SIAM J. MATRIX ANAL. APPL.
Vol. 29, No. 4, pp. 1343–1362

NEW FAST AND ACCURATE JACOBI SVD ALGORITHM. II*

ZLATKO DRMAČ† AND KREŠIMIR VESELIĆ‡

Abstract. This paper presents a new one-sided Jacobi SVD algorithm for triangular matrices computed by revealing QR factorizations. If used in the preconditioned Jacobi SVD algorithm, described in part one of this paper, it delivers superior performance leading to the currently fastest method for computing SVD decomposition with high relative accuracy. Furthermore, the efficiency of the new algorithm is comparable to the less accurate bidiagonalization-based methods. The paper also discusses underflow issues in floating point implementation and shows how to use perturbation theory to fix the imperfectness of the machine arithmetic.

Key words. Jacobi method, singular value decomposition, eigenvalues

AMS subject classifications. 15A09, 15A12, 15A18, 15A23, 65F15, 65F22, 65F35

DOI. 10.1137/05063920X

1. Introduction.

Jacobi iteration is one of the time-honored methods for computing the spectral decomposition $H = V\Lambda V^T$ of a real symmetric matrix H. The early discovery in 1846 is certainly due to the simplicity and the elegance of the method as well as to the geniality of C. G. J. Jacobi, who called it "ein leichtes Verfahren" and applied it to compute the secular perturbations of the planets. Jacobi's original article [25] is a masterpiece of applied mathematics and may even today be read with profit by both students and scientists. The simplicity of the Jacobi method is not only theoretical but also computational, and in this respect it may well be compared with Gaussian elimination. Thus, with coming of automatic computation, the Jacobi method was soon rediscovered by Goldstine, Murray, and von Neumann [17], who provided the first detailed implementation and error analysis.

In our recent work [14] we introduced a preconditioner for the Hestenes variant [24] of the Jacobi method for SVD computation of general matrices. We have shown that rank revealing QR factorization can serve as a versatile preconditioner which enables efficient execution of Jacobi iterations on the triangular factor. The idea of QR iterations as preconditioner for SVD computation is well known (see [33], [27], [15]), but thus far it has not been fully exploited in the context of the Jacobi method. It is both simple and powerful: If $AP = Q(R^T\ 0)^T$ is the Businger–Golub QR factorization of A, then the Hestenes one-sided Jacobi algorithm applied to $X = R^T$ converges much faster than if applied to A. (If R is singular, then the second QR factorization $R^T P_1 = Q_1 \left(R_1^T \quad 0 \right)^T$ provides nonsingular $X = R_1^T$.) In [14] Jacobi iterations on triangular matrices are used as a *black-box* procedure: starting with $X^{(0)} = X$, the sequence $X^{(k+1)} = X^{(k)}V^{(k)}$ converges to $X_\infty = U\Sigma$ and the product of Jacobi rotations $V^{(0)}V^{(1)}\cdots$ converges to V. The SVD of X is $X = U\Sigma V^T$, where the matrix

*Received by the editors August 30, 2005; accepted for publication (in revised form) by M. Chu June 5, 2007; published electronically January 4, 2008. This work was supported by the Volkswagen–Stiftung grant *Designing Highly Accurate Algorithms for Eigenvalue and Singular Value Decompositions.*

http://www.siam.org/journals/simax/29-4/63920.html

†Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia (drmac@math.hr). The work of this author was supported by the Croatian Ministry of Science and Technology under grant 0037120 (*Numerical Analysis and Matrix Theory*).

‡Lehrgebiet Mathematische Physik, Fernuniversität Hagen, Postfach 940, D-58084 Hagen, Germany (kresimir.veselic@FernUni-Hagen.de).

V is obtained not from the accumulated product of Jacobi rotations but rather in an a posteriori manner, using the relation $V = X^{-1}X_\infty$. Assembling the SVD of A from the SVD of X is straightforward.

In this report we unwrap the black box and present a new one-sided Jacobi SVD method for triangular matrices. A new pivot strategy is introduced in section 2. We use the triangular structure to reduce the flop count and memory traffic. At the same time, faster convergence is achieved using the knowledge of the asymptotic behavior of Jacobi iterations. We also use the structure of the SVD of triangular matrices, obtained from the theory of symmetric quasi-definite matrices. A new ordering of rotations is also designed to improve the use of fast cache memory. Underflow problems in floating point implementation of the algorithm are solved using perturbation theory in section 3. Numerical testing of the new preconditioned Jacobi SVD algorithm (cf. [14, Algorithm 4] with the new triangular SVD method from this paper) is presented in section 4. The results presented in section 4.3 carry the main message of [8], [14], and this paper: *Our new Jacobi SVD algorithm is more accurate than the bidiagonalization-based QR (SGESVD) and divide-and-conquer (SGESDD) algorithms from LAPACK* [1]. *Moreover, the new algorithm can compute the SVD faster than SGESVD, and it is not much slower than SGESDD.* Concluding remarks and discussion of future work are given in section 5.

2. One-sided Jacobi SVD on $n \times n$ preconditioned triangular matrices. The Jacobi transformation $X^{(k+1)} = X^{(k)}V^{(k)}$ transforms pivot columns p_k, q_k chosen by pivot strategy (ordering) $k \mapsto (p_k, q_k)$. An example is the row-cyclic strategy, which is periodic, and in one full sweep of $n(n-1)/2$ rotations it rotates at the pivot positions $(1,2),(1,3),\ldots,(1,n);\ (2,3),\ldots,(2,n);\ (3,4),\ldots,(3,n);\ \ldots,(n-2,n);\ (n-1,n)$. The convergence of $X^{(k)}$ is studied in terms of the convergence of $H^{(k)} = (X^{(k)})^T X^{(k)}$ towards the diagonal form, and its rate is usually measured using the off-norm, $\mathbf{\Omega}(H^{(k)}) = \|H^{(k)} - \mathrm{diag}(H^{(k)})\|_F$. In the case of the row-cyclic strategy the convergence is asymptotically quadratic [20]: If $\mathbf{\Omega}(H^{(0)})$ is sufficiently small and if the diagonal entries of $H^{(0)}$ are sorted, then $\mathbf{\Omega}(H^{(n(n-1)/2)}) \leq \mathrm{const} \cdot \mathbf{\Omega}(H^{(0)})^2$.

In practice, the Jacobi rotation $V^{(k)}$ is executed only if the cosine of the angle between $X^{(k)}(:,p_k)$ and $X^{(k)}(:,q_k)$ is greater than a tolerance which is usually n times the round-off $\boldsymbol{\varepsilon}$. Otherwise, the rotation is skipped. If $n(n-1)/2$ consecutive rotations with all possible pivot pairs are skipped, i.e.,

$$\max_{i \neq j} \frac{|X(:,i)^T X(:,j)|}{\|X(:,i)\|_2 \|X(:,j)\|_2} \leq n\boldsymbol{\varepsilon}, \tag{2.1}$$

then the iterations are stopped and numerical convergence is declared. Demmel and Veselić [8] showed that (2.1) is important for high relative accuracy. If the floating point Jacobi rotation is implemented as in [11], then the procedure can compute the singular values in the full range of machine numbers.

Our new pivot strategy is based on the fact that the initial matrix $X = X^{(0)}$ is triangular, nonsingular, and with additional structure implied by the Businger–Golub column pivoting. In this section, we outline the key ingredients of the new approach.

2.1. SVD of lower triangular matrix. As discussed in [14], the preconditioned Jacobi SVD algorithm computes a rank revealing QR factorization $AP = Q\binom{R}{0}$, and then it applies one-sided Jacobi SVD to $X = R^T$. (In some cases, it uses second QR factorization $R^T = Q_1 \begin{pmatrix} R_1^T & 0 \end{pmatrix}^T$ and then $X = R_1^T$.) The reason is that RR^T is more diagonal than R^TR. In addition to the arguments in [14], we give an

illustration by simple MATLAB example and show that the gap revealing property of the pivoted QR factorization (cf. [31], [32]) opens a connection to the theory of symmetric quasi-definite matrices.

Example 2.1. We generate in MATLAB $A \in \mathbb{R}^{50\times 50}$ with entries uniformly distributed over $[0,1]$, and compare the column norms of A and $X = R^T$. Further, we compare the diagonality of the Gram matrices $H_s = R_c^T R_c$ and $M_s = X_c^T X_c$, where R_c, X_c are obtained by column equilibration, e.g., $X_c = X\mathrm{diag}(\|X(:,i)\|_2^{-1})$. The results shown in Figure 2.1 strongly suggest that the Jacobi method should diagonalize $X^T X$ faster than $R^T R$. In the case of graded A (column norms vary in length) the positive effect of using X instead of R is even stronger. (See the second row in Figure 2.1.) An analysis of perfect behavior of the column norms of X can be found in [12].

It appears that the Jacobi iterations $X^{(k+1)} = X^{(k)}V^{(k)}$ in general work better if the initial X is lower triangular. An explanation is given in the following theorem.

