On finding the eigenvalues of real symmetric tridiagonal matrices

By A. J. Fox and F. A. Johnson*

The speed of the Sturm sequence algorithm for determining the eigenvalues of a tridiagonal matrix is shown to be much enhanced when used in conjunction with the LL^T or QR method. Comparative speeds are provided for programs based either on the simple Sturm sequence approach or on one of the two composite techniques. The LL^T program, which was found to be significantly faster than the other two, is described in detail.

1. Introduction

A general real symmetric or a complex hermitian matrix can be transformed into symmetric or hermitian tridiagonal matrices which have the same eigenvalues (Wilkinson, 1960). The hermitian tridiagonal matrix can be further transformed to a real symmetric tridiagonal matrix with the same eigenvalues by replacing the off-diagonal complex elements by their moduli. Thus an important class of practical eigenvalue problems can be reduced to the simpler problem of finding the eigenvalues of a real symmetric tridiagonal matrix.

In the following let A be the real symmetric tridiagonal matrix with diagonal elements $a_1, \dots a_n$, off-diagonal elements $b_2, \dots b_n$ and eigenvalues $\lambda_1, \dots \lambda_n$. We shall assume that A has been normalized so that $1 > \lambda_i > -1$ for all i: this can be readily accomplished by dividing A by the maximum value of M_i where M_i is the sum of the moduli of all the elements in the ith row.

It is well known that if all b_i are non-zero the eigenvalues are all different (e.g. see Ortega, 1960). Clearly if one or more of the b_i are zero (in practice if $|b_i| < \epsilon_1$ where ϵ_1 is a suitably small positive quantity) the matrix A can be factorized into the direct sum of two or more smaller matrices. This reduces the computational labour, since the total amount of work required to find the eigenvalues of two small matrices is less than that required for a single large matrix. This feature is of considerable importance in improving the convergence of any method for finding the eigenvalues, and in particular for the QR and LL^T methods described below. It is also an advantage in the eigenvector problem if one uses Wilkinson's method (1958) since one can find the eigenvectors of each submatrix separately.

In the following we shall assume that this *primary* factorization (as well as the normalization) has been carried out and thus the eigenvalues are all different. For convenience we shall label them λ_1 to λ_n so that

$$1 > \lambda_1 > \lambda_2 \dots \lambda_n > -1.$$

It was the purpose of the work reported in this paper to determine which of the currently available methods provided the fastest means of determining the eigenvalues of a tridiagonal matrix. A detailed comparison was undertaken of programs based on the Sturm sequence, LL^T and QR techniques, and the results of this comparison are presented. The program based on the LL^T approach proved to be the fastest and so its complete algorithm is presented. Together with the accompanying notes, this algorithm indicates the detailed strategy employed for choosing accelerating origin shifts. Also it shows how both the value of the characteristic polynomial and a partial Sturm sequence may be obtained from the LL^T algorithm itself.

2. Sturm sequence method

The usual method of finding the eigenvalues of a tridiagonal matrix A is that of calculating the Sturm sequence (Ortega, 1960). The number of sign agreements in the sequence shows the number of eigenvalues greater than or equal to x for a matrix $(A - xI)$.

The standard procedure is to select trial values of x and use the sign count to update an array of both upper and lower bounds for the eigenvalues. Initially all elements of the array of lower bounds is set to -1 and the upper bounds to $+1$. The trial values of x may then be chosen by binary chopping between the bounds for a particular eigenvalue. For convenience one might choose to determine the smallest eigenvalue first and then work systematically through the matrix. It may be noted that whilst trying to determine one eigenvalue, information might well be provided on the upper and lower bounds for other eigenvalues. Clearly the eigenvalues may be calculated to the desired accuracy by making the upper and lower bounds appropriately close.

An indication of the speed of the Sturm sequence method, using a simple binary chopping technique, is given in **Fig. 1.** One important disadvantage of the method is that the algorithm must be applied to the full nth order matrix whilst determining each of the n eigenvalues.

3. Improved Sturm sequence method

Once an eigenvalue has been isolated by the Sturm sequence method and its upper and lower bound narrowed to a suitable range, an interpolation method would give an improvement in convergence over a chopping technique.

