SIAM J. SCI. COMPUT.
Vol. 36, No. 3, pp. C290–C308
© 2014 Society for Industrial and Applied Mathematics

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

AN IMPROVED DQDS ALGORITHM*

SHENGGUO LI†, MING GU‡, AND BERESFORD N. PARLETT‡

Abstract. In this paper we present an improved DQDS algorithm for computing all the singular values of a bidiagonal matrix to high relative accuracy. There are two key contributions: a novel deflation strategy that improves the convergence for badly scaled matrices, and some modifications to certain shift strategies that accelerate the convergence for most bidiagonal matrices. These techniques together ensure linear worst case complexity of the improved algorithm (denoted by V5). Our extensive numerical experiments indicate that V5 is typically 1.2x–4x faster than DLASQ (the LAPACK-3.4.0 implementation of DQDS) without any degradation in accuracy. On matrices for which DLASQ shows very slow convergence, V5 can be 3x–10x faster. We develop a hybrid algorithm (HDLASQ) by combining our improvements with the aggressive early deflation strategy (AggDef2 in [Y. Nakatsukasa, K. Aishima, and I. Yamazaki, *SIAM J. Matrix Anal. Appl.*, 33 (2012), pp. 22–51]). Numerical results show that HDLASQ is the fastest among these different versions.

Key words. DQDS, singular value, d-deflation, DLASQ, LAPACK

AMS subject classifications. 15A18, 65F15, 65F30

DOI. 10.1137/120881087

1. Introduction.

The DQDS (differential quotient difference with shifts) algorithm of [4] computes the singular values of bidiagonal matrices to high relative accuracy. A detailed account of its efficient implementation, which is essentially the LAPACK code, can be found in [16]. One critical technique responsible for its success is the rather complex shift strategy. An aggressive early deflation (AED) strategy is suggested in [12] to further enhance performance for large matrices.

Our work is motivated by the recent discovery of significant slowdown of the LAPACK routine DLASQ[1] on certain matrices which have diagonal and off-diagonal entries that are of varying magnitude and are disordered (far from monotone decreasing). The improvements of our algorithm are

1. the ability to deflate converged singular values that are far from the bottom of the matrix,
2. an improved bound on the current smallest singular value,
3. some modifications to certain shift strategies, and
4. a guarantee that the convergence is never slower than that for the bisection algorithm.

*Submitted to the journal's Software and High-Performance Computing section June 14, 2012; accepted for publication (in revised form) March 12, 2014; published electronically June 24, 2014.
http://www.siam.org/journals/sisc/36-3/88108.html

†College of Computer Science, and the State Key Laboratory for High Performance Computation, National University of Defense Technology, Changsha 410073, China (nudtlsg@gmail.com). The research of this author was supported by 973 Program of China under grant 61312701001, by Natural Science Project of NUDT (JC120201), and in part by National Natural Science Foundation of China (60921062 and 61201328).

‡Department of Mathematics, University of California, Berkeley, CA 94720 (mgu@math.berkeley.edu, parlett@math.berkeley.edu). The research of the second author was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under contract DE-AC02-05CH11231, and by NSF awards CCF-0830764 and CCF-1319312.

[1]Version 3.4.0 or earlier. The latest version of DLASQ has already adapted our strategies and therefore runs in linear time in the worst case.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Our improved implementation (denoted by V5) is 1.2x–4x faster than DLASQ in general and up to 10x faster for matrices for which the DQDS algorithm converges slowly. It should be mentioned that the DQDS algorithm has found other uses than computation of the singular values, and the ideas in this paper may be of value in those situations as well [5].

V5 is never slower than the bisection algorithm in convergence, which establishes its linear complexity in the worst case and ensures a more robust implementation of the DQDS algorithm. There is a similar result for the QR algorithm with Wilkinson shift for the symmetric tridiagonal eigenproblems [14].

V5 and AED are complementary: AED may help for matrices which are easy for DQDS, whereas V5 improves the convergence for the disordered matrices. By combining V5 with AED, a more efficient algorithm is obtained in section 6. This hybrid algorithm is denoted by HDLASQ and is shown to be faster than DLASQ, V5, and AggDef2 (proposed in [12]) for both easy and difficult matrices.

This paper is organized as follows. Sections 2 and 3 introduce the DQDS algorithm and its LAPACK implementation DLASQ, respectively. Section 4 discusses the five important improvements over DLASQ. Section 5 proves the linear worst case complexity of V5. Section 6 compares V5 with DLASQ and AggDef2, and introduces the hybrid algorithm HDLASQ. Numerous experiments on synthetic and practical matrices are also reported in section 6. Section 7 presents the conclusions.

2. Essential information on DQDS. In this section we gather essential properties of the DQDS algorithm. The reader is expected to have some acquaintance with this algorithm. For an introduction to the topic we recommend [13], and for historical details see [4]. By convention our bidiagonal matrices are all upper bidiagonal. Our notation is

$$B = \begin{bmatrix} a_1 & b_1 & & & & \\ & a_2 & b_2 & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & & & & a_{n-1} & b_{n-1} \\ & & & & & a_n \end{bmatrix}, \tag{2.1}$$

and all the elements of B are positive.

In addition to the DQDS algorithm, we refer to a simpler procedure *OQD* (orthogonal qd) that transforms B into $\hat{B}$ so that

$$BB^T = \hat{B}^T\hat{B};$$

see [4].

ALGORITHM 1. [OQD]

$\tilde{a}_1 = a_1$
for $k = 1, \ldots, n-1$
  $\hat{a}_k := \sqrt{\tilde{a}_k^2 + b_k^2}$
  $\hat{b}_k := b_k * (a_{k+1}/\hat{a}_k)$
  $\tilde{a}_{k+1} := \tilde{a}_k * (a_{k+1}/\hat{a}_k)$
end for
$\hat{a}_n := \tilde{a}_n$

ALGORITHM 2. [DQDS]

$d_1 = q_1 - s,$
for $k = 1, \ldots, n-1$
  $\hat{q}_k := d_k + e_k$
  $\hat{e}_k := e_k * (q_{k+1}/\hat{q}_k)$
  $d_{k+1} = d_k * (q_{k+1}/\hat{q}_k) - s$
end for
$\hat{q}_n := d_n$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

The DQD algorithm can be obtained by squaring the variables in the OQD algorithm,

$$q_k = a_k^2, \quad e_k = b_k^2, \quad k = 1, 2, \ldots, n, \quad b_n = 0. \tag{2.2}$$

The DQDS algorithm is obtained by incorporating a shift in the DQD algorithm. The shift will be denoted by s.

The point of introducing OQD is to exhibit the relation of the $\{q, e\}$ variables to the $\{a, b\}$ variables of B. However, an alternative connection is to define

$$L = \begin{bmatrix} 1 & & & & & \\ e_1 & 1 & & & & \\ & e_2 & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & 1 & \\ & & & & e_{n-1} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} q_1 & 1 & & & & \\ & q_2 & 1 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & 1 \\ & & & & & q_n \end{bmatrix}.$$

With some algebraic manipulations, one can show that the DQDS algorithm implements the transform

$$\hat{L}\hat{U} = UL - sI. \tag{2.3}$$

One step of the DQDS transform is to compute $\hat{L}$ and $\hat{U}$ from L and U.