THEOREM 2.1. *Let $X \in \mathbb{R}^{n\times n}$ be a lower triangular matrix with the partition*

$$X = \begin{pmatrix} X_{11} & 0 \\ X_{21} & X_{22} \end{pmatrix}, X_{11} \in \mathbb{R}^{k\times k}, \quad \text{and let } \sigma_{\min}\left(\begin{pmatrix} X_{11} \\ X_{21} \end{pmatrix}\right) > \sigma_{\max}(X_{22}). \tag{2.2}$$

Let the SVD of X be given with the partition

$$X = U\Sigma V^T = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix} \begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}^T. \tag{2.3}$$

Then the matrix V is more block diagonal than U in the following sense:

(i) *In the Löwner partial order $V_{11}^T V_{11} \succ V_{21}^T V_{21}$, $V_{22}^T V_{22} \succ V_{12}^T V_{12}$. (For symmetric matrices S_1 and S_2, $S_1 \succ S_2$ if and only if $S_1 - S_2$ is positive definite.) As a consequence, $\sigma_{\min}(V_{11}) > 1/\sqrt{2}$ and $\sigma_{\min}(V_{22}) > 1/\sqrt{2}$.*

(ii) *Let $\mathcal{U}_k$, $\mathcal{V}_k$, $\mathcal{I}_k$ be the subspaces spanned by the first k columns of U, V, and the identity matrix I, respectively. If the angles ψ_k, θ_k are defined as $\psi_k = \angle(\mathcal{V}_k, \mathcal{I}_k)$, $\theta_k = \angle(\mathcal{U}_k, \mathcal{I}_k)$, then $\psi_k < \pi/4$ and*

$$\tan\psi_k \le \frac{\sigma_{k+1}}{\sigma_k} \tan\theta_k.$$

Proof. Consider the block partition of the cross product matrix $H = X^T X$,

$$X^T X = \begin{pmatrix} X_{11}^T X_{11} + X_{21}^T X_{21} & X_{21}^T X_{22} \\ X_{22}^T X_{21} & X_{22}^T X_{22} \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}.$$

The gap assumption (2.2) implies that for any shift $\xi \in \mathcal{S} = (\lambda_{\max}(H_{22}), \lambda_{\min}(H_{11}))$ both $H_{11} - \xi I$ and $\xi I - H_{22}$ are positive definite. Therefore, the matrix $H - \xi I$ is symmetric quasi-definite, and the matrix V must have the special structure of the eigenvector matrix of quasi-definite matrices [16]. In particular, $V_{11}^T V_{11} \succ V_{21}^T V_{21}$, $V_{22}^T V_{22} \succ V_{12}^T V_{12}$, yielding $\sigma_{\min}(V_{11}) > 1/\sqrt{2}$, $\sigma_{\min}(V_{22}) > 1/\sqrt{2}$. From this and $X_{11}V_{11} = U_{11}\Sigma_1$ it follows that U_{11} is nonsingular. Further, from the SVD of X it follows that

$$V_{11}^{-1}V_{12} = -(V_{22}^{-1}V_{21})^T = \Sigma_1^{-1}U_{11}^{-1}U_{12}\Sigma_2 \ \text{ and } \ \|V_{11}^{-1}V_{12}\|_2 \le \frac{\sigma_{k+1}}{\sigma_k}\|U_{11}^{-1}U_{12}\|_2.$$

Finally, from the CS decomposition of partitioned V we have $\|V_{11}^{-1}V_{12}\|_2 = \tan\psi_k$. ∎

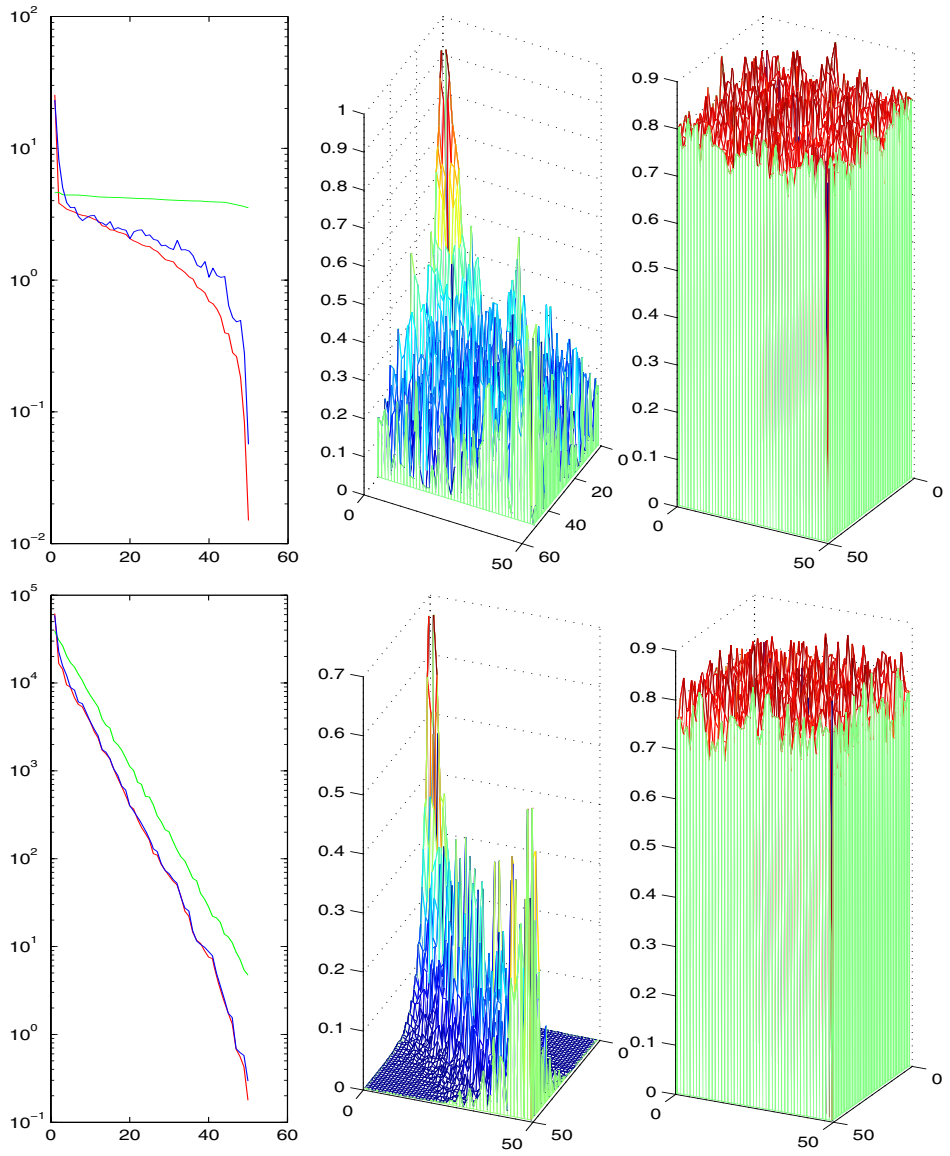

FIG. 2.1. *Example* 2.1*: In the first plot, the top line denotes* sorted *column norms of* A*, the middle line denotes the column norms of* $X = R^T$*, and lowest line denotes the singular values. The next two plots are obtained by* $meshz(abs(M_s - \mathrm{diag}(M_s)))$ *and* $meshz(abs(H_s - \mathrm{diag}(H_s)))$*, respectively. In the second row of the figure, the plots correspond to matrix* A *with graded columns.*

Remark 2.1. Part (ii) of Theorem 2.1 is due to Chandrasekaran and Ipsen [6] but under the assumption that X and one of the matrices U_{11}, V_{11} is nonsingular. The contribution of (ii) is in establishing a connection between the separation condition (2.2) and the theory of symmetric quasi-definite matrices to conclude (i), which yields nonsingularity of V_{11}. Further, our "weak separation" condition (2.2) is weaker than the usual condition $\sigma_{\min}(X_{11}) > \sigma_{\max}(X_{22})$.

Remark 2.2. Comparing X^TX with $XX^T = \begin{pmatrix} X_{11}X_{11}^T & X_{11}X_{21}^T \\ X_{21}X_{11}^T & X_{22}X_{22}^T + X_{21}X_{21}^T \end{pmatrix}$ we see that

one important difference is that the monotonicity principle acts inside different diagonal blocks. Thus, it can increase or decrease the initial separation gap. Also note that in the case of "strong separation" $\sigma_{\min}(X_{11}) > \sigma_{\max}((X_{21}\ X_{22}))$, even the matrix U has the property (i). However, property (ii) implies that V is more block diagonal. Hence, we prefer V to be the product of Jacobi rotations.

In the Jacobi SVD algorithm applied to $X = R^T$, the matrix V is built as the accumulated product of Jacobi rotations. The structure of V (more block diagonal than U) is an additional argument to apply Jacobi rotations to the columns of X.

2.2. How to exploit the triangular form. Classical pivot strategies in the one-sided Jacobi method are not designed to preserve any zero pattern of the input matrix. In fact, the incapability to preserve created zeros was the main reason for the poor performance, as compared with bidiagonalization-based methods.

