained by unitary transformations. This expectation was largely confirmed for the first method by my initial experimental work early in 1959, and I believe J. H. Wilkinson has also done some work which supports it.

Acknowledgements

I particularly wish to thank Mr. Christopher Strachey for his helpful advice and criticism during my work on this paper, and I am also grateful to Mr. J. H. Wilkinson, who read the paper at an early stage, for valuable suggestions.

References

FIKE, C. T. (1959). "Note on the Practical Computation of Proper Values," *J. Assoc. Comput. Mach.*, Vol. 6, p. 360.

GIVENS, W. (1958). "Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form," *J. Soc. Indust. Appl. Math.*, Vol. 6, p. 26.

RUTISHAUSER, H. (1958). "Solution of Eigenvalue Problems with the LR-transformation," *Nat. Bur. Standards Appl. Math. Ser.*, No. 49, p. 47.

STRACHEY, C., and FRANCIS, J. G. F. (1961). "The Reduction of a Matrix to Codiagonal Form by Eliminations," *The Computer Journal*, Vol. 4, p. 168.

WILKINSON, J. H. (1959). "Stability of the Reduction of a Matrix to Almost Triangular and Triangular Forms by Elementary Similarity Transformations," *J. Assoc. Comput. Mach.*, Vol. 6, p. 336.

WILKINSON, J. H. (1960). "Householder's Method for the Solution of the Algebraic Eigenproblem," *The Computer Journal*, Vol. 3, p. 23.

Appendix A

Practical Results

The Pegasus programs which employ the two methods we have described for applying the QR transformation have been used on a number of matrices of widely differing types. From our concluding remarks it is clear that, for real unsymmetric matrices, the second method (the method of double iterations) is of greater interest, and we shall therefore confine our discussion to results obtained in using it.

In the accompanying tables details are given relating to five typical, real, unsymmetric matrices which have the following main characteristics:

NAME	ORDER	EIGENVALUES
$\boldsymbol{A}$	10	10 real
$\boldsymbol{B}$	15	15 real
$\boldsymbol{C}$	19	13 real, 3 conjugate complex pairs
$\boldsymbol{D}$	24	12 conjugate complex pairs
$\boldsymbol{E}$	27	21 real, 7 occurring twice; 3 complex pairs, one occurring twice

All the elements of matrices $\boldsymbol{A}$ and $\boldsymbol{C}$ are printed while matrix $\boldsymbol{B}$ is an example given by Wilkinson (1959b). Matrix $\boldsymbol{D}$, of which we give only the elements in the last 12 columns, has a 12×12 zero submatrix in its top left-hand corner and $10^7 I_{12}$ in its bottom left-hand corner; thus

$$[\boldsymbol{d}_1, \boldsymbol{d}_2 \ldots, \boldsymbol{d}_{12}] = 10^7[\boldsymbol{e}_{13}, \boldsymbol{e}_{14}, \ldots, \boldsymbol{e}_{24}].$$

Many of the elements of matrix $\boldsymbol{E}$ are zero, and the other elements (ignoring their signs) take only 80 distinct values. In Tables 6 and 7 these values are numbered and the matrix is shown in symbolic form; a period represents zero and the number of the value of each other element is given, a bar over it indicating change of sign. This matrix may be thought to be far from typical, and it is in fact possible by simple, if laborious, operations to split it up into independent submatrices—two matrices of orders 4 and 5 each occurring twice, a matrix of order 8, and a single zero eigenvalue. However, this sort of matrix does seem to occur in practice and it may

be quite reasonable to expect an eigenvalue-finding program to cope with it—particularly when, as in this case, the repeated eigenvalues are relatively well determined.

The exact number of iterations required to find the eigenvalues of a matrix depends to some extent upon finer points of machine characteristics and program details than are usually given. These points include, for example, the number of significant digits in the data, the number of guard bits used, the word length of the machine, and the size of elements treated as zero when testing convergence. Another point in the present case concerns the criteria for choosing the origin shifts immediately after deflation. In the Pegasus program, before an iteration a copy is made of the elements that will be changed on the principle diagonal and on the two adjacent diagonals. This enables the roots of the last 2×2 submatrix, on which the shifts depend, to be compared in all circumstances with the roots of the same submatrix as it was before the previous iteration. On other machines the trouble of copying these elements may not be worth the increase in efficiency gained, and after deflation one may have to repeat one or both of the origin shifts used for the previous iteration.

