Acta Numerica (1995), *pp.* 459–491

The New qd Algorithms

Beresford N. Parlett *
Department of Mathematics
University of California
Berkeley CA 94720,
USA
E-mail: parlett@math.berkeley.edu

CONTENTS

1 Introduction 459
2 Bidiagonals versus Tridiagonals 462
3 Stationary qd Algorithms 465
4 Progressive qd Algorithms 467
5 The LR Algorithm for J Matrices 468
6 Gram–Schmidt Factors 471
7 The Meaning of d_i 473
8 Incorporation of Shifts 474
9 Accuracy 475
10 Speed 482
11 Parallel Implementation 484
12 Eigenvectors 485
13 Singular Values of Bidiagonals 489
References 490

1. Introduction

Let us think about ways to find both eigenvalues and eigenvectors of tridiagonal matrices. An important special case is the computation of singular values and singular vectors of bidiagonal matrices. The discussion is addressed both to specialists in matrix computation and to other scientists whose main interests lie elsewhere. The reason for hoping to communicate with two such diverse sets of readers at the same time is that the content of the survey, though of recent origin, is quite elementary and does not demand familiarity with much beyond triangular factorization and the

* The author is grateful for partial support under Contract ONR N00014-90-J-1372.

Gram–Schmidt process for orthogonalizing a set of vectors. For some readers the survey will cover familiar territory but from a novel perspective. The justification for presenting these ideas is that they lead to new variations of current methods that run a lot faster while achieving greater accuracy.

Tridiagonal matrices have received a great deal of attention since the 1950s. The (i, j) entries of these matrices vanish if $|i - j| > 1$ and the interest in them stems from the fact that they include the most narrowly banded representations of a matrix that can be obtained by a finite number of similarity transformations using rational expressions in the matrix entries together with square roots. This statement needs a little justification because the rational canonical form (RCF) is, by construction, *the* matrix with fewest nonzero entries that can be achieved by rational operations on the data. For an $n \times n$ matrix the RCF has only n parameters while a tridiagonal form has $3n-2$, but $2n-1$ when normalized, and so RCF seems preferable.

The fact is that, apart from theorists doing exact computations, the RCF is not used in eigenvalue computations. The main reason is that the representation is too condensed for standard floating-point computation. The coefficients of the characteristic polynomial have to be known to many, many more decimal places than do the matrix entries in order to determine the eigenvalues to the same accuracy.

Beyond that one might add that the RCF is a Hessenberg matrix (entry (i, j) vanishes if $i - j > 1$) and, by design, there are no further useful similarity transformations that can be applied to it. In contrast to the RCF there are infinitely many tridiagonal matrices in a similarity class and so there is hope of computing a sequence of them that converges to bidiagonal or diagonal form. This is where our interest lies.

It should be mentioned that the tridiagonal form is probably also too condensed for the most difficult cases, (see Parlett (1992)), but it is rich enough to suffice for many applications and we shall stay with it here.

Our topic, the new qd algorithms, will be developed as the consequence of two ideas.

The first concerns the representation of tridiagonal matrices and we mention it briefly here. In the eigenvalue context there is no loss of generality in supposing that our tridiagonals are normalized so that all entries in positions $(i, i+1)$ are either 0 or 1. Zeros here make calculations easier so we may assume that all these entries are 1. Such tridiagonals are denoted by J. Most, but not all, such J permit triangular factorization

$$J = LU,$$

where the precise forms L and U are shown at the beginning of the next section. It is clear that in the $n \times n$ case L and U together are defined by $2n-1$

parameters; exactly the same degree of freedom as in J. Section 3 argues that the pair L, U is preferable to J itself. Consequently all transformations on tridiagonals should be re-examined in this representation.

The second idea relates to the LR algorithm (LR) discovered by H. Rutishauser in 1957, (see Rutishauser (1958)), and presented in Section 5 here. When LR is rewritten in the L, U representation one obtains the (progressive) qd algorithm. The letters q and d are lower case because they do not stand for matrices and the matrix-computation community tries to reserve capital letters for matrices. Thus q here has nothing to do with the Q of the QR factorization.

In the new representation the LR algorithm spends most of its time computing the triangular factorization not of a single matrix, but of a product, namely UL. It is not well enough appreciated that finding the LU factorization of any product BC is equivalent to applying a generalized Gram–Schmidt process to the rows of B and the columns of C so that $B = \hat{L}P^*$, $C = Q\hat{U}$, and P^*Q is diagonal. When this Gram–Schmidt process is applied to UL in an efficient manner one obtains a little-known variant of qd, called the *differential* qd algorithm (dqd), that requires a little more arithmetic effort than qd itself. Rutishauser discovered dqd, (see Rutishauser (1958)) near the end of his life and never mentioned the shifted version dqds that K. V. Fernando and I discovered, (see Fernando and Parlett (1994)), independently of Rutishauser's work, in 1991, while trying to improve on the Demmel and Kahan QR algorithm (Demmel and Kahan, 1990) for computing singular values of bidiagonals. The connection of dqd with the generalized Gram–Schmidt process on bidiagonals is new and constitutes the second idea that underpins this survey.

The presentation here runs completely counter to history. The paper by Fernando and Parlett (1994), develops dqds in the singular-value context, gives historical comments and shows the connections with continued fractions. However, none of that is necessary and it is simplicity we pursue here.

This survey develops several qd algorithms (six in all) in a matrix context in the most elementary way. It is not difficult to see several directions in which these ideas may be generalized or modified to good effect.

The differential forms of qd algorithms are the right ones for parallel computation.

A sceptic might say that since no one uses LR algorithms there is no point in finding fancy versions of them. In the general case there is still much work to be done in finding clever shift strategies that approach an eigenvalue in a stable way. However, in the symmetric case even the current simple shift strategies achieve high relative accuracy in all eigenvalues and are between 2 and 10 times faster than QR; see Fernando and Parlett (1994). Yet the

most powerful argument in favor of qd algorithms may turn out to be the efficient computation of accurate eigenvectors.

The general plan of this survey is conveyed adequately by the table of contents.

2. Bidiagonals versus Tridiagonals

Bidiagonal matrices of a special form will play a leading role in this essay. Whenever possible 6×6 matrices will be used to illustrate the general pattern.

$$L = \begin{bmatrix} 1 & & & & & \\ l_1 & 1 & & & & \\ & l_2 & 1 & & & \\ & & l_3 & 1 & & \\ & & & l_4 & 1 & \\ & & & & l_5 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_1 & 1 & & & & \\ & u_2 & 1 & & & \\ & & u_3 & 1 & & \\ & & & u_4 & 1 & \\ & & & & u_5 & 1 \\ & & & & & u_6 \end{bmatrix}.$$

To save space these matrices may be written as

$$L = bidiag\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ & l_1 & l_2 & l_3 & l_4 & l_5 & \end{pmatrix},$$

$$U = bidiag\begin{pmatrix} & 1 & 1 & 1 & 1 & 1 \\ u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & \end{pmatrix}.$$

The pair L, U determine two triangular matrices; first

$$J = LU = \begin{bmatrix} u_1 & 1 & & & & \\ l_1u_1 & l_1+u_2 & 1 & & & \\ & l_2u_2 & l_2+u_3 & 1 & & \\ & & l_3u_3 & l_3+u_4 & 1 & \\ & & & l_4u_4 & l_4+u_5 & 1 \\ & & & & l_5u_5 & l_5+u_6 \end{bmatrix},$$

which may be written

$$J = tridiag\begin{pmatrix} & 1 & & 1 & \bullet & 1 & \\ u_1 & & l_1+u_2 & & \bullet & & l_5+u_6 \\ & l_1u_1 & & l_2u_2 & \bullet & l_5u_5 & \end{pmatrix},$$

and second

$$J' = UL = \begin{bmatrix} u_1+l_1 & 1 & & & & \\ u_2l_1 & u_2+l_2 & 1 & & & \\ & u_3l_2 & u_3+l_3 & 1 & & \\ & & u_4l_3 & u_4+l_4 & 1 & \\ & & & u_5l_4 & u_5+l_5 & 1 \\ & & & & u_6l_5 & u_6 \end{bmatrix},$$

which may be written

$$J' = tridiag \begin{pmatrix} & 1 & & 1 & \bullet & 1 & \\ u_1 + l_1 & & u_2 + l_2 & & \bullet & \bullet & u_6 \\ & u_2 l_1 & & u_3 l_2 & \bullet & u_6 l_5 & \end{pmatrix}.$$

Note that both tridiagonals have their superdiagonal entries, that is, entries $(j, j+1)$, equal to 1. Also note that $J' = L^{-1}JL$. The reader should note the pattern of indices in J and J' because frequent reference will be made to them throughout the survey.

The attractive feature here is that because the 1's need not be represented explicitly the factored form of J requires no more storage than J itself; there are $2n-1$ parameters for $n \times n$ matrices in each case.

Advantages of the factored form

1 L, U determines the entries of J to greater than working-precision accuracy because the addition and multiplication of l's and u's is implicit. Thus J_{ii} is given by $l_{i-1} + u_i$ implicitly but by $fl(l_{i-1} + u_i)$ explicitly.

2 The mapping $L, U \rightarrow J$ is naturally parallel; for example, $l * u$ gives the off-diagonal entries. In contrast the mapping $J \rightarrow L, U$, that is, Gaussian elimination, is intrinsically sequential.

3 Singularity of J is detectable by inspection when L and U are given, but only by calculation from J.

4 Solution of $Jx = b$ takes half the time when L and U are available.

Disadvantages of the factored form

The mapping $J \rightarrow L, U$ is not everywhere defined. Even when the factorization exists it can happen that $\|L\|$ and $\|U\|$ greatly exceed $\|J\|$. This is very bad for applying the LR algorithm but harmless when eigenvectors are to be calculated. So we should be careful to consider the goal before stigmatizing a process as unstable. Moreover in the eigenvalue context we are free to replace J by $J - \sigma I = LU$ for some suitably chosen shift σ that gives acceptable L and U.

Frequently we make the *nonzero assumption*: $l_j u_j \neq 0$, $j = 1, \ldots, n-1$. If $u_n = 0$ then a glance at $J' = UL$ shows that its last row is zero. However,

it is not valid to simply discard l_{n-1} along with u_n. In other words we do not have a factored form of the leading $(n-1)\times(n-1)$ principal submatrix of J' unless l_{n-1} is negligible compared to u_{n-1}. Similarly if u_1 is zero we must not ignore l_1. In fact any zero values among $\{l_j,\ u_j;\ j=1,\ldots,n-1\}$ are readily exploited.

Splitting: If $l_j = 0,\ j < n$, then the spectrum of J is the union of the spectra of two smaller tridiagonals given in factored form by $\{l_i,\ u_i;\ i=1,j\}$ and $\{l_i,\ u_i;\ i=j+1,n\}$. Here $l_n = 0$.

Singularity: If $u_j = 0,\ j \le n$, then zero is an eigenvalue. However, some computation is necessary to deflate this eigenvalue and obtain L and U factors of an $(n-1)\times(n-1)$ matrix. One pass of the qd algorithm (given later) will suffice in exact arithmetic.

