A REAL TRIPLE DQDS ALGORITHM FOR THE NONSYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEM [*]

CARLA FERREIRA[†] AND BERESFORD PARLETT[‡]

Abstract. The paper discusses the following topics: attractions of the real tridiagonal case, relative eigenvalue condition number for matrices in factored form, *dqds*, *triple dqds*, error analysis, new criteria for splitting and deflation, eigenvectors of the balanced form, twisted factorizations and generalized Rayleigh quotient iteration. We present our fast real arithmetic algorithm and compare it with alternative published approaches.

Key words. LR, *dqds*, unsymmetric tridiagonal matrices, balanced form, twisted factorizations

AMS subject classifications. 65F15

1. Introduction.

The *dqds* algorithm was introduced in 1994 in [9] as a fast and extremely accurate way to compute all the singular values of a bidiagonal matrix B. This algorithm implicitly performs the Cholesky LR iteration on the tridiagonal matrix $B^T B$ and it is used in LAPACK. However the *dqds* algorithm can also be regarded as executing, implicitly, the LR algorithm applied to any tridiagonal matrix with 1's on the superdiagonal. Our interest here is in real unsymmetric matrices which may, of course, have some complex eigenvalues. In contrast to the QR algorithm, the LR algorithm preserves tridiagonal form and this feature makes *dqds* attractive. It is natural to try to retain real arithmetic and yet permit complex conjugate pairs of shifts. Our analogue of the *double shift* QR algorithm of J. G. F. Francis [14] is the *triple step dqds* algorithm. We explain why three steps are needed. However the main goal of this paper is to derive our implicit implementation (*3dqds*) of this 3-steps process which relies on the *implicit L* analogue of the *implicit Q* theorem. See Section 3.4.1 and Theorem 3.1.

In order to focus on our *3dqds* algorithm we assume an extensive background for our reader. The unsymmetric eigenvalue problem can be almost ill-posed and such cases are not easily apparent. A tridiagonal matrix requires so little storage that it seems feasible to compute approximate eigenvalues together with an indication of the number of digits that are robust in the presence of computer arithmetic. We decided to provide relative condition numbers (for factored forms) for each computed eigenvalue even when the user does not request it. The extra cost, in storage and arithmetic operations is surprisingly low, $2n$ storage and $\mathcal{O}(n)$ computing. See Section 7.2 for details. We omit any history of the contributions to the field, even the seminal work of H. Rutishauser who invented the *qd* algorithm and the LR algorithm. We must however mention that he also discovered the so-called differential form of *qd* but did not appreciate its accuracy and never published it. That understanding came much later in the computation of singular values of bidiagonal matrices. See [9]. We do describe the double LR algorithm for complex conjugate shifts because of its relation to our *triple dqds* algorithm. We say nothing about the need for an eigensolver devoted to tridiagonal matrices because that issue is covered admirably by Bini, Gemignani, and Tisseur in [1]. We do give pseudocode for a complete program but hope it will not produce distractions from our main concern, the *3dqds* algorithm. We provide error analyses for both *dqds* and *3dqds*.

[*] The research of the first author was partially financed by Portuguese Funds through FCT (Fundação para a Ciência e a Tecnologia) within the Projects UIDB/00013/2020 and UIDP/00013/2020.

[†]Centro de Matemática, Universidade do Minho, 4710-057 Braga (caferrei@math.uminho.pt).

[‡]Department of Mathematics and the Computer Science Division of the Electrical Engineering and Computer Science Department, University of California, Berkeley, California 94720 (parlett@math.berkeley.edu).

A novel feature of our approach is the usefulness of keeping matrices in factored form. We also acknowledge the preliminary work on this problem by Z. Wu in [39].

We do not follow Householder conventions except that we reserve capital Roman letters for matrices. Section 2 describes other relevant methods, Section 3 presents standard, but needed, material on LR, *dqds*, single and double shifts and the implicit L theorem. Section 4 develops our *3dqds* algorithm, Section 5 is our error analysis, Section 6 our splitting, deflation and shift strategy.

Section 7 analyzes applications of factored forms - the computation of eigenvectors using twisted factorizations of the balanced form, relative condition numbers and the generalized Rayleigh quotient iteration. Finally, Section 8 presents our numerical tests using MATLAB and Section 9 gives our conclusions.

2. Other methods relevant to *3dqds*.

2.1. 2 steps of LR = 1 step of QR. For a symmetric positive definite tridiagonal matrix 2 steps of the LR (Cholesky) algorithm produces the same matrix as 1 step of the QR algorithm. Less well known is the article by H. Xu [40] which extends this result when the symmetric matrix is not positive definite. The catch here is that the LR transform, if it exists, does not preserve symmetry. The remedy is to regard similarities by diagonal matrices as "trivial", always available, operations. Indeed, diagonal similarities cannot introduce zeros into a matrix. So, when successful, 2 steps of LR are diagonally similar to one step of QR. Even less well known is a short paper by J. Slemons [32] showing that for a tridiagonal matrix, not necessarily symmetric, 2 steps of of LR are diagonally equivalent to 1 step of HR, see [2]. Note that when symmetry disappears then QR is out of the running because it does not preserve the tridiagonal property.

The point of listing these results is to emphasize that 2 steps of LR gives twice as many shift opportunities as 1 step of QR or HR. Thus convergence can be more rapid with LR (or *dqds*) than with QR or HR. This is one of the reasons that *dqds* is faster than QR for computing singular values of bidiagonals. This extra speed is an additional bonus to the fundamental advantage that *dqds* delivers high relative accuracy in all the singular values. The one drawback to *dqds*, for bidiagonals, is that the singular values must be computed in monotone increasing order; QR allows the singular values to be found in any order.

In our case, failure is always possible and so there is no constraint on the order in which eigenvalues are found. The feature of having more opportunities to shift leads us to favor *dqds* over QR and HR. See the list of other methods which follows. We take up the methods in historical order and consider only those that preserve tridiagonal form.

2.2. Cullum's complex QR algorithm. As part of a program that used the Lanczos algorithm to reduce a given matrix to tridiagonal form in [4], Jane Cullum used the fact that an unsymmetric tridiagonal matrix may always be balanced by a diagonal similarity transformation [18]. She then observed that another diagonal similarity with 1 or i produces a symmetric, but complex, tridiagonal matrix to which the (complex) tridiagonal QR algorithm may be applied. The process is not backward stable because the relation

$$\cos^2 \tau + \sin^2 \tau = 1$$

is not a constraint on $|\cos \tau|$ and $|\sin \tau|$ when τ is complex. Despite the possibility of breakdown the method proved satisfactory in practice. We have not used it in our comparisons because we are persuaded by 2.1 that it is outperformed by the complex *dqds* algorithm, described below.

2.3. Liu's HR algorithm. In [15] Alex Liu found a variation on the HR algorithm of Angelika Bunse-Gerstner that, in exact arithmetic, is guaranteed not to breakdown - but the price is a temporary increase in bandwith. This procedure has only been implemented in MAPLE and we do not include it in our comparison.

2.4. Complex *dqds*. In his thesis David Day [5] developed a Lanczos-style algorithm to reduce a general matrix to tridiagonal form and, as with Jane Cullum, needed a suitable algorithm to compute its eigenvalues. He knew of the effectiveness of *dqds* in the symmetric positive definite case and realized that *dqds* extends formally to any tridiagonal that admits triangular factorization. The code uses complex arithmetic because of the possible presence of complex conjugate pairs of eigenvalues.

We compare our real *3dqds* algorithm with its explicit version - the three steps of *dqds* are computed explicitly in complex arithmetic - in a more sophisticated version of David Day's complex *dqds* code.

2.5. Ehrlich-Aberth algorithm. This very careful and accurate procedure was presented by Bini, Gemignani and Tisseur in [1]. It finds the zeros of the characteristic polynomial $p(\cdot)$ and exploits the tridiagonal form to evaluate $p'(z)/p(z)$ for any z. The polynomial solver improves a full set of approximate zeros at each step. Initial approximations are found using a divide-and-conquer procedure that delivers the eigenvalues of the top and bottom halves of the matrix T. The quantity $p'(z)/p(z)$ is evaluated indirectly as $\left[\text{trace}(zI-T)^{-1}\right]$ using a QR factorization of $zI-T$. Since T is not altered there is no deflation to assist efficiency. Very careful tests exhibit the method's accuracy - but it is very slow compared to *dqds*-type algorithms.

3. LR and *dqds*. The reader is expected to have had some exposure to the QR and/or LR algorithms so we will be brief.

3.1. LU factorization. Any $n \times n$ matrix A permits unique triangular factorization $A = LD\widetilde{U}$ where L is unit lower triangular, D is diagonal, $\widetilde{U}$ is unit upper triangular, if and only if the leading principal submatrices of orders $1, \dots, n-1$ are nonsingular.

In this paper we follow common practice and write $U = D\widetilde{U}$ so that the "pivots" (entries of D) lie on U's diagonal. Throughout this paper any matrix L is unit lower triangular while U is simply upper triangular.

3.2. LR transform with shift. Note that U is "right" triangular and L is "left" triangular and this explains the standard name LR. For any shift σ let

$$A - \sigma I = LU, \tag{3.1}$$

$$\widehat{A} = UL + \sigma I. \tag{3.2}$$

Then $\widehat{A}$ is the LR(σ) transform of A. Note that

$$\widehat{A} = L^{-1}(A - \sigma I)L + \sigma I = L^{-1}AL.$$

We say that the shift is restored (in contrast to *dqds* - see below). The LR algorithm consists of repeated LR transforms with shifts chosen to enhance convergence to upper triangular form. For the theory see [29, 30, 36, 37].

In contrast to the well known QR algorithm, the LR algorithm can breakdown and can suffer from element growth, $\|L\| >> \|A\|$, $\|U\| >> \|A\|$. However LR preserves the banded form of A while QR does not (except for the Hessenberg form).

When a matrix A is represented by its entries then the shift operation $A \longrightarrow A - \sigma I$ is trivial. When a matrix is given in factored form the shift operation is not trivial.

3.3. The *dqds* algorithm. From now on we focus on tridiagonal matrices in J-*form* which means that entries $(i, i+1)$ are all 1, $i = 1, \ldots, n-1$. Any tridiagonal matrix $C = \text{tridiag}(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c})$ that does not split (unreduced), $b_i c_i \neq 0$, is diagonally similar to a J-form. Entries $(i+1, i)$ equal $b_i c_i$. Throughout this paper all J matrices have this form. See [11, Section 2.2] on representations of tridiagonals.

If $J - \sigma I$ permits triangular factorization

$$J - \sigma I = LU$$

then L and U must have the following form

$$L = \begin{bmatrix} 1 & & & & \\ l_1 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-2} & 1 & \\ & & & l_{n-1} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} u_1 & 1 & & & \\ & u_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & u_{n-1} & 1 \\ & & & & u_n \end{bmatrix}. \tag{3.3}$$

It is an attractive feature of LR that

$$UL = \widehat{J}$$

is also of J-form. Thus the parameters l_i, $i = 1, \ldots, n-1$, and u_j, $j = 1, \ldots, n$, determine the matrices L and U above and implicitly define two tridiagonal matrices LU and UL.

The *qds* algorithm is equivalent to the LR algorithm but only the factors L, U are formed, not the J matrices. The *progressive* transformation is from L, U to $\widehat{L}, \widehat{U}$,

$$\widehat{L}\widehat{U} = UL - \sigma I. \tag{3.4}$$

Notice that the shift is not restored and so $\widehat{U}\widehat{L}$ is not similar to UL,

$$\widehat{U}\widehat{L} = \widehat{L}^{-1}(UL)\widehat{L} - \sigma I. \tag{3.5}$$

Equating entries in each side of equation (3.4) gives

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

The algorithm *qds* fails when $\widehat{u}_i = 0$ for some $i < n$. When $\sigma = 0$ we write simply *qd*, not *qds*.

In 1994 a better way was found to implement *qds*(σ). These are *called differential qd* algorithms. See [22] for more history. This form uses an extra variable d but never forms matrix products.

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

By definition, *dqd*=*dqds*(0). In practice we choose to compute, and store, $\widehat{u}_i$ and $\widehat{l}_i$ separately from u_i and l_i. This allows us to reject a transform, choose a new shift, and proceed smoothly to another step. Only when the transform is accepted will we write $\widehat{u}_i$ and $\widehat{l}_i$ over u_i and l_i.

A word on terminology. In Rutishauser's original work $q_i = u_i$, $e_i = l_i$; and the q_i's were certain *quotients* and the e_i's were called *modified differences*. In fact the *qd* algorithm led to the LR algorithm, not vice-versa. The reader can find more information concerning *dqds* in [22, 24]

One virtue of the *dqds* and QR transforms is that they work on the whole matrix so that large eigenvalues are converging near the top, albeit slowly, while the small ones are being picked off at the bottom.

We summarize some advantages and disadvantages of the factored form.

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, for instance, the (i, i) entry of J is given by $l_{i-1} + u_i$ implicitly but $fl(l_{i-1} + u_i)$ explicitly.
2. Singularity of J is detectable by inspection when L and U are given, but only by calculation from J. So, LU reveals singularity, J does not.
3. LU defines the eigenvalues better than J does (usually). There is more on this in [7].
4. Solution of $Jx = b$ takes half the time when L and U are available.

Disadvantages of the factored form

The mapping $J, \sigma \mapsto L, U$ is not everywhere defined for all pairs J, σ and can suffer from element growth. This defect is not as serious as it was when the new transforms were written over the old ones. For tridiagonals we can afford to double the storage and map L, U into different arrays $\widehat{L}, \widehat{U}$. Then we can decide whether or not to accept $\widehat{L}, \widehat{U}$ and only then would L and U be overwritten. So the difficulty of excessive element growth has been changed from disaster to the non-trivial but less intimidating one of, after rejecting a transform, choosing a new shift that will not spoil convergence and will not cause another rejection.

Now we turn to our main question of *dqds*(σ): how can complex shifts be used without having to use complex arithmetic? This question has a beautiful answer for QR and LR iterations.

3.4. Implicit shifted LR for J matrices.

3.4.1. Double shift LR algorithm. We use the J, L and U notation from the previous section. Consider two steps of the LR algorithm with shifts σ_1 and σ_2,

$$\begin{aligned} J_1 - \sigma_1 I &= L_1 U_1 \\ J_2 &= U_1 L_1 + \sigma_1 I \\ J_2 - \sigma_2 I &= L_2 U_2 \\ J_3 &= U_2 L_2 + \sigma_2 I. \end{aligned} \tag{3.6}$$

Then, with matrices $\boldsymbol{\mathcal{L}} = L_1 L_2$ and $\boldsymbol{\mathcal{U}} = U_2 U_1$, we have

$$J_3 = \boldsymbol{\mathcal{L}}^{-1} J_1 \boldsymbol{\mathcal{L}} \tag{3.7}$$

and the triangular factorization

$$\begin{aligned} \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}} &= L_1(J_2 - \sigma_2 I)U_1 = L_1(U_1 L_1 + \sigma_1 I - \sigma_2 I)U_1 \\ &= L_1 U_1 \left[L_1 U_1 + \sigma_1 I - \sigma_2 I\right] = (J_1 - \sigma_1 I)(J_1 - \sigma_2 I) \\ &= J_1^2 - (\sigma_1 + \sigma_2)J_1 + \sigma_1\sigma_2 I =: M. \end{aligned} \tag{3.8}$$

An important observation from (3.8) is that column 1 of M is proportional to column 1 of $\boldsymbol{\mathcal{L}}$,

$$M\boldsymbol{e}_1 = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}\boldsymbol{e}_1 = \boldsymbol{\mathcal{L}}\boldsymbol{e}_1 u_{11}, \quad u_{11} = m_{11}.$$

According to the following theorem, matrix $\boldsymbol{\mathcal{L}}$ is determined by its first column and we can compute J_3 from J_1 without using J_2.

Theorem 3.1. [Implicit L theorem] *If H_1 and H_2 are unreduced upper Hessenberg matrices and $H_2 = L^{-1}H_1L$, where L is unit lower triangular, then H_2 and L are completely determined by H_1 and column 1 of L.*

We omit the proof and refer to [11, pp. 66–68].

So the application to J_1 and J_3, using (3.7), is to choose column 1 of $\boldsymbol{\mathcal{L}}$ (which has only three nonzero entries since J_1 is tridiagonal and J_1^2 is pentadiagonal) to be

$$\mathcal{L}_1 = I + \boldsymbol{m}_1\boldsymbol{e_1}^T \tag{3.9}$$

where $\boldsymbol{m}_1 = \begin{bmatrix}0 & m_{21}/m_{11} & m_{31}/m_{11} & 0 & \dots & 0\end{bmatrix}^T$ and perform a first explicit similarity transform on J_1,

$$\mathcal{L}_1^{-1}J_1\mathcal{L}_1 =: K.$$

Observe that K is not tridiagonal. In the 6×6 case

$$K = \begin{bmatrix} x & 1 & & & & \\ x & x & 1 & & & \\ + & x & x & 1 & & \\ + & & x & x & 1 & \\ & & & x & x & 1 \\ & & & & x & x \end{bmatrix}. \tag{3.10}$$

Next we apply a sequence of elementary similarity transformations such that each transformation pushes the 2×1 bulge one row down and one column to the right. Finally the bulge is chased off the bottom to restore the J-form. In exact arithmetic, the implicit L theorem ensures that this technique of *bulge chasing* gives

$$J_3 = (\mathcal{L}_1 \dots \mathcal{L}_{n-1})^{-1}J_1(\mathcal{L}_1 \dots \mathcal{L}_{n-1}) \quad \text{and} \quad \boldsymbol{\mathcal{L}} = \mathcal{L}_1 \dots \mathcal{L}_{n-1}.$$

In section 4.1 below we will see the details on $\mathcal{L}_j$, $j = 2, \dots, n-1$.

If matrix J_1 and shifts σ_1 and σ_2 are real then factors L_1, U_1, L_2, U_2 and matrices J_2, J_3 will all be real. Now suppose that J_1 is real and σ_1 is complex. Then, by (3.8), J_3 will be real if, and only if, $\sigma_2 = \overline{\sigma}_1$. The reason is that M is real,

$$M = J_1^2 - 2(\Re\sigma_1)J_1 + |\sigma_1|^2 I,$$

so that $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ are real and J_3 is the product of real matrices. Note however that J_2, and factors L_2, U_2, will be complex, given that L_1, U_1 and shifts σ_1, σ_2 are all complex. As we have described above, it is possible to skip this complex matrix J_2 and go straight from real J_1 to real J_3. So, in the case of complex eigenvalues (which for real matrices occur in complex conjugate pairs) we will be able to apply a complex conjugate pair of shifts implicitly and avoid complex arithmetic. Recall that we are seeking an algorithm that uses only real arithmetic and converges to real Schur form.

3.4.2. Connection to *dqds* algorithm. In figure 3.1 we examine the two steps of the LR transform derived in the previous section but with a significant difference. Instead of J_1 being an arbitrary matrix in J-form, we assume that it is given to us in the form U_0L_0. A different way of introducing this factorization is saying that our initial matrix is J_0 (not J_1) and we always consider an additional unshifted LR step for constructing real factored forms

$$J_0 = L_0U_0 \quad \text{and} \quad J_1 = U_0L_0$$

so that *dqds* starts with factors L_0, U_0. The dqds algorithm can not start with J_1 unless its UL factorization is available.

$$\begin{tikzcd}
 & J_1 \arrow[rr, "\mathrm{LR}(\sigma_1)"] \arrow[dr, "-\sigma_1"] & & J_2 \arrow[rr, "\mathrm{LR}(\sigma_2)"] \arrow[dr, "-\sigma_2"] & & J_3 \arrow[dr] & \\
L_0, U_0 \arrow[ur] \arrow[rr, "dqds(\sigma_1)"'] & & L_1, U_1 \arrow[ur, "\sigma_1"] \arrow[rr, "dqds(\sigma_2-\sigma_1)"'] & & L_2, U_2 \arrow[ur, "\sigma_2"] \arrow[rr, "dqds(-\sigma_2)"'] & & L_3, U_3
\end{tikzcd}$$

FIGURE 3.1. *Implicit two steps of LR and three steps of dqds.*

The crucial observation in Figure 3.1 is that the implicit LR algorithm forms only the J matrices while *dqds*, on the bottom line, works only on the factors L,U. So with LR the only J matrix which is skipped by an implicit double step is J_2 and we go from $J_1 = U_0L_0$ to $J_3 = L_3U_3$. The *dqds* algorithm cannot stop with L_2, U_2 because it is only when we take the product U_2L_2 and add back the shift σ_2 that we get the matrix J_3; it requires a third step to obtain L_3, U_3 which define J_3. The *triple dqds* algorithm will skip the factors L_1, U_1, L_2, U_2 and will go from L_0, U_0 to L_3, U_3 performing implicitly three *dqds* steps.