The size of that part of the matrix involved in each iteration depends upon the values of p and q, as described in the paper, and upon the number of eigenvalues already found. These parameters have a considerable effect on the time taken in iterating, but different matrices follow quite different patterns. Thus, for example, in most cases the subdiagonal elements do not become zero except in the normal process of finding a pair of eigenvalues, and $q = 1$ throughout. However, this is far from true of matrix $\boldsymbol{E}$ which becomes split up to a great extent (not always or only on account of its repeated eigenvalues). When a matrix, such as $\boldsymbol{D}$, has all of its eigenvalues complex, then no two consecutive subdiagonal elements will both converge to zero, and we shall find that normally $p = q$ for all iterations. On the other hand, often $p > q$ for a matrix such as $\boldsymbol{B}$. The behaviour of this matrix, which is characteristic of its type, is shown in Table 1.

Another factor which will have an effect on the time taken by the program is whether or not the machine has built-in floating-point operations. There are about 27 multiplications and 24 additions at each stage of an iteration which are done with floating-point arithmetic. These take an appreciable proportion of the time on Pegasus, on which they have to be specially programmed, but this proportion decreases as the order of the matrix is increased.

Table 1

Iteration	1 2 3 4	5 6	7 8 9	10 11	12	13 14	15 16
Order – No. of rows involved (as $q-1$)	15 15 15 15	13 13	11 11 11	9 9	7	5 5	3 3
No. of columns involved	15 15 15 15	13 13	11 11 3	7 4	5	3 3	3 3
No. of roots found	2	2	2	2	2	2	3

Table 2

Matrix	*A*	*B*	*C*	*D*	*E*
Order	10	15	19	24	27
Significant bits in largest element (including guard bits)	25	27	24	36	26
Guard bits used	0	6	6	7	6
Maximum error in roots found as integer in least significant guard place	20	8	10	31	16
Number of (double) iterations	13	16	28	21	19
Time taken (in minutes and seconds) for (*a*) reduction to almost triangle (*b*) complete process	0.07 2.27	0.19 3.57	0.35 6.34	1.00 10.38	1.26 6.42

In Table 2 the number of iterations needed, and the time taken to find the eigenvalues of each of the five matrices, are set out with some of the other relevant details. In each case the convergence criterion for the subdiagonal elements was exactly zero. To appreciate the times given we should mention that Pegasus has an addition time of about 0·3 msec, and a multiplication time of about 2 msec. Floating-point addition in this program takes about 10 msec.

The matrices are given in Tables 3 to 7, and their eigenvalues in Tables 8 to 13. In the latter, the eigenvalues of $\boldsymbol{A}$, $\boldsymbol{B}$, etc., are given accurate to 2 or 3 decimal places; the almost triangular matrices formed are represented by a subscript T, e.g. $\boldsymbol{C}_T$, and where I know their accurate eigenvalues I give the figures that differ from those of the original matrices. The eigenvalues of $\boldsymbol{A}_F$, $\boldsymbol{B}_F$, etc., are the actual eigenvalues found, and, again, the figures that differ are given. Notice that the eigenvalues of $\boldsymbol{A}_F$ in Table 8 have no figures after the decimal point because no guard bits are used in their calculation. The frequencies of occurrence of the eigenvalues of matrix $\boldsymbol{E}$ are given in Table 12.