However, if the initial matrix is triangular, quite a few rotations in the first sweep can use and partially preserve the zero structure. Let X denote the array in the memory occupied by the iterates in the Jacobi SVD algorithm. Initially, X is lower triangular; see (2.4). Let the columns of X be partitioned into four blocks, $X_{[1]}$, $X_{[2]}$, $X_{[3]}$, $X_{[4]}$, of dimensions $n \times n_i$, respectively, where each n_i is approximately $n/4$. Further, let $X^{[1]} = \big(X_{[1]}, X_{[2]}\big)$, $X^{[2]} = \big(X_{[3]}, X_{[4]}\big)$.

$$X = \left(\begin{array}{cc|cc||cc|cc} \blacksquare & o & 0 & 0 & 0 & 0 & 0 & 0 \\ \blacksquare & \blacksquare & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \blacksquare & \blacksquare & \blacksquare & o & 0 & 0 & 0 & 0 \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 & 0 & 0 & 0 \\ \hline\hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & o & 0 & 0 \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 & 0 \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & o \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{array}\right) \equiv \big(X_{[1]}, X_{[2]}, X_{[3]}, X_{[4]}\big) \equiv (X^{[1]}, X^{[2]}). \tag{2.4}$$

We will say that rotations are applied in $X_{[i]}^{\circlearrowleft}$ ($X_{\circlearrowleft}^{[i]}$) if we implicitly transform $X_{[i]}^T X_{[i]}$ ($(X^{[i]})^T X^{[i]}$) by following one full sweep of some pivot strategy. Further, for two blocks $X_{[i]}$ and $X_{[j]}$ ($X^{[i]}$ and $X^{[j]}$) of X, rotating in $X_{[i]} \leftrightarrow X_{[j]}$ ($X^{[i]} \leftrightarrow X^{[j]}$) means choosing, in some order, all pivot pairs with the first pivot column from $X_{[i]}$ ($X^{[i]}$) and the second one from $X_{[j]}$ ($X^{[j]}$).

Consider the most natural greedy approach. Rotating in $X_{[4]}$ is efficient because all columns are, and remain during rotations, in a canonical subspace of dimension $n_4 \approx n/4$. Thus, only the submatrix $X_{[4]}(n_1 + n_2 + n_3 + 1 : n, 1 : n_4)$ is transformed. This reduces the operation count of rotations applied in $X_{[4]}^{\circlearrowleft}$ by a factor of four. In the same way, transformations of the columns of $X_{[3]}$ by any strategy is computed in a subspace of dimension $n_3 + n_4 \approx n/2$. The same holds for the transformations of the columns of $X^{[2]}$. Repeated transformations of the columns of $X_{[3]}$ (and independently $X_{[4]}$) are still in a lower-dimensional subspace, and thus very efficient. Savings in the transformations of the columns of $X_{[2]}$ are modest, but not worthless. Note that this strategy in the first sweep transforms more often closer to the diagonal, which seems reasonable given the structure of the initial matrix (see Example 2.1).

DEFINITION 2.2. *The greedy triangular sweep for the partition* (2.4) *of lower triangular matrix is defined as the following ordering of Jacobi rotations:*

$$X_{[3]}^{\circlearrowleft},\ X_{[4]}^{\circlearrowleft},\ X_{[3]} \leftrightarrow X_{[4]},\ X_{[3]}^{\circlearrowleft},\ X_{[4]}^{\circlearrowleft};\ X_{[1]}^{\circlearrowleft},\ X_{[2]}^{\circlearrowleft},\ X_{[1]} \leftrightarrow X_{[2]},\ X_{[1]}^{\circlearrowleft},\ X_{[2]}^{\circlearrowleft};\ X^{[1]} \leftrightarrow X^{[2]}. \tag{2.5}$$

In each bulk of rotations ($X_{[i]}^{\circlearrowleft}$, $X_{[j]} \leftrightarrow X_{[k]}$, $X^{[j]} \leftrightarrow X^{[k]}$) pivot ordering is arbitrary and implemented to transform only the nontrivial parts of the corresponding submatrices.

Thus, in the greedy triangular sweep, the blocks of zeros denoted by "0" in (2.4) are at some point used to reduce the complexity (number of operations and memory traffic), as discussed above. This is particularly important because the first sweep is the busiest one. The positions denoted by "o" are treated as nonzero entries, i.e., they are not used to save operations.

This technique can be applied recursively by refining the partition (2.4).

2.3. Cubic convergence. Mascarenhas [26] observed that in the row-cyclic strategy the off-diagonal entries converge to zero at different rates and showed that by using special quasi-cyclic strategies the Jacobi method can attain cubic asymptotic convergence. Here the term quasi-cyclic refers to a modified row-cyclic strategy in which slowly convergent positions are visited more often. To motivate quasi-cyclic strategy, assume that $H = X^T X$ is almost diagonal, $\mathbf{\Omega}(H) = O(\epsilon)$, and introduce the following block partitions:

$$H = \left(\begin{array}{c|c} H^{[11]} & H^{[12]} \\ \hline H^{[21]} & H^{[22]} \end{array}\right) = \left(\begin{array}{c|c|c|c} H_{[11]} & H_{[12]} & H_{[13]} & H_{[14]} \\ \hline H_{[21]} & H_{[22]} & H_{[23]} & H_{[24]} \\ \hline H_{[31]} & H_{[32]} & H_{[33]} & H_{[34]} \\ \hline H_{[41]} & H_{[42]} & H_{[43]} & H_{[44]} \end{array}\right). \tag{2.6}$$

For simplicity, assume that the eigenvalues of H are well separated. If the row-cyclic strategy is first applied to the diagonal blocks $H^{[11]}$ and then to $H^{[22]}$, their off-diagonal norms will be reduced from $O(\epsilon)$ to $O(\epsilon^2)$. Rotating in the row-cyclic fashion inside the block $H^{[12]}$ reduces its norm to the order of $O(\epsilon^3)$. Visiting the diagonal blocks $H^{[11]}$ and $H^{[22]}$ once more will reduce their off-norms to $O(\epsilon^4)$. Repeating this pattern recursively on a finer partition of H, we obtain the quasi-cyclic ordering which visits all pivot pairs from

$$H_{[33]}, H_{[44]}, H_{[34]}, H_{[33]}, H_{[44]}, H_{[11]}, H_{[22]}, H_{[12]}, H_{[11]}, H_{[22]}, H^{[12]}, \tag{2.7}$$

respectively. The pivot positions inside each block are visited using a row-cyclic strategy.

THEOREM 2.3 (see Rhee and Hari [28]). *Let the diagonal entries of H in* (2.6) *be ordered from large to small, and let no two diagonal entries from different blocks $H_{[ii]}$, $H_{[jj]}$ be affiliated with the same eigenvalue. Let $\boldsymbol{\delta} = \min_{\lambda_i \neq \lambda_j} |\lambda_i - \lambda_j|$ and*

$$\Gamma_1 = \frac{\sqrt{\mathbf{\Omega}(H^{[11]})^2 + \mathbf{\Omega}(H^{[22]})^2}}{\boldsymbol{\delta}/3}, \quad \Gamma_2 = \left(\frac{\|H^{[12]}\|_F}{\boldsymbol{\delta}/3}\right)^{2/3}, \quad \Gamma \equiv \Gamma(H) = \max\{\Gamma_1, \Gamma_2\}.$$

Let H' be the matrix computed after the quasi-cycle (2.7). *If $\Gamma(H) < 1/4$, then $\Gamma(H') < (49/25)\Gamma(H)^3$. Further, if $\Gamma(H) = \Gamma_1 < 1/4$, then $\mathbf{\Omega}(H') \leq (18/\boldsymbol{\delta}^2)\mathbf{\Omega}(H)^3$.*

Rhee and Hari pointed out that the reduction of $\mathbf{\Omega}(H)$ is only quadratic if Γ_2 dominates Γ_1. One of the key points in our new preconditioned algorithm is that preconditioning makes the dominance of Γ_2 over Γ_1 very unlikely; see Example 2.1. Intuitively, it is then a reasonable strategy to take care of Γ_1 first and then follow the strategy (2.7) in its implicit form (2.5). We expect positive effects of the cubic convergence mechanism even in the first sweep, before the conditions for the cubic convergence are fulfilled.

Let us summarize the elements of the first sweep and the benefits of working on $X = R^T$, as compared to the classical application of the Hestenes Jacobi SVD on A:

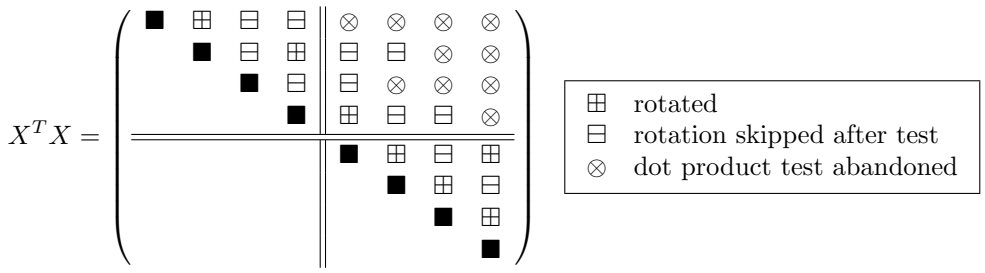

FIG. 2.2. *Example of modified row-cyclic strategy: if two consecutive rotations in a row are skipped, then the remaining pivot positions in that row are not even tested against the threshold.*

(i) R is computed efficiently by BLAS 3 operation; smaller dimension in the case $m > n$;

(ii) preconditioning effect (X^TX closer to diagonal than A^TA; see Example 2.1);

(iii) a priori known structure of X^TX and of Jacobi rotations (Theorem 2.1);

(iv) greedy sweep (2.5) exploits triangular structure to save flops and reduce memory traffic; it transforms more often where most needed, and at the same time it follows cubically convergent strategy (2.7).