An essential ingredient in the success of DQDS is that it never explicitly forms any products UL or LU but works entirely with the factors. Furthermore, LU and B^TB are connected by a diagonal similarity transformation [13],

$$LU = \Delta B^T B \Delta^{-1}, \tag{2.4}$$

where

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

Therefore LU and B^TB have the same eigenvalues, and $\sigma(B) = \sqrt{\lambda(LU)}$.

Recall that one step of the LR algorithm [15] transforms the matrix A into matrix $\hat{A}$ by

$$\begin{aligned} A - sI &= \tilde{L}\tilde{U}, \\ \hat{A} &= \tilde{U}\tilde{L} + sI, \end{aligned}$$

where $\tilde{L}$ is a lower triangular matrix and $\tilde{U}$ is an upper triangular matrix. Note that $\hat{A} = \tilde{L}^{-1}A\tilde{L}$. We say that the shift is restored. In contrast to the LR algorithm, the DQDS transform is a nonrestoring similarity transformation on UL,

$$\hat{U}\hat{L} = \hat{L}^{-1}(\hat{L}\hat{U})\hat{L} = \hat{L}^{-1}(UL - sI)\hat{L}. \tag{2.5}$$

Notice that $\hat{U}\hat{L}$ is not similar to UL and the shift s is not restored. After each DQDS transform, all eigenvalues are reduced by s. Because of the nonrestoring feature, the DQDS algorithm must keep track of the accumulated shifts, $S = \sum_{i=1}^{k} s_i$.

The DQDS algorithm is the repeated applications of the DQDS transform with a sequence of well chosen shifts. This algorithm checks, before each transform, whether e_{n-2} or e_{n-1} is negligible. If so, then one or two eigenvalues have been computed

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

accurately and the computation resumes on a smaller array. This process is called deflation. A well-known feature of the QR-type and LR-type algorithms is that the larger entries migrate gradually to the top of the matrix and the smaller ones precipitate to the bottom. The smaller they are, the quicker they fall. That is why the classical deflation strategies detect the bottom 2×2 submatrix.

Detecting small entries above the bottom is also very important. Setting a negligible e_k to zero where $k < n-2$ is usually called *splitting* [16] instead of deflation. In this case, a big problem breaks into two independent smaller problems. It is vital to check for splitting since, if $e_k = 0$, the two subarrays should be treated independently, whereas a tiny $e_k \neq 0$ could significantly slow down convergence if left untreated. The criteria for negligibility have received careful attention and the tests can be complicated. A novel deflation strategy is proposed in section 4.1 for the case when an intermediate variable (e_k or d_k for $k < n-2$) is negligible.

In the 1960s Kahan proved that a bidiagonal matrix defines all its singular values to high relative accuracy; see [3]. In the context of DQDS this precious feature will be preserved if all of the $\{q, e\}$ variables remain positive [4]. Thus the shift s must obey

$$s < \lambda_{\min}(BB^T). \tag{2.6}$$

This constraint means that the singular values must be found in monotone increasing order. This is in contrast to the QR algorithm with shifts, but the QR algorithm does not claim to compute the eigenvalues to high relative accuracy except when the shifts are zero.

The $\{q, e\}$ variables require so little storage that it is sensible to compute $\{\hat{q}, \hat{e}\}$ in a separate location from $\{q, e\}$, and then decide whether to accept or reject the transform. If any new variables are negative, the algorithm rejects the transform and chooses a smaller shift. This is called a *failure*. The ability to reject a transform permits an aggressive shift strategy.

2.1. Some theoretical results. In the following sections we use the B notation to describe the DQDS algorithm.

THEOREM 2.1 (Theorem 2 in [4]). *Apply the DQD transform (not DQDS) to a positive bidiagonal matrix B, and $d_1, \ldots, d_n$ are the intermediate values. Then*

1. $\sigma^2_{\min}(\hat{B}) \leq \min_k\{d_k\}$,
2. $[(BB^T)^{-1}]_{k,k} = d_k^{-1}$,
3. $(\sum_{k=1}^n d_k^{-1})^{-1} \leq \sigma^2_{\min}$.

From item 1 $d_{\min} = \min_k\{d_k\} \geq \sigma^2_{\min}(\hat{B})$. Asymptotically, $d_{\min}$ is a very good estimate of $\sigma^2_{\min}(\hat{B})$, and therefore DLASQ records $d_{\min}$ as a guide to choosing a shift. But at the early stage, $d_{\min}$ may be too big. There is more on this topic later.

COROLLARY 2.2. *With the notation in Theorem* 2.1*, we have*

$$\frac{1}{n} d_{\min} \leq \sigma^2_{\min}(\hat{B}) \leq d_{\min} \leq n\sigma^2_{\min}(\hat{B}). \tag{2.7}$$

Corollary 2.2 implies that $d_{\min}$ becomes negligible when the matrix becomes nearly singular. It is the basis of our deflation strategy in section 4.1. There are similar results for the DQDS algorithm.

THEOREM 2.3 (Theorem 3 in [4]). *If the DQDS with shift $s \geq 0$ transforms positive bidiagonal matrix B into positive $\hat{B}$ with intermediate quantities $d_1, \ldots, d_n$, then*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

1. $\sigma^2_{\min}(\hat{B}) \le \min_k\{d_k\}$,
2. $[(BB^T)^{-1}]_{k,k} < d_k^{-1}$,
3. $(\sum_{k=1}^n d_k^{-1})^{-1} < \sigma^2_{\min}(\hat{B})$.

COROLLARY 2.4.

$$\frac{1}{n}d_{\min} < \sigma^2_{\min}(\hat{B}) \le d_{\min} < n\sigma^2_{\min}(\hat{B}). \tag{2.8}$$

These two corollaries have the following very important practical implications:

- A good shift should always be in $[\frac{d_{\min}}{n}, d_{\min}]$.
- A very small $\sigma_n^2(\hat{B})$ automatically implies a very small $d_{\min}$. If $d_{\min}$ is negligible then it is possible to deflate the matrix even if $d_{\min} = d_k$, $k \ll n$; see section 4.1.

3. Features of the current LAPACK implementation. The algorithm presented in [16] was designed for speed on all of the test matrices available before 2000. It was, on average, 5x faster than the Demmel–Kahan QR code for computing singular values [16]. The drive for efficiency produced a more complex shift strategy than that used in the previous LR/QR codes for the tridiagonal eigenvalue problems. As an example, by unrolling the last three minor steps, the code keeps values d_{n-2}, d_{n-1}, and d_n available so that, in the asymptotic regime, there is enough information to make a good approximation to the new $\sigma_{\min}$ even after one or two eigenvalues are deflated. For example, d_{n-2} might be $d_{\min}$ for the deflated array.

The code keeps the information for both $\{q, e\}$ and $\{\hat{q}, \hat{e}\}$. If any entry in $\hat{q}$ or $\hat{e}$ is negative, the transform is rejected and the step is considered a failure and aborted. Consequently, an aggressive shift strategy may be employed. But the number of failures should be kept small. Too many failures will surely degrade the performance. To enhance data locality, the variables are held in one linear array,

$$Z = \{q_1, \hat{q}_1, e_1, \hat{e}_1, q_2, \hat{q}_2, e_2, \hat{e}_2, \ldots, q_n, \hat{q}_n\}.$$

The DQDS maps $\{q, e\}$ to $\{\hat{q}, \hat{e}\}$ and vice versa to avoid data movement. This is called *ping-pong* implementation in [16].

