LAPACK WORKING NOTE 166: COMPUTING THE BIDIAGONAL SVD USING MULTIPLE RELATIVELY ROBUST REPRESENTATIONS

PAUL R. WILLEMS[†‡], BRUNO LANG[‡], AND CHRISTOF VÖMEL[§]

Technical Report UCB//CSD-05-1376
Computer Science Division
UC Berkeley

Abstract. We describe the design and implementation of a new algorithm for computing the singular value decomposition of a real bidiagonal matrix. This algorithm uses ideas developed by Großer and Lang that extend Parlett's and Dhillon's MRRR algorithm for the tridiagonal symmetric eigenproblem. One key feature of our new implementation is, that k singular triplets can be computed using only $\mathcal{O}(nk)$ storage units and floating point operations, where n is the dimension of the matrix. The algorithm will be made available as routine `xBDSCR` in the upcoming new release of the LAPACK library.

Key words. Bidiagonal Singular Value Decomposition, Tridiagonal Symmetric Eigenproblem, MRRR algorithm, Coupling Relations, LAPACK library

AMS subject classifications. 15A18, 65-04, 65F15

1. Introduction.

Starting in the mid 90s, Dhillon and Parlett developed the algorithm of Multiple Relatively Robust Representations (MRRR) that computes k numerically orthogonal eigenvectors of a symmetric tridiagonal matrix $T \in \mathbb{R}^{n\times n}$ with $\mathcal{O}(nk)$ cost [7, 8, 15, 17, 18]. This algorithm has subsequently been extended by Großer and Lang using so-called *coupling relations* [10–12] for the stable computation of the bidiagonal Singular Value Decomposition (bSVD). Due to recent improvements for the tridiagonal MRRR algorithm (see, e.g. [9, 20, 21]) as well as for the coupling technique itself, the references [11, 12] no longer describe the most efficient implementation of the bidiagonal MRRR algorithm. The present paper focuses on these recent developments and our resulting new implementation, which is to be incorporated as routine `xBDSCR` into the next release of the widely used LAPACK library [1].

Throughout this article, we have tried to present the MRRR algorithm and its adaption to the bSVD via coupling relations in such a way that readers without prior expertise in this area should be able to follow the arguments and understand the inner workings of the algorithms in an intuitive way. That is, we will present the topics in enough detail but without too much theory, giving all needed references to update the latter on the way, if desired.

First we will recall some basic concepts and fix our notation. For a bidiagonal matrix $B \in \mathbb{R}^{n\times n}$, the problem bSVD consist of finding orthogonal matrices U and V and a diagonal matrix Σ such that

$$B = U\Sigma V^T. \tag{1.1}$$

[†]Central Institute for Applied Mathematics, Research Centre Jülich, Germany
[‡]Applied Computer Science and Scientific Computing, University of Wuppertal, Germany, E-mail: `{willems,lang}@math.uni-wuppertal.de`
[§]Computer Science Division, University of California, Berkeley, CA 94720, USA. E-mail: `voemel@eecs.berkeley.edu`. The work of this author has been supported by a grant from the National Science Foundation (Cooperative Agreement no. ACI-9619020). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

We will follow the convention that Σ contains the singular values $\sigma_1, \ldots, \sigma_n$ of B in descending order. The columns u_i of U and v_i of V are called the left and right singular vectors, respectively, of B. For the scope of this paper we will furthermore assume B to be upper bidiagonal, as it differs from the lower bidiagonal case only concerning the roles of U and V.

Any algorithm solving the bSVD should guarantee small deviations from orthogonality for the matrices U and V

$$||U^T U - I|| = \mathcal{O}(n\epsilon), \quad ||V^T V - I|| = \mathcal{O}(n\epsilon), \tag{1.2}$$

along with small residuals

$$||BV - U\Sigma|| = \mathcal{O}(n\epsilon||B||), \tag{1.3}$$

where ϵ denotes the machine precision. In addition, for some applications (but not all) it is required that the computed singular values approximate the exact ones to high relative accuracy, so our algorithm should be able to deliver this if requested.

The problem bSVD can be reduced to the tridiagonal symmetric eigenproblem (tSEP) in two ways, using three different matrices. One way works with the *normal equations* of B to compute

$$B^T B = V\Sigma^2 V^T, \quad BB^T = U\Sigma^2 U^T. \tag{1.4}$$

Alternatively, one can use the Jordan-Wielandt form of B to compute U and V simultaneously via

$$\begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} = Q \begin{bmatrix} -\Sigma & 0 \\ 0 & \Sigma \end{bmatrix} Q^T, \text{ where } Q = \frac{1}{\sqrt{2}} \begin{bmatrix} U & U \\ -V & V \end{bmatrix}. \tag{1.5}$$

Note that (1.5) can be permuted to be symmetric tridiagonal, resulting in the so-called *Golub-Kahan matrix*.

A very efficient method for tSEP is the MRRR algorithm by Dhillon and Parlett, which we will describe in Section 2. In practice, the straight-forward use of MRRR to solve either (1.4) or (1.5) does not necessarily imply that both (1.2) and (1.3) hold. This has been observed by Großer and Lang [10–12] and we will explain their results in Section 3. They proposed a remedy to this problem using so-called *coupling relations* to adapt the MRRR algorithm for the bSVD.

We have found that in order to get an efficient and robust computer implementation of this method, the strategy presented in [11, 12] could be improved. This is our main contribution and described together with the coupling relations in Section 4. Finally, in Section 5, we describe additional important issues relevant for high performance, and we will compare our implementation with the Divide&Conquer and QR routines from LAPACK [1].

Some words concerning notation. The symbols already introduced above will remain fixed throughout this article. For example, B will always denote the upper bidiagonal matrix of dimension n for which we want to compute its bSVD. We will refer to the diagonal elements of B as a_i and to its offdiagonal elements as b_i. Besides this, we will deal extensively with diagonal matrices D, R, and unit lower or upper bidiagonal matrices L, U. These matrices have only a linear number of nontrivial entries each, which we will denote using a single index as d_i, r_i, l_i, u_i, respectively. It will be convenient to define for a given m-vector x and $k \in \mathbb{N}$, $\mathrm{diag}(x, k)$ as square matrix of order $m + |k|$ with its k'th offdiagonal being x and all other entries set to zero. As an example using this notation, we could write B alternatively as

$$B = \mathrm{diag}([a_1, \dots, a_n], 0) + \mathrm{diag}([b_1, \dots, b_{n-1}], +1).$$

One possible point of confusion we are aware of is the fact that we use U both for the left singular vectors as well as for some upper unit bidiagonal matrix, and u_i for the columns of U as well as for offdiagonal elements. We did not want to break with one of these uses, as they are part of an evolved and wide-spread standard, and the meaning will always be clear from the context. Additional notation will be specified where needed.

2. The MRRR Algorithm. A quite new method for the tridiagonal symmetric eigenproblem is the MRRR algorithm by Dhillon and Parlett (MRRR stands for "Multiple Relatively Robust Representations"). For brevity, we will sometimes call it simply "MRRR".

For a symmetric tridiagonal matrix $T \in \mathbb{R}^{n \times n}$, the MRRR algorithm is able to compute k eigenpairs (λ_i, q_i) in optimal $\mathcal{O}(kn)$ time, at the same time guaranteeing small residual norms

$$||Tq_i - \lambda_i q_i|| = \mathcal{O}(n\epsilon||T||) \tag{2.1}$$

and numerically orthogonal eigenvectors with

$$|q_i^T q_j| = \mathcal{O}(n\epsilon), \;\; j \neq i. \tag{2.2}$$

For these reasons, together with the fact the MRRR is well suited for parallelization, the MRRR algorithm is sometimes inofficially also dubbed "holy grail".

In this section, we want to give a short overview of the basic principles underlying MRRR, as this will be necessary for the remainder of the paper. We will omit most of the theory, but give enough information such that readers without prior knowledge of the algorithm should be able to get an intuitive understanding. For a more detailed description see [6, 8, 18].

Since LAPACK 3.0, the algorithm has been included as routine `xSTEGR`. There have been many recent improvements of this implementation, especially for the support of partial spectra and better robustness (see for example [9]).

In order to describe the MRRR algorithm, we first need to establish the concept of relative distances between eigenvalues, which is defined in slightly varying ways in the literature. In accordance with [8] we will use

$$\mathrm{reldist}(\lambda, \mu) := \frac{|\lambda - \mu|}{|\lambda|}. \tag{2.3}$$

Then the *relative gap* of an eigenvalue is defined as

$$\mathrm{relgap}(\lambda) := \min \left\{ \mathrm{reldist}(\lambda, \mu) \mid \lambda \neq \mu \in \mathit{spec}(T) \right\}. \tag{2.4}$$

An eigenvalue is (relatively) *isolated* or a *singleton*, if its relative gap exceeds some threshold (for example 10^{-3}). A group $\lambda_{c:d}$ of successive non-singleton eigenvalues $\lambda_c, \lambda_{c+1}, \dots, \lambda_d$ is called a *cluster*.