Here is another way to describe the diagonal arrows in Figure 3.1 for the relation between double shift LR and *triple dqds*:

$$\begin{array}{rlcl}
\multicolumn{2}{c}{\textbf{double shift LR}} & & \textbf{\textit{triple dqds}} \\
J_1 = & U_0L_0 & \multirow{2}{*}{$\Big\}$} & \multirow{2}{*}{$L_1U_1 = U_0L_0 - \sigma_1 I$} \\
J_1 - \sigma_1 I = & L_1U_1 & & \\
J_2 = & U_1L_1 + \sigma_1 I & \multirow{2}{*}{$\Big\}$} & \multirow{2}{*}{$L_2U_2 = U_1L_1 - (\sigma_2 - \sigma_1)I$} \\
J_2 - \sigma_2 I = & L_2U_2 & & \\
J_3 = & U_2L_2 + \sigma_2 I & & L_3U_3 = U_2L_2 - (-\sigma_2)I
\end{array} \tag{3.11}$$

So the similarity (3.7) corresponds to

$$L_3U_3 = \boldsymbol{\mathcal{L}}^{-1}(U_0L_0)\boldsymbol{\mathcal{L}} \tag{3.12}$$

and, in contrast to a single *dqds* step, a *triple dqds* step (implicit) restores the shifts.

Observe that in the *triple dqds* step (3.11) we find factors L_3, U_3 such that $J_3 = L_3U_3$ and these factors (different factors) would only occur in LR in the following step with a new shift σ_3,

$$\begin{array}{rlcl}
J_3 = & U_2L_2 + \sigma_2 I & \multirow{2}{*}{$\Big\}$} & \multirow{2}{*}{$L_3U_3 = U_2L_2 - (\sigma_3 - \sigma_2)I$} \\
J_3 - \sigma_3 I = & L_3U_3 & & \\
J_4 = & U_3L_3 + \sigma_3 I. & &
\end{array} \tag{3.13}$$

So to make explicit *dqds* equivalent to LR with shifts σ_i and σ_{i+1} it is necessary to use the differences $(\sigma_{i+1} - \sigma_i)$ with *dqds*. In other words, successive shifts σ_i and σ_{i+1} in LR lead to the *dqds* step

$$L_{i+1}U_{i+1} = U_i L_i - (\sigma_{i+1} - \sigma_i)I.$$

3.4.3. Single shift LR and double *dqds*. Analogously to a double shift, a single shift LR step is equivalent to two steps of *dqds* when we consider the implicit implementation of these shifted algorithms.

$$\begin{array}{rlcl} & \textbf{single shift LR} & & \textbf{\textit{double dqds}} \\ J_1 = & U_0 L_0 & \multirow{2}{*}{$\Big\}$} & \\ J_1 - \sigma_1 I = & L_1 U_1 & & L_1 U_1 = U_0 L_0 - \sigma_1 I \\ J_2 = & U_1 L_1 + \sigma_1 I & & L_2 U_2 = U_1 L_1 - (-\sigma_1) I \end{array} \tag{3.14}$$

Similar to (3.7) and (3.12),

$$J_2 = \boldsymbol{\mathcal{L}}^{-1} J_1 \boldsymbol{\mathcal{L}} \quad \text{and} \quad L_2 U_2 = \boldsymbol{\mathcal{L}}^{-1}(U_0 L_0)\boldsymbol{\mathcal{L}} \tag{3.15}$$

with $\boldsymbol{\mathcal{L}} = L_1$. Here matrix M is tridiagonal,

$$\boldsymbol{\mathcal{L}\mathcal{U}} = L_1 U_1 = J_1 - \sigma_1 =: M$$

and matrix K corresponding to (3.10) has only one bulge in entry $(3, 1)$ (instead of a 2×1 bulge).

Recall from Section 3.4.1 that the implicit double LR algorithm uses the technique of *bulge chasing*. This technique is also applied for implicit single shifts.

The next section develops a form of bulge chasing for the *triple dqds* algorithm (*3dqds*). We did not develop this technique for the *double dqds* algorithm (*2dqds*) because in our shift strategy we will always use double shifts (only initially we use single *dqds* with zero shifts). See Section 6.3 for details on the shift strategy in our complete algorithm.

4. *Triple dqds* algorithm.

We use the term *3dqds* as a shorthand for our *triple dqds* algorithm which, using bulge chasing, implements implicitly the three *dqds* steps (3.11) equivalent to an implicit double shift LR step. Although the *3dqds* algorithm has been primarily developed to avoid complex arithmetic in the case of consecutive complex shifts σ_1 and $\sigma_2 = \overline{\sigma}_1$ in the presence of complex eigenvalues, it can be applied to the case of two real shifts σ_1 and σ_2. To cover both cases, all we need is the sum and the product of the pair of shifts, $\mathsf{sum} = \sigma_1 + \sigma_2$ and $\mathsf{prod} = \sigma_1\sigma_2$, to form matrix M in (3.8),

$$M = (U_0 L_0)^2 - \mathsf{sum}(U_0 L_0) + \mathsf{prod} I. \tag{4.1}$$

Using (3.12) the idea is to transform U_0 into L_3 and L_0 into U_3 by bulge chasing in each matrix,

$$L_3 U_3 = \underbrace{\boldsymbol{\mathcal{L}}^{-1} U_0}\,\underbrace{L_0 \boldsymbol{\mathcal{L}}}.$$

Notice that we need to transform an upper bidiagonal into a lower bidiagonal and vice-versa. From the uniqueness of the LU factorization, when it exists, it follows, see [21], that there is a unique hidden matrix X such that

$$L_3 = \boldsymbol{\mathcal{L}}^{-1} U_0 X^{-1}, \quad X L_0 \boldsymbol{\mathcal{L}} = U_3.$$

The matrix $\boldsymbol{\mathcal{L}}$ is given, from section 3.4.1, as a product

$$\boldsymbol{\mathcal{L}} = \mathcal{L}_1 \dots \mathcal{L}_{n-1}\mathcal{L}_n$$

($\mathcal{L}_n = I$) and we will gradually construct the matrix X in corresponding factored form $X_n \cdots X_2 X_1$. In fact we will write each X_i as a product

$$X_i = Y_i Z_i.$$

Matrices $\mathcal{L}_i$ and Y_i are elementary matrices, $\mathcal{L}_i = I + \boldsymbol{m}_i \boldsymbol{e}_i^T$ and $Y_i = I + \boldsymbol{w}_i \boldsymbol{e}_i^T$, but Z_i is not. The details are quite complicated and will be shown in the following sections.

4.1. Chasing the bulges. Starting with the factors L_0, U_0 and the shifts σ_1, σ_2, we normalize column 1 of M in (4.1) to form $\mathcal{L}_1$, spoil the bidiagonal form with

$$\underbrace{\mathcal{L}_1^{-1} U_0}\underbrace{L_0 \mathcal{L}_1}$$

and at each *minor* step i, $i = 1, \dots, n$, matrices Z_i, $\mathcal{L}_i$ and Y_i are chosen to chase the bulges. After n minor steps, we obtain L_3 and U_3,

$$\begin{aligned} L_3 U_3 &= \underbrace{\mathcal{L}_n^{-1} \cdots \mathcal{L}_1^{-1} U_0 Z_1^{-1} Y_1^{-1} \cdots Z_n^{-1} Y_n^{-1}}\underbrace{Y_n Z_n \cdots Y_1 Z_1 L_0 \mathcal{L}_1 \cdots \mathcal{L}_n} \\ &= \underbrace{\mathcal{L}_n^{-1} \cdots \mathcal{L}_1^{-1} U_0 X_1^{-1} \cdots X_n^{-1}}\underbrace{X_n \cdots X_1 L_0 \mathcal{L}_1 \cdots \mathcal{L}_n} \\ &= \underbrace{\boldsymbol{\mathcal{L}}^{-1} U_0 X^{-1}}\underbrace{X L_0 \boldsymbol{\mathcal{L}}} \end{aligned}$$

All the work of bulge chasing will be confined to two matrices F and G. Initially,

$$F = U_0, \qquad G = L_0$$

and, finally,

$$F = L_3, \qquad G = U_3.$$

For a pair of shifts σ_1 and σ_2 (real or a complex conjugate pair), the *triple dqds* algorithm has the following matrix formulation:

```
3dqds(σ1, σ2) :
    % step 1
    $F = U_0;\ G = L_0$
    $F = FZ_1^{-1};\ G = Z_1G$
    $F = \mathcal{L}_1^{-1}F;\ G = G\mathcal{L}_1$          [form $\mathcal{L}_1$ using (3.9) and (4.1)]
    $F = FY_1^{-1};\ G = Y_1G$

    % steps 2 to n-3
    for $i = 2, \ldots, n-3$
        $F = FZ_i^{-1};\ G = Z_iG$
        $F = \mathcal{L}_i^{-1}F;\ G = G\mathcal{L}_i$
        $F = FY_i^{-1};\ G = Y_iG$        [$Z_i$ with one, $\mathcal{L}_i$ with two and $Y_i$ with three
    end for                                 nonzero off-diagonal entries]

    % step n-2
    $F = FZ_{n-2}^{-1};\ G = Z_{n-2}G$
    $F = \mathcal{L}_{n-2}^{-1}F;\ G = G\mathcal{L}_{n-2}$
    $F = FY_{n-2}^{-1};\ G = Y_{n-2}G$        [$Y_{n-2}$ with two nonzero off-diagonal entries]

    % step n-1
    $F = FZ_{n-1}^{-1};\ G = Z_{n-1}G$
    $F = \mathcal{L}_{n-1}^{-1}F;\ G = G\mathcal{L}_{n-1}$
    $F = FY_{n-1}^{-1};\ G = Y_{n-1}G$        [$Y_{n-1}$ and $\mathcal{L}_{n-1}$ with one nonzero off-diagonal entry]

    % step n
    $\mathcal{L}_n = I;\ Y_n = I$
    $F = FZ_n^{-1};\ G = Z_nG$        [$Z_n$ diagonal]
    $L_3 = F;\ U_3 = G$
```

4.2. Details of *3dqds*. In this section we will go into important details of the *3dqds* algorithm. Consider L_0 with subdiagonal entries $l_1, \ldots, l_{n-1}$ and U_0 with diagonal entries $u_1, \ldots, u_n$, as defined in Section 3.3, and consider matrices L_3 and U_3 with subdiagonal entries $\widehat{l}_1, \ldots, \widehat{l}_{n-1}$ and diagonal entries $\widehat{u}_1, \ldots, \widehat{u}_n$, respectively.

For each iteration of *3dqds*, at the beginning of a minor step i, $i = 2, \ldots, n-3$, the *active* 4×4 windows of F and G are

$$F = \begin{bmatrix} \ddots & & & & & \\ \ddots & 1 & & & & \\ & \widehat{l}_{i-1} & u_i & 1 & & \\ & \boldsymbol{x_l} & & u_{i+1} & 1 & \\ & \boldsymbol{y_l} & & & u_{i+2} & \ddots \\ & & & & & \ddots \end{bmatrix}, \qquad G = \begin{bmatrix} \ddots & \ddots & & & & \\ & \widehat{u}_{i-1} & 1 & & & \\ & & \boldsymbol{x_r} & & & \\ & & \boldsymbol{y_r} & 1 & & \\ & & \boldsymbol{z_r} & l_{i+1} & 1 & \\ & & & & \ddots & \ddots \end{bmatrix}. \tag{4.2}$$

We denote the entries $F_{i+1,i-1}$ and $F_{i+2,i-1}$ by $\begin{bmatrix} x_l & y_l \end{bmatrix}^T$, the 2×1 bulge in F, and the entries $G_{i,i}$, $G_{i+1,i}$ and $G_{i+2,i}$ by $\begin{bmatrix} x_r & y_r & z_r \end{bmatrix}^T$. The bulge in G is $\begin{bmatrix} y_r & z_r \end{bmatrix}^T$. In practice, as the bulges both change value and position, these 5 auxiliary variables are enough to accomplish all the calculations. The subscripts l and r in x, y and z derive from "left" and "right", respectively, and observe that these variables are not from matrices X, Y and Z described in Section 4.1.

Each minor step i, $i = 2, \ldots, n-3$, consists of the following 3 parts. The values in x_l, y_l, x_r, y_r, z_r change and they move one column right and one row down.

Minor step *i*

(a) $F \longleftarrow F Z_i^{-1}$ puts 0 into $F_{i,i+1}$ and 1 into $F_{i,i}$
$G \longleftarrow Z_i G$ turns $G_{i,i+1}$ into 1 and changes $G_{i,i}$

$$Z_i^{-1} = \begin{bmatrix} \ddots & & & & & \\ & 1 & & & & \\ & & \frac{1}{u_i} & -\frac{1}{u_i} & & \\ & & 0 & 1 & & \\ & & & & 1 & \\ & & & & & \ddots \end{bmatrix}, \qquad Z_i = \begin{bmatrix} \ddots & & & & & \\ & 1 & & & & \\ & & u_i & 1 & & \\ & & 0 & 1 & & \\ & & & & 1 & \\ & & & & & \ddots \end{bmatrix},$$

$$F Z_i^{-1} = \begin{bmatrix} \ddots & & & & & \\ \ddots & 1 & & & & \\ & \widehat{l}_{i-1} & \mathbf{1} & \mathbf{0} & & \\ & x_l & & u_{i+1} & 1 & \\ & y_l & & & u_{i+2} & \ddots \\ & & & & & \ddots \end{bmatrix}, \quad Z_i G = \begin{bmatrix} \ddots & \ddots & & & & \\ & \widehat{u}_{i-1} & 1 & & & \\ & & x_r & \mathbf{1} & & \\ & & y_r & 1 & & \\ & & z_r & l_{i+1} & 1 & \\ & & & & \ddots & \ddots \end{bmatrix}$$

$$x_r \longleftarrow x_r * u_i + y_r$$

(b) $F \longleftarrow \mathcal{L}_i^{-1} F$ puts 0 into $F_{i+1,i-1}$ and $F_{i+2,i-1}$, and moves the bulge to column i
$G \longleftarrow G \mathcal{L}_i$ creates $\widehat{u}_i$ in $G_{i,i}$ and changes $G_{i+1,i}$, $G_{i+2,i}$ and $G_{i+3,i}$ below it

$$\begin{array}{l} x_l \longleftarrow -x_l/\widehat{l}_{i-1} \\ y_l \longleftarrow -y_l/\widehat{l}_{i-1} \end{array}, \quad \mathcal{L}_i^{-1} = \begin{bmatrix} \ddots & & & & \\ & 1 & & & \\ & x_l & 1 & & \\ & y_l & & 1 & \\ & & & & \ddots \end{bmatrix}, \quad \mathcal{L}_i = \begin{bmatrix} \ddots & & & & \\ & 1 & & & \\ & -x_l & 1 & & \\ & -y_l & & 1 & \\ & & & & \ddots \end{bmatrix}$$

$$
\mathcal{L}_i^{-1}F = \begin{bmatrix}
\ddots & & & & & \\
\ddots & 1 & & & & \\
 & \widehat{l}_{i-1} & \mathbf{1} & \mathbf{0} & & \\
 & \mathbf{0} & x_l & u_{i+1} & 1 & \\
 & \mathbf{0} & y_l & & u_{i+2} & \ddots \\
 & & & & & \ddots
\end{bmatrix}, \quad
G\mathcal{L}_i = \begin{bmatrix}
\ddots & \ddots & & & & & \\
 & \widehat{u}_{i-1} & 1 & & & & \\
 & & \boldsymbol{\widehat{u}_i} & \mathbf{1} & & & \\
 & & x_r & 1 & & & \\
 & & y_r & l_{i+1} & 1 & & \\
 & & z_r & & l_{i+2} & 1 & \\
 & & & & & \ddots & \ddots
\end{bmatrix}
$$

$$
\begin{aligned}
\widehat{u}_i &\longleftarrow x_r - x_l \\
x_r &\longleftarrow y_r - x_l \\
y_r &\longleftarrow z_r - y_l - x_l * l_{i+1} \\
z_r &\longleftarrow -y_l * l_{i+2}
\end{aligned}
$$

(c) $G \longleftarrow Y_iG$ puts 0 into $G_{i+1,i}$, $G_{i+2,i}$ and $G_{i+3,i}$, and moves the bulge to column $i+1$
$F \longleftarrow FY_i^{-1}$ creates $\widehat{l}_i$ in $F_{i+1,i}$ and changes $F_{i+2,i}$ and $F_{i+3,i}$ (bulge in F) below it

$$
\begin{aligned}
x_r &\longleftarrow x_r/\widehat{u}_i \\
y_r &\longleftarrow y_r/\widehat{u}_i \\
z_r &\longleftarrow z_r/\widehat{u}_i
\end{aligned}, \quad
Y_i^{-1} = \begin{bmatrix}
\ddots & & & & & \\
 & 1 & & & & \\
 & x_r & 1 & & & \\
 & y_r & & 1 & & \\
 & z_r & & & 1 & \\
 & & & & & \ddots
\end{bmatrix}, \quad
Y_i = \begin{bmatrix}
\ddots & & & & & \\
 & 1 & & & & \\
 & -x_r & 1 & & & \\
 & -y_r & & 1 & & \\
 & -z_r & & & 1 & \\
 & & & & & \ddots
\end{bmatrix}
$$

$$
FY_i^{-1} = \begin{bmatrix}
\ddots & & & & & & \\
\ddots & 1 & & & & & \\
 & \widehat{l}_{i-1} & \mathbf{1} & \mathbf{0} & & & \\
 & \mathbf{0} & \boldsymbol{\widehat{l}_i} & u_{i+1} & 1 & & \\
 & \mathbf{0} & x_l & & u_{i+2} & 1 & \\
 & & y_l & & & u_{i+3} & \ddots \\
 & & & & & & \ddots
\end{bmatrix}, \quad
Y_iG = \begin{bmatrix}
\ddots & \ddots & & & & & \\
 & \widehat{u}_{i-1} & 1 & & & & \\
 & & \boldsymbol{\widehat{u}_i} & \mathbf{1} & & & \\
 & & \mathbf{0} & x_r & & & \\
 & & \mathbf{0} & y_r & 1 & & \\
 & & \mathbf{0} & z_r & l_{i+2} & 1 & \\
 & & & & & \ddots & \ddots
\end{bmatrix}
$$

$$
\begin{aligned}
\widehat{l}_i &\longleftarrow x_l + y_r + x_r * u_{i+1} \\
x_l &\longleftarrow y_l + z_r + y_r * u_{i+2} \\
y_l &\longleftarrow z_r * u_{i+3}
\end{aligned}
\qquad\qquad
\begin{aligned}
x_r &\longleftarrow 1 - x_r \\
y_r &\longleftarrow l_{i+1} - y_r \\
z_r &\longleftarrow -z_r
\end{aligned}
$$

The result of this minor step is that the active windows of F and G shown in (4.2) have been moved down and to the right by one place.

Naturally steps $1, n-2, n-1, n$ are slightly different and their practical implementation may be found in [11, pp.147–157].

4.3. Comparison of *dqds* and *3dqds*. In this section we present a detailed version of the inner loop of *3dqds* and compare one step of *3dqds* with three steps of simple *dqds* in terms of arithmetic effort.

Here is the inner loop of *3dqds*. See Appendix A for the whole *3dqds* algorithm.

```
3dqds($\sigma_1, \sigma_2$) :
        for $i = 2, \dots, n-3$
            $x_r = x_r * u_i + y_r$
            $x_l = -x_l/\widehat{l}_{i-1};\ \ y_l = -y_l/\widehat{l}_{i-1};$
            $\widehat{u}_i = x_r - x_l;$
            $x_r = y_r - x_l;\ \ y_r = z_r - y_l - x_l * l_{i+1};\ \ z_r = -y_l * l_{i+2}$
            $x_r = x_r/\widehat{u}_i;\ \ y_r = y_r/\widehat{u}_i;\ \ z_r = z_r/\widehat{u}_i$
            $\widehat{l}_i = x_l + y_r + x_r * u_{i+1}$
            $x_l = y_l + z_r + y_r * u_{i+2};\ \ y_l = z_r * u_{i+3}$
            $x_r = 1 - x_r;\ \ y_r = l_{i+1} - y_r;\ \ z_r = -z_r$
        end for
```

In contrast,

```
dqds($\sigma$) :
        $d_1 = u_1 - \sigma$
        for $i = 1, \dots, n-1$
            $\widehat{u}_i = d_i + l_i$
            $t = u_{i+1}/\widehat{u}_i$                                  (4.3)
            $d_{i+1} = d_i t - \sigma$
            $\widehat{l}_i = l_i t$
        end for
        $\widehat{u}_n = d_n$.
```

In practice, each d_{i+1} may be written over its predecessor in a single variable d and, using and auxiliary variable t, only one division is needed.