2.4. How to adapt to the nearly band structure. Upon completion of the first sweep, the array X is dense. However, X is the result of the preconditioning by a rank revealing QR factorization, followed by a greedy sweep. Hence, X must have structure that can be exploited. We expect that $H = X^TX$ is a scaled diagonally dominant (s.d.d.) matrix in the sense of [2] and that its off-diagonal mass is distributed close to the diagonal. This is a typical nonpathological situation. The pathological case occurs in the presence of multiple or tightly clustered singular values.

It is known that the convergence of Jacobi iterations is improved if the threshold for the rotation is set higher at the beginning of the process and then gradually reduced to the final level. Such a strategy is not suitable for the implicit Jacobi SVD algorithm because $O(n)$ flops are required to compute a single pivot element. Further, in the nonpathological case we may expect that many rotations with pivot positions far from diagonal will be skipped. In fact, in each row of the row-cyclic pivoting, each skipped rotation increases the probability that the next rotation in that row will be skipped as well. Hence, it is reasonable to have a pivot strategy which will dynamically adapt to the structure of the matrix and anticipate small pivots. This strategy saves many unnecessary dot products, and it can be applied inside the blocks of the quasi-cyclic strategy (2.5) as well as globally on X.

The scheme on Figure 2.2 gives the main idea. The basic strategy is row-cyclic with de Rijk's pivoting[1] [9], but if in a row i a certain number of consecutive rotations is skipped (because the pivot elements are sufficiently small), the control of the row-cycling moves to the next row—the remaining pivot positions of the ith row are not visited. This strategy is motivated by the following reasons. First, it is very likely that the $\otimes$-positions in Figure 2.2 will pass the tolerance check, so we save unnecessary dot products. Second, even if the $\otimes$-positions do not satisfy the tolerance criterion, they are expected to be much smaller than the pivot positions closer to the diagonal, and it is more useful for the overall convergence to reduce those positions close to the diagonal.

[1] Due to preconditioning and monotonicity of Jacobi rotation (it increases larger and decreases smaller diagonal pivots), de Rijk's pivoting is identity most of the time.

Certainly, this modification of the row-cyclic strategy may bring no savings in the pathological case. However, it does no harm—in that case it simply reduces to the classical strategy. Also, this modification is in general not convergent, so an additional switch turns it off after at most 3 or 4 modified sweeps. After that, the classical full sweep does the final cleanup. Numerical evidence shows that in a nonpathological case the expected total number of rotations in 3 or 4 sweeps of this predict-and-skip strategy is much smaller than in one classical full sweep.

2.5. Cache-aware pivot strategy. In sections 2.1–2.4 we considered modifications to reduce the number of operations and to improve the convergence rate of the Jacobi SVD algorithm. Numerical evidence shows that those modifications, combined with the preconditioning, substantially improve the one-sided Jacobi SVD algorithm. However, the algorithm still transforms a full square array, where the basic operations (dot product and plane rotation) both have $O(n)$ operations per $O(n)$ memory references. Pivot strategies in Jacobi methods are usually not designed to enhance temporal and spatial data locality, which results in numerous cache misses, thus degrading the performance. Fortunately, a well-known tiling technique fits very nicely into our previous modifications and considerably improves data locality.

Introduce a parameter b (block size expressed in number of columns) and partition the columns of X in $\lceil n/b \rceil$ blocks (the first $\lceil n/b \rceil - 1$ blocks with b columns each, the last block with the remaining $n - b(\lceil n/b \rceil - 1)$ columns). This introduces a $\lceil n/b \rceil \times \lceil n/b \rceil$ block partition in $H = X^T X$ and the new strategy is to visit all blocks in the usual row-cyclic fashion (on the block-level), where at the beginning of the rth block row, after rotating in the diagonal block (r, r), we allow the possibility of transforming the next k diagonal blocks (and, optionally, to repeat transformations in the block (r, r)) before entering the block $(r, r + 1)$. The parameter k is a small integer (typically $0, 1, 2$) depending on X, n, b and cache parameters. It influences the convergence rate (this is easily seen by taking, for example, $k = 1$) and memory access patterns. Inside each block all positions are visited row by row. We call this strategy "tiled row-cyclic." Its detailed description is shown in Algorithm 1.

PROPOSITION 2.4. *The tiled row-cyclic strategy with $k = 0$ is equivalent to the row-cyclic strategy: both strategies compute the same matrix after the full cycle of $n(n-1)/2$ rotations. Thus, it is convergent.*

The proof is straightforward. The proof of global convergence for $k > 0$ is only a technical matter [22]. The tiled row-cycling can be immediately deployed inside the blocks of (2.5), and it can be modified following the lines of section 2.4.

3. Underflow and overflow—problems and solutions by perturbation theory. Reliable software implementation of an SVD algorithm must take care of underflow and overflow exceptions. This is particularly important for our new preconditioned Jacobi SVD algorithm (cf. [14, Algorithm 4] with the triangular Jacobi SVD as described in previous sections), because it is designed to compute the singular values in the full range of floating point numbers. For example, if $\sigma_{\max}(A) \approx 10^{30}$, $\sigma_{\min}(A) \approx 10^{-30}$, but $\min_{D=\mathrm{diag}} \kappa_2(AD)$ is moderate, then we can approximate all singular values to high relative accuracy even in single precision arithmetic (with round-off unit $\varepsilon \approx 10^{-8}$). (For comparison, bidiagonalization-based methods cannot guarantee any correct digit in the singular values below $\varepsilon\sigma_{\max}$, which is in this case approximately 10^{22}.) Jacobi SVD computation in the full range of floating point numbers requires nonstandard implementation of Jacobi rotation because it can get denormalized or flushed to identity, even in cases where its action is nontrivial [11].

In this section we discuss some other problems related to underflow and overflow.

Algorithm 1 Tiled row-cyclic strategy with tile size b.

```
{Simplified description of one full sweep}
N_bl = ⌈n/b⌉
for r = 1 to N_bl do
  i = (r − 1) · b + 1
  for d = 0 to k do {Do the blocks (r, r), ..., (r + k, r + k)}
    i = i + d · b
    for p = i to min{i + b − 1, n} do
      for q = p + 1 to min{i + b − 1, n} do
        rotate pivot pair (p, q)
      end for
    end for
  end for
  i = (r − 1) · b + 1
  for c = r + 1 to N_bl do
    j = (c − 1) · b + 1
    for p = i to min{i + b − 1, n} do
      for q = j to min{j + b − 1, n} do
        rotate pivot pair (p, q)
      end for
    end for
  end for
end for
```

The underflow and overflow thresholds are denoted by $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$, respectively. The round-off unit of the working precision is denoted by $\boldsymbol{\varepsilon}$.

3.1. Scaling against overflow. Overflow issues are resolved simply by multiplying the matrix by a suitable scalar factor. However, even this simple operation can introduce unacceptably large errors. For instance, LAPACK's driver routine xGESVD computes $\alpha = \max_{i,j} |A_{ij}|$ and scales the input matrix A with $(1/\alpha)\sqrt{\boldsymbol{\nu}}/\boldsymbol{\varepsilon}$ (if $\alpha < \sqrt{\boldsymbol{\nu}}/\boldsymbol{\varepsilon}$) or with $(1/\alpha)\boldsymbol{\varepsilon}\sqrt{\boldsymbol{\omega}}$ (if $\alpha > \boldsymbol{\varepsilon}\sqrt{\boldsymbol{\omega}}$).

Example 3.1. Take in MATLAB $A = \left(\begin{smallmatrix} 1.0e250 & 0 \\ 0 & 1.0e{-}201 \end{smallmatrix}\right)$, $d = \mathrm{diag}(A)$, $\sigma = \mathrm{svd}(A)$. A is (bi)diagonal, and its singular values are on the diagonal. However,

$$d = \begin{pmatrix} 9.999999999999999e+249 \\ 1.000000000000000e-201 \end{pmatrix}, \quad \sigma = \begin{pmatrix} 9.999999999999999e+249 \\ 1.000000000\underline{16167}e-201 \end{pmatrix}.$$

To explain this, let $\alpha = \max_{i,j} |A_{ij}|$, $\boldsymbol{\varepsilon} = eps/2$, $\boldsymbol{\omega} = realmax$, $\boldsymbol{\nu} = realmin$, $s = \boldsymbol{\varepsilon}\sqrt{\boldsymbol{\omega}}/\alpha$, and scale A with s. The singular values of sA are on its diagonal; scaling the diagonal of sA with $1/s$ changes the $(2,2)$ entry precisely to $1.000000000016167e{-}201$. Five digits in the second singular value of a 2×2 diagonal matrix are lost due to scaling $\sigma = (1/s) * (s * d)$. (In MATLAB, $\boldsymbol{\omega} \approx 1.79 \cdot 10^{308}$, $\boldsymbol{\nu} \approx 2.22 \cdot 10^{-308}$.) The problem is not removed if s is changed to the closest integer power of 2. Note that in this example $\lambda = \mathrm{eig}(A)$ returns $\lambda = d$.