Any tridiagonal matrix that does not split, or its transpose, is diagonally similar to a form with 1's above the diagonal, that is, a J matrix. So for eigenvalue hunting there is no loss of generality in using this normalization. However, if the normalization is not convenient then there is an alternative factorization of tridiagonals that was considered by H. Rutishauser; see Rutishauser (1990). Let

$$N = bidiag\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

$$B = bidiag\begin{pmatrix} & e_1 & & e_2 & & e_3 & & e_4 & & e_5 & \\ q_1 & & q_2 & & q_3 & & q_4 & & q_5 & & q_6 \end{pmatrix}.$$

Then

$$NB = tridiag\begin{pmatrix} & e_1 & & e_2 & \bullet & & e_5 & \\ q_1 & & e_1+q_2 & & \bullet & \bullet & & e_5+q_6 \\ & q_1 & & q_2 & & q_5 & & \end{pmatrix}$$

and

$$BN = tridiag\begin{pmatrix} & e_1 & & e_2 & \bullet & & e_5 & \\ q_1+e_1 & & q_2+e_2 & & \bullet & \bullet & & q_6 \\ & q_2 & & q_3 & & q_6 & & \end{pmatrix}.$$

Note that, after we identify $e_i = l_i,\ q_i = u_i$, for all i, $J = DNBD^{-1}$ and $J' = DBND^{-1}$ with $D = diag(1, e_1, e_1e_2, \ldots, e_1\cdots e_5)$.

Since all the various qd algorithms relate naturally to J matrices we will stay with the L, U factors rather than the N, B representation.

In the past most attention has been paid to the *positive case*: $l_i > 0$, $i = 1,\ldots,n-1,\ u_j > 0,\ j = 1,\ldots,n$. Note in passing the following standard results.

Lemma 1 If $l_i u_i > 0$, $i = 1, \ldots, n-1$, then J is symmetrizable by a diagonal similarity and the number of positive (negative) u_i is the number of positive (negative) eigenvalues.

Lemma 2 If $l_i u_{i+1} > 0$, $i = 1, \ldots, n-1$, then J' is symmetrizable by a diagonal similarity and the number of positive (negative) u_i is the number of positive (negative) eigenvalues.

For a real symmetric or complex Hermitian matrix a preliminary reduction to tridiagonal form has proved to be a stable and efficient step in computing the eigenvalues. In the general case preliminary reduction to tridiagonal form has been less successful. Stability is not guaranteed for any current methods; see Parlett (1992). Sometimes users are lucky but as larger and larger matrices are tried unsatisfactory experiences are more frequent. It may well be that the tridiagonal form is too compact for the difficult cases. The attentive reader will note in the following pages that some of the algorithms can be extended to fatter forms, such as block tridiagonal, but such ideas will not be pursued here.

3. Stationary qd Algorithms

Triangular factors change in a complicated way under translation. Given L and U of the form given in Section 2 the task here is to compute $\bar{L}$ and $\bar{U}$ so that

$$J - \sigma I = LU - \sigma I = \bar{L}\bar{U}$$

for a given suitable shift σ. Equating entries on each side shows that

$$\begin{aligned} l_i + u_{i+1} - \sigma &= \bar{l}_i + \bar{u}_{i+1}, \quad i = 0, \ldots, n-1, \; l_0 = 0, \\ l_i u_i &= \bar{l}_i \bar{u}_i, \quad i = 1, \ldots, n-1. \end{aligned}$$

These relations yield the so called stationary qd algorithm:

stqd(σ): $\bar{u}_1 = u_1 - \sigma$;
for $i = 1, n-1$
$\bar{l}_i = l_i u_i / \bar{u}_i$
$\bar{u}_{i+1} = l_i + u_{i+1} - \sigma - \bar{l}_i$
end for.

Naturally it fails if $\bar{u}_i = 0$ for some $i < n$.

At this point the sceptical reader might object that stqd(σ) is exactly the same algorithm that would be obtained by forming $J - \sigma I$ and performing Gaussian elimination. Indeed if stqd(σ) is executed with the operations proceeding from left to right, for example,

$$\bar{u}_{i+1} = fl(fl(fl(l_i + u_{i+1}) - \sigma) - \bar{l}_i),$$

then the two procedures are not just mathematically equivalent but also computationally identical. However, it is not necessary to follow this left-to-right ordering. For example, one could write

$$\bar{u}_{i+1} = (l_i - \bar{l}_i - \sigma) + u_{i+1}$$

and, if the compiler respects parentheses, then stqd(σ) will quite often produce different output than Gaussian elimination on $J - \sigma I$. If l_i and $\bar{l}_i$ are much larger than u_{i+1} then the second form is more accurate than the first.

The preceding thoughts lead to an alternative algorithm, easily missed, for $\bar{L}$ and $\bar{U}$. It involves more arithmetic effort and an auxiliary storage cell but has some striking advantages in accuracy for finite-precision arithmetic.

To derive the algorithm define variables $\{t_i\}$ by

$$t_{i+1} \equiv \bar{u}_{i+1} - u_{i+1} = l_i - \bar{l}_i - \sigma.$$

Observe that

$$\begin{aligned} t_{i+1} &= l_i - l_i u_i/\bar{u}_i - \sigma \\ &= l_i(\bar{u}_i - u_i)/\bar{u}_i - \sigma \\ &= t_i l_i/\bar{u}_i - \sigma. \end{aligned}$$

For reasons that are not clear Rutishauser called the associated algorithm the *differential* form of stqd. We call it dstqd.

dstqd(σ): $t_1 = -\sigma$;
for $i = 1, n-1$
$\bar{u}_i = u_i + t_i$
$\bar{l}_i = u_i(l_i/\bar{u}_i)$
$t_{i+1} = t_i(l_i/\bar{u}_i) - \sigma$
end for
$\bar{u}_n = u_n + t_n$.

In practice the t-values may be written over each other in a single variable t. If the common subexpression $l_i/\bar{u}_i$ is recognized then only one division is needed. Thus dstqd exchanges a subtraction for a multiplication so the extra cost is not excessive.

At first sight stqd may not seem relevant to the eigenvalue problem but if λ is a very accurate approximation to an eigenvalue then dstqd(λ) is needed to approximate the associated eigenvector; see Section 12. In this application huge values among the $\{\bar{u}_i\}$ are to be expected and do not have deleterious effect on the computed eigenvector. In fact dstqd(σ) may be used to find eigenvalues too by extracting a good approximate eigenvalue from the t-values for a shift.

4. Progressive qd Algorithms

This section seeks the triangular factorization of $J' - \sigma I$, not $J - \sigma I$:

$$J' - \sigma I = UL - \sigma I = \hat{L}\hat{U}$$

for a suitable shift σ. Equating entries on each side of the defining equation gives the so-called rhombus rules of H. Rutishauser (see Rutishauser (1954), in German and, in English, Henrici (1958)):

$$u_{i+1} + l_{i+1} - \sigma = \hat{l}_i + \hat{u}_{i+1} \quad \text{and} \quad l_i u_{i+1} = \hat{l}_i \hat{u}_i.$$

These relations give the so-called progressive qd algorithm with shift which we call qds(σ).

qds(σ): $\hat{u}_1 = u_1 + l_1 - \sigma$;
for $i = 1, n-1$
$\hat{l}_i = l_i u_{i+1}/\hat{u}_i$
$\hat{u}_{i+1} = u_{i+1} + l_{i+1} - \sigma - \hat{l}_i$
end for.

The algorithm qds fails when $\hat{u}_i = 0$ for some $i < n$. In contrast to the stationary algorithm the mapping $\sigma, L, U \rightarrow \hat{L}, \hat{U}$ is nontrivial even when $\sigma = 0$. When $\sigma = 0$ we write simply qd, not qds.

At this point the skeptical reader might object that qds(σ) is exactly the same algorithm that would be obtained by forming $J' - \sigma I$ and performing Gaussian elimination. Indeed if the operations are done proceeding from left to right, for example,

$$\hat{u}_{i+1} = fl(fl(fl(u_{i+1} + l_{i+1}) - \sigma) - \hat{l}_i),$$

then the two procedures are not just mathematically equivalent but also computationally identical. However it is not necessary to follow this ordering. For example, one could write

$$\hat{u}_{i+1} = (l_{i+1} - \hat{l}_i - \sigma) + u_{i+1}$$

and, if the compiler respects parentheses, then the output will quite often be different.

There is an alternative implementation of qds that is easy to miss. In fact Rutishauser never wrote it down. The new version is slightly slower than qds but has compensating advantages. Here we derive it simply as a clever observation leaving to later sections the task of making it independent of qds.

As suggested in an earlier paragraph we might define an auxiliary variable

$$d_{i+1} \equiv u_{i+1} - \hat{l}_i - \sigma \; (= \hat{u}_{i+1} - l_{i+1}).$$

Observe that

$$\begin{aligned} d_{i+1} &= u_{i+1} - (l_i u_{i+1}/\hat{u}_i) - \sigma \\ &= (u_{i+1}(\hat{u}_i - l_i)/\hat{u}_i) - \sigma \\ &= (u_{i+1} d_i/\hat{u}_i) - \sigma. \end{aligned}$$

Rutishauser seems to have discovered the unshifted version two or three years before he died, perhaps 15 years after discovering qd, but he did not make much use of it; see Rutishauser (1990). He called it the *differential* qd algorithm (dqd for short) and so we call the new shifted algorithm dqds (*d*ifferential qd with *s*hifts).

dqds(σ): $d_1 = u_1 - \sigma$;
for $i = 1, n-1$
$\hat{u}_i = d_i + l_i$
$\hat{l}_i = l_i(u_{i+1}/\hat{u}_i)$
$d_{i+1} = d_i(u_{i+1}/\hat{u}_i) - \sigma$
end for
$\hat{u}_n = d_n$.

By definition, dqd = dqds(0). In the positive case dqd requires *no subtractions* and enjoys very high relative stability; see Section 8. In practice each d_{i+1} may be written over its predecessor in a single variable d. Looking ahead we mention that the quantity $\min |d_j|$ gives useful information on the eigenvalue nearest 0.

5. The LR Algorithm for J Matrices

As mentioned in Section 1 our exposition reverses the historical process. Rutishauser discovered LR by interpreting qd in terms of bidiagonal matrices, a brilliant and fruitful insight. This is worth explaining. By definition, the LR transform of J is J' and of J' is the matrix J'' defined in two steps by

$$J' = \hat{L}\hat{U}, \quad J'' = \hat{U}\hat{L}.$$

Now qd applied to L and U yields $\hat{L}$ and $\hat{U}$ and so defines J'' implicitly. There is no need to form J' or J''.

When shifts are employed the situation is a little more complicated. It is necessary to look at two successive steps with shifts σ_1 and σ_2.