Table 3: Matrix *A*

−3360147	+12700259	−8759239	+13758458	−15928949	+8150598	+16332450	−38138100	+36250338	−54402238
+924120	+3654401	+1537060	+3516663	+12381154	−10155927	−4227114	+26300588	−22816599	+25294184
−2038193	−7472721	+7589804	−12745092	+28603110	−15956903	−19868285	+57290730	−49901355	+66325076
+1033571	+7357176	−268136	+7283295	+4231460	−4319692	+4901481	+8817744	−9691392	+15311072
+804793	+6518376	−4381893	+11555654	−17256997	+13237810	+16864383	−40702516	+40329048	−52764354
−956075	−4259899	+3856081	−7055870	+13560511	−7935016	−10897566	+30769288	−27049063	+41380960
+972197	+6411527	−2873830	+10588002	−6932121	+4015466	+10674457	−14958266	+13519476	−15710641
−694134	−3673931	+4396381	−6105317	+14598775	−6232239	−8829453	+26413007	−18396035	+20760525
+517619	+4042981	−4207104	+7115249	−13150298	+10029400	+10395106	−27067685	+24819163	−26414792
−165743	−697517	+675792	−1124599	+2363906	−2092270	−1758468	+5417694	−5875247	+11057995

Table 4: Matrix *C*

+327015	+96257	−103968	+95269	−80615	+53382	−27732	−41065	+48447	−59875	+57718	−36830	+22474	−5976	+13543	−16652	+16585	−10655	+4134
−83132	+449289	−65414	+56299	−42268	+36342	−14765	−40653	+23603	−32086	+38738	−17943	+16345	−5280	+6649	−9130	+11146	−5101	+3168
−76105	+32868	+289464	+93520	−72347	+61050	−25506	−32395	+20549	−26293	+67110	−29763	+29493	−5228	+5720	−15381	+19455	−8694	+5100
−70234	+30767	−112414	+440768	−29172	+33709	−9886	−23436	+21682	−50412	+36493	−6883	+21856	−3016	+5858	−13900	+10379	−2022	+5075
−65987	+30418	−96891	+26976	+327132	+51983	−23233	−15325	+23621	−39891	+33703	−26783	+34115	−2729	+6114	−11395	+9157	−7653	+6664
−57797	+27217	−86775	+23986	−83514	+419260	−10209	−10378	+24375	−32074	+31799	−28475	+19387	−172	+7378	−9994	+9390	−7865	+4377
−27469	+14136	−38065	+10992	−40202	−2512	+383159	−4281	+11538	−13864	+14772	−13669	+6819	−858	+3277	−3820	+4221	−3809	+1435
+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	−62500	+0	+0	+0	+0	+0
+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	−125000	+0	+0	+0	+0
+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	−125000	+0	+0	+0
+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	−125000	+0	+0
+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	−125000	+0
+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	+0	−62500
+35612	−38767	−30917	−20456	+7150	+29432	+8462	+5601	+1753	+1285	−2148	−1547	−298	+35989	−128	−814	−1796	−974	−578
+76790	+5937	−107362	−43307	+5457	+55603	+17404	+1892	+7727	+2762	−4595	−3402	−1463	−1327	+49619	−2333	−4361	−2376	−1578
−6231	+88668	−20226	−94854	−21612	+50196	+20195	−1027	+1037	+8375	−1909	−2850	−2154	−2296	−1886	+54348	−4586	−2718	−2090
−40789	+8165	+94165	+27752	−95181	+5267	+16257	−2437	−2731	−1989	+8364	+1146	−1120	−2479	−2842	−4627	+53627	−1551	−1585
−41822	−15755	+33334	+113998	+4824	−90149	+3664	−1687	−3128	−4387	+2491	+7754	+1383	−1679	−2261	−4037	−1933	+47346	−623
−19129	−9877	+15020	+25326	+47308	−40045	−32158	−321	−1210	−1801	+1100	+1507	+4787	−443	−743	−1317	−450	+60	+33209

