An Implementation of the dqds Algorithm (Positive Case)*

Beresford N. Parlett[†] and Osni A. Marques[‡]

Abstract

The dqds algorithm was introduced in 1994 to compute singular values of bidiagonal matrices to high relative accuracy but it may also be used to compute eigenvalues of tridiagonal matrices. This paper discusses in detail the issues that have to be faced when the algorithm is to be realized on a computer: criteria for accepting a value, for splitting the matrix, and for choosing a shift to reduce the number of iterations, as well as the relative advantages of using IEEE arithmetic when available. Ways to avoid unnecessary over/underflows are described.

In addition some new formulae are developed to approximate the smallest eigenvalue from a twisted factorization of a matrix. The results of extensive testing are presented at the end. The list of contents is a valuable guide to the reader interested in specific features of the algorithm.

*This work was partly supported by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract No. DE-AC03-76SF00098, and partly by the National Science Foundation Cooperative Agreement No. ACI-9619020, NSF Grant No. ACI-9813362, and the Department of Energy Grant No. DE-FG03-94ER25219. The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

[†]Mathematics Department and Computer Science Division, EECS Department, University of California, Berkeley, CA 94720, USA.

[‡]National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA.

Contents

1 The dqds Transform

H. Rutishauser introduced the qd algorithm (*not* dqds) and the Z notation in 1953/54, see [9], in connection with transformations of continued fractions. However our interpretation is aimed at tridiagonal matrices. Write

$$Z = (q_1, e_1, q_2, e_2, \ldots, e_{n-1}, q_n, e_n)$$

but, by convention, $e_n = 0$. We associate with Z two bidiagonal matrices

$$U = \text{bidiag}\begin{pmatrix} & 1 & & 1 & & . & & 1 & & 1 & \\ q_1 & & q_2 & & . & & . & & q_{n-1} & & q_n \end{pmatrix}$$

and

$$L = \text{bidiag}\begin{pmatrix} 1 & & 1 & & 1 & & . & & 1 & & 1 \\ & e_1 & & e_2 & & . & & . & & e_{n-1} & \end{pmatrix}.$$

Rutishauser's *qds* transform (qd with shifts) and the *dqds* transform each with shift τ map Z into $\hat{Z}$ where the associated bidiagonals $\hat{L}$, $\hat{U}$ satisfy

$$\hat{L}\hat{U} = UL - \tau I$$

provided that the transformation does not break down. However the formulae in *dqds* are different from those in *qds* as shown later in this section. The algorithms consist of repeated applications of the transforms with various shifts τ and it is obligatory to introduce σ, the accumulated sum of all shifts τ used so far. At any stage in the algorithm, the current qd-array Z and the current σ define a matrix

$$\sigma I + LU$$

that has the same eigenvalues as the matrix LU associated with the original Z. In exact arithmetic if initially Z is positive, and if all $\tau = 0$, then Z converges, very slowly, to

$$(\lambda_1, 0, \lambda_2, \ldots, \lambda_{n-1}, 0, \lambda_n, 0)$$

where $\lambda_1 > \lambda_2 > \ldots \lambda_n > 0$, are the wanted eigenvalues. The shift τ is used to hasten convergence. In practice the algorithm uses *deflation*: as soon as e_{n-1} is negligible q_n is declared an eigenvalue and n is replaced by $n-1$.

In exact arithmetic the LR, *qds* and *dqds* transforms are the same but the advantage of *qds* and *dqds* is that L and U together hold more information than the product UL used by LR and the advantage of *dqds* over *qds* is that, in the positive case, in finite precision, *dqds* preserves the eigenvalues to high relative accuracy (in the absence of underflow) whereas *qds* does not. Next we offer a few historical remarks.

The first d in *dqds* stands for differential- a somewhat misleading adjective coined by Rutishauser in his notes. The algorithm is quite distinct from the LR and QR *flows* introduced in the 1980's. See [1] and [12], [13].

Rutishauser never used the *dqds* transform except with $\tau = 0$ and he seems to have invoked that option (we call it *dqd*) only when his *preferred*, and faster algorithm, *qds* or "qd with shifts", got into difficulties. He never published the *dqd* algorithm. See [11, Appendix]. The lower case letters qd stand for quotient-difference, the name he chose in 1953/54 for his operationally minimal implementation.

The *dqds* algorithm was rediscovered independently by Fernando and Parlett in 1992 and they showed that the extra multiplication, compared to Rutishauser's *qds*, allowed *dqds* to compute all the eigenvalues, however small, to high relative accuracy. See [3]. Here ends the historical commentary.

Here is the transform applied to a segment of Z, $Z(i0 : n0)$, with shift τ.

$$
\begin{array}{ll}
\text{dqds } (1\div): & d = q(i0) - \tau \\
& \textbf{for } i = i0, n0-1 \textbf{ do} \\
& \quad \hat{q}(i) = d + e(i) \\
& \quad temp = q(i+1)/\hat{q}(i) \\
& \quad \hat{e}(i) = e(i) * temp \\
& \quad d = d * temp - \tau \\
& \textbf{end for}
\end{array}
$$

For contrast we present Rutishauser's qd transform with shift

$$
\begin{array}{ll}
\text{qds:} & \hat{q}(i0) = q(i0) + e(i0) - \tau \\
& \textbf{for } i = i0, n0-1 \textbf{ do} \\
& \quad \hat{e}(i) = e(i) * q(i+1)/\hat{q}(i) \\
& \quad \hat{q}(i+1) = (q(i+1) - \hat{e}(i)) + e(i+1) - \tau \\
& \textbf{end for}
\end{array}
$$

No intermediate variables are needed in *qds* and the arithmetic effort is minimal. Note that the intermediate quantity in *dqds* satisfies

$$d(i+1) = q(i+1) - \hat{e}(i) - \tau.$$

The initial array Z is rarely the primary data. For example, to compute the singular values of a bidiagonal

$$B = \text{bidiag}\left\{ \begin{array}{ccccccccc} & b_1 & & b_2 & \cdot & b_{n-2} & & b_{n-1} & \\ a_1 & & a_2 & & \cdot & \cdot & a_{n-1} & & a_n \end{array} \right\}$$

one defines $q_i = a_i^2$, $e_i = b_i^2$, $i = 1, \ldots, n$, and remembers, at the end, to take the square roots of the eigenvalues (of B^*B) which are computed by the dqds algorithm.

Given a symmetric tridiagonal matrix T with diagonal entries α_i, off diagonals β_i, $i = 1, \ldots, n$ and a scalar ρ such that $\rho I + T$ is positive definite one computes Z by Gaussian elimination as follows.

$q_1 = \alpha_1 + \rho$
for $j = 1, n-1$ **do**
  $e_j = (\beta_j / q_j) \cdot \beta_j$
  $q_{j+1} = \alpha_{j+1} - e_j + \rho$
end for

An alternative, careful, expression for q_{j+1} is

$$q_{j+1} = (\max(\alpha_{j+1}, \rho) - e_j) + \min(\alpha_{j+1}, \rho).$$

One must remember to subtract ρ from the eigenvalues computed by the algorithm in order to recover those of T.

1.1 Overflow

In [3] it was shown that *dqds* preserves eigenvalues to high relative accuracy in the absence of overflow and underflow. In this section we identify and eliminate those exceptions that are "unnecessary".

The concerns of this subsection arise almost exclusively in single precision where the exponent range is so small that $macheps^{-6}$ overflows and $macheps^{6}$

underflows. The rest of the paper is independent of the material presented here. Example 1 is important to the understanding of *dqds* when the exponent range is narrow.

If Z is positive and if $\tau < \lambda_{min}$ then the new qd-array $\hat{Z}$ computed by *dqds* will also be positive. If $\tau > \lambda_{min}$ then $d_{min} = \min_i d_i$ will be negative and if some $\hat{q}(i) = 0$ then the next $d = \infty$ and the one after that is $\infty \cdot 0 - \tau$ which is recorded (in IEEE conforming arithmetic units) as NaN (Not a Number). See [4].

When $d_{min} > 0$ then the new variables $\hat{q}(i)$ and $\hat{e}(i)$ are bounded by old values: $\hat{e}(i) \leq q(i+1)$, $\hat{q}(i) = d_i + e(i) \leq q(i+1) + e(i)$. However for the variable $temp$ we can only say

$$\begin{aligned} temp_i &:= q(i+1)/(d_i + e(i)) \\ &\leq q(i+1)/e(i), \\ &\leq q_{max}/e_{min}. \end{aligned}$$

Thus there is danger of overflow unless the quantities q_{max} and e_{min} are monitored. Sometimes reversal of the qd array (see Section 6.2) can avert an overflow and sometimes a careful check for splitting (see Section 3) can allow a tiny $e(i)$ to be neglected. Unfortunately these measures are not sufficient to avoid all overflows and a small example is given next. Suppose that 10^{38} and 10^{-38} are the thresholds for overflow and underflow.

Example 1

q	e	d (true)	$\hat{q}$	$\hat{e}$	$temp$
10^{-25}	10^{20}	10^{-25}	10^{20}	10^{20}	1
10^{20}	10^{-25}	10^{-25}	$2 10^{-25}$	$\frac{1}{2} 10^{20}$	$\frac{1}{2}(10^{20}/10^{-25}) =$ overflow
10^{20}	10^{20}	$\frac{1}{2} 10^{20}$	$\frac{3}{2} 10^{20}$	$\frac{2}{3} 10^{-25}$	$\frac{2}{3}(10^{-25}/10^{20}) =$ underflow
10^{-25}	0	$\frac{1}{3} 10^{-25}$	$\frac{1}{3} 10^{-25}$	0	$-----$

Even though $\hat{Z}$ is well defined the algorithm *dqds* $(1\div)$ in Section 1 provokes overflow in $temp$. Looking ahead to Section 3 we can say that even though $e_2 = 10^{-25}$ appears to be negligible compared to its neighbors the criterion for setting e_2 to zero is not satisfied and so the replacement of e_2 by 0 would provoke large relative changes in the smaller eigenvalues. Note that the determinant $\Pi_i q_i$ is preserved by the transformation.

The obvious remedy in this case is simple but expensive.

$$
\begin{array}{ll}
\text{dqds } (2\div): & d = q(i0) - \tau \\
& \textbf{for } i = i0, n0-1 \textbf{ do} \\
& \quad \hat{q}(i) = d + e(i) \\
& \quad \hat{e}(i) = q(i+1) * (e(i)/\hat{q}(i)) \\
& \quad d = q(i+1) * (d/\hat{q}(i)) - \tau \\
& \textbf{end for}
\end{array}
$$

The quotients are bounded by one and overflow will not occur. If $\tau = 0$ or $dmin > 0$ then no intermediate quantity exceeds $\lambda_{max}(LU)$.

Unfortunately *dqds* $(2\div)$ escapes the disaster (Scylla) of overflow only to fall into the misfortune (Charybdis) of underflow.

q	e	d (comp)	$\hat{q}$ (comp)	$\hat{e}$ (comp)
10^{-25}	10^{20}	10^{-25}	10^{20}	10^{20}
10^{20}	10^{-25}	$10^{20}(10^{-25}/10^{20}) = 0$	10^{-25}	10^{20}
10^{20}	10^{20}	0	10^{20}	10^{-25}
10^{-25}	0	0	0	0

Note that the determinant $\Pi_i \hat{q}_i = 0$ instead of 10^{-10}! The small eigenvalues of $\hat{Z}$ have huge relative errors.

There is a way out of the difficulty: test at each step. The parameter $sfmin$ is the smallest number whose reciprocal is representable.

$$
\begin{array}{ll}
\text{dqds (safe):} & d = q(i0) - \tau \\
& \textbf{for } i = i0, n0-1 \textbf{ do} \\
& \quad \hat{q}(i) = d + e(i) \\
& \quad \textbf{if } (\hat{q}(i) = 0) \textbf{ then} \\
& \quad\quad \hat{e}(i) = 0 \\
& \quad\quad d = q(i+1) - \tau \\
& \quad \textbf{else if } (safemin * q(i+1) \le \hat{q}(i)) \textbf{ then} \\
& \quad\quad temp = q(i+1)/\hat{q}(i) \\
& \quad\quad \hat{e}(i) = e(i) * temp
\end{array}
$$

$d = d * temp - \tau$

else

$\hat{e}(i) = q(i+1) * (e(i)/\hat{q}(i))$

$d = q(i+1) * (d/\hat{q}(i)) - \tau$

end if

end for

This algorithm produces the correct values and, in general, is close in arithmetic operations to *dqds* (1÷) but it does suffer from tests in the inner loop.

There is a subtle point to be made here. If τ exceeds λ_{min} because of an aggressive shift strategy then a d can be negative and a $\hat{q}$ can vanish. Our code expects this to happen and reacts appropriately. However when $\tau = 0$ then, in exact arithmetic, the *dqd* transform is well defined and may be used as a default after a failure ($d_i \leq 0$) in *dqds*. Thus it is essential to have code that can execute a dqd step without overflow or unnecessary underflow. By scaling up the initial Z as much as possible the occurrence of underflow is minimized.

Our policy is perhaps too cautious. We keep variables e_{min} and q_{max} up to date. As shown at the beginning of the section $temp \leq q_{max}/e_{min}$ for any shift $\tau \leq \lambda_{min}$. Our strategy is:

if $(safemin * q_{max} \leq e_{min})$ **then**

use dqds (1÷)

else

use dqd (safe) (i.e. $\tau = 0$)

end if

It is not essential to force $\tau = 0$ in the safe version of *dqds* but we chose to do it.