The implementation of DQDS in the LAPACK, as of 2000, is based on the following perception:

> An explicit conditional statement (if-then-else) in an inner loop impedes efficient implementation on a pipelined arithmetic unit.

Consequently, the division in the DQDS loop, $t = q_{k+1}/\hat{q}_k$, is not protected from incurring an exception (division by zero or overflow). However, the powerful feature of arithmetic units conforming to IEEE floating point standard 754 is that computation is not held up by an exception. At the end of the loop, the code tests whether an ∞ or an NaN (not a number) occurred and acts appropriately.

The LAPACK implementation also assumes that a good compiler will implement the intrinsic FORTRAN functions such as MIN(A,B) or MAX(A,B) efficiently. Hence the valuable variable $d_{\min}$ is computed via $d_{\min}$= MIN($d_k, d_{\min}$) in the inner loop.

This implementation also gives up the knowledge of the index at which $d_{\min}$ receives its final value since this would require an explicit conditional,

```
if d_k < d_min then
    dmink = k; d_min = d_k;
end if,
```

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

where the position of $d_{\min}$ is denoted by $dmink$. However, our efficiency enhancements require knowledge of the index of $d_{\min}$. This explicit conditional statement need not impede performance provided that it is placed not in its natural position, but immediately after the division. This is because division is so slow relative to other operations, such as comparison, that the conditional statement can be completed before the preceding division finishes. Technically, this requires us to update $d_{\min}$ one minor step late, but that is easily dealt with. Algorithm 3 is the inner loop of a new implementation.

ALGORITHM 3. [inner loop of the DQDS algorithm]

for $k = 1$ to $n - 1$
  $\hat{q}_k = d_k + e_k$; $t = q_{k+1}/\hat{q}_k$;
  if $d_k < 0$ **then**
    exit *(early failure)*;
  else
    if $d_k < d_{\min}$ **then**
      dmink $= k$; $d_{\min} = d_k$;
    end if
  end if
  $\hat{e}_k = e_k \cdot t$; $d_{k+1} = d_k \cdot t - s$;
end for
if $d_n < 0$ **then**
  continue *(late failure)*;
else if $d_n < d_{\min}$ **then**
  dmink $= n$; $d_{\min} = d_n$;
end if

If $d_k < 0$ $(k < n)$ and shift $s > 0$, then by Algorithm 2 the DQDS algorithm may fail in three possible ways: $\hat{q}_j$ $(k < j < n)$ is negative or zero, or $\hat{q}_n < 0$. The case $(d_k < 0, k < n)$ is called an *early failure*. A *late failure* occurs when the arrays $\{\hat{q}_k, \hat{e}_k\}$ are all positive except for the last $\hat{q}_n = d_n < 0$. In the case of a late failure, a smaller shift can be chosen as $s + d_n$, which is guaranteed to succeed. This property was discovered by H. Rutishauser. See [16] for details.

4. Improvements. In this section, we summarize the improvements of our implementation over DLASQ, which can be divided into two types: *deflation strategies* and *shift strategies*.

4.1. Setting negligible $d_{\min}$ to zero. Recall that the DQDS algorithm is nonrestoring. The algorithm tries at every step to make the current matrix singular. The positive $\{q, e\}$ array defines both a positive bidiagonal matrix B and the matrices L, U; see [13]. It can happen that a leading principal submatrix of B becomes almost singular long before any negligible entries appear at the bottom of the matrix. This situation is not easily detected by a simple inspection of the entries $\{q_k, e_k\}$ or, say, $\{a_k, b_k\}$.

To explain, it is best to go back to the OQD transform from B^T to $\hat{B}$ and consider

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

the process after $(k-1)$ minor steps shown in the following equation:

$$B^{(k)} = Q_k B^T = \begin{bmatrix} \hat{a}_1 & \hat{b}_1 & & & & & & & \\ 0 & \hat{a}_2 & \hat{b}_2 & & & & & & \\ & 0 & \cdot & \cdot & & & & & \\ & & 0 & \hat{a}_{k-1} & \hat{b}_{k-1} & & & & \\ & & & 0 & \tilde{a}_k & 0 & & & \\ & & & & b_k & a_{k+1} & 0 & & \\ & & & & & b_{k+1} & a_{k+2} & & \\ & & & & & & \cdot & \cdot & 0 \\ & & & & & & & b_{n-1} & a_n \end{bmatrix}. \tag{4.1}$$

The striking feature is that row k is a singleton: its entry is $\tilde{a}_k$ and $\tilde{a}_k^2 = d_k$. Rows $(1:k)$ are upper bidiagonal, and rows $(k:n)$ are lower bidiagonal. Such matrices are often described as *twisted*. Corollary 2.2 says $\tilde{a}_k$ is small when $\hat{B}$ is nearly singular. How small must $\tilde{a}_k$ be to declare it negligible? Define new matrices $\tilde{B}$ and E, using (4.1), by

$$B^{(k)} = Q_k B^T = \tilde{B} + E,$$

where row k of $\tilde{B}$ is null and $E = \mathrm{diag}(0, \ldots, 0, \tilde{a}_k, 0, \ldots, 0)$. Observe that $E\tilde{B} = E^T\tilde{B} = 0$. Hence,

$$\begin{aligned} B^{(k)T}B^{(k)} &= (\tilde{B}+E)^T(\tilde{B}+E) \\ &= \tilde{B}^T\tilde{B} + \tilde{B}^TE + E^T\tilde{B} + E^TE \\ &= \tilde{B}^T\tilde{B} + E^TE. \end{aligned} \tag{4.2}$$

By Weyl's monotonicity theorem [7, Thm. 3.3.16]), for $i = 1:n$,

$$|\sigma_i^2 - \tilde{\sigma}_i^2| \le \|E^2\| = d_k. \tag{4.3}$$

Here $\{\tilde{\sigma}_i^2\}$ are the ordered singular values of $\tilde{B}$. The nonrestoring character of the DQDS algorithm entails that the desired eigenvalues are $\{\sigma_i^2 + S\}$, where S is the accumulated sum of shifts so far. Consequently, d_k $(\tilde{a}_k^2)$ may be set to zero when

$$d_k \le \epsilon S \le \epsilon(S + \sigma_i^2).$$

How does one deflate by exploiting such an event? Examination of the inner loop of *OQD*, Algorithm 1, shows that with $\tilde{a}_k = 0$, the algorithm simply moves the remaining variables into new positions

$$\hat{a}_j = b_j, \quad \hat{b}_j = a_{j+1}, \quad j = k, \ldots, n-1, \quad \text{and } \hat{a}_n = \tilde{a}_k = 0.$$

The bidiagonal $\hat{B}$ reveals its singularity ($\hat{a}_n = 0$), but deflation requires that $\hat{b}_{n-1}$ also vanishes. More work needs to be done since $\hat{b}_{n-1} = a_n$ may not be negligible. There are two options.

Option A. Apply the *OQD* transform to $\hat{B}$ to obtain $\bar{B}$, and note that $\bar{b}_{n-1} = \hat{b}_{n-1}(\hat{a}_n/\bar{a}_{n-1}) = 0$ and $\bar{a}_n = \tilde{a}_{n-1}(\hat{a}_n/\bar{a}_{n-1}) = 0$. The new singular value is $S + \hat{a}_n = S$, and $n \leftarrow n-1$. In practice, the DQD transform, not OQD, is used.