From a distant point of view, the MRRR algorithm can be seen as a sophisticated variant of inverse iteration without need for explicit reorthogonalization. A closer perspective reveals two simple but elegant ingredients responsible for its immense success:

1. A method based on so-called *twisted factorizations* to compute, for a *relatively* isolated eigenvalue λ, in $\mathcal{O}(n)$ work an eigenvector $\bar{q}$ satisfying

$$|\sin\angle(q,\bar{q})| = \mathcal{O}\left(n\epsilon/\mathrm{relgap}(\lambda)\right), \tag{2.5}$$

where q denotes the true eigenvector. We will describe this technique in more detail shortly; let it for now suffice that twisted factorizations are a generalization of the standard LDL^T and URU^T bidiagonal factorizations.

2. Eigenvectors are shift-invariant, but the relative distances of eigenvalues are not. More precisely, if a shift $\mu \approx \lambda$ close to an eigenvalue is chosen, the relative gap of $\lambda' = \lambda - \mu$ with respect to $T' := T - \mu I$ becomes

$$\mathrm{relgap}_{T'}(\lambda') = \mathrm{relgap}_T(\lambda)\frac{|\lambda|}{|\lambda-\mu|} \gg \mathrm{relgap}_T(\lambda).$$

With these two ideas, the obviuos approach is to repeatedly shift the matrix until an eigenvalue is relatively isolated and the corresponding eigenvector can be computed using twisted factorizations.

As we will see shortly, in order to make this strategy work, it is necessary that each encountered shifted matrix defines its eigenvalues and eigenvectors to high relative accuracy. To this end, the MRRR algorithm employs the concept of *Relatively Robust Representations (RRRs)* of a matrix. Any set of numbers defining a matrix is called a representation of the matrix. A representation is relatively robust, if small relative changes in these numbers only cause small relative changes in the eigenvalues and eigenvectors. If this holds only for some eigenpairs, the representation is called a *partial RRR*.

It is an interesting fact that most tridiagonal matrices represented directly by their diagonal and offdiagonal elements do not have this property, but a representation based on a bidiagonal factorization of the matrix usually does (see [6] for more details). Therefore the algorithm does not work directly on tridiagonal matrices, but on LDL^T factorizations of these matrices instead, such that the data (L, D) forms an RRR.

Armed with these concepts and ideas, the MRRR algorithm can now informally be described as follows. First, an RRR (L, D) is found for the original matrix T, possibly by shifting T. Then the eigenvalues of interest are approximated accurately enough to categorize them into singletons and clusters (for example using bisection or the dqds algorithm [15, 19]). For each singleton, the eigenvector can be computed directly using twisted factorizations. Because the relative gap of a singleton is per definitionem large enough, this leads to excellent results according to (2.5).

If there is a cluster of eigenvalues, the algorithm chooses a shift τ close to the cluster, s.t. $LDL^T - \tau I =: L^+D^+(L^+)^T$ and (L^+, D^+) does again form a partial RRR for the eigenvalues in the cluster. This factorization is computed using the stationary *differential qds algorithm* (dstqds) [6, 15], as shown in Algorithm 2.1. This transformation is carefully designed to allow a mixed relative error analysis, that is, tiny relative changes to the input (L, D) and the output (L^+, D^+) give an exact relation. Note that this property, together with the fact that (L, D) and (L^+, D^+) are ensured to be (partial) RRRs, are essential to guarantee that the shifting process does not spoil the relation between the eigenpairs of $L^+D^+(L^+)^T$ and of LDL^T. This allows to treat $L^+D^+(L^+)^T$ recursively in the same fashion.

Algorithm 2.1 Factorize $LDL^T - \tau I = L^+D^+(L^+)^T = U^+R^+(U^+)^T$ using the differential stationary (left side) and progressive (right side) qds-transformations.

DSTQDS	DPQDS
Input: L, D, τ	**Input:** L, D, τ
Output: L^+, D^+, S^+	**Output:** U^+, R^+, P^+
$s_1^+ = -\tau$	$p_n^+ = d_n - \tau$
for $i = 1 : n-1$ **do**	**for** $i = n-1 : -1 : 1$ **do**
$\quad d_i^+ = d_i + s_i^+$	$\quad r_{i+1}^+ = d_i l_i^2 + p_{i+1}^+$
$\quad l_i^+ = d_i l_i / d_i^+$	$\quad u_i^+ = l_i d_i / r_{i+1}^+$
$\quad s_{i+1} = l_i^+ l_i s_i^+ - \tau$	$\quad p_i^+ = p_{i+1}^+ d_i / r_{i+1}^+ - \tau$
endfor	**endfor**
$d_n^+ = d_n + s_n^+$	$r_1^+ = p_1^+$

A convenient way to describe the resulting flow of computation is as traversal of a *representation tree*. A node in this tree is given by an index range of eigenvalues, a partial RRR for these eigenvalues and the accumulated shift from the root. Leaf nodes have only one index, otherwise each index of a node is contained in exactly one of the index ranges of the node's children.

Twisted Factorizations. We will finish our description of the MRRR algorithm giving a more detailed explanation of the method used to compute highly accurate eigenvectors with orthogonality levels inversely proportional to the *relative* gaps of the eigenvalues.

Given an RRR (L, D), we can (for example using bisection) compute an approximation $\bar\lambda$ to an eigenvalue λ of LDL^T satisfying

$$|\lambda - \bar\lambda| = \mathcal{O}(\epsilon|\lambda|). \tag{2.6}$$

Then the idea is to find a vector $\bar q$ with a small *relative* residual

$$||(LDL^T - \bar\lambda I)\bar q|| = \mathcal{O}(n\epsilon|\bar\lambda|). \tag{2.7}$$

The reward is revealed by the classical gap theorem [2, 14], because if q denotes the true eigenvector, we get the desired result (2.5).

In order to ensure (2.7), a double factorization

$$LDL^T - \bar\lambda I = L^+D^+(L^+)^T = U^+R^+(U^+)^T \tag{2.8}$$

is computed using the stationary and progressive differential qds transformations shown in Algorithm 2.1. If one or both of these factorizations does not exist, the following method can be easily modified; see for example [8]. Assuming for now that they do exist, this opens n possible ways to compute an approximation $q^{(k)}, 1 \le k \le n$, to the eigenvector q via

$$\begin{aligned} q_k^{(k)} &= 1, \\ q_i^{(k)} &= -l_i^+ q_{i+1}^{(k)}, \quad i = k-1, \dots, 1, \\ q_{i+1}^{(k)} &= -u_i^+ q_i^{(k)}, \quad i = k, \dots, n-1. \end{aligned} \tag{2.9}$$

Formally, applying (2.9) is equivalent to solving the system

$$N_k G_k (N_k)^T q^{(k)} = \gamma_k e_k, \quad q_k^{(k)} = 1, \tag{2.10}$$

where $N_k G_k N_k^T = LDL^T - \bar{\lambda} I$, $G_k = \operatorname{diag}(d_1, \ldots, d_{k-1}, \gamma_k, r_{k+1}, \ldots, r_n)$ and N_k is a tridiagonal matrix with

$$(N_k)_{1:k,1:k} = L^+_{1:k,1:k} \text{ and } (N_k)_{k:n,k:n} = U^+_{k:n,k:n}.$$

The matrix $N_k G_k N_k^T$ is called a *twisted factorization* of $LDL^T - \bar{\lambda} I$ (also alled BABE-factorization for "Burn At Both Ends") and k the *twist index*, because it can be obtained by applying the Gaussian elimination process from top to row k and then in backward direction from bottom to row k.

The remaining question now is, which k is best. As according to (2.10) the residual for $q^{(k)}$ is $\gamma_k e_k$, any twist index k minimizing $|\gamma_k|$ is an obvious candidate. Comparing $(N_k G_k N_k^T)_{k,k}$ with $(LDL^T - \bar{\lambda} I)_{k,k}$ gives

$$\gamma_k = d_k^+ + r_k^+ - \left((LDL^T)_{k,k} - \bar{\lambda}\right), \quad k = 1 : n.$$

A more stable way to compute γ_k is

$$\gamma_k = s_k + p_k + \tau,$$

using the intermediate quanities s_k^+ and p_k^+ of Algorithm 2.1 for the factorizations (2.8).

It is shown in [6] that if k is chosen such that $|\gamma_k|$ is minimized (or small enough), the resulting vector $q^{(k)}$ will indeed fulfill (2.7) and therefore also (2.5).

Another way to understand the above method is as a variant of inverse iteration. A special and interesting property of the twist indices is that if $\bar{\lambda}$ approximates λ to high relative accuracy, i.e., if (2.6) is fulfilled, the index k minimizing $|\gamma_k|$ will correspond to the component of the eigenvector with largest absolute value. (Actually, this is only true in the limit case $\bar{\lambda} \to \lambda$; see [6] for details). Therefore, e_k is guaranteed to be an excellent choice as starting vector. The simplicity of this right hand side then allows to compute the first iterate using only multiplications in (2.9). Recent developments of this technique by Parlett and Vömel in [21] even allow more steps of inverse iteration with twisted factorizations, again using only multiplications to avoid spoiling the relative accuracy of the vectors.