1.2 Underflow

The emphasis so far has been on overflow. However underflow, marked by flushing to zero, also undermines the high accuracy property of the algorithm. If the true value of a variable is too small to be represented then there is nothing to be done. On the other hand we can have expressions of the form

$a(b/c)$ which will underflow as written but can return a correct value when rewritten as $(a/c)b$. Neither the $(1\div)$ nor the $(2\div)$ version is safe from these "unnecessary" underflows but we can modify the test in the safe *dqd* given above so that such underflows do not occur. The new test is

$$\textbf{else if } (safemin * q(i+1) \leq \hat{q}(i) \ \text{ and } \ safemin * \hat{q}(i) \leq q(i+1)) \ \textbf{ then}$$

When should safe *dqd* be invoked? In contrast to overflow we do not have an easily computed *lower* bound on $\hat{e}(i)$ nor $\hat{q}(i)$ so we test after each *dqds* transform. If $emin = 0$ or $dmin = 0$ we assume that the underflow was not necessary, we disregard $\hat{Z}$, and invoke safe *dqd* on Z. Such caution degrades performance slightly on difficult cases but on the LAPACK test matrices of type 16 (wild exponent ranges) our code, in single precision, did compute correctly some tiny eigenvalues that had previously been recorded as 0. The same phenomenon in double precision is shown in Section 15.

2 The Prototype dqds Algorithm

The final procedure for computing the eigenvalues of a tridiagonal matrix with the aid of the dqds algorithm is made complicated by five features:

> splitting, flipping, ping-pong, an aggressive shift strategy,
> and over/underflow.

These features receive due attention above or below. For the moment let us ignore them and see how simple the resulting program can be. As each eigenvalue is accepted the qd-array Z discards the last q and the last e. One while loop gives the whole procedure.

```
while Z unfinished do
   examine Z's final entries,
      if negligible then reduce Z accordingly end if
   if Z unfinished then
      choose a shift (less than λ_min)
      apply the dqds transform to Z
   end if
```

end while

The body of the while loop given above constitutes what we will call below 'a good step'. It has three vital parts:

1. Deflate any converged eigenvalues
2. Choose a shift
3. Invoke dqds with that shift

The whole procedure may be put in one line,

while Z unfinished **do** take a good step **end while**

The complications in the actual program are of two kinds. The low level ones are those hidden in the procedure Goodstep. The high level ones force us to embed our while loop inside another one.

These high level troubles are not obvious. In order to guarantee high relative accuracy the code accepts the limitation of computing the eigenvalues in monotone increasing order. Thus the code is constrained to bring the smallest eigenvalue to the end of Z. However this cannot be done if one or more of the e-values in Z vanishes: No information can cross over a zero e_j. Consider, as an extreme case, the array $Z = (1, 0, 2, 0, 3)$. This corresponds to the diagonal matrix $(1, 2, 3)$. When the 0's are replaced by tiny positive numbers then the simple code described above will waste much time slowly converting $(1, 2, 3)$ into $(3, 2, 1)$. This phenomenon was called 'disorder of the latent roots' by Rutishauser. If an e-value is negligible then we say that Z 'splits' into two independent qd-arrays Z_1 and Z_2 that may be processed independently. A split enhances efficiency but complicates the program. Even to check for any negligible e's seems to require a pass through the Z-array and this will degrade performance. Details are discussed in Section 3.

The program looks for the smallest eigenvalue to appear at the end of the qd-array. If, at some stage, the small entries in Z are at the beginning then it is prudent to simply reverse the array. This is called a flip.

$$\text{Flip}(q_1, e_1, q_2, e_2, q_3) = (q_3, e_2, q_2, e_1, q_1).$$

Flipping is equivalent to reversing the associated tridiagonal matrix, an operation that preserves the eigenvalues.

Splitting and Flipping are high level complications. The ping-pong formulation and aggressive shifting are low level features that are discussed later.

A switch to allow the user to select either relative accuracy or absolute accuracy (error $< \epsilon\|T\|$) also complicates the program and after extensive tests we have simply disabled this option because the reduction in total time using absolute accuracy was only 10% or 15%.

3 Splitting

3.1 Monotonicity Properties

In order to justify the criteria for neglecting an e_i certain properties of the transforms are needed. These elementary results have not appeared elsewhere so we present them here. This subsection may be skipped without loss of continuity.

At any step in the algorithm we possess L, U, and $\sigma \geq 0$ but might wish we had $\bar{L}$ and $\bar{U}$ satisfying $\bar{L}\bar{U} = LU + \sigma I$. So Lemma 1 can be useful.

Lemma 1 *Let L and U be the bidiagonals associated with the positive qd-array Z and let $\bar{Z}$ be the qd-array associated with $\bar{L}$ and $\bar{U}$ defined by*

$$\bar{L}\bar{U} = \sigma I + LU, \quad \sigma > 0. \tag{1}$$

Then $\bar{e}_k < e_k$ and $\bar{q}_{k+1} > q_{k+1} + \sigma$ for $k = 1, \ldots, n-1$.

Proof. By equating corresponding entries in (1)

$$\begin{aligned}
\bar{q}_1 &= q_1 + \sigma, \\
\bar{e}_k &= \frac{e_k q_k}{\hat{q}_k}, \\
\bar{q}_{k+1} &= q_{k+1} + (e_k - \bar{e}_k) + \sigma, \quad k = 1, \ldots, n-1.
\end{aligned}$$

The relation $\bar{q}_1 \geq q_1 + \sigma$ is the base for an inductive argument: $\bar{q}_k \geq q_k + \sigma$ implies that $\bar{e}_k = (q_k/\bar{q}_k)e_k < e_k$ and hence $\bar{q}_{k+1} > q_{k+1} + \sigma$. $\square$

Even more useful than $\bar{L}$ and $\bar{U}$ would be the quantities $\bar{d}_i$ that occur in the dqd transform of $\bar{Z}$. Lemma 2 assures us that $\bar{d}_i > \sigma$.

Lemma 2 *As in Lemma 1 let $\bar{L}\bar{U} = LU + \sigma I$. Let $\{\bar{d}_i\}$ be the auxiliary quantities that appear in the dqd transform of $\bar{Z} = \{\bar{q}_1, \bar{e}_1, \ldots, \bar{e}_{n-1}, \bar{q}_n\}$. Then*

$$d_i > \sigma, \qquad i = 1, 2, \ldots, n.$$

Proof. In [3] it was shown that

$$\bar{d}_j = \frac{1}{[(\bar{L}\bar{U})^{-1}]_{jj}}.$$

Since $B_{jj} \leq \lambda_{max}(B)$ for any matrix B that is diagonally similar to a positive definite matrix and since $\bar{L}\bar{U}$ (as well as its inverse) is such a matrix,

$$d_j \geq \frac{1}{\lambda_{max}[(\bar{L}\bar{U})^{-1}]} = \lambda_{min}(\bar{L}\bar{U}), \qquad j = 1, 2, \ldots, n.$$

Thus

$$d_j \geq \lambda_{min}(LU + \sigma I) = \sigma + \lambda_{min}(LU) > \sigma$$

with strict inequality since $Z = \{q_1, e_1, \ldots, e_{n-1}, q_n\}$ is assumed positive in Lemma 1. $\square$

We will often be in possession of the auxiliary d's after invoking dqds with a shift $\tau > 0$ on Z to produce another positive qd-array $\hat{Z}$. For testing e_j we would prefer to have auxiliary $\overset{o}{d}$'s that come from dqd ($\tau = 0$) applied to Z. Fortunately $0 < d_i < \overset{o}{d}_i, \;\; i = 1, \ldots, n$.

Lemma 3 *Consider a successful dqds transform with shift τ, $0 < \tau < \lambda_{min}$ that maps Z into $\tilde{Z}$. Let $\{d_i = d_i(\tau)\}_1^n$ be the associated d's but write $\overset{o}{d}_i$ for $d_i(0)$. Then*

$$0 < d_i < \overset{o}{d}_i, \quad i = 1, \ldots, n.$$

Proof. From the dqds transform in Section 1

$$d_{i+1} = \frac{d_i}{d_i + e_i} q_{i+1} - \tau$$

whereas

$$\overset{o}{d}_{i+1} = \frac{\overset{o}{d}_i}{\overset{o}{d}_i + e_i} q_{i+1}$$

for $i = 1, 2, \ldots, n-1$. Initially $d_1 = q_1 - \tau < q_1 = \overset{o}{d}_1$. Since $x/(x+e)$ is monotone increasing in x for $x > 0$, then if $d_i < \overset{o}{d}_i$ it follows that

$$d_{i+1} < \frac{\overset{o}{d}_i}{\overset{o}{d}_i + e_i} q_{i+1} - \tau < \frac{\overset{o}{d}_i}{\overset{o}{d}_i + e_i} q_{i+1} = \overset{o}{d}_{i+1} .$$

By induction the claim holds for $i = 1, \ldots, n$. □

3.2 Results of Demmel/Kahan and Li

For this section we revert to the bidiagonal case. Thus

$$B = \text{bidiag}\begin{pmatrix} & b_1 & & b_2 & & \cdot & & b_{n-2} & & b_{n-1} & \\ a_1 & & a_2 & & \cdot & & \cdot & & a_{n-1} & & a_n \end{pmatrix}$$

where $a_i = \sqrt{q_i}$, $b_i = \sqrt{e_i}$. The goal is to find conditions on an off-diagonal entry b_k that permit it to be set to 0 without causing a large relative error in any singular value of B. In [2] several lemmas and theorems were proved to justify a criterion based on a couple of recurrences. These recurrences are quite expensive. Later, in [6], Reng-Cang Li found alternative recurrences that gave sharper (better) criteria for neglecting b_k. All the results mentioned above are impressive, not to say intimidating. Indeed Li's paper [6] does not present his justification explicitly but only as a Corollary of theorems in other papers [5, Th1]. Those theorems require significant preparation on the part of the reader.

We are not going to reproduce all that theory. Instead we explain, in much simpler terms, why these criteria are appropriate and natural.

When b_k, $1 < k < n-1$, is set to 0 the resulting matrix is a direct sum of $\text{diag}(B_1, B_2)$. The surprise is that B may be related to $\text{diag}(B_1, B_2)$ as a *multiplicative* perturbation:

$$B = D_l \text{diag}(B_1, B_2), \quad B = \text{diag}\,(B_1, B_2) D_r, \tag{2}$$

where D_l and D_r have the form

$$\begin{pmatrix} I & F \\ O & I \end{pmatrix}$$

An old result of Ostrowski (rediscovered by Eisenstat and Ipsen) says that the *relative* change in any singular value due to annihilating b_k is bounded by $\|D_l D_l^t - I\|$ and by $\|D_r^t D_r - I\|$ and $\|\cdot\|$ is the spectral norm. These bounds equal $\|F\| + \|F\|^2$ and when F is tiny they are essentially $\|F\|$.

The old result comes from Weyl's theorem that says that no eigenvalue of a symmetric matrix can change by more than the (spectral) norm of an (additive) perturbation. So, if $\tilde{\sigma}$ is any singular value of $\overset{o}{B} := (B_1, B_2)$, then it is only necessary to rewrite (2) above in the illuminating form

$$BB^t - \tilde{\sigma}^2 I = D_l(\overset{o}{B}\overset{o}{B}{}^t - \tilde{\sigma}^2 I)D_l^t + \tilde{\sigma}^2(D_l D_l^t - I). \tag{3}$$

The first term on the right is singular and the second is an additive perturbation with norm $\tilde{\sigma}^2\|D_l^t D_l - I\|$. Hence there is a singular value σ of B satisfying

$$|\sigma^2 - \tilde{\sigma}^2| \leq \tilde{\sigma}^2\|D_l^t D_l - I\|.$$

So,

$$|\sigma - \tilde{\sigma}| \leq \frac{\tilde{\sigma}}{\tilde{\sigma} + \sigma}\,\tilde{\sigma}\|D_l^t D_l - I\| < \tilde{\sigma}\|D_l^t D_l - I\|. \tag{4}$$

Once the idea of using D_l and D_r is absorbed it is not hard to find out what F is in each case. Certainly it is a multiple of b_k. Not only is $\frac{1}{b_k}F$ rank-one but it has a single non-zero row or column, either the last column of B_1^{-1} or the first row of B_2^{-1}. Indeed the recurrences mentioned above generate the entries in these two vectors. However Demmel/Kahan generate the 1-norm whereas Li generates the 2-norm of these vectors. This brings us to the question of cost.

In the context of the singular value QR algorithm with zero shift applied to B both criteria require $2n$ divisions to test all b_k but the 1-norm requires fewer multiplications. The miracle is that in the context of the dqd algorithm the auxiliary quantity d_k is precisely $\|\frac{1}{b_k}F\|^{-2}$ for one of Li's tests (row 1 of B_2^{-1}). Thus $\|F\| \leq \varepsilon$ becomes Li's test

$$e_k := b_k^2 \leq \varepsilon^2 d_k. \tag{5}$$

The other test (involving column 1 of B_1^{-1}) would require running dqd on the reversed, or flipped qd array to produce auxiliary quantities $\overset{o}{d}_i$, $i = n, n-1, \ldots, 1$. Then one could test $e_k \leq \varepsilon^2\, \overset{o}{d}_{k+1}$ but that does not come free.

