A recursive algorithm for determining the eigenvalues of a quindiagonal matrix

D. J. Evans

Department of Mathematics, Loughborough University of Technology, Loughborough, Leicestershire, LE11 3TU*

A recursive algorithm for the implicit derivation of the determinant of a quindiagonal matrix is derived in terms of its leading principal minors. Its form is more compact and simpler than that previously presented in the literature by Sweet (1969). Its envisaged use is for deriving the eigenvalues of quindiagonal matrices by Newton's or similar root finding methods.

The derived algorithm simplifies considerably for the case of symmetric matrices yielding a Sturmian sequence of polynomials from which the eigenvalues can be obtained by use of the well known bisection process.

(Received May 1973)

1. Introduction

Recent computational techniques for the solution of matrix eigenvalues have all emphasised the fact that the characteristic polynomial of a tridiagonal matrix can be obtained implicitly with great ease by computing (for a specified value of λ) a simple sequence of polynomials derived from its leading principal minors (Wilkinson, 1965). Since quindiagonal matrices occur frequently in boundary value problems involving fourth order derivatives, it seems pertinent, therefore, to investigate the structure of such a matrix to develop a similar sequence of polynomials which expresses the characteristic equation implicitly.

2. A recursive relation for determinant evaluation. The Unsymmetric case

It is well known that the required eigenvalues are given by the determinantal equation

$$\det (C - \lambda I) = 0 \ , \qquad (2.1)$$

or in full matrix notation,

$$\det \begin{vmatrix} c_1-\lambda, & d_1, & e_1 & & & & \\ b_2, & c_2-\lambda, & d_2, & e_2 & & 0 & \\ a_3, & b_3, & c_3-\lambda, & d_3, & e_3 & & \\ & a_4, & b_4, & c_4-\lambda, & d_4, & e_4 & \\ & & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot & \cdot \\ & & & & \cdot & \cdot & e_{n-2} \\ & 0 & & a_{n-1} & b_{n-1} & c_{n-1}-\lambda, & d_{n-1} \\ & & & & a_n, & b_n, & c_n-\lambda \end{vmatrix} = 0$$

A simple Laplace expansion of the leading principal minors of small order can be shown to give the following sequence of polynomials $P_i(\lambda) \equiv P_i$, $i = 0, 1, 2$ and 3.

$$\begin{aligned} P_0 &= 1, \\ P_1 &= (c_1 - \lambda)P_0, \\ P_2 &= (c_2 - \lambda)P_1 - d_1 b_2 P_0, \qquad (2.2) \\ P_3 &= (c_3 - \lambda)P_2 - d_2 b_3 P_1 - a_3 e_1 (c_2 - \lambda) + a_3 d_1 d_2 + b_3 b_2 e_1. \end{aligned}$$

The evaluation of P_4 can be obtained in terms of P_3 and previous members of the set of polynomials P_i by bordering it horizontally with the matrix row elements 0, a_4, b_4, $(c_4 - \lambda)$ and vertically with the matrix column elements 0, e_2, d_3, $(c_4 - \lambda)$. (Burnside and Panton, 1904).

When P_4 is expanded all the terms which contain $(c_4 - \lambda)$ are included in the term $(c_4 - \lambda)P_3$. In addition to this, the expansion will consist of the product of every other element of the fourth column by every other element of the fourth row, every such product being multiplied by a certain factor. Thus, if the cofactors of $(c_3 - \lambda)$, $(c_2 - \lambda)$, d_2 and b_3 in the expansion of P_3 by C_3, C_2, D_2 and B_3, then the required factor which multiplies any product, i.e. $a_4 d_3$ in the expansion of P_4 is the same as the factor which multiplies $b_3(c_4 - \lambda)$ with the sign changed.