Table 4.1 below shows that the number of floating point operations required by one step of *3dqds* is comparable to three steps of *dqds* (table expresses only the number of floating point operations in the inner loops).

	3dqds	3 *dqds* steps
Divisions	5	3
Multiplications	6	6
Additions	5	3
Subtractions	6	3
Assignments	16	12
Auxiliary variables	5	2

TABLE 4.1
Operation count of 3dqds and 3 dqds steps.

However to accomplish a complex conjugate pair of shifts these 3 *dqds* steps will be complex in contrast to our *3dqds* which uses only real arithmetic. Thus 3 steps of complex *dqds* take more time than one step of *3dqds* (complex arithmetic raises the cost by a factor of about 4 [6, p.163]).

5. Error analysis.

We turn to the effect of finite precision arithmetic on our algorithms. First consider the dqds algorithm.

5.1. *dqds.*

It is well known that even in exact arithmetic the *dqds* iteration, applied to the factors L, U of a J matrix can break down due to a zero pivot in the new factors. The *dqds* transform, just like the LR transform, is unstable. In the early days when the new was written over the old immediately breakdown was a disaster. Today all users can afford to store the new factors separately from the old and simply reject a transform with unacceptable element growth, choose a new shift, and continue the iteration. A rejection is a nuisance, not a disaster.

One of the attractions of *dqds* is that it has *high mixed relative stability*, to be explained below. One of us proved this in [9] in the context of singular values of bidiagonals and eigenvalues of real symmetric tridiagonals. Since this desired property is independent of symmetry, we take this opportunity to present the result again in the context of J matrices.

To set up notation for the proof we consider real bidiagonal factors L and U of a real J matrix together with a real shift σ and use the *dqds* transform to obtain output $\widehat{L}, \widehat{U}$ satisfying

$$\widehat{L}\widehat{U} = UL - \sigma I. \tag{5.1}$$

We assume no anomalies occur, i.e. no divisions by zero, no overflow/underflow.

Theorem 5.1 ([9], Theorem 4). *Let dqds(σ) map L, U into computed $\widehat{L}, \widehat{U}$ with no anomalies. Then well chosen small relative changes in the entries of both input and output matrices, of at most 3 ulps each, produces new matrices, one pair mapped into the other, in exact arithmetic, by dqds(σ).*

Our analysis consists to write down the exact relations satisfied by the computed quantities $\widehat{L}, \widehat{U}$ and then to work out among them an exact *dqds* transform whose input is close to L, U and output is close to $\widehat{L}, \widehat{U}$. The diagram in Figure 5.1 is an excellent summary.

$$\begin{tikzcd}
L, U \arrow[r, "\text{dqds}", "\text{computed}"'] \arrow[d, leftrightarrow, "\begin{array}{r}\text{change each}\\ l_i \text{ by 1 ulp}\\ u_i \text{ by 3 ulps}\end{array}"'] & \widehat{L}, \widehat{U} \arrow[d, leftrightarrow, "\begin{array}{l}\text{change each}\\ \widehat{l}_i, \widehat{u}_i \text{ by 2 ulps}\end{array}"] \\
\overset{\frown}{L}, \overset{\frown}{U} \arrow[r, "\text{dqds}", "\text{exact}"'] & \breve{L}, \breve{U}
\end{tikzcd}$$

$$\breve{L}\breve{U} = \overset{\frown}{U}\overset{\frown}{L} - \sigma I$$

Figure 5.1. *Effects of roundoff for dqds.*

The model of arithmetic we assume is that the floating point result of a basic arithmetic operation $\odot$ (one of the four binary operations $+$, $-$, $*$ and $/$) satisfies

$$\text{fl}(x \odot y) = (x \odot y)(1 + \varepsilon) = (x \odot y)/(1 + \eta) \tag{5.2}$$

where ε and η depend on x, y, and the operation $\odot$, and satisfy

$$|\varepsilon| < \epsilon, \quad |\eta| < \epsilon. \tag{5.3}$$

The quantity ϵ is called variously *roundoff unit*, *machine precision* or *macheps*. We will choose freely the form (ε or η) that suits the analysis. We will also use the acronym *ulp* which stands for *units in the last place held* and it is the natural way to refer to relative differences between numbers.

Our result is possible because of the simple form of the recurrence for the auxiliary variable $\{d_i\}_{i=1}^n$. In exact arithmetic

$$d_1 = u_1 - \sigma, \quad d_{i+1} = \frac{d_i u_{i+1}}{d_i + l_i} - \sigma, \quad i = 1, \dots, n-1.$$

Proof. We consider the floating point implementation of *dqds* in (4.3). The computed quantities $\widehat{L}, \widehat{U}$ are governed by the following exact relations.

$$\begin{aligned}
\widehat{u}_i &= \mathrm{fl}(d_i + l_i) = (d_i + l_i)/(1+\varepsilon_+) \\
t &= \mathrm{fl}(u_{i+1}/\widehat{u}_i) = \frac{u_{i+1}(1+\varepsilon_/)}{\widehat{u}_i} = \frac{u_{i+1}(1+\varepsilon_/)(1+\varepsilon_+)}{d_i + l_i} \\
d_{i+1} &= \mathrm{fl}\left(\mathrm{fl}(d_i * t) - \sigma\right) = \frac{d_i t(1+\varepsilon_*) - \sigma}{1+\varepsilon_{i+1}} \\
\widehat{l}_i &= \mathrm{fl}(l_i * t) = l_i t/(1+\varepsilon_{**})
\end{aligned}$$

The symbol ε_{**} represents the rounding error in the second multiplication $l_i * t_i$. All the ε's obey (5.3) and depend on i but we supress this dependence on i except for the subtraction of the shift σ. Here ε_{i+1} accounts for the error in subtracting the real shift σ. To be consistent we must also use $d_i(1+\varepsilon_i)$, where ε_i is defined in minor step $i-1$, and $(1+\varepsilon_1)d_1 = u_1 - \sigma$. Here t is just an auxiliary variable for the analysis.

Now we can write an exact *dqds* transform using $[\,\cdot\,]$ to surround our chosen variables.

$$\begin{aligned}
\left[\widehat{u}_i(1+\varepsilon_+)(1+\varepsilon_i)\right] &= \left[d_i(1+\varepsilon_i)\right] + \left[l_i(1+\varepsilon_i)\right] \\
\left[d_{i+1}(1+\varepsilon_{i+1})\right] &= \frac{\left[d_i(1+\varepsilon_i)\right]\left[u_{i+1}(1+\varepsilon_/)(1+\varepsilon_+)(1+\varepsilon_*)\right]}{\left[d_i(1+\varepsilon_i)\right] + \left[l_i(1+\varepsilon_i)\right]} - \sigma \\
\left[\widehat{l}_i(1+\varepsilon_*)(1+\varepsilon_{**})\right] &= \frac{\left[l_i(1+\varepsilon_i)\right]\left[u_{i+1}(1+\varepsilon_/)(1+\varepsilon_+)(1+\varepsilon_*)\right]}{\left[d_i(1+\varepsilon_i)\right] + \left[l_i(1+\varepsilon_i)\right]}
\end{aligned}$$

We can read off the perturbations, defining $\widehat{l}_i, \widehat{u}_{i+1}$ and $\breve{l}_i, \breve{u}_i$ on the way to an exact transform:

$$\begin{aligned}
l_i &\longrightarrow l_i(1+\varepsilon_i) =: \widehat{l}_i & \qquad \widehat{l}_i &\longrightarrow \widehat{l}_i(1+\varepsilon_*)(1+\varepsilon_{**}) =: \breve{l}_i \\
u_{i+1} &\longrightarrow u_{i+1}(1+\varepsilon_/)(1+\varepsilon_+)(1+\varepsilon_*) =: \widehat{u}_{i+1} & \qquad \widehat{u}_i &\longrightarrow \widehat{u}_i(1+\varepsilon_+)(1+\varepsilon_i) =: \breve{u}_i
\end{aligned}$$

The perturbations are as claimed in the theorem: 3 ulps for u_i and 1 ulp for l_i, and 2 ulps each for $\widehat{l}_i$ and $\widehat{u}_i$ as shown in Figure 5.1. Notice that our choices of $\widehat{L}, \widehat{U}$ and $\breve{L}, \breve{U}$ are not in general machine representable.

When $\sigma = 0$ the $(1+\varepsilon_i)$ factors are omitted but still 3 ulps are needed for u_{i+1}. □

The remarkable feature here is that element growth does not impair the result. However, Theorem 5.1 does not guarantee that *dqds* returns accurate eigenvalues. For that, an extra requirement is needed such as positivity of all the parameters u_j, l_j in the computation, as is the case for the eigenvalues of $B^T B$ where B is upper bidiagonal.

We mention that Yao Yang considered the roundoff in *dqds* in his dissertation at UC, Berkeley, in 1994 [41]. He had two results. He gave an $n = 4$ example to show that even *dqd* (no shift in *dqds*) is not backward stable. He also produced an *a posteriori* (computable) bound on the error in the exact product $\widehat{L}\widehat{U}$ of the output matrices. Unfortunatly, his dissertation has not been published but his results are stated and proved in [22].

5.2. *3dqds.* Each minor step in the algorithm consists of 3 simple transformations on work matrices F and G. All three parts arise from similarities that chase the bulges in the transformation from U_0L_0 to L_3U_3. See section 4. Two of these transformations are elementary transformations of the form $I + \boldsymbol{v}\boldsymbol{e}_j^T$, with inverse $I - \boldsymbol{v}\boldsymbol{e}_j^T$, and $\boldsymbol{v}$ has at most 3 nonzero entries. We examine the condition number of these 3 transforms. Consult Section 4.2 to follow the details.

- The active part of Z_i is

$$\begin{bmatrix} u_i & 1 \\ 0 & 1 \end{bmatrix} \qquad \text{and} \qquad \operatorname{cond}(Z_i) \simeq \max\left\{|u_i|, |u_i|^{-1}\right\}.$$

- The active part of $\mathcal{L}_i$ is

$$\begin{bmatrix} 1 & & \\ -x_l/\widehat{l}_{i-1} & 1 & \\ -y_l/\widehat{l}_{i-1} & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad \operatorname{cond}(\mathcal{L}_i) \simeq 1 + \left(\frac{x_l}{\widehat{l}_{i-1}}\right)^2 + \left(\frac{y_l}{\widehat{l}_{i-1}}\right)^2.$$

- The active part of Y_i is

$$\begin{bmatrix} 1 & & & \\ -x_r/\widehat{u}_i & 1 & & \\ -y_r/\widehat{u}_i & 0 & 1 & \\ -z_r/\widehat{u}_i & 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad \operatorname{cond}(Y_i) \simeq 1 + \left(\frac{x_r}{\widehat{u}_i}\right)^2 + \left(\frac{y_r}{\widehat{u}_i}\right)^2 + \left(\frac{z_r}{\widehat{u}_i}\right)^2.$$

The variables x_l, y_l, x_r, y_r, z_r are formed from additions and multiplications of previous quantities. Note that u_i is part of the input and so is assumed to be of acceptable size. We see that it is tiny values of $\widehat{l}_{i-1}$ and $\widehat{u}_i$ that lead to an ill-conditioned transform at minor step i. In the simple *dqds* algorithm a small value of $\widehat{u}_i$ (relative to u_{i+1}) leads to a large value of $\widehat{l}_i$ and d_{i+1}. In *3dqds* the effect of 3 consecutive transforms is more complicated. The message is the same: reject any transform that has more then modest element growth. In practice, $|\widehat{u}_i|$ and $|\widehat{l}_{i-1}|$ are monitored and a transform is rejected if either quantity is too big (bigger than $1/\sqrt{\varepsilon}$). The computed eigenvalues are used as input for Rayleigh quotient correction in the original balanced matrix.

In order to understand the intricate arguments below we have found it essential to absorb the contents of Sections 4.1 and 4.2, in particular the division of the typical inner loop of *3dqds* in three parts **(a)**, **(b)** and **(c)**. The three *dqds* similarities have morphed into a sequence of $n-1$ similarities of FG (implicit) each of which in its turn is composed of three transformations of F and G by matrices Z_i, $\mathcal{L}_i$, Y_i (with exact inverses) at minor step, or loop, i where we concentrate our attention.

Minor step i.

Recall that at the start the bulges x_l, y_l are in column $i-1$ of F while x_r, y_r, z_r are in column i of G. See (4.2). The values in these bulges change and they move one column right and one row down. In analysis, not practice, as the bulges both change value and position, new variables are created and denoted by augmentation of the subscripts. See Table 5.1. By the end of minor step i new values are given to all the bulge variables to be ready for the next step. The most active is x_r, the entry on the diagonal of G. The loop i updates x_r four times so we find

$$x_r,\ x_{r_1},\ x_{r_2},\ x_{r_3},\ x_{r_4}$$

and the last value x_{r_4} becomes x_r at the next loop $i+1$. Its position changes from $G_{i,i}$ to $G_{i+1,i+1}$. This change in position occurs at operation 14 of the 16 arithmetic operations in *3dqds*.

To follow the analysis below the reader should have reference to Section 4.2. At minor step i the inner loop transforms columns $i-1$, i of F and i, $i+1$ of G.

To anticipate our result we are going to show that very small relative, but well chosen, perturbations in the input and output variables of each part, **(a)**, **(b)**, and **(c)**, separately, of loop i yield exact, albeit implicit, transformations of F and G. Of course, the input and output variables are different for each part.

Part	Input	Output
(a)	x_r, y_r, u_i	x_{r_1}
(b)	$\widehat{l}_{i-1}$, l_{i+1}, l_{i+2}	$\widehat{u}_i$
	x_l, y_l	x_{l_1}, y_{l_1}
	x_{r_1}, y_r, z_r	x_{r_2}, y_{r_1}, z_{r_1}
(c)	l_{i+1}, u_{i+2}, u_{i+3}, $\widehat{u}_i$	$\widehat{l}_i$
	x_{l_1}, y_{l_1}	x_{l_2}, y_{l_2}
	x_{r_2}, y_{r_1}, z_{r_1}	x_{r_3}, y_{r_2}, z_{r_2}
		x_{r_4}, y_{r_3}, z_{r_3}

TABLE 5.1
Input and output variables of 3dqds algorithm.

Note that the output variables for Part **(b)** may be perturbed (again) as input variables of Part **(c)**. We will point out the two (of 16) operations at which our perturbations fail to give an exact implementation of the whole of loop i. That would be a result as strong as the one for real *dqds*.

As said above, if $\boldsymbol{e}_j$ denotes column j of I and $\boldsymbol{v}$ is a vector satisfying $\boldsymbol{e}_j^T\boldsymbol{v} = \boldsymbol{0}$ then the exact inverse of $I - \boldsymbol{v}\boldsymbol{e}_j^T$ is $I + \boldsymbol{v}\boldsymbol{e}_j^T$ since $(I - \boldsymbol{v}\boldsymbol{e}_j^\top)(I + \boldsymbol{v}\boldsymbol{e}_j^\top) = I - \boldsymbol{v}\boldsymbol{e}_j^\top\boldsymbol{v}\boldsymbol{e}_j^\top = I - (\boldsymbol{e}_j^\top\boldsymbol{v})\boldsymbol{v}\boldsymbol{e}_j^\top = I$. Hence the attraction of using elementary matrices for Parts ***(b)*** and ***(c)***. The matrix Z_i, whose active part is $\begin{bmatrix} u_i & 1 \\ 0 & 1 \end{bmatrix}$, is not elementary but the action of its inverse is implicit in creating 0 and 1 in F and it is only Z_i that acts on G. Thus $1/u_i$ is never used explicitly and, again, there is no error in the implicit use of Z_i^{-1} on F. These observations help to explain the welcome accuracy of *3dqds* in practice.

In the analysis in each statement we use a subscript on ε as an indicator of the operation. For example,

$$\mathrm{fl}\left(a+b+c*d\right) = \mathrm{fl}\left(\mathrm{fl}(a+b) + \mathrm{fl}(c*d)\right) = \left[(a+b)(1+\varepsilon_{+}) + c\cdot d(1+\varepsilon_{*})\right](1+\varepsilon_{++}).$$

We find it simpler to not name the perturbed variables but to indicate them by judicious use of parentheses and square brackets. We use either a dot or juxtaposition to represent an exact multiplication.

Loop $\boldsymbol{i}$, $1 < i < n-2,$ in Section 4.3.

Part ($\boldsymbol{a}$)
$F \longleftarrow FZ_i^{-1}$ puts $F_{i,i+1} = 0$ and $F_{i,i} = 1$. No errors.
$G \longleftarrow Z_iG$ turns $G_{i,i+1} = 1,$ updates x_r in $G_{i,i}$.

1 $x_{r1} = \text{fl}\left(\text{fl}(x_r * u_i) + y_r\right) = \left[x_r \cdot u_i(1+\varepsilon_*) + y_r\right](1+\varepsilon_+)$

$$\boldsymbol{x_{r1} = \left[x_r(1+\varepsilon_+)\right]\left[u_i(1+\varepsilon_*)\right] + \left[y_r(1+\varepsilon_+)\right]}$$

Part ($\boldsymbol{b}$)
$F \longleftarrow \mathcal{L}_i^{-1}F$ puts 0 into $F_{i+1,i-1}$ and $F_{i+2,i-1},$ and moves the bulge to column i.
$G \longleftarrow G\mathcal{L}_i$ creates $\widehat{u}_i$ in $G_{i,i}$ (an *LU* output) and creates $x_{r_2}, y_{r_1}, z_{r_1}$(bulge in G) below it.

2 $x_{l_1} = -\,\text{fl}(x_l/\widehat{l}_{i-1}) = -x_l/\widehat{l}_{i-1}(1+\varepsilon_/)$ $\boldsymbol{x_{l_1} = -\left[x_l(1+\varepsilon_/)\right]/\widehat{l}_{i-1}}$

3 $y_{l_1} = -\,\text{fl}(y_l/\widehat{l}_{i-1}) = -y_l/\widehat{l}_{i-1}(1+\varepsilon_/)$ $\boldsymbol{y_{l_1} = -\left[y_l(1+\varepsilon_/)\right]/\widehat{l}_{i-1}}$

Note that $\widehat{l}_{i-1}$ is created in loop $i-1$ and $(1+\varepsilon_/)$ differs from $(1+\varepsilon_/)$ in Op. 3.

4 $\widehat{u}_i = \text{fl}(x_{r_1} - x_{l_1}) = (x_{r_1} - x_{l_1})/(1+\varepsilon_-)$ $\boldsymbol{\left[(1+\varepsilon_-)\widehat{u}_i\right] = x_{r_1} - x_{l_1}}$

5 $x_{r_2} = \text{fl}(y_r - x_{l_1}) = (y_r - x_{l_1})/(1+\varepsilon_-)$ $\boldsymbol{\left[(1+\varepsilon_-)x_{r_2}\right] = y_r - x_{l_1}}$

6 $y_{r_1} = \text{fl}\left(\text{fl}(z_r - y_{l_1}) - \text{fl}(x_{l_1} * l_{i+1})\right) = \left[(z_r - y_{l_1})/(1+\varepsilon_-) - x_{l_1}\cdot l_{i+1}(1+\varepsilon_*)\right]/(1+\varepsilon_{--})$

$$\boldsymbol{\left[(1+\varepsilon_-)(1+\varepsilon_{--})y_{r1}\right] = z_r - y_{l_1} - x_{l_1}\left[l_{i+1}(1+\varepsilon_*)(1+\varepsilon_-)\right]}$$

7 $z_{r_1} = -\,\text{fl}(y_{l_1} * l_{i+2}) = y_{l_1}l_{i+2}(1+\varepsilon_*)$ $\boldsymbol{z_{r_1} = -y_{l_1}\left[l_{i+2}(1+\varepsilon_*)\right]}$

Part ($\boldsymbol{c}$)
$F \longleftarrow FY_i^{-1}$ creates $\widehat{l}_i$ in $F_{i+1,i}$ (an *LU* output) and creates x_{l_2}, y_{l_2} (bulge in F) below it.
$G \longleftarrow Y_iG$ puts 0 into $G_{i+1,i}$, $G_{i+2,i}$ and $G_{i+3,i}$, and moves the bulge to column $i+1$.