* *Ministry of Aviation, Royal Radar Establishment, Malvern, Worcs.*

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

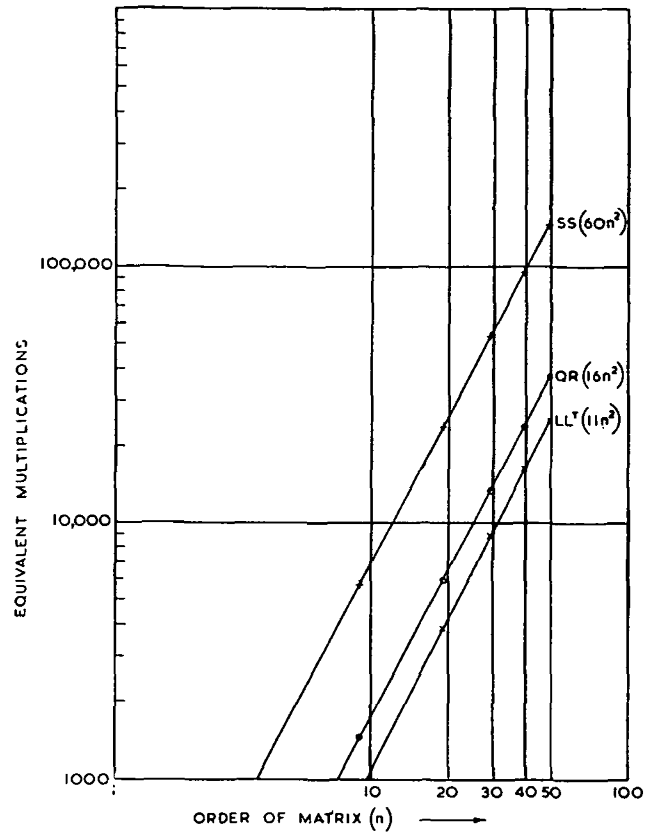

Fig. 1.—Comparison of equivalent multiplications

The values of the determinant $|A - xI|$ provides a possible though not ideal function for interpolating. Clearly other more appropriate functions for interpolating might be devised. Nevertheless the fundamental drawback of the Sturm sequence method remains; no reduction in the size of a matrix is possible after an eigenvalue has been determined.

4. The QR method

An alternative approach to the problem of improving the Sturm sequence method is provided by the QR method (see e.g. Ortega and Kaiser, 1963).

The QR algorithm for determining the eigenvalue of a matrix A may, for the purpose of exposition, be described in terms of the following two operations:

$$\left.\begin{array}{ll}\text{decompose} & A_k \text{ into } Q_kR_k \\ \text{and form} & A_{k+1} := R_kQ_k\end{array}\right\} \text{for } k = 0, 1 \ldots \infty$$

where A is considered to be the initial matrix A_0, Q_k is an orthogonal matrix and R_k an upper triangular matrix. All members of the sequence A_k are symmetric tridiagonal matrices having the same eigenvalues, and A_k tends to a diagonal matrix with diagonal elements (i.e. eigenvalues) ordered in descending moduli as k tends to infinity. For large k each off-diagonal element $b_j^{(k)}$ tends to zero as the ratio $(\lambda j/\lambda_{j-1})^k$ tends to zero.

The advantage of this method is that with each iteration the entire matrix is processed and, after any initial ordering, all the elements tend to their final value. As the off-diagonal element corresponding to the smallest eigenvalue becomes zero (i.e. $|b_n| < \epsilon$) the matrix of order n may be factorized into a single element and a matrix of order $n - 1$. The progressive reduction in the size of the matrix makes it attractive as a replacement for the interpolation method in the Sturm sequence, since it gives a gain of nearly two in speed. Often an off-diagonal element other than the nth becomes zero and the matrix can be factorized into two or more direct sums. This factorization further reduces the work required in subsequent iterations (see Table 5). In the special cases where one of these secondary matrices is a 1×1 or 2×2 matrix, no further iterations are required as the solution of these cases is trivial. In all cases where an off-diagonal element is reduced to zero we shall designate the resulting factorization as *secondary* factorization.

It is advantageous to precede the QR method by the Sturm sequence method because the convergence rate of the QR method is intrinsically rather slow. To improve this one can introduce an origin shift so that the smallest eigenvalue of the new matrix $A - xI$ is very nearly zero. This will clearly result in a very rapid convergence of $b_n^{(k)}$ to zero, thus leading to an early secondary factorization. The initial choices of the origin shift therefore came from the Sturm sequence arrays. The origin shift was further improved with each iteration of the QR method by means of an accelerated inverse linear interpolation procedure based on the values of the characteristic polynomial. The most appropriate range at which to change over the strategy from the Sturm sequence technique to the QR technique was determined empirically.

It will be seen from Fig. 1 that a significant reduction in program time was obtained by use of the QR method.

5. The LL^T method

Another technique suitable for speeding the final stages of the Sturm sequence method is the LL^T method.