A simple rule by which the factor of any such product can be found is obtained by finding the fourth element which completes the rectangle formed by the leading term $(c_4 - \lambda)$ and the two elements in the fourth row and column which enter into this product. Substituting the cofactor with a negative sign for the constituent of P_3 so found, yields the required factor in each case. Thus, we obtain for P_4 the expression,

$$P_4 = (c_4 - \lambda)P_3 - b_4 d_3 C_3 - a_4 e_2 C_2 + a_4 d_3 B_3 + b_4 e_2 D_2 \qquad (2.3)$$

where C_3, C_2, B_3 and D_2 are defined as

$$C_3 = (-1)^{3+3} \begin{vmatrix} c_1 - \lambda, & d_1 \\ b_2, & c_2 - \lambda \end{vmatrix} = P_2 \ ,$$

$$C_2 = (-1)^{2+2} \begin{vmatrix} c_1 - \lambda, & e_1 \\ a_3, & c_3 - \lambda \end{vmatrix} = (c_3 - \lambda)P_1 - a_3 e_1 P_0 \ ,$$

$$B_3 = (-1)^{2+3} \begin{vmatrix} c_1 - \lambda, & e_1 \\ b_2, & d_2 \end{vmatrix} = -(d_2 P_1 - e_1 b_2) \ ,$$

$$D_2 = (-1)^{2+3} \begin{vmatrix} c_1 - \lambda, & d_1 \\ a_3, & b_3 \end{vmatrix} = -(b_3 P_1 - a_3 d_1) \ .$$

Thus, the expression for P_4 can be written as

$$\begin{aligned} P_4 = (c_4 - \lambda)P_3 - b_4 d_3 P_2 - a_4 e_2[(c_3 - \lambda)P_1 - a_3 e_1 P_1] \\ + a_4 d_3 (d_2 P_1 - e_1 b_2) + b_4 e_2 (b_3 P_1 - a_3 d_1) \ . \qquad (2.4) \end{aligned}$$

A similar operation of row and column bordering yields the following expression for P_5,

$$\begin{aligned} P_5 = (c_5 - \lambda)P_4 - b_5 d_4 P_3 - a_5 e_3[(c_4 - \lambda)P_2 - a_4 e_2 P_2] \\ + a_5 d_4[d_3 P_2 - e_2(b_3 P_1 - a_3 d_1)] \\ + b_5 e_3[b_4 P_2 - a_4(d_2 P_1 - b_2 e_1)] \ . \qquad (2.5) \end{aligned}$$

A comparison of the equations (2.2), (2.4) and (2.5) enables one to derive the general form for the characteristic equation for $i = n$. Thus, the value of $\det (C - \lambda I)$ can be easily obtained from the recursion formula by computing the sequence of polynomials $P_i(\lambda) \equiv P_i$, $R_i(\lambda) \equiv R_i$, $S_i(\lambda) \equiv S_i$ for $i = 0(1)n$ such that

$$\begin{aligned} P_0 &= 1 \qquad &; R_0 = 0;\ S_0 = 0 \\ P_1 &= (c_1 - \lambda)P_0 \qquad &; R_1 = 1;\ S_1 = 1 \\ P_2 &= (c_2 - \lambda)P_1 - b_2 d_1 P_0; & \\ R_2 &= d_1 \qquad &; S_2 = b_2 \end{aligned}$$

*Now Department of Computer Studies, Loughborough University of Technology, Loughborough, Leics. LE11 3TU.

$$\begin{aligned}
P_3 &= (c_3 - \lambda)P_2 - b_3d_2P_1 - a_3e_1(c_2 - \lambda)\\
&\quad + a_3d_2R_2 + b_3e_1S_2;\\
R_3 &= d_2P_1 - e_1S_2 \qquad ; \; S_3 = b_3P_1 - a_3R_2\\
P_4 &= (c_4 - \lambda)P_3 - b_4d_3P_2 - a_4e_2[(c_3 - \lambda)P_1 - a_3e_1P_0]\\
&\quad + a_4d_3R_3 + b_4e_2S_3;\\
R_4 &= d_3P_2 - e_2S_3 \qquad ; \; S_4 = b_4P_2 - a_4R_3 \ .
\end{aligned}$$