8 $x_{r_3} = -\,\text{fl}(x_{r_2}/\widehat{u}_i) = -x_{r_2}/\widehat{u}_i(1+\varepsilon_/)$ $\boldsymbol{x_{r_3} = -\left[x_{r_2}(1+\varepsilon_/)\right]/\widehat{u}_i}$

9 $y_{r_2} = -\,\text{fl}(y_{r_1}/\widehat{u}_i) = -y_{r_1}/\widehat{u}_i(1+\varepsilon_/)$ $\boldsymbol{y_{r_2} = -\left[y_{r_1}(1+\varepsilon_/)\right]/\widehat{u}_i}$

10 $z_{r_2} = -\,\text{fl}(z_{r_1}/\widehat{u}_i) = -z_{r_1}/\widehat{u}_i(1+\varepsilon_/)$ $\boldsymbol{z_{r_2} = -\left[z_{r_1}(1+\varepsilon_/)\right]/\widehat{u}_i}$

11 $\widehat{l}_i = \text{fl}\left(\text{fl}(x_{l_1} + y_{r_2}) + \text{fl}(x_{r_3} * u_{i+1})\right) = \text{fl}\left((x_{l_1} + y_{r_2})/(1+\varepsilon_+) + x_{r_3}\cdot u_{i+1}(1+\varepsilon_*)\right)$

$$= \left[(x_{l_1} + y_{r_2})/(1+\varepsilon_+) + x_{r_3}\cdot u_{i+1}(1+\varepsilon_*)\right]/(1+\varepsilon_{++})$$

$$\boldsymbol{\left[(1+\varepsilon_+)(1+\varepsilon_{++})\widehat{l}_i\right] = x_{l_1} + y_{r_2} + x_{r_3}\left[u_{i+1}(1+\varepsilon_*)(1+\varepsilon_+)\right]}$$

Part (*c*) (cont.)

12 $x_{l_2} = \mathrm{fl}\left(\mathrm{fl}(y_{l_1} + z_{r_2}) + \mathrm{fl}(y_{r_2} * u_{i+2})\right) = \mathrm{fl}\left((y_{l_1} + z_{r_2})/(1+\varepsilon_+) + y_{r_2} \cdot u_{i+2}(1+\varepsilon_*)\right)$

$= \left[(y_{l_1} + z_{r_2})/(1+\varepsilon_+) + y_{r_2} \cdot u_{i+2}(1+\varepsilon_*)\right]/(1+\varepsilon_{++})$

$\boldsymbol{\left[(1+\varepsilon_+)(1+\varepsilon_{++})x_{l_2}\right] = y_{l_1} + z_{r_2} + y_{r_2}\left[u_{i+2}(1+\varepsilon_*)(1+\varepsilon_+)\right]}$

13 $y_{l_2} = \mathrm{fl}(z_{r_2} * u_{i+3}) = z_{r_2} \cdot u_{i+3}(1+\varepsilon_*)$ $\boldsymbol{y_{l_2} = z_{r_2}\left[u_{i+3}(1+\varepsilon_*)\right]}$

14 $x_{r_4} = \mathrm{fl}(1 - x_{r_3}) = (1 - x_{r_3})/(1+\varepsilon_-)$ $\boldsymbol{(1+\varepsilon_-)x_{r_4} = 1 - x_{r_3}}$

15 $y_{r_3} = \mathrm{fl}(l_{i+1} - y_{r_2}) = (l_{i+1} - y_{r_2})/(1+\varepsilon_-)$ $\boldsymbol{(1+\varepsilon_-)y_{r_3} = l_{i+1} - y_{r_2}}$

16 $z_{r_3} = \mathrm{fl}(-z_{r_2}) = -z_{r_2}$ $\boldsymbol{z_{r_3} = -z_{r_2}}$

No error in negation.

end loop

Some comments. In Op. 4, for example, when cancellation occurs (x_{r_1} and x_{l_1} have same exponent) there is no error in subtraction but $\widehat{u}_i$'s uncertainty increases.

We perturb x_{r_2} in Op. 5, as an output in Part (*b*), and also in Op. 8, as an input in Part (*c*). Similarly, we perturb y_{r_1} in Op. 6, in Part (*b*), and also in Op. 9 in Part (*c*). We use plain z_{r_1} in Op. 7, as an output in Part (*b*), and perturb z_{r_1} in Op. 10, as an input in Part (*c*).

We did not need to perturb x_{l_1} nor y_{l_1} in Ops. 2 and 3 in Part (b) and used x_{l_1} and y_{l_1} in Ops. 5 and 6, still Part (b), as well as Op. 11 and 12, in Part (c). So x_{l_1} and y_{l_1} did preserve their identities for the whole of loop i.

In Ops. 5 and 6 the perturbations we heaped on x_{r_2} and y_{r_1} were to avoid perturbing x_{l_1} and y_{l_1}. It seemed just too messy to try and carry the perturb x_{r_2} and y_{r_1} through the later operations in Part (c) that use them, such as Ops. 8 and 9.

Minor step 1 has a slightly different analysis but we omit the details which may be derived using a similar analysis.

In summary,

Theorem 5.2. *If 3dqds is executed in standard floating point IEEE standard arithmetic with no invalid operations then suitable small perturbations (2 ulps maximum) of Parts (a), (b), and (c) produce an exact instance of each part in every minor step.*

6. Implementation details.

6.1. Deflation ($\boldsymbol{n \leftarrow n-1}$). Some of our criteria for deflating come from [24], others are new. Consider both matrices UL and LU and the trailing 2×2 blocks,

$$\begin{bmatrix} l_{n-1}+u_{n-1} & 1 \\ l_{n-1}u_n & u_n \end{bmatrix}, \qquad \begin{bmatrix} l_{n-2}+u_{n-1} & 1 \\ l_{n-1}u_{n-1} & l_{n-1}+u_n \end{bmatrix}.$$

Deflation ($n \leftarrow n-1$) removes l_{n-1} as well as u_n. Looking at entry $(n-1,n-1)$ of UL shows that a necessary condition is that l_{n-1} be negligible compared to u_{n-1},

$$|l_{n-1}| < tol \cdot |u_{n-1}|, \tag{6.1}$$

for a certain tolerance tol close to roundoff unit ε.

The (n,n) entries of UL and LU suggest either $u_n + acshift$ or $l_{n-1}+u_n+acshift$ as eigenvalues where $acshift$ is the accumulated shift. Recall that simple *dqds* is a non-restoring transform (see (3.5)). To make these consistent we require that

$$|l_{n-1}| < tol \cdot |u_n + acshift|. \tag{6.2}$$

Finally we must consider the change $\delta\lambda$ in the eigenvalue λ caused by setting $l_{n-1}=0$. We estimate $\delta\lambda$ by starting from UL with $l_{n-1}=0$ and then allowing l_{n-1} to grow. To this end let J be UL with $l_{n-1}=0$ and $(u_n, \boldsymbol{y}^T, \boldsymbol{x})$ be the eigentriple for J. Clearly $\boldsymbol{y}^T = \boldsymbol{e}_n^T$. Now we consider perturbation theory with parameter l_{n-1}. The perturbing matrix δJ, as l_{n-1} grows, is

$$l_{n-1}(\boldsymbol{e}_{n-1}+\boldsymbol{e}_n u_n)\boldsymbol{e}_{n-1}^T.$$

By first order perturbation analysis

$$|\delta\lambda| = \frac{|\boldsymbol{y}^T \delta J \boldsymbol{x}|}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$$

and $\|\boldsymbol{y}\|_2 = 1$ in our case. So,

$$|\delta\lambda| = \frac{\left|l_{n-1}\boldsymbol{e}_n^T(\boldsymbol{e}_{n-1}+\boldsymbol{e}_n u_n)\boldsymbol{e}_{n-1}^T\boldsymbol{x}\right|}{\|\boldsymbol{x}\|_2} = \frac{|l_{n-1}u_n||x_{n-1}|}{\|\boldsymbol{x}\|_2}$$

and we use the crude bound $\dfrac{|x_{n-1}|}{\|\boldsymbol{x}\|_2} < 1$. So, we let l_{n-1} grow until the change

$$|\delta\lambda| < |l_{n-1}u_n|$$

in eigenvalue $\lambda = u_n$ is no longer acceptable. Our condition for deflation is then

$$|l_{n-1}u_n| < tol \cdot |acshift + u_n|. \tag{6.3}$$

A similar first order perturbation analysis for LU with $l_{n-1}=0$ will give our last condition for deflation. For the eigentriple $(u_n, \boldsymbol{y}^T, \boldsymbol{x})$ we also have $\boldsymbol{y}^T = \boldsymbol{e}_n^T$. The perturbing matrix is now

$$l_{n-1}\boldsymbol{e}_n\left(\boldsymbol{e}_{n-1}^T u_{n-1} + \boldsymbol{e}_n^T\right)$$

and

$$|\delta\lambda| = \frac{\left|l_{n-1}\boldsymbol{e}_n^T\boldsymbol{e}_n(\boldsymbol{e}_{n-1}^T u_{n-1}+\boldsymbol{e}_n^T)\boldsymbol{x}\right|}{\|\boldsymbol{x}\|_2} = |l_{n-1}|\frac{|u_{n-1}x_{n-1}+x_n|}{\|\boldsymbol{x}\|_2} < |l_{n-1}|\left(|u_{n-1}|+1\right).$$

Finally we require

$$|l_{n-1}|\left(|u_{n-1}|+1\right) < tol \cdot |acshift + u_n|. \tag{6.4}$$

6.2. Splitting and deflation ($n \leftarrow n-2$). Recall that the implicit L theorem was invoked to justify the *3dqds* algorithm. This result fails if any l_k, $k < n-1$ vanishes. Consequently, checking for negligible values among the l_k is a necessity, not a luxury for increased efficiency. Consider $J = UL$ in block form

$$\left[\begin{array}{cc:cc} & J_1 & & \\ & & 1 & \\ \hdashline & \mu & & \\ & & J_2 & \end{array}\right]$$

where $\mu = u_{k+1}l_k$, $k < n-1$. We can replace μ by 0 when

$$\text{spectrum}(J_1) \cup \text{spectrum}(J_2) = \text{spectrum}(J), \quad \textit{to working accuracy.}$$

However we are not going to estimate the eigenvalues of J_1 and J_2. Instead we create a local criterion for splitting at $(k+1,k)$ as follows. Focus on the principal 4×4 window of J given by

$$\left[\begin{array}{cc:cc} u_{k-1}+l_{k-1} & 1 & & \\ u_k l_{k-1} & u_k + l_k & 1 & \\ \hdashline & u_{k+1}l_k & u_{k+1}+l_{k+1} & 1 \\ & & u_{k+2}l_{k+1} & u_{k+2}+l_{k+2} \end{array}\right].$$

Now J_1 and J_2 are both 2×2 and our local criterion is

$$\det(J_1) \cdot \det(J_2) = \det(J), \quad \textit{to working accuracy.} \tag{6.5}$$

Let us see what this yields. Perform block factorization on J and note that the Schur complement of J_1 in J is

$$J_2' = J_2 - \begin{bmatrix} 0 & \mu \\ 0 & 0 \end{bmatrix} J_1^{-1} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

with

$$J_1^{-1} = \frac{1}{det_1} \begin{bmatrix} u_k + l_k & -1 \\ -u_k l_{k-1} & u_{k-1} + l_{k-1} \end{bmatrix}$$

where

$$det_1 = \det(J_1) = u_{k-1}(u_k + l_k) + l_{k-1}l_k.$$

Thus

$$J_2' = \begin{bmatrix} u_{k+1} + l_{k+1} & 1 \\ u_{k+2}l_{k+1} & u_{k+2} + l_{k+2} \end{bmatrix} - \begin{bmatrix} \mu(u_{k-1} + l_{k-1})/det_1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Since det is linear by rows and the second rows are equal

$$\det(J_2) - \det(J_2') = \mu(u_{k-1} + l_{k-1})(u_{k+2} + l_{k+2})/det_1.$$

Our criterion reduces to splitting only when

$$\det(J_2') = \det(J_2), \quad \text{to working accuracy.}$$

Thus we require

$$|l_k u_{k+1}(u_{k+2} + l_{k+2})(u_{k-1} + l_{k-1})/det_1| < tol \cdot |\det(J_2)| \,.$$

Since

$$det_2 = \det(J_2) = u_{k+1}(u_{k+2} + l_{k+2}) + l_{k+1} l_{k+2},$$

the criterion for splitting J at $(k+1, k)$ is then

$$|l_k u_{k+1}(u_{k+2} + l_{k+2})(u_{k-1} + l_{k-1})| < tol \cdot |det_1 det_2| \,. \tag{6.6}$$

Finally, to remove l_k we also need l_k to be negligible compared to u_k,

$$|l_k| < tol \cdot |u_k|. \tag{6.7}$$

Deflation ($n \leftarrow n-2$)

We use the same criterion for deflation ($n \leftarrow n-2$), but because $l_{k+2} = l_n = 0$ there is a common factor det_2 on each side of (6.6). Deflate the trailing 2×2 submatrix when

$$|l_{n-2}| < tol \cdot |u_{n-2}| \tag{6.8}$$

and

$$|l_{n-2}(u_{n-3} + l_{n-3})| < tol \cdot |u_{n-3}(u_{n-2} + l_{n-2}) + l_{n-3} l_{n-2}| \,. \tag{6.9}$$

We omit the role of *acshift* here because it makes the situation more complicated. We have to recall that *3dqds* uses restoring shifts and *acshift* is always real. So, for complex shifts, det_2 is not going to zero. In fact

$$|det_2| \geq |\Im(\lambda)|^2$$

where λ is an eigenvalue of J_2.

When $n = 3$ these criteria simplify a lot. Both reduce to

$$|l_1| < tol \cdot |u_1|.$$

6.3. Shift strategy. As with LR, the *dqds* algorithm with no shift gradually forces large entries to the top and brings small entries towards the bottom. We want to use a shift as soon as the trailing 2×2 principal submatrix appears to be converging. We use the size of the last two entries of L to make the judgement. The code executes a *dqds* transform with a zero shift if

$$l_{n-1} > 10^{-2} \qquad \text{and} \qquad l_{n-2} > 10^{-2}.$$

Otherwise, a *3dqds* transform is executed with

$$\texttt{sum} = l_{n-1} + (u_{n-1} + u_n), \qquad \texttt{prod} = u_{n-1} u_n,$$

the trace and the determinant of the trailing 2×2 submatrix of UL. This will let us converge to either two real eigenvalues in the bottom 2×2 or a single 2×2 block with a complex conjugate pair of eigenvalues.

An unexpected reward for having both transforms available is to cope with a rejected transform. Our strategy is simply to use the other transform with the current shift. More precisely, given `sum` and `prod`, if *3dqds*(`sum`, `prod`) is rejected we try *dqds*(u_n); if *dqds(0)* is rejected, we try *3dqds*(δ, δ) with $\delta = \sqrt{\varepsilon}$. We have to admit the possibility of a succession of rejections and in this case we don't want to move away from the previous shift too much, just a small amount so that the transformation does not breakdown. See Algorithm 4 in Appendix B for details. The number of rejections is recorded and added to the total number of iterations.

7. Factored forms.

7.1. Eigenvectors from twisted factorizations of the balanced form $\boldsymbol{\Delta T}$. A salient property of an unreduced real tridiagonal matrix $C = \mathrm{tridiag}(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c})$ (no off-diagonal entry vanishes) is that it can be balanced by a diagonal similarity easily and, once the matrix is balanced, it can be made real symmetric by changing the signs of certain rows. However, changing the signs is not a similarity transformation and would not preserve the eigenvalues. It is accomplished by premultiplying by a so-called *signature matrix* $\Delta = \mathrm{diag}(\delta_1, \dots, \delta_n)$, $\delta_i = \pm 1$. So we can write

$$\Delta T = SCS^{-1} \tag{7.1}$$

where T is real symmetric and S is diagonal positive definite, $S = \mathrm{diag}(s_1, \dots, s_n)$ with $s_1 = 1$, $s_i = \left(|c_1 c_2 \cdots c_{i-1}|/|b_1 b_2 \cdots b_{i-1}|\right)^{1/2}$, $i = 2, \dots, n$. See [11, Section 2.2.3]

Let λ be a simple eigenvalue of ΔT with eigenvector equations

$$\Delta T \boldsymbol{x} = \boldsymbol{x}\lambda, \qquad \boldsymbol{y}^* \Delta T = \lambda \boldsymbol{y}^*. \tag{7.2}$$

Recall that $\boldsymbol{x}$, $\boldsymbol{y}$ and λ may be complex and $\boldsymbol{y}^*\boldsymbol{x} \neq 0$, since λ is simple. An attraction of the ΔT representation is that the row eigenvector $\boldsymbol{y}^*$ is determined by the right (or column) eigenvector $\boldsymbol{x}$. Transpose $\Delta T \boldsymbol{x} = \boldsymbol{x}\lambda$ and insert $I = \Delta^2$ to find

$$(\boldsymbol{x}^{\mathsf{T}}\Delta)\Delta T = \lambda(\boldsymbol{x}^{\mathsf{T}}\Delta). \tag{7.3}$$

Compare with $\boldsymbol{y}^*\Delta T = \lambda \boldsymbol{y}^*$ to see that $\boldsymbol{y}^* = \boldsymbol{x}^{\mathsf{T}}\Delta$. See [13, 31].

The so-called twisted factorizations generalize the lower and upper bidiagonal factorizations. These factorizations gained new popularity as they were used for the purpose of computing eigenvectors of symmetric tridiagonal matrices [10, 23]. The idea is to begin both a top-to-bottom and a bottom-to-top factorization until they meet at, say, the k-th row, where they will have to be glued together. The index k is called the *twist index* or the *twist position*.

Observe that the eigenvector equations $\Delta T \boldsymbol{x} = \boldsymbol{x}\lambda$ and $(\boldsymbol{x}^{\mathsf{T}}\Delta)\Delta T = \lambda(\boldsymbol{x}^{\mathsf{T}}\Delta)$ are equivalent to

$$(T - \lambda\Delta)\boldsymbol{x} = \boldsymbol{0} \qquad \text{and} \qquad \boldsymbol{x}^{\mathsf{T}}(T - \lambda\Delta) = \boldsymbol{0}$$

where $T - \lambda\Delta$ is symmetric. Now suppose that we have $\widetilde{\lambda}$ as an approximation to an eigenvalue λ of ΔT and that $T - \widetilde{\lambda}\Delta$ admits both lower and upper bidiagonal factorizations, starting the Gaussian elimination at the first row and at the last row, respectively,

$$T - \widetilde{\lambda}\Delta = LDL^{\mathsf{T}} = URU^{\mathsf{T}},$$

where L is unit lower bidiagonal and U is unit upper bidiagonal. Matrices D and R are the diagonals holding the pivots of the elimination process. Let $(L)_{i+1,i} = \ell_i$, $(U)_{i,i+1} = u_i$, $i = 1, \dots, n-1$, and

$D = \mathrm{diag}(d_1, \dots, d_n)$, $R = \mathrm{diag}(r_1, \dots, r_n)$, $i = 1, \dots, n$. Then, for each twist index $k = 1, \dots, n$, we can construct a twisted factorization of $T - \widetilde{\lambda}\Delta$ as

$$T - \widetilde{\lambda}\Delta = N_k G_k N_k^{\mathsf{T}} \tag{7.4}$$

where

$$N_k = \begin{bmatrix} 1 & & & & & & & \\ \ell_1 & 1 & & & & & & \\ & \ddots & \ddots & & & & & \\ & & \ell_{k-1} & 1 & u_{k+1} & & & \\ & & & & 1 & u_{k+2} & & \\ & & & & & \ddots & \ddots & \\ & & & & & & 1 & u_n \\ & & & & & & & 1 \end{bmatrix}, \quad G_k = \begin{bmatrix} d_1 & & & & & & \\ & \ddots & & & & & \\ & & d_{k-1} & & & & \\ & & & \gamma_k & & & \\ & & & & r_{k+1} & & \\ & & & & & \ddots & \\ & & & & & & r_n \end{bmatrix}.$$

The only new quantity is the *twist element* $\gamma_k = (G_k)_{k,k}$ and one formula for it is

$$\gamma_k = d_k + r_k - (T - \widetilde{\lambda}\Delta)_{k,k} \tag{7.5}$$

and another is

$$\gamma_1 = r_1, \quad \gamma_n = d_n, \quad \gamma_{k+1} = \gamma_k r_{k+1}/d_k, \ k = 1, \dots, n-1. \tag{7.6}$$

These formulae are not difficult to obtain. See [38].