Table 5: Columns 13–24 of Matrix *D*

−14474423	−131712	−3621621	+8007199	+18426	−114076	+212382	+88976	−366	+103	+467	−42
−4803085	−5293550	−3388545	+394367	+58003	+241732	−94628	−101040	+3403	+2201	+439	+22
−79183463	−1087801	−30132515	+47176735	−278455	−286649	+1016545	+342127	+2396	+3390	+3040	−179
+23445712	+248069	+5857590	−37703554	−15429483	+11901207	−5812416	−12686644	+904936	+529218	+73050	−3974
+20688429	+220404	+4800310	+46532072	−70759421	+34024177	+31823948	−67378645	+11801724	+7618614	+2666763	−32300
−149677	+31152	−26028	−5829119	+9140130	−4629746	−3909610	+8705095	−587244	−336347	−234635	−4674
+2968855	−2159	+595178	−22109673	+22986602	−10022617	−12839556	+2689524	−3942830	−2817666	−1328559	+20327
+8179800	−135373	+1388751	−15390858	−30987938	+17322457	+21998721	−16381606	+24185115	+14856046	+3071427	−153779
+143634	+5778	+15122	−3746947	+10973801	−1726005	−5597851	+4736726	−2940441	−1611434	−389974	+378597
+72302	+5339	+1324	−5538733	+13606028	−2206514	−7527955	+5234403	−2840061	−2446568	−563917	−59736
−132477	−6401	+18149	−3843105	+5690140	−1873325	−3998121	+1205198	−7440543	−6122070	−1446659	+922506
+53883	+1669	+4979	+187022	−364889	+198814	+146619	−32636	−6734	−3848	−1082	+93
−1067859	−6759	−251068	+340573	−11995	+4346	+7408	+419	+149	+118	+49	+0
−17122958	−1423731	−2430339	+11068870	−203631	+53516	+101283	+8985	+901	+762	+343	−47
−311287	+257536	−1309585	−489637	−14232	+9329	+9477	−1012	+430	+329	+132	+10
+281236	+145642	−518874	−886464	−4400	+5466	+976	−1779	+271	+201	+75	+8
−565278	−19481	−212371	+189742	−8901	+3418	+3888	−159	+101	+78	+31	+0
+116887	+29786	−258719	−2840441	+2098251	−922402	−1279659	−162817	+1143	+207	−549	−193
−1641111	−317722	+742783	−602814	+4016247	−1986793	−1835966	+308951	−21977	−14095	−4099	−261
+1000549	−74375	+621369	−762049	−1950846	+1215073	+142809	−907141	+46252	+28802	+9750	−140
+1292032	+55836	+7921	+2226784	−6701465	+3784748	+2378200	−884701	−129149	−73385	−19428	+1772
+1594328	+251390	−661256	+4064	−4346759	+2254295	+1651079	−685185	+45372	+27543	+7345	+144
+937028	+179326	−374696	+502643	−6944301	+3299703	+3610920	−61336	−1762	−734	−4059	+761
+573549	+19061	+31598	+1371633	−3371640	+1885101	+1250609	−390817	−67202	−38299	−10352	+907