for $i = 5(1)n - 1$,

$$\begin{aligned}
P_i &= (c_i - \lambda)P_{i-1} - b_id_{i-1}P_{i-2}\\
&\qquad - a_ie_{i-2}\,[(c_{i-1} - \lambda)P_{i-3} - a_{i-1}e_{i-3}P_{i-4}]\\
&\qquad\qquad + a_id_{i-1}R_{i-1} + b_ie_{i-2}S_{i-1};\\
R_i &= d_{i-1}P_{i-2} - e_{i-2}S_{i-1};\; S_i = b_iP_{i-2} - a_iR_{i-1}
\end{aligned}$$

and finally for $i = n$,

$$\begin{aligned}
\det(C - \lambda I) = P_n &= (c_n - \lambda)P_{n-1} - b_nd_{n-1}P_{n-2}\\
&\quad - a_ne_{n-2}[(c_{n-1} - \lambda)P_{n-3} - a_{n-1}e_{n-3}P_{n-4}]\\
&\quad + a_nd_{n-1}R_{n-1} + b_ne_{n-2}S_{n-1} \ . \qquad (2.6)
\end{aligned}$$

Similarly, the evaluation of $P_n'(\lambda) = \dfrac{d}{d\lambda}[\det(C - \lambda I)]$ can be carried out by the formulae listed below:

$$\begin{aligned}
P_0' &= 0 \qquad ; \; R_0' = 0;\; S_0' = 0\\
P_1' &= -1 \qquad ; \; R_1' = 0;\; S_1' = 0\\
P_2' &= (c_2 - \lambda)P_1' - b_2d_1P_0' - P_1;\; R_2' = 0;\; S_2' = 0\\
P_3' &= (c_3 - \lambda)P_2' - b_3d_2P_1' - P_2 + a_3e_1;\\
R_3' &= d_2P_1';\; S_3' = b_3P_1'\\
P_4' &= (c_4 - \lambda)P_3' - b_4d_3P_2' + a_4e_2P_1\\
&\quad - a_4e_2\,[(e_3 - \lambda)P_1' - a_3e_1P_0'] + a_4d_3R_3'\\
&\quad + b_4e_2S_3' - P_3;\\
R_4' &= d_3P_2' - e_2S_3';\; S_4' = b_4P_2' - a_3R_3'
\end{aligned}$$

for $i = 5(1)n - 1$;

$$\begin{aligned}
P_i' &= (c_i - \lambda)P_{i-1}' - b_id_{i-1}P_{i-2}' - P_{i-1} + a_ie_{i-2}P_{i-3}\\
&\quad - a_ie_{i-2}[(c_{i-1} - \lambda)P_{i-3}' - a_{i-1}e_{i-3}P_{i-4}']\\
&\quad + a_id_{i-1}R_{i-1}' + b_ie_{i-2}S_{i-1}';\\
R_i' &= d_{i-1}P_{i-2}' - e_{i-2}S_{i-1}';\; S_i' = b_iP_{i-2}' - a_iR_{i-1}' \qquad (2.7)
\end{aligned}$$

with $i = n$,

$$\begin{aligned}
\frac{d}{d\lambda}[\det(C - \lambda I)] = P_n' &= (c_n - \lambda)P_{n-1}' - b_nd_{n-1}P_{n-2}'\\
&\quad - P_{n-1} + a_ne_{n-2}P_{n-3}\\
&\quad - a_ne_{n-2}[(c_{n-1} - \lambda)P_{n-3}' - a_{n-1}e_{n-3}P_{n-4}']\\
&\quad + a_nd_{n-1}R_{n-1}' + b_ne_{n-2}S_{n-1}' \ .
\end{aligned}$$

Finally, the two recursive formulae (2.6) and (2.7) are used in Newton's iterative method for finding an eigenvalue of the matrix C, in the form

$$\lambda_{k+1} = \lambda_k - [P_n(\lambda_k)/P_n'(\lambda_k)],\; k \geqslant 0 \ . \qquad (2.8)$$

where λ_0 is an initial estimate.