As Li remarks at the end of [6] the introduction of nonrestoring shifts into dqds complicates the situation significantly. That is the focus of Section 3.4.

3.3 In the Beginning

It is worthwhile to apply both of Li's tests at the start of the algorithm. The variables e_{min} and q_{max} are formed when the data are checked. If $(e_{min} > \varepsilon^2 q_{max})$ then there will be no splits and the standard dqds subroutine may be employed in the interest of efficiency.

The following example shows that there may be no splits on the original data and yet after one dqd transform the new array may have all its e's negligible. This encourages us to apply Li's tests for at least two iterations.

Example 2 (From No Splits to All Splits) Consider a Toeplitz qd-array of order 10 with $q_i = \varepsilon$, all i, and $e_i = \varepsilon^{-1}$. Here ε is the single precision roundoff unit, $\varepsilon \approx 10^{-7}$. In single precision ε^6 underflows.

On the first dqd transform there are no splits. Moreover $\hat{q}_i = \varepsilon^{-1}$, $i = 1, \ldots, n-1$, $\hat{e}_i = \varepsilon$, $i = 1, \ldots, n-1$, but $\hat{q}_n = \varepsilon^{19} = \text{underflow} = 0$. On the second dqd transform the first $n-1$ d's compute to ε^{-1} and, by Li's test, each $\hat{e}_i$ is then set to 0. If we applied Li's reverse test to the array $\hat{Z}$ all d's would be 0 and no splits would be recorded.

pseudo code for initial checks for splits

input: $Z = (q, e)$, e_{min}, q_{max}
flip Z if warranted
if $(e_{min} \leq \varepsilon^2 q_{max})$ **then**
    apply Li's reverse test on Z;
    dqd: $Z \longrightarrow \hat{Z}$ with Li's test;
        (e_{min} is updated)
else
    $\tau \longleftarrow 0$
    dqds: $Z \longrightarrow \hat{Z}$
        (e_{min} is updated)
end if
update q_{max} $(= \max_i \hat{q}(i))$

if $(e_{min} \leq \varepsilon^2 q_{max})$ **then**

  apply Li's reverse test on $\hat{Z}$;

  dqd: $\hat{Z} \longrightarrow Z$ with Li's test;

    (e_{min} is updated)

else

  $\tau \longleftarrow 0$

  dqds: $\hat{Z} \longrightarrow Z$

    (e_{min} is updated)

end if

On completion the latest qd-array is in Z, the old array is in $\hat{Z}$ and all possible splitting using both of Li's tests have been recorded.

It is possible to repeat this testing cycle until no new splits are recorded but we decided to run it just twice.

3.4 When to Neglect e_j

There are, at least, two obstacles to invoking Li's test inside the main while loop. First the algorithm uses *dqds* with $\tau > 0$ for most steps. Li's test is still valid when $\tau > 0$, by Lemma 3, but will be stricter than necessary. Second is the presence of σ, the accumulated shift. We want the eigenvalues of $\sigma I + LU$, not of LU alone. It is not cheap to incorporate σ into Li's criterion.

From a practical point of view the presence of σ has lead us to a very simple set of tests. Since σ is a lower bound on the eigenvalues of $\sigma I + LU$ the following test is always valid.

- σ-test: if $e_j \leq \varepsilon^2\sigma$ then set e_j to 0.

Li was thinking of an implementation of dqd that would test e_j within the inner loop and thus at every step of the algorithm. In order to keep our dqds transform free of tests the checking for a split will be a separate calculation undertaken only when the variable e_{min} is small enough. The code presented here keeps this test calculation as cheap as possible.

This leaves us with the task of using Li's test in some form because the σ-test is useless when $\sigma = 0$ and even when σ is tiny. Our solution is to exploit the ping-pong implementation of the algorithm. For our purposes here it means that two qd-arrays are available, *Old* Z and Z. Their eigenvalues differ by τ. Let $\overset{o}{d}_i$ denote the auxiliary variables computed in the dqds transform of $Old\,Z$ to Z. From the algorithm in Section 1

$$q_i = \overset{o}{d}_i + old\,e_i, \quad i = 1, 2, \ldots, n-1.$$

Consequently $\overset{o}{d}_i$ can be recovered as $q_i - old\,e_i$. Li's test applied to $Old\,Z$ is 'neglect $old\,e_i$ if $old\,e_i \leq \varepsilon^2 d_i = \varepsilon^2(q_i - old\,e_i)$'. Since $1+\varepsilon^2$ computes to 1 the test may be simplified.

- Li's test: neglect $old\,e_k$ if $old\,e_k \leq \varepsilon^2 q_k$.

Note that setting $old\,e_k$ to 0 would automatically force e_k to 0 since

$$e_k = old\,e_k \cdot \frac{old\,q_{k+1}}{q_k}.$$

Moreover, in 'finite precision', if $old\,e_k$ is negligible then

$$\begin{aligned}\overset{o}{d}_{k+1} &= old\,q_{k+1}\frac{\overset{o}{d}_k}{\overset{o}{d}_k + old\,e_k} - \tau \\ &= old\,q_{k+1} - \tau\end{aligned}$$

just as though $old\,e_k$ were 0.

Thus, *at no cost*, we can discover a split but with a one step delay.

It is not *necessary* to recognize splits as soon as they are warranted. The only danger in delaying a valid split is that the smallest eigenvalue might be trapped in the upper part of the qd-array. This could produce the wretched situation that the shifts would be constrained by the top part and so not hasten convergence of the bottom part. This would degrade efficiency severely.

When should our two tests be invoked? Since our implementation keeps the variables e_{min} and q_{max} up to date it is natural to invoke the testing loop only when

$$old\,e_{min} \leq \varepsilon^2 q_{max} \text{ or } e_{min} \leq \varepsilon^2\sigma. \tag{6}$$

This guarantees that if a split is warranted by our tests then the loop will be invoked.

The way splits are marked is discussed next.

3.5 Marking Splits

The Z-array may split up into many subarrays. In order to keep the code simple the dqds transform is applied only to the last unsplit segment $i0 : n0$. The parameter $n0$ never increases and decreases when, and only when, an eigenvalue is deflated. Until a segment is finished $i0$ never decreases and increases when, and only when, a split occurs in $i0 : n0$.

Suppose that the first split occurs at e_j when the value of σ is σ'. The segment $1 : j$ of Z then freezes until the segment $j + 1 : n$ is finished. By that stage $\sigma = \sigma'' \geq \sigma'$. When computation resumes on segment $1 : j$ it is essential to know the old value σ'. The only book-keeping required when a split occurs is to record the current value σ. The *natural* place to keep this information is in the location of the negligible e_j. The negative sign attached to σ signals that a split has occurred.

The pseudo-code for the segment Spltck (short for Split Check) finds the index '*splt*' where the last negligible e-value occurs.

Spltck: $\quad splt \leftarrow i0 - 1$
for $k \leftarrow i0, n0 - 3$ **do**
  if e_k negligible **then**
    $e_k \quad \leftarrow -\sigma$
    $splt \leftarrow k$
  end if
end for

By construction of $i0$, either $i0 = 1$ or else $e_{(i0-1)} < 0$. Several e's may be found negligible during one call of Spltck, each one is marked (by $-\sigma$) but only the last one is recorded by *splt*. Thus after each call to Spltck the new segment is given by

$$i0 \leftarrow splt + 1.$$

The loop stops at $k = n0 - 3$ because e_{n-2} and e_{n-1} are checked at every step. When they become negligible we have deflation, not a split.

4 The High Level Program

When splitting is incorporated into the program there must be an inner loop to diagonalize the last unsplit segment and an outer loop over the separate segments. This structure demands one extra piece of book-keeping in the outer loop. The choice of shift makes heavy use of information obtained in the *previous* dqds transform. At the start of a new unsplit segment there is no previous dqds transform available. Inside the inner while loop there is no way to know whether the current segment is new and so the outer loop must set a flag to signal this situation. We may do this by setting the variable $dmin$ to a negative value. The current segment is always $Z(i0 : n0)$.

> Input $Z(1 : n)$, a positive qd-array (but $e(n) = 0$).
> call Prologue *(discussed in Section 10)*
> $n0 = n$
> **while** $(n0 \geq 1)$ **do**
> $\sigma = -e(n0)$ * reset σ *
> $i0 = n0$ * seek $i0$ *
>   **while** $(i0 > 1$ and $e(i0 - 1) > 0)$ **do** $i0 = i0 - 1$ **end while**
> $dmin = -0$ * signal a new segment *
>   **while** $(i0 \leq n0)$ **do**
>   call Goodstep$(i0, n0, Z, \sigma, dmin)$
>   **if** $emin$ is small enough **then**
>   check for splits; update $i0$, $emin$, $qmax$;
>   **end if**
>   **end while**
> **end while**
> call Epilogue *(discussed in Section 11)*

Later we will complicate the while loop that finds $i0$ so that it computes q_{min} and e_{max} as well. These values give us a cheap lower bound on the Gersgorin disks. We set $dmin = -\max(0, q_{min} - 2\sqrt{q_{min}e_{max}})$ and give justification in Section 6.3.1 but here is the motivation.

When Z's matrix is close to one of low rank a stage will occur when all the small eigenvalues have been found and the smallest eigenvalue of the remaining Z array is far from 0. Our shift strategy shifts too cautiously in this situation and $\tau = q_{min} - 2\sqrt{q_{min}e_{max}}$ is a much better start than $\tau = 0$.

For example, in one case all e's were $O(10^{-15})$ and all q's were $O(10^{-1})$. Thus at the start of a new unsplit segment the variable $dmin$ carries a reasonable shift that overrides the regular choices because it is flagged by not being positive.

5 Low Level Complications

Pseudocode for Goodstep($i0, n0, Z, \sigma, dmin$):

1. **while** $(e(n0-1)$ or $e(n0-2))$ negligible **do**
   record eigenvalues
   reduce $n0$
   end while
   if $(n0 < i0)$ return **end if**

2. **if** warranted **then**
   flip qd array
   update $qmax$, $emin$
   end if

3. **if** no danger of overflow or new segment **then**
   choose a shift
4. **repeat**
   call dqds; output $dmin, emin$
   if (shift too big or d_{min}=NaN or underflow) **then**
   if ($d_{min} < 0$) **then**
   choose another shift
   else (a NaN or underflow)
   call safe dqd; output $dmin, emin$
   end if
   end if
   until $dmin > 0$
   else
   call safe dqd; output $dmin, emin$
   end if

5. **update** σ

As written above Step 4 could give rise to an infinite loop. For the sake of efficiency we want to escape this loop after 3 steps at most. The choice $\tau = 0$ ensures a successful transform but the phenomenon of 'late failure', discussed later, exhorts us not to panic and so set $\tau = 0$ immediately. Frequently a failed shift is too large only in the 5th decimal place of λ_{min}.

The various parts of Goodstep are discussed in turn below.

6 A Good Step

6.1 Test for Eigenvalues (Eigtest)

Convergence Versus Deflation

In this section let $n = n0$. The goal of the dqds transform is to drive q_n, the last q, to zero. Even if $q_n = 0$ it is still not valid to deflate, i. e. to reduce n by 1, because e_{n-1} must also be negligible. Note that, with $\sigma = 0$,

$$\hat{e}_{n-1} = e_{n-1} q_n / \hat{q}_{n-1} = q_n \cdot \frac{e_{n-1}}{d_{n-1} + e_{n-1}} < q_n,$$

so that one more transform, after q_n is negligible, will ensure that the new e_{n-1} will also be negligible.

However we shall not retain this way of thinking because convergence (is q_n close enough to 0?) is secondary to deflation ($n \leftarrow n-1$ or $n-2$) and that is what we seek. If $e_{n-1} = 0$ then $q_n + \sigma$ is an eigenvalue however large q_n may be.

Accepting Eigenvalues

We check e_{n-2} as well as e_{n-1} because there is a short section of code that computes the eigenvalues in the 2×2 case to high relative accuracy. See Section 8.

From Section 3.4 e_{n-1} is negligible if $old\, e_{n-1} \le \varepsilon^2 q_{n-1}$ or $e_{n-1} \le \varepsilon^2\sigma$. Similarly e_{n-2} is negligible if $old\, e_{n-2} \le \varepsilon^2 q_{n-2}$ or $e_{n-2} \le \varepsilon^2\sigma$. By using Li's reverse test we may neglect e_{n-1} if $e_{n-1} \le \varepsilon^2 q_n$ and e_{n-2} if $e_{n-2} \le \varepsilon^2 q_{n-1}(q_n/(q_n+e_{n-1}))$. This is because the dqd algorithm on the flipped array yields $d_n = q_n$ and $d_{n-1} = q_{n-1}q_n/(q_n+e_{n-1})$. Since $a+b \le 2\max(a,b)$ we have invoked the following simple tests (perhaps these tests are too severe):

$$\text{if } old\, e_{n-1} \le \varepsilon^2 q_{n-1} \quad \text{or} \quad e_{n-1} \le \varepsilon^2(\sigma + q_n) \text{ then neglect } e_{n-1}$$

$$\text{if } old\, e_{n-2} \le \varepsilon^2 q_{n-2} \quad \text{or} \quad e_{n-2} \le \varepsilon^2\left(\sigma + q_{n-1}\frac{q_n}{q_n+e_{n-1}}\right) \text{ then neglect } e_{n-2}.$$

Note that the second test is only invoked when e_{n-1} is not negligible, so the division is proper. We have softened the test on the *old* values by multiplying ε^2 by 10^4. This was the largest value that caused no deterioration in accuracy on our LAPACK test bed of matrices.