The LL^T algorithm for determining the eigenvalues of a matrix A may be summarized as follows:

$$\left.\begin{array}{ll}\text{decompose} & A_k \text{ into } L_kL_k^T \\ \text{and form} & A_{k+1} := L_k^TL_k\end{array}\right\} \text{for } k = 0, 1 \ldots \infty$$

where A is considered to be the initial matrix A_0, L_k is a lower triangular matrix and L_k^T its transpose. All members of the sequence A_k are symmetric tridiagonal matrices having the same eigenvalues, and A_k tends to a diagonal matrix with diagonal elements (i.e. eigenvalues) ordered in descending moduli. For large k each off-diagonal element $b_j^{(k)}$ tends to zero as the ratio $(\lambda_j/\lambda_{j-1})^{k/2}$ tends to zero. This apparent disadvantage of the LL^T method in having only half the convergence rate of the QR method is compensated for by requiring only half the computation of the QR method per iteration (see **Table 1**).

Like the QR method, the LL^T method possesses the

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

Table 1

Number of technique and arithmetic operations for one iteration of each method

TECHNIQUE (Applied to $n \times 3$ matrix)	ARITHMETIC OPERATIONS			TECHNIQUE OPER-ATIONS
	$+$	$\times$	$\div$	
QR + determinant evaluation	$6n$	$4n$	$2n$	n
Sturm sequence	$\sim\frac{3n}{2}$	$3n$	—	n
LL^T sign reversal at the ith element	$i-1$	—	$i-1$	$i-1$

important property of secondary factorization. However, the LL^T decomposition may be unstable when the matrix A is not positive definite, and this fact greatly alters the strategy for choosing the accelerating origin shift. More precisely, the trial values of the origin shift during each LL^T iteration must be chosen to be less than the lowest eigenvalue.

It will be convenient during the subsequent discussion of the LL^T method to be able to refer to details of the LL^T algorithm with origin shifts. (E.g. see Ortega and Kaiser, 1963.) Let the square of the diagonal elements of the lower triangular matrix L be $dl_1^2, \ldots dl_n^2$ and the squares of the off-diagonal elements be $dm_2^2 \ldots dm_n^2$. Define the vectors

$$c_i = a_i - x_0 \qquad (1)$$

where x_0 is the last origin shift, then if $\bar{x}_0$ is the current origin shift the decomposition becomes

$$\begin{aligned} y &:= \bar{x}_0 - x_0 \\ dl_1^2 &:= c_1 - y \\ \left.\begin{aligned} dm_i^2 &:= b_i^2/dl_{i-1}^2 \\ dl_i^2 &:= c_i - y - dm_i^2 \end{aligned}\right\} & \; i = 2, \ldots n \end{aligned} \qquad (2)$$

and the recomposition becomes

$$\begin{aligned} \left.\begin{aligned} \bar{c}_{i-1} &:= dl_{i-1}^2 + dm_i^2 \\ \bar{b}_i^2 &:= dl_i^2 \times dm_i^2 \end{aligned}\right\} & \; i = 2, \ldots n \\ \bar{c}_n &:= dl_n^2 \\ \bar{x}_0 &:= x_0. \end{aligned} \qquad (3)$$

In this way we effect the transformation

$$A - x_0 I \rightarrow \bar{A} - \bar{x}_0 I.$$

Clearly, as $\bar{b}_n{}^2$ tends to zero, we have

$$\bar{c}_n \rightarrow \lambda_n - \bar{x}_0$$

where λ_n is the nth eigenvalue.

Now it was mentioned above that the effectiveness of the acceleration of the convergence depends on the rapidity with which x_0 can be made to approach λ_n from below. Rutishauser (1956) suggested the following technique. If we know two different values of x, say x_1 and x_2, such that $\lambda_n > x_1 > x_2$, and we know the value of the characteristic polynomial of A for x_1 and x_2, say $F(x_1)$ and $F(x_2)$, then a better choice for the origin shift is obtained by linear extrapolation to zero F. Thus we have

$$x := x_1 - F(x_1)(x_1 - x_2)/(F(x_1) - F(x_2))$$

since it is readily proved that

$$\lambda_n > x > x_1 > x_2$$

and thus is a better choice than either x_1 or x_2.

The extrapolation method of calculating the next origin shift was compared with that of using the last diagonal element (Rutishauser, 1960) or of using the lower solution of the bottom quadratic submatrix (Francis, 1962). All three strategies were compared within the context of halting the decomposition when the matrix ceased to be positive definite. One immediate advantage of the extrapolation technique is that, in principle, it leaves the matrix in a positive definite form. Also the origin shifts produced by this method were observed to be accurate even when the diagonal elements were initially greatly disordered and before many LL^T iterations had been performed. Only when the roots are ordered and the relevant off-diagonal elements are small do the other two methods produce comparable results.