If the root is simple, then this method has quadratic convergence, that is

$$|\lambda_{k+1} - \lambda| \propto |\lambda_k - \lambda|^2 \ . \qquad (2.9)$$

However, if the root to which the process is converging is multiple then (2.9) has linear convergence. It is possible to modify the formula to take account of multiplicity of roots if we know the order of multiplicity of the root. In general, we will not have this information available.

Alternatively, having found λ_1 by Newton's method, we may switch to the more efficient Secant iterative method,

$$\lambda_{k+1} = \lambda_k - (\lambda_k - \lambda_{k-1})P_n(\lambda_k)/[P_n(\lambda_k) - P_n(\lambda_{k-1})],\quad k \geqslant 1 \ . \qquad (2.10)$$

The iteration is continued until satisfactory convergence to the eigenvalue λ is obtained, i.e. until $|(\lambda_{k+1} - \lambda_k)/\lambda_k| < \varepsilon$, where ε is a small specified tolerance.

Having computed one or more of the eigenvalues by any of the above techniques, it is desirable to ensure that in further searches for eigenvalues, we do not redetermine those already found. Thus, we require some technique of suppressing the known eigenvalues. The recommended technique is as follows. If approximations $\lambda^{(1)}, \ldots, \lambda^{(s)}$ to eigenvalues are known, then instead of iterating with $P_n(\lambda)$ use

$$G_n(\lambda) = P_n(\lambda) / \prod_{i=1}^{s} (\lambda - \lambda^{(i)}) \ . \qquad (2.11)$$

Note that $G_n(\lambda)$ is again implicitly defined through the implicit determination of $P_n(\lambda)$ and the derivative of $G_n(\lambda)$ for use in Newton's method is given by

$$G_n'(\lambda) = G_n(\lambda)\left\{P_n'(\lambda)/P_n(\lambda) - \sum_{i=1}^{s}(\lambda - \lambda^{(i)})^{-1}\right\} \ , \qquad (2.12)$$

from which an expression for $G_n'(\lambda)/G_n(\lambda)$ can be obtained.

3. The symmetric case

When the quindiagonal matrix C is symmetric, i.e.,

$$d_i = b_{i+1},\; i = (1)n - 1 \text{ and } e_j = a_{j+2},\; j = 1(1)n - 2,$$

the recursive sequence (2.2) and (2.3) become,

$$\begin{aligned}
P_0 &= 1\\
P_1 &= (c_1 - \lambda)P_0,\\
P_2 &= (c_2 - \lambda)P_1 - b_2^2P_0, \qquad (3.1)\\
P_3 &= (c_3 - \lambda)P_2 - b_3^2P_1 - a_3^2(c_2 - \lambda) + 2a_3b_2b_3 \ ,
\end{aligned}$$

and

$$\begin{aligned}
P_4 &= (c_4 - \lambda)P_3 - b_4^2P_2 - a_4^2\{(c_3 - \lambda)P_1 - a_3^2P_0\}\\
&\qquad + 2a_4b_4\{b_3P_1 - b_2a_3P_0\} \ .
\end{aligned}$$

Further applications of the bordering technique developed in Section 2, produces the recursive sequence P_i for $i = 5, 6, \ldots$ in the simpler form

$$\begin{aligned}
P_i &= (c_i - \lambda)P_{i-1} - b_i^2P_{i-2} - a_i^2\{(c_{i-1} - \lambda)P_{i-3} - a_{i-1}^2P_{i-4}\}\\
&\quad + 2a_ib_i\{b_{i-1}P_{i-3} - b_{i-2}a_{i-1}P_{i-4}\\
&\qquad + b_{i-3}a_{i-1}a_{i-2}P_{i-5} - \ldots\}
\end{aligned}$$

and finally in a more compact notation, for $i = n$,

$$\begin{aligned}
P_n &= (c_n - \lambda)P_{n-1} - b_n^2P_{n-2}\\
&\quad - a_n^2\{(c_{n-1} - \lambda)P_{n-3} - a_{n-1}^2P_{n-4}\}\\
&\quad + 2\sum_{j=1}^{n-2}(-1)^{j+1}b_nb_{n-j}\Big[\prod_{r=n-j+1}^{n}a_r\Big]P_{n-j+2} \ . \qquad (3.2)
\end{aligned}$$