In shifted LR

$$\begin{aligned} J_1 - \sigma_1 I &= L_1 U_1, \\ J_2 &= U_1 L_1 + \sigma_1 I, \\ J_2 - \sigma_2 I &= L_2 U_2, \\ J_3 &= U_2 L_2 + \sigma_2 I. \end{aligned}$$

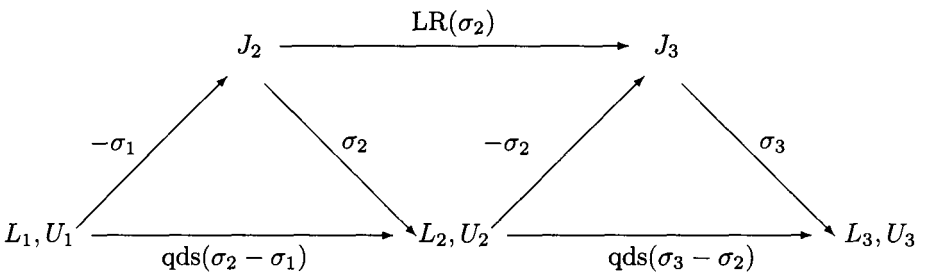

Fig. 1. Relation of LR to qds

In other words, the shifts are restored so that J_1, J_2, J_3 are similar. Note that

$$\begin{aligned} J_2 = U_1 L_1 + \sigma_1 I &= L_1^{-1}(J_1 - \sigma_1 I)L_1 + \sigma_1 I \\ &= L_1^{-1} J_1 L_1. \end{aligned}$$

However, if J_2 is not to be formed one cannot *explicitly* add σ_1 back to the diagonal. On the other hand

$$J_2 - \sigma_2 I = U_1 L_1 - (\sigma_2 - \sigma_1 I) = L_2 U_2.$$

Thus to find L_2 and U_2 from L_1 and U_1 it is only necessary to apply qds($\sigma_2 - \sigma_1$). In other words to get qds equivalent to LR with shifts $\{\sigma_i\}_{i=1}^{\infty}$ it is necessary to use the differences $(\sigma_i - \sigma_{i-1})$ with qds. In LR the shifts should converge to an eigenvalue of the original J or J'. In qds the shifts should converge to 0 and $u_n \to 0$, $l_{n-1} \to 0$ too and all shifts must be accumulated. It is worth recording the relationship in a diagram in Figure 1.

In practice the LR algorithm avoids explicit calculation of the L's and U's and the transformation $J_i \to J_{i+1}$ is effected via a sequence of elementary similarity transformations. As implemented in the late 1950s and early 1960s, LR proved insufficiently reliable and was displaced by the QR algorithm in the mid 1960s. In one important class of applications both LR and qds were accurate and efficient: the positive case $l_i > 0$, $u_i > 0$ for all i.

For comparison purposes we present two implementations of LR: the explicit and the implicit shift versions. Let

$$J = tridiag \begin{pmatrix} & 1 & & 1 & & \bullet & & 1 & \\ \alpha_1 & & \alpha_2 & & \bullet & & \bullet & & \alpha_n \\ & \beta_1 & & \beta_2 & & \bullet & & \beta_{n-1} & \end{pmatrix},$$

let $J - \sigma I = LU$, $\hat{J} = UL + \sigma I$ and write $\hat{J}$ in the same notation as J. The matrix L may be written as a product of lower-triangular plane transformers N_i. So

$$U = N_{n-1}^{-1} \cdots N_1^{-1}(J - \sigma I), \quad \hat{J} = U N_1 \cdots N_{n-1} + \sigma I$$

and the active part of N_i is $\begin{pmatrix} 1 & 0 \\ e_i & 1 \end{pmatrix}$, where the multiplier e_i is in position $(i+1, i)$. The following diagram shows a typical stage.

$$\begin{pmatrix} 1 & 0 \\ -e_j & 1 \end{pmatrix} \begin{pmatrix} d_j e_{j-1} & d_j & 1 \\ \beta_j e_{j-1} & \beta_j & \alpha_{j+1} - \sigma \end{pmatrix} = \begin{pmatrix} \hat{\beta}_{j-1} & d_j & 1 \\ 0 & 0 & d_{j+1} \end{pmatrix},$$

$$\begin{pmatrix} d_j & 1 \\ 0 & d_{j+1} \\ 0 & \beta_{j+1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ e_j & 1 \end{pmatrix} = \begin{pmatrix} d_j + e_j & 1 \\ e_j d_{j+1} & d_{j+1} \\ e_j \beta_{j+1} & \beta_{j+1} \end{pmatrix}.$$

LR (explicit shift σ): $d_1 = \alpha_1 - \sigma$;
for $i = 1, n-1$
if $d_i = 0$ **then** exit (fail)
$e_i = \beta_i / d_i$
$\hat{\alpha}_i = d_i + (e_i + \sigma)$
$d_{i+1} = \alpha_{i+1} - (e_i + \sigma)$
$\hat{\beta}_i = d_{i+1} * e_i$
end for
$\alpha_n = d_n + \sigma$.

In practice we write d and e for d_i and e_i.

To derive the implicit shift version we note first that

$$\hat{J} = N_{n-1}^{-1} \cdots N_1^{-1} J N_1 \cdots N_{n-1}$$

and, of more importance, that the 2×2 submatrix

$$\begin{pmatrix} \hat{\beta}_{j-1} & d_j \\ \beta_j e_{j-1} & \beta_j \end{pmatrix}$$

has rank one in exact arithmetic. Thus the multiplier e_j could be computed from $e_j = \beta_j e_{j-1} / \hat{\beta}_{j-1}$ and then the shift σ disappears from the inner loop.

The initial value $e_1 = \beta_1 / (\alpha_1 - \sigma)$ is the only occasion on which σ appears. Indeed if $\alpha_1 \gg \sigma$ then much of the information in σ is irretrievably lost.

The algorithm is sometimes described as chasing the bulge $\beta_j e_{j-1}$ in position $(j+1, j-1)$ down the matrix and off the bottom as $j = 2, 3, \ldots, n-1$ and n. We write δ instead of d to emphasize that these quantities differ from the corresponding ones in the explicit shift algorithm.

LR (implicit shift σ): $\delta = \alpha_1$
if $\delta = \sigma$ **then** exit (fail)
$e = \beta_1 / (\delta - \sigma)$
for $i = 1, n-1$
$\hat{\alpha}_i = \delta + e$
$\hat{\beta}_i = \beta_i - e * (\hat{\alpha}_i - \alpha_{i+1})$
$\delta = \alpha_{i+1} - e$

if $\hat{\beta}_i = 0$ **then** exit (fail)
$e = e * \beta_{i+1}/\hat{\beta}_i$
end for
$\hat{\alpha}_n = \delta$.

The attraction of this algorithm is that it employs nothing but explicit similarities on J.

6. Gram–Schmidt Factors

This section shows that *dqd* could have been discovered independently of *qd*.

To most people the Gram–Schmidt process is the standard way of producing an orthonormal set of vectors $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k$ from a linearly independent set $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_k$. The defining property is that $span(\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_j) = span(\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_j)$, for each $j = 1, 2, \ldots, k$. The matrix formulation of this process is the QR factorization: $F = QR$, where $F = [\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_k]$, $Q = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k]$ and R is $k \times k$ and upper triangular.

The generalization of this process to a pair of vector sets $\{\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_k\}$ and $\{\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k\}$ is so natural that there can be little objection to keeping the name Gram–Schmidt; the context determines immediately whether one or two sets of vectors are involved. Denote by F^* the conjugate transpose of F.

Theorem 1 Let F and G be complex $n \times k$ matrices, $n \geq k$, such that G^*F permits triangular factorization:

$$G^*F = \tilde{L}\tilde{D}\tilde{R},$$

where $\tilde{L}$ and $\tilde{R}$ are unit triangular (left and right), respectively, and $\tilde{D}$ is diagonal. Then there there exist unique $n \times k$ matrices $\tilde{Q}$ and $\tilde{P}$ such that

$$F = \tilde{Q}\tilde{R}, \quad G = \tilde{P}\tilde{L}^*, \quad \tilde{P}^*\tilde{Q} = \tilde{D}.$$

Remark. When $G = F$ the traditional QR factorization is recovered with an unconventional normalization: generally $Q = \tilde{Q}\tilde{D}^{-1/2}$. F^*F permits triangular factorization when, and only when, the leading $k - 1$ columns of F are linearly independent.

Remark. In practice, when $n = k$ and $\tilde{D}$ is invertible one can omit Q and write $F = \tilde{P}^{-1}(\tilde{D}\tilde{R})$, $G = \tilde{P}\tilde{L}^*$ and still call it *the* Gram–Schmidt factorization. The important feature is the uniqueness of $\tilde{Q}$ and $\tilde{P}$. The columns of $\tilde{Q}$ and rows of $\tilde{P}^*$ form a pair of dual bases for the space of n-vectors (columns) and its dual (the row n-vectors). There is no notion of orthogonality or inner product here; $p_i^* q_j = 0$ says that p_i^* annihilates q_j, $i \neq j$.

We omit the proof of the theorem to save space.

The Gram–Schmidt factorization leads directly to the differential qd algorithms. Let us show how.

Corollary 1 Let bidiagonal matrices L and U be given as in Section 2. If UL permits factorization

$$UL = \hat{L}\hat{D}\hat{R} = \hat{L}\hat{U},$$

where $\hat{L}$ and $\hat{R}$ are unit bidiagonal, then there exist unique matrices $\tilde{P}$ and $\tilde{Q}$ such that

$$U = \hat{L}\tilde{P}^*, \quad L = \tilde{Q}\hat{R}, \quad \tilde{P}^*\tilde{Q} = \hat{D}.$$

Remark. In words, apply Gram–Schmidt to the columns of L and the rows of U, in the natural order, to obtain $\hat{U}$ and $\hat{R}$. Then note that $\hat{U} = \hat{D}\hat{R}$

Note that if $u_i = 0$, $i < n$, then UL does not permit triangular factorization. However, the theorem allows $u_n = 0$. When $u_n \neq 0$ then U is invertible and so is $\hat{D}$. In this case we can rewrite the factorization as

$$KL = \hat{D}\hat{R} = \hat{U}, \quad UK^{-1} = \hat{L}, \quad K = \hat{D}\tilde{Q}^{-1}.$$

The matrix K is hidden when we just write $UL = \hat{L}\hat{U}$. However, the identification of K with the Gram–Schmidt process goes only half way in the derivation of the dqd algorithm. The nature of the Gram–Schmidt process shows that $\tilde{P}$ and $\tilde{Q}$ are upper Hessenberg matrices. Fortunately $\tilde{Q}$ and $\tilde{P}$ are *special* Hessenberg matrices that depend on only $2n$ parameters, not $n(n-1)/2$. We are going to show that they may be written as the product of $(n-1)$ simple matrices that are non-orthogonal analogues of plane rotations. That means that L may be changed into $\hat{U}$ and U into $\hat{L}$ by a sequence of simple transformations and neither K, $\tilde{Q}$ nor $\tilde{P}$ need appear explicitly.

Definition 1 A *plane transformer*, in plane (i,j), $i \neq j$, is an identity matrix except for the entries (i,i), (i,j), (j,i) and (j,j). The 2×2 submatrix they define must be invertible.