3. The Black-Box Approach Fails. As already mentioned in Section 1, there are mainly two different ways of reducing the bSVD to the tSEP. The first approach employs the *normal-equations* and computes eigendecompositions

$$B^T B = V \Sigma^2 V^T \text{ and } BB^T = U \Sigma^2 U^T,$$

which together give us the desired bSVD $B = U \Sigma V^T$ of B.

Alternatively, one can use the so-called *Golub-Kahan* matrix T_{GK} of B, which is defined as

$$T_{GK} := P_{ps} \cdot \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} \cdot P_{ps}^T,$$

where P_{ps} is a "perfect shuffle" permutation mapping a vector $x \in \mathbb{R}^{2n}$ to $P_{ps} x = [x_{n+1}, x_1, x_{n+2}, x_2, \ldots, x_{2n}, x_n]^T$. It is easy to see that T_{GK} is symmetric tridiagonal with a zero diagonal and the entries of B interleaved on the offdiagonals, i.e.,

$$T_{GK} = \operatorname{diag}\left([a_1, b_1, a_2, \ldots, b_{n-1}, a_n], \pm 1\right).$$

Given an eigendecomposition $T_{GK} = Q\Lambda Q^T$, in exact arithmetic Q will have the structure

$$Q = \frac{1}{\sqrt{2}} \cdot P_{ps} \cdot \begin{bmatrix} U & U \\ -V & V \end{bmatrix},$$

so that the singular vectors of B can easily be extracted from the odd- and even-numbered rows of Q. Furthermore, the eigenvalues of T_{GK} are simply the positive and negative singular values of B. This has the additional benefit that if an adaptible method like the MRRR algorithm is used for the computation of Q and Λ, only n eigenpairs are needed, reducing the computation time by half.

It is tempting to use the MRRR algorithm as a "black box" on either the normal equations or the Golub-Kahan matrix to obtain the SVD of our bidiagonal matrix B. However, this leads to major numerical instabilities. In the following we will give only a short description of these problems as a motivation for the remainder of the paper. A more detailed analysis can be found in [11, 12].

Using the MRRR algorithm to compute eigendecompositions of B^TB and BB^T separately can result in bad residuals for the singular vectors, in the sense of (1.3). One reason is that MRRR uses different shifts when working on B^TB and BB^T, although in exact arithmetic the spectra of these matrices are identical. But the real source of failure turns out to be more subtle, as it lies in computing the tridiagonal factorizations. To be more concrete, let us assume that on the first level of the representation trees of B^TB and BB^T, the same shift μ is chosen for some cluster. Then, in order to proceed to the next level, the factorizations

$$\begin{aligned} B^TB - \mu I &= \hat{L}\hat{D}\hat{L}^T \\ BB^T - \mu I &= \check{L}\check{D}\check{L}^T \end{aligned}$$

are computed with the dstqds transformation in Algorithm 2.1. As already mentioned, these factorizations are carefully designed and highly accurate. Nevertheless, rounding errors do still occur, and they can cause the local eigenvalues $\hat{\lambda}_i$ of $\hat{L}\hat{D}\hat{L}^T$ and $\check{\lambda}_i$ of $\check{L}\check{D}\check{L}^T$ to be different. In fact, it has been shown in [12] that their *absolute* difference can be as large as

$$|\hat{\lambda}_i - \check{\lambda}_i| = \mathcal{O}(\sigma_i \epsilon). \tag{3.1}$$

For the purpose of computing well coupled singular vector pairs, this difference can be (and will be) devastating, especially for tight clusters of large singular values σ_i. Essentially, the explicit separate factorizations destroy most of the relationship between the computed vectors.

However, it should be mentioned at this point that the high absolute deviation (3.1) has no negative effect on the tridiagonal MRRR algorithm itself, which is why we still get nicely orthogonal matrices U and V. The large residuals $||Bv_i - u_i\sigma_i||$ come from the fact that for tight clusters, MRRR computes in some sense a random orthogonal basis, where the randomness is caused by the rounding errors during the factorizations. But as all representations are ensured to be RRR's and the cluster is tight, this does spoil neither the orthogonality levels $||U^TU - I||$ and $||V^TV - I||$, nor the "tridiagonal residuals" $||B^TBv_i - (\hat{\lambda}_i + \mu)v_i||$ and $||BB^Tu_i - (\check{\lambda}_i + \mu)u_i||$.

For the Golub-Kahan matrix the situation is reversed. The residuals (1.3) of the singular vectors are always excellent, but the deviation from orthogonality (1.2) can become bad for some bidiagonals with tight clusters of very small singular values.

The main reason lies in the fact that every minor of T_{GK} with an odd dimension is singular. This can make it more difficult for the algorithm to find RRR's for the next level. Additionally, we often observe quite differing element growth in the even-numbered and odd-numbered elements of the representations, which can intuitively cause a "displacement of information". This is problematic, as only one twist index is used to compute two singular vectors (via Q).

At this point we want to give an example of the impact of these problems on the orthogonality level of the vectors. As test matrix we took the upper bidiagonal

$$B := \operatorname{diag}\left([1, \alpha, \ldots, \alpha], 0\right) + \operatorname{diag}\left([\alpha, \ldots, \alpha], +1\right), \quad \alpha = 200\epsilon. \tag{3.2}$$

This matrix has one singular value around one and the rest clustered at 10^{-27}. We tested for different dimensions of B the newest tridiagonal MRRR implementation `DSTEGR` on the resulting Golub-Kahan matrix T_{GK}. The following table shows in the second column the measured orthogonality $||Q^TQ - I||/(n\epsilon)$ for the tSEP posed by T_{GK}. The third column shows the orthogonality $\max(||U^TU - I||, ||V^TV - I||)/(n\epsilon)$ of the extracted singular vectors from Q, and in the fourth column the same measure is shown for the results obtained by our software `DBDSCR`.

n	`DSTEGR` on T_{GK}		`DBDSCR` on B
	tSEP	bSVD	bSVD
100	4.45	$> 10^{10}$	3.71
200	5.10	$> 10^{10}$	4.13
400	2.93	$> 10^{10}$	3.80

Note that Q itself fulfills the requirements (2.2) for the tSEP posed by T_{GK} nicely, but the singular vectors extracted from Q are effectively useless. As can be seen in the last column, the coupling techniques which we will present in the next section avoid this problem.

A short note on notation. In the remaining part of this paper we will have to deal constantly with the three matrices B^TB, BB^T and T_{GK} and the various bidiagonal factorizations occuring in their respective representation trees for the MRRR algorithm. In order to distinguish between these matrices, we will continue to use superscripts $\wedge, \vee$ and $\sim$, as introduced in this section and presented in the following diagram.

$$\begin{tikzcd}
B^TB \arrow[d, "\mu^2"] & T_{GK} \arrow[d, "\mu"] & BB^T \arrow[d, "\mu^2"] \\
\hat{L}\hat{D}\hat{L}^T & \tilde{L}\tilde{D}\tilde{L}^T & \check{L}\check{D}\check{L}^T
\end{tikzcd}$$

4. The Bidiagonal MRRR Algorithm. As we have seen in the preceding section, the main problem when trying to solve the bSVD by the application of MRRR to B^TB and BB^T lies in the separate factorizations, which can cause the local eigenvalues to drift apart.

In [12], Großer and Lang proposed a solution to this problem. They devised so called *coupling relations*, which link the factorizations

$$B^TB - \mu^2 I =: \hat{L}\hat{D}\hat{L}^T, \quad T_{GK} - \mu I =: \tilde{L}\tilde{D}\tilde{L}^T, \text{ and } BB^T - \mu^2 I =: \check{L}\check{D}\check{L}^T$$

in a backward stable way. As a consequence, it is only necessary to do one of the factorizations explicitly; the remaining two factorizations can then be computed implicitly using only multiplications and divisions. This guarantees that the eigenvalues of $\hat{L}\hat{D}\hat{L}^T$ and $\check{L}\check{D}\check{L}^T$ will agree to most of their digits.

This then suggests a new algorithm for the bSVD based on the MRRR algorithm. The idea is to apply MRRR simultaneously on B^TB, T_{GK} and BB^T (which we sometimes call "The Three Matrices") with identical shifts μ^2 for B^TB and BB^T and μ for T_{GK}. But the dstqds factorizations needed to proceed from one level to the next are always done only for one of the matrices—in most cases this will be T_{GK}, see below—whereas the above mentioned coupling relations are used to keep track of the other two factorizations *implicitly*. The backward stable nature of the coupling relations will ensure that the eigenvalues of a representation $\hat{L}\hat{D}\hat{L}^T$ in the tree for B^TB and the corresponding representation $\check{L}\check{D}\check{L}^T$ for BB^T always remain relatively close. As a consequence, if upon encountering a singleton the singular vectors are computed using the coupled representations of B^TB and BB^T, we get vectors with small residuals *and* good orthogonality levels.