Now since the polynomials $P_0, P_1, P_2, \ldots, P_{n-1}, P_n$ form a sequence consisting of the leading principal minors of $|C - \lambda I|$, where C is a symmetric quindiagonal matrix, then they form a properly signed interleaved sequence of polynomials (i.e. all $P_k(\lambda) > 0$ for a sufficiently large value of λ either positive or negative and the zeros of $P_k(\lambda)$ strictly separate those of $P_{k+1}(\lambda)$) and thus with the aid of the separation theorem (Wilkinson, 1965), it can be shown that the sequence of polynomials $P_0, P_1, \ldots, P_n$ form a Sturm sequence of polynomials in the interval $(-\infty, +\infty)$.

The fundamental property of such polynomials facilitate the calculation of the roots by the process of bisection, i.e. the number of disagreements in the sign $s(\lambda)$ in the sequence P_i, $i = 0, 1, 2, \ldots, n$ being equal to the number of roots of $P_n(\lambda)$ smaller than λ.

From Gerschgorins' theorem, the eigenvalues are all contained in the union of the n intervals

$$c_i \pm (|b_i| + |b_{i+1}| + |a_i| + |a_{i+2}|),\quad i = 1, 2, \ldots, n$$

with

$$b_1 = b_{n+1} = a_1 = a_2 = a_{n+1} = a_{n+2} = 0 \ .$$

Hence the expression

$$\begin{matrix}\max\\ \min\end{matrix}\left\{c_i \pm (|b_i| + |a_i| + |b_{i+1}| + |a_{i+2}|)\right\} \ , \qquad (3.3)$$

can be input to the bisection process as initial upper and lower bounds for the eigenvalues.

To carry out the above algorithm in floating point arithmetic without the fear of underflow and overflow occurring, we follow a similar procedure to Barth, *et al.* (1967) and replace the sequence of polynomials $P_i(\lambda)$ by the sequence $p_i(\lambda)$ defined by

$$p_i(\lambda) = P_i(\lambda)/P_{i-1}(\lambda), \ (i = 1, 2, \ldots, n) \ . \tag{3.4}$$

The polynomials $p_i(\lambda)$ can be shown with a little analysis to satisfy the relationships,

$$\begin{aligned}
p_0 &= 1, \\
p_1 &= (c_1 - \lambda), \\
p_2 &= (c_2 - \lambda) - b_2^2/p_1, \\
p_3 &= (c_3 - \lambda) - b_3^2/p_2 - a_3^2(c_2 - \lambda)/p_2p_1 + 2a_3b_2b_3/p_2p_1,
\end{aligned}$$

$$\begin{aligned}
p_n = (c_n - \lambda) - b_n^2/p_{n-1} - a_n^2\{(c_{n-1} - \lambda)/p_{n-1}p_{n-2} \\
- a_{n-1}^2/p_{n-1}p_{n-2}p_{n-3}\} + \\
2\sum_{j=1}^{n-2}(-1)^{j+1}b_nb_{n-j}\left[(a_n/p_{n-j-1}p_{n-j})\prod_{r=n-j+1}^{n-1}(a_r/p_r)\right] .
\end{aligned} \tag{3.5}$$

With the use of the modified sequence of polynomials p_i, $i = 0, 1, 2, \ldots, n$ in (3.5), we find that the number of negative p_i now gives $s(\lambda)$, the number of eigenvalues smaller than λ.

The calculation of the products p_1p_2, $p_1p_2p_3$, etc in the sequence (3.5) also requires careful consideration when λ is close to an eigenvalue to ensure numerical stability.