Let us describe the first minor step in mapping $L \to \hat{U}$, $U \to \hat{L}$. We seek an invertible 2×2 matrix such that

$$\begin{pmatrix} x & z \\ -y & w \end{pmatrix}\begin{pmatrix} 1 & 0 \\ l_1 & 1 \end{pmatrix} = \begin{pmatrix} \hat{u}_1 & 1 \\ 0 & 1 \end{pmatrix},$$

$$\begin{pmatrix} u_1 & 1 \\ 0 & u_2 \end{pmatrix}\begin{pmatrix} w & -z \\ y & x \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \hat{l}_1 & * \end{pmatrix} \cdot det,$$

where $det = xw + yz$ and $*$ may be anything. A glance at the last column of the top equation shows that $z = w = 1$. The 0 in the $(2,1)$ entry on the right shows that $y = l_1$ and the 0 in the $(1,2)$ entry in the second equation shows that $x = u_1$. Thus $det = wx + yz = u_1 + l_1 \equiv \hat{u}_1$. ¿From the $(2,1)$ entry of the second equation we learn that

$$u_2 l_1 = u_2 y = \hat{l}_1 det = \hat{l}_1 \hat{u}_1.$$

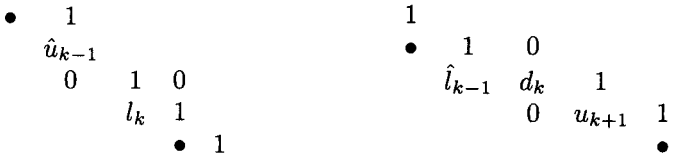

Fig. 2. Active entries

Finally, and of most interest,

$$* = u_2 x/det = u_2 u_1/\hat{u}_1.$$

This is the intermediate quantity d_2 in dqd and we see it here as something that gets carried to the next minor step. If we write $d_1 = u_1$ we obtain the start of the inner loop of dqd:

$$\begin{aligned} \hat{u}_1 &= d_1 + l_1, \\ \hat{l}_1 &= l_1(u_2/\hat{u}_1), \\ d_2 &= d_1(u_2/\hat{u}_1). \end{aligned}$$

The typical minor step is similar. It is instructive to look at the matrices part way through the transformation $L \to \hat{U}$, $U \to \hat{L}$ as shown in Figure 2.

At minor step k the plane transformed is $(k, k+1)$ and the active part of the plane transformer is $\begin{pmatrix} d_k & 1 \\ -l_k & 1 \end{pmatrix}$ on the left and $\begin{pmatrix} 1 & -1 \\ l_k & d_k \end{pmatrix}$ on the right, with $det = \hat{u}_{k+1}$. Finally at the end of minor step $(n-1)$ the trailing 2×2 submatrices are

$$\begin{pmatrix} \hat{u}_{n-1} & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 0 \\ \hat{l}_{n-1} & d_n \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \hat{l}_{n-1} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & d_n \end{pmatrix}.$$

If $d_n \neq 0$ we simply multiply row n on the left by d_n and divide column n on the right by d_n as a final similarity transformation. When $d_n = 0$ the matrices $\hat{L}$ and $\hat{R}$ remain invertible. Thus $\hat{u}_n = d_n \hat{r}_{nn} = 0 \cdot 1 = 0$.

So we have derived the dqd algorithm without reference to qd. Of more significance is the fact that the quantities d_i, $i = 1, \ldots, n$, provide useful information about UL that qd does not reveal and so dqd facilitates the choice of shift.

7. The Meaning of d_i

Theorem 2 Consider L and U as described in Section 2. If U is invertible then the quantities d_i, $i = 1, \ldots, n$, generated by the dqd algorithm applied to L and U satisfy

$$d_i^{-1} = [(UL)^{-1}]_{ii}, \quad i = 1, \ldots, n.$$

Proof. The algorithm may be considered as transforming L to $\hat{U}$ by premultiplications and U to $\hat{L}$ by inverse multiplications on the right as described in the previous section. At the end of the $(k-1)$th plane transformation the situation is as indicated below:

$$G_{k-1}L = \begin{bmatrix} \bullet & \bullet & & & \\ & \hat{u}_{k-1} & 1 & & \\ & 0 & 1 & 0 & \\ & & l_k & 1 & \\ & & & \bullet & \bullet \end{bmatrix}, \quad \begin{bmatrix} \bullet & & & & \\ \bullet & 1 & 0 & & \\ & \hat{l}_{k-1} & d_k & 1 & \\ & & 0 & u_{k+1} & 1 \\ & & & \bullet & \bullet \end{bmatrix} = UG_{k-1}^{-1},$$

where

$$G_{k-1} = \Phi_{k-1}\Phi_{k-2}\cdots\Phi_1, \quad \Phi_i \text{ transforms plane } (i, i+1), \quad G_0 = I_n.$$

The striking fact is that row k of $G_{k-1}L$ and column k of UG_{k-1}^{-1} are singletons. If $\boldsymbol{e}_j$ denotes column j of I_n then

$$\boldsymbol{e}_k^t G_{k-1}L = \boldsymbol{e}_k^t, \qquad \boldsymbol{e}_k d_k = UG_{k-1}^{-1}\boldsymbol{e}_k, \quad 1 \le k \le n.$$

Rearranging these equations yields, for $k = 1, 2, \ldots, n$,

$$\begin{aligned} d_k^{-1} &= (\boldsymbol{e}_k^t G_{k-1})(G_{k-1}^{-1}\boldsymbol{e}_k d_k^{-1}) \\ &= (\boldsymbol{e}_k^t L^{-1})(U^{-1}\boldsymbol{e}_k) = [(UL)^{-1}]_{kk}. \end{aligned}$$

□

In the positive case ($l_i > 0$, $u_i > 0$), UL is diagonally similar to a symmetric positive-definite matrix.

Corollary 2 In the positive case,

$$\left(\sum_{i=1}^{n} d_i^{-1}\right)^{-1} < \lambda_{\min}(UL) \le \min_i d_i.$$

Proof. For any matrix M that is diagonally similar to a positive-definite symmetric matrix

$$\max_j m_{jj} \le \lambda_{\max}[M] < \operatorname{trace}[M].$$

Take $M = (UL)^{-1}$. □

Even in the general case, as $u_n \to 0$, $\min_i |d_i|$ becomes an increasingly accurate approximation to $|\lambda_{\min}|$.

8. Incorporation of Shifts

The algorithms and theorems presented so far serve only as background. LR, QR and qd algorithms are only as good as their shift strategies. In practice one uses qds and dqds, the shifted versions of qd and dqd.

The derivation of dqds(σ) in terms of a Gram–Schmidt process is not obvious. Formally we write $UL - \sigma I = (U - \sigma L^{-1})L = \hat{L}\hat{U}$ and apply the Gram–Schmidt process to the columns of L and the rows of $U - \sigma L^{-1}$ to obtain

$$L = G\hat{R}, \quad U - \sigma L^{-1} = \hat{L}F, \quad FG = \hat{D}.$$

Eliminating G yields

$$L = (G\hat{D}^{-1})(\hat{D}\hat{R}) = F^{-1}\hat{U}, \quad U - \sigma L^{-1} = \hat{L}F.$$

At first sight the new term $-\sigma L^{-1}$ appears to spoil the derivation of F as a product of plane transformers. However, it is not necessary to know all the terms of L^{-1} but only the $(i+1, i)$ entries immediately below the main diagonal. The change from the unshifted case is small. The active parts of the two transformations are given by

$$\begin{pmatrix} d_i & 1 \\ -l_i & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ l_i & 1 \end{pmatrix} = \begin{pmatrix} \hat{u}_i & 1 \\ 0 & 1 \end{pmatrix}, \quad \text{as before,}$$

and the new relation

$$\begin{pmatrix} d_i & 1 \\ \sigma l_i & u_{i+1} - \sigma \end{pmatrix}\begin{pmatrix} 1 & -1 \\ l_i & d_i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \hat{l}_i & d_{i+1} \end{pmatrix} \cdot det.$$

The last row yields

$$det = d_i + l_i = \hat{u}_i, \quad \text{as before,}$$

$$\begin{aligned} \hat{l}_i \cdot det &= \hat{l}_i\hat{u}_i = \sigma l_i + (u_{i+1} - \sigma)l_i = u_{i+1}l_i, \quad \text{as before,} \\ d_{i+1} \cdot det &= -\sigma l_i + (u_{i+1} - \sigma)d_i = u_{i+1}d_i - \sigma\hat{u}_i. \end{aligned}$$

This is dqds(σ).

If one looks at the two matrices part way through the transformations $L \to \hat{U}$, $U - \sigma L^{-1} \to \hat{L}$, the singleton column in the second matrix (from Theorem 2) has disappeared and the relation of d_i to $(UL)^{-1}$ is more complicated.

Theorem 3 Consider L and U as described in Section 2. If U is invertible and $UL - \sigma I$ permits triangular factorization, with $\sigma \neq 0$, then the intermediate quantities d_i, $i = 1, \ldots, n$, generated by dqds(σ) applied to L and U satisfy

$$\frac{1 + \sigma[(\hat{U}UL)^{-1}]_{k,k-1}}{d_k + \sigma} = [(UL)^{-1}]_{k,k}.$$

We omit the proof.

9. Accuracy

The differential qd algorithms dqd and dqds are new to the scene of matrix computations. One feature that makes them attractive is that they seem

to be more accurate than their rivals. In particular, in the positive case, all eigenvalues can be found to high relative accuracy as long as the shifts preserve positivity.

Let us begin with an extreme example.

Example 1. Take $n = 64$, $u_i = 1$, $i = 1, \ldots, 64$, $l_i = 2^{16} = 65536$, $i = 1, \ldots, 63$. Although $\det(LU) = 1$ the smallest eigenvalue is $\mathcal{O}(10^{-304})$.

In just 2 iterations dqd computed $\lambda_{\min}$ to full working precision. In contrast qd returns 0, a satisfactory answer relative to the matrix norm. Yet dqd does preserve the determinant to working accuracy, provided underflow and overflow are absent, while qd, LR and QR do not.

The reason for dqd's accuracy is that $\hat{u}_{64} = d_{64}$ reaches the correct tiny value through 63 multiplications and divisions. There are no subtractions.

A little extra notation is needed to describe the stability results compactly. When there is no need to distinguish l's from u's we follow Rutishauser and speak of a qd array

$$Z = \{u_1, l_1, u_2, l_2, \ldots, l_{n-1}, u_n\}.$$

The right unit for discussing relative errors is the ulp (1 unit in the last place held) since it avoids reference to the magnitudes of the numbers involved. In Example 1 the error in the computed eigenvalue $< \frac{1}{2}$ulp despite 2×63 divisions and multiplications.

Given Z the dqds algorithm in finite precision produces a representable output $\hat{Z}$. We write this $\hat{Z} = fl(\text{dqds}) \cdot Z$. Now we introduce two ideal qd arrays $\vec{Z}$ and $\breve{Z}$ such that, *in exact arithmetic*, dqds with shift σ maps $\vec{Z}$ into $\breve{Z}$. Moreover $\vec{Z}$ is a tiny relative perturbation of Z, and $\hat{Z}$ is a tiny relative perturbation of $\breve{Z}$. See Figure 3.

The proof of the following result may be found in [2].

Theorem 4 In the absence of division by zero, underflow or overflow, the Z diagram commutes and, for all k, $\vec{u}_k$ ($\vec{l}_k$) differs from u_k (l_k) by 3 (1) ulps at most, and $\hat{u}_k$ ($\hat{l}_k$) differs from $\breve{u}_k$ ($\breve{l}_k$) by 2 (2) ulps, at most.