This section is divided into two parts. First we will present the coupling relations, and then we describe our approach to incorporate them in an efficient and practicable way into an MRRR algorithm for the bSVD.

4.1. The Coupling Relations. At the core of the new algorithm lies the capability to convert between shifted factorizations of the matrices B^TB, BB^T and T_{GK} in a backward stable way. In the following, we will summarize the main results from [11, 12] needed to understand and implement the algorithm.

We will not give detailed proofs of the coupling relations in this paper, as they can be quite technical (and the following pages are already technical enough). Their main ingredient is that shifted factorizations of the three matrices can be related by

$$\begin{pmatrix} BB^T - \mu^2 I & 0 \\ 0 & B^TB - \mu^2 I \end{pmatrix} = P_{ps}^T (T_{GK} + \mu I)(T_{GK} - \mu I) P_{ps}, \tag{4.1}$$

and the fact that $T_{GK} - \mu I = \tilde{L}\tilde{D}\tilde{L}^T$ implies $T_{GK} + \mu I = \bar{L}\bar{D}\bar{L}^T$ with $\bar{d}_i = -\tilde{d}_i, i = 1 : 2n$ and $\bar{l}_i = -\tilde{l}_i, i = 1 : 2n-1$. A deeper analysis of this simple relation leads to the following result.

LEMMA 4.1. *Let the decompositions*

$$T_{GK} - \mu I = \tilde{L}\tilde{D}\tilde{L}^T = \tilde{U}\tilde{R}\tilde{U}^T = \tilde{N}_r\tilde{G}_r\tilde{N}_r^T, \;\; r = 1 : 2n,$$

exist and be RRRs. Then the decompositions

$$\begin{aligned} B^TB - \mu^2 I &= \hat{L}\hat{D}\hat{L}^T = \hat{U}\hat{R}\hat{U}^T = \hat{N}_k\hat{G}_k\hat{N}_k^T, \;\; k = 1 : n, \\ BB^T - \mu^2 I &= \check{L}\check{D}\check{L}^T = \check{U}\check{R}\check{U}^T = \check{N}_k\check{G}_k\check{N}_k^T, \;\; k = 1 : n, \end{aligned}$$

also form RRRs, and for $i = 1 : n$ the diagonal pivots and twist elements are given by

$$\begin{aligned} \hat{d}_i &= -\tilde{d}_{2i-1}\tilde{d}_{2i}, & \check{d}_i &= -\tilde{d}_{2i}\tilde{d}_{2i+1}, \\ \hat{r}_i &= -\tilde{r}_{2i-2}\tilde{r}_{2i-1}, & \check{r}_i &= -\tilde{r}_{2i-1}\tilde{d}_{2i}, \\ \hat{\gamma}_i &= \mu\tilde{\gamma}_{2i-1}, & \check{\gamma}_i &= \mu\tilde{\gamma}_{2i}, \end{aligned} \tag{4.2}$$

where we set $\tilde{d}_{2n+1} := \tilde{d}_1$ and $\tilde{r}_0 := \tilde{r}_{2n}$. The elements $\hat{l}_i, \check{l}_i, \hat{u}_i, \check{u}_i$ can then be determined using

$$\hat{l}_i\hat{d}_i = \hat{u}_i\hat{r}_{i+1} = a_ib_i \quad \textit{and} \quad \check{l}_i\check{d}_i = \check{u}_i\check{r}_{i+1} = a_{i+1}b_i. \tag{4.3}$$

for $i = 1 : n-1$.

Proof. The couplings (4.2) follow from (4.1), although some more technical argumentation is needed, which is beyond the scope of this paper; see Lemma 3.1 and Corollary 3.2 in [12]. The identities (4.3) result from the fact that the offdiagonal elements $a_i b_i$ of $B^T B$ and $a_{i+1} b_i$ of BB^T are not affected by the shift. □

The requirement that each of the $2n$ twisted factorizations of $T_{GK} - \mu I$ has to be an RRR is redundant, as it is shown in [6] that if a twisted factorization is an RRR for some twist index k, then this also holds true for all twist indices.

Concerning the local eigenvalues, it was proved in [11] that if $\hat{L}\hat{D}\hat{L}^T$ and $\check{L}\check{D}\check{L}^T$ are set up from $\tilde{L}\tilde{D}\tilde{L}^T$ using (4.2) and (4.3), the relative distance of the respective eigenvalues $\hat{\lambda}_i$ and $\check{\lambda}_i$ obeys

$$\mathrm{reldist}(\hat{\lambda}_i, \check{\lambda}_i) = \mathcal{O}(\epsilon). \tag{4.4}$$

So the local eigenvalues $\hat{\lambda}_i$ and $\check{\lambda}_i$ will agree to most of their digits.

A special point in the above coupling relations is that they are completely oblivious to the way in which the factorization of $T_{GK} - \mu I$ is computed. Therefore they are also valid in the case of successive factorizations, which occur naturally during the MRRR algorithm. As an example, let us assume we apply MRRR to T_{GK}. Let us omit the index ranges of the eigenvalues for now and denote by $(\tilde{L}^{(i)}, \tilde{D}^{(i)}, \mu^{(i)}), i = 1, 2, \ldots$ a path in the representation tree, i.e.,

$$\begin{aligned}
&T_{GK} - \mu^{(1)} I =: \tilde{L}^{(1)} \tilde{D}^{(1)} (\tilde{L}^{(1)})^T \text{ and} \\
&\tilde{L}^{(i)} \tilde{D}^{(i)} (\tilde{L}^{(i)})^T - \mu^{(i+1)} I =: \tilde{L}^{(i+1)} \tilde{D}^{(i+1)} (\tilde{L}^{(i+1)})^T, \;\; i = 2, 3, \ldots .
\end{aligned}$$

Then we can use Lemma 4.1 to set up the corresponding paths $(\hat{L}^{(i)}, \hat{D}^{(i)}, \nu^{(i)})$ for $B^T B$ and $(\check{L}^{(i)}, \check{D}^{(i)}, \nu^{(i)})$ for BB^T, where the shifts $\nu^{(i)}$ are related to the GK shifts $\mu^{(i)}$ by

$$\sum_{j=1}^{i} \nu^{(j)} = \left(\sum_{j=1}^{i} \mu^{(j)} \right)^2 .$$

Evaluating this recurrence gives

$$\nu^{(i)} = \mu^{(i)} (2\bar{\mu}^{(i-1)} + \mu^{(i)}), \text{ with } \bar{\mu}^{(i-1)} := \sum_{j=1}^{i-1} \mu^{(j)}. \tag{4.5}$$

Together with (4.4) this means that we are able to run MRRR implicitly on $B^T B$ and BB^T in parallel with identical shifts, thereby guaranteeing that the local eigenvalues of the corresponding representations $\hat{L}^{(i)} \hat{D}^{(i)} (\hat{L}^{(i)})^T$ and $\check{L}^{(i)} \check{D}^{(i)} (\check{L}^{(i)})^T$ are always relatively close. This is already one big step towards the solution compared to the separate application of MRRR on $B^T B$ and BB^T.

However, we are still doing much work with the Golub-Kahan matrix and its translates. As was already hinted at in Section 3, there are two major problems when working with T_{GK}, namely that finding good shifts μ s.t. $T_{GK} - \mu I$ can be proved to be an RRR is hard and that element growth in the factorizations can lead to problems when computing the vectors. The latter is now resolved, as we can use the couplings from Lemma 4.1 to compute the vectors directly with translates of $B^T B$ and BB^T, which results in good orthogonality *and* small residuals because of (4.4). Concerning the first issue, the following Lemma summarizes another coupling relation presented in [12], which utilizes the intermediate quantities arising during the differential qds transformations in Algorithm 2.1 in order to avoid factorizing $T_{GK} - \mu^{(1)}I$ and to use $B^TB - \nu^{(1)}$ instead.