4. Numerical results

The recursive algorithmic process given by the relationship (2.6) and (2.7) was checked for validity by obtaining the eigenvalues of the quindiagonal matrix

$$S = \begin{bmatrix}
a^2 + bc, & 2ab, & b^2, & & 0 \\
2ca, & a^2 + 2bc, & 2ab, & b^2, & \\
c^2, & 2ca, & a^2 + 2bc, & \ddots & b^2 \\
 & \ddots & 2ca, & a^2 + 2bc, & 2ab, \\
 & 0 & c^2, & 2ca, & a^2 + bc
\end{bmatrix} = J^2$$

where $J = \text{tridiag}\,(c, a, b)$. The eigenvalues of J are known to be

$$\lambda s = a + 2\sqrt{bc}\cos\{s\pi/(N + 1)\}, \ s = 1, 2, \ldots, N \tag{4.1}$$

An ALGOL program using equations (2.8) and (2.10) was completed for the Loughborough University ICL 1904A computer and a starting approximation of

$$\lambda_0 = [a + 2\sqrt{bc}\cos(s\pi/N)]^2$$

was used in the iteration process. The results obtained for the matrix $S = J^2$ where $J = \text{tridiag}\,(1, 2, 2)$ of order 10 correct to 7 significant places in decreasing order were as follows:

0·22220438,2; 0·19179357,2; 0·14839644,2; 0·10080441,2; 0·57721367,1; 0·25519195,1; 0·68067270,0; 0·50959032,0; 0·14396281,0; 0·21837196, −1;

which agree exactly with the theoretical results obtained from using (4.1).

Similarly, an ALGOL procedure based on the program given by Barth, *et al.* (1967) which computes the eigenvalues of the (10 × 10) test matrix, i.e.

$$\begin{bmatrix}
5 & -4 & 1 & & & & \\
-4 & 6 & -4 & 1 & & 0 & \\
1 & -4 & 7 & -4 & 1 & & \\
 & \ddots & \ddots & \ddots & \ddots & \ddots & \\
 & & 1 & -4 & 12 & -4 & 1 \\
 & 0 & & 1 & -4 & 13 & -4 \\
 & & & & 1 & -4 & 14
\end{bmatrix},$$

by the method of bisection using the Sturm sequence of polynomials p_i, $i = 1, 2, \ldots, n$ given by (3.6) yields the following results correct to 7 significant figures

2·0588891,1; 1·7336869,1; 1·4616481,1; 1·1943311,1; 9·4729464,0; 7·5412116,0; 5·9764481,0; 4·3530204,0; 2.5718218,0; 5·9900089, −1.

Both ALGOL programs are presented in section 5.