The proof is based on making small changes to Z and $\hat{Z}$ so that the *computed* sequence of d's is exact for $\vec{Z}$ and $\breve{Z}$. There is no requirement of positivity so it is possible to have $\|\hat{Z}\| >>> \|Z\|$. Some people call this a *mixed stability* result because one had to perturb both input and output to get an exact dqds mapping. For example, such mixed accuracy results are the best that can be said about the trigonometric functions in computer systems; the output is within an ulp of the exact trigonometric function of a value within one ulp of the given argument.

Theorem 4 does not guarantee that dqds returns accurate eigenvalues in

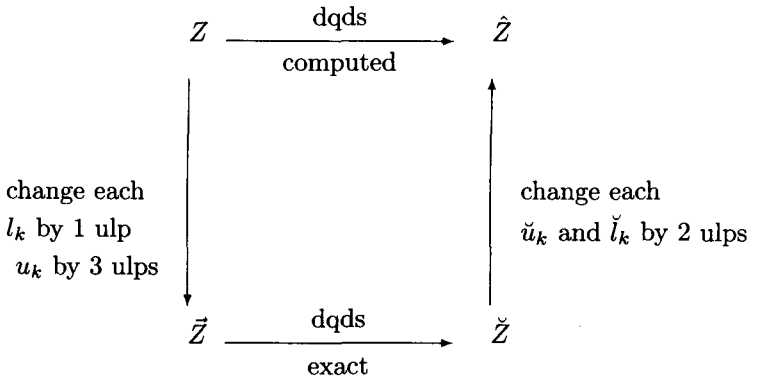

Fig. 3. Effects of roundoff

all cases, even when we only want errors to be small relative to $\|J\|$. We call such accuracy absolute rather than relative:

$$|\lambda_i - \hat{\lambda}_i| < \epsilon\|J\| \quad \text{versus} \quad |\lambda_i - \hat{\lambda}_i| < \epsilon \max\{|\lambda_i|, |\hat{\lambda}_i|\}.$$

In Fernando and Parlett (1994) a corollary to Theorem 4 establishes high relative accuracy for all eigenvalues computed by dqds *in the positive case.* The corollary is stated in terms of singular values but the algorithm computes the squares of those singular values, that is, the eigenvalues of a positive-definite matrix similar to a product LU.

Given these nice results it is natural to seek a conventional backward error analysis that says that $\hat{Z}$ is the exact output of dqds applied to some array $\overset{o}{Z}$, where either $\|Z - \overset{o}{Z}\| < M\epsilon\|Z\|$ or, better, $|u_i - \overset{o}{u}_i| < M\epsilon|u_i|$, $|l_i - \hat{l}_i| < M\epsilon|l_i|$, for all i, which we write as $|Z - \overset{o}{Z}| < M\epsilon|Z|$. Here M is some reasonable constant.

This was one task given to Yao Yang for his doctoral research here and we quote here two of the results from his 1994 paper. Yang's first discovery was an unpleasant surprise. Even in the positive case it is not always true that $|Z - \overset{o}{Z}| < M\epsilon|Z|$. Here is an example.

Example 2. There is no small backward error for dqd.

$$\begin{aligned} n &= 5, \ \sigma = 0, \text{ single precision: } 1 + 10^{-8} \to 1, \\ u &= (1, 1, 1, 1, 1), \\ l &= (10^4, 10^4, 10^4, 10^4, 0). \end{aligned}$$

We omit tiny irrelevant terms in what follows.

$$\begin{aligned} \hat{u} &= (10^4 + 1, 10^4, 10^4, 10^4, 10^{-16} - 10^{-20}), \\ \hat{u} &\text{ exactly } (10^4 + 1, 10^4 + \mathbf{10^{-4}}, 10^4 + 10^{-8}, 10^4 + 10^{-12}, 10^{-16} - 10^{-20}), \\ \hat{l} &= (1 - 10^{-4}, 1, 1, 1, 0), \end{aligned}$$

$$\begin{aligned} \overset{o}{u} &= (0, 1 - 10^{-4}, 1, 1, 1), \\ \overset{o}{l} &= (10^4 + 1, 10^4, 10^4, 10^4, 0). \end{aligned}$$

Thus $u_1 = 1$ must be changed to $\overset{o}{u}_1 = 0$ in order to have dqd$\cdot \overset{o}{Z} = \hat{Z}$. The last 3 steps in computing $\overset{o}{Z}$ are instructive.

1. $\overset{o}{u}_2 = \hat{u}_2 - \overset{o}{l}_2 + \hat{l}_1 = 10^4 - 10^4 + (1 - 10^{-4}) = 1 - 10^{-4}$

whereas the true $\hat{u}_2$ is $10^4 + 10^{-4}$ which would yield the ideal $\overset{o}{u}_2 = 1$.

2. $\overset{o}{l}_1 = \hat{l}_1 \hat{u}_1 / \overset{o}{u}_2 = 10^4/(1 - 10^{-4}) = 10^4 + 1$ instead of 10^4.
3. $\overset{o}{u}_1 = \hat{u}_1 - \overset{o}{l}_1 = (10^4 + 1) - (10^4 + 1) = 0$ instead of 1.

Note that $\overset{o}{u}_1 + \overset{o}{l}_1 = u_1 + l_1$ exactly!

This result shows why there was no backward error analysis in Fernando and Parlett (1994). Further investigation by Yang showed that the fault is in the formulation of the task, not in the algorithm. Recall from Section 2 that associated with any qd array Z are matrices L, U and their products J and J'.

Here is Yang's second result.

Theorem 5 If dqds(σ) maps Z into $\hat{Z}$ in finite-precision arithmetic obeying (9.1) below and if both arrays are positive then there is a unique array $\overset{o}{Z}$ such that in exact arithmetic dqds(σ) maps $\overset{o}{Z}$ into $\hat{Z}$. Moreover the tridiagonal matrices J' and $\overset{o}{J}'$ associated with Z and $\overset{o}{Z}$ satisfy

$$|J' - \overset{o}{J}'| < 5\epsilon |J'|,$$

where ϵ is the roundoff unit.

The inequality is interpreted element-by-element. This is a strong result and consistent with Theorem 4. The amplification factor 5 is a worst-case bound. Contemplation of the proof shows that in most cases the errors in executing dqds(σ) in finite precision can be accounted for by perturbing J's entries (not Z's) by 1 or 2 ulps.

The strength of the result comes from the simplicity of the proof and what makes the proof simple is that, in exact arithmetic, dqds(σ) is equivalent to qds(σ) and qds brings in no intermediate quantities. Thus we may define $\overset{o}{Z}$ by $\overset{o}{Z} = \text{qds}^{-1} \cdot fl(\text{dqds})Z$. Here fl denotes a result obtained with floating-point arithmetic.

The diagram in Figure 4 illustrates the strategy.

The invertibility of qds(σ) is proved by observing that if the output is positive and given then qds(σ) statements may be used in reverse order $(n, n-1, \ldots, 1)$ to recover the unique input.

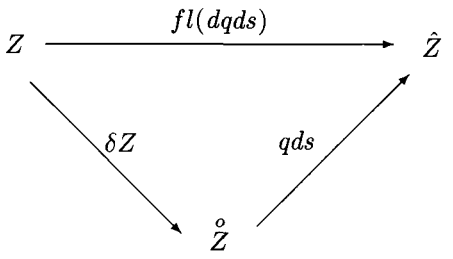

Fig. 4. Definition of $\overset{o}{Z}$

The model of arithmetic assumes the presence of a guard digit so that subtraction has the same relative accuracy as the other fundamental operations.

$$fl(a\square b) = (a\square b)(1+\eta), \tag{9.1}$$

where $\eta \leq \epsilon$, the roundoff unit (or arithmetic precision) which does not depend on a, b or $\square$, and $\square = \pm, *$ or $/$.

Proof of Theorem 5. First we write down the relationships satisfied by $\hat{Z}$. Subscripted variables γ, δ and ϵ denote roundoff quantities as needed by the model (9.1).

$$\begin{aligned}
fl(\text{dqds}) : d_1 &= fl(u_1 - \sigma) = (u_1 - \sigma)(1+\gamma_0'); \\
\textbf{for} \quad i &= 1, n-1 \\
\hat{u}_i &= fl(d_i + l_i) = (d_i + l_i)(1+\epsilon_i) \\
t_i &= fl(u_{i+1}/\hat{u}_i) = u_{i+1}(1+\delta_i)/\hat{u}_i, \ t_i \text{ should stay in a register} \\
\hat{l}_i &= fl(l_i * t_i) = l_i u_{i+1}(1+\delta_i)(1+\delta_i')/\hat{u}_i \\
d_{i+1} &= fl(d_i * t_i - \sigma) = (d_i u_{i+1}(1+\delta_i)(1+\gamma_i)/\hat{u}_i - \sigma)(1+\gamma_i') \\
\textbf{end for} & \\
\hat{u}_n &= d_n.
\end{aligned}$$

By definition $qds \cdot \overset{o}{Z} = \hat{Z}$, exactly, so with $\hat{l}_0 = 0$,

$$\begin{aligned}
\text{qds}: \ \textbf{for} \quad i &= 1, n-1 \\
\hat{u}_i &= \overset{o}{u}_i - \hat{l}_{i-1} + \overset{o}{l}_i - \sigma \\
\hat{l}_i &= \overset{o}{l}_i \overset{o}{u}_{i+1} / \hat{u}_i \\
\textbf{end for} & \\
\hat{u}_n &= \overset{o}{u}_n - \hat{l}_{n-1} - \sigma.
\end{aligned}$$

Of course $\overset{o}{Z}$ is determined in the order $\overset{o}{u}_n, \overset{o}{l}_{n-1}, \overset{o}{u}_{n-1}, \overset{o}{l}_{n-2}, \ldots, \overset{o}{u}_1$, but that

is irrelevant here. Next we eliminate $\hat{u}_i$ and $\hat{l}_i$ using the two sets of equations from *fl(*dqds*)* and qds. In what follows we omit terms that are $\mathcal{O}(\epsilon^2)$.