Option B. The procedure invoked in the aggressive early deflation algorithm (see [12]) is useful here. In our implementation, we use this option since we found

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

that it usually saves some floating point operations over Option A. We describe it briefly.

Apply a carefully chosen sequence of plane rotations on the right of $\hat{B}$ to chase the entry $\hat{b}_{n-1}$ up to the last column of $\hat{B}$. The sequence of "planes" is $(n-1,n)$, $(n-2,n),(n-3,n),\ldots,(1,n)$. The single nonzero entry ψ in the last column is called the bulge. Its initial value is $\hat{b}_{n-1}$ and it shrinks as it rises. The expectation is that the bulge will become negligible quickly in fewer than 10 rotations.

Let $\mathbf{e}_i$, $i=1,\ldots,n$, be the ith column of an $n\times n$ identity matrix (to distinguish it from e_i). When the bulge is in position (k,n), $k\le n-1$, the matrix can be written as $\overset{o}{B}+E$, where $E=\psi\mathbf{e}_k\mathbf{e}_n^T$ and $\overset{o}{B}$'s last row and column are null. Then $(\overset{o}{B}+E)(\overset{o}{B}+E)^T=\overset{o}{B}\overset{o}{B}^T+\psi^2\mathbf{e}_k\mathbf{e}_k^T$. It turns out that the criterion for neglecting the bulge ψ is the same as the criterion for neglecting d_k, namely,

$$\psi^2\le\epsilon S,$$

and ψ^2 is expressible in terms of $\{q_i,e_i\}$.

To be more specific, let $\psi=\sqrt{x}$. The bulge moves upward by one position by applying a Givens transformation from the right [12],

$$\begin{bmatrix} * & * & & & \\ & * & * & & \\ & & * & \sqrt{e_k} & \\ & & & \sqrt{q_{k+1}} & \sqrt{x} \\ & & & & 0 \end{bmatrix} \overset{G_k}{\longrightarrow} \begin{bmatrix} * & * & & & \\ & * & * & & \\ & & & \sqrt{\bar{e}_k} & \sqrt{\bar{x}} \\ & & & \sqrt{\bar{q}_{k+1}} & 0 \\ & & & & 0 \end{bmatrix},$$

where G_k is an orthogonal matrix of the form $\left[\begin{smallmatrix} c & s \\ s & -c \end{smallmatrix}\right]$ to keep the entries of B positive. The values $\bar{q}_{k+1}$, $\bar{e}_k$, and $\bar{x}$ can be computed as

$$\bar{q}_{k+1}=q_{k+1}+x,\quad \bar{e}_k=\frac{q_{k+1}e_k}{q_{k+1}+x},\quad \bar{x}=\frac{xe_k}{q_{k+1}+x}.$$

The whole algorithm is described as follows, where $\bar{q},\bar{e}$, and $\bar{x}$ are also denoted by q,e, and x, respectively.

ALGORITHM 4. [Option B]
$x=e_{n-1}$;
for $k=n$-1, n-2, $\ldots$, 2
  $t_1=q_k$;
  $q_k=q_k+x$;
  $t_2=1/q_k$;
  $x=x\cdot e_{k-1}\cdot t_2$;
  $e_{k-1}=e_{k-1}\cdot t_1\cdot t_2$;
  if x is negligible, **break**;
end for
if $k=1$, $q_1=q_1+x$;

Remark 1. The deflation strategy, setting negligible $d_{\min}$ to zero, is the most important improvement over DLASQ and is "tailor-made" for the disordered matrices for which the DQDS algorithm shows slow convergence. This deflation will be called *d-deflation strategy* in later sections.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

4.2. Improved criterion for late deflation. A crude, but adequate, criterion for setting b_{n-1} (in B) to zero is

$$|b_{n-1}| < c\epsilon(\sqrt{S} + \sigma_{\min}(B)),$$

where ϵ is the machine precision and c is a modest constant, for example, $c = 10$ in DLASQ and in our implementation. In the context of a $\{q, e\}$ array, this same criterion becomes approximately

$$e_{n-1} < (c\epsilon)^2 S. \tag{4.4}$$

Note that $q_n = d_n$ usually becomes very small; see Lemma 5.3 in [1]. Instead of testing only e_{n-1}, a more refined criterion arises from considering the trailing 2×2 submatrix of BB^T,

$$\begin{bmatrix} a_{n-1}^2 + b_{n-1}^2 & b_{n-1}a_n \\ b_{n-1}a_n & a_n^2 \end{bmatrix}.$$

By Weyl's theorem no eigenvalue of BB^T changes by no more than $b_{n-1}a_n$ if the 2×2 matrix $\left[\begin{smallmatrix} 0 & b_{n-1}a_n \\ b_{n-1}a_n & 0 \end{smallmatrix}\right]$ is subtracted from the trailing 2×2 submatrix shown above. However the $(n-1, n-1)$ entry of BB^T still involves b_{n-1}^2. No eigenvalue of BB^T can change by more than b_{n-1}^2 if it is neglected. We can set it to zero when it is negligible compared to either $S + \lambda_{\min}(BB^T)$ or a_{n-1}^2. Consequently the crude criterion (4.4) may be replaced by the following pair of tests:

$$b_{n-1}^2 < c\epsilon \max(S, a_{n-1}^2) \text{ and } b_{n-1}a_n < c\epsilon(S + \lambda_{\min}(BB^T).$$

In the context of $\{q, e\}$ array, we use

$$e_{n-1} < c\epsilon \max(S, q_{n-1}) \text{ and } e_{n-1}q_n < (c\epsilon S)^2. \tag{4.5}$$

The same arguments can give us a more refined test for splitting B whenever e_k and $e_k q_{k+1}$ are negligible.

4.3. Updating the upper bound. In his original papers on the qd algorithm, Rutishauser proposed updating the upper and lower bounds, *sup* and *inf*, on $\sigma_{\min}^2(B)$ at all times. The original paper [4] on the DQDS algorithm followed the recommendation but the implementation DLASQ [16] omitted to update *sup* when a transform failed.

A transform $\{q, e\}$ into $\{\hat{q}, \hat{e}\}$ fails if any entry in $\hat{q}$ or $\hat{e}$ is nonpositive (but $\hat{q}_n = 0$ is permitted). In this case the connection to a bidiagonal $\hat{B}$ is lost. It can happen only if the shift s exceeds $\sigma_{\min}(B)^2$, and thus *sup* can be set to s. This is valuable information.

Here is the pseudocode, with s being the shift.

ALGORITHM 5. [updating the upper bound]

if shift s succeeds, **then**
  $sup = \min\{d_{\min}, sup\text{-}s\}$,
else if shift s fails, **then**
  $sup = \min\{s, sup\}$,
end if

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

In case of failure, the transformed $\hat{q}$ and $\hat{e}$ are discarded. The shift is usually computed as $s = \alpha \cdot d_{\min}$ in DLASQ, where α is a parameter in $(0, 1)$. We replace DLASQ's $d_{\min}$ by *sup,* and the reward is a reduction in the number of failures. As mentioned before, $d_{\min}$ is used to estimate $\lambda_{\min}(BB^T)$. At the early stage, $d_{\min}$ may be too large and *sup* may be a much better upper bound than $d_{\min}$.