When e_{n-2} is negligible the simple deflating code is

$$\begin{aligned} big &= \text{larger root of trailing } 2\times2 \text{ submatrix (Section 8)} \\ q_n &= q_n q_{n-1}/big + \sigma \\ q_{n-1} &= big + \sigma \\ n &= n-2 \end{aligned}$$

These simple codes become more complicated in the ping-pong implementation discussed in Section 9. The code that tests e_{n-1} and e_{n-2} is in a repeat loop so that control passes out of this segment only when either $n0 < i0$ or else neither e_{n-1} nor e_{n-2} is negligible. Goodstep is complete if $n0 < i0$.

6.2 Check for Flipping

The goal of the algorithm is to drive $q(n0)$ to 0. If $q(i0) < q(n0)$ then it seems plausible that convergence would be faster if the array were flipped. In principle one could make more elaborate schemes for checking whether the smallest eigenvalue is 'located' near the top of the matrix. To introduce a bias against flipping we demand that

$$1.5q(i0) < q(n0).$$

A rival test would demand that $2q(i0)e(i0) < q(n0)e(n0-1)$ before flipping but so far we have used the simpler test.

We make the check only after an eigenvalue has been deflated at the previous step (signaled by $n0in > n0$) or when the segment is 'new', i. e. has just been passed from the outer while loop. After flipping we set $dmin$ to -0 so that the flipped array is treated as 'new'. Thus

if $(dmin < 0$ or $n0in > n0)$ **then**
  if$(1.5q(i0) < q(n0))$ **then**
    call Flip
    if $(n0in.gt.n0)$ $dmin = -0$ **end if**
    update e_{min} and q_{max}
  end if
end if

6.3 Choice of Shift

At an abstract level both qds (Rutishauser's qd with shifts) and dqds are equivalent to LR and two LR steps are equivalent to one QR step. So one might expect convergence rates to be similar. The advantage of dqds over the other transforms is the auxiliary variable d and the fact that $dmin$ is an increasingly good approximation to λ_{min}. Section 6.3.2. The index of $dmin$ (i. e. the index j such that $d_j = dmin$) can also be useful in 'locating' λ_{min} before it migrates to the end of the array.

The chief feature of the implementation given here is the decision to dispense with $dmin$'s index. To make up for this omission we unroll the last two steps of dqds and record d_n, d_{n-1}, d_{n-2} as well as $dmin$, $dmin1$, $dmin2$, where $dmin1 = \min_{i \leq n-1} d_i$ and $dmin2 = \min_{j \leq n-2} d_j$. These six values give the index of $dmin$ in the asymptotic regime when $dmin = d_n$, or d_{n-1}, or d_{n-2}.

It could be the case that the use of $dmin$'s index can be made cost effective, but that is for the future.

Our shift strategy is essentially one long if-statement giving a different value to the shift τ for each of about 10 different situations. Each formula uses information from the previous dqds-transform, in particular the last 3 values of the auxiliary variable d.

At the start of processing a new segment of Z there is no previous transformation. This situation is signaled by $dmin \leq 0$. In early versions of this program we used the obvious choice

$$\tau = 0$$

when $dmin < 0$, because we seek the smallest eigenvalue. Now we use the Gersgorin shift when it positive.

6.3.1 The Gersgorin Shift

If the minimum point among all Gersgorin disks is positive then it serves as a better shift than 0. A straightforward computation of this point $\min_i(q_i + e_i - \sqrt{q_i e_{i-1}} - \sqrt{q_{i+1} e_i})$ costs more than a dqds transform because of all the square roots. Our shift strategy ensures that most of the time the minimum Gersgorin point is negative. It is only when the e's are much smaller than the q's that it is appropriate to consider Gersgorin. We use a crude lower bound $q_{min} - 2\sqrt{q_{min} e_{max}}$ because q_{min} and e_{max} are cheap to compute in the loop that finds $i0$ at the start of a new segment. Moreover we only update q_{min} and e_{max} while $q_{min} \geq 4e_{max}$ so that, in most cases, this calculation stops almost immediately. In special cases (all q's > 0.01, all e's $< 10^{-9}$) this feature is most valuable.

6.3.2 Use of $dmin$

At each call of Goodstep there are four situations at each choice of shift: at Step 1 Eigtest found either 0, 1, 2, or more than 2 eigenvalues.
The first situation is the basic one and the others are variations on the first. Before describing the selection we recall some results on eigenvalue bounds. See [8, Sections 4.5 and 11.7].

Let $\|\boldsymbol{x}\| = 1$, $\rho = \rho(\boldsymbol{x}) = \boldsymbol{x}^* A \boldsymbol{x}$, $\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{x}\rho$ for any symmetric matrix A. Let λ be the closest eigenvalue of A to ρ and let gap be the distance of ρ from the rest of A's spectrum. Then

$$\lambda \;>\; \rho - \|\boldsymbol{r}\|, \tag{7}$$

$$\lambda \quad > \quad \rho - \|\boldsymbol{r}\|^2/gap. \tag{8}$$

Our main application of this result is to a tridiagonal T with $\boldsymbol{x} = (0, \ldots, 0, 1)^*$. In that case

$$\rho = \alpha_n, \qquad \|\boldsymbol{r}\| = \beta_{n-1}.$$

In our application $\beta_{n-1}^2 = q_n e_{n-1}$ and $T =$ symmetrized UL, as shown at the end of this subsection.

We also recall some results on the intermediate quantities d_j computed by the dqds transform with shift τ. See [3].

If $\tau = 0$ then

$$\frac{1}{d_j} = [(UL)^{-1}]_{jj} < \frac{1}{\lambda_{min}(UL)}. \tag{9}$$

If $\tau > 0$ then

$$\begin{aligned} \lambda_j(\hat{U}\hat{L}) \quad &= \quad \lambda_j(UL) - \tau, \\ d_{min} \quad &\geq \quad \lambda_{min}(\hat{U}\hat{L}), \\ \frac{1}{d_j} \quad &\geq \quad [(\hat{U}\hat{L})^{-1}]_{jj}. \end{aligned}$$

As τ increases from 0 to $\lambda_{min}(UL)$ so does $dmin$ decrease

$$\text{from } 1/\max_j [(UL)^{-1}]_{jj} \text{ to } 0.$$

Thus the smaller the value of $dmin$ the better it approximates $\lambda_{min}(\hat{U}\hat{L})$ with equality when, and only when, $dmin = 0$. However $dmin$ is always too big and we would prefer to have a lower bound. Our program is set up to reject, as a failure, any dqds transform in which $dmin$ is not positive. The penalty for choosing τ too large is a wasted dqds transform, except in the case of late failure discussed below, and in view of all this we use a fairly aggressive choice of shift and hope to keep failures at the 2 or 3% level.

In order to keep the dqds transform as simple as possible we record $dmin = \min_{1 \leq j \leq n} d_j$ but not its location. To make up for this loss we 'unroll' the last two steps of the dqds transform and this yields, at no cost, 6 useful d-values: d_n, d_{n-1}, d_{n-2}, and $dmin$, $dmin1$, $dmin2$. Here

$$\begin{aligned} dmin1 \quad &= \quad \min_{1 \leq j \leq n-1} d_j, \\ dmin2 \quad &= \quad \min_{1 \leq j \leq n-2} d_j. \end{aligned}$$

Our shift formulae make heavy use of these 6 values.

It may turn out that giving up the precise location of $dmin$, when it is less than $n-2$, is a tactical error. More study is needed.

One more rather subtle point must be borne in mind. Let M_{n-1} denote the leading principal $(n-1)\times(n-1)$ submatrix of M. Let T_n denote the symmetrized version of $UL = U_n L_n$. Then

$$T_{n-1} \neq \mathrm{sym}\,(U_{n-1}L_{n-1}).$$

The matrices differ only in the last diagonal entries which are respectively $q_{n-1}+e_{n-1}$ and q_{n-1}. Now $dmin1$ approximates $\lambda_{min}(U_{n-1}L_{n-1})$ while we want to approximate $\lambda_{min}(T_{n-1})$. When $dmin1 = d_{n-1}$ then we expect the associated eigenvector of $U_{n-1}L_{n-1}$ to be dominated by its last entry. So we sometimes use some fraction φ of $dmin1 + \frac{1}{2}e_{n-1}$ as an approximation to $\lambda_{min}(T_{n-1})$. The choice of φ has been a worry. We use $\varphi = 0.75$ but have no theory to back it up.

Now we turn to our shift selection. It is a long if-then-else statement.

In order to simplify expressions (for humans) we use

$$\begin{aligned} \alpha_n &= q_n = d_n, & \beta_{n-1} &= \sqrt{q_n e_{n-1}}, \\ \alpha_{n-1} &= q_{n-1}+e_{n-1}, & \beta_{n-2} &= \sqrt{q_{n-1}e_{n-2}} \\ \beta_{n-3} &= \sqrt{q_{n-2}e_{n-3}}. \end{aligned}$$

By taking $\boldsymbol{e}_{n-1}$ as an approximate eigenvector of T_{n-1} and using its residual norm we conclude that some eigenvalue exceeds $\alpha_{n-1} - \sqrt{\beta_{n-1}^2 + \beta_{n-2}^2}$. This is easier for us than the Gersgorin value $\alpha_{n-1} - \beta_{n-1} - \beta_{n-2}$.

In the actual code n is replaced by $n0$.

6.3.3 No Eigenvalues Found in Eigtest ($n0in = n0$)

The variable $n0in$ is the value of $n0$ on entry to Eigtest.

Case 1. If $dmin \le 0$ then $\tau = -dmin$.
This is the case corresponding to a new qd-segment. No old information available. See Section 6.3.1.

Cases 2 and 3. $dmin = d_n$ and $dmin1 = d_{n-1}$.

This is the asymptotic case that determines the rate of convergence (a misleading term when we strive for between 3 and 4 iterations per eigenvalue, on average). Our goal is to use (8) in the tests given in Section 6.3.2 and so we must appproximate $\lambda_{min}(T_{n-1}) - \alpha_n$ by a value $gap1$. To do this we guess at $gap2$ to approximate $\lambda_{min}(T_{n-2}) - \alpha_{n-1}$.

$$gap2 = \frac{3}{4} dmin2 - \alpha_{n-1}.$$

Now we estimate $gap1$ by

if $(gap2 > 0)$ and $gap2^2 > \beta_{n-2}^2$ **then**

$$gap1 = \alpha_{n-1} - \frac{\beta_{n-2}^2}{gap2} - d_n$$

else

$$gap1 = \alpha_{n-1} - \sqrt{\beta_{n-1}^2 + \beta_{n-2}^2} - d_n,$$

end if

Finally

if $(gap1 > 0$ and $gap1^2 > \beta_{n-1}^2)$ **then**

$$\tau = \max(d_n - \frac{\beta_{n-1}^2}{gap1}, \frac{1}{2} d_n) \qquad \text{(Case 2)}$$

else (Gersgorin)

$$\begin{cases} x_1 = \max\{0,\ d_n - \beta_{n-1}\}, & \text{row } n, \\ x_2 = \max\{0,\ \alpha_{n-1} - \sqrt{\beta_{n-1}^2 + \beta_{n-2}^2}\}, & \text{row } n-1, \\ \tau = \max\{\frac{1}{3} d_n,\ \min\{x_1,\ x_2\}\} & \text{(Case 3)} \end{cases}$$

end if

Note that $\tau \geq d_n/2$ (Case 2) or $\tau \geq d_n/3$ (Case 3). Here lies the aggression in our shift strategy.

We expect Case 2 to occur often. The formulae are simpler than those for Cases 4 and 5 and give good accuracy. Nevertheless it is possible that the approximations used for Case 4 would be even better when used in Case 2. More study is needed.

Case 4. Not quite asymptotic.

(a) $dmin = d_n$ but $dmin1 \neq dn1$.

(b) $dmin \neq dn$ but $dmin1 = dn1$.