For $i = 1$

$$\begin{aligned} \overset{o}{u}_1 + \overset{o}{l}_1 - \sigma &= \hat{u}_1 = ((u_1 - \sigma)(1 + \gamma_0') + l_1)(1 + \epsilon_1) \\ &= u_1 + l_1 - \sigma + (u_1 - \sigma)(\gamma_0' + \epsilon_1) + \epsilon_1 l_1. \end{aligned}$$

In the positive case $\lambda_{\min}(J) \leq$ any 'pivot' in elimination, that is, $\sigma < \lambda_{\min}(J) \leq \min_j u_j$. Thus, in the positive case

$$\left| \frac{\overset{o}{u}_1 + \overset{o}{l}_1 - u_1 - l_1}{u_1 + l_1} \right| < \frac{u_1 \cdot 2\epsilon + l_1 \epsilon}{u_1 + l_1} \leq 2\epsilon.$$

For $2 \leq i < n$

$$\begin{aligned} \overset{o}{u}_i + \overset{o}{l}_i &= \hat{u}_i + \hat{l}_{i-1} + \sigma \\ &= (d_i + l_i)(1 + \epsilon_i) + l_{i-1} u_i (1 + \delta_{i-1})(1 + \delta_{i-1}')/\hat{u}_{i-1} + \sigma \\ &= d_{i-1} u_i (1 + \delta_{i-1})(1 + \gamma_{i-1})(1 + \gamma_{i-1}')(1 + \epsilon_i)/\hat{u}_{i-1} \\ &\quad -\sigma(1 + \gamma_{i-1}')(1 + \epsilon_i) \\ &\quad +l_i(1 + \epsilon_i) + l_{i-1} u_i (1 + \delta_{i-1})(1 + \delta_{i-1}')/\hat{u}_{i-1} + \sigma. \end{aligned}$$

Now we can combine the terms involving u_i, noting that

$$\hat{u}_{i-1} = (d_{i-1} + l_{i-1})(1 + \epsilon_{i-1}).$$

So, omitting terms of $\mathcal{O}(\epsilon^2)$, we have

$$\begin{aligned} \overset{o}{u}_i + \overset{o}{l}_i &= u_i + l_i + \frac{d_{i-1}}{d_{i-1} + l_{i-1}} u_i (\delta_{i-1} + \gamma_{i-1} + \gamma_{i-1}' + \epsilon_i - \epsilon_{i-1}) \\ &\quad + \frac{l_{i-1}}{d_{i-1} + l_{i-1}} u_i (\delta_{i-1} + \delta_{i-1}' - \epsilon_{i-1}) + \epsilon_i l_i - \sigma(\gamma_{i-1}' + \epsilon_i). \end{aligned}$$

Now use repeated terms like $\delta_{i-1} - \epsilon_{i-1}$ to simplify the roundoff terms:

$$\begin{aligned} \overset{o}{u}_i + \overset{o}{l}_i &= u_i + l_i + u_i(\delta_{i-1} - \epsilon_{i-1}) \\ &\quad + \frac{d_{i-1}}{d_{i-1} + l_{i-1}} u_i \gamma_{i-1} + \frac{l_{i-1}}{d_{i-1} + l_{i-1}} u_i \delta_{i-1}' \\ &\quad + \left(\frac{d_{i-1}}{d_{i-1} + l_{i-1}} u_i - \sigma \right) (\gamma_{i-1}' + \epsilon_i) + l_i \epsilon_i. \end{aligned} \tag{9.2}$$

Thus, in the positive case, using $\sigma < u_i$,

$$\begin{aligned} |\overset{o}{u}_i + \overset{o}{l}_i - (u_i + l_i)| &< u_i 2\epsilon + u_i \epsilon + l_i \epsilon + \max\left\{ \frac{d_{i-1} u_i}{d_{i-1} + l_{i-1}}, \sigma \right\} 2\epsilon \\ &< u_i 2\epsilon + u_i \epsilon + l_i \epsilon + u_i 2\epsilon = u_i 5\epsilon + l_i \epsilon < 5\epsilon(u_i + l_i). \end{aligned}$$

For the subdiagonal entries we have, almost immediately,

$$\overset{o}{l}_i \overset{o}{u}_{i+1} = \hat{l}_i \hat{u}_i = l_i u_{i+1}(1+\delta_i)(1+\delta_i'), \tag{9.3}$$

so

$$\left| \frac{\overset{o}{l}_i \overset{o}{u}_{i+1} - l_i u_{i+1}}{l_i u_{i+1}} \right| < 2\epsilon$$

in all cases, positive or not. This completes the proof. □

In the course of the proof the general case has been covered. Backward stability is guaranteed in neither a relative sense nor a norm sense. When there is element growth, that is, when $|d_i|$ or $|\hat{u}_i|$ or $|\hat{l}_i|$ greatly exceeds $|l_i|$ or $|u_i|$, then locally (in position i) $\overset{o}{J}'$ differs strongly from J'.

To state the result in terms of arrays treat $diag(M)$ as a linear array and define

$$\begin{aligned} ones &= (1, 1, \ldots, 1), \\ \hat{l} &= (0, \hat{l}_1, \ldots, \hat{l}_{n-1}), \\ d &= (d_1, \ldots, d_n). \end{aligned}$$

Corollary 3 In the general case, in the absence of under/overflow or divide by zero, it is element growth that governs $\overset{o}{J}' - J'$. Entry-by-entry,

$$\begin{aligned} |diag(\overset{o}{J}') - diag(J')| &< \epsilon\{2|u| + |\sigma||ones| + |\hat{l}| + |\hat{u}| + 2|d|\}, \\ |offdiag(\overset{o}{J}') - offdiag(J')| &< 2\epsilon |offdiag(J')|. \end{aligned}$$

Proof. Relation (9.2) in the proof of Theorem 5 holds in the general case. By omitting $\mathcal{O}(\epsilon^2)$ terms we may undo the equation for d_i from *fl*(dqds).

$$\begin{aligned} \frac{d_{i-1}u_i}{d_{i-1}+l_{i-1}} &= \left(\frac{d_i}{1+\gamma_{i-1}'} + \sigma \right) (1+\epsilon_{i-1})/(1+\delta_{i-1})(1+\gamma_{i-1}) \\ &= (d_i + \sigma - \gamma_{i-1}' d_i)(1 - \delta_{i-1} - \gamma_{i-1} + \epsilon_{i-1}) \\ &= (d_i + \sigma)(1 + \mathcal{O}(\epsilon)), \end{aligned}$$

Similarly we undo the equation for $\hat{l}_{i-1}$:

$$\begin{aligned} \frac{l_{i-1}u_i}{d_{i-1}+l_{i-1}} &= \hat{l}_{i-1}(1+\epsilon_{i-1})/(1+\delta_{i-1})(1+\delta_{i-1}') \\ &= \hat{l}_{i-1}(1+\mathcal{O}(\epsilon)). \end{aligned}$$

Now we can rewrite (9.2) in terms of $d_i, \hat{l}_{i-1}$ and $\hat{u}_i$.

$$\begin{aligned} \overset{o}{l}_i + \overset{o}{u}_i &= u_i + l_i + u_i(\delta_{i-1} - \epsilon_{i-1}) + (d_i + \sigma)\gamma_{i-1} + \hat{l}_{i-1}\delta_{i-1}' \\ &\quad + d_i\gamma_{i-1}' + \epsilon_i(d_i + l_i). \end{aligned}$$

So

$$\begin{aligned} |\overset{o}{u}_i + \overset{o}{l}_i - (u_i + l_i)| \quad &< \quad |u_i|2\epsilon + |d_i|\epsilon + |\sigma|\epsilon + |\hat{l}_{i-1}|\epsilon \\ &\quad + |d_i|\epsilon + |\hat{u}_i|\epsilon, \end{aligned}$$

omitting $\mathcal{O}(\epsilon^2)$ terms, as claimed in the corollary. The off-diagonal entries of $\overset{o}{J}$ do enjoy backward stability in a relative sense since the product rhombus rule is preserved to within 2 ulps as shown in (9.3) in the proof of Theorem 5. □

10. Speed

All algorithms of LR type succeed or fail according to the sequence of shifts. The basic algorithms, when all shifts are zero, are just too slow. On the other hand zero is the natural shift when J is singular – unless it provokes element growth.

In practice all these algorithms find one or two eigenvalues at a time, deflate them from the matrix and proceed on the remaining submatrix. so the task, at each step, is to pursue these somewhat contradictory goals.

P1. The shift should approximate an eigenvalue, preferably a small one. The more accurate the better.
P2. The shift should not cost too much to compute.
P3. The shift should not lead to element growth in the transformation.

The positive case. Here P3 may be satisfied by always approximating the smallest eigenvalue from below. In this way positivity is preserved and dqds delivers high accuracy. Moreover the auxiliary quantities $\{d_i\}$ provide upper bounds on $\lambda_{\min}(J)$ that become very tight. In Theorem 3 the complicated quantity $[(\hat{U}UL)^{-1}]_{k,k-1}$ turns out to be positive and thus $\lambda_{\min}[J] < d_k + \sigma$. So,

$$\lambda_{\min}(J) < \min_j (d_j + \sigma).$$

With little extra expense dqds can return both the value of $\min_j d_j$ and the last index k at which the minimum occurs.

When $\sigma = 0$ The lower bound happens to be the Newton approximation to $\lambda_{\min}(J)$ from 0 for the characteristic polynomial of J. It is too expensive and too pessimistic to be a frequent choice except at the start of the algorithm.

The index k is useful in the selection of a cheap and realistic estimate of $\lambda_{\min}(J)$. We say that Z (and J) is in the *asymptotic regime* when $k = n - 1$ or $k = n$. At this point we remind the reader that all the algorithms we consider here may be run from the bottom of the matrix to the top, and so we may assume that $k > n/2$. In the asymptotic regime $\min_j d_j$ is an extremely good estimate of $\lambda_{\min}(J)$ but, of course, is a little too large.

A simple strategy that has satisfactory results in comparison with rival methods is to select a small subarray of Z surrounding row k and compute the Newton approximation to 0 for the polynomial associated with the subarray. This approximation is essentially

$$\left(\sum_{i=k-p}^{\min(n,k+p)} d_i^{-1} \right)^{-1}$$

for $p = 2$ or 3. This expression is modified appropriately when k is close to n. In the current implementation both Z and $\hat{Z}$ are available at the end of a transformation. Consequently, given d_k, it is easy to recompute neighboring d_i from $d_{i+1} = d_i u_{i+1}/(d_i + l_i) - \sigma$ going up or down.

Another strategy was proposed by Rutishauser (1960). If a shift σ causes qds (or dqds) to fail because $\hat{u}_n < 0$ but all $\hat{u}_i > 0$, $i < n$, then $\sigma + \hat{u}_n$ is an extremely good lower bound on $\lambda_{\min}(J)$, good to $\mathcal{O}(|\sigma - \lambda_{\min}|^3)$ asymptotically, see Fernando and Parlett (1994) for a new proof. So the bad transform is rejected and dqds is applied with the good shift.

In order to show that these qd algorithms are worth attention we quote some timing results from Fernando and Parlett (1994). On a test bed of several challenging cases of orders 20 to 100 a dqds code with the above shift strategy was between 4 and 11 times faster than LINPACK codes besides being more accurate. In more recent comparisons with code used in LAPACK (QR-based) routines the dqds program was, on the average, twice as fast.

It is likely that the current shifts will be replaced by better ones soon.

The symmetric indefinite case. Given a T that is symmetric, tridiagonal, but not positive definite we reduce it to the positive-definite case as follows. First compute the lower Gershgorin bound by

$$\begin{aligned}
&bound = \alpha_1 - |\beta_1| \\
&\textbf{for } i = 2, n-1 \\
&\quad bound = \min\{bound, \alpha_i - |\beta_{i-1}| - |\beta_i|\} \\
&\textbf{end for} \\
&bound = \min\{bound, \alpha_n - |\beta_{n-1}|\}.
\end{aligned}$$

Next factor $T + bound \cdot I = LU$, with care, as follows:

$$\begin{aligned}
&u_1 = \alpha_1 + bound \\
&\textbf{for } i = 2, n \\
&\quad l_{i-1} = \beta_i / u_{i-1} \\
&\quad u_i = (\max\{\alpha_i, bound\} - l_{i-1}) + \min\{\alpha_i, bound\} \\
&\textbf{end for.}
\end{aligned}$$

It is because both *bound* and l_{i-1} are positive that we can use max and min

to avoid computing $(big + little) - big$, the perennial danger when adding three quantities.