The negligibility of *sup* or $d_{\min}$ reveals the convergence of a shifted singular value to 0. DLASQ does not check its negligibility and so computes a complicated shift when shift $= 0$ would suffice.

4.4. Twisted shift in the last *p* rows. When $d_{\min}$ is near the bottom, we use the twisted factorization to choose a shift. We call this shift strategy *restricted twisted shift strategy* since it is only used when $d_{\min}$ is in the last p rows. Our experiments suggest $p = 20$ is a good choice. A similar strategy is used in DLASQ (Cases 4 and 5), when $d_{\min}$ is in the last two rows.

The idea behind this shift strategy is to compute an approximate eigenvector z of BB^T by focusing on the submatrix around $d_{\min}$. Using z and the Rayleigh quotient residual, we can compute a lower bound ϕ on the smallest eigenvalue with high likelihood that $\phi < \sigma_{\min}(B)$, and ϕ can be used as a shift; see section 6.3.3 of [16] for details.

With techniques similar to those in [1], we can show that the order of convergence for the twisted shift strategy is 1.5; see the following theorem.

THEOREM 4.1. *Assume the bidiagonal matrix B has positive nonzero entries. For the DQDS algorithm with the twisted shift strategy, the sequence $\{e_{n-1}^{(l)}\}_{l=0}^{\infty}$ converges to* 0 *with order of convergence* 1.5*.*

We leave the proof of Theorem 4.1 to the supplementary materials [9].

4.5. Using a suitable 2-by-2. Kahan [8] suggests a better method for computing an upper bound that requires little overhead. The idea is that the smallest singular value of the submatrix around $d_{\min}$ must be a better upper bound than $d_{\min}$ itself.

Let us consider the ODQ algorithm first. Let k be the index where $\tilde{a}_k = \min_i(\tilde{a}_i)$, and assume $k > 1$; see (4.1). By the interlacing property of the singular values [6], we know that the smallest singular value of any principle submatrix of $B^{(k)}$ is larger than $\sigma_{\min}(B)$. Thus the smallest singular value of $\begin{bmatrix} \hat{a}_{k-1} & \hat{b}_{k-1} \\ & \tilde{a}_k \end{bmatrix}$ is an upper bound on $\sigma_{\min}(B)$. This claim is also valid for the DQDS algorithm.

4.6. Our whole shift strategy. In this subsection we summarize the structure of our shift strategy. We modify the shifts of DLASQ [16] in two places.

- Cases 4 and 5 in DLASQ use the twisted shift strategy when $d_{\min}$ is in the last ($p =$) 2 rows. We replace it by $p = 20$.
- We replace all $d_{\min}$ by *sup*.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

ALGORITHM 6. [Our shift strategy]

(1) **if** $d_{\min} < 0$,
  $s = 0$; – Case 1
(2) **if** no eigenvalue deflated
  - **if** ($d_{\min}$ in the last 20 rows)
    - **if** ($d_{\min}$ in the last two rows),
      use the old shift strategy; – Case 2 and 3
    - **else**
      use the *twisted shift strategy*; – Case 4 and 5
    - **end if**
  - **else**
    - update *sup* via the technique in section 4.5;
    - use the old strategy and replace $d_{\min}$ by *sup*; – Case 6
(3) **else**
  use the old shift strategy.
(4) **end if**

As in DLASQ, $d_{\min} < 0$ is used to flag a new segment after splitting [16]. See section 6.3 in [16] for the specific definitions of different cases.

5. Finite step convergence property. In this section we justify our claim of linear worst case complexity of our improved algorithm. This property has allowed us to implement our algorithm without fear of reaching an iteration limit before convergence occurs.

Algorithm 7 shows the prototype of our algorithm. The main differences with DLASQ are that Algorithm 7 updates the upper bound *sup*, while DLASQ does not, and that Algorithm 7 uses the d-deflation strategy while, DLASQ does not. For simplicity, other techniques are not included in Algorithm 7 such as *splitting* and *flipping*; see [16] for details. One step of the DQDS transform which either succeeds or fails is called one *DQDS iteration*, or just one *iteration*.

Our claim is as follows. For an $n \times n$ bidiagonal matrix B with positive elements, Algorithm 7 can compute each singular value in about

$$\Upsilon = \lceil \log_{1/\beta}(n/\epsilon) \rceil \tag{5.1}$$

iterations, where $\beta = \max(\alpha, 1-\alpha)$ and ϵ is the machine precision, and in total it requires about $O(n\Upsilon)$ iterations to find all singular values.

We first do one DQD transform on matrix B and then initialize $sup = d_{\min}$. By Corollaries 2.2 and 2.4, we know $sup \le n\sigma^2_{\min}(B)$. Denote the upper bound after the kth DQDS iteration by $sup^{(k)}$. Since $sup^{(k)}$ is used as a guide to choosing shift and each iteration would make it smaller at least by $(1-\beta)\cdot sup^{(k)}$, the upper bound after $k+1$ iterations satisfies

$$sup^{(k+1)} \le \beta sup^{(k)} \le \beta^k n\sigma^2_{\min}(B),$$

where $\beta = \max(\alpha, 1-\alpha)$ and α is defined as in Algorithm 7.

Thus, for $k \ge \log_{1/\beta}(n/\epsilon)$, we would have

$$sup^{(k+1)} \le \beta^k n\sigma^2_{\min}(B) \le \epsilon\sigma^2_{\min}(B).$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

If $k \geq \Upsilon$ and matrix B is still not deflated by classical strategies, by Corollaries 2.2 and 2.4, B must be deflated by the d-deflation strategy in section 4.1. Therefore, Algorithm 7 requires no more than Υ iterations per singular value.

ALGORITHM 7. [Prototype of our algorithm]

```
Choose a parameter α ∈ (0  1);
while Z unfinished do
    (1) deflate array Z
        if Z's final entries are negligible then
            reduce Z accordingly;
        else
            if  d_min is small enough, use d-deflation strategy;
        end if
    (2) choose a shift
        if sup is small enough, then
            choose s = 0;
        else
            s = α· sup;
        end if
    (3) apply a DQDS transform to Z
        update sup;
        if DQDS fails, go to step (2);
end while
```

If $\alpha = \frac{3}{4}$, $\epsilon = 10^{-16}$, and $n = 1000$, the algorithm above requires no more than 152 iterations to compute each singular value. Equation (5.1) gives a guide to setting the average iteration number required for all singular values. It is set to 30 in DLASQ (LAPACK-3.4.0), which is too small, and DLASQ may fail for the difficult matrices in section 6. Equation (5.1) suggests that a number around 100 is a better choice. In the absolute majority of cases, our algorithm converges very fast, in the range of about 10 iterations per singular value. However, there are some bidiagonal matrices for which it indeed requires $O(\Upsilon)$ iterations to find the smallest singular value, e.g., the difficult matrices in section 6. After the smallest singular value is found, convergence is very quick. The worst case iteration bound is very similar to that in the zero-in algorithm and to its recent variant for finding zeros of a general univariate nonlinear equation (see [2, 18]).

Just as in DLASQ [16], in practice we can gradually increase α, if previous shifts succeed, and decrease α if previous shifts have failed consecutively three times.