Our implementation of fast scaled Jacobi rotations uses the column norms and the cosines of the angles between the columns (cf. [11]); i.e., we can compute the singular values of X even if $\sigma_{\min}(X) \approx \boldsymbol{\nu}$, $\sigma_{\max}(X) \approx \boldsymbol{\omega}$. Since the largest singular value of an $n \times n$ X is bounded by $\sqrt{n}\max_j \|X(:,j)\|_2$, it is enough to have initial X scaled so that its maximal column is not larger than $\boldsymbol{\omega}/\sqrt{n}$. Since the largest

value of any column norm of X is $\sqrt{n}\boldsymbol{\omega}$, even in the most extreme case, the scaling factor against overflow does not have to be smaller than $1/n$. In the nonextreme case $(\max_j \|X(:,j)\|_2 / \min_j \|X(:,j)\|_2 < \boldsymbol{\omega}\boldsymbol{\varepsilon}/\sqrt{n})$ we scale X to have maximal column norm at $\sqrt{\boldsymbol{\omega}}/\sqrt{n}$.

Remark 3.1. A concrete implementation of BLAS and LAPACK may contain computational routines which are optimized for speed at the cost of reduced computational range. For instance, xNRM2() is sometimes implemented as SQRT(xDOT()), which works correctly for vector norms in the range $(\sqrt{\boldsymbol{\nu}}, \sqrt{\boldsymbol{\omega}})$. In such cases, scaling of A must ensure that the maximal column is not larger than $\sqrt{\boldsymbol{\omega}}/\sqrt{n}$ in the Euclidean norm. If the spectrum of singular values spreads over the full range of normalized numbers and if all of them are wanted to high relative accuracy, then enforcing $\sqrt{\boldsymbol{\omega}}/\sqrt{n}$ as the maximum column norm may damage smallest singular values.

Remark 3.2. The one-sided Jacobi SVD can be adapted to work even beyond the range of working precision. Let $A = A_0 D_0$ with diagonal D_0, where A cannot be stored because of underflow/overflow, but both A_0 and D_0 can. Fast scaled Jacobi rotations work on the pair A_0, D_0 and deliver the result in factored form. The only required modification is that the array of scaling factors for fast rotations be initialized to $\mathrm{diag}(D_0)$ instead of a vector of ones.

3.2. An unusual underflow problem. Preliminary tests of software implementation of the preconditioned Jacobi SVD algorithm showed undesirable behavior in some cases of strongly graded matrices: the convergence of the Jacobi iterations was swift, the total number of rotations was very small, but the run time was unacceptably long. This called for detailed step-by-step analysis. Recall that the algorithm first computes[2] $AP = Q_0 \left(\begin{smallmatrix} R \\ 0 \end{smallmatrix}\right)$, then it computes the QR factorization $L = QT$ of $L = R^T$, and finally it applies our new Jacobi SVD algorithm to the lower triangular matrix $X = T^T$.

To our surprise, in some examples the second QR factorization without pivoting was considerably slower than the QR factorization with column pivoting. In these examples, L was structured, strongly graded, with deep gap in the spectrum:

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}, \text{ with } \|L_{11}^{-1}\|_2^{-1} \gg \|L_{22}\|_2 \text{ and } \|L_{21}\|_2 \ll \|L_{11}\|_2.$$

We traced the problem to computation with many denormalized floating point numbers during the computation of the factorization $L = QT$. From the block-partitioned QR factorization $L = \left(\begin{smallmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{smallmatrix}\right)\left(\begin{smallmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{smallmatrix}\right)$ we have $T_{12} = Q_{11}^T L_{11}^{-T} L_{21}^T L_{22}$ and $\|T_{12}\|_2 \leq \|Q_{11}\|_2 \|L_{11}^{-1}\|_2 \|L_{22}\|_2 \|L_{21}\|_2$. This is the well-known QR mechanism that eventually forces the $(1,2)$ block to zero, and it is clearly seen in Example 3.2.

Example 3.2. This example illustrates how denormalized numbers appear in the QR factorization. Using SGEQRF from LAPACK we compute

$$L = \begin{pmatrix} 1.0e{+}20 & 0 & 0 \\ 1.0e{-}15 & 1.0e{-}06 & 0 \\ 1.0e{-}20 & 1.0e{-}25 & 1.0e{-}21 \end{pmatrix}, \quad T = \begin{pmatrix} -0.10e{+}21 & -0.99e{-}41 & 0.00e{+}00 \\ 0 & -0.10e{-}05 & -0.99e{-}40 \\ 0 & 0 & 0.99e{-}21 \end{pmatrix}.$$

From the point of view of numerical accuracy, these denormalized numbers in the upper triangle of T are as good as zeros—backward stability and the forward errors are given with respect to column norms, and these are well preserved if the entries

[2] For simplicity, we give only one branch of the algorithm.

of the initial A are normalized numbers. Unfortunately, if the denormals are created and transformed inside an optimized black-box routine, they can cause considerable slowdown (take, e.g., $n > 1000$).

The performance of Jacobi rotations applied to a lower triangular matrix can also be degraded by the denormalized numbers. The dot products needed to compute the rotation angles can be extremely slow (many of the summands can be denormalized and are as good as zeros in the final result), and rotations with small angles during the first greedy sweep may generate additional denormalized entries where there are zeros in the upper triangle.

3.2.1. Solution by artificial perturbation. How do we deal with this problem? We can ignore it, because it is automatically solved with proper implementation of the machine arithmetic or with the *set to zero* underflow. However, it is very instructive to solve it in the framework of the imperfect arithmetic. The first natural idea is to set small off-diagonal matrix entries to zero. This has to be done by inspection after each transformation in QR factorization (or in the Jacobi iterations). This may kill the performance of highly optimized blocked code, and the underflows may still appear because they actually grow in places of zero entries. Furthermore, setting an entry to zero means perturbing the data, which may not be allowed because it causes unacceptably large perturbation. (For instance setting in Example 3.2 the element L_{31} to zero introduces large perturbation in the third row of L. In some cases this may be unacceptable.)

Now consider the opposite approach—using artificial perturbation we destroy all zeros and increase small entries. Let X be an $n \times n$ lower triangular and nonsingular matrix. The goal is to replace X with $X + \delta X$, where the perturbation δX is (i) small enough so that it does not introduce errors larger than the initial uncertainty of the SVD caused by computing X; (ii) big enough to prevent underflows in the next QR factorization or in Jacobi iterations; (iii) small enough so that it does not interfere with the convergence and that it does not prevent the use of the lower triangular structure; and (iv) small enough so that it does not preclude stable a posteriori computation of the right singular vectors.

This means that the perturbation δX has to be columnwise and also rowwise small. We use ζ to denote an appropriate threshold value used in the construction of $\delta X \equiv \delta X_\zeta$, for instance, $\zeta = \sqrt{\boldsymbol{\nu}}$, $\zeta = \sqrt{\boldsymbol{\nu}/\boldsymbol{\varepsilon}}$, or $\zeta = \boldsymbol{\varepsilon}/n$. The choice of ζ depends on $\kappa_2(X)$, from smallest values in the nonextreme case, $\kappa_2(X) < \boldsymbol{\omega\varepsilon}$, to largest in the case[3] $\kappa(X) \approx \omega^2\boldsymbol{\varepsilon}$.

DEFINITION 3.1. *For lower triangular matrix X and small positive parameter ζ, define δX_ζ as follows:*

$$(\delta X_\zeta)_{ij} = \left\{ \begin{array}{c} 0 \quad \textit{if } |X_{ij}| \geq \zeta \min\{|X_{ii}|, |X_{jj}|\} \\ -X_{ij} + \operatorname{sign}(X_{ij})\zeta \min\{|X_{ii}|, |X_{jj}|\} \quad \textit{else} \end{array} \right\} \textit{ for } i \geq j; \tag{3.1}$$

$$(\delta X_\zeta)_{ij} = -\operatorname{sign}(X_{ji})\zeta \min\{|X_{ii}|, |X_{jj}|\} \textit{ for } i < j. \tag{3.2}$$

The perturbed matrix $\tilde{X} = X + \delta X_\zeta$ is not triangular. On the other hand, efficient implementation of the first sweep of the quasi-cyclic strategy (section 2.2) is based on the triangular form. We now show that in application of the pivoting (2.5) to $\tilde{X}$ we can use the same technique as in section 2.2 and treat $\tilde{X}$ as triangular. More

[3]Extreme cases are relevant only if the singular values are well determined by the data and all wanted to high relative accuracy. Of course, such extreme cases are difficult if the machine arithmetic is not well implemented.

precisely, in the first bulk of rotations $\tilde{X}_{[3]}^{\circlearrowleft}$ we transform only $\tilde{X}_{[3]}(n_1+n_2+1:n,:)$, thus ignoring the perturbation added to $X_{[3]}(1:n_1+n_2,:)$. The perturbed zeros inside the block $\tilde{X}_{[3]}(1:n_1+n_2,:)$ are used to drown denormalized numbers during the phase $X^{[1]} \leftrightarrow X^{[2]}$. The same strategy is applied to $\tilde{X}_{[4]}^{\circlearrowleft}$, where in particular $\tilde{X}_{[4]}(1:n_1+n_2+n_3, 1:n_4)$ is treated as zero and not transformed. In other words, we ignore the perturbation δX_ζ whenever we need a triangular structure to apply (2.5) as described in section 2.2.