LEMMA 4.2. *Let the factorizations*

$$B^TB - \mu^2 I = \hat{L}\hat{D}\hat{L}^T = \hat{U}\hat{R}\hat{U}^T = \hat{N}_k\hat{G}_k\hat{N}_k^T, \quad k = 1:n,$$

be computed using Algorithm 2.1 with intermediate quantities $\hat{S}$ and $\hat{P}$. Then the decompositions

$$\begin{aligned} T_{GK} - \mu I &= \tilde{L}\tilde{D}\tilde{L}^T = \tilde{U}\tilde{R}\tilde{U}^T = \tilde{N}_k\tilde{G}_k\tilde{N}_k^T, \quad k = 1:2n, \\ BB^T - \mu^2 I &= \check{L}\check{D}\check{L}^T = \check{U}\check{R}\check{U}^T = \check{N}_k\check{G}_k\check{N}_k^T, \quad k = 1:n, \end{aligned}$$

are given by

$$\check{d}_i = \frac{\hat{s}_{i+1}}{\hat{s}_i}\hat{d}_i \quad \check{r}_i = \frac{\hat{p}_i}{\hat{p}_{i+1}}\hat{r}_{i+1} \quad \check{\gamma}_i = -\mu^2\hat{\gamma}_i\frac{\hat{r}_{i+1}}{\hat{s}_i\hat{p}_{i+1}}$$

and

$$\begin{array}{lll} \tilde{d}_{2i-1} = \dfrac{\hat{s}_i}{\mu} & \tilde{r}_{2i-1} = \dfrac{\hat{p}_i}{\mu} & \tilde{\gamma}_{2i-1} = \dfrac{\hat{\gamma}_i}{\mu} \\ \tilde{d}_{2i} = -\dfrac{\hat{d}_i}{\tilde{d}_{2i-1}} & \tilde{r}_{2i} = -\dfrac{\check{r}_i}{\tilde{r}_{2i-1}} & \tilde{\gamma}_{2i} = -\dfrac{\check{\gamma}_i}{\mu} \end{array}$$

for $i = 1:n$, setting $\hat{s}_{n+1} := -\mu^2, \hat{r}_{n+1} := \hat{p}_{n+1} := 1$.

Proof. See Lemma 2.3 and Corollaries 2.4 and 2.5 in [12]. ◻

The elements $\check{l}_i, \check{u}_i, \tilde{l}_i, \tilde{u}_i$ can be obtained as in (4.3) using $\check{l}_i\check{d}_i = \check{u}_i\check{r}_{i+1} = a_{i+1}b_i$, $\tilde{l}_{2i-1}\tilde{d}_{2i-1} = \tilde{u}_{2i-1}\tilde{r}_{2i} = a_i$ and $\tilde{l}_{2i}\tilde{d}_{2i} = \tilde{u}_{2i}\tilde{r}_{2i+1} = b_i$. Again it holds that if $(\hat{L}, \hat{D})$ is an RRR, then $(\check{L}, \check{D})$ is too and (4.4) is fulfilled, i.e., the eigenvalues of $\hat{L}\hat{D}\hat{L}^T$ and $\check{L}\check{D}\check{L}^T$ are relatively close [11, Theorem 5.4].

4.2. Modified MRRR algorithm with Couplings. In this section we will present the structure of our new implementation of the adapted MRRR algorithm for the bSVD with embedded couplings.

First we want to outline the main difference between our approach and the algorithm presented in [12]. There, another coupling transformation was used on deeper levels to couple directly from $(\hat{L}^+, \hat{D}^+)$ to $(\check{L}^+, \check{D}^+)$, similar to Lemma 4.2 for the first level [12, p. 15]. Unfortunately, this transformation is based on an implicit partial factorization of $\tilde{L}\tilde{D}\tilde{L}^T$. Therfore it cannot guarantee (4.4) and the only way to use this transformation is to compute the eigenvalues of $\hat{L}^+\hat{D}^+(\hat{L}^+)^T$ and $\check{L}^+\check{D}^+(\check{L}^+)^T$ to full accuracy and to compare them [12, p. 18]. The algorithm in [12] was based on the original presentation of the tridiagonal MRRR algorithm in [6], where the eigenvalues were computed to full accuracy on each level of the tree anyway. In this context, the quality of the couplings could be checked easily.

However, as we pointed out in Section 2 based on [9], it is sufficient for the MRRR algorithm to refine the eigenvalues on each level only until they can be categorized into singletons and clusters. For example, with the cluster tolerance set to 10^{-3}, this implies essentially that merely the first three decimal digits of the eigenvalues have to be computed on each level. As the computation of the eigenvalues is by far the most expensive part during the MRRR algorithm, this optimization results in a serious speedup of the implementation.

If we wanted to employ the direct coupling strategy from [12], we would have essentially two options. We could ignore the above optimization and still refine the eigenvalues to full precision on each level, which would pose a serious and unnecessary runtime overhead. Alternatively, we could use the optimization but skip checking the quality of the couplings, resulting in a loss of robustness of the method. In our opinion, both options are unacceptable.

Therefore we propose a different strategy for deeper levels. We do the steps from one level to the next with the local translate $\tilde{L}\tilde{D}\tilde{L}^T$ of the Golub-Kahan matrix and use Lemma 4.1 to set up the respective representations in the trees of B^TB and BB^T. As this coupling guarantees (4.4), we do not need to refine the eigenvalues to full accuracy anymore.

As outlined in the previous section, the new algorithm can be described as implicitly running MRRR on the matrices B^TB, T_{GK} and BB^T simultaneously with equivalent shifts. As a consequence, we are working in some sense on a (synchronized) *three-layered representation tree* with nodes $[\hat{L}\hat{D}\hat{L}^T, \tilde{L}\tilde{D}\tilde{L}^T, \check{L}\check{D}\check{L}^T, \bar{\mu}, jl : ju]$. The local index range $jl : ju$ denotes the subset of desired singular values of B (resp. eigenvalues of B^TB) and $\bar{\mu}$ is the accumulated shift from T_{GK}, i.e., we have

$$\begin{aligned} B^TB - \bar{\mu}^2 I &= \hat{L}\hat{D}\hat{L}^T, \\ T_{GK} - \bar{\mu} I &= \tilde{L}\tilde{D}\tilde{L}^T, \\ BB^T - \bar{\mu}^2 I &= \check{L}\check{D}\check{L}^T. \end{aligned}$$

Note that in order to be correct, the corresponding index range for the eigenvalues of T_{GK} would be $n + jl : n + ju$. For the sake of shorter indices we will omit this detail; that is we refer to the $(n + i)$-th eigenvalue of T_{GK} and its translates $\tilde{L}\tilde{D}\tilde{L}^T$ as $\tilde{\lambda}_i$, due to the fact that we are only interested in the n positive eigenvalues of T_{GK}.

Recall that in the tridiagonal MRRR algorithm, the main tasks to be done for each node are:

1. Refine the local eigenvalues of the shifted matrix in order to identifiy clusters and singletons.
2. For singletons, compute the eigenvector using twisted factorizations.
3. For clusters, find a shift close to the cluster resulting in a new partial RRR for the eigenvalues in the cluster. Compute and store the new representation for the work on the next level.

These steps remain essentially the same in our approach. But as we use Lemma 4.2 to handle the couplings for the root node and Lemma 4.1 for deeper levels, the above steps are applied to different matrices, depending on the level in the trees we are currently on. Therefore we will treat the root node (i.e. level zero) and the deeper levels separately in the following detailed description of our algorithm.

The Root Node. Given the upper bidiagonal B and a range $il : iu$ for the singular values of interest, in theory the root of our three-layered tree becomes

$$[B^TB, T_{GK}, BB^T, 0, il : iu],$$

but using Lemma 4.2 we will only need to work with B^TB. A structural overview of the steps performed for the root node is given in Algorithm 4.1.

Algorithm 4.1 Bidiagonal MRRR, Root Node.

Input: Upper bidiagonal matrix B, range $il : iu$ of desired singular triplets.

```
Refine Eigenvalues λ̂_{il:iu} of B^T B enough to identify clusters.
for each singleton λ̂_s do
    compute λ̂_s to full accuracy
    compute B^T B − λ̂_s I = N̂_k Ĝ_k N̂_k^T, k = 1 : n
    compute v_s for a suitable twist index k̂
    couple (N̂_k Ĝ_k N̂_k^T, k = 1 : n) → (Ň_k Ǧ_k Ň_k^T, k = 1 : n)      (Lemma 4.2)
    compute u_s for a suitable twist index ǩ
endfor
for each cluster λ̂_{c:d} do
    find close shift μ^2 s.t. B^T B − μ^2 I = L̂ D̂ L̂^T and
      (L̂, D̂) forms a partial RRR for the eigenvalues c : d
    modify eigenvalues for the next level: λ̂_i := λ̂_i − μ^2, i = c : d
    store L̂, D̂, Ŝ, μ, and c : d
endfor
```

First, the eigenvalues $\hat\lambda_i = \sigma_i^2, i = il : iu$, are refined. For each singleton $\hat\lambda_s$, we take advantage of the fact that Lemma 4.2 allows the direct coupling of twisted factorizations of $B^TB - \nu I$ and $BB^T - \nu I$. That is, after refining $\hat\lambda_s$ to high relative accuracy, we use Algorithm 2.1 to compute the twisted factorizations $B^TB - \hat\lambda_s I = \hat N_k \hat G_k \hat N_k^T, k = 1 : n$. (Note that, as B is upper bidiagonal, B^TB can be written as LDL^T with $d_i = a_i^2, l_i = b_i/a_i$. So Algorithm 2.1 can be applied directly). Then the right singular vector v_s is computed using (2.9) for a suitable twist index $\hat k$. For the left singular vector, we invoke Lemma 4.2 to set up the twisted factorizations $BB^T - \hat\lambda_s I = \check N_k \check G_k \check N_k^T, k = 1 : n$ directly, and we use them to compute u_s. Note that it is possible to choose a different twist index $\check k$ for the left vector. As the couplings are backward stable this results in excellent residuals $||Bv_s - \sigma_s u_s||$.