6. Numerical results. We have implemented our algorithm in Fortran 77 by incorporating our new deflation and new shift strategies into the DLASQ routines. All experiments were performed on a laptop with 4G memory and Intel Core i7-2640M CPU. For compilation we used the `gfortran` compiler and the optimization flag `-O3` and linked the codes to optimized BLAS and LAPACK, which are obtained by using ATLAS [17].

6.1. Comparing each new technique. Recently, we have been able to construct a number of bidiagonal matrices of different dimensions for which DLASQ converges so slowly that it requires more iterations than allowed in the code. In these matrices, the diagonals are highly disordered and both the diagonal and off-diagonal

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

TABLE 1
The difficult bidiagonal matrices.

	n	Description of the bidiagonal matrix B	Source
1	544	Matrix_1	[10]
2	1000	Matrix_2	[10]
3	1087	Matrix_3	[10]
4	1088	Matrix_4	[10]
5	5000	random bidiagonal matrix	

TABLE 2
The speedup of Vi over DLASQ.

Matrix	n	DLASQ	V1	V2	V3	V4	V5
1	544	1.00	1.62	1.75	2.14	2.10	2.83
2	1000	1.00	1.76	1.76	1.80	2.06	2.74
3	1087	1.00	4.98	5.20	7.88	7.88	9.99
4	1088	1.00	1.79	2.14	2.41	2.52	3.31
5	5000	1.00	1.04	1.04	1.07	1.14	1.20

TABLE 3
The average iterations for each singular value.

Matrix	1	2	3	4	5
DLASQ	45.4	26.0	74.8	30.7	10.5
V5	11.81	7.19	7.62	8.85	7.78

entries are of massively varying orders of magnitude. Figure 1(a) shows the values of the diagonal elements of one such matrix, where the x-axis denotes the index of the diagonal elements and the y-axis denotes their values. The plot of the first 800 diagonal elements goes up and down, while the last 200 diagonal elements are nearly equal to one.

In this section, we use these difficult matrices and a random bidiagonal matrix whose entries are from Gaussian distribution to show the improvements of our modifications. There are five main modifications over DLASQ. Consequently, we have five versions of improved algorithms by adding the new techniques one by one. We use the following notation to denote them:

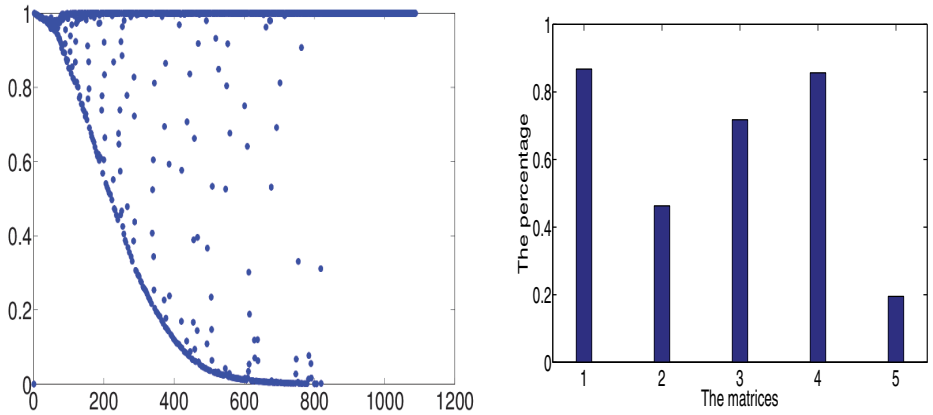
- V1: d-deflation strategy in section 4.1,
- V2: new deflation strategies in section 4.2,
- V3: updating the upper bound,
- V4: twisted shift in last 20 rows,
- V5: using suitable 2-by-2 submatrix.

Vi denotes a version after adding a new technique to the previous version. Thus V5 is the best improved algorithm. Note that the technique *setting negligible* $d_{\min}$ *to zero* has been implemented in LAPACK-3.4.1,[2] and V1 performs similarly to the DLASQ in LAPACK-3.4.1. Table 2 shows the speedups of V5 over DLASQ in terms of time. The average iterations of V5 for each singular value are shown in Table 3. From the tables we can see that V5 requires far fewer iterations and is about 3x–10x faster than DLASQ for these difficult matrices.

This superior performance of V5 is largely due to the d-deflation strategy in section 4.1, which is almost "tailor-made" for such difficult matrices. It can deflate

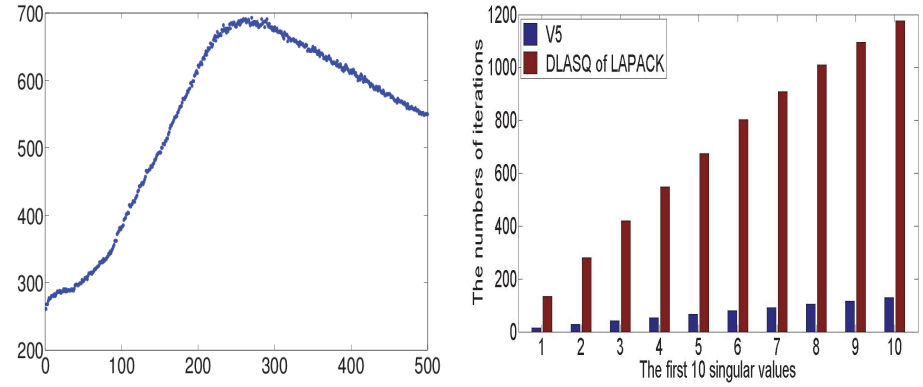
[2]This technique is added to LAPACK-3.4.0 by Demmel, Kahan, and Lipshitz.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) The values of diagonal elements of Matrix_3. (b) The percentage of singular values deflated by d-deflation strategy in section 4.1.

FIG. 1. *Results for these difficult matrices.*



(a) The locations of $d_{\min}$ when finding the first 500 singular values. (b) The number of iterations for the first 10 singular values.

FIG. 2. *More results for Matrix_3 in Table* 1.

40%–80% of the singular values of such matrices (see Figure 1(b)). It is also very interesting to note that it usually can deflate about 20% of the singular values of a general bidiagonal matrix.

For the Matrix_3, Figure 2(a) shows the locations of negligible $d_{\min}$ when finding the first 500 singular values, which are all deflated by the d-deflation strategy. For this matrix, V5 achieves about 10x speedup. We compare the number of iterations when finding the first 10 singular values of Matrix_3. The results are shown in Figure 2(b). For the first 10 singular values, DLASQ takes 120 iterations on average.

The last four improvements aim to accelerate the convergence for general bidiagonal matrices. From Table 2, we can see that each of these techniques improves the performance. Since different matrices may have very different properties, we cannot expect these techniques to help a lot for all matrices. We further use all the matrices in `stetester` [11] to test each improvement by adding them one by one. Each

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

TABLE 4
More bidiagonal matrices.