A very useful feature of these twisted factorizations is that they can deliver a very accurate approximation to the column eigenvector $\boldsymbol{x}$ (and row eigenvector $\boldsymbol{x}^{\mathsf{T}}\Delta$). Since $N_k^{-1}\boldsymbol{e}_k = \boldsymbol{e}_k$ and $G_k^{-1}\gamma_k\boldsymbol{e}_k = \boldsymbol{e}_k$, solving a system of the form

$$N_k G_k N_k^{\mathsf{T}} \boldsymbol{z} = \gamma_k \boldsymbol{e}_k \tag{7.7}$$

is equivalent to solving

$$N_k^{\mathsf{T}} \boldsymbol{z} = \boldsymbol{e}_k \tag{7.8}$$

which leads to the recurrence

$$\begin{aligned} z_k &= 1, \\ z_i &= -\ell_i z_{i+1}, \qquad i = k-1, k-2, \dots, 1, \\ z_i &= -u_i z_{i-1}, \qquad i = k+1, k+2, \dots, n. \end{aligned} \tag{7.9}$$

The above is just inverse iteration to obtain $\boldsymbol{z}$ (and $\boldsymbol{z}^{\mathsf{T}}\Delta$) as an approximation to λ's eigenvector $\boldsymbol{x}$ (and $\boldsymbol{x}^{\mathsf{T}}\Delta$) with residual norm

$$\frac{\|(T - \widetilde{\lambda}\Delta)\boldsymbol{z}\|}{\|\boldsymbol{z}\|} = \frac{|\gamma_k|}{\|\boldsymbol{z}\|}.$$

Therefore a natural choice for the twist index would be k such that

$$|\gamma_k| = \min_{i=1,\dots,n} |\gamma_i|.$$

This strategy to choose an initial guess for the eigenvector provides, as a by-product, the diagonal entries of $(T - \widetilde{\lambda}\Delta)^{-1}$ since $\left[(T - \widetilde{\lambda}\Delta)^{-1}\right]_{k,k} = \gamma_k^{-1}$. See [38, Lemma 2.3].

If Δ is definite, one important result presented in [7, 8] is that we can always find a twist index k such that

$$|\gamma_k| \leq \sqrt{n}|\widetilde{\lambda} - \lambda|.$$

Since (7.9) uses only multiplications, the computed vector will be very good provided that $\widetilde{\lambda}$ is accurate enough. In the general case, to judge the accuracy of the eigenvectors, we compute column (and row) residual norm relative to the eigenvalue,

$$\frac{\|\Delta T\boldsymbol{z} - \widetilde{\lambda}\boldsymbol{z}\|}{|\widetilde{\lambda}|\|\boldsymbol{z}\|} = \frac{\|\big(T - \widetilde{\lambda}\Delta\big)\boldsymbol{z}\|}{|\widetilde{\lambda}|\|\boldsymbol{z}\|} = \frac{\|\boldsymbol{z}^{\mathsf{T}}\big(T - \widetilde{\lambda}\Delta\big)\|}{|\widetilde{\lambda}|\|\boldsymbol{z}^{\mathsf{T}}\Delta\|}. \tag{7.10}$$

This is a stricter measure than the usual $\dfrac{\|\Delta T\boldsymbol{z} - \widetilde{\lambda}\boldsymbol{z}\|}{\|\boldsymbol{z}\|\|\Delta T\|}$.

In [25] we show that unique tridiagonal "backward error" matrices can be designated for an approximate pair of complex eigenvectors (column and row) or two approximate real eigenvectors.

7.2. Relative eigenvalue condition numbers. The condition number of every eigenvalue of a real symmetric matrix is 1, but only in the absolute sense. The relative condition number can vary. In the unsymmetric case even the absolute condition numbers can rise to ∞ and little is known about relative errors. In [13] several *relative condition numbers* with respect to eigenvalues were derived. Some of them use bidiagonal factorizations instead of the matrix entries and so they shed light on when eigenvalues are less sensitive to perturbations of factored forms than to perturbations of the matrix entries. These condition numbers are measures of *relative sensitivity*, i.e., measures of the relative variation of an eigenvalue with respect to the largest relative perturbation of each of the nonzero entries of the representation of the matrix. So the perturbations we consider are of the form $|\delta p_i| \leq \eta|p_i|$, $0 < \eta \ll 1$. In this section we present the relative condition number for the entries of the matrix C and for the LU factorization of the J-form.

Assume that $\lambda \neq 0$ is a simple eigenvalue of real tridiagonal matrix $C = \text{tridiag}(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c})$. Let $\Delta T = SCS^{-1}$ be the balanced form (7.1) of C and $\big(\lambda, \boldsymbol{x}, \boldsymbol{x}^{\mathsf{T}}\Delta\big)$ be an eigentriple of ΔT,

$$\Delta T\boldsymbol{x} = \boldsymbol{x}\lambda, \qquad (\boldsymbol{x}^{\mathsf{T}}\Delta)\Delta T = \lambda(\boldsymbol{x}^{\mathsf{T}}\Delta), \qquad \lambda \neq 0, \tag{7.11}$$

and recall that ΔT and C eigenvectors are simply related by

$$C\big(S^{-1}\boldsymbol{x}\big) = \big(S^{-1}\boldsymbol{x}\big)\lambda, \qquad \big(\boldsymbol{x}^{\mathsf{T}}\Delta S\big)C = \lambda\big(\boldsymbol{x}^{\mathsf{T}}\Delta S\big). \tag{7.12}$$

The relative condition number with respect to λ for the entries of C is

$$\text{relcond}(\lambda; C) = \frac{\big|\boldsymbol{x}^{\mathsf{T}}\Delta S\big|\,|C|\,\big|S^{-1}\boldsymbol{x}\big|}{|\lambda|\,\big|\big(\boldsymbol{x}^{\mathsf{T}}\Delta S\big)\big(S^{-1}\boldsymbol{x}\big)\big|},$$

where $|M|_{ij} = |M_{ij}|$, for any matrix M. Since S is diagonal it follows that

$$\text{relcond}(\lambda; C) = \frac{\big|\boldsymbol{x}^{\mathsf{T}}\Delta\big|\,|S|\,|C|\,\big|S^{-1}\big|\,|\boldsymbol{x}|}{|\lambda||\boldsymbol{x}^{\mathsf{T}}\Delta\boldsymbol{x}|} = \frac{\big|\boldsymbol{x}^{\mathsf{T}}\Delta\big|\,|\Delta T|\,|\boldsymbol{x}|}{|\lambda||\boldsymbol{x}^{\mathsf{T}}\Delta\boldsymbol{x}|} = \text{relcond}(\lambda; \Delta T). \tag{7.13}$$

We have just shown that, in general, for any scaling matrix X invertible and diagonal, the expression for relcond$(\lambda; C)$ yields relcond$(\lambda; XCX^{-1}) = $ relcond$(\lambda; C)$. See [13, Lemma 6.2].

When C is unreduced it is also diagonally similar to a J-form,

$$J = DCD^{-1} = \text{tridiag}(\mathfrak{b}, \boldsymbol{a}, \mathbf{1})$$

where $D = \text{diag}(1, c_1, c_1c_2, \ldots, c_1c_2\cdots c_{n-1})$ and $\mathfrak{b} = \text{diag}(b_1c_1, b_2c_2, \ldots, b_{n-1}c_{n-1})$. Now assume that J permits bidiagonal factorization $J = LU$ and write

$$\Delta T = FJF^{-1}, \qquad F = SD^{-1}, \tag{7.14}$$

to obtain

$$LU\left(F^{-1}\boldsymbol{x}\right) = \left(F^{-1}\boldsymbol{x}\right)\lambda, \qquad \left(\boldsymbol{x}^{\mathsf{T}}\Delta F\right)\mathcal{L}\mathcal{U} = \lambda\left(\boldsymbol{x}^{\mathsf{T}}\Delta F\right), \qquad \lambda \neq 0.$$

Recall that $L = I + \mathring{L}$ and $U = \text{diag}(u_1, \ldots, u_n)\left(I + \mathring{U}\right)$ with

$$\mathring{L} = \begin{bmatrix} 0 & & & & \\ l_1 & 0 & & & \\ & \ddots & \ddots & & \\ & & l_{n-2} & 0 & \\ & & & l_{n-1} & 0 \end{bmatrix} \quad \text{and} \quad \mathring{U} = \begin{bmatrix} 0 & u_1^{-1} & & & \\ & 0 & u_2^{-1} & & \\ & & \ddots & \ddots & \\ & & & 0 & u_{n-1}^{-1} \\ & & & & 0 \end{bmatrix}.$$

For the cost of solving two bidiagonal linear systems,

$$\boldsymbol{v}^{\mathsf{T}}\left(I + \mathring{\mathcal{U}}\right) = \left(\boldsymbol{x}^{\mathsf{T}}\Delta F\right) \text{ for } \boldsymbol{v}^{\mathsf{T}} \qquad \text{and} \qquad \mathcal{L}\boldsymbol{w} = \mathring{\mathcal{L}}\left(F^{-1}\boldsymbol{x}\right) \text{ for } \boldsymbol{w},$$

we obtain the following expression of the relative condition number for the entries of L and U,

$$\text{relcond}(\lambda; L, U) := \frac{|\boldsymbol{v}|^{\mathsf{T}}|F^{-1}\boldsymbol{x}| + |\boldsymbol{x}^{\mathsf{T}}\Delta F||\boldsymbol{w}|}{|\boldsymbol{x}^{\mathsf{T}}\Delta\boldsymbol{x}|}. \tag{7.15}$$

See [13, Section 6.3]. Next we deal with a case of a simple zero eigenvalue. Although the right hand side of (7.15) is a nonzero finite number for a simple eigenvalue $\lambda = 0$, observe that the perturbations we consider for U, that is, $|\delta u_i| \leq \eta|u_i|$, produce $u_n + \delta u_n = 0$ whenever $u_n = 0$. This means that singularity is preserved or, equivalently, that the zero eigenvalue is preserved. Therefore, it seems appropriate to set $\text{relcond}(0; L, U) = 0$.

We use eigenvalue approximation $\widetilde{\lambda}$ from *Rayleigh Quotient Iteration (RQI)* and eigenvector approximations $\boldsymbol{z}$ and $\boldsymbol{z}^{\mathsf{T}}\Delta$ obtained from (7.8) to compute the relative condition numbers (7.13) and (7.15). The residual norms for ΔT are given by (7.10) but for $J = LU$, with eigenvector approximations $F^{-1}\boldsymbol{z}$ and $\boldsymbol{z}^{\mathsf{T}}\Delta F$, by

$$\frac{\|F^{-1}(T - \widetilde{\lambda}\Delta)\boldsymbol{z}\|}{|\widetilde{\lambda}|\|F^{-1}\boldsymbol{z}\|} \quad \text{and} \quad \frac{\|\boldsymbol{z}^{\mathsf{T}}(T - \widetilde{\lambda}\Delta)F\|}{|\widetilde{\lambda}|\|\boldsymbol{z}^{\mathsf{T}}\Delta F\|}. \tag{7.16}$$

7.3. Generalized Rayleigh Quotient Iteration. In addition to computing both column and row eigenvector approximations from twisted factorizations of ΔT, the algorithm described in Section 7.1 can also be used to improve the accuracy of the eigenvalue approximation $\widetilde{\lambda}$ by performing a Rayleigh Quotient Iteration. So, our code will return an eigenpair approximation $(S^{-1}\boldsymbol{z}, \boldsymbol{z}^{\mathsf{T}}\Delta S)$ for C together with an improved eigenvalue estimate, the generalized Rayleigh quotient,

$$\frac{(\boldsymbol{z}^{\mathsf{T}}\Delta S)C(S^{-1}\boldsymbol{z})}{(\boldsymbol{z}^{\mathsf{T}}\Delta S)(S^{-1}\boldsymbol{z})} = \frac{(\boldsymbol{z}^{\mathsf{T}}\Delta)\Delta T\boldsymbol{z}}{\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}} = \widetilde{\lambda} + \frac{(\boldsymbol{z}^{\mathsf{T}}\Delta)(\Delta T - \widetilde{\lambda}I)\boldsymbol{z}}{\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}}. \tag{7.17}$$

Given the twisted factorization in (7.4) and (7.7), the *Rayleigh quotient correction* is given by

$$\rho := \frac{(\boldsymbol{z}^{\mathsf{T}}\Delta)(\Delta N_k G_k N_k^{\mathsf{T}})\boldsymbol{z}}{\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}} = \frac{\boldsymbol{z}^{\mathsf{T}}\gamma_k\boldsymbol{e}_k}{\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}} = \frac{\gamma_k}{\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}},$$

since $z_k = 1$, where γ_k is given in (7.6) and (7.5).

Recall that for $x \in \mathbb{C}$, $\Re(x)$ denotes the real part of x. The following lemma extends Lemma 12 in [8, pg. 886] to the unsymmetric case.

LEMMA 7.1. *Let $T - \widetilde{\lambda}\Delta = N_k G_k N_k^T$ and $N_k G_k N_k^T\boldsymbol{z} = \gamma_k\boldsymbol{e}_k$, $z_k = 1$. Then the Rayleigh quotient ρ with respect to $\Delta T - \widetilde{\lambda}I$ is*

$$\rho = \frac{\gamma_k}{\boldsymbol{z}^T\Delta\boldsymbol{z}}$$

and

$$\frac{\left\|\big(\Delta T - (\widetilde{\lambda}+\rho)I\big)\boldsymbol{z}\right\|}{\|\boldsymbol{z}\|} = \frac{|\gamma_k|}{\|\boldsymbol{z}\|}\left(\frac{|\boldsymbol{z}^T\Delta\boldsymbol{z}|^2 - \omega_k}{|\boldsymbol{z}^T\Delta\boldsymbol{z}|^2}\right)^{1/2} \tag{7.18}$$

where $\omega_k = 2\delta_k\Re(\boldsymbol{z}^T\Delta\boldsymbol{z}) - \|\boldsymbol{z}\|^2$.

Proof. Let $\Delta = \operatorname{diag}(\delta_1,\dots,\delta_n)$, $\delta_i = \pm 1$. Since

$$\big(\Delta T - (\widetilde{\lambda}+\rho)I\big)\boldsymbol{z} = \big(\Delta T - \widetilde{\lambda}I\big)\boldsymbol{z} - \rho\boldsymbol{z} = \Delta N_k G_k N_k^{\mathsf{T}}\boldsymbol{z} - \rho\boldsymbol{z} = \delta_k\gamma_k\boldsymbol{e}_k - \rho\boldsymbol{z},$$

then

$$\begin{aligned}\left\|\big(\Delta T - (\widetilde{\lambda}+\rho)I\big)\boldsymbol{z}\right\|^2 &= \|\delta_k\gamma_k\boldsymbol{e}_k - \rho\boldsymbol{z}\|^2 = (\delta_k\overline{\gamma_k}\boldsymbol{e}_k - \overline{\rho\boldsymbol{z}})^{\mathsf{T}}\cdot(\delta_k\gamma_k\boldsymbol{e}_k - \rho\boldsymbol{z})\\ &= |\gamma_k|^2 + |\rho|^2\|\boldsymbol{z}\|^2 - 2\delta_k\Re(\overline{\gamma}_k\rho)\\ &= |\gamma_k|^2 + \frac{|\gamma_k|^2}{|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2}\|\boldsymbol{z}\|^2 - \frac{2\delta_k|\gamma_k|^2}{|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2}\Re(\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z})\\ &= \frac{|\gamma_k|^2}{|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2}\left(|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2 + \|\boldsymbol{z}\|^2 - 2\delta_k\Re(\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z})\right).\end{aligned}$$

Thus,

$$\frac{\left\|\big(\Delta T - (\widetilde{\lambda}+\rho)I\big)\boldsymbol{z}\right\|}{\|\boldsymbol{z}\|} = \frac{|\gamma_k|}{\|\boldsymbol{z}\|}\left(\frac{|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2 - \big(2\delta_k\,\Re(\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}) - \|\boldsymbol{z}\|^2\big)}{|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2}\right)^{1/2}.$$

Observe that, since $z_k = 1$,

$$|\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}|^2 + \|\boldsymbol{z}\|^2 - 2\delta_k\,\Re(\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}) = \|\boldsymbol{z}\|^2 + |\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z} - \delta_k|^2 - 1 > 0.$$

If the easily checked condition

$$\omega_k := 2\delta_k\,\Re(\boldsymbol{z}^{\mathsf{T}}\Delta\boldsymbol{z}) - \|\boldsymbol{z}\|^2 > 0 \tag{7.19}$$

is satisfied, we obtain a decrease in the residual norm by using the Rayleigh quotient; the pair $(\widetilde{\lambda}+\rho, \boldsymbol{z})$ is a better approximate eigenpair than $(\widetilde{\lambda}, \boldsymbol{z})$. □

When $\Delta = I$ (symmetrizable case) the condition (7.19) reduces to $2\|\boldsymbol{z}\|^2 - \|\boldsymbol{z}\|^2 = \|\boldsymbol{z}\|^2 > 0$ and the Rayleigh quotient correction always gives an improvement. In this case (7.18) simplifies to

$$\frac{\left\|\left(T-(\widetilde{\lambda}+\rho)I\right)\boldsymbol{z}\right\|}{\|\boldsymbol{z}\|} = \frac{|\gamma_k|}{\|\boldsymbol{z}\|}\left(\frac{\|\boldsymbol{z}\|^4-\|\boldsymbol{z}\|^2}{\|\boldsymbol{z}\|^4}\right)^{1/2} = \frac{|\gamma_k|}{\|\boldsymbol{z}\|}\left(\frac{\|\boldsymbol{z}\|^2-1}{\|\boldsymbol{z}\|^2}\right)^{1/2}. \tag{7.20}$$

Given an approximation $\widetilde{\lambda}$ to an eigenvalue λ of ΔT we compute the twisted factorization of $T - \widetilde{\lambda}\Delta$ and use inverse iteration (7.8) to obtain λ's column and row eigenvector approximations, $\boldsymbol{z}$ and $\boldsymbol{z}^{\mathsf{T}}\Delta$. The Rayleigh quotient correction (7.17) gives a new approximation $\widetilde{\lambda}+\rho$ for λ. We may repeat this process until there is no improvement in the residual (7.18). Although RQI can misbehave for non-normal matrices, we can use (7.19) to judge improvement. Our code *3dqds* examines ω_k and whenever it is greater than zero we apply RQI, otherwise not.

8. Numerical Examples. The need for a tridiagonal eigensolver is admirably covered in Bini, Gemignani and Tisseur [1], many parts of which have been of great help to us. We refer to the Ehrlich-Aberth algorithm (see Section 2.5) as *BGT* and to our code simply as *3dqds*, although we combine *3dqds* with real *dqds* as described in Section 6.3.

Here the *exact* eigenvalue λ_i is computed in quadruple precision, using MATLAB Symbolic Math Toolbox with variable-precision arithmetic, and $\underset{\sim}{\lambda}_i$ denotes the computed eigenvalue in double precision (unit roundoff $2.2\,10^{-16}$). We compare our *3dqds* algorithm with its explicit version, refered as *ex3dqds* (the three steps of *dqds* are computed explicitly in complex arithmetic, see Figure (3.1)), with a MATLAB implementation of *BGT* and with the *QR* algorithm on an upper Hessenberg matrix (MATLAB function eig).

All the experiments were performed in MATLAB (R2020b) on a LAPTOP-KVSVAUU8 with an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz and 8 GB RAM, under Windows 10 Home. No parallel operations were used. We acknowledge that the MATLAB tests do not reflect Fortran performance, but even in MATLAB environment the ratio of elapsed times is an important feature.