Table 6: Matrix ∇E

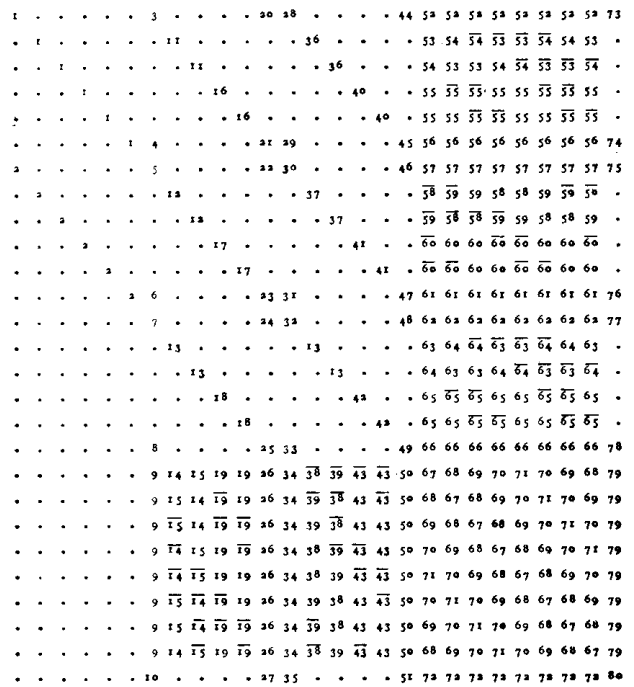

Table 7

1	−2874	17	−2226	33	−96343	49	−138432	65	+70976
2	+28740	18	−27142	34	−1053459	50	+405267	66	−109778
3	−3920	19	+44842	35	−1219423	51	−376838	67	−1350490
4	+789	20	+646	36	+3351	52	+7192	68	−987969
5	+20857	21	−535	37	−23613	53	+3306	69	−444425
6	−5600	22	−4587	38	+634536	54	+1369	70	−369932
7	−214716	23	−2353	39	+262834	55	+1473	71	−476456
8	+35630	24	+35182	40	+1686	56	−2386	72	−501656
9	+336803	25	−25040	41	−11885	57	−50688	73	+10582
10	+355584	26	−81781	42	−138241	58	+23298	74	+632
11	−1204	27	+6565	43	+315257	59	+9651	75	−74576
12	+1978	28	+10325	44	−2064	60	+10379	76	−4455
13	−62472	29	−2089	45	+1650	61	+16815	77	+580896
14	+135894	30	−72768	46	+14546	62	+393577	78	+34600
15	+56289	31	+14722	47	−11629	63	+171483	79	−729763
16	−563	32	+315693	48	−114534	64	+71031	80	−1404278

Appendix B

Programs like the two described in this paper—for finding the eigenvalues of a matrix by the QR transformation—contain a procedure for reducing the matrix to almost triangular form. It might seem more compatible with the techniques of the QR transformation to use either Givens' or Householder's method for this reduction, as these employ unitary transformations, but the method of eliminations which I have used has been shown to be very satisfactory and is more efficient. This method is described, for example, in papers by Wilkinson (1959b) and by Strachey and Francis (1961). In the appendix of the latter paper an algorithm is given which incorporates the reduction to almost lower triangular form of a real matrix. This can easily be transposed to suit our purpose for producing an almost upper triangular matrix, and can be modified in an obvious way for the complex case.

In the second place, the root-finding part of the program requires procedures for testing convergence, for finding the origin shifts, and for performing the iterations on the matrix which constitute the QR transformation.

Table 8

Eigenvalues of A	A_F
+407361.43	365
−1067060.23	057
+1357863.87	866
−1777648.52	652
+2461373.71	375
−5512964.66	965
+7013550.23	553
+15637089.47	108
+18107433.58	414
+26312963.12	957

Table 9

Eigenvalues of B	B_F
−605.47	.52
+3587.14	.19
+43990.81	.83
+73704.19	.23
+151487.87	.87
+173583.27	.17
+257611.17	.25
+308201.60	.50
+369921.48	.48
+517836.12	.12
+976578.96	.84
+1296443.15	.11
+1593256.12	.14
+4830173.88	.92
+6294127.73	.78

Table 10

Eigenvalues of C	C_T	C_F
+4242.56	.65	.64
+4468.94	.82	.83
+4055.57	.52	.50
+4977.32	.42	.44
+3068.90	.95	9.05
+1455I.56	.57	.55
+2065O.39	.26	.25
+2971Z.12	.06	.19
+37022.57	.78	.72
+47051.95	.93	.94
+49873.33	.32	.38
+55404.78	.72	.66
+3856 73.25	.18	.16
±34090.56i	.51	.52
+400323.32	.35	.33
±3661 3.50i	.53	.48
+360523.24	.31	.32
±56234.13i	.08	.09
+342105.38	.33	.38

Table 12

Eigenvalues of E	
+0	X1
+2.31	X2
−2860.95	X2
−2861.67	X2
−2864.88	X1
−2868.81	X1
−6147.93	X1
−6236.22	X2
−6556.82	X2
−6816.68	X1
−404792.48	X1
−459802.14	X2
−472652.86 ± 203617.32i	X1
−536166.56 ± 142661.02i	X2
−1342218.39	X2
−5277068.52	X1