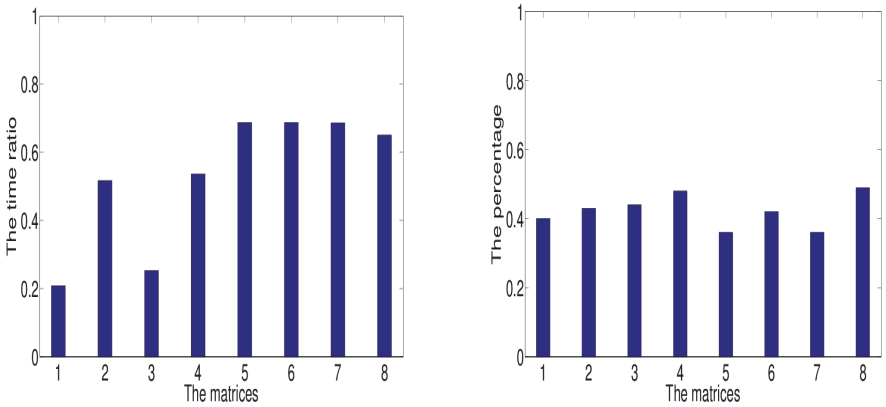
	n	Description of the bidiagonal matrix B	Source
1	15,005	Cholesky factor of Glued Wilkinson matrix	[11]
2	15,005	Cholesky factor of Glued Clement matrix	[11]
3	30,010	Cholesky factor of Glued Wilkinson matrix	[11]
4	30,010	Cholesky factor of Glued Clement matrix	[11]
5	2901	Cholesky factor of T_nasa2910_1.dat	[10]
6	3258	Cholesky factor of T_bcsstkm10_3.dat	[10]
7	4098	Cholesky factor of T_sts4098_1.dat	[10]
8	5472	Cholesky factor of T_nasa1824_3.dat	[10]

technique is helpful for most of these matrices. Due to space limitations, we do not include all these results.

6.2. More tests. To further show the improvement of V5, we use more matrices to test our algorithm. These matrices can be divided into two classes: (1) matrices from applications, and (2) matrices constructed for testing the DQDS algorithm.

These matrices are illustrated in Table 4. The last four are from industrial applications which are collected in LAPACK tester `stetester` and can be obtained from a website maintained by Marques [10]. To measure the computing time accurately, we intentionally choose some big matrices.

The results are shown in Figures 3(a) and 3(b). Figure 3(a) shows the ratios over DLASQ in terms of time. For these matrices from industrial applications, V5 saves about 30% in time. For these glued matrices, which are difficult for the DQDS algorithm, V5 has even more speedups, being about 2x–5x faster. Figure 3(b) provides more information. For these matrices, d-deflation strategy deflates about 40% of all singular values. Figure 3(b) further supports our conclusion that the d-deflation strategy greatly improves the performance of the DQDS algorithm.



(a) The time ratio of V5 over DLASQ.

(b) The percentage of singular values deflated by d-deflation strategy in section 4.1.

FIG. 3. *More results for matrices from construction and industry.*

For these 254 matrices in `stetester` for which DLASQ needs more than $1.5e$–2 seconds, V5 usually saved about 20% in time and was slower than DLASQ only for 32 of them but never slower than DLASQ by more than 0.0145 seconds. Among these

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

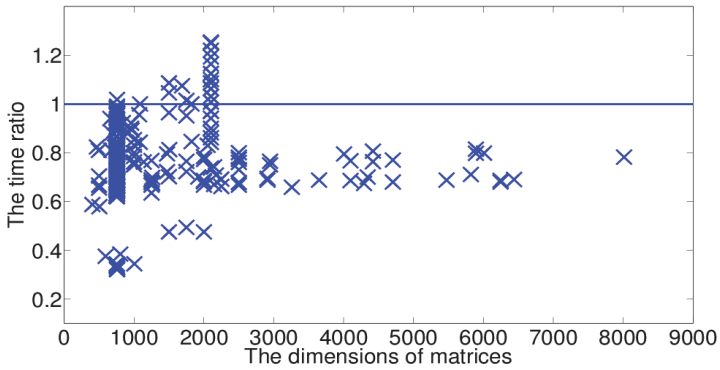
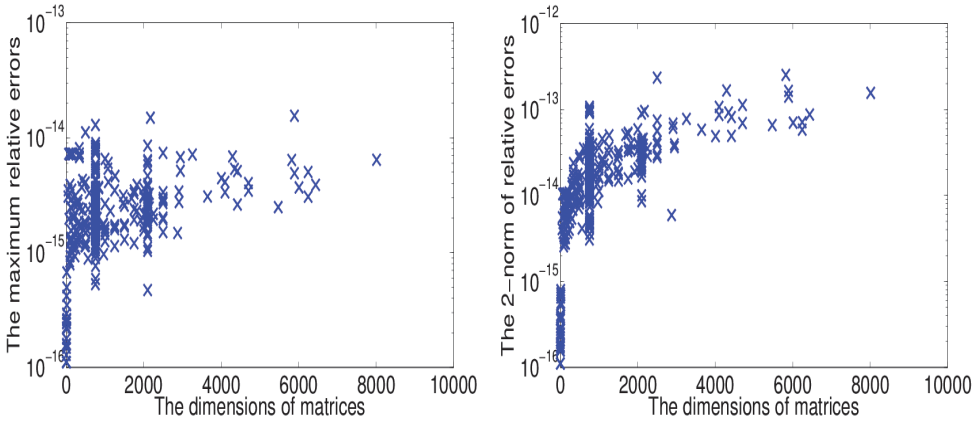


FIG. 4. *The time of V5 versus DLASQ for matrices in LAPACK tester.*



(a) Maximum relative error. (b) 2-norm error.

FIG. 5. *The comparison of accuracy with bisection.*

254 matrices, V5 was 3x faster for 4 of them, 2x faster for 14 of them, 1.5x faster for 47 of them, and 1.2x faster for 174 of them. The results are shown in Figure 4.

6.3. The test of accuracy. To show the accuracy of the singular values computed by our algorithm, we compared the singular values obtained by our algorithm with those obtained from a bisection algorithm. We first change the singular value problem of a bidiagonal matrix into the eigenvalue problem of a tridiagonal matrix of double size with zero diagonals, and then use a bisection algorithm to find its eigenvalues. We assume the eigenvalues computed by the bisection algorithm are "correct." The results of maximum relative error are shown in Figure 5(a). Figure 5(b) shows the 2-norm of the relative errors.

6.4. Comparisons with aggressive early deflation. The aggressive early deflation strategy [12] is designed to enhance the DQDS algorithm. This deflation strategy can help a lot for matrices which are easy for DQDS. In this subsection, we demonstrate that AED does not help for those difficult matrices in Table 1. There are two versions of AED in [12], denoted by AggDef1 and AggDef2; see [12] for details.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 5
Comparison of V5 with AggDef2 [12] for matrices in Table 1.

	Methods	Matrix_1	Matrix_2	Matrix_3	Matrix_4	Matrix_5
n		544	1000	1087	1088	5000
Speedup $\frac{\text{Time(DLASQ)}}{\text{Time(method)}}$	AggDef2	1.06	1.05	1.01	1.08	1.02
	V5	2.83	2.74	9.99	3.31	1.20
Max. Rel. Error	DLASQ	6.22e-15	9.54e-15	4.07e-14	2.42e-14	9.47e-15
	AggDef2	6.66e-15	9.33e-15	3.97e-14	2.35e-14	9.52e-15
	V5	3.66e-15	7.99e-15	3.85e-15	5.66e-15	6.27e-15

TABLE 6
Some bidiagonal matrices for testing HDLASQ.