The value of $F(\bar{x}_0)$ may be obtained from the Sturm sequence algorithm. However, this is an unnecessarily lengthy way of doing so since $F(\bar{x}_0)$ can be readily calculated from equations (2, 3) and it is given by the expression

$$F(\bar{x}_0) = dl_1^2 \times dl_2^2 \ldots \times dl_n^2. \qquad (4)$$

Again, in order to make this acceleration technique effective the trial value of the origin shift must lie close to the eigenvalue. This initial processing is done using a Sturm sequence method which is required to provide two lower bounds lying close to the lowest eigenvalue.

It was observed that this information can be obtained using the LL^T algorithm itself. For the quantities dl_i^2 for the origin shift x were related to the Sturm sequence functions $f_i(x)$ as follows:

$$\begin{aligned} dl_1^2 &= f_1(x) \\ dl_i^2 &= f_i(x)/f_{i-1}(x) \quad i = 2 \ldots n. \end{aligned} \qquad (5)$$

Thus if the matrix $A - xI$ is to be positive definite all the $dl_i^2 > 0$ but if $x > \lambda_n$ (i.e. if $A - xI$ is not positive definite) at least one $dl_i^2 < 0$. Indeed if the attempted decomposition fails when dl_k^2 is zero or negative then there are at least $k - 1$ eigenvalues greater than or equal to the current origin shift $\bar{x}_0$ (this following from the Sturm sequence algorithm).

This information is of course less than that provided by the complete Sturm sequence. The algorithm only provides precise upper and lower bounds for the smallest

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

eigenvalue, and incomplete information on the lower bounds of the remaining eigenvalue, and incomplete information on the lower bounds of the remaining eigenvalues. Nevertheless there is a considerable reduction in the complexity of the algorithm as may be seen from Table 1. Furthermore at each changeover from a chopping to an extrapolating technique half of the LL^T algorithm has already been executed. The size of the resultant program is, of course, also reduced. On these grounds we therefore decided to exploit this property of the LL^T algorithm and not to incorporate the complete Sturm sequence algorithm.

The above considerations shaped the general strategy of the LL^T method. A detailed exposition of the precise algorithm devised is given in the Appendix. It is clear from Fig. 1 that this approach enabled the LL^T program to produce an even faster method for calculating eigenvalues than the QR method.

6. Comparison of methods

The simple Sturm sequence method, the QR method and the LL^T method were all programmed in ALGOL following the schemes indicated above. These programs were then run on the RREAC computer and compared for speed.

Except in the case of the simple Sturm sequence program the number of iterations per root does not provide an appropriate assessment of the program speed. This follows since the computational complexity of the LL^T and QR algorithms is widely different. Also, such a count would completely mask the saving introduced by the following considerations: first the reduction in the order of the matrix by secondary factorization at various stages during the computation. Secondly, the fact that the partial Sturm sequence method is likely to be halted at the kth element where k is very much less than the current order of the matrix. The first point is of particular importance when trying to assess acceleration techniques of the types we have considered. This arises since their general effect is to reduce the number of iterations required whilst the matrix is of high order at the expense of increasing the number of iterations which might have to be performed on lower order secondary factors.

In order to provide an accurate and machine-independent comparison of the relative running times of the programs, the number of operations performed in the inner loops of the programs was counted. Both the number of times each technique was applied and also the number of arithmetic operations used, were counted using the scheme provided in Table 1.

The first matrix used for testing purposes was of the form $a_i = 0{\cdot}5$ for $i = 1 \ldots n$ and $b_i = 0{\cdot}25$ for $i = 2 \ldots n$. This matrix is particularly suitable for testing purposes because it possesses an analytic solution.

$$\lambda_j = \cos^2\left(\frac{j}{2(n+1)}\right) \quad \text{for} \quad j = 1 \ldots n.$$

Table 2

Test matrix and eigenvalues

DIAGONAL	CO-DIAGONAL	EIGENVALUES
0·7150 7000	0·0000 0000	0·8381 8541
0·4272 1000	0·1395 2000	0·7578 7017
0·7122 6000	0·1138 9000	0·7473 4873
0·4282 3000	0·1738 5000	0·4358 4777
0·7017 7000	0·0216 8100	0·4277 3464
0·4405 2000	0·1289 9000	0·3955 4758
0·4347 4000	0·0035 0160	0·3836 0934
0·4286 2000	0·0025 3720	0·3022 7655
0·4278 4000	0·0000 0000	0·4278 4000

Table 3

Effect of initial interpolation interval on QR method

INTERPOLATION INTERVAL	TECHNIQUE OPERATIONS		ARITHMETIC OPERATIONS		
	SS	QR	$+$	$\times$	$\div$
0·001	423	63	1012	1521	126
0·005	268	85	912	1144	170
0·010	242	85	873	1066	170
0·050	148	122	954	932	244
0·100	124	135	996	912	270

Further, it only exhibits single element secondary factorization, except for the final 2×2 submatrix, a feature which makes comparison between the two methods less dependent on the precise strategy chosen.