Warning
The long analysis that follows for (a) and (b) uses new refined bounds and leads to only 20 lines of code and approximately 4 divisions. It may be skipped without loss of continuity. Let

$$L = \text{bidiag}\begin{pmatrix} \sqrt{q_1} & & \sqrt{q_2} & & \sqrt{q_3} & & \cdot & & \sqrt{q_{n-1}} & & \sqrt{q_n} \\ & \sqrt{e_1} & & \sqrt{e_2} & & \cdot & & \cdot & & \sqrt{e_{n-1}} & \end{pmatrix}.$$

For (a) consider one step of inverse iteration starting with $\boldsymbol{e}_n d_n$ and yielding $\boldsymbol{z}$.

$$(L^t L - \tau I)\boldsymbol{z} = \hat{L}\hat{L}^t \boldsymbol{z} = \boldsymbol{e}_n d_n = \boldsymbol{e}_n \hat{q}_n.$$

Since $\hat{L}^{-1}\boldsymbol{e}_n = \boldsymbol{e}_n/\sqrt{\hat{q}_n}$,

$$\begin{aligned} \hat{L}^t \boldsymbol{z} &= \frac{\boldsymbol{e}_n}{\sqrt{\hat{q}_n}}, \\ z(n) &= 1, \\ z(i) &= -z(i-1)\sqrt{\hat{e}_i/\hat{q}_i}, \qquad i < n. \end{aligned}$$

As shown in Section 7, the Rayleigh quotient $\rho(\boldsymbol{z})$ satisfies

$$\rho(\boldsymbol{z}) = \frac{dmin}{\|\boldsymbol{z}\|^2},$$

and

$$\frac{\|\hat{L}\hat{L}^t \boldsymbol{z} - \boldsymbol{z}\rho(\boldsymbol{z})\|}{\|\boldsymbol{z}\|} = \rho(\boldsymbol{z})\sqrt{\|\boldsymbol{z}\|^2 - 1}.$$

Thus there is an eigenvalue λ of $\hat{L}\hat{L}^t$ satisfying

$$\rho(\boldsymbol{z})(1 - \sqrt{\|\boldsymbol{z}\|^2 - 1}) \leq \lambda \leq \rho(\boldsymbol{z}). \tag{10}$$

and this function is a useful lower bound provided that $\|\boldsymbol{z}\|^2 \leq 2$. In the absence of a satisfactory estimate of

$$\text{gap}(\rho) = \min\{|\rho - \mu| : \mu \neq \lambda,\ \mu \text{ an eigenvalue}\}$$

we do not employ a refined bound.

How accurately should $\varphi = \|\boldsymbol{z}\|^2 - 1$ be estimated? The recurrence for φ is simple:

$$\begin{array}{ll} \text{initial condition:} & term = \hat{e}_{n-1}/\hat{q}_{n-1}; \quad \varphi = 0; \end{array}$$

$$\begin{aligned}
&\varphi = \varphi + term \\
&\textbf{for } i = n-2, 1, -1 \textbf{ do} \\
&\quad old = term \\
&\quad term = term * \hat{e}_i/\hat{q}_i \\
&\quad \varphi = \varphi + term \\
&\textbf{end for}
\end{aligned} \tag{11}$$

Our first consideration is to run the loop until two consecutive terms are less than 1% of the current φ; repeat until

$$100 \max(term, old) < \varphi.$$

If $\varphi \geq 1$ then the lower bound in (10) is negative and our effort is wasted. However, in the spirit of an aggressive strategy we wish to choose $\tau \geq dmin/4$ in these cases. Consequently when $\varphi \geq 9/16$ we will not employ (10). So we repeat the for loop until

$$100 \max(term, old) < \varphi \quad \text{or} \quad 9/16 < \varphi.$$

We then increase the computed φ by 5% to compensate for truncating the loop. Finally set $\gamma = dmin$ and

$$\begin{aligned}
&\textbf{if } (\varphi < 9/16) \textbf{ then} \\
&\quad shift = \gamma \frac{1 - \sqrt{\varphi}}{1 + \varphi} \\
&\textbf{else} \\
&\quad shift = \frac{\gamma}{4} \\
&\textbf{end if}
\end{aligned} \tag{12}$$

For (b) ($dmin \neq dn$ but $dmin = dn1$) create a twisted factorization $L^t L - \tau I = N N^t$ with twist at $n-1$. Here $N^t = \hat{L}^t$ except for the last two rows shown below. See [7]. N requires three new values: $\gamma_{n-1}, \overset{o}{e}_{n-1}, \overset{o}{q}_n$. The last three rows of the right twisted factor N^t are shown here.

$$\begin{bmatrix} \sqrt{\hat{q}_{n-2}} & \sqrt{\hat{e}_{n-2}} & 0 \\ 0 & \sqrt{\gamma_{n-1}} & 0 \\ 0 & \sqrt{\overset{o}{e}_{n-1}} & \sqrt{\overset{o}{q}_n} \end{bmatrix}$$

By equating entries in $L^tL - \tau I = \overset{o}{L}{}^t \overset{o}{L}$ we find

$$\overset{o}{q}_n = q_n - \tau; \quad \overset{o}{e}_{n-1} = q_n e_{n-1}/\overset{o}{q}_n; \quad s_{n-1} = -\tau(1 + e_{n-1}/\overset{o}{q}_n)$$

and, from the code given in Case 5 below,

$$\begin{aligned} \gamma_{n-1} &= d_{n-1} + s_{n-1} + \tau, \\ &= d_{n-1} + [s_n(e_{n-1}/\overset{o}{q}_n) - \tau] + \tau \\ &= d_{n-1} - \tau e_{n-1}/(q_n - \tau) < d_{n-1} = dmin. \end{aligned} \tag{13}$$

Our estimate of λ_{min} is based on one step of inverse iteration starting from $\boldsymbol{e}_{n-1}\gamma_{n-1}$ (in case (a) we started with $\boldsymbol{e}_n d_n$):

$$(L^tL - \tau I)\boldsymbol{z} = NN^t\boldsymbol{z} = \boldsymbol{e}_{n-1}\gamma_{n-1}.$$

Since $N\boldsymbol{e}_{n-1} = \boldsymbol{e}_{n-1}\sqrt{\gamma_{n-1}}$,

$$\begin{aligned} N^t\boldsymbol{z} &= \boldsymbol{e}_{n-1}\sqrt{\gamma_{n-1}}, \\ z(n-1) &= 1, \\ z(i) &= -z(i-1)\sqrt{\hat{e}_i/\hat{q}_i}, \quad i < n-1, \\ z(n) &= -\sqrt{\overset{o}{e}_{n-1} / \overset{o}{q}_n} \\ &= -\sqrt{q_n e_{n-1}}/|q_n - \tau|. \end{aligned}$$

In addition

$$\begin{aligned} \rho(\boldsymbol{z}) &= \frac{\gamma_{n-1}}{\|\boldsymbol{z}\|^2} \\ \frac{\|NN^t\boldsymbol{z} - \boldsymbol{z}\rho(\boldsymbol{z})\|}{\|\boldsymbol{z}\|} &= \rho(\boldsymbol{z})\sqrt{\|\boldsymbol{z}\|^2 - 1}. \end{aligned}$$

We can use the same loop (11) as in Case (a) but with different initial conditions, namely

$$\begin{aligned} term &= \hat{e}_{n-2}/\hat{q}_{n-2} \\ \varphi &= z(n)^2 = q_n e_{n-1}/(q_n - \tau)^2. \end{aligned}$$

Compute γ_{n-1} from (13), set $\gamma = \gamma_{n-1}$, and the same code (12) may be used as in Case (a) for shift.

Case 5. $dmin = d_{n-2}$.

This condition suggests the use of a twisted factorization of $L^tL - \tau I$ with twist at position $n-2$. The upper part of the factorization is given by $\hat{L}\hat{L}^t$ but we do not have the lower part. Write $L^tL - \tau I = \overset{o}{L}{}^t \overset{o}{L}$. The lower part of $\overset{o}{L}$ is

$$\begin{bmatrix} \sqrt{\overset{o}{e}_{n-3}} & \sqrt{\overset{o}{q}_{n-2}} & & \\ & \sqrt{\overset{o}{e}_{n-2}} & \sqrt{\overset{o}{q}_{n-1}} & \\ & & \sqrt{\overset{o}{e}_{n-1}} & \sqrt{\overset{o}{q}_{n}} \end{bmatrix}$$

and the differential stationary algorithm yields

$$\begin{aligned} s_n &= -\tau \\ \overset{o}{q}_n = &= q_n + s_n \\ \overset{o}{e}_{n-1} &= q_n(e_{n-1}/\overset{o}{q}_n) \\ s_{n-1} &= s_n(e_{n-1}/\overset{o}{q}_n) - \tau \\ \overset{o}{q}_{n-1} &= q_{n-1} + s_{n-1} \\ \overset{o}{e}_{n-2} &= q_{n-1}(e_{n-2}/\overset{o}{q}_{n-1}) \\ s_{n-2} &= s_{n-1}(e_{n-2}/\overset{o}{q}_{n-1}) - \tau. \end{aligned}$$

That is all that we need. The lower part of the twisted factor N^t is

$$\begin{bmatrix} \sqrt{\hat{q}_{n-3}} & \sqrt{\hat{e}_{n-3}} & & \\ & \sqrt{\gamma_{n-2}} & & \\ & \sqrt{\mathring{e}_{n-2}} & \sqrt{\mathring{q}_{n-1}} & \\ & & \sqrt{\mathring{e}_{n-1}} & \sqrt{\mathring{q}_{n}} \end{bmatrix}.$$

The quantity γ_{n-2} is given by

$$\begin{aligned}
\gamma_{n-2} &= \hat{q}_{n-2} + \mathring{q}_{n-2} - (q_{n-2} + e_{n-2} - \tau) \\
&= (d_{n-2} + e_{n-2}) + (q_{n-2} + s_{n-2}) - (q_{n-2} + e_{n-2} - \tau) \\
&= d_{n-2} + s_{n-2} + \tau \\
&= d_{n-2} + [s_{n-1}(e_{n-2}/\, \mathring{q}_{n-1}) - \tau] + \tau \\
&= d_{n-2} + s_{n-1}(e_{n-2}/\, \mathring{q}_{n-1}), \\
&= d_{n-2} + s_{n-1}[e_{n-2}/(\mathring{q}_{n-1} + s_{n-1})].
\end{aligned}$$

Write the twisted factorization as $L^t L - \tau I = N N^t$ and define $\boldsymbol{z}$ by

$$N N^t \boldsymbol{z} = \boldsymbol{e}_{n-2}\gamma_{n-2}, \quad z(n-2) = 1.$$

Thus

$$\begin{aligned}
N^t \boldsymbol{z} &= \boldsymbol{e}_{n-2}\sqrt{\gamma_{n-2}}, \\
z(n-1) &= -\sqrt{\mathring{e}_{n-2} \,/\, \mathring{q}_{n-1}} \\
z(n) &= -z(n-1)\sqrt{\mathring{e}_{n-1} \,/\, \mathring{q}_{n}} = \sqrt{\mathring{e}_{n-2}\mathring{e}_{n-1} \,/(\mathring{q}_{n-1}\mathring{q}_{n})} \\
z(i) &= -z(i+1)\sqrt{\hat{e}_i/\hat{q}_i}, \quad i < n-2 \\
\rho(\boldsymbol{z}) &= \gamma_{n-2}/\|\boldsymbol{z}\|^2 \\
z(n-1)^2 + z(n)^2 &= \frac{\mathring{e}_{n-2}}{\mathring{q}_{n-1}}\left(1 + \frac{\mathring{e}_{n-1}}{\mathring{q}_{n}}\right) \\
&= \frac{e_{n-2}q_{n-1}}{(q_{n-1}+s_{n-1})^2}\left(1 + \frac{e_{n-1}q_n}{(q_n+s_n)^2}\right).
\end{aligned}$$

Thus the new entries, $\overset{o}{q}$ and $\overset{o}{e}$, are not needed explicitly and the variable $s = s_{n-1} = -\tau(1 + e_{n-1}/(q_n - \tau))$ suffices.

As in Case 4 we sum the $z(i)^2$, $i \neq n-2$, until the sum settles down to 1% or exceeds 9/16 whichever comes first. In the latter case we use $\frac{1}{4}\rho$ as a default shift. Otherwise, using our latest estimate of $\|\boldsymbol{z}\|^2$,

$$\tau = \rho\left(1 - \sqrt{\|\boldsymbol{z}\|^2 - 1}\right) \approx \gamma_{n-2}\left(1 - \sqrt{\|\boldsymbol{z}\|^2 - 1}\right)/\|\boldsymbol{z}\|^2.$$

Recall that $s_{n-1} < 0$ and the virtue of the approximations used above lies in the use of γ_{n-2} and $\gamma_{n-2} < d_{n-2} = d_{min}$.
Case 5 costs approximately 5 divisions (3 for the loop).

Case 6. $dmin \neq d_n$ nor d_{n-1} nor d_{n-2}.

This is the typical situation in early stages. Too much caution can provoke very slow convergence, too little caution provokes too many failures. Our escape is to increase the fraction of $dmin$ used if Case 6 occurred at the previous step. This information is available free of charge.

if (Case 6 last step) **then**
$$f = \frac{1}{4} + \frac{3}{4}f$$
else if (Case 6 just failed)
$$f = \frac{1}{12}$$
else
$$f = \frac{1}{4}$$
end if
$$\tau = f \cdot dmin$$

Let us consider a few instances of Case 6. If $dmin$ is much too large so that the selection $\tau = \frac{1}{4}dmin$ causes failure, and not a late failure, then τ is reset to $\frac{1}{4}\tau$, i. e. $dmin/16$. If that succeeds we use $\tau = \frac{1}{12}$ (new $dmin$) the next time. On the other hand if $dmin$ is close to λ_{min} and $\lambda_{min} \gg \max_j e_j$ then improvement with $dmin/4$ will be modest because the shift is too cautious. However the next iteration uses (1/2) (*new dmin*) and, after that, if Case 6 persists, (2/3)(*new dmin*) and then (7/9)(*new dmin*). At some stage either Case 6 no longer holds or a failure occurs and τ is reduced.

The treatment of Case 6 is the weak point of this implementation. If the program is given a qd-array that has almost converged (small e's) to eigenvalues in non-monotonic order then the calculation will reorder the eigenvalues slowly. The smaller the e's the slower is the reordering. Fortunately these cases seem to be rare.