Bessel matrix

Bessel matrices, associated with generalized Bessel polynomials [26], are nonsymmetric tridiagonal matrices defined by $B_n^{(a,b)} = \text{tridiag}(\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\gamma})$ with

$$\begin{aligned}
\alpha_1 &= -\frac{b}{a}, \quad \gamma_1 = -\alpha_1, \quad \beta_1 = \frac{\alpha_1}{a+1}, \\
\alpha_j &:= -b\frac{a-2}{(2j+a-2)(2j+a-4)}, \quad j = 2,\dots,n, \\
\gamma_j &:= b\frac{j+a-2}{(2j+a-2)(2j+a-3)}, \quad \beta_j := -b\frac{j}{(2j+a-1)(2j+a-2)}, \quad j = 2,\dots,n-1.
\end{aligned}$$

Parameter b is a scaling factor and most authors take $b = 2$ and so do we. The case $a \in \mathbb{R}$ is the most investigated in literature. The eigenvalues of $B_n^{(a,b)}$, well separated complex eigenvalues, suffer from ill-conditioning that increases with n - close to a defective matrix. In Pasquini [26] it is mentioned that the ill-conditining seems to reach its maximum when a ranges from -8.5 to -4.5. We pay a lot of attention to these matrices because they are an interesting family for our purposes. Each picture teaches us a lot about the behavior of eigenvalues.

Our examples take $B_n^{(-8.5,2)}$, $B_n^{(-4.5,2)}$ and $B_n^{(12,2)}$ for $n = 40, 50$. We show pictures for MATLAB (double precison), *BGT* and *3dqds* to illustrate the extreme sensitivity of some of the eigenvalues.

The results of *ex3dqds* are visually identical to *3dqds*, so we don't show them. In exact arithmetic the spectrum lies on an arc in the interior of the moon-shaped region. Our pictures show this region and the eigenvalues computed in quadruple precision (labeled as *exact*).

The results for $B_{18}^{(-8.5,2)}$, $n = 18, 25$, are shown in Figure 8.1, without RQI (Rayleigh Quotient Iteration) and with one RQI. Observe on the real line that our approximations with one RQI (c) lie on top of the *BGT* approximations. We include the pictures (b) and (d) to show well the extreme sensitivity of the eigenvalues. Note how the eigenvalues move out of the moon-shaped inclusion region.

In Figure 8.2 we show the results for $B_n^{(-4.5,2)}$, $n = 20, 25$, and $B_n^{(12,2)}$, $n = 40, 50$, without RQI. The reader is invited to see the large effect of changing n from 20 to 25, in (a) and (b), and from 40 to 50, in (c) and (d). Notice that our results are slightly but consistently better than those of the other two methods.

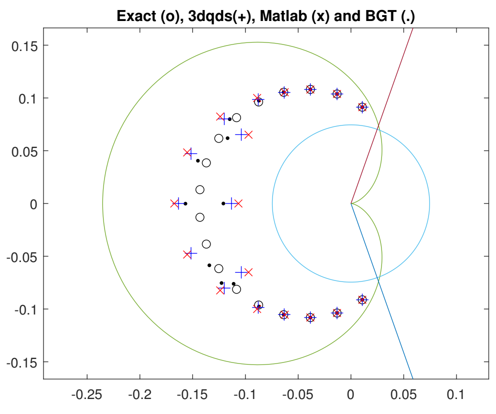
Table 8.1 shows the minimum and maximum relative errors, $rel_{min} = \min_i |\lambda_i - \widetilde{\lambda}_i|/|\lambda_i|$ and $rel_{max} = \max_i |\lambda_i - \widetilde{\lambda}_i|/|\lambda_i|$. The relative condition numbers relcond$(\lambda; C)$ and relcond$(\lambda; L, U)$ (see (7.13) and (7.15)) and residual norms (see (7.10) and (7.16)) are shown in Table 8.2. We show both condition numbers because MATLAB and *BGT* only use matrix entries and *3dqds* uses L, U factors.

	eig		*BGT*		*3dqds*	
(a,b); n	rel_{min}	rel_{max}	rel_{min}	rel_{max}	rel_{mim}	rel_{max}
$(-8.5,2)$; 18	$1.6\ 10^{-6}$	$2.7\ 10^{-1}$	$7.1\ 10^{-7}$	$2.3\ 10^{-1}$	$5.9\ 10^{-7}$	$2.3\ 10^{-1}$
$(-8.5,2)$; 25	$2.5\ 10^{-1}$	$1.9\ 10^{0}$	$1.3\ 10^{-1}$	$1.9\ 10^{0}$	$2.4\ 10^{-1}$	$1.8\ 10^{0}$
$(-4.5,2)$; 20	$4.1\ 10^{-7}$	$3.2\ 10^{-1}$	$1.0\ 10^{-7}$	$2.1\ 10^{-1}$	$1.5\ 10^{-8}$	$1.2\ 10^{-1}$
$(-4.5,2)$; 25	$2.0\ 10^{-1}$	$1.3\ 10^{0}$	$5.7\ 10^{-2}$	$1.2\ 10^{0}$	$2.0\ 10^{-1}$	$7.3\ 10^{-1}$
$(12,2)$; 40	$3.3\ 10^{-15}$	$1.3\ 10^{-1}$	$1.1\ 10^{-15}$	$1.9\ 10^{-1}$	$2.1\ 10^{-15}$	$1.7\ 10^{-1}$
$(12,2)$; 50	$7.0\ 10^{-15}$	$3.5\ 10^{-1}$	$8.5\ 10^{-16}$	$4.3\ 10^{-1}$	$6.5\ 10^{-15}$	$3.4\ 10^{-1}$

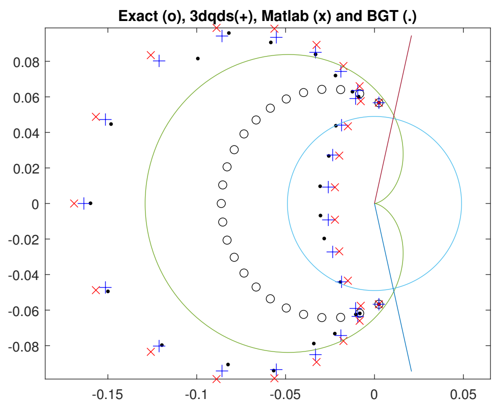
TABLE 8.1
Relative errors for computed eigenvalues from $B_n^{(-8.5,2)}$, $B_n^{(12,2)}$, and ($B_n^{(-4.5,2)}$ with one RQI).

	relcond$(\lambda; L, U)$		relcond$(\lambda; C)$		*max residuals*	
(a,b); n	*min*	*max*	*min*	*max*	$J = LU$	ΔT
$(-8.5,2)$; 18	$4.3\ 10^{8}$	$6.6\ 10^{13}$	$1.5\ 10^{10}$	$2.2\ 10^{15}$	$2.1\ 10^{-14}$	$1.8\ 10^{-14}$
$(-8.5,2)$; 25	$8.0\ 10^{10}$	$4.3\ 10^{13}$	$4.3\ 10^{12}$	$2.6\ 10^{15}$	$3.9\ 10^{-14}$	$1.2\ 10^{-14}$
$(-4.5,2)$; 20	$8.1\ 10^{7}$	$3.5\ 10^{14}$	$3.5\ 10^{9}$	$1.7\ 10^{16}$	$3.5\ 10^{-14}$	$3.1\ 10^{-14}$
$(-4.5,2)$; 25	$6.3\ 10^{8}$	$2.6\ 10^{14}$	$4.6\ 10^{10}$	$1.9\ 10^{16}$	$4.5\ 10^{-14}$	$5.0\ 10^{-14}$
$(12,2)$; 40	$9.3\ 10^{1}$	$7.0\ 10^{15}$	$1.4\ 10^{2}$	$1.8\ 10^{16}$	$1.3\ 10^{-15}$	$4.6\ 10^{-14}$
$(12,2)$; 50	$1.8\ 10^{2}$	$1.1\ 10^{16}$	$2.7\ 10^{2}$	$3.8\ 10^{16}$	$1.9\ 10^{-15}$	$7.7\ 10^{-15}$

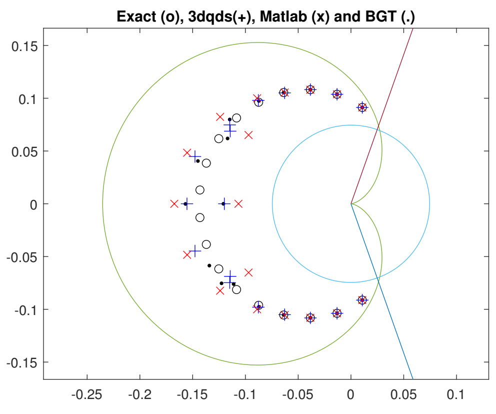
TABLE 8.2
Relative condition numbers and residual norms for $B_n^{(-8.5,2)}$, $B_n^{(-4.5,2)}$, and $B_n^{(12,2)}$.



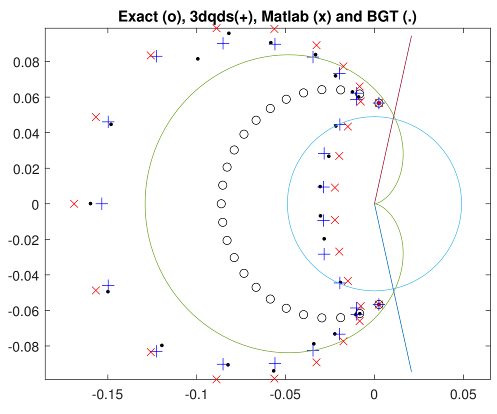
(a) $n = 18$; no RQI



(b) $n = 25$; no RQI



(c) $n = 18$; one RQI



(d) $n = 25$; one RQI

FIGURE 8.1. *Eigenvalues of* $B_n^{(-8.5,2)}$, $n = 18, 25$.

Clement matrix

The so-called *Clement* matrices (see [3])

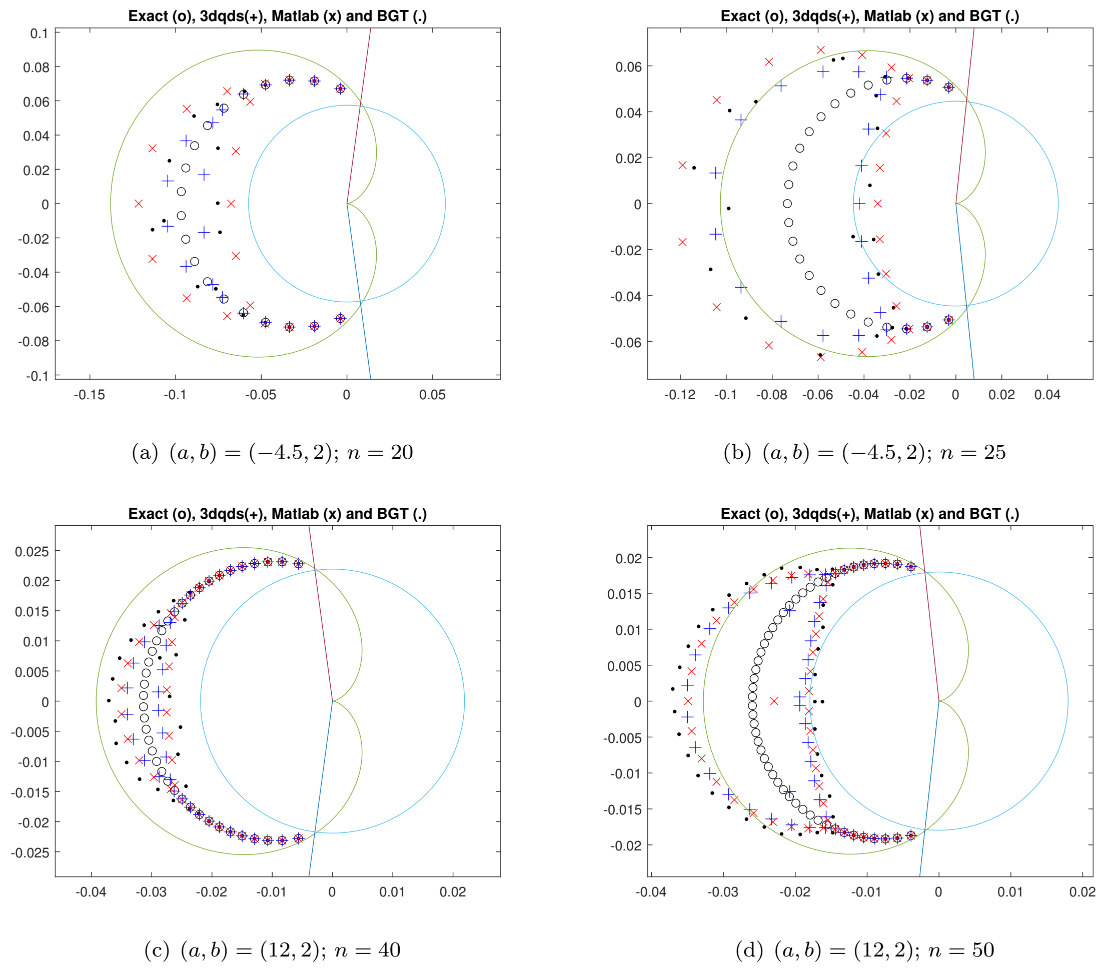
$$C = \text{tridiag}(\boldsymbol{b}, \boldsymbol{0}, \boldsymbol{c})$$

with $b_j = j$ and $c_j = b_{n-j}$, $j = 1, \ldots, n-1$, have real eigenvalues,

$$\begin{aligned} &\pm n-1, n-3, \ldots, 1, \qquad \text{for } n \text{ even}, \\ &\pm n-1, n-3, \ldots, 0, \qquad \text{for } n \text{ odd}. \end{aligned}$$

These matrices posed no serious difficulties. The initial zero diagonal obliges the *dqds* based methods to take care when finding an initial *LU* factorization.

The *3dqds* and *ex3dqds* codes use only real shifts as they should and the accuracy (approximately the same) is less than *BGT* but satisfactory. One RQI reduces errors to $\mathcal{O}(\varepsilon)$.



(a) $(a, b) = (-4.5, 2)$; $n = 20$

(b) $(a, b) = (-4.5, 2)$; $n = 25$

(c) $(a, b) = (12, 2)$; $n = 40$

(d) $(a, b) = (12, 2)$; $n = 50$

FIGURE 8.2. *Eigenvalues of* $B_n^{(-4.5,2)}$ *and* $B_n^{(12,2)}$ *(without RQI).*

Our numerical tests have $n = 50, 100, 200, 400, 800$. The relative condition number relcond$(\lambda; C)$ ranges from 10^0 to $4\ 10^2$ and it is smaller at the ends of the spectrum. The maximum residual norm for C is $\mathcal{O}(10^{-11})$. The minimum and maximum relative errors, rel_{min} and rel_{max}, are shown in Table 8.3. Note the poor performance of MATLAB's eig (so much for backward stability).

The CPU elapsed times are presented in Table 8.4. We put (+) whenever a RQI is used. Since we compare MATLAB versions of all the codes we acknowledge that the elapsed times are accurate to only about 0.02 seconds. However, this is good enough to show the striking time ratios between *BGT* and the *dqds* codes.

We draw the readers attention that for $n = 400$ our algorithm is about 200 times faster than *BGT* but when n rises to 1000 it is over 600 times faster. This is finding the eigenvalues to the same accuracy, namely $\mathcal{O}(\varepsilon)$. In addition we provide eigenvectors and condition numbers.

An important further comment which illustrates challenges of the unsymmetric eigenvalue problem is that in these examples for $n \geq 200$ the scaling matrices F used above (see (7.14)) are not

	eig		*BGT*		*3dqds*	
n	rel_{min}	rel_{max}	rel_{min}	rel_{max}	rel_{mim}	rel_{max}
50	$4.4\ 10^{-16}$	$7.4\ 10^{-11}$	0	0	$1.7\ 10^{-16}$	$4.7\ 10^{-15}$
100	$1.6\ 10^{-15}$	$1.6\ 10^{-3}$	0	$1.8\ 10^{-16}$	0	$2.1\ 10^{-14}$
200	$4.3\ 10^{-16}$	$1.6\ 10^{1}$	0	$1.1\ 10^{-15}$	0	$9.4\ 10^{-14}$
400	$5.7\ 10^{-16}$	$5.5\ 10^{1}$	0	$5.6\ 10^{-16}$	0	$7.6\ 10^{-13}$
800	$2.7\ 10^{-15}$	$4.4\ 10^{2}$	0	$1.2\ 10^{-15}$	0	$1.8\ 10^{-12}$

Table 8.3
Relative errors for Clement matrices (without RQI).

n	eig	*BGT*	*ex3dqds (+)*	*3dqds (+)*
100	0.011	0.83	0.014	0.009
200	0.097	2.01	0.020	0.014
400	0.28	8.33	0.036	0.025
800	0.90	35.70	0.080	0.066
1000	1.49	67.02	0.120	0.094

Table 8.4
CPU time in seconds for Clement matrices.

representable. This limitation indicates why we use the ΔT form for computing the eigenvectors. The overflow problem, which also arises for S^{-1} (see (7.12)), although not so quickly, explains why *BGT* confines its attention to $n = 50$, but we go further because of our approach.

Matrix with clusters

Matrix in Test 5 in [1],

$$C = D^{-1}\,\mathrm{tridiag}(\mathbf{1}, \boldsymbol{\alpha}, \mathbf{1}), \quad D = \mathrm{diag}(\boldsymbol{\beta}), \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^n$$
$$\alpha_k = 10^{5(-1)^k} \cdot (-1)^{\lfloor k/4 \rfloor}, \;\; \beta_k = (-1)^{\lfloor k/3 \rfloor}, \quad k = 1, \ldots, n,$$

seems to be a challenging test matrix. It was designed to have large, tight clusters of eigenvalues around 10^{-5}, -10^5 and 10^5. Half the spectrum is around 10^{-5} and the rest is divided unevenly between -10^5 and 10^5. The diagonal alternates between entries of absolute value 10^5 and 10^{-5} and so, for *dqds* codes, there is a lot of rearranging to do. When $n \geq 100$ it is not clear what is meant by accuracy.

The matrix has a repetitive structure and the diagonal entries are a good guide to the eigenvalues. For $n = 100$ and for the large real eigenvalues near $\pm 10^5$ the eigenvectors have spikes $(-10^{-5}, -1, 10^{-5})$ (complex conjugate pairs have spikes $(10^{-5}, 1, -10^{-7}, -1, 10^{-5})$, at the appropriate places, and negligible elsewhere. Hence the numerical supports for many eigenvectors are disjoint. The essential structure of the matrix is exhibited with $n = 10$,

$$C = \begin{pmatrix} 10^{-5} & 1 & & & & & & & & \\ 1 & 10^5 & 1 & & & & & & & \\ & -1 & -10^{-5} & -1 & & & & & & \\ & & -1 & 10^5 & -1 & & & & & \\ & & & -1 & 10^{-5} & -1 & & & & \\ & & & & 1 & -10^5 & 1 & & & \\ & & & & & 1 & -10^{-5} & 1 & & \\ & & & & & & 1 & 10^5 & 1 & \\ & & & & & & & -1 & -10^{-5} & -1 \\ & & & & & & & & -1 & -10^5 \end{pmatrix}$$

and it has 5 eigenvalues near 0, 3 eigenvalues near 10^5 and 2 near -10^5. All the eigenvalues are well-conditioned and the three codes obtain the correct number of eigenvalues in each cluster.

When $n = 20$ there are 10 eigenvalues near 10^{-5}, 6 near -10^5 and 4 near 10^5; relcond$(\lambda; C)$ and relcond$(\lambda; L, U)$ are all less than $1.2\ 10^1$; the maximum residual norm for C and $J = LU$ is $\mathcal{O}(10^{-2})$. For the eigenvalues with small modulus, *BGT* and *3dqds* (with 2 RQI, in average) compute approximations with relative errors of $\mathcal{O}(\varepsilon)$, whereas eig yield larger relative errors, as large as 10^{-6}. See Table 8.5.

	eig		*BGT*		*3dqds* (+ +)	
λ	rel_{min}	rel_{max}	rel_{min}	rel_{max}	rel_{mim}	rel_{max}
$\lambda \approx -10^5$	$2.0\ 10^{-20}$	$1.9\ 10^{-15}$	$2.0\ 10^{-20}$	$7.1\ 10^{-17}$	$2.0\ 10^{-20}$	$8.6\ 10^{-11}$
$\lambda \approx 10^5$	$5.0\ 10^{-31}$	$2.2\ 10^{-13}$	$5.0\ 10^{-31}$	$2.2\ 10^{-14}$	$5.0\ 10^{-31}$	$1.0\ 10^{-10}$
$\|\lambda\| \approx 10^{-5}$	$1.2\ 10^{-8}$	$2.0\ 10^{-6}$	$3.0\ 10^{-17}$	$2.7\ 10^{-16}$	$8.0\ 10^{-17}$	$2.0\ 10^{-16}$

Table 8.5
Relative errors for the three clusters in Test 5, with $n = 20$.