PROPOSITION 3.2. *The application of the first sweep of the quasi-cyclic strategy* (2.5) *to* $\tilde{X}$ *(as described above) is rowwise backward stable. Further, the perturbation of the upper triangle can be ignored in the a posteriori computation of the right singular vectors.*

Proof. To justify this manipulation with δX_ζ, we first note that it contains tiny rowwise relative perturbations X. Indeed, for $i<j$ we have

$$\frac{|(\delta X_\zeta)_{ij}|}{\|X(i,:)\|_2} = \zeta \frac{\min\{|X_{ii}|, |X_{jj}|\}}{\|X(i,:)\|_2} \leq \zeta, \text{ where in case of a graded matrix } \frac{|X_{jj}|}{\|X(i,:)\|_2} \ll 1.$$

The key argument is shown for the simplest (n_1, n_2) block partition of $\tilde{X}$,

$$\tilde{X} = \begin{pmatrix} \tilde{X}_{11} & \tilde{X}_{12} \\ \tilde{X}_{21} & \tilde{X}_{22} \end{pmatrix} = \begin{pmatrix} \tilde{X}_{11} & 0 \\ \tilde{X}_{21} & \tilde{X}_{22} \end{pmatrix} + \begin{pmatrix} 0 & \tilde{X}_{12} \\ 0 & 0 \end{pmatrix} \equiv \tilde{X}_{\llcorner} + \delta\tilde{X}_{\llcorner}.$$

Obviously, $\|\tilde{X}_{12}(i,:)\|_2 \leq \sqrt{n_2}\zeta\|X(i,:)\|_2$ for all $i=1,\ldots,n_1$. Suppose we transform the last n_2 columns of $\tilde{X}$, following some pivot strategy, but we simply do not reference $\tilde{X}_{12}$ (that is, we work on $\tilde{X}_{\llcorner}$ and use its triangular structure). The computed matrix is represented by the backward perturbation analysis as

$$\begin{aligned} \tilde{X}' &= \begin{pmatrix} \tilde{X}_{11} & 0 \\ \tilde{X}_{21} & \tilde{X}_{22}+\delta\tilde{X}_{22} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \hat{W} \end{pmatrix} + \begin{pmatrix} 0 & \tilde{X}_{12} \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \tilde{X}_{11} & \tilde{X}_{12}\hat{W}^T \\ \tilde{X}_{21} & \tilde{X}_{22}+\delta\tilde{X}_{22} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \hat{W} \end{pmatrix}, \quad \hat{W}^T\hat{W} = I, \end{aligned}$$

where $\|\delta\tilde{X}_{22}(i,:)\|_2 \leq O(n)\boldsymbol{\varepsilon}\|\tilde{X}_{22}(i,:)\|_2$, $i=1,\ldots,n_2$, which means that ignoring $\tilde{X}_{12}$ is equivalent to replacing it with $\tilde{X}_{12}\hat{W}^T$ and then applying $\hat{W}$. Since right-handed orthogonal transformation does not change the row-norms of the involved matrices, the rowwise backward stability is preserved. ∎

If we need only to compute the QR factorization of X, or to compute only the singular values, then we can allow even bigger δX_ζ. Each X_{ij} in the lower triangle with the property $|X_{ij}| < \zeta|X_{jj}|$ is replaced with $\mathrm{sign}(X_{ij})\zeta|X_{jj}|$, and thus $(\delta X_\zeta)_{ij} = -X_{ij} + \mathrm{sign}(X_{ij})\zeta|X_{jj}|$ for all $i>j$. Simultaneously, the position X_{ji} in the upper triangle is set to $(\delta X_\zeta)_{ji} = -\mathrm{sign}(X_{ij})\zeta|X_{ii}|$. Note that $\|\delta X_\zeta(:,j)\|_2 \leq \sqrt{n}\zeta|X_{jj}|$, i.e., the perturbation is columnwise small. Since the matrix X is computed in the QR factorization with pivoting, the condition number of column scaled X is moderate, which means that computations with X and $X+\delta X_\zeta$ will give equally good singular value approximations.

4. Numerical testing. In this section we present software implementation of the new algorithm described in [14, Algorithm 4] and in this report. We give the results of preliminary testing of the algorithm with respect to numerical accuracy and efficiency (run times compared with those of existing algorithms). Our goal

is numerically reliable software implemented to reach a reasonable fraction of the efficiency of the less accurate bidiagonalization-based methods. In other words, we want to make the high accuracy of the Jacobi SVD algorithm so affordable that the new algorithm becomes attractive as one of the methods of choice for dense full SVD computation.

Carefully designed testing of the software is also a test of the theory. It shows how sharp the theoretical bounds are and also gives new insights into the cases of input matrices which are on the boundaries of the theoretical assumptions. Good test cases give insights into the behavior of the algorithm and may induce modifications which improve the efficiency of the algorithm. The feedback loop created in this way is part of the research process. In fact, the material of section 3 is the result of numerical tests of an early version of the code. Further, during the tests of our code we found serious problem in the LAPACK implementations xGEQPF and xGEQP3 of the QR factorization with column pivoting; see [13] for a provably stable implementation.

We test single precision (32-bit representation, $\boldsymbol{\varepsilon} \approx 5.3 \cdot 10^{-8}$) implementation. It is always assumed that the nonzero entries of input matrix are normalized floating point numbers.

4.1. Measuring error—distance to what? One difficulty in testing a new SVD software on a large set of pseudorandom matrices is how to provide reference (exact) values of Σ, U, and V, which are used to estimate the accuracy of the computed approximations $\tilde{\Sigma}$, $\tilde{U}$, $\tilde{V}$. One could start by generating pseudorandom numerically orthogonal $\overline{U}$, diagonal $\overline{\Sigma}$, and numerically orthogonal $\overline{V}$ and then define $A = \mathit{computed}(\overline{U}\ \overline{\Sigma}\ \overline{V}^T)$. However, the numerical SVD of A may be very much different from $\overline{U}\ \overline{\Sigma}\ \overline{V}^T$. Using the same algorithm in higher precision is useful but not always—depending on the matrix, it is possible that both procedures compute with large errors. The alternative is to use existing, tested(!), and trusted double precision software to compute the SVD of a given test matrix A. In our case, this means DGESVD and/or DGESDD[4] from LAPACK, but this will be useful only as long these procedures guarantee at least eight digits of accuracy, that is, for (roughly) $\kappa(A) < 1/\boldsymbol{\varepsilon} \approx 10^8$. Our choice of the reference procedure is the classical one-sided Jacobi SVD with de Rijk's [9] pivoting, implemented in double precision.

If $\tilde{\sigma}_1 \geq \cdots \geq \tilde{\sigma}_n$ and $\hat{\sigma}_1 \geq \cdots \geq \hat{\sigma}_n$ are the computed and the reference singular values computed in higher precision, then the forward errors of interest are[5]

$$\mathbf{e}_i = \frac{|\tilde{\sigma}_i - \hat{\sigma}_i|}{\hat{\sigma}_i}, \quad i = 1, \ldots, n, \quad \mathbf{e} = \max_{i=1:n} \mathbf{e}_i. \tag{4.1}$$

4.2. Test matrices. Our primary targets are the matrices of the form $A = BD$, where D is diagonal and B is well conditioned with equilibrated (unit in Euclidean norm) columns. In that case the relative error in the output is governed by the condition number $\kappa(B)$ independent of D. To illustrate this property we need to generate test matrices $A = BD$, where B has given $\kappa(B)$ and unit columns. Moreover, the matrices should be generated so systematically that the maximal measured forward errors attain the predicted theoretical bounds, and that experimental data show that no accuracy can be guaranteed if the assumptions of the theory are not satisfied. In

[4] During the testing we accidentally found an example of serious failure of the DGESDD procedure from the SUN performance library—a ghost singular value of the size of the largest one appeared in the dominant part of the spectrum.

[5] Here by definition $0/0 = 0$.

that case we will have experimental evidence that both the theory and the numerical testing are done properly.

We use the algorithm of Stewart [29] to generate random orthogonal matrices distributed uniformly with respect to the Haar measure over the orthogonal group $\mathcal{O}(n)$. If W_1 and W_2 are two such matrices, and if S is diagonal with given condition number $\kappa(S)$, we compute $C = W_1 S W_2$. Then we use the fact that for the matrix $C^T C$ there always exists an orthogonal W_3 such that the diagonal entries of $W_3^T (C^T C) W_3$ are all equal to $\mathrm{Trace}(C^T C)/n$. Then the matrix $B = CW_3$ has equilibrated columns and condition number κ. If we generate diagonal D, then $A = BD$. There are several ways to generate the matrix W_3; see, e.g., [5], [7]. The distributions of the diagonal entries of S and D can be chosen in different ways. We use the modes provided in the LAPACK test matrix generators (parameter MODE in DLATM1), and the chosen modes are denoted by $\mu(S)$ and $\mu(D)$. Thus, each generated matrix A has four parameters, $p(A) = (\kappa(S), \mu(S), \kappa(D), \mu(D))$. For each fixed $p(A)$, we use three different random number generators provided in the LAPACK testing library (LAPACK/TESTING/MATGEN/), and with each of them we generate a certain number of samples (test matrices). In this way, we have an automated generator of pseudorandom matrices with certain relevant parameters varying systematically in a given range; for instance, $\kappa(S)(= \kappa(B))$ is set to take the values 10, 10^2, $10^3, \ldots, 10^8$.

4.3. Test results. Our new algorithm is implemented in a LAPACK-style routine SGEPVD, which is in an early stage of development. We have done no serious profiling in order to optimize it for a particular architecture. The bulk of the code (rotations) is still on BLAS 1 level, and we have plans to change this in near future. Nevertheless, the obtained results are surprisingly good and encouraging. The results presented in this report were obtained on an HP X2100 workstation (1.9GHz Pentium 4, 1Gb RAM, 8 Kb L1 cache, 256 Kb L2 cache), using GNU FORTRAN compiler F77 3.3.1 under SuSE Linux. The LAPACK library is compiled from the source code and linked with the BLAS library from Intel's MKL 6.1.