6.3.4 One Eigenvalue Found in Eigtest ($n0 = n0in - 1$)

We note that the values d_n and $dmin$ refer to the eigenvalue accepted in Eigtest and deflated. Thus we are in the position of 'no eigenvalues found' Section 6.3.3 but with less information. Essentially $dmin \leftarrow dmin1$, $d_n \leftarrow d_{n-1}$, etc. We could try to imitate the strategy in Cases 2 and 3 but with no natural candidate for $gap2$. Instead we use a more powerful but more expensive choice that we call refined Rayleigh quotient and describe, in detail, in Section 7. Strictly speaking this is not an $O(1)$ formula for τ but, in extensive tests, it cost no more than 6 divisions (the minimum is 4).

Cases 7 and 8.

if ($dmin1 = d_{n-1}$ and $dmin2 = d_{n-2}$) **then**
  compute ρ (Rayleigh quotient) and $\|\boldsymbol{r}\|$
  $gap = \frac{1}{2}dmin2 - \rho$
  if ($gap > 0$ and $gap^2 > \|\boldsymbol{r}\|^2$) **then**
    $\tau = \max\left(\rho - \|\boldsymbol{r}\|^2/gap,\ \frac{1}{3}dmin1\right)$
  else
    $\tau = \max\left(\rho - \|\boldsymbol{r}\|,\ \frac{1}{3}dmin1\right)$
  end if
end if

These choices correspond to formulae (8) and (7) at the beginning of Section 6.3.2.

Case 9, non-asymptotic case.

$$\tau = \begin{cases} \frac{1}{2}\, dmin1, & \text{if } dmin1 = d_{n-1}, \\ \\ \frac{1}{4}\, dmin1, & \text{otherwise.} \end{cases}$$

6.3.5 Two Eigenvalues Found in Eigtest $(n0 = n0in - 2)$

In this situation $d_n, d_{n-1}, dmin1$ all refer to deflated quantities. However the refined Rayleigh quotient option is available. For gap we use the Gersgorin disk for the current α_{n-1} provided that $e_{n-1} < q_{n-1}/2$.

Case 10, asymptotic case.

if $(dmin2 = d_{n-2}$ and $2e_{n-1} < q_{n-1})$ **then**
  compute ρ (Rayleigh quotient) and $\|\boldsymbol{r}\|$
  $gap = \alpha_{n-1} - \beta_{n-2} - \rho$
  if $(gap > 0$ and $gap^2 > \|\boldsymbol{r}\|^2)$ **then**
    $\tau = \max\left(\rho - \|\boldsymbol{r}\|^2/gap,\ \dfrac{1}{3}dmin2\right)$
  else
    $\tau = \max\left(\rho - \|\boldsymbol{r}\|,\ \dfrac{1}{3}dmin2\right)$
  end if
end if

These choices correspond to formulae (8) and (7) at the beginning of Section 6.3.2.

Case 11, non-asymptotic case. $\tau = \frac{1}{4}\, dmin2$.

6.3.6 More Than Two Eigenvalues Found in Eigtest

Set $\tau = 0$.

6.4 Failure Loop

If $\tau > \lambda_{min}(UL)$ then $d_j < 0$ for some $j < n$ in the dqds transform.

The occurrence of NaN (Not a Number)

Suppose that $\hat{q}_i > 0$ for $i < j$, but $\hat{q}_j = 0$. Then

$$\begin{aligned}
d_j &= -e_j < 0 \\
temp &= q_{j+1}/\hat{q}_j = +\infty \\
\hat{e}_j &= e_j \cdot temp = +\infty \\
d_{j+1} &= d_j \cdot temp - \tau = -\infty \\
\hat{q}_{j+1} &= d_{j+1} + e_{j+1} = -\infty \\
temp &= q_{j+2}/\hat{q}_{j+1} = -0 \\
\hat{e}_{j+1} &= e_{j+1} \cdot temp = -0 \\
d_{j+2} &= d_{j+1} \cdot temp - \tau = (-\infty) \cdot (-0) - \tau = \text{NaN}.
\end{aligned}$$

Thus division by 0 for $j < n-2$ causes all variables after d_{j+2} to be NaN, including $dmin$. Our response is to set $\tau = 0$. The test is as follows

if $(dmin \neq dmin)$ **then** { **go to** $safedqd$} **end if**.

In IEEE arithmetic NaN is the only value not equal to itself. The payoff for having NaNs is that our inner loop in dqds is free of tests.

Convergence Masked by Negative d_n

Sometimes all values of d are positive except the last which is so small that we have convergence, in particular $\sigma + q_n$ is evaluated as σ. In such a case it is a pity to invoke another dqds transform just because $d_n = q_n < 0$.

if ($dmin < 0$ and $dmin1 > 0$ and $\hat{e}_{n-1}$ is negligible
  and $|\hat{q}_n|$ is negligible) **then**
  $\hat{q}_n \longleftarrow 0$
  $dmin \longleftarrow |dmin|$
end if

Note that with the ping-pong implementation ($Z \longrightarrow \hat{Z}$, $\hat{Z} \longrightarrow Z$) $\hat{q}$ and $\hat{e}$ here, will become q and e at the next invocation of Eigtest and will force deflation.

Late failure

If $dmin1 > 0$ but $dmin = d_n < 0$ then we have 'late failure'. This was introduced by Rutishauser in [10] and specialized to our case in [3]. There it is shown that $\tau + dmin$ is an extremely accurate lower bound on λ_{min} so this is our next shift and is guaranteed to succeed.

Early failure

When $dmin1 < 0$ then we set $\tau \leftarrow \tau/4$ and try again. This is a somewhat panicky reaction because in many cases τ is less than 0.1% too big. However there are cases when τ is much too large and we want a rapid descent of τ to 0. We allow two successive early failures before we set $\tau = 0$ to ensure success.

Here is the pseudo-code for this segment

> **repeat**
>   call Dqds($\tau, dmin$)
> $it = it + 1$
>   **if** $(dmin \neq dmin)$ **then**
> $\tau = 0$
>   **else if** $(dmin < 0)$ **then**
>   **if** (two times here) **then**
> $\tau = 0$
>   **else if** $(dmin1 > 0)$ **then**
> $\tau = \tau + dmin$
>   **else**
> $\tau = \frac{1}{4}\tau$
>   **end if**
>   **end if**
> **until** $dmin \geq 0$

6.5 Check for a Split

In the context of a ping-pong implementation ($Z \rightarrow ZZ,\ ZZ \rightarrow Z$) we only check for splits after 'pong' steps $ZZ \rightarrow Z$. This is because it is only e-values that are marked with $-\sigma$, not ee-values. Recall that it is only after a call to Spltck that the top index $i0$ can increase. See Section 3.

The code only invokes this check if $old\ e_{min} < 10^4\,\varepsilon^2\, old\ q_{max}$ or if $e_{min} < \varepsilon^2\sigma$ and so a split is likely to be found. The test must also update e_{min} and q_{max} in case a split is found.

7 Rayleigh Quotient Residual Bounds

We present some new eigenvalue bounds that exploit the Cholesky factorization and so we begin with more generality than needed for dqds. Let $\boldsymbol{u}$ be any unit vector and consider one step of inverse iteration using any symmetric matrix A. We invoke a specific A later. We employ a slightly unusual normalization. Write

$$A\boldsymbol{v} = \boldsymbol{u}\gamma, \qquad \boldsymbol{v}^t\boldsymbol{u} = 1.$$

Then

$$\rho(\boldsymbol{v}) = \frac{\boldsymbol{v}^t A\boldsymbol{v}}{\|\boldsymbol{v}\|^2} = \frac{\gamma}{\|\boldsymbol{v}\|^2}$$

and so

$$\begin{aligned}
\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{v}) &= \frac{(A\boldsymbol{v} - \boldsymbol{v}\rho)}{\|\boldsymbol{v}\|}, \\
&= \frac{\boldsymbol{u}\gamma}{\|\boldsymbol{v}\|} - \frac{\boldsymbol{v}\gamma}{\|\boldsymbol{v}\|^3}, \\
&= \frac{\gamma}{\|\boldsymbol{v}\|^3}\left(\boldsymbol{u}\|\boldsymbol{v}\|^2 - \boldsymbol{v}\right). \\
\|\boldsymbol{r}\| &= \frac{\gamma}{\|\boldsymbol{v}\|^3}\left(\|\boldsymbol{v}\|^4 + \|\boldsymbol{v}\|^2 - 2\|\boldsymbol{v}\|^2\right)^{1/2}, \quad (\text{ because } \ \boldsymbol{v}^t\boldsymbol{u} = 1) \\
&= \frac{\gamma}{\|\boldsymbol{v}\|^2}\left(\|\boldsymbol{v}\|^2 - 1\right)^{1/2} = \rho\left(\|\boldsymbol{v}\|^2 - 1\right)^{1/2}.
\end{aligned}$$

Invoke the lower bound (8) from Section 6.3.2. The eigenvalue λ closest to ρ satisfies

$$\begin{aligned}
\lambda &\geq \rho - \frac{\|\boldsymbol{r}\|^2}{gap} \\
&= \rho\left[1 - \frac{(\|\boldsymbol{v}\|^2 - 1)\rho}{gap}\right].
\end{aligned} \tag{14}$$

The closer $\|\boldsymbol{v}\|$ is to 1 the better is the bound. Now apply (14) to the case when

$$A = B^tB, \qquad \boldsymbol{u} = (0, \ldots, 0, 1)^t, \quad \text{and} \quad dmin = d_n.$$

Here

$$B = \mathrm{bidiag}\begin{pmatrix} & \sqrt{e_1} & & \sqrt{e_2} & & . & & \sqrt{e_{n-2}} & & \sqrt{e_{n-1}} & \\ \sqrt{q_1} & & \sqrt{q_2} & & . & & . & & \sqrt{q_{n-1}} & & \sqrt{q_n} \end{pmatrix}.$$

The condition $dmin = d_n \, (= q_n)$ suggests that the last entry in $\boldsymbol{v}$, $v(n) = \boldsymbol{u}^t\boldsymbol{v} = 1$, is dominant. Solving

$$B^tB\boldsymbol{v} = \boldsymbol{u}\gamma$$

shows that $\gamma = q_n = d_n$ and $v(j) = -\sqrt{e_j/q_j} \cdot v(j+1)$, $j = n-1, \ldots, 2, 1$. Denote $\boldsymbol{v}(1:n-1)$ by $\boldsymbol{x}$ to find

$$\begin{aligned} \|\boldsymbol{v}\|^2 &= 1 + \|\boldsymbol{x}\|^2, \\ &= 1 + \sum_{j=1}^{n-1}\left(\prod_{i=j}^{n-1}\frac{e_i}{q_i}\right). \end{aligned}$$

and

$$\rho = \frac{q_n}{1 + \|\boldsymbol{x}\|^2},$$

so that

$$\lambda \geq \rho\left(\frac{1 - \|\boldsymbol{x}\|^2\rho}{gap}\right).$$

Our idea is to use this formula provided that

$$x_{n-1}^2 = \frac{e_{n-1}}{q_{n-1}} \leq \frac{1}{2}$$

and to calculate $\|\boldsymbol{x}\|^2$ correct to 1% More precisely

$j = n - 1$
$prod = \dfrac{e_j}{q_j}$
$sum = prod$
repeat
  $j = j - 1$
  $oldprod = prod$
  $prod = prod \cdot \left(\dfrac{e_j}{q_j}\right)$
  $sum = sum + prod$
until $(100 \cdot \max(prod, oldprod) < sum)$
$sum = 1.05 \cdot sum$

Note that we continue until two successive terms are less than $sum/100$ and then we increase our estimate of $\|\boldsymbol{x}\|^2$ by 5%. We measured the number of times through the loop for our test matrices and the largest value was 3. To estimate gap we use the default procedure in Sections 6.3.4 and 6.3.5;

$$gap = \begin{cases} \frac{3}{4}dmin2 - \rho, & \text{one eigenvalue found} \\ \\ \alpha_{n-1} - \beta_{n-2} - \rho, & \text{two eigenvalues found.} \end{cases}$$

Finally

if ($gap > 0$ and $gap^2 > \rho^2 \cdot \|\boldsymbol{x}\|^2$) **then**
  use (8) for τ $(\rho - \|\boldsymbol{r}\|^2/\text{gap})$
else
  use (7) for τ $(\rho - \|\boldsymbol{r}\|)$
end if

8 The 2×2 Case

There is a special subroutine SLAS2 in the BLAS for the accurate computation of the singular values of a 2×2 bidiagonal matrix. To invoke it here would require the extraction of $\sqrt{q_1}$, $\sqrt{q_2}$, $\sqrt{e_1}$ and the subsequent squaring of the output. There has to be a better way. There is also a subroutine SLAE2 for calculating the eigenvalues of a 2×2 real symmetric matrix but its use would not guarantee high relative accuracy.