Now dqds may be applied to the positive case. Note that the eigenvectors are not altered by a shift. So, if eigenvectors are wanted, they may be computed as shown in Section 12 and the eigenvalues of T found afterwards by taking a Rayleigh quotient. This is to avoid subtracting *bound* from the computed eigenvalues that are very close to bound.

The general case. There are several open problems. We should expect to reject transforms from time to time when excessive element growth occurs. It is also possible to compute a dstqd transform (the stationary algorithm) at the same time as dqds for the same access to Z.

In other words, it is feasible to compute $\tilde{L}\tilde{U} = LU - \sigma I$ and $\hat{L}\hat{U} = UL - \sigma I$ at the same time. If σ is not too close to 0 then computation can proceed if either of the pairs $\tilde{L}, \tilde{U}$ and $\hat{L}, \hat{U}$ avoids element growth. It is also possible to apply dqds and dstqd to the reversal of Z, that is, $(u_n, l_{n-1}, u_{n-1}, \ldots)$. Our goal is to push $\min_j |d_j|$ to the closest end of the array.

When excessive growth occurs in $\hat{Z}$ it is not hard to evaluate the recurrence that governs the derivative of $\{p_i\}$ with respect to the shift. Here p_i is the characteristic polynominal of the top i by i submatrix of J. Given p_j' as well as p_j at a bad place one can calculate a new shift that will take the new p_j away from 0.

For each Z there is a bad set $Bad(Z)$ in $\mathbb{C}$ consisting of values that should not be used as shifts for qds. It would be useful to understand something of how $Bad(Z)$ changes under qds; that is, how does $Bad(\hat{Z})$ relate to $Bad(Z)$?

An alternative approach is to develop block versions of these algorithms.

11. Parallel Implementation

The algorithms qds and stqd seem to be irrevocably sequential in nature. In contrast the differential versions are less so.

Let us consider dqds from this point of view. The algorithm may be split into two parts.

Part 1. Compute $d = (d_1, \ldots, d_n)$ via

$$d_1 = u_1 - \sigma, \quad d_{i+1} = d_i u_{i+1}/(d_i + l_i) - \sigma, \quad i = 1, \ldots, n.$$

Part 2. As vector operations on l, u and d compute

$$\hat{u} = d + l, \quad l = (l_1, \ldots, l_{n-1}, 0),$$
$$\hat{l} = l * d^{\uparrow}/\hat{u},$$

where

$$d^{\uparrow} = (d_2, d_3, \ldots, d_n, 0).$$

Part 2 is ideal for vector or parallel processors.

It is interesting that Part 1 may, in principle, be executed in $\mathcal{O}(\log_2 n)$ time on a parallel processor but unfortunately the method seems to be unstable in finite precision. See the interesting paper of Mathias (1994). The technique is called 'parallel prefix' in computer science communities

The idea is to consider each d_i as a ratio p_i/q_i and rewrite the recurrence as

$$\frac{p_{i+1}}{q_{i+1}} = \frac{p_i(u_{i+1} - \sigma) + q_i l_i \sigma}{p_i + q_i l_i}$$

or

$$\begin{pmatrix} p_{i+1} \\ q_{i+1} \end{pmatrix} = \begin{pmatrix} u_{i+1} - \sigma & \sigma l_i \\ 1 & l_i \end{pmatrix} \begin{pmatrix} p_i \\ q_i \end{pmatrix} = M_i \begin{pmatrix} p_i \\ q_i \end{pmatrix}.$$

Note that each 2×2 matrix M_i is known a priori and we can start with $(p_1, q_1) = (u_1 - \sigma, 0)$. Consequently d_i is completely determined by column 1 of

$$N(i) = M_i M_{i-1} \cdots M_1.$$

The problem has been reduced to computing all the partial products indicated above. There is an intriguing way to compute the N's from the M's in $2(\log_2 n)$ parallel steps.

The pattern is indicated in the following diagram in Figure 5. This complicated algorithm can be written compactly in the following form.
Standard MATLAB notation is
$[i : j : k] = (i, i + j, i + 2j, \ldots, k)$; $[i : k] = [i : 1 : k]$.
Let $p = \lceil \log_2 n \rceil$.

Initialize $N(i) = M_i$, $i = 1, 2, \ldots, n$.
for $j = 2^{[1:p]}$, $i = [j : j : n]$; $N(i) = N(i) * N(i - j/2)$;
for $j = 2^{[p-1:-1:1]}$, $i = [3 * j/2 : j : n]$; $N(i) = N(i) * N(i - j/2)$.

When $\sigma = 0$ the M's are all lower triangular and the task is simplified significantly and is well suited to implementation on the CM2 and CM5 massively parallel computers.

The strong potential for parallel implementation lies, not here, but at a coarser level in the computation of eigenvectors once the eigenvalues are known accurately. This is the topic of the next section.

12. Eigenvectors

Suppose that Z $(= L, U)$ is given along with an accurate approximation λ to an eigenvalue of $J = LU$. If λ were exact then

$$(LU - \lambda I)\boldsymbol{v} = 0, \quad \|\boldsymbol{v}\| = 1 \tag{12.1}$$

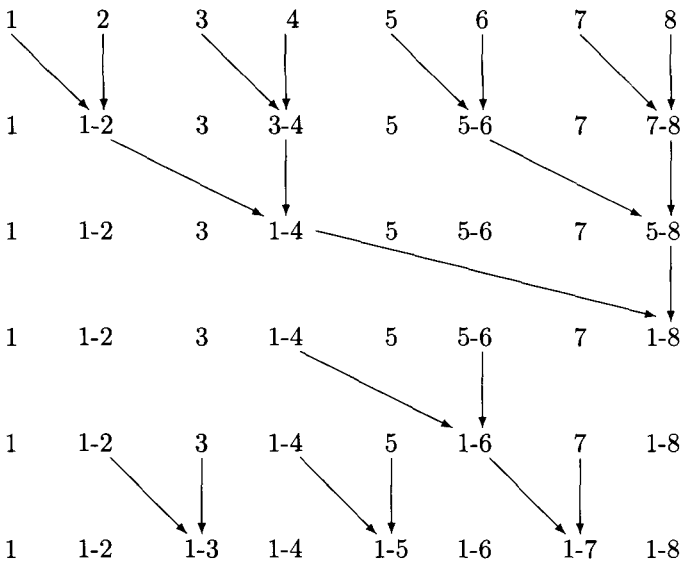

Fig. 5. Parallel prefix

is the equation defining a wanted eigenvector $\boldsymbol{v}$. By applying the stationary qd algorithm dstqd(λ) to Z we obtain, in the absence of breakdown, $\bar{Z}$ such that

$$LU - \lambda I = \bar{L}\bar{U}. \tag{12.2}$$

Since $\bar{L}$ is unit triangular it is invertible and it remains to solve $\bar{U}\boldsymbol{v} = 0$, that is, $\bar{u}_i v_i + v_{i+1} = 0$, $i = 1, \ldots, n-1$. A little care must be exercised to avoid unnecessary overflows and underflows. Let k be the index of a maximal entry of $\boldsymbol{v}$. Then

$$v_k = 1, \quad v_i = -v_{i+1}/\bar{u}_i, \quad i = k-1, \ldots, 1, \quad v_{j+1} = -v_j \bar{u}_j, \quad j = k, \ldots, n-1.$$

Note that $\tilde{u}_n$ is not used and this is appropriate since $\tilde{u}_n$ must vanish in exact arithmetic.

In practice there are two important changes to be made to the simple algorithm given above. First note that the matrix of interest is often not J but $\boldsymbol{\beta}^{-1}J\boldsymbol{\beta}$ for some diagonal $\boldsymbol{\beta}$. So (12.2) changes to

$$\boldsymbol{\beta}^{-1}LU\boldsymbol{\beta} - \lambda I = (\boldsymbol{\beta}^{-1}\bar{L}\boldsymbol{\beta})(\boldsymbol{\beta}^{-1}\bar{U}\boldsymbol{\beta}) \tag{12.3}$$

and

$$\boldsymbol{\beta}^{-1}\bar{U}\boldsymbol{\beta} = bidiag\begin{pmatrix} \bar{u}_1 & \beta_1 & & & \\ & \bar{u}_2 & \beta_2 & & \\ & & \bullet & \bullet & \\ & & & & \beta_{n-1} \\ & & & & \bar{u}_n \end{pmatrix},$$

where

$$\boldsymbol{\beta} = diag(1, \beta_1, \beta_1\beta_2, \ldots, \beta_1\beta_2 \cdots \beta_{n-1}).$$

So the bidiagonal system is

$$\bar{u}_i v_i + \beta_i v_{i+1} = 0, \quad i = 1, \ldots, n-1. \tag{12.4}$$

The second observation is that, even for extremely accurate approximations λ, the final value $\bar{u}_n$ is far from vanishing and the output from (12.4) or (12.3) is too often disappointing. Roundoff error spoils $\bar{U}$.

There is a remedy which is not complicated and appears to be new. In addition the new approach yields very good values for k, the index of a maximal or nearly maximal component of $\boldsymbol{v}$. At the heart of dstqd is the t-recurrence:

$$t_{i+1} = t_i * l_i/(u_i + t_i) - \lambda \tag{12.5}$$

with $t_1 = -\lambda$. Once the t-vector is known $\bar{\boldsymbol{u}} = (\bar{u}_1, \ldots, \bar{u}_n)$ and $\bar{\boldsymbol{l}} = (\bar{l}_1, \ldots, \bar{l}_{n-1}, 0)$ may be found by vector operations: $\bar{\boldsymbol{u}} = \boldsymbol{u} + \boldsymbol{t}$, $\bar{\boldsymbol{l}} = \boldsymbol{u} * \boldsymbol{l}/\bar{\boldsymbol{u}}$. The only thing that distinguishes a true eigenvalue λ from a non eigenvalue is that, in exact arithmetic, $0 = \bar{u}_n = u_n + t_n$. This gives a final value $t_n = -u_n$ and the 2-term recurrence may be solved in reverse order:

$$t_i = u_i/\left(\frac{l_i}{t_{i+1} + \lambda} - 1\right) = \frac{u_i(t_{i+1} + \lambda)}{l_i - t_{i+1} - \lambda}. \tag{12.6}$$

In exact arithmetic with an exact λ the $\boldsymbol{t}$-vectors from the forward and the backward passes will be the same but we use $\overset{o}{t}_i$ to denote the output of the backward pass. In practice $\boldsymbol{t}$ and and $\overset{o}{\boldsymbol{t}}$ are not the same. We choose a k to satisfy

$$|t_k - \overset{o}{t}_k| = \min_j |t_j - \overset{o}{t}_j|.$$

Then we define $\bar{\boldsymbol{t}} = (t_1, \ldots, t_k, \overset{o}{t}_{k+1}, \ldots, \overset{o}{t}_n)$, and compute an approximation $\boldsymbol{x}$ to $\boldsymbol{v}$ from

$$x_k = 1, \quad x_i = -\beta_i x_{i+1}/(t_i + u_i), \quad i = k-1, \ldots, 1, \tag{12.7}$$

$$x_{j+1} = -x_j(\overset{o}{t}_j + u_j)/\beta_j, \quad j = k, \ldots, n-1. \tag{12.8}$$

Note that t's are used going back from k, while $\overset{o}{t}$'s are used going forwards.