4.3.1. Computing the full SVD. The test matrices of the form $A = BD$ are generated as follows. We take $A = ((W_1 S W_2) W_3) D$ as described in section 4.2, and $\kappa(S) = 10^i$, $i = 1, \ldots, 8$, and $\kappa(D) = 10^{2j}$, $j = 0, \ldots, 7$. For each fixed pair (i, j) we generate diagonal S and D each with four different distributions of the diagonal entries (as specified by the parameter MODE). This gives for each fixed $(\kappa(S) = 10^i, \kappa(D) = 10^{2j})$ 16 different types of matrices, giving a total of $64 \cdot 16 = 1024$ classes. The matrices are generated in four nested loops; the outer loop controls $\kappa(B) = \kappa(S)$. Hence, the matrices are divided into eight groups with fixed $\kappa(B)$.[6] Finally, we choose the row and the column dimensions, m and n, and the test procedure is ready.

Once we compute $A \approx \tilde{U} \tilde{\Sigma} \tilde{V}^T$, $\tilde{\Sigma} = \mathrm{diag}(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$, we can immediately estimate the quality of the computed decomposition using the following computed quantities:

$$\mathbf{r} = \textit{computed}\left(\frac{\|A - \tilde{U}\tilde{\Sigma}\tilde{V}^T\|_F}{\|A\|_F}\right) \quad \text{(should be at most } f(m,n)\boldsymbol{\varepsilon},\ f \text{ moderate)};$$

$$\mathbf{o}_U = \textit{computed}\left(\max_{i,j} |(\tilde{U}^T\tilde{U})_{ij} - \delta_{ij}|\right) \quad \text{(should be at most } O(m\boldsymbol{\varepsilon})\text{)};$$

$$\mathbf{o}_V = \textit{computed}\left(\max_{i,j} |(\tilde{V}^T\tilde{V})_{ij} - \delta_{ij}|\right) \quad \text{(should be at most } O(n\boldsymbol{\varepsilon})\text{)}.$$

[6] This helps in the interpretation of the results of the experiments and explains the shapes of the graphs on the figures given here.

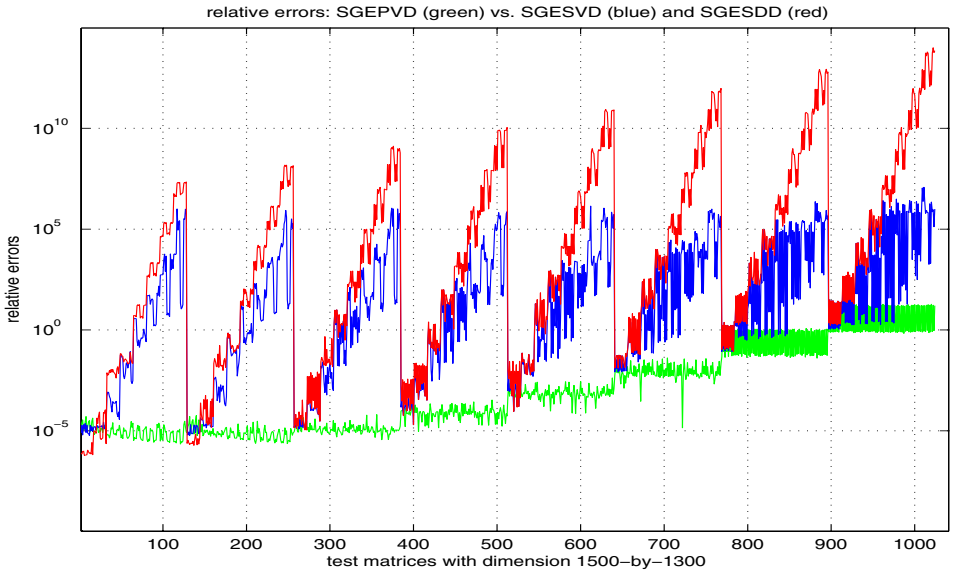

FIG. 4.1. *Maximal relative errors in the computed singular values for* 1500×1300 *matrices. The top curve (worst case) describes the accuracy of SGESDD. The middle curve represents the errors of SGESVD, and the lowest curve (smallest relative errors) belongs to SGEPVD.*

These measures are useful to test the correctness of the code and backward stability in the matrix norm sense. It is easy to show that $\mathbf{r}$, $\mathbf{o}_U$, and $\mathbf{o}_V$ can be computed sufficiently accurately (best using higher precision) to be used as relevant measures of the quality of the computed decomposition. Thus, the standard error bound can be a posteriori numerically checked. Our procedure, called SGEPVD,[7] has successfully passed all three tests; $\mathbf{r}$, $\mathbf{o}_U$, $\mathbf{o}_V$ were in the allowed ranges. Note that SGEPVD returns the singular vectors numerically orthogonal up to the theoretical bound $m\varepsilon$ ($n\varepsilon$), which, as in other algorithms dealing with numerical orthogonality, for large m and n may not look satisfactory in single precision ($\varepsilon \approx 10^{-7}$ and, e.g., $m = n = 4000$).

Before we go over to the comparison of SGEPVD with SGESVD and SGESDD from LAPACK, we should point out that SGEPVD computes the SVD to higher accuracy and also provides an estimate of the maximal relative error by computing an approximation of $\|B^{\dagger}\|_2$.

We show only two out of the many tests performed during code development. Or first test ran with $m = 1500$, $n = 1300$, and with one pseudorandom matrix in each class. Compare the maximal relative errors in computed singular values for all 1024 test cases, shown on Figure 4.1. It is clearly seen that the accuracy of SGEPVD depends on $\kappa(B)$, while the other two methods depend on $\kappa(A)$. Any SVD algorithm that starts with bidiagonalization is at risk to have error behaving like the SGESVD and SGESDD errors in Figure 4.1. The best caption for this figure is the title of [8]. Note that SGESVD returns much better results than SGESDD. To the best of our knowledge, this fact is not mentioned elsewhere in the literature. Also, note the considerable upward bias in the relative errors of the bidiagonalization-based procedures (cf. [30]).

The timings for this example are shown on Figure 4.2. We immediately note that the new Jacobi SVD algorithm is not that much slower than the bidiagonalization-

[7] PVD is the acronym for principal value decomposition, an old name for SVD.

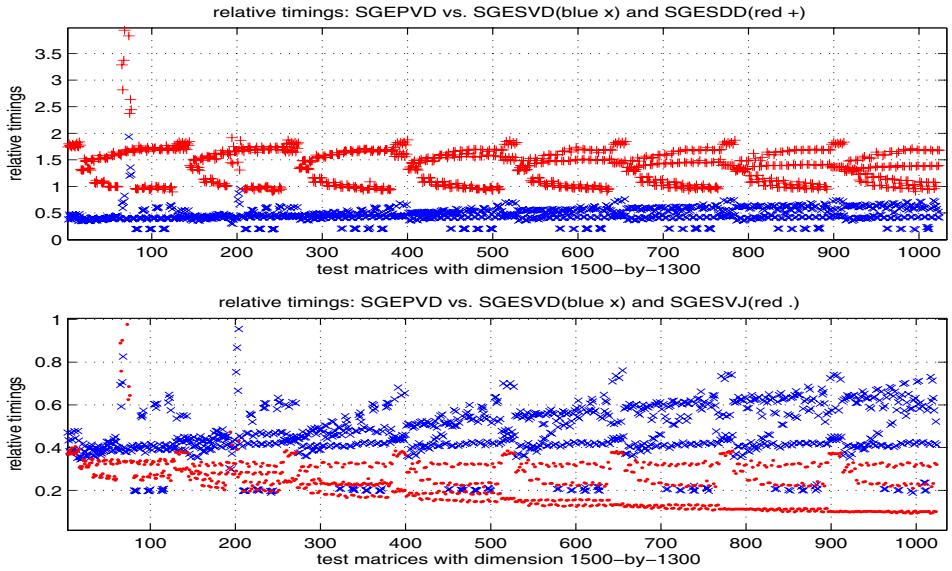

FIG. 4.2. *Computing full SVD: Relative timings for* 1500×1300 *matrices on a Pentium* 4 *machine with Intel MKL* 6.1 *library. In the first plot, the crosses denote* $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVD}}$ *and the pluses* $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESDD}}$. *The second plot shows* $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVD}}$ *(crosses) and* $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVJ}}$ *(dots). See Remark* 4.1.

based fast methods. In fact, it outperforms the QR algorithm and is on average less than twice as slow as the divide-and-conquer algorithm. The worst-case performance for SGEPVD is on matrices with weak column scaling and a singular spectrum composed of many tight clusters (examples above the 1.5 mark on Figure 4.2). In all other cases the time of SGEPVD is on average 1.5 times the time of SGESDD. Here, again, we stress the fact that the results obtained by SGEPVD enjoy much better numerical properties and that the time of SGEPVD includes computed error bound—better results and additional information are computed in reasonable time. Thus, for a fair comparison one should consider both Figures 4.1 and 4.2 before deciding which algorithm is the better choice for a particular application. The second plot on Figure 4.2 shows $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVD}}$ and $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVJ}}$, where SGESVJ denotes our careful implementation (cf. [11]) of de Rijk's [9] one-sided Jacobi SVD. (De Rijk's one-sided Jacobi SVD is much more efficient than other classical cyclic Jacobi algorithms.) It is interesting that de Rijk reported superior performance of his Jacobi SVD code over the SSVDC routine from LINPACK (Golub–Reinsch SVD) on CYBER 205. Note that our code SGEPVD runs up to ten times faster than SGESVJ.