Our response is to tackle the case on its own merits. We seek the eigenvalues of

$$\begin{pmatrix} q_1 + e_1 & \sqrt{q_2 e_1} \\ \sqrt{q_2 e_1} & q_2 \end{pmatrix}.$$

We may arrange that $q_1 \geq q_2$. Rutishauser's formulae for the eigenvalues, see [8, Chapter 9], are

$$q_1 + e_1 + t\sqrt{q_2 e_1}, \quad q_2 - t\sqrt{q_2 e_1}$$

where $t \geq 0$ is the smaller root of the quadratic

$$t^2 + 2\left(\frac{\delta}{\sqrt{q_2 e_1}}\right)t - 1 = 0$$

and

$$\delta = \frac{(q_1 - q_2) + e_1}{2} \geq \frac{e_1}{2}.$$

A standard formula for t is

$$t = \frac{\sqrt{q_2 e_1}}{\delta + \sqrt{\delta^2 + q_2 e_1}}$$

and the larger root r may be written as

$$r = q_1 + e_1 + \frac{q_2 e_1}{\xi}. \tag{15}$$

In order to avoid large intermediate quantities ξ is computed from

$$\xi = \begin{cases} \delta[1 + \sqrt{1 + (q_2 e_1/\delta)/\delta}], & \text{if } q_2(e_1/\delta) < \delta, \\ \delta + \sqrt{\delta(\delta + q_2 e_1/\delta)}, & \text{otherwise .} \end{cases}$$

Note that $e_1/\delta < 2$ and $\xi > \sqrt{q_2 e_1}$. So the third term in (15) satisfies

$$\frac{q_2 e_1}{\xi} \leq \sqrt{q_2 e_1} \leq \sqrt{q_1 e_1} \leq \frac{1}{2}(q_1 + e_1)$$

and is below the mean of the first two terms. The smaller root comes from dividing the product $q_1 q_2$ by the larger root r.

From Rutishauser's formulae the smaller root is

$$q_2 - t\sqrt{q_2 e_1} = \sqrt{q_2}(\sqrt{q_2} - t\sqrt{e_1})$$

and $0 \leq t < 1$. Thus if $e_1 \leq (macheps)^2 q_2$ then the eigenvalues are q_1 and q_2 to working precision and there is no need to compute ξ. The only subtraction in the whole calculation is $q_1 - q_2 \geq 0$.

High relative accuracy follows from the fact that our algorithm can be interpreted as one step of the dqds algorithm with shift $s =$ the smaller root and dqds enjoys high relative accuracy in the nonnegative case, see [3]. More precisely $\hat{q}_1 = \xi$, $\hat{q}_2 = 0$, and the larger root is

$$r = \hat{q}_1 + \hat{e}_1 + s = ((q_1 - s) + e_1) + e_1 \frac{q_2}{\hat{q}_1} + s.$$

Pseudocode for the 2×2 Case

$$
\begin{array}{l}
\textbf{if } (q_1 < q_2) \textbf{ then } \text{swap } (q_1, q_2) \textbf{ end if} \\
\textbf{if } (e_1 > macheps^2 q_2) \textbf{ then} \\
\quad t = ((q_1 - q_2) + e_1)/2 \\
\quad s = q_2(e_1/t) \\
\quad \textbf{if } (s \le t) \textbf{ then} \\
\qquad s = q_2 e_1/(t(1 + \sqrt{1 + s/t})) \\
\quad \textbf{else} \\
\qquad s = q_2 e_1/(t + \sqrt{t(t+s)}) \\
\quad \textbf{end if} \\
\quad t = q_1 + (s + e_1) \\
\quad q_2 = q_2(q_1/t) \\
\quad q_1 = t \\
\textbf{end if} \\
\text{root1} = q_1(+\sigma) \\
\text{root2} = q_2(+\sigma)
\end{array}
$$

9 Ping-pong Implementation

Rutishauser realized that in the context of a continued fraction it is somewhat unnatural to give different names, q and e, to the variables and so he introduced

$$Z = (q_1, e_1, q_2, e_2, \ldots, e_{n-1}, q_n, e_n)$$

instead. This format acknowledges the 'locality' in qd algorithms. The next step is to allocate two arrays, say Z and ZZ to the algorithm. So that dqds maps Z to ZZ or vice versa.

There are two benefits that accrue from doubling the storage.

1. The ping-pong implementation alternates the mappings $Z \to ZZ$ and $ZZ \to Z$ and wastes no time simply moving variables from one location to another.

2. In case of failure, when the shift τ exceeds λ_{min}, it is trivial to try again with a new shift. The old array was not altered.

We have gone one more step in this direction. In order to improve 'locality' even more we use one array Z of length $4n$, defined as follows

$$Z = (q_1, qq_1, e_1, ee_1, q_2, qq_2, e_2, ee_2, \ldots, q_n, qq_n, e_n, ee_n)$$

where the last two values e_n and ee_n are treated as zero. This notation is hard on humans but nice for computers. The association is

$$\begin{aligned} q(j) &= Z(4j-3), \quad e(j) = Z(4j-1) \\ qq(j) &= Z(4j-2), \quad ee(j) = Z(4j). \end{aligned}$$

To distinguish between ping and pong we use the integer variable pp; $pp = 0$ for ping, and $pp = 1$ for pong. Here is the dqds transform in Z notation without the code for $dmin$ and $emin$.

$d = Z(1 + pp) - \tau$
for $j = 1, n-1$
  $Z(4j - pp - 2) = d + Z(4j + pp - 1)$
  $temp = Z(4j + pp + 1)/Z(4j - pp - 2)$
  $Z(4j - pp) = Z(4j + pp - 1) \cdot temp$
  $d = d \cdot temp - \tau$
end for
$Z(4n - pp - 2) = d$

In order to avoid unnecessary index calculations the loop is written out twice, one for $pp = 0$, the other for $pp = 1$. The calculation moves through Z with a local range of 6 indices at most. The reader is referred back to Section 1.1 that justifies the use of this fast dqds code when $safemin * q_{max} \leq e_{min}$.

The LAPACK convention that the user supply q's and e's as separate arrays prevents the use of Rutishauser's sensible idea of a single qd array and neutralizes our extension to permit the whole algorithm to operate on a single array Z of length $4n$. Our approach would not confer an advantage until $4n$ exceeds the cache size.

We have experimented with writing separate subroutines for ping and pong, thus removing the variable pp from the code. On some platforms the difference in speed is noticeable but not enough to persuade us to use it.

A test in the inner loop, (if $d \leq 0$) return ,is needed for arithmetic units that do not conform to IEEE754. See Section 13 for more details.

10 Prologue

Cautious programming requires that we check that the input is proper, namely

1. initial index $\leq$ final index
2. $0 \leq Z(i)$, all i.

If either condition fails calculation is halted immediately with err set to an appropriate value.

However there is more work to do. The top subroutine expects to receive the data in Rutishauser's Z format, $q(1), e(1), q(2), e(2), \ldots$ and it must be rearranged for the ping-pong implementation described in Section 9. This is easily done by moving items from last to first, i. e.

$$
\begin{array}{l}
\textbf{for } k = 2*n,\ 2,\ -1 \\
\quad Z(2*k) \leftarrow 0 \\
\quad Z(2*k-1) \leftarrow Z(k) \\
\quad Z(2*k-2) \leftarrow 0 \\
\quad Z(2*k-3) \leftarrow Z(k-1) \\
\textbf{end for}
\end{array}
$$

At the same time we compute the sum of the data which happens to be the trace of LU. At this time diagonal arrays are easily detected.

Note that if the $trace$ is 0 then all the eigenvalues are 0 and the program can terminate immediately with no calculation. Finally, if $trace > 0$ then it is sensible to scale Z by 2^m so that $trace \cdot 2^m$ is close to (overflow threshold)$^{1/2}$. This device makes better use of the exponent range of the number representation but care must be taken to avoid overflow in intermediate quantities created in choosing shifts.

11 Epilogue

At the start of Epilogue

$$Z = (q_1, qq_1, e_1, ee_1, q_2, qq_2, e_2, ee_2, \ldots)$$

but all the e's are negligible. The eigenvalues are in the q's.
Move all q's to the front: $Z(k) \leftarrow Z(4k-3)$, $k = 1, n$. Then we sort the q's, if necessary, into monotone decreasing order and, at the same time, we note the positions of any breaks in monotonicity in the q's. This knowledge is relevant if a standard sort routine is eventually replaced by a merge-sort routine.

$m = 0$
for $k = 1, n$
    if $(Z(k-1) < Z(k))$ **then**
        $m = m + 1$
        $Z(3 * n + m) = k$
    end if
end for

Finally any scaling done in Prologue is undone and the sum of the eigenvalues is computed and stored in $Z(2n+1)$ for comparison with the *trace* that is stored in $Z(2n+2)$. The value of m is stored in $Z(3n)$.

12 Absolute or Relative Accuracy?

The attraction of the dqds algorithm is that it can compute all the eigenvalues of a positive array with high relative accuracy with either small or no penalty in time compared with, say, the root free QR algorithm. That is fine, but suppose that the user is satisfied with absolute accuracy and wants speed. How much faster will our algorithm perform if the acceptance tests in Eigtest are relaxed? In addition we ask whether our algorithm can be modified nicely to allow either choice, relative or absolute, by the user? More precisely we do *not* want a parameter 'absrel' passed down into the low level code. The difficulty is that for relative accuracy the test for convergence is $q_n < \epsilon\sigma$ and σ is changing at each step whereas for absolute accuracy we demand $q_n < \epsilon\|Z\|$.

An ingenious solution was proposed by I.S. Dhillon. Create an extra parameter eigtest and update it in the code in exactly the same way as σ. However eigtest is initialized to 0 for relative accuracy and to $max_i(q_i + e_i)$ for absolute accuracy. With this mechanism eigtest gradually rises from

$max_i(q_i + e_i)$ to $max_i(q_i + e_i) + \lambda_{max} < 2\lambda_{max}$. Any quantity less than $\varepsilon\, eigtest$ is set to zero.

We found only a 10% or 15% speed up when using absolute accuracy instead of relative. This was deemed insufficient improvement to warrant inclusion.

13 Non-IEEE Platforms

If the computer system does not permit floating point exceptions such as 'divide by zero' or '$0 \cdot \infty$' then it is necessary to make a test (d ¡ 0) inside the inner loop of dqds. Such a test prevents the efficient pipelined implementation of the code and causes a significant degradation of performance on some machines. The reader is referred back to Section 1.1 where a two division version of dqd is presented. To make the code safe it is necessary to insert an extra test immediately after $\hat{q}(i) = d + e(i)$,

$$\textbf{if } (d < 0) \textbf{ return}.$$

To permit our code to run on any platform we pass a logical parameter *ieee* to the dqds subroutine. If ieee is true then *dqds* (1 /div) is used, otherwise the 2 division plus test version described here.

This slowdown in dqds ($2\div$) raises a subtle point. The dqd transform ($\tau = 0$) cannot fail and there is no need for the test ($d < 0$). Now it happens that each iteration after which an eigenvalue is detected usually employs a tiny or zero value of τ. This suggests an alternative strategy for the subroutine Eigtest. Instead of looking for negligible e_{n-1} (deflation) the program should check for convergence (q_n negligible) and when this occurs the next iteration invokes dqd, not dqds, to make e_{n-1} negligible. On average 25% of the iterations would use dqd with a resulting reduction in execution time. We have not implemented this strategy in order to keep the IEEE and non-IEEE versions as close as possible to each other.

14 Fatal Errors

If the program terminates satisfactorily the value of *err* is 0. On exit, a positive value of *err* signals premature termination caused by a fatal error. The first two cases concern invalid data. Table 1 below gives the meaning attached to positive values. Recall the *nin* is the length of the q-array.

err	Subroutine	Meaning
1	prologue	$nin < 1$
2	prologue	bad data: $e(i) \leq 0$ or $q(i) \leq 0$, for some i.
3	geteigs	a split was marked by a positive value in e
4	geteigs	current block of Z not diagonalized after $10n$ iterations (in inner while loop)
5	geteigs	termination criterion of outer while loop not met. Program created more than nin unreduced blocks.

Discussion of Table 1.

2. The program is intended to run on positive data, $q(i) > 0$, $i = 1, nin$, $e(i) > 0$, $i = 1, nin-1$. However zero values of e indicate that Z is a direct sum of unreduced subarrays and the program deals with this case naturally. We do not allow zero values of q because such data does not come from the LU factorization of a positive definite tridiagonal matrix.

   The values 3 and 5 should never occur. They indicate violations of the logic of the code.

3. The program inspects the e-array for negligible values. Any such value is overwritten by $-$(current value of σ, the accumulated shifts). When the time comes to process a segment that was split off at an earlier stage the code searches from the bottom for the first nonpositive e-value and sets σ to its negation. This value should never be negative.

4. We have set a maximum value, called big, on the number of dqds transformations allowed to diagonalize an unreduced section. We have set big to $10n$ for an array of length n. This is equivalent to $5n$ QR iterations except that our shift strategy is more powerful than the Wilkinson shift for tridiagonals. The code terminate with $err = 4$ if convergence occurred but was not detected by Eigtest.

5. The outer while loop is over the unreduced subarrays of Z. With nin entries in q the maximal number of subarrays is nin. So $whila$ should never attain the value $nin + 1$.

15 Timings and Comparisons

As mentioned in Section 1 the code may be used to compute singular values of a bidiagonal matrix B as well as the eigenvalues of a symmetric tridiagonal matrix T.

Here are the codes used in the comparisons.

DBDSQR 1.0 (the original LAPACK 1.0 code for singular values). This is based on the Demmel-Kahan (1991) algorithm which uses a neatly coded bidiagonal QR transformation with 0 shift to compute the small singular values to high relative accuracy. When the singular values less than $\|B\|/10^3$ have been found the program switches to the standard shift strategy for the sake of efficiency.