Matrix	n	Description of the bidiagonal matrix B	Source
Mat1	30000	$\sqrt{q_i} = n + 1 - i, \sqrt{e_i} = 1$	[12]
Mat2	30000	$\sqrt{q_i} = n + 1 - i, \sqrt{e_i} = \sqrt{q_i}/5$	[12]
Mat3	30000	Toeplitz: $\sqrt{q_i} = 1$, $\sqrt{e_i} = 2$	[11, 12]
Mat4	30000	Cholesky factor of tridiagonal (1, 2, 1) matrix	[11, 4]

Since AggDef2 is more efficient than AggDef1, we only compare our algorithms with AggDef2.

When comparing the accuracy, we also assume that the results by the bisection method are correct. From the results in Table 5, we can see that for these difficult matrices, AggDef2 has nearly the same performance as DLASQ. Our improved algorithm (V5) is the most accurate and fastest among these three versions.

By combining our improvements with AggDef2, we obtain a hybrid algorithm, HDLASQ, which is summarized in Algorithm 8 and is similar to the algorithm in [12].

ALGORITHM 8. [HDLASQ]
Inputs: bidiagonal matrix $B \in R^{n\times n}$, deflation frequency p

1. **while** size of current segment B is larger than $\sqrt{n}$ **do**
2. run p iterations of DQDS by calling V5;
3. perform aggressive early deflation;
4. **end while**
5. run DQDS until all singular values are computed.

As mentioned in [12], setting p too large may deteriorate the rate of convergence, but the performance does not much depend on p. For example, HDLASQ with $p = 200$ was about 10% slower than letting $p = 20$ for matrices in Table 6 (except Mat1, for which AggDef2 is particularly effective). In our implementation,[3] we let $p = 50$.

For these disordered matrices in Table 1, HDLASQ has nearly the same performance as V5. For the matrices that are good for AED (shown in Table 6), the comparison results of HDLASQ with other methods are shown in Table 7, from which we can see that HDLASQ is faster than AggDef2, DLASQ, and V5. Furthermore, Figure 6 shows the time ratio of HDLASQ over DLASQ for matrices in `stetester` for which DLASQ needs more than $1.5e{-}2$ seconds. By comparing Figure 6 with Figure 4, we can also see that HDLASQ is usually better than V5.

[3]We used the codes of AggDef, available at http://www.opt.mist.i.u-tokyo.ac.jp/~nakatsukasa.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

TABLE 7
The comparisons of HDLASQ with AggDef2 and V5 for matrices in Table 6.

	Methods	Mat1	Mat2	Mat3	Mat4
n		30000	30000	30000	30000
Speedup $\frac{\text{Time(DLASQ)}}{\text{Time(method)}}$	V5	1.66	1.48	1.39	1.40
	AggDef2	76.1	1.99	1.57	1.59
	HDLASQ	79.5	2.18	1.67	1.69

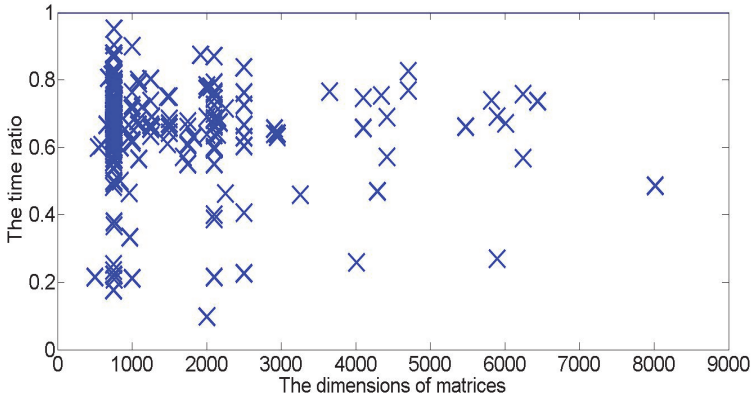


FIG. 6. *The time of HDLASQ versus DLASQ for matrices in LAPACK tester.*

7. Conclusions. In this paper we first propose a novel deflation strategy for the DQDS algorithm, named *d-deflation strategy*, which is different from the classic deflation strategies and greatly improves the DQDS algorithm. Note that the shifts for the DQDS algorithm must be a fraction of the upper bound *sup*. Together with the technique of *updating the upper bound*, they ensure the linear worst case complexity of our improved DQDS algorithm V5. Additional modifications to certain shift strategies are also included. These improvements together make V5 up to 10x faster for those difficult matrices and 1.2x–4x faster in general without any loss of accuracy. By combining V5 with AED, we propose a hybrid algorithm (HDLASQ) which is shown to be faster than DLASQ, V5, and AggDef2.

Acknowledgments. The authors are very grateful to the anonymous referees and the associated editor Chen Greif for their valuable suggestions, and would like to acknowledge many helpful discussions with J. Demmel, W. Kahan, and B. Lipshitz. The authors also thank O. Marques for making his collection of difficult bidiagonal matrices available for numerical experiments.

REFERENCES

[1] K. AISHIMA, T. MATSUO, K. MUROTA, AND M. SUGIHARA, *On convergence of the DQDS algorithm for singular value computation*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 522–537.
[2] R. P. BRENT, *Algorithms for Minimization without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ, 1972.
[3] J. W. DEMMEL AND W. KAHAN, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Comput., 11 (1990), pp. 873–912.
[4] K. V. FERNANDO AND B. N. PARLETT, *Accurate singular values and differential qd algorithms*, Numer. Math., 67 (1994), pp. 191–229.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[5] C. Ferreiva, *The Unsymmetric Tridiagonal Eigenvalue Problem*, Ph.D. thesis, University of Minho, Portugal, 2007.
[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[7] R. Horn and C. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991.
[8] W. Kahan, *Personal communication*, 2012.
[9] S. G. Li, M. Gu, and B. N. Parlett, *Supplementary materials to "An improved DQDS algorithm,"* 2014. Available online at http://sisc.siam.org/sisc_files/2014/03/01/00045458/04/45458_4_supp_1_n1srgw_sc.pdf.
[10] O. Marques, *Osni Marques's Homepage*, http://crd-legacy.lbl.gov/∼osni/.
[11] O. A. Marques, C. Vömel, J. W. Demmel, and B. N. Parlett, *Algorithm* 880: *A testing infrastructure for symmetric tridiagonal eigensolvers*, ACM Trans. Math. Software, 35 (2009), 13.
[12] Y. Nakatsukasa, K. Aishima, and I. Yamazaki, *dqds with aggressive early deflation*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 22–51.
[13] B. N. Parlett, *The new QD algorithms*, Acta Numer., 4 (1995), pp. 459–491.
[14] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Classics Appl. Math. 20, SIAM, Philadelphia, 1998.
[15] B. N. Parlett and M. Gutknecht, *From QD to LR, or how were the QD and LR algorithms discovered?*, IMA J. Numer. Anal., 31 (2011), pp. 741–754.
[16] B. N. Parlett and O. Marques, *An implementation of the DQDS algorithm (positive case)*, Linear Algebra Appl., 309 (2000), pp. 217–259.
[17] R. C. Whaley, A. Petitet, and J. J. Dongarra, *Automated empirical optimizations of software and the ATLAS project*, Parallel Comput., 27 (2001), pp. 3–35.
[18] G. Wilkins and M. Gu, *A modified Brent's method for finding zeros of functions*, Numer. Math., 123 (2012), pp. 177–188.

Downloaded 08/19/14 to 218.249.35.58. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.