5. ALGOL programs

```
procedure quindiageigen (a, b, cc, d, e, lambda1, n, eps);
value n;
integer n;
array a, b, cc, d, e;
real lambda1, eps;
comment Procedure determines by the Newton iteration method
   the eigenvalue of a pentadiagonal matrix where the array cc
   denotes the diagonal, b and d the lower and upper sub-diagonal,
   and a and e the lower and upper sub sub-diagonal elements.
   Iteration is continued until the eigenvalue lambda1 is obtained
   to an accuracy specified by eps, m being the iteration count.
begin
integer i, m;
real lambda2;
array c, p, pd, r, s[0: n];
13: for i := 1 step 1 until n do
begin
  c[i] := cc[i] − lambda1;
end;

m := m + 1;
p[0] := 1; r[1] := s[1] := 1; p[1] := c[1];
r[2] := d[1]; s[2] := b[2];
p[2] := c[2] × p[1] − b[2] × d[1] × p[0];
r[3] := d[2] × p[1] − e[1] × s[2];
s[3] := b[3] × p[1] − a[3] × r[2];
p[3] := c[3] × p[2] − b[3] × d[2] × p[1] − a[3] × e[1]
     × c[2] + a[3] × d[2] × r[2] + b[3] × e[1] × s[2];
for i := 4 step 1 until n do
begin
  r[i] := d[i − 1] × p[i − 2] − e[i − 2] × s[i − 1];
  s[i] := b[i] × p[i − 2] − a[i] × r[i − 1];
  p[i] := c[i] × p[i − 1] − b[i] × d[i − 1] × p[i − 2]
    −a[i] × e[i − 2] × (c[i − 1] × p[i − 3] − a[i − 1]
    ×e[i − 3] × p[i − 4]) + a[i] × d[i − 1] × r[i − 1]
    +b[i] × e[i − 2] × s[i − 1];
end;

pd[0] := r[1] := s[1] := 0;
pd[1] := −1; r[2] := s[2] := 0;
pd[2] := c[2] × pd[1] − b[2] × d[1] × pd[0] − p[1];
r[3] := d[2] × pd[1]; s[3] := b[3] × pd[1];
pd[3] := c[3] × pd[2] − b[3] × d[2] × pd[1] − p[2] +
    a[3] × e[1];
for i := 4 step 1 until n do
begin
  r[i] := d[i − 1] × pd[i − 2] − e[i − 2] × s[i − 1]:
  s[i] := b[i] × pd[i − 2] − a[i] × r[i − 1];
  pd[i] := c[i] × pd[i − 1] − b[i] × d[i − 1] ×
    pd[i − 2] − p[i − 1] + a[i] × e[i − 2] ×
    p[i − 3] − a[i] × e[i − 2] × (c[i − 1] × pd[i − 3] −
    a[i − 1] × e[i − 3] × pd[i − 4]) + a[i] × d[i − 1] ×
    r[i − 1] + b[i] × e[i − 2] × s[i − 1];
end;

lambda2 := lambda1 − p[n]/pd[n];
if abs((lambda2 − lambda1)/lambda1) < eps then goto 12;
lambda1 := lambda2;
goto 13;
12: end of quindiageigen;
procedure quindibisect (c, b, d, dd, beta, n, m1, m2, eps1, relfeh)
    res: (eps2, z, x);
value n, m1, m2, eps1, relfeh;
real eps1, eps2, relfeh; integer n, m1, m2, z;
```

```
array c, b, d, dd, x, beta;
comment  c is the diagonal, b the sub-diagonal, d the sub sub-
  diagonal, beta the squared sub-diagonal and dd the squared sub
  sub-diagonal of a symmetric quindiagonal matrix of order n.
  Input to vectors b[i], beta[i], d[i] should begin with i = 2,
  2, 3 and 3 respectively. The value of relfeh is machine dependent
  and is the precision of the arithmetic used, i.e., for a t-digit
  binary mantissa relfeh is of the order 2↑-t.
  The eigenvalues lambda[m1], . . ., lambda[m2], where m2 is
  not less than m1 and lambda[i + 1] is not less than lambda[i],
  are calculated by the method of bisection and stored in the
  vector x. Bisection is continued until the upper and lower
  bounds for an eigenvalue differ by less than eps1, unless at some
  earlier stage, the upper and lower bounds differ only in the
  least significant digits. eps2 gives an extreme upper bound for
  the error in any eigenvalue, but for certain types of matrices
  the small eigenvalues are determined to a very much higher
  accuracy. In this case, eps1 should be set equal to the error
  to be tolerated in the smallest eigenvalue. It must not be set
  equal to zero;
begin real h, xmin, xmax; integer i;
  comment Calculation of xmin, xmax;
  d[1] := d[2] := 0;
  dd[1] := dd[2] := 0;
  beta[1] := b[1] := 0;
  xmin := c[n] - abs(b[n]) - abs(d[n]);
  xmax := c[n] + abs(b[n]) + abs(d[n]);
    h := abs(b[n - 1]) + abs(d[n - 1]) + abs(d[n]);