DSTERF (the Pal-Walker-Kahan version of root free QR). This is LAPACK's current program for computing eigenvalues of T. In general the small eigenvalues are not computed to high relative accuracy because they are not determined to high relative accuracy by the entries in T.

DLASQ1 2.0 (the LAPACK 2.0 routine for singular values of B). This is the first implementation of dqds. Work on the code was begun in Berkeley in 1992 and was completed independently by K.Vince Fernando in 1994. The code does not assume IEEE arithmetic. The program was delivered without enough documentation to understand the reasons for the various features and it turned out to be significantly slower than DSTERF (=PWK) for finding eigenvalues. This presents the user with a trade-off between high relative accuracy (when the data warrants it) and speed whereas the original promise of the dqds algorithm (see [3]) was that it might dominate PWK on both counts. The new code is sometimes faster and sometimes slower than PWK but the timings are close except on the SUN Ultra 30.

We now mention a few results from extensive tests on the new version.

Arithmetic Effort. On all cases in our challenging collection of test matrices

$$\# \text{ divisions} < 3n^2,$$

where n is the order of the matrix. It is more informative to give an operation count rather than the number of iterations. The coefficient 3 was a pleasant surprise.

Rejection rate. (shift exceeds λ_{min}) This varies between 0 and 6% but is usually under 2% except for the nastiest test matrices. Recall that the shift strategy must balance the (obvious) cost of a rejected transform and the (subtle) cost of shifts that are too cautious. Clearly there is room for further study of this feature.

IEEE platforms. There is a significant performance payoff for using IEEE arithmetic, in particular infinity and NaN arithmetic (see details below). The IEEE mode permits the code to remove a test from the inner loop of the dqds transform, see Section 13.

Notation. Henceforth IEEE and non-IEEE refer to the LAPACK 3.0 DLASQ1 subroutine (it supersedes DLASQ1 2.0). The average speedups are as follows in Table 1, for 3 machines: an HP712, IBM RS6000, and SUN Ultra 30 (the results on the HP712 and IBM RS6000 were obtained with the LAPACK 3.0 code, June 30, 1999, while on the SUN Ultra 30 with the LAPACK 3.0 code, modified on December 14, 1999).

Warning
There are machines (SGI, for example) which provide an IEEE option only by slowing down every arithmetic operation and thus negating the goal of the IEEE floating point standard. On such machines the non-IEEE version of the new code should be chosen.

	HP712	IBM RS6000	SUN Ultra 30
non-IEEE / IEEE	1.70	1.80	1.28
DLASQ1 2.0 / IEEE	2.97	3.16	2.64
DSTERF / IEEE	0.92	1.02	0.65

Table 1: IEEE

Performance Comparisons.

Here are the results on 9 test matrices, which are described below, for the same machines used in Table 1.
Here is the how the tables are organized. There are 6 rows:

Row (1) matrix dimension

Row (2) Runtime(IEEE) in seconds

Row (3) Runtime(non-IEEE) / Runtime(IEEE). This measures the benefit of IEEE arithmetic. High relative accuracy is attained.

Row (4) Runtime(DLASQ1 2.0) / Runtime(IEEE). The ratios measure advantages of the new code for IEEE machines.

Row (4*) Runtime(DLASQ1 2.0) / Runtime(non-IEEE). The ratios measure the relative efficiency of the two versions of dqds which ignore the advantages of IEEE arithmetic.

Row (5) Runtime(DBDSQR 1.0) / Runtime(IEEE). The ratios measure improvement over the Demmel-Kahan (QR) algorithm.

Row (6) Runtime(DSTERF) / Runtime(IEEE). This row shows that there is little or no time penalty (except on SUNs) for computing the eigenvalues to high relative accuracy.

There are 10 columns, the last nine for the 9 test matrices, and the first for the Average over all these. All runs are double precision.

HP712
From Table 2, we see that IEEE speeds up the code 27% to 83%, 70% on average. The speed up over DLASQ1 2.0 is 1.40x to 7.48x, average 2.97x. The speedup over the DBDSQR 1.0 averages 4.88x. The code is sometimes faster and sometimes slower than DSTERF, 8% slower on average, but faster if Matrix #4 is omitted.

IBM RS6000
From Table 3, we see that IEEE speeds up the code 57% to 102%, 80% on average. The speed up over DLASQ1 2.0 is 1.34x to 8.52x, average 3.16x. The speedup over DBDSQR 1.0 averages 5.37x. The code is sometimes faster and sometimes slower than DSTERF, 2% faster on average, 8% if Matrix #4 is omitted.

SUN Ultra 30
From Table 4, we see that IEEE speeds up 34%, 28% on average. The speed up over DLASQ1 2.0 is 1.15x to 7.82x, average 2.64x. The speedup over

	Avg	#1	#2	#3	#4	#5	#6	#7	#8	#9
(1)		330	494	496	500	966	1687	2000	2000	2053
(2)		0.08	0.21	0.25	0.01	0.77	2.59	3.92	3.72	2.95
(3)	1.70	1.61	1.78	1.83	1.27	1.73	1.77	1.75	1.76	1.82
(4)	2.97	1.40	7.48	1.78	4.18	1.57	1.60	1.40	1.52	5.82
(4*)	1.78	0.87	4.20	0.97	3.29	0.91	0.90	0.80	0.86	3.20
(5)	4.88	3.36	4.51	8.82	3.00	4.66	4.72	5.10	4.62	5.14
(6)	0.92	0.66	1.01	1.21	0.27	0.99	0.99	1.09	1.01	1.09

Table 2: HP712

	Avg	#1	#2	#3	#4	#5	#6	#7	#8	#9
(1)		330	494	496	500	966	1687	2000	2000	2053
(2)		0.08	0.20	0.24	0.01	0.73	2.68	4.36	3.88	2.85
(3)	1.80	1.68	1.92	1.95	1.57	1.89	1.85	1.56	1.79	2.02
(4)	3.16	1.48	8.52	1.95	5.29	1.82	1.69	1.34	1.59	4.73
(4*)	1.74	0.88	4.44	1.00	3.37	0.96	0.91	0.86	0.89	2.34
(5)	5.37	3.68	4.90	9.55	5.43	5.17	4.76	4.78	4.66	5.44
(6)	1.02	0.69	1.23	1.24	0.57	1.21	1.11	0.97	1.12	1.10

Table 3: IBM RS6000

DBDSQR 1.0 averages 2.75x. The code is 35% slower than DSTERF on average, 31% if Matrix #4 is omitted.

	Avg	#1	#2	#3	#4	#5	#6	#7	#8	#9
(1)		330	494	496	500	966	1687	2000	2000	2053
(2)		0.04	0.08	0.06	0.004	0.40	1.40	1.94	2.04	1.57
(3)	1.28	1.23	1.34	1.29	1.20	1.26	1.28	1.33	1.27	1.28
(4)	2.64	1.23	7.82	1.35	4.61	1.25	1.27	1.15	1.19	3.91
(4*)	2.06	1.00	5.85	1.04	3.86	0.99	0.99	0.86	0.94	3.04
(5)	2.75	2.11	3.29	3.74	2.24	2.54	2.51	3.03	2.46	2.81
(6)	0.65	0.51	0.89	0.72	0.34	0.66	0.64	0.77	0.65	0.71

Table 4: SUN Ultra 30

Descriptions of test matrices: all except #7 and #8 have clusters of close values.

- #1 γ_{330}. This is a glued Wilkinson matrix-type bidiagonal B. Start with an 11 by 11 bidiagonal with diag = (1,11,21,31,41,51,41,31,21,11,1) and 10 off-diagonal 1's. 30 copies of this are joined together by an off-diagonal entry $\gamma = 10^{-4}$.

  The next 2 matrices were produced by Dr. I. S. Dhillon, IBM, Almaden.

- #2 inder_494. The eigenvalues are selected in geometric progression from *macheps* to 1.0 but with a random sign. The leftmost eigenvalue was approximately -0.86 so the matrix was translated by 0.86 to make its smallest eigenvalue *macheps*. Consequently there is a concentration at 0.86.

- #3 inder_496. A tight cluster of 247 eigenvalues at *macheps*, another tight cluster of 248 at 2.0, and a singleton at 1.0.

- #4 lapack_500. A random bidiagonal matrix with each entry of the form e^x where x is chosen uniformly from the interval $[2\ln(ulp),\ -2\ln(ulp)]$. For double precision $ulp \approx 2 \times 10^{-16}$.

  Three symmetric tridiagonal matrices supplied by George Fann of the Pacific Northwest Laboratories (Washington). They arise from reduction to tridiagonal form of matrices generated in the modeling of molecules using Moller_Plesset theory. The first two arrived positive definite and the third was made so by a suitable translation. Their chief feature is the presence of large clusters of eigenvalues agreeing to more than three decimals.

- #5 fann_966, #6 fann_1687, #9 fann_2053

- #7 tridiagonal [1 2 1] matrix.

- #8 bidiagonal from random normal(0,1) dense matrix (a "random" example).

Additional experiments.

Additional tests were performed on the SUN Ultra 30, using matrices defined in the same way as #4 above. We looked at the smallest eigenvalues of the matrices to see the effects of underflow, as shown in Tables 5, 6 and 7. We have paid a modest performance penalty in order to guard against unnecessary underflows, see Section 1.2, and these examples show the reward. DLASQ1 2.0 does not deliver high relative accuracy in the small eigenvalues in these admittedly extreme cases.

References

[1] P. Deift, T. Nanda, and C. Tomei, *ODEs and the Symmetric Eigenvalue Problem*, SIAM. J. Num. Anal.,vol. 20 (1983).

[2] J. Demmel and W. Kahan, 'Accurate singular values of bidiagonal matrices'. SIAM J. Sci. Stat. Comput., vol. 11 (1990), pp. 873–912.

[3] K. V. Fernando and B. N. Parlett, 'Accurate Singular Values and Differential qd Algorithms'. Numerische Mathematik, vol. 67, (March 1994), no. 2, pp. 191-229.

[4] *'Standard for Binary Floating Point Arithmetic'*, ANSI/IEEE, Standard 754-1985, New York, 1985.

[5] Ren-Cang Li, 'Relative perturbation theory: (I) Eigenvalue and singular value variations'. SIAM J. Matrix Anal. Appl., vol. 19 (1998), pp. 956–982.

[6] R. C. Li, 'On Deflating Bidiagonal Matrices'. Manuscript, Dept. of Mathematics, University Of California, Berkeley, 1994.

[7] B. N. Parlett and I. S. Dhillon, 'Fernando's Solution to Wilkinson's Problem: An Application of Double Factorization'. Linear Algebra and Its Applications, vol. 267 (1997), pp. 247-279.

[8] B. N. Parlett, 'The Symmetric Eigenvalue Problem'. (2nd Edition) SIAM, Philadelphia, 1998. 398 pp.

[9] H. Rutishauser, 'Der Quotienten-Differenzen-Algorithmus' Z. Angew. Math. Phys., vol. 5 (1954), pp. 233–251.

i	DLASQ1 3.0	DLASQ1 2.0	DBDSQR 1.0
1	5.054705201724986D-201	0.000000000000000D+00	5.054705201724984D-201
2	3.317864966646925D-149	0.000000000000000D+00	3.317864966646925D-149
3	4.733752490783757D-144	0.000000000000000D+00	4.733752490783753D-144
4	9.828918027083800D-111	9.828918027083799D-111	9.828918027083799D-111
$\vdots$			

Table 5: $n = 176$, λ_{max} =1.768773459351182D+31.

i	DLASQ1 3.0	DLASQ1 2.0	DBDSQR 1.0
1	0.000000000000000D+00	0.000000000000000D+00	0.000000000000000D+00
2	1.057826100728532D-155	0.000000000000000D+00	1.057826100728532D-155
3	3.333457324674396D-145	0.000000000000000D+00	3.333457324674396D-145
4	4.796619388807402D-145	0.000000000000000D+00	4.796619388807396D-145
5	2.332162748580873D-94	2.332162748580872D-94	2.332162748580873D-94
$\vdots$			

Table 6: $n = 220$, λ_{max} =1.222681157167759D+31.

i	DLASQ1 3.0	DLASQ1 2.0	DBDSQR 1.0
1	0.000000000000000D+00	0.000000000000000D+00	0.000000000000000D+00
2	1.609689649050070D-255	0.000000000000000D+00	1.609689649050069D-255
3	7.950613279965629D-223	0.000000000000000D+00	7.950613279965628D-223
4	1.604282196061483D-219	0.000000000000000D+00	1.604282196061484D-219
5	2.682262848923080D-186	0.000000000000000D+00	2.682262848923082D-186
6	1.347884608105250D-164	0.000000000000000D+00	1.347884608105250D-164
7	2.474348548254357D-146	0.000000000000000D+00	2.474348548254356D-146
8	4.886735556232942D-124	4.886735556232942D-124	4.886735556232941D-124
$\vdots$			

Table 7: $n = 343$, λ_{max} =1.147283779644497D+31.

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

[11] H. Rutishauser, *'Lectures on Numerical Mathematics'*, Birkhäuser, Boston,1990.

[12] D. S. Watkins, Isospectral Flows, SIAM Rev., vol. 26 (1984), pp. 379–391.

[13] D. S. Watkins and L. Elsner, Self-Similar Flows, Linear Algebra and Its Applications, vol. 110 (1988), pp. 213–242.