In order to exhibit the power of secondary factorization the matrix stated in **Table 2** was used; this matrix exhibits all types of secondary factorization.

The initial stage in assessing the programs was to optimize the QR program by choosing the most suitable range at which to change over the origin shifting technique. The results pertaining to runs using several trial values of the change-over range are recorded in **Table 3**; the value 0·05 was chosen as being the optimum range. It will be noted that the performance of the program is sensitive to the choice of this parameter. The LL^T method, however, was not found to be particularly sensitive to the choice.

The optimized QR method was then compared with the LL^T method and the simple SS method for matrices of various orders. The number of arithmetic operations required is recorded in **Table 4.** In order to display the comparative performances of the two methods in a form which may be more readily assimilated, a graph of number of "equivalent multiplications" for each method is presented in Fig. 1. The artificial parameter

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

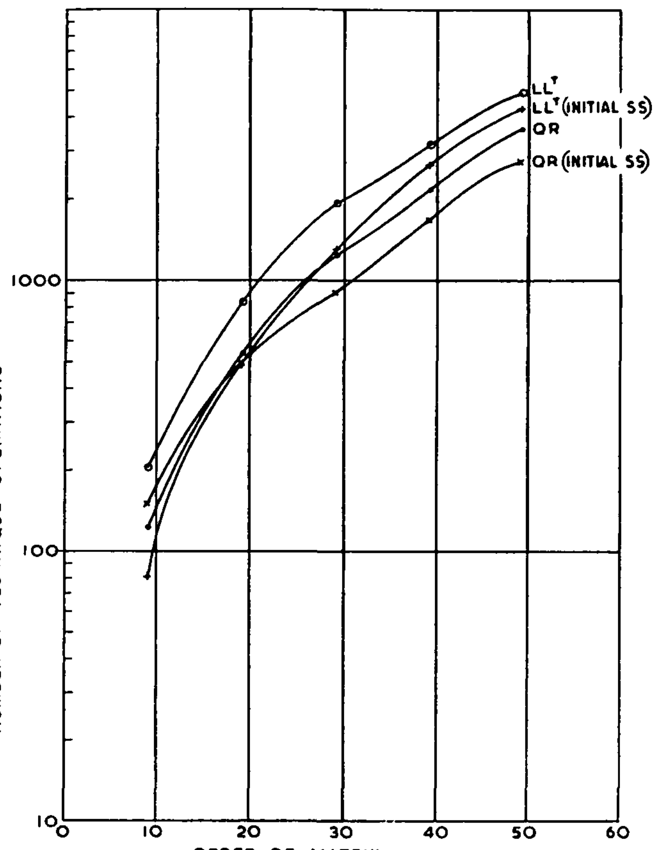

Fig. 2.—Comparison of technique operations

"equivalent multiplication" is found by forming the sum of arithmetic operations of multiplication, division and addition, with each division being counted as being equivalent in machine time to one and a quarter multiplications, and each addition (or subtraction) as equivalent to one quarter of a multiplication.

It is well known that the number of operations required to determine the eigenvalues of a tridiagonal matrix is proportional to the order of the matrix squared. From Fig. 1 it was deduced that the constant of proportionality for the first text matrix was 60 for the simple Sturm sequence method, 16 for the QR method and 11 for the LL^T method. Reference to **Fig. 2** and Table 1 shows that the superiority of the LL^T method is gained because of the small number of arithmetic operations necessary for each technique operation. The evaluation of the characteristic polynomial directly from the LL^T algorithm is obviously of great value in achieving this. In view of its superiority only the algorithm for the LL^T method is presented (see Appendix).