Other scaled test matrices

Here we consider other test matrices from [1]. The eigenvalues of these matrices have a variety of distributions, in particular, the eigenvalues in Test 4 and Test 7 are distributed along curves. See Figure 8.3. All these matrices are given in the form

$$C = D^{-1}\,\mathrm{tridiag}(\mathbf{1}, \boldsymbol{\alpha}, \mathbf{1}), \quad D = \mathrm{diag}(\boldsymbol{\beta}), \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^n.$$

$$
\begin{aligned}
&\textbf{Test 1}: \alpha_k = (-1)^{\lfloor k/8 \rfloor},\ \ \beta_k = (-1)^k/k, \quad k = 1,\dots,n.\\
&\textbf{Test 3}: \alpha_k = k,\ \ \beta_k = n-k+1, \quad k = 1,\dots,n.\\
&\textbf{Test 4}: \alpha_k = (-1)^k,\ \ \beta_k = 20\cdot(-1)^{\lfloor k/5 \rfloor}, \quad k = 1,\dots,n.\\
&\textbf{Test 6}: \alpha_k = 2,\ \ \beta_k = 1, \quad k = 1,\dots,n.\\
&\textbf{Test 7}: \alpha_k = \frac{1}{k} + \frac{1}{n-k+1},\ \ \beta_k = \frac{1}{k}(-1)^{\lfloor k/9 \rfloor}, \quad k = 1,\dots,n.\\
&\textbf{Test 9}: \alpha_k = 1, \quad k = 1,\dots,n; \quad \beta_k = \begin{cases} 1 & \text{if } k < n/2 \\ -1 & \text{if } k \ge n/2 \end{cases}.
\end{aligned}
\tag{8.1}
$$

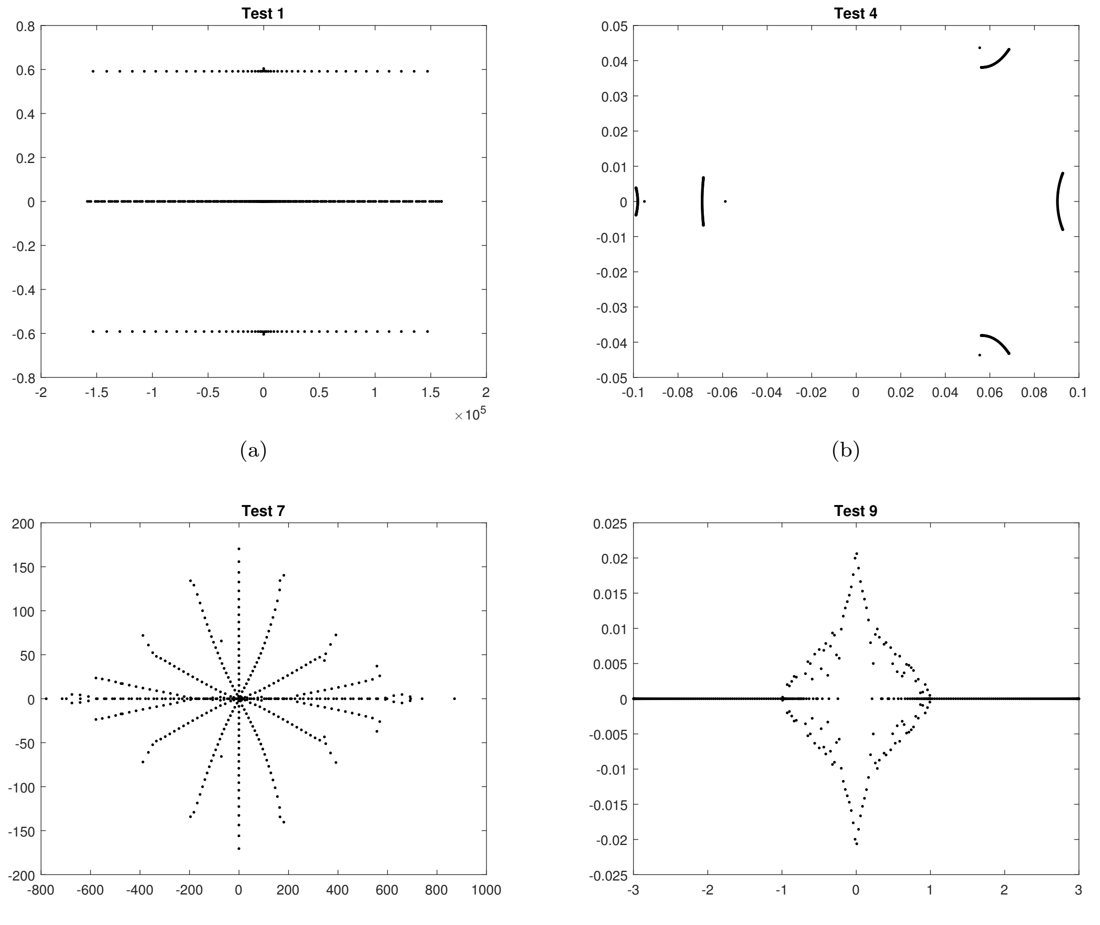


FIGURE 8.3. *Eigenvalues of matrices in Tests 1, 4, 7 and 9 for $n = 400$.*

The extreme relative errors, condition numbers and residual norms for the three codes, Matlab's eig, *BGT* and *3dqds*, are shown in Tables 8.6 and 8.7.

	eig		*BGT*		*3dqds*		
Test	rel_{min}	rel_{max}	rel_{min}	rel_{max}	rel_{mim}	rel_{max}	
1	$2.3\ 10^{-17}$	$3.3\ 10^{-13}$	$9.8\ 10^{-19}$	$2.8\ 10^{-16}$	$9.8\ 10^{-19}$	$1.0\ 10^{-15}$	*(+)*
3	0	$1.1\ 10^{-14}$	0	$1.1\ 10^{-14}$	0	$1.1\ 10^{-14}$	*(+)*
4	$2.8\ 10^{-16}$	$5.5\ 10^{-15}$	$4.3\ 10^{-18}$	$1.1\ 10^{-16}$	$6.6\ 10^{-18}$	$1.4\ 10^{-16}$	*(+ +)*
6	$2.8\ 10^{-18}$	$4.5\ 10^{-13}$	$3.0\ 10^{-19}$	$1.3\ 10^{-13}$	$3.0\ 10^{-19}$	$3.3\ 10^{-14}$	*(+)*
7	$2.2\ 10^{-17}$	$6.1\ 10^{-14}$	$1.9\ 10^{-18}$	$3.5\ 10^{-16}$	$1.5\ 10^{-18}$	$8.0\ 10^{-16}$	*(+)*
9	$3.1\ 10^{-17}$	$1.2\ 10^{-14}$	$1.4\ 10^{-18}$	$6.7\ 10^{-16}$	$1.4\ 10^{-18}$	$3.2\ 10^{-15}$	*(+)*

Table 8.6
Relative errors for matrices in (8.1) *for* $n = 100$.

	relcond$(\lambda; L, U)$		relcond$(\lambda; C)$		*max residuals*	
Test	*min*	*max*	*min*	*max*	$J = LU$	ΔT
1	$1.0\ 10^{0}$	$3.8\ 10^{2}$	$1.0\ 10^{0}$	$1.6\ 10^{2}$	$4.8\ 10^{-12}$	$1.8\ 10^{-11}$
3	$1.0\ 10^{0}$	$2.3\ 10^{0}$	$1.0\ 10^{0}$	$1.1\ 10^{1}$	$4.9\ 10^{-14}$	$1.3\ 10^{-12}$
4	$1.3\ 10^{0}$	$2.8\ 10^{0}$	$1.3\ 10^{0}$	$2.4\ 10^{1}$	$3.5\ 10^{-8}$	$1.3\ 10^{-7}$
6	$1.0\ 10^{0}$	$5.0\ 10^{1}$	$1.0\ 10^{0}$	$4.1\ 10^{3}$	$1.3\ 10^{-10}$	$1.3\ 10^{-10}$
7	$1.5\ 10^{0}$	$5.5\ 10^{2}$	$1.3\ 10^{0}$	$2.4\ 10^{1}$	$3.0\ 10^{-10}$	$1.5\ 10^{-9}$
9	$1.1\ 10^{0}$	$7.2\ 10^{2}$	$1.0\ 10^{0}$	$2.1\ 10^{2}$	$3.3\ 10^{-9}$	$3.3\ 10^{-9}$

Table 8.7
Relative condition numbers and residual norms for matrices in (8.1) *for* $n = 100$.

Table 8.8 reports the CPU time in seconds required by *3dqds* versus the time required by Matlab's eig, *BGT* and *ex3dqds* with n ranging from 400 to 1000. Examples were chosen to represent the best, worst, and average efficiency of *BGT*.

Liu matrix

Z. A. Liu [15] devised an algorithm to obtain one-point spectrum unreduced tridiagonal matrices of arbitrary dimension $n \times n$. These matrices have only one eigenvalue, zero with multiplicity n, the Jordan form consists of one Jordan block and so the eigenvalue condition number is infinite. Our code *3dqds* computes this eigenvalue exactly (and also the generalized eigenvectors) using the following method which is part of the prologue. See [12].

The best place to start looking for eigenvalues of a tridiagonal matrix $C = \text{tridiag}(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c})$ is at the arithmetic mean which we know ($\mu = \text{trace}(C)/n$). Before converting to J-form and factoring, we check whether μ is an eigenvalue by using the 3-term recurrence to solve

$$(\mu I - C)\boldsymbol{x} = \boldsymbol{e}_n p_n(\mu) / \prod_{i=1}^{n-1} c_i .$$

Test; n	eig	*BGT*	*ex3dqds* (+)	*3dqds* (+)
3; 400	0.11	3.12	0.07	0.03
6; 400	0.003	53.0	0.04	0.03
9; 400	0.39	19.5	0.52	0.32
3; 800	0.34	13.5	0.13	0.08
6; 800	0.01	360.2	0.12	0.08
9; 800	1.28	84.5	1.28	0.94
3; 1000	0.77	18.76	0.14	0.10
6; 1000	0.02	443.3	0.14	0.09
9; 1000	2.12	145.0	1.68	1.31

Table 8.8
CPU time in seconds for matrices in Tests 3, 6 and 9.

Here

$$x_1 = 1, \quad x_2 = (\mu - a_2)/c_1, \quad x_{j+1} = \frac{1}{c_j}\left[(\mu - a_j)x_j - b_{j-1}x_{j-1}\right], \quad j = 2, \dots, n-1,$$

and

$$\upsilon := (\mu - a_n)x_n - b_{n-1}x_{n-1}\left(= p_n(\mu)/\prod_{i=1}^{n-1} c_i\right).$$

If, by chance, υ vanishes, or is negligible compared to $\|\boldsymbol{x}\|$, then μ is an eigenvalue (to working accuracy) and $\boldsymbol{x}$ is an eigenvector. To check its multiplicity we differentiate with respect to μ and solve

$$(\mu I - C)\boldsymbol{y} = \boldsymbol{x}$$

with $y_1 = 0$, $y_2 = 1 = x_2'$ $(= x_1)$. If

$$\upsilon' = p_n'(\mu)/\prod_{i=1}^{n-1} c_i := (\mu - a_n)y_n - b_{n-1}y_{n-1} + x_n$$

vanishes, or is negligible w.r.t. $\|\boldsymbol{y}\|$, then we continue the same way until the system is inconsistent or there are n generalized eigenvectors.

Usually $\upsilon \neq 0$ and the calculation appears to have been a waste. This is not quite correct. In exact arithmetic, triangular factorization of $\mu I - C$ or $\mu I - J$, where $J = DCD^{-1}$, will break down if, and only if, x_j vanishes for $1 < j < n$. So our code examines $\min_j |x_j|$ and if it is too small w.r.t. its neighbors and w.r.t. $\|\boldsymbol{x}\|$ then we do not choose μ as our initial shift. Otherwise we do obtain initial L and U from $J - \mu I = LU$.

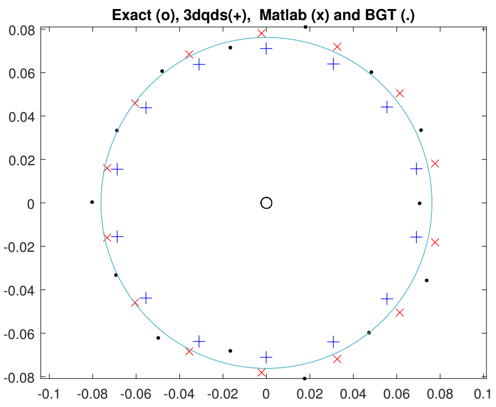
For comparison purposes, we ignored our prologue and give to our *3dqds* code the Liu matrices for $n = 14$ and $n = 28$, tridiag$(\mathbf{1}, \boldsymbol{\alpha}^n, \boldsymbol{\gamma}^n)$ defined by

$$\boldsymbol{\alpha}^{14} = \left[0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0\right],$$
$$\boldsymbol{\gamma}^{14} = \left[-1, 1, 1, -1, 1, -1, -1, -1, 1, -1, 1, 1, -1\right],$$

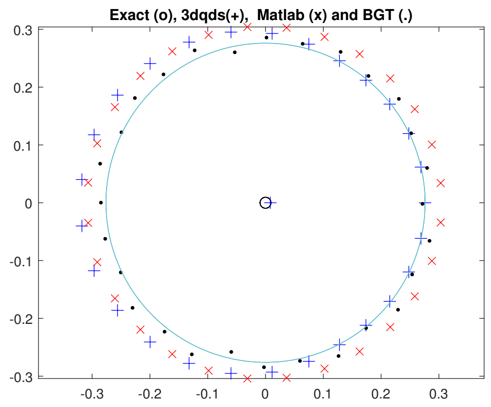
and

$$\boldsymbol{\alpha}^{28} = \left[0,0,0,0,0,0,-1,1,0,0,0,0,0,-1,1,0,0,0,0,0,1,-1,0,0,0,0,0,0\right],$$
$$\boldsymbol{\gamma}^{28} = \left[-1,1,1,-1,1,-1,-1,-1,1,-1,1,1,-1,-1,-1,1,1,-1,1,-1,-1,-1,1,-1,1,1,-1\right].$$

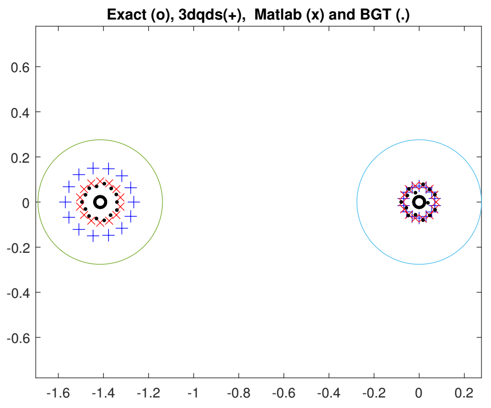
The accuracy of the approximations delivered by *3dqds* is as good as the accuracy of those provided by MATLAB and *BGT*. The absolute errors are $\mathcal{O}(10^{-2})$ for $n = 14$ and $\mathcal{O}(10^{-1})$ for $n = 28$. The number of iterations needed for *3dqds* to converge is less than $3n$. See Figure 8.4(a) and 8.4(b). We show the numerical results along with the circles $z = \sqrt[n]{\varepsilon}$.



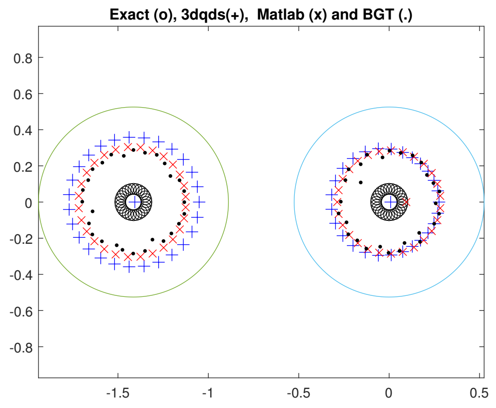
(a) $n = 14$



(b) $n = 28$



(c) $n = 28$



(d) $n = 56$

FIGURE 8.4. *Eigenvalues of Liu matrices, (a) and (b), and glued Liu matrices, (c) and (d).*

We also considered *glued* Liu matrices which are defined as the direct sum of two Liu matrices, shifting one of them by $\sqrt{2}$ and letting the glue between them be ε. Roundoff will give us two clusters, one around 0, the other around $\sqrt{2}$. This is not a one-point spectrum matrix and all three methods give the results expected by perturbation theory. See Figures 8.4(c) and 8.4(d). This is a very unstable example, the condition numbers all exceed 10^{10}.

9. Conclusions. Following the broad success of the HQR algorithm to compute eigenvalues of real square matrices it seems natural to use a sequence of similarity transforms to reduce an initial real matrix to eventual triangular form and also deflate eigenvalues from the bottom of the matrix as they converge. Any real (unreduced) tridiagonal matrix is easily put into J-form (all superdiagonal entries are 1) and such matrices ask for the use of the LR (not QR) algorithm since it preserves the J-form. The potential breakdown of the LR transform, from a 0 pivot, was a strong deterrent in the early days (1960s) but today is a mild nuisance as explained in Section 5.1. A further incentive is that the whole procedure can be carried out in real arithmetic since complex conjugate pairs of eigenvalues are determined from 2×2 submatrices that converge and may be deflated in a manner similar to real eigenvalues. The more recent success of the *dqds* transform in computing singular values of bidiagonal matrices encouraged us to keep out J matrices in factored form: $J - \sigma I = LU$, $\widehat{J} = UL$, because, in exact arithmetic, the two algorithms, LR and *dqds*, are equivalent. In addition the *dqds* transform of today is numerically superior to the original, and seminal, *qd* transform discovered by Heinz Rutishauser in 1954 [27] and which gave rise to the LR algorithm itself.

In order to hasten convergence we will need to apply complex conjugate pairs of shifts to our current $LU = J$ matrix. It is well known how to do this entirely in real arithmetic in the context of the LR algorithm. To the best of our knowledge this has not been tried in the context of *dqds*. The main contribution of this paper is the solution to this challenge. We realized that three, not two, transforms are required to return to real factors L and U when complex shifts are applied consecutively. This is the nature of our explicit version, a local detour invoking complex arithmetic. We went further and produced a subprogram *3dqds* that accomplishes the same goal but in (exact) real arithmetic. This implicit version is more efficient than the explicit but is sensitive to roundoff error in its initial step. Experts will recall the papers on "washout of the shift" in the implicit shift HQR algorithm in the 1980s. We can not prove that our algorithm is backward stable. In fact we dought that it is. However we do show that the three parts of the inner loop separately enjoy high mixed relative stability.

In the process of implementing our new features we were led to a novel and detailed criterion for deciding when our $J = LU$ matrix has split into two or more unreduced submatrices. We check for splits at every iteration. Our new subprograms must only be applied to unreduced matrices. We also gave attention to the choice of a new shift when a factorization fails and when to start using the bottom 2×2 submatrix for shifting.

We save a lot of space by confining our eigenvector calculations to the ΔT form so that only one vector need be stored. From it we can compute the relative condition numbers that we need. Instructions are given how to generate the eigenvectors for the original and the J-form represention s.

10. Acknowledgments. The authors would like to thank Associate Editor Martin H. Gutknecht and the anonymous referees for forcing us to look more deeply into an error analysis of our *triple dqds* algorithm (first version) and to give a clearer presentation of its mathematical analysis and implemention details (last version).

REFERENCES

[1] D. A. Bini, L. Gemignani and F. Tisseur. *The Ehrlich-Aberth method for the nonsymmetric tridiagonal eigenvalue problem.* SIAM J. Matrix Anal. Appl., 27(1):153-175, 2005.

[2] A. Bunse-Gerstner. *An analysis of the HR algorithm for computing the eigenvalues of a matrix.* Linear Algebra and Its Applications, 35:155-173, 1981.

[3] P. A. Clement. *A class of triple-diagonal matrices for test purposes.* SIAM Review, 1 (vol.1), January, 1959.

[4] J. K. Cullum.*A QL procedure for computing the eigenvalues of complex symmetric tridiagonal matrices.* SIAM J. Matrix Anal. Appl., 17(1):83-109, 1996.