An attractive feature of (12.7) and (12.8) is that huge values of t_i and $\overset{o}{t}_j$, including ∞, do not impair the accuracy: From (12.6) $t_i = \infty$ implies $x_i = 0$ for $i = k-1, \ldots, 1$, and $\overset{o}{t}_{j+1} = \infty$ implies $\overset{o}{t}_j + u_j = 0$ implies $x_{j+1} = 0$, for $j = k, \ldots, n-1$. Wherever $x_i = 0$ the adjacent x-values are simply related by $\beta_{i-1}x_{i-1} + \beta_i x_{i+1} = 0$ when $\boldsymbol{\beta}^{-1}LU\boldsymbol{\beta}$ is symmetric and

by $l_{i-1}u_{i-1}x_{i-1}/\beta_{i-1} + \beta_i x_{i+1} = 0$, in general. So, in the symmetric case, the full algorithm is: $x_k = 1$,

$$\begin{aligned} x_i &= \begin{cases} -\beta_{i+1}x_{i+2}/\beta_i, & \text{if } x_{i+1} = 0, \\ -\beta_i x_{i+1}/(t_i + u_i), & \text{otherwise,} \end{cases} \quad i = k-1, \ldots, 1, \\ x_{j+1} &= \begin{cases} -\beta_{j-1}x_{j-1}/\beta_j, & \text{if } x_j = 0, \\ -x_j(\overset{o}{t}_j + u_j)/\beta_j, & \text{otherwise,} \end{cases} \quad j = k, \ldots, n-1. \end{aligned} \tag{12.9}$$

With (12.9) small entries in $\boldsymbol{x}$ are found by multiplication and division, not subtraction.

Let us now justify the choice of k. Suppose that all arithmetic operations are exact but, because λ is not an exact eigenvalue, the $\overset{o}{t}$-recurrence is not justified in using $\overset{o}{t}_n = -u_n$. Consequently there is truncation error and $\boldsymbol{x}$ is not an eigenvector. What can we say about $\boldsymbol{x}$? If $T = \boldsymbol{\beta}^{-1}LU\boldsymbol{\beta}$ is symmetric then the residual

$$\boldsymbol{r} = (T - \lambda I)\boldsymbol{x}$$

vanishes in every component except the kth and at this position $|r_k| = |t_k - \overset{o}{t}_k|$. It is this result that justifies our choice of k. The assumption of symmetry is not essential but it simplifies the exposition. In what follows recall that $\beta_i^2 = l_i u_i$.

There are three cases. We consider r_j and use (12.9).

$$\begin{aligned} j < k. \qquad & \beta_{j-1}x_{j-1} + (l_{j-1} + u_j - \lambda)x_j + \beta_j x_{j+1} \\ &= x_j\left\{-\frac{\beta_{j-1}^2}{\bar{u}_{j-1}} + l_{j-1} + u_j - \lambda - \bar{u}_j\right\}. \end{aligned}$$

Now use (12.5):

$$\begin{aligned} r_j &= x_j\left\{-\frac{l_{j-1}u_{j-1}}{t_{j-1} + u_{j-1}} + l_{j-1} + u_j - \lambda - \left(\frac{t_{j-1}l_{j-1}}{t_{j-1} + u_{j-1}} - \lambda + u_j\right)\right\} \\ &= x_j \cdot 0. \end{aligned}$$

The case $j > k$ is similar but with $\overset{o}{t}_i$ replacing t_i and we omit it.

$$\begin{aligned} j = k. \qquad & \beta_{k-1}x_{k-1} + (l_{k-1} + u_k - \lambda)x_k + \beta_k x_{k+1} \\ &= x_k\left\{-\frac{\beta_{k-1}^2}{\bar{u}_{k-1}} + l_{k-1} + u_k - \lambda - \bar{u}_k\right\}. \end{aligned}$$

A little algebraic simplification together with (12.5) shows that

$$r_k = \left\{\frac{t_{k-1}l_{k-1}}{t_{k-1} + u_{k-1}} - \lambda - \overset{o}{t}_k\right\} = t_k - \overset{o}{t}_k .$$

Roundoff errors make very little difference. More precisely it can be shown

that with appropriately chosen 2 ulp perturbations to $L, U, t, \overset{o}{t}, \boldsymbol{x}$, but not to λ, the equation corresponding to $(T - \lambda I)\boldsymbol{x} = \boldsymbol{e}_k(t_k - \overset{o}{t}_k)$ holds exactly. Here $\boldsymbol{e}_k$ is column k of I.

The significance of this result is that when λ is a very accurate approximation the quantities t_k and $\overset{o}{t}_k$ are quite often less than $\epsilon\|T\|$ and their difference is even smaller. Here ϵ is the precision of the arithmetic unit. Thus we compute vectors $\boldsymbol{x}$ whose residual norms $\|(T - \lambda I)\boldsymbol{x}\|/\|\boldsymbol{x}\|$ are significantly less than $\epsilon\|T\|$. This never happens with standard inverse iteration (i.e. TINVIT in the LINPACK package).

The cost for $\boldsymbol{x}$ from (12.9) appears to be $3n$ divisions (n for $\boldsymbol{t}$, n for $\overset{o}{\boldsymbol{t}}$, n for $\boldsymbol{x}$) but for vector calculations dstqd may be rewritten so that the n divisions in (12.9) become n multiplications: $\bar{l}_i = \beta_i/\bar{u}_i$, $t_{i+1} = \bar{l}_i l_i t_i - \lambda$, $x_i = -\bar{l}_i x_{i-1}$. By way of comparison standard inverse iteration needs n calls for random numbers plus $2n$ divisions for a vector. So the new method is no slower than standard inverse iteration.

When two eigenvalues differ by about $\sqrt{\epsilon}\,||LU||$ the vectors $\boldsymbol{x}$ computed by (12.9) need to be refined by another step of inverse iteration in order to obtain eigenvectors orthogonal to working accuracy.

When m eigenvalues of T are very close (differing by n ulps, say) we can take care to pick m different values of k in order to try to produce outputs that are orthogonal. There is no need to perturb computed eigenvalues that are equal to working precision. This careful choice of k suffices in many but not all cases of clusters. Fortunately there is an extra modification of the method that takes care of the difficult cases by using appropriate submatrices of T. Since this modification is independent of qd algorithms it will not be described here, but see Parlett (1994).

The method described in this section is 'embarrassingly' parallel. Each processor is assigned a copy of Z and one or more eigenvalues. No communication is needed between processors.

13. Singular Values of Bidiagonals

Let

$$B = bidiag\begin{pmatrix} & b_1 & & b_2 & & \bullet & & b_{n-1} & \\ a_1 & & a_2 & & \bullet & & \bullet & & a_n \end{pmatrix}.$$

The goal is to find B's singular values $\sigma_1, \ldots, \sigma_n$. Since $\{\sigma_i^2\}$ is the eigenvalue set of B^tB this is the eigenvalue problem of a positive-semidefinite symmetric tridiagonal with the constraint that B^tB is not to be formed explicitly. Note that B^tB is singular if, and only if, $a_i = 0$ for one or more i values. In these cases B^tB is a direct sum of smaller tridiagonals and the 0 singular values may be deflated by applying qd or dqd independently to the appropriate submatrices as will be explained below. If some $b_i = 0$ the reduction to

a direct sum is immediate and requires no arithmetic effort. Consequently there is no loss of generality in concentrating on the generic case: $a_i \neq 0$, $i = 1, \ldots, n$, $b_j \neq 0$, $j = 1, \ldots, n-1$.

We forsake symmetry and formally put B^tB in the J-matrix format.

Define

$$\Delta = diag(1, \pi_1, \pi_1\pi_2, \ldots, \ldots, \pi_1 \cdots \pi_{n-1}), \quad \pi_i = a_i b_i.$$

It is readily verified that

$$\Delta B^t B \Delta^{-1} = J = LU,$$

where

$$L = bidiag\begin{pmatrix} 1 & & 1 & & \bullet & & \bullet & & 1 \\ & b_1^2 & & b_2^2 & & \bullet & & b_{n-1}^2 & \end{pmatrix},$$

$$U = bidiag\begin{pmatrix} & 1 & & 1 & & \bullet & & 1 & \\ a_1^2 & & a_2^2 & & \bullet & & \bullet & & a_n^2 \end{pmatrix},$$

and so we define $l_i = b_i^2$, $u_i = a_i^2$. Thus the singular-value problem leads to the positive case and dqds may be used to find the σ_i^2 in increasing order.

One begins with dqd (not dqds) in order to capture any tiny singular values and to make use of the lower and upper bounds in Corollary 2 to Theorem 2.

The only blemish is that the original data have been squared and thus the domain of application is smaller than for QR techniques since we must avoid overflow and underflow. However, there is an algorithm oqd described in Fernando and Parlett (1994) that has the same range as QR but the advantages of qd. The price that oqd pays for the extended range is that it must take a square root in the inner loop in the same way that QR techniques do. The response to this choice is easy: If exponent range permits then square up and use dqds, otherwise use oqd.

In order to find the right singular vector for σ_i^2 we apply the stationary algorithm dstqds to L, U with shift σ_i^2 to obtain $\tilde{L}$ and $\tilde{U}$ satisfying

$$\tilde{L}\tilde{U} = \Delta(B^tB - \sigma_i^2 I)\Delta^{-1}$$

whence

$$B^tB - \sigma_i^2 I(\Delta^{-1}\tilde{L}\Delta)(\Delta^{-1}\tilde{U}\Delta)$$

and the techniques of Section 12 may be envoked.

REFERENCES

J. Demmel and W. Kahan (1990), 'Accurate singular values of bidiagonal matrices', *SIAM J. Sci. Sta. Comput.* **11**, 873–912.

K. V. Fernando and B. N. Parlett (1994), 'Accurate singular values and differential qd algorithms', *Numerische Mathematik* **67**, 191–229.

P. Henrici (1958), 'The quotient-difference algorithm', *Nat. Bur. Standards Appl. Math. Series* **19**, 23–46.

R. Mathias (1994), 'The instability of parallel prefix matrix multiplication', preprint, Dept. of Mathematics, College of William and Mary, Williamsburg, VA 23187.

B. N. Parlett (1992), 'Reduction to tridiagonal form and minimal realizations', *SIAM J. on Matrix Analysis and Applications* **13**, 567–593.

B. N. Parlett (1994) 'Orthogonal eigenvectors for symmetric tridiagonals', to appear.

H. Rutishauser (1954), 'Der Quotienten-Differenzen-Algorithmus', *Z. Angew. Math. Phys.* **5**, 233–251.

H. Rutishauser (1958), 'Solution of eigenvalue problems with the LR-transformation', *Nat. Bur. Standards Appl. Math. Series* **49**, 47–81.

H. Rutishauser (1960), 'Uber eine kubisch konvergente Variante der LR-Transformation', *Z. Angew. Math. Mech.* **11**, 49–54.

H. Rutishauser (1990), *Lectures on Numerical Mathematics*, Birkhäuser, Boston.

Yao Yang (1994) 'Backward error analysis for dqds', submitted for publication.