The performance figures for both the LL^T and the QR methods are greatly enhanced when dealing with a matrix which exhibits more than just single element secondary factorization. From the convergence conditions it is clear that secondary factorization will tend to arise when certain of the roots are widely spread in value. The second test matrix shown in Table 2 has

Table 4

Comparison of methods

OPERATION	MATRIX ORDER / METHOD	9	19	29	39	49
Addition	LL^T	481	2096	5161	9230	14298
	QR	954	3940	8833	15592	25213
	SS	2524	10602	23881	42061	65194
Multiplication	LL^T	402	1606	3838	6576	9994
	QR	932	3625	7691	13809	22275
	SS	5049	21204	47763	84123	130389
Division	LL^T	280	1293	3242	5942	9301
	QR	244	1064	2494	4344	7038
	SS	0	0	0	0	0

Table 5

Comparison of LL^T and QR methods for first and second test matrices. Both matrices are of order 9

MATRIX	LL^T METHOD			QR METHOD		
	+	×	+	+	×	+
First	481	402	280	954	932	244
Second	343	308	189	645	738	138

such wide gaps between groups of eigenvalues and does exhibit more than single element secondary factorization. The improvement in performance this brings about may be noted in **Table 5** where the number of arithmetic operations required for both the first and second test matrices are compared. Another class of matrices where secondary factorization aids the speed of convergence is provided by matrices which have close roots without any corresponding off-diagonal elements being small. The matrices W_{2n+1} which have

$$\begin{aligned} a_i &= n + 1 - i \quad (i = 1, \ldots n + 1) \\ a_i &= i - n - 1 \quad (i = n + 2, \ldots, 2n + 1) \\ b_i &= 1 \end{aligned}$$

are examples of this class since their larger eigenvalues occur in extremely close pairs. Two test matrices of this type, one with $n = 10$ and the other with $n = 15$, were examined using the LL^T program. In both cases, after the first 13 roots had been obtained the matrix factorized into two submatrices of equal order between which each of the close pairs of eigenvalues were split.

Clearly secondary factorization provides a technique for significantly increasing the speed of the LL^T algorithm, with the unimportant side effect of not necessarily providing the eigenvalues ordered in size.

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

It may be noted that this device cannot be incorporated in algorithms, such as that of Rutishauser and Schwartz (1963), for general band matrices.

Both the LL^T method and the QR method showed high accuracy; the errors were typically of the order of 10^{-8}. This was achieved using a computer which works to an accuracy of 28 binary digits and without recourse to double-length arithmetic procedures. Nevertheless a cumulative error did build up in the later roots due to the successive processing in the LL^T or QR algorithm and the the effect of errors caused by earlier secondary factorization. Errors are, of course, most severe when the diagonal elements a_{kk} *increase* as k increases. For the matrix A, which was of the form

$$a_i = i/n + 1 \quad (i = 1 \ldots n)$$
$$b_i = 1/n + 1 \quad (i = 2 \ldots n),$$

it was observed that the error in the last root, as calculated by the LL^T method, was of the order of 15×10^{-8} when $n = 100$. This error was dropped to 4×10^{-8} and the overall computing time was halved for the matrix $-A$.

Conclusion

It has been demonstrated that the LL^T algorithm provides a method of speeding up the later stages of the Sturm sequence algorithm. For the test matrices considered the speed of the composite algorithm was more than five times greater than that of the simple Sturm sequence algorithm. This increase was achieved, in part, because of the successive reduction in the order of the matrix after each determination of an eigenvalue; this feature is, of course, not available to purely Sturm sequence methods. Further improvements to the given algorithm by means of more sophisticated origin shifting techniques, may be possible. However, our experience so far is that such refinements produce no significant increase in speed.

Appendix

Details of the LL^T algorithm

In order to render the detailed LL^T algorithm more lucid it is preceded by a glossary for the identifiers used.

Identifier	*Description*
up 1, *low* 1	coefficient of first / last diagonal element in current submatrix after *primary* factorization
up 2, *low* 2	coefficient of first / last off-diagonal element in current submatrix after *secondary* factorisation
i,j,k,l	working integers
x	change in trial origin shift
shift	last established origin shift
*x*1	trial origin shift
*x*2	lower bound for current eigenvalue
*x*0	upper bound for current eigenvalue
*x*12	*x*1 − *x*2
*f*1, *f*2	value of characteristic polynomial corresponding to *x*1 / *x*2
*f*12	*f*1 − *f*2
*z*1, *z*2	working variables when evaluating quadratic factors
stf, *stb*	step forward / back special variable to be used when trial value *x*1 lies close to or outside limits *x*0 and *x*2
lb [1 : *m*]	array of lower bounds for eigenvalues
$dl^2\ dm^2$[1 : *m*]	arrays for use in LL^T algorithm with partial Sturm sequence
chop only	when true the strategy is in initial chopping mode otherwise false
*to x*2	when true chopping is towards *x*2 when false chopping is towards *x*0
root	when true have just obtained an eigenvalue directly from a single element or quadratic factor; otherwise false

```
procedure eigenvalue(a,b,m,eps1,eps2);
value m,eps1,eps2;
real array a,b;
real eps1,eps2;
integer m;
begin
comment eigenvalue finds the eigenvalues of the normalized
        m×m tridiagonal matrix whose diagonal elements
        are in a[1:m] and off-diagonal elements in
        b[2:m] and places them in a;
integer up1,low1,up2,low2,i,j,k,l;
real x,x0,x1,x2,z1,z2,stb,stf,shift,x12,f12,f1,f2;
real array dm2,dl2,lb[1:m];
boolean root,to x2,chop only;
for i:=2 step 1 until m do b[i]:=b[i]↑2;
lb[1]:=−1;
low1:=0;
primfac:up1:=low1+1;
if up1<m then
  begin
  comment test for primary factorization;
  for low1:=up1 step 1 until m−1 do
    begin
    lb[low1+1]:=−1;
```