[5] D. Day. *Semi-duality in the two-sided lanczos algorithm.* Ph.D thesis, University of California, Berkeley, 1993.

[6] J. W. Demmel. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[7] I. S. Dhillon and B. N. Parlett. *Multiple representations to compute orthogonal egenvectors of symmetric tridiagonal matrices.* Linear Algebra and its Applications, 387:1-28, 2004.

[8] I. S. Dhillon and B. N. Parlett. *Orthogonal eigenvectors and relative gaps.* SIAM J. Matrix Anal. Appl., 25:858-899, 2004.

[9] K. V. Fernando and B. Parlett. *Accurate singular values and differential qd algorithms.* Numerische Mathematik, 67:191-229, 1994.

[10] K. V. Fernando. *On computing an eigenvector of a tridiagonal matrix. Part I: Basic results.* SIAM J. Matrix Anal. Appl., 18:1013-1034, 1997.

[11] C. Ferreira. *The unsymmetric tridiagonal eigenvalue problem.* Ph.D Thesis, University of Minho, 2007. `http://hdl.handle.net/1822/6761`

[12] C. Ferreira and B. Parlett. *Convergence of LR algorithm for a one-point spectrum tridiagonal matrix.* Numerische Mathematik, 113(3):417-431, 2009.

[13] C. Ferreira, B. Parlett and Froilán M. Dopico. *Sensitivity of Eigenvalues of an unsymmetric tridiagonal matrix.* Numerische Mathematik, DOI: 10.1007/s00211-012-0470-z, April 2012.

[14] J. G. F. Francis. *The QR transformation - a unitary analogue to the LR transformation, Parts I and II.* Computer Journal, 4:265-272 and 332-245, 1961/62.

[15] Z. A. Liu. *On the extended HR algorithm.* Technical Report PAM-564, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, USA, 1992.

[16] W. Kahan, B. N. Parlett and E. Jiang. *Residual bounds on approximate eigensystems of non-normal matrices.* SIAM Journal on Numerical Analysis, 19:470-484, 1982.

[17] N. Newman and J. Todd. *The Evaluation of Matrix Inversion Programs.* Journal of SIAM, 4(6):466-476, 1958.

[18] B. N. Parlett and C. Reinsch. *Balancing a matrix for calculation of Eigenvalues and Eigenvectors.* Numerische Mathematik, 13:292-304, 1969.

[19] B. N. Parlett. *The rayleigh quocient iteration and some generalizations for non-normal matrices.* Mathematics of computation, 28(127):679-693, 1974.

[20] B. N. Parlett. *The contribution of J. H. Wilkinson to numerical analysis.* A history of Scientific Computing, p. 25, Edited by Stephen G. Nash, ACM Press, 1990.

[21] B. N. Parlett. *Reduction to tridiagonal form and minimal realizations*, SIAM J. Matrix Anal. Appl., 13:567-593, 1992.

[22] B. N. Parlett. *The new qd algorithms.* Acta Numerica, 459-491, 1995.

[23] B. N. Parlett and I. S. Dhillon. *Fernando's Solution to Wilkinson's Problem: an Application of Double Factorization.* Linear Algebra and Its Applications, 267:247-279, 1997.

[24] B. N. Parlett and Osni A. Marques. *An implementation of the dqds algorithm.* Linear Algebra and Its Applications, 309:217-259, 2000.

[25] B. Parlett, Froilán M. Dopico and and C. Ferreira. *The inverse eigenvector problem for real tridiagonal matrices.* SIAM J. Matrix Anal. Appl., 37(2):577–597, 2016.

[26] L. Pasquini. *Accurate computation of the zeros of the generalized Bessel polynomials.* Numerische Mathematic, 86:507-538, 2000.

[27] H. Rutishauser. *Der Quotienten-Differenzen-Algorithmus.* Z. angew. Math. Physik 5:233-251, 1954. Cited in [36].

[28] H. Rutishauser. *Der Quotienten-Differenzen-Algorithmus.* Mitt. Inst. angew. Math. ETH, n°.7, Birkhäuser, Basel, 1957. Cited in [36].

[29] H. Rutishauser. *Solution of eigenvalue problems with the LR-transformation.* National Bureau of Standards Applied Mathematics series 49:47-81, 1958.

[30] H. Rutishauser and H.R. Schwarz. *The LR transformation method for symmetric matrices.* Numerische Mathematik, 5:273-289, 1963.

[31] J. Slemons. *Toward the solution of the eigenproblem: nonsymmetric tridiagonal matrices.* Ph.D thesis, University of Washington, Seattle, 2008.

[32] J. Slemons. *The Result of Two Steps of the LR Algorithm is Diagonally Similar to the Result of One Step of the HR Algorithm* . SIAM J. Matrix Anal. Appl., 31(1):68–74, 2009.

[33] J. Todd. *The condition of finite segments of the Hilbert matrix. Contributions to the solution of systems of linear equations and the determination of eigenvalues.* National Bureau of Standards Applied Mathematics, 39:109-116, 1954.

[34] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra. The Behavior of Nonnormal Matrices and Operators.* Princeton University Press, 2005.

[35] F. K. Vince and B. N. Parlett. *Accurate singular values and differential qd algorithms.* Numerische Mathematic, 67:191-229, 1994.

[36] D. S. Watkins. *QR-like algorithms - An overview of convergence theory and practice.* Lectures in Applied Mathematics, 32:879-893, 1996.

[37] D. S. Watkins and L. Elsner. *Convergence of algorithms of decomposition type for the eigenvalue problem.* Linear Algebra and Its Applications, 143:19-47, 1991.

[38] P. R. Willems and B. Lang. *Twisted factorizations and qd-type transformations for the MR^3 algorithm - new representations and analysis.* SIAM J. Matrix Anal. Appl., 33(2):523-553, 2012.

[39] Z. Wu. *The Triple dqds Algorithm for Complex Eigenvalues.* Ph.D thesis, University of California, Berkeley, 1996.

[40] H. Xu. *The relation between the QR and LR algorithms.* SIAMJ. Matrix Anal. Appl., 19(2):551-555, 1998.

[41] Yao Yang. *Error Analysis of the qds and dqds Algorithms.* Ph.D thesis, University of California, Berkeley, 1994.

Appendix A. *3dqds* algorithm.

```
$[\widehat{\boldsymbol{l}}, \widehat{\boldsymbol{u}}] = \boldsymbol{3dqds}(\boldsymbol{l}, \boldsymbol{u}, \texttt{sum}, \texttt{prod})$

% $\texttt{sum} = (\sigma_1 + \sigma_2)$; $\texttt{prod} = \sigma_1\sigma_2$
% $\boldsymbol{l} = [l_1, l_2, \dots, l_{n-1}]$; $\boldsymbol{u} = [u_1, u_2, \dots, u_n]$
% $\widehat{\boldsymbol{l}} = [\widehat{l}_1, \widehat{l}_2, \dots, \widehat{l}_{n-1}]$; $\widehat{\boldsymbol{u}} = [\widehat{u}_1, \widehat{u}_2, \dots, \widehat{u}_n]$

% step 1
$x_r = 1$;  $y_r = l_1$;  $z_r = 0$
% the effect of $Z_1$
$x_r = x_r * u_1 + y_r$
% the matrix $\mathcal{L}_1^{-1}$
$x_l = (u_1 + l_1)^2 + u_2 l_1 - \texttt{sum}(u_1 + l_1) + \texttt{prod}$
$y_l = -u_2 l_1 u_3 l_2 / x_l$
$x_l = -u_2 l_1 (u_1 + l_1 + u_2 + l_2 - \texttt{sum}) / x_l$
% the effect of $\mathcal{L}_1$
$\widehat{u}_1 = x_r - x_l$;
$x_r = y_r - x_l$;  $y_r = z_r - y_l - x_l * l_2$;
$z_r = -y_l * l_3$
% the matrix $Y_1^{-1}$
$x_r = x_r / \widehat{u}_1$;  $y_r = y_r / \widehat{u}_1$;  $z_r = z_r / \widehat{u}_1$
% the effect of $Y_1^{-1}$
$\widehat{l}_1 = x_l + y_r + x_r * u_2$
$x_l = y_l + z_r + y_r * u_3$;  $y_l = z_r * u_4$
% the effect of $Y_1$
$x_r = 1 - x_r$;  $y_r = l_2 - y_r$;  $z_r = -z_r$

% steps 2 to n-3
for $i = 2, \dots, n-3$
    % the effect of $Z_i$
    $x_r = x_r * u_i + y_r$
    % the matrix $\mathcal{L}_i^{-1}$
    $x_l = -x_l / \widehat{l}_{i-1}$;  $y_l = -y_l / \widehat{l}_{i-1}$;
    % the effect of $\mathcal{L}_i$
    $\widehat{u}_i = x_r - x_l$;
    $x_r = y_r - x_l$;  $y_r = z_r - y_l - x_l * l_{i+1}$;
    $z_r = -y_l * l_{i+2}$
    % the matrix $Y_i^{-1}$
    $x_r = x_r / \widehat{u}_i$;  $y_r = y_r / \widehat{u}_i$;  $z_r = z_r / \widehat{u}_i$
    % the effect of $Y_i^{-1}$
    $\widehat{l}_i = x_l + y_r + x_r * u_{i+1}$
    $x_l = y_l + z_r + y_r * u_{i+2}$;  $y_l = z_r * u_{i+3}$
    % the effect of $Y_i$
    $x_r = 1 - x_r$;  $y_r = l_{i+1} - y_r$;  $z_r = -z_r$
end for

% step n-2
% the effect of $Z_{n-2}$
$x_r = x_r * u_{n-2} + y_r$
% the matrix $\mathcal{L}_{n-2}^{-1}$
$x_l = -x_l / \widehat{l}_{n-3}$;  $y_l = -y_l / \widehat{l}_{n-3}$;
% the effect of $\mathcal{L}_{n-2}$
$\widehat{u}_{n-2} = x_r - x_l$;
$x_r = y_r - x_l$;  $y_r = z_r - y_l - x_l * l_{n-1}$
% the matrix $Y_{n-2}^{-1}$
$x_r = x_r / \widehat{u}_{n-2}$;  $y_r = y_r / \widehat{u}_{n-2}$
% the effect of $Y_{n-2}^{-1}$
$\widehat{l}_{n-2} = x_l + y_r + x_r * u_{n-1}$
$x_l = y_l + y_r * u_n$
% the effect of $Y_{n-2}$
$x_r = 1 - x_r$;  $y_r = l_{n-1} - y_r$

% step n-1
% the effect of $Z_{n-1}$
$x_r = x_r * u_{n-1} + y_r$
% the matrix $\mathcal{L}_{n-1}^{-1}$
$x_l = -x_l / \widehat{l}_{n-2}$
% the effect of $\mathcal{L}_{n-1}$
$\widehat{u}_{n-1} = x_r - x_l$;
$x_r = y_r - x_l$
% the matrix $Y_{n-1}^{-1}$
$x_r = x_r / \widehat{u}_{n-1}$
% the effect of $Y_{n-1}^{-1}$
$\widehat{l}_{n-1} = x_l + x_r * u_n$
% the effect of $Y_{n-1}$
$x_r = 1 - x_r$

% step n
% the effect of $Z_n$
$x_r = x_r * u_n$
% the matrix $\mathcal{L}_n^{-1} = I$
% the effect of $\mathcal{L}_n$
$\widehat{u}_n = x_r$;
% the matrix $Y_n^{-1} = I$
```

Appendix B. Pseudocode for the whole algorithm.

Algorithm 1 **wrapper for 3dqds**

```
Input: vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$
Output: eigenvalues of tridiag($\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c}$)

  top = 1                                    ▷ code works on submatrix top : $n$
  split(1) = top                             ▷ vector split saves all active top's
  indsplit = 1                               ▷ index for split
  nits = 0                                   ▷ number of iterations
  itmax = $100n$                             ▷ maximum number of iterations
  acshift = 0                                ▷ accumulated shift; simple dqds is not restoring

  find $\boldsymbol{l}$ and $\boldsymbol{u}$ of $J$ form   [Algorithm 5]   ▷ vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ for $J = LU$
  while (top + 1 < $n$  and  its < itmax) do        ▷ code should mantain top + 1 < $n$
      deflate as warranted   [Algorithm 2]          ▷ deflation may reduce $n$
      find splits, if any   [Algorithm 3]           ▷ splits may increase top
      if ($l_{n-1} > 10^{-2}$  and  $l_{n-2} > 10^{-2}$) then   ▷ entries at the bottom are not small
          [$\boldsymbol{l_1}, \boldsymbol{u_1}$, fail] = dqds($\boldsymbol{l}$(top : $n$), $\boldsymbol{u}$(top : $n$), 0)   ▷ simple dqds with zero shift
          if fail then                              ▷ fail is a boolean for failure
              [$\boldsymbol{l_1}, \boldsymbol{u_1}$, shift, fail] = recover($\boldsymbol{l}$(top : $n$), $\boldsymbol{u}$(top : $n$))   [Algorithm 4]
          end if
      else
          sum = $l_{n-1} + (u_{n-1} + u_n)$
          prod = $u_{n-1}u_n$
          [$\boldsymbol{l_1}, \boldsymbol{u_1}$, fail] = 3dqds($\boldsymbol{l}$(top : $n$), $\boldsymbol{u}$(top : $n$), sum, prod)   ▷ triple dqds
          if fail then
              [$\boldsymbol{l_1}, \boldsymbol{u_1}$, shift, fail] = recover($\boldsymbol{l}$(top : $n$), $\boldsymbol{u}$(top : $n$), sum, prod)
          end if
      end if
      if fail then
          return "too many failures, no convergence."
      end if
      $\boldsymbol{l} = \boldsymbol{l_1}$; $\boldsymbol{u} = \boldsymbol{u_1}$
      acshift=acshift+shift                         ▷ update accumulated shift
      its=its+1
  end while
```

Algorithm 2 deflation body

```
                                          ▷ deflate as warranted; deflation may reduce $n$

tol $= 10\varepsilon$                     ▷ tolerance for deflation; $\varepsilon$ = roundoff unit

repeat
    if $criteria_{2\times 2}$ then        ▷ deflation $2\times 2$ criteria (9.8) and (9.9)
        ssum $= (l_{n-1} + (u_{n-1} + u_u))/2$
        disc $= \big((l_{n-1} + (u_{n-1} - u_u))/2\big)^2 + u_n l_{n-1}$    ▷ discriminant
        $t = \sqrt{|\texttt{disc}|}$
        if disc $< 0$ then                ▷ complex conjugate pair
            $x_1 =$ ssum $+ \boldsymbol{i}t$
            $x_2 =$ ssum $- \boldsymbol{i}t$         ▷ no use of complex arithmetic
        else if ssum $== 0$ then          ▷ real pair
            $x_1 = t$
            $x_2 = -t$
        else
            $x_1 = \text{sign}(\texttt{ssum}) * \big(|\texttt{ssum}| + t\big)$    ▷ no subtractions
            $x_2 = u_{n-1}u_n/x_1$
        end if
        eigvals($[n-1, n]$) $= [x_1, x_2] +$ acshift    ▷ eigvals stores the eigenvalues
        $n = n - 2$
    else if $criteria_{1\times 1}$ then   ▷ deflation $1\times 1$ criteria (6.1) – (6.4)
        eigvals($n$) $=$ acshift $u_n +$ acshift
        $n = n - 1$
    end if
until deflation criteria not met
```

Algorithm 3 splitting body

```
                                          ▷ find splits, if any, define top after a split

tol $= 10\varepsilon$                     ▷ tolerance for splitting; $\varepsilon$ = roundoff unit

if $n >$ top $+ 2$ then
    $k = n - 3$
    while $\big(k >$ top and $criteria_{split}$ not met$\big)$ do    ▷ splitting criteria (9.6) and (9.7)
        $k = k - 1$
    end while
    if $k >$ top then                     ▷ there is a split
        indsplit $=$ indsplit $+ 1$
        split(indsplit) $=$ top
        $l_k =$ acshift                   ▷ $l_k$ saves accumulated shift of the previous segment
        top $= k + 1$
    end if                                ▷ if the condition for splitting is not met, there is nothing to do
end if
```

Algorithm 4 recover

Input: vectors $\boldsymbol{l}, \boldsymbol{u}$, real `sum`, `prod` (or vectors $\boldsymbol{l}, \boldsymbol{u}$)
Output: vectors $\boldsymbol{l_1}, \boldsymbol{u_1}$, real `shift`, boolean `fail`

```
$\delta = \sqrt{\varepsilon}$                              ▷ shift increment; $\varepsilon$ = roundoff unit
if nargin == 2 then                          ▷ nargin = number of input arguments
    simple = true                            ▷ failure in dqds with zero shift
    sum = $\delta$                                ▷ sum and prod for 3dqds
    prod = $\delta$
    shift = 0                                ▷ in case of successive failures
else
    simple = false                           ▷ failure in 3dqds
    shift = $u_n$                                 ▷ shift for simple dqds
end if

fail=true                                    ▷ boolean for failure
nfail = 0                                    ▷ number of failures
maxfail = $10n$                                ▷ maximum number of failures allowed

while (fail and nfail < maxfail) do          ▷ increase shift and reverse choice of transform
    if simple then
        sum = sum$(1 + \delta)$
        prod = prod$(1 + \delta)^2$
        $[\boldsymbol{l_1}, \boldsymbol{u_1},$ fail$]$ = 3dqds($\boldsymbol{l}, \boldsymbol{u}$, sum, prod)    ▷ switch to 3dqds
        simple = false
        if fail==false then                  ▷ successful recovery with 3dqds
            shift = 0
        end if
    else
        shift = shift + $\delta$
        $[\boldsymbol{l_1}, \boldsymbol{u_1},$ fail$]$ = dqds($\boldsymbol{l}, \boldsymbol{u}$, shift)         ▷ switch to simple dqds
        simple = true
    end if
    nfail=nfail+1
end while                                    ▷ after a failure the oposite transform is used next
```

Algorithm 5 initial *LU* factorization

```
Input: vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$        ▷ LU factorization of tridiag$(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{c})$ in J form
Output: vectors $\boldsymbol{l}, \boldsymbol{u}$, real shift, boolean fail

  nfail $= 0$                                   ▷ number of failures
  maxfail $= 10n$                               ▷ maximum number of failures allowed
  $\boldsymbol{b} = \boldsymbol{b} \cdot * \boldsymbol{c}$                         ▷ element-wise product; off-diagonal of J
  delta $= \min\big(1/2, 2 * \min(\mathrm{abs}(\boldsymbol{a}(\boldsymbol{a}\sim = 0)))\big)$     ▷ shift increment in case of failure
                                   ▷ one eight of the minimum nonzero diagonal element

  shift $= 0$                                   ▷ in case of failure take $J - \mathit{shift} \cdot I = LU$
  $[\boldsymbol{l}, \boldsymbol{u}, \mathtt{fail}] = LUfact(\boldsymbol{a}, \boldsymbol{b}, \mathtt{shift})$   [Algorithm 6]      ▷ LU factorization of J
  while (fail and nfail < maxfail) do
    nfail=nfail+1
    shift = shift + delta                       ▷ after a failure the shift is increased
    $[\boldsymbol{l}, \boldsymbol{u}, \mathtt{fail}] = LUfact(\boldsymbol{a}, \boldsymbol{b}, \mathtt{shift})$
  end while

  if fail then
    return "Too many failures, no initial factorization."
  end if
```

Algorithm 6 *LUfact*

```
Input: vectors $\boldsymbol{a}, \boldsymbol{b}$, real shift       ▷ LU factorization of $J =$ tridiag$(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{1})$ without pivoting
Output: vectors $\boldsymbol{l}, \boldsymbol{u}$, boolean fail

  tolg $= 1/\sqrt{\varepsilon}$                  ▷ tolerance for element growth; $\varepsilon =$ roundoff unit
  fail=false                                    ▷ boolean for failure

  $\boldsymbol{u}(1) = \boldsymbol{a}(1)$
  for i $= 1 : n-1$ do
    $\boldsymbol{l}(i) = \boldsymbol{b}(i)/\boldsymbol{u}(i)$
    $\boldsymbol{u}(i+1) = \boldsymbol{a}(i+1) - \boldsymbol{l}(i)$
  end for

  if (any(isnan([$\boldsymbol{l}, \boldsymbol{u}$])) or any(abs([$\boldsymbol{l}, \boldsymbol{u}$])) >tolg ) then     ▷ checking for element growth
    fail=true
  end if
```