Now assume that a cluster $\hat\lambda_{c:d}$ of eigenvalues of B^TB has been identified. In the same manner as MRRR applied to B^TB alone would proceed, we choose a shift μ^2 s.t. $B^TB - \mu^2 = \hat L\hat D\hat L^T$ forms a partial RRR for its eigenvalues with indices $c : d$. Then we can again use Lemma 4.2 to set up the remaining data for the child node $[\hat L\hat D\hat L^T, \tilde L\tilde D\tilde L^T, \check L\check D\check L^T, \mu, c : d]$.

Deeper Levels. On deeper levels of the tree we work on nodes of the form

$$\left[\hat L\hat D\hat L^T, \tilde L\tilde D\tilde L^T, \check L\check D\check L^T, \bar\mu, jl : ju\right],$$

where $jl : ju$ is a subset of the root index range $il : iu$ with $jl < ju$. The computational structure for deeper levels is shown in Algorithm 4.2.

Again we start by refining the eigenvalues $\hat\lambda_i$, $i = jl : ju$, of $\hat L\hat D\hat L^T$. For singletons $\hat\lambda_s$, we want to compute the singular vectors u_s and v_s as for the root node using twisted factorizations

$$\hat L\hat D\hat L^T - \hat\lambda_s I = \hat N_k^+ \hat G_k^+ (\hat N_k^T)^+, \quad k = 1 : n, \quad \text{and} \tag{4.6}$$

$$\check L\check D\check L^T - \hat\lambda_s I = \check N_k^+ \check G_k^+ (\check N_k^T)^+, \quad k = 1 : n. \tag{4.7}$$

Algorithm 4.2 Bidiagonal MRRR, deeper levels (*level* ≥ 1).

if *level* $= 1$ **then**
  Retrieve $\hat{L}, \hat{D}, \hat{S}$, GK-shift from root $\bar{\mu}$ and cluster bounds $jl : ju$
else
  Retrieve $\tilde{L}, \tilde{D}$, GK-shift from root $\bar{\mu}$ and cluster bounds $jl : ju$
  couple $(\tilde{L}, \tilde{D}) \to (\hat{L}, \hat{D})$ *(Lemma 4.1)*
endif
Refine Eigenvalues $jl : ju$ of $\hat{L}\hat{D}\hat{L}^T$ enough to identify clusters
if singletons found **then**
  if *level* $= 1$ **then**
    couple $(\hat{L}, \hat{D}, \hat{S}) \to (\check{L}, \check{D})$ *(Lemma 4.2)*
  else
    couple $(\tilde{L}, \tilde{D}) \to (\check{L}, \check{D})$ *(Lemma 4.1)*
  endif
  for each singleton $\hat{\lambda}_s$ **do**
    compute $\hat{\lambda}_s$ to full accuracy
    compute $\hat{L}\hat{D}\hat{L}^T - \hat{\lambda}_s I = \hat{N}_k \hat{G}_k \hat{N}_k^T$, $k = 1 : n$, to get v_s
    compute $\check{L}\check{D}\check{L}^T - \hat{\lambda}_s I = \check{N}_k \check{G}_k \check{N}_k^T$, $k = 1 : n$, to get u_s
  endfor
endif
if new clusters found **then**
  if *level* $= 1$ **then**
    couple $(\hat{L}, \hat{D}, \hat{S}) \to (\tilde{L}, \tilde{D})$ *(Lemma 4.2)*
  endif
  for each cluster $\hat{\lambda}_{c:d}$ **do**
    approximate eigenvalues $\tilde{\lambda}_{c:d}$ based on $\hat{\lambda}_{c:d}$
    find close shift μ s.t. $\tilde{L}\tilde{D}\tilde{L}^T - \mu I = \tilde{L}^+\tilde{D}^+\tilde{(L^+)}^T$ is an RRR
    transform shift: $\nu := \mu(2\bar{\mu} + \mu)$
    modify eigenvalues for the next level: $\hat{\lambda}_i := \hat{\lambda}_i - \nu, i = c : d$
    store $\tilde{L}, \tilde{D}$, $\bar{\mu} + \mu$ and $c : d$
  endfor
endif

Unfortunately, compared to the root node case we have lost the advantage to be able to couple directly from $(\hat{N}_k^+, \hat{G}_k^+)$ to $(\check{N}_k^+, \check{G}_k^+)$. This leaves essentially two options. First we could compute the twisted factorizations $\tilde{L}\tilde{D}\tilde{L}^T - \tilde{\lambda}_s I = \tilde{N}_k^+ \tilde{G}_k^+ (\tilde{N}_k^T)^+, k = 1 : 2n$, and then use Lemma 4.1 to set up the data $\hat{N}_k^+, \hat{G}_k^+, \check{N}_k^+, \check{G}_k^+, k = 1 : n$, to compute the vectors. As this coupling obeys (4.4), it does lead to excellent results.

However, there is a drawback to this approach. In practice, the computation of eigenpairs during the MRRR algorithm can be accelerated using a specialized Rayleigh Quotient Iteration (RQI) for twisted factorizations, as described in [6, 10]. Doing this for $\tilde{L}\tilde{D}\tilde{L}^T$ is undesirable, as this matrix is of dimension $2n$ and therefore the loops in the RQI take twice as many operations as for the matrices $\hat{L}\hat{D}\hat{L}^T$ and $\check{L}\check{D}\check{L}^T$. For this reason it was proposed in [10] to forfeit the couplings at this point, i.e., to do RQI on $\hat{L}\hat{D}\hat{L}^T$ for v_s and $\hat{\lambda}_s$, and then to use the resulting approximation $\hat{\lambda}_s$ to do the factorization of $\check{L}\check{D}\check{L}^T - \hat{\lambda}_s I$ explicitly to compute u_s. This does not spoil the residuals, because, as we are on a deeper level of the tree, the local eigenvalues of $\hat{L}\hat{D}\hat{L}^T$ and $\check{L}\check{D}\check{L}^T$ are typically very small compared to the singular values of B. Therefore, the resulting absolute deviation (3.1) does not cause much harm at this point.

After dealing with the singletons we still have to handle possibly upcoming new (sub-)clusters $\hat{\lambda}_{c:d}$ on deeper levels, where $jl \le c < d \le ju$. To do the step to the next level, we use the translate $\tilde{L}\tilde{D}\tilde{L}^T$ of the Golub-Kahan matrix with a suitable shift μ to compute a new partial RRR $\tilde{L}\tilde{D}\tilde{L}^T - \mu I = \tilde{L}^+\tilde{D}^+(\tilde{L}^+)^T$. Then we apply Lemma 4.1 to set up the representations

$$\hat{L}\hat{D}\hat{L}^T - \nu I = \hat{L}^+\hat{D}^+(\hat{L}^+)^T \quad \text{and} \quad \check{L}\check{D}\check{L}^T - \nu I = \check{L}^+\check{D}^+(\check{L}^+)^T$$

for the child node

$$\left[\hat{L}^+\hat{D}^+(\hat{L}^+)^T, \tilde{L}^+\tilde{D}^+(\tilde{L}^+)^T, \check{L}^+\check{D}^+(\check{L}^+)^T, \bar{\mu}+\mu, c:d\right].$$

Its local eigenvalues are related via $\tilde{\lambda}_i = \sigma_i - \bar{\mu}$ and

$$\hat{\lambda}_i = \sigma_i^2 - \bar{\mu}^2 = (\sigma_i - \bar{\mu})(\sigma_i + \bar{\mu}) = \tilde{\lambda}_i(2\bar{\mu} + \tilde{\lambda}_i).$$

Remember that $\tilde{\lambda}_i$ refers actually to the $(n+i)$-th eigenvalue of $\tilde{L}\tilde{D}\tilde{L}^T$. Together with (4.5) we can therefore express the relations between the local eigenvalues $\hat{\lambda}_i$ and $\tilde{\lambda}_i$, and between the local shifts μ and ν as

$$\hat{\lambda}_s = \mathrm{conv}(\tilde{\lambda}_s, \bar{\mu}), \quad \nu = \mathrm{conv}(\mu, \bar{\mu}), \quad \text{where} \quad \mathrm{conv}(x, y) := x(2y + x). \tag{4.8}$$

This relation is needed for two reasons. First, after choosing μ, we need ν to get initial guesses $\hat{\lambda}_i - \nu$ for the local eigenvalues of $\hat{L}^+\hat{D}^+(\hat{L}^+)^T$. Additionally, in order to choose μ sensibly, some approximation to the eigenvalues $\tilde{\lambda}_{c:d}$ of $\tilde{L}\tilde{D}\tilde{L}^T$ is needed. We want to avoid performing any direct eigenvalue computations for $\tilde{L}\tilde{D}\tilde{L}^T$, as this would be expensive, so we approximate $\tilde{\lambda}_i$ from $\hat{\lambda}_i$ instead. The relation (4.8) implies that $\tilde{\lambda}_i$ is defined as the larger root of the quadratic equation $x^2 + 2\bar{\mu}x - \hat{\lambda}_i$. Care has to be taken to compute this root in a stable way, see [13] for example.