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

```
    if m=2 then begin low1:=2; go to pquad end;
    if b[low1+1]<eps1 then
      begin
      if low1=up1 then go to primfac;
      if low1-up1>1 then go to start;
  pquad: z1:=b[low1]-a[up1]×a[low1];
      z2:=0·5×(a[up1]+a[low1]);
      a[low1]:=if z2=0 then sqrt(z1)
                 else sqrt(0·25×(a[up1]-a[low1]) ↑ 2
                   +b[low1])×sign(z2);
      a[up1]:=a[low1]+z2;
      a[low1]:=-z1/a[up1];
      go to primfac
      end
    end;
  low1:=m;
  comment set parameters initially;
start:up2:=up1+1;
  low2:=low1;
  shift:=f2:=x1:=0;
    x2:=-1; x0:=1;
  root:=false
  comment reset parameters for next submatrix;
ll: if root then
    begin
    if a[l]<lb[l] then for j:=l-1 step -1 until up2-1
      do lb[j]:=-1;
    x2:=if a[l]<lb[l-1] then a[l] else lb[l-1];
    x0:=shift+a[up2-1]+0·1;
    if x2>x0 then x0:=1;
    x1:=x2+(x0-x2)/(low2-up2+2);
    root:=false
    end;
  x:=0;
  stb:=stf:=·003;
    to x2:=chop only:=true;
  comment perform ll decomposition;
decomp:x:=x1-shift;
  i:=up2-1;
  dl2[i]:=a[i]-x;
  if dl2[i]<eps2 then go to fail;
  for i:=up2 step 1 until low2 do
    begin
    dm2[i]:=b[i]/dl2[i-1];
    dl2[i]:=(a[i]-dm2[i])-x;
    if dl2[i]<eps2 then
      begin
      comment decomposition has failed reset x0 and
        lower bounds;
  fail:x0:=x1;
      for j:=i-1 step -1 until up2-1 do if lb[j]<x1
        then lb[j]:=x1;
      stf:=0·003;
      if chop only and (x1-x2)<$_{10}{}^{-4}$ then
      begin
        comment special case for eigenvalues close to or
                less than x2;
        x1:=if x1-stb>-1 then x1-stb else -1·1;
        stb:=10×stb
        end   else
        begin
        comment reset x1 by chopping to x2;
        x1:=if x1-x2<eps2 then x1-eps2
          else x1+0·6×(x2-x1);
        if not to x2 then chop only:=false
        end;
      goto decomp
      end of action on failure
    end of decomposition;
  comment decomposition has succeeded start recomposi-
    tion;
  stb:=·003;
  f1:=dl2[up2-1];
  for i:=up2 step 1 until low2 do
    begin
    a[i-1]:=dl2[i-1]+dm2[i];
    b[i]:=dl2[i]×dm2[i];
    f1:=f1×dl2[i]
    end;
  a[low2]:=dl2[low2];
  shift:=x1;
  comment test for secondary factorization;
sec: for j:=low2 step -1 until up2 do if b[j]<eps1 then
      begin
      if j<low2-1 then
        begin
        comment matrix has factored reset a and up2;
        for k:=up2-1 step 1 until j-1 do a[k]:=a[k]
          +shift
        up2:=j+1;
        go to ll
        end;
      if j=low2-1 then
        begin
        comment quadratic factor;
  squad:l:=low2-1;
        z1:=b[low2]-a[l]×a[low2];
        z2:=0·5×(a[l]+a[low2]);
        a[low2]:=if z2=0 then sqrt(z1)
                   else sqrt(0·25×(a[l]-a[low2])
                     ↑ 2+b[low2])×sign(z2);
        a[l]:=a[low2]+z2;
        a[low2]:=-z1/a[l]+shift;
        low2:=low2-2
        end     else
        begin
        comment single element;
  single:l:=low2;
        low2:=low2-1
        end;
      a[l]:=a[l]+shift;
      root:=true;
      if low2>up2 then go to sec else
    lq:if low2=up2 then go to squad else
      if low2=up2-1 then go to single else
        begin
        comment see if there are any more submatrices;
        if up2-up1≤ 1 then go to primfac;
```