Table 11

Eigenvalues of D	D_T	D_F
−11282.520	.524	.520
±145828.471i	.864	.711
+1.635	.671	.676
±1470349.782i	.756	.789
−388212.196	.216	.242
±1554206.150i	.154	.133
−12195.232	.235	.223
±2351590.145i	.150	.180
−80862.116	.143	.141
±3224211.811i	.863	.914
−975810.479	.484	.469
±6562160.465i	.489	.477
−210739.234	.244	.273
±7193238.178i	.184	.148
−63883.445	.422	.422
±9964946.178i	.199	.195
−84057.238	.235	.246
±13339174.124i	.070	.070
−668500.212	.225	.238
±22318797.232i	.236	.258
−341960.249	.228	.234
±43542600.370i	.386	.422
−1395902.215	.215	.207
±93506806.642i	.670	.648

Table 13

Eigenvalues of E_F
+0.00
+2.33
+2.27
−2861.83
−2861.55
−2860.98
−6236.08
−6236.47
−2860.97
−6556.72
−6556.86
−536166.56 ± 142660.97i
−536166.56 ± 142661.23i
−459802.14
−459802.13
−1342218.33
−1342218.36
−2868.77
−2864.86
−6147.98
−6816.66
−404792.47
−472652.87 ± 203617.50i
−5277068.50

Our principal interest is in these iterations and we give below two procedures for performing them, corresponding to the two methods described in the paper. For the first procedure the initial matrix is assumed to be complex and to have been reduced to almost triangular form so that its subdiagonal elements are real. (These elements can be made real during the reduction by a simple transformation which puts the modulus of the largest element in the subdiagonal position at each stage before the elimination in each column.) For the second procedure all the elements are real.

The elements of the matrix are stored in floating-point form for the two algorithms (though in the paper we have shown how fixed-point arithmetic and representation can be used). The elements can thus be of any size, but the truncation errors in them must be of the same order of magnitude and this must be attained if necessary by scaling the rows and columns initially. It should always be possible to take out a scaling factor and express the matrix in fixed-point form without any appreciable loss of significant figures.

The notation employed is the same informal publication-language version of Algol used by Strachey and Francis (1961). In the first algorithm it has proved convenient to extend Algol to allow the type declarators **complex** and **complex array.** Where the meaning may not be quite clear notes are given in the form of comments.

SINGLE COMPLEX QR ITERATION

```
Procedure  QR Iteration 1 (a) order (n) shift (ζ)
Value  n, ζ
Integer  n
Complex  ζ                                                    Comment  See note 1
Complex array  a[1 : n, 1 : n]                                Comment  See note 2
Begin  Real κ Complex w
       Real array ν[1 : n - 1]
       Complex array μ[1 : n]
       Integer i, j, q
       For i = n - 2, i - 1  while (i ⩾ 1) ∧ (abs a_{i-1,i} > δ)  do q = i
                                                              Comment  Find q.  See note 3
       For i = q step 1 until n do a_{i,i} = a_{i,i} - ζ      Comment  Shift origin
       For i = q step 1 until n - 1 do                        Comment  Reduce to triangle
         Begin κ = √(|a_{i,i}|^2 + a^2_{i+1,i})
               μ_i = a_{i,i}/κ
               ν_i = a_{i+1,i}/κ
               a_{i,i} = κ
               For j = i + 1 step 1 until n do                Comment  Row operation
                 Begin w = a_{i,j}
                       a_{i,j} = μ̄_i . a_{i,j} + ν_i . a_{i+1,j}
                       a_{i+1,j} = μ_i . a_{i+1,j} - ν_i . w
                 End
         End
       μ_n = a_{n,n}/|a_{n,n}|
       a_{n,n} = |a_{n,n}|                                    Comment  Triangle
       For j = q step 1 until n - 1 do                        Comment  Inverse operation
         Begin For i = q step 1 until j do                    Comment  Column operation
                 Begin w = a_{i,j}
                       a_{i,j} = μ_j . a_{i,j} + ν_j . a_{i,j-1}
                       a_{i,j+1} = μ̄_j . a_{i,j+1} - ν_j . w
                 End
               a_{j+1,j} = ν_j . a_{j+1,j-1}
               a_{j+1,j+1} = μ̄_{j+1} . a_{j+1,j-1}
         End
       For i = q step 1 until n do
         Begin a_{i,n} = μ_n . a_{i,n}
               a_{i,i} = a_{i,i} + ζ
End
                 End
```

Comment *Notes.*—(1) ζ is the origin shift, in general complex. Where we require the modulus of a complex number, say x, we write $|x|$ and we denote its complex conjugate value by $\bar{x}$.