5. The Software `xBDSCR`. The bidiagonal MRRR algorithm as described in the last section has been realized as software `xBDSCR` in FORTRAN 77 and is to be incorporated into the upcoming release of the LAPACK library. In this section we want to discuss several practical issues concerning the implementation.

Refining the eigenvalues. Internally the approximations to the eigenvalues $\lambda_i := \hat{\lambda}_i$ of $\hat{L}\hat{D}\hat{L}^T$ are handled as half-open intervals $[\underline{\lambda}_i, \overline{\lambda}_i)$, with $\underline{\lambda}_i \le \lambda_i < \overline{\lambda}_i$. In order to identify singletons, neighboring intervals are repeatedly refined using bisection until $\overline{\lambda}_i \le \underline{\lambda}_{i+1}$ and $\mathrm{reldist}(\overline{\lambda}_i, \underline{\lambda}_{i+1})$ is larger than the cluster tolerance, or until $\mathrm{reldist}(\underline{\lambda}_i, \overline{\lambda}_{i+1})$ is smaller than the cluster tolerance or the relative width of the intervals becomes smaller than 4ϵ. In the first case, the eigenvalues can safely be regarded as separated, whereas in the second case they cannot.

Computing the eigenpairs. As already mentioned, the final computation of an eigenpair is actually done using Rayleigh Quotient Iteration with twisted factorizations (Alg. 4.1, lines 3-5 and Alg. 4.2, lines 15-16). For more details on this technique see [6, pp. 136ff]. Note that for each singular triplet, we only need to do this iteration for $\hat{L}\hat{D}\hat{L}^T$ (or B^TB). The coupling relations guarantee that the resulting refined eigenvalue $\hat{\lambda}$ approximates the corresponding eigenvalue of $\check{L}\check{D}\check{L}^T$ to high relative accuracy, therefore it can directly be used to compute the right singular vector (see Alg. 4.2, line 17).

Data storage. Algorithms 4.1 and 4.2 describe only the computations for each node in the tree, but not in which order the nodes are to be visited. In theory, this order has no effect on the algorithm at all. In practice, however, the data for each new child node has to be stored somewhere until it is visited.

It suffices to store enough information to rebuild the three representations belonging to a node using the coupling relations. For level one, we can employ Lemma 4.2 and therefore need only the elements of $\hat{L}, \hat{D}$ and $\hat{S}$ from the dstqds-factorization of $B^TB-\mu^2I$. For deeper levels, the elements of $\tilde{L}, \tilde{D}$ are enough to set up the other two representations $\hat{L}\hat{D}\hat{L}^T$ and $\check{L}\check{D}\check{L}^T$ via Lemma 4.1. Taken together, we need to store $4n + \mathcal{O}(1)$ numbers for any node in the three-layered tree. As each node represents at least two singular triplets, we can use for example the storage for the first two left and right singular vectors belonging to the node temporarily for this purpose.

With this approach, a breadth-first traversal of the tree is sensible, as this avoids unnecessary swapping of the node data. A similar technique is used in [9] for the implementation of the tridiagonal MRRR algorithm.

IEEE arithmetic. The MRRR algorithm has to deal with possible breakdowns in the factorizations. This is easy to accomplish if support of the IEEE-754 standard for floating point arithmetic is present, or at least an equivalent handling of NaN's (see [10, p. 47]). If this is not the case, special care is necessary to avoid divisions by zero and overflows. The new version `xSTEGR` of the tridiagonal MRRR algorithm works with or without IEEE support [9], and we adapted the employed techniques for the factorizations within `xBDSCR`.

However, we also have to take care of possible division by zero when using the couplings in Lemma 4.2. It was shown in [10, p. 80] how to fix this in the case that IEEE arithmetic is present. In a similar manner as with the factorizations, these modifications were extended for the case that IEEE-arithmetic is not supported.

So, as with the new `xSTEGR`, our code does not need IEEE arithmetic, but is able to exploit it. Preliminary tests indicated approximately a 10% performance improvement with IEEE support, due to the fact that the innermost loop can be formulated with fewer conditionals.

Preprocessing. In [10] it was noticed that for some kinds of matrices, the algorithm can benefit from having it preceded by some sweeps of the bidiagonal QR method, which is described for example in [4]. This sort of preprocessing for the original matrix B was integrated in the code, although only some QR sweeps are done per default, as updating the vectors afterwards with the employed orthogonal rotations is not cheap.

Splitting. The MRRR algorithm and consequently its bidiagonal adaption work only on unreduced matrices, that is, no offdiagonal of the original tridiagonal T, respective no element of the bidiagonal B should be zero.

If some offdiagonal element b_i of the original upper bidiagonal matrix B is zero, the matrix can be split into two submatrices $B_{1:i,1:i}$ and $B_{i+1:n,i+1:n}$, which then can be treated independently. If a diagonal element a_i is zero, an elegant way to "deflate" this zero out is to apply one sweep of the implicit zero-shift QR method, described in [4]. This results in a matrix B' with $b'_{i-1} = b'_{n-1} = a'_n = 0$ [4, p. 21], i.e., one zero singular value has been rotated out nicely and we can split the matrix into three blocks $B_{1:i-1,1:i-1}$, $B_{i:n-1,i:n-1}$ and $B_{n,n}$, the latter one being trivial.

Extensive splitting of the matrix should be exploited wherever possible, as it has a beneficial effect on both orthogonality and runtime. Therefore it is sensible to replace very small elements of B by zero if this affects the SVD only slightly. Standard absolute perturbation theory for the bSVD [3, Cor. 5.1] shows that setting an element c_i of B to zero can cause an absolute change of $|c_i|$ in the singular values and consequently also in the residual (1.3). This suggests the absolute splitting criterion

$$|c_i| \le \kappa n \epsilon ||B|| \quad \Rightarrow \quad c_i := 0, \tag{5.1}$$

with some small constant κ. However, doing this implies that the singular values will not be computed to high relative accuracy.

Our implementation employs a 2-phase splitting. In the first phase we split the matrix as much as possible *without* spoiling the relative accuracy of the singular values, using one or more passes of the implicit zero-shift QR method and the relative splitting criteria described in [4, p. 18]. Based on this *relative split*, we apply the absolute splitting criterion (5.1) on each of the blocks and, if necessary, do again a zero-shift QR-sweep to deflate zeros on the block-diagonals. This results in the *absolute split*, where the blocks are unreduced and subblocks of the relative split.

Then the core bidiagonal MRRR algorithm is applied to each block in the absolute split. Should relative accuracy be desired (indicateable by a flag when calling `xBDSCR`), the singular values are afterwards refined to high relative accuracy for the respective "father" block in the relative split. Note that there is no need to refine the computed singular vectors in order to get good orthogonality and small residuals.

This splitting approach has two advantages. First, we can always apply the absolute splitting criterion, even if relative accuracy is desired, and if so, we exploit the smaller blocks in the relative split to save runtime when refining the singular values.

6. Comparison with other methods. In this section, we compare our MRRR based bidiagonal SVD code with other algorithms available in LAPACK. In its current release 3.0, LAPACK provides two driver routines, `xGESVD` and `xGESDD`, for computing the singular value decomposition of a general rectangular matrix. In both cases, the matrix is first transformed to bidiagonal form; afterwards the singular values are computed from the bidiagonal matrix using the QR algorithm `xBDSQR` or Divide & Conquer `xBDSDC`, respectively. As part of the new release of LAPACK, we will provide a similar driver for our algorithm `xBDSCR`. In the following, we compare the performance of the three computational kernels `xBDSCR`, `xBDSQR`, and `xBDSDC` for the bidiagonal SVD.

As testbed we used a Pentium 4, 2.8GHz processor with 512kb cache. All routines and the LAPACK library were compiled using the Intel Fortran Compiler, version 8.1, with compiler options -O3, -tpp7 and -mp.

Figure 6.1 shows the speedup of the bidiagonal MRRR algorithm over the Divide & Conquer algorithm for computing the *full* SVD. For all matrices considered, the QR algorithm was at least five times (in some cases several hundred times) slower than these two algorithms; therefore the QR data are not shown in the pictures.