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024

```
        comment reset up2,low2 and shift for the next
          submatrix;
        low2:=up2-2;
        up2:=up1+1;
        shift:=0;
        go to if low2>up2 then sec else lq
        end
      end test for secondary factorization;

    if root then go to ll else
      begin
      x12:=x1-x2;
      f12:=f1-f2;
      x2:=x1;
      f2:=f1;
      if chop only or abs(x12)> ·01 then
      begin
      comment reset x1 by binary chopping to x0;
      to x2:=false;
      if (x0-x1)<10^{-4} then
        begin
        comment special case for eigenvalue close to or
                      greater than x0;
        x1:=if x1+stf<1 then x1+stf else 1;
        stf:=stf×10
        end  else
        x1:=(x0+x1)/2
      end  else
      begin
      comment reset x1 by extrapolation;
      if f12≠0 and x12≠0 then x1:=x1-x12×f1/f12
      end;
   goto decomp
   end
end
end procedure eigenvalue;
```

General comments

(*a*) The *basic sequence* of operations is as follows:

1. Chop towards $x2$ until there is a successful decomposition.
2. Chop towards $x0$ until there is a failure in the decomposition.
3. Again chop towards $x2$ until the difference between $x1$ and $x2$ is less than 0·01.
4. Extrapolate towards $x2$.

It is possible, however, for a situation to arise where $x0$ and $x2$ do not straddle a root and so special action must be taken. This situation could arise in one of two ways. First, it is possible for secondary factorization to occur before roots are fully ordered, and so produce incorrect upper and lower bounds for the next root. Secondly, it is possible for the extrapolated origin shift to agree with the "true" eigenvalue to almost the full word-length of the machine. Then, if several iterations have to be performed with this choice of origin shift, the round-off errors could change the "true" eigenvalue to a value below $x2$.

(*b*) *The algorithm for choosing the initial trial value of* $x1$ leaves scope for experiment. Apart from the algorithm shown we also tried using the last diagonal element or the lower solution of the bottom quadratic factor (Francis, 1962).

Choosing the latter method produced a low number of Sturm sequence iterations but a large number of LL^T iterations. The method adopted here had a large number of Sturm sequence iterations but a much lower number of LL^T iterations. The third approach lay roughly between the other two. Each method was assessed by counting the number of equivalent multiplications required for a given matrix. It was observed that difference between the techniques was not great (approximately 5–10% variation in the number of equivalent multiplications). This insensitivity appears to arise because of the competing nature of the Sturm sequence and LL^T strategies.

(*c*) For the tests on RREAC which has a 36-bit word-length the following values were used for the constants. $eps1:=10^{-10}$; $eps2:=10^{-8}$.

References

FRANCIS, J. "The Q.R. Transformation: A Unitary Analogue to the L.R. Transformation." (1961) Part I. *The Computer Journal*, Vol. 4, p. 265; (1962) Part II. *The Computer Journal*, Vol. 4, p. 332.

ORTEGA, J. M. (1960). "On Sturm Sequences for Tridiagonal Matrices," *J.A.C.M.*, Vol. 7, p. 260.

ORTEGA, J. M., and KAISER, H. F. (1963). "The LL^T and QR methods for symmetric tridiagonal matrices," *The Computer Journal*, Vol. 6, p. 99.

RUTISHAUSER, H. (1956). "Solution of Eigenvalue Problems with the LR-Transformation," *N.B.S. Applied Maths. Series* 49, p. 47.

RUTISHAUSER, H. (1960). "Über eine kubisch Konvergente Variante der LR-Transformation," *Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 40, p. 49.

RUTISHAUSER, H., and SCHWARZ, H. R. (1963). "The LR Transformation Method for Symmetric Matrices," *Num. Math.*, Vol. 5, p. 273.

WILKINSON, J. H. (1958). "The Calculation of Eigenvectors of Codiagonal Matrices," *The Computer Journal*, Vol. 1, p. 90.

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

Downloaded from https://academic.oup.com/comjnl/article/9/1/98/348182 by guest on 13 March 2024