  if c[n - 1] + h > xmax then xmax := c[n - 1] + h;
  if c[n - 1] - h < xmin then xmin := c[n - 1] - h;
  for i := n - 2 step -1 until 1 do
  begin
    h := abs(b[i]) + abs(d[i]) + abs(b[i + 1]) +
      abs(d[i + 2]);
    if c[i] + h > xmax then xmax := c[i] + h;
    if c[i] - h < xmin then xmin := c[i] - h;
  end i;
  eps2 := relfehx(if xmin + xmax > 0 then xmax else -
    xmin);
  if eps1 < 0 then eps1 := eps2;
  eps2 := 0·5 × eps1 + 7 × eps2;
  comment Inner block;
  begin integer a, k; real xl, xu, xo; array wu[m1: m2],
    p[1: n];
    xo := xmax;
    for i := m1 step 1 until m2 do
    begin x[i] := xmax; wu[i] := xmin;
    end i;
    z := 0;
    comment Loop for the kth eigenvalue;
    for k := m2 step -1 until m1 do
    begin xu := xmin;
      for i := k step -1 until m1 do
      begin if xu < wu[i] then
        begin xu := wu[i]; goto contin
        end
      end i;
      contin: if xo > x[k] then xo := x[k];
        for x1 := (xu + xo)/2 while xo - xu > 2 × relfeh
            × (abs(xu) + abs(xo)) + eps1 do
        begin integer r, j;
          array p[0: n];
          real prod, prod2, sum, prod p;
          z := z + 1
          comment sturm sequence;
          a := 0; p[0] = 0;
          p[1] := c[1] - x1;
          if p[1] < 0 then a := a + 1;
          p[2] := (c[2] - x1) - beta[2]/(if p[1] ≠ 0
              then p[1] else relfeh);
          if p[2] < 0 then a := a + 1;
          p[3] := (c[3] - x1) - beta[3]/(if p[2] ≠ 0
              then p[2] else relfeh)
            -dd[3] × (c[2] - x1)/(if (p[2] × p[1]) ≠ 0
              then (p[2] × p[1]) else relfeh)
            +2 × d[3] × b[2] × b[3]/(if (p[2] × p[1]) ≠ 0
              then (p[2] × p[1]) else relfeh);
          if p[3] < 0 then a := a + 1; prodp = p[1] × p[2];
          for i := 4 step 1 until n do
          begin
            sum := 0;
            prod2:= 1;
            prodp := prodp × p[i - 1];
            for j := i - 2 step - 1 until 1 do
            begin
              prod := 1;
              For r: = i - j + 1 step 1 until i do prod:=
              prod × d[r];
              prod 2: = prod2 × p[i - j - 2];
              sum := sum + 2 × ((-1)↑(j + 1)) ×
                b[i - j] × prod × prod2;
            end j;
            p[i] := (c[i] - x1) - beta[i]/(if p[i - 1] ≠ 0
                  then p[i - 1] else relfeh) - dd[i]
                  × ((c[i - 1] - x1)/(if (p[i - 1]
                  × p[i - 2]) ≠ 0 then (p[i - 1]
                  × p[i - 2]) else relfeh - dd[i - 1]/
                  if (p[i - 1] × p[i - 2] × p[i - 3]) ≠ 0
                  × then (p[i - 1] × p[i - 2] ×
                  × p[i - 3]) else relfeh)) + sum—
                 if prodp ne 0 then prodp else relfeh);
           if p[i] < 0 then a := a + 1;
          end i;
          if a < k then
          begin if a < m1 then xu := wu[m1] := x1
          else
          begin xu := wu[a + 1] := x1;
          if x[a] > x1 then x[a] := x1;
          end;
          end;
          else xo := x1;
        end x1;
        x[k] := (xo + xu)/2;
      end k;
    end inner block;
  end quindibisect;
```

Acknowledgements

The author is indebted to Lorna A. Chester and R. G. Turner for programming assistance.

References

Barth, W., Martin, R. S., and Wilkinson, J. H. (1967). Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection. *Num. Math.*, Vol. 9, pp. 386-393.

Burnside, W. S., and Panton, A. W. (1904). *Theory of Equations*, Vol. 2, p. 20, Dublin University Press.

Sweet, Roland A. (1969). A recurrence relation for the Determinant of a Penta-diagonal Matrix. *CACM*, Vol. 12, pp. 330-332.

Wilkinson, J. H. (1965). *The algebraic eigenvalue problem*, Oxford University Press.