The matrices underlying the picture were designed for testing the robustness of the algorithms. In particular, many of them have very tight, and sometimes large, clusters of singular values. This situation can be favorable for the Divide & Conquer algorithm, which benefits from heavy deflation. By contrast, tight clusters may

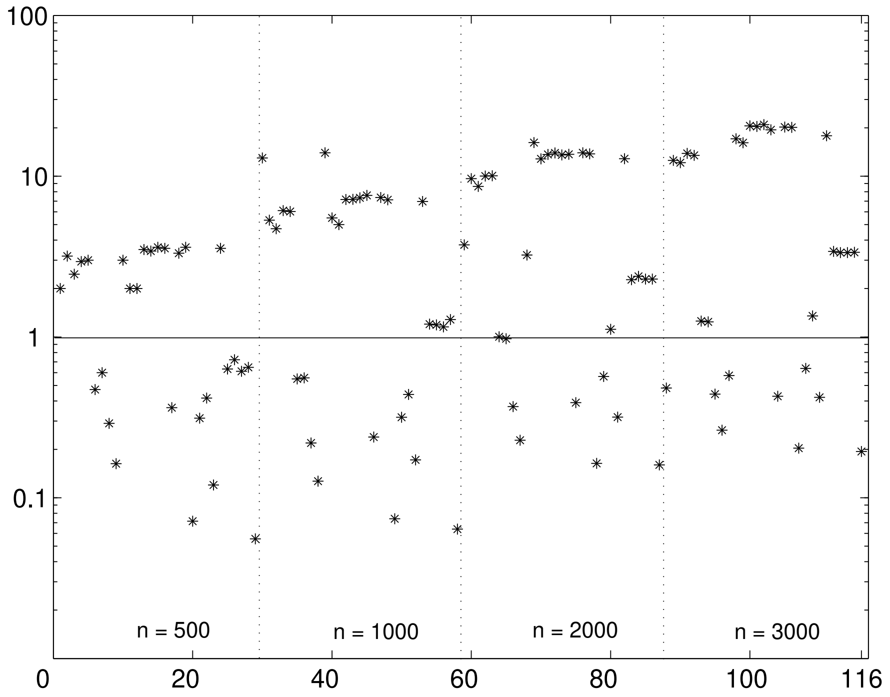

FIG. 6.1. *Speedup of the bidiagonal MRRR algorithm over the Divide & Conquer algorithm for matrices of varying dimension and distribution of the singular values.*

%	DBDSCR	DBDSDC	DBDSQR
100%	2.84	6.63	368
50%	1.56	—	—
25%	1.38	—	—
10%	0.63	—	—

TABLE 6.1
Average execution times in seconds for computing a random subset of consecutive singular triplets for the test matrix defined in (3.2) with dimension 2000. Each test was repeated 10 times.

force the bidiagonal MRRR algorithm to descend several levels in the representation tree, thus increasing its operation count. For these reasons, none of the two algorithms is consistently superior, in particular for the small matrices. As can be seen in the picture, with increasing matrix dimension the optimal $O(n^2)$ complexity of the bidiagonal MRRR becomes decisive, such that this algorithm is faster in most cases.

Our bidiagonal MRRR routine `xBDSCR` provides the option to compute only selected singular vectors. Table 6.1 shows that this feature can indeed reduce the computation time significantly. For algorithmic reasons, neither the QR nor the Divide & Conquer routine can provide partial SVDs.

Concerning accuracy, each of the three bSVD routines yielded deviations from orthogonality and residuals within the bounds (1.2) and (1.3), respectively. The errors of the bidiagonal MRRR algorithm tend to be larger than those of the remaining two methods, but only by a moderate factor between 10 and 20. As our routine is strongly based on the newest implementation `xSTEGR`, the behaviour is very similar to that algorithm concerning the comparision of orthogonality, residuals and runtime with the Divide & Conquer and QR routines. Therefore we refer readers interested in a more detailed discussion of test cases to [5].

7. Conclusions. We have described improvements to the bidiagonal MRRR algorithm and its realization in our new software implementation, which allows the computation of subsets of k singular values and vectors at $\mathcal{O}(nk)$ cost. Due to the nature of both the QR and the Divide & Conquer algorithms, this functionality was not available; the whole set of singular values and vectors had to be computed at full cost with respect to operations and storage.

As the bidiagonal MRRR algorithm is structurally very similar to the tridiagonal MRRR algorithm for tSEP, it inherits the superior features of the latter. The theoretical complexity of the (bidiagonal) MRRR algorithm is $\mathcal{O}(n^2)$, versus $\mathcal{O}(n^3)$ for the QR algorithm, and Divide & Conquer lies in between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$, depending on the matrix and the amount of deflation. Additionally, the (bidiagonal) MRRR algorithm is naturally parallelizable since the computation for the (singular) vectors within a cluster does not depend on the computation for any other cluster. Furthermore, most of the computation time is spent refining the eigenvalues, and this part is perfectly parallelizable. We plan to develop a parallel version of our algorithm for ScaLAPACK in the future.

As a major challenge for future research we consider the task to devise a stable coupling scheme between successively shifted factorizations of B^TB and BB^T. This would significantly improve and simplify the bidiagonal MRRR algorithm, as then there would be no need to work with the Golub-Kahan matrix anymore. To this end it would be sufficient to develop reliable and cheap criteria in order to test if the couplings proposed in [12] are stable.

An alternative solution to this problem would be to eliminate the need for deeper level couplings at all, that is to improve the tridiagonal MRRR algorithm in a way that the depth of the representation tree remains limited to one. This would be a major achievement indeed, but at this state of research it appears to be a very distant goal. As one possible plan of attack in this direction we see a combination of multistep inverse iteration as presented in [21] with some variant of the submatrix method for tightly clustered eigenvalues [16].

Acknowledgements. The authors thank Beresford Parlett and James Demmel for many enlightening discussions. Their contributions have greatly influenced the work leading to this paper. We also thank Osni Marques to let us build upon his test suite for `xSTEGR` in order to develop a test environment for our own code. In addition, Paul Willems thanks the Research Centre Jülich for financial support during a research visit in Berkeley at the end of 2004 and Beresford Parlett and James Demmel for their hospitality during this time.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, PA, 3. ed., 1999.

[2] C. Davis and W. Kahan, *The rotation of eigenvectors by a perturbation. III*, SIAM Journal on Numerical Analysis, 7(1) (1970), pp. 1–47.

[3] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[4] J. W. Demmel and W. Kahan, *Accurate singular values of bidiagonal matrices*, SIAM Journal on Scientific Computing, 11 (1990), pp. 873–912.
[5] J. W. Demmel, O. A. Marques, B. N. Parlett, and C. Vömel, *Performance and accuracy of the symmetric eigensolvers in LAPACK.* University of California, Berkeley, 2005. In preparation.
[6] I. S. Dhillon, *A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*, PhD thesis, University of California, Berkeley, 1997.
[7] I. S. Dhillon and B. N. Parlett, *Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices*, Linear Algebra and its Applications, 387 (2004), pp. 1–28.
[8] ———, *Orthogonal eigenvectors and relative gaps*, SIAM Journal on Matrix Analysis and Applications, 25 (2004), pp. 858–899.
[9] I. S. Dhillon, B. N. Parlett, and C. Vömel, *The design and implementation of the MRRR algorithm*, Tech. Rep. UCBCSD-04-1346, University of California, Berkeley, 2004. (also as LAPACK Working Note #162).
[10] B. Grosser, *Ein paralleler und hochgenauer $O(n^2)$ Algorithmus für die bidiagonale Singulärwertzerlegung*, PhD thesis, Bergische Universität Wuppertal, Fachbereich Mathematik, Wuppertal, Germany, 2001. In German.
[11] B. Grosser and B. Lang, *On symmetric eigenproblems induced by the bidiagonal SVD.* To appear in SIAM J. on Matrix Analysis and Applications.
[12] B. Grosser and B. Lang, *An $O(n^2)$ algorithm for the bidiagonal SVD*, Linear Algebra and its Applications, 358 (2003), pp. 45–70.
[13] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 2nd ed., 2002.
[14] B. N. Parlett, *The symmetric eigenvalue problem*, Prentice Hall, Englewood Cliffs, NJ, 1980.
[15] ———, *Acta Numerica*, Cambridge University Press, 1995, ch. The new qd algorithms, pp. 459–491.
[16] ———, *Invariant subspaces for tightly clustered eigenvalues of tridiagonals*, BIT Num. Math., 36 (1996), pp. 542–562.
[17] B. N. Parlett and I. S. Dhillon, *Fernando's solution to Wilkinson's problem: an application of double factorization*, Linear ALgebra and its Applications, 267 (1997), pp. 247–279.
[18] ———, *Relatively robust representations of symmetric tridiagonals*, Linear Algebra and its Applications, 309 (2000), pp. 121–151.
[19] B. N. Parlett and O. Marques, *An implementation of the dqds algorithm (positive case)*, Linear ALgebra and its Applications, 309 (2000), pp. 217–259.
[20] B. N. Parlett and C. Vömel, *How the MRRR algorithm can fail on tight eigenvalue clusters*, Tech. Rep. UCBCSD-04-, University of California, Berkeley, 2004. (also as LAPACK Working Note #163).
[21] ———, *LAPACK working note: How to improve FP vectors in the MRRR algorithm by twisted inverse iteration*, Tech. Rep. UCBCSD-04-1365, University of California, Berkeley, 2004.