(2) The almost triangular matrix has complex elements. However, the subdiagonal elements are real and where these are involved in the iteration real arithmetic is used.

(3) The quantity δ, which is a global variable, is the convergence criterion.

DOUBLE QR ITERATION

```
Procedure  QR Iteration 2  (a) order (n) shift (σ, ρ)
Value  n, σ, ρ
Integer  n
Real  σ, ρ
Array  a[1 : n, 1 : n]
Begin  Integer i, j, p, q
       Real α,η,κ
       Array γ[1 : 3]  ψ[1 : 2]
       For i = n − 2, i − 1 while (i ⩾ 1)                                  Comment  Find p
                          ∧ (abs a_{i+1,i} > δ)
                          ∧ {abs[a_{i+1,i} . a_{i+2,i+1}( abs (a_{i+1,i+1} − a_{i+2,i+2} − σ) − abs a_{i+3,i+2})/
                                (a_{i+1,i+1}(a_{i+1,i+1} − σ) + a_{i+1,i+2} . a_{i+2,i+1} + ρ)] > δ}
                      do p = i
       For i = p, i − 1 while (i ⩾ 1) ∧ (abs a_{i−1,i} > δ) do q = i       Comment  Find q
       For i = p step 1 until n − 1 do                                     Comment  Iterate
         Begin  If i = p then
                  Begin  γ1 = a_{p,p} (a_{p,p} − σ) + a_{p,p+1} . a_{p+1,p} + ρ
                         γ2 = a_{p+1,p}(a_{p,p} + a_{p+1,p+1} − σ)
                         γ3 = a_{p+1,p} . a_{p+2,p+1}
                         a_{p+2,p} = 0
                  End
                  Otherwise
                  Begin  γ1 = a_{i,i−1}
                         γ2 = a_{i+1,i−1}
                         γ3 = if i ⩽ n − 2 then a_{i+2,i−1} otherwise 0
                  End
                κ = if γ1 ⩾ 0 then √(γ1² + γ2² + γ3²) otherwise − √(γ1² + γ2² + γ3²)
                α = if κ ≠ 0 then γ1/κ + 1 otherwise 2                     Comment  See note
                ψ1 = if κ ≠ 0 then γ2/(γ1 + κ) otherwise 0
                ψ2 = if κ ≠ 0 then γ3/(γ1 + κ) otherwise 0
                If i ≠ q then a_{i,i−1} = (if i = p then − a_{i,i−1} otherwise − κ)
                For j = i step 1 until n do                                Comment  Row operation
                  Begin  η = α(a_{i,j} + ψ1 . a_{i+1,j} + [if i ⩽ n − 2 then ψ2 . a_{i+2,j}
                                                           otherwise 0])
                         a_{i,j} = a_{i,j} − η
                         a_{i+1,j} = a_{i+1,j} − ψ1 . η
                         If i ⩽ n − 2 then a_{i+2,j} = a_{i+2,j} − ψ2 . η
                  End
                For j = q step 1 until (if i ⩽ n − 2 then i + 2 otherwise n) do   Comment  Column operation
                  Begin  η = α(a_{j,i} + ψ1 . a_{j,i+1} + [if i ⩽ n − 2 then ψ2 . a_{j,i+2}
                                                           otherwise 0])
                         a_{j,i} = a_{j,i} − η
                         a_{j,i+1} = a_{j,i+1} − ψ1 . η
                         If i ⩽ n − 2 then a_{j,i+2} = a_{j,i+2} − ψ2 . η
                  End
                If i ⩽ n − 3 then
                  Begin  η = α . ψ2 . a_{i+3,i+2}
                         a_{i+3,i} = − η
                         a_{i+3,i+1} = − ψ1 . η
                         a_{i+3,i+2} = a_{i+3,i+2} − ψ2 . η
                  End
         End
End
```

Comment *Note.*—κ can only be zero when $i = n - 1$ because in all other cases $\gamma_3 \neq 0$ due to the subdiagonal elements being non-zero.