In the second test we have 500×350 matrices, with two examples in each of 1024 classes. The results are given in Figure 4.3.

Remark 4.1. Note the few outliers above mark 2 on Figures 4.2 and 4.3. They correspond to matrices on which SGEPVD actually performed very well, with a low number of rotations and swift convergence. (We checked this by inspecting the details of those particular runs. In fact, on these matrices even the classical one-sided Jacobi, usually much slower, comes close to our new method.) However, since our threshold for perturbation used to trap denormals was set to $\zeta = \sqrt{\boldsymbol{\nu}} \approx 10^{-19}$, some of them were

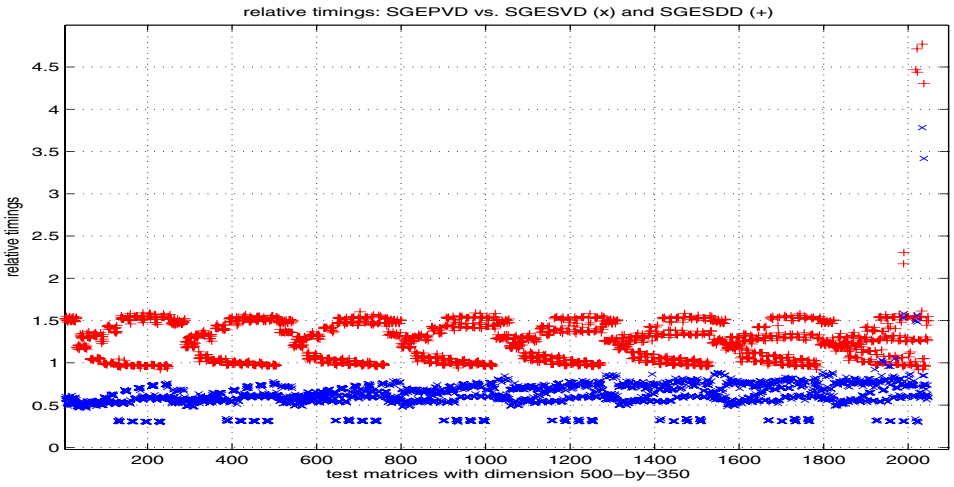

FIG. 4.3. *Computing full SVD: Relative timings for* 500×350 *matrices on a Pentium* 4 *machine with Intel MKL* 6.1 *library. The crosses denote* $time_{\text{SGEPVD}}/time_{\text{SGESVD}}$ *and the pluses are* $time_{\text{SGEPVD}}/time_{\text{SGESDD}}$.

not captured, and imperfect denormalized arithmetic caused considerable slowdown. We used this value of ζ to illustrate the problem. In practice the threshold can be set higher; e.g., if in those cases $\zeta = \varepsilon/n$, or if *set to zero* underflow is in function, then the run time reflects the actual flop count and the outliers disappear.

Remark 4.2. We have noted that using a double accumulated dot product in preparation of Jacobi rotation reduces the total number of rotations, especially in cases of multiple or tightly clustered singular values. Unfortunately, in the MKL BLAS library DSDOT performs poorly in comparison to SDOT, and the savings in number of rotations do not reduce the total run time.

Remark 4.3. The results may vary on the same machine with different BLAS libraries. We note that with the GOTO BLAS [18], [19] all routines run faster (as compared to the MKL library), but the relative speedup is smallest in our algorithm. This is because GOTO BLAS 3 outperforms MKL 6.1 BLAS 3, but GOTO BLAS 1 is no match for really efficient MKL 6.1 BLAS 1. Since our code still depends on BLAS 1 (dot products and plane rotations), switching to GOTO BLAS has mixed consequences. A hybrid of the two libraries would be much better for our algorithm.

4.3.2. Computing only Σ. We have established that the singular values are computed by the new algorithm as predicted by the theory—our variant of the Jacobi SVD complies with [8], [10]. In the previous section we showed that it computes the full SVD with efficiency comparable to fast bidiagonalization-based approaches. If only the singular values are needed, reaching a similar level of relative efficiency seems to be mission impossible, for we would need a Jacobi-based algorithm that computes Σ in time comparable to the time needed to bidiagonalize the matrix! However, the results of [14, Algorithm 1] with the rank revealing QR factorization [4], [3] are encouraging.

The test matrices of the form $A = BD$ are generated as in section 4.3.1, with 2048 examples in each test run. We show the results of two tests; in the first the matrices were 500×350 and in the second $m = 1000$, $n = 700$. The maximal measured relative errors $\mathbf{e}$ are not shown because they behave as shown in section 4.3.1. In Figure 4.4

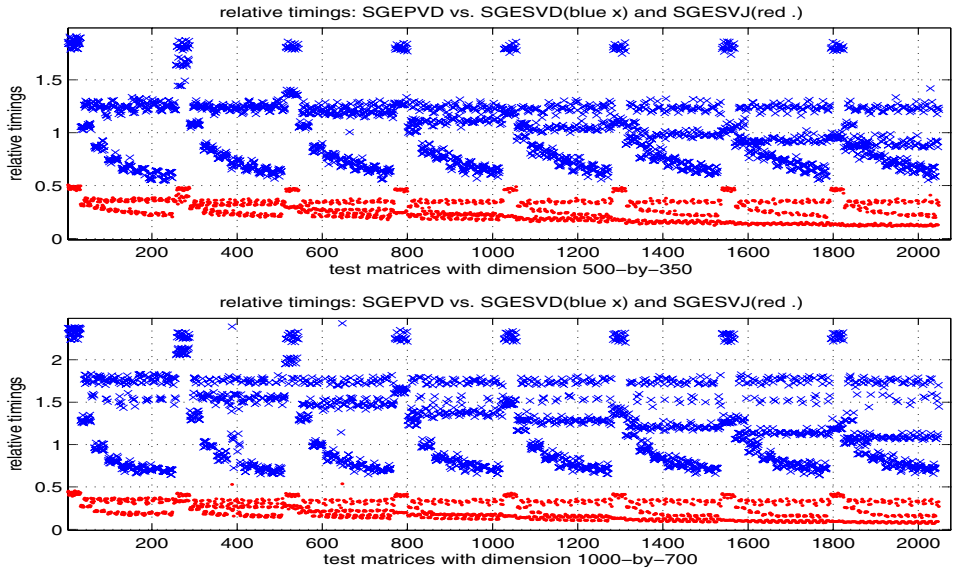

FIG. 4.4. *Computing only* Σ*: Relative timings for* 500×350 *and* 1000×700 *matrices on a Pentium* 4 *machine with Intel MKL* 6.1 *library. The crosses denote* $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVD}}$ *and the dots are* $time_{\mathrm{SGEPVD}}/time_{\mathrm{SGESVJ}}$.

we display the relative timings and compare SGEPVD with SGESDD (or SGESVD) and with the classical one-sided Jacobi SVD with de Rijk's pivoting (SGESVJ). The speedup of the new algorithm over SGESVJ ranges from a factor of 2 up to a factor of 15. (We note that SGEPVD computes Σ and also an estimate of the scaled condition number.) It is interesting to note that in cases of well separated singular values (especially in cases of badly scaled matrices) the new method computes all singular values faster than bidiagonalization-based methods. In less favorable cases of clustered singular values the new method is slower with a factor of up to 2.43, depending on the input matrix.

5. Conclusion and future work. The most important message of [14] and this paper is that the question of the ultimate dense nonstructured full SVD method is still open. What is desired is the most efficient algorithm capable of computing the SVD to optimal (numerically feasible) accuracy warranted by the data. Because the one-sided Jacobi SVD is more accurate than any other algorithm that first bidiagonalizes the matrix, in our quest for the ultimate dense nonstructured SVD algorithm we follow the Jacobi idea. We have enhanced the basic Jacobi SVD algorithm with a preconditioning stage and new iterative scheme for preconditioned triangular matrices.

Our results show that the new preconditioned Jacobi-type SVD algorithm can be competitive in efficiency with fast bidiagonalization-based methods (QR and divide-and-conquer algorithms) without trading accuracy for speed. *In fact, the new algorithm computes the SVD more accurately and almost twice as fast as the SGESVD from LAPACK, and the speedup factor over the equally accurate standard one-sided Jacobi SVD (enhanced with de Rijk's pivoting) typically ranges between three and ten.* (See Figures 4.1 and 4.2.) The new Jacobi SVD algorithm remains slower (with factors ranging from 1 to 1.75) than the less accurate divide-and-conquer SVD (SGESDD from LAPACK). If only the singular values are computed, our method can be up to

fifteen times faster than the standard one-sided Jacobi SVD, and it is up to twice as slow as bidiagonalization-based singular value computation. (In our tests with dimensions up to 1000 the efficiency was within factors 0.63–2.43 of SGESVD and SGESDD.)

The new algorithm has an advantage over bidiagonalization-based methods in cases of matrices with low numerical rank, because it starts with a rank revealing factorization (cf. [23]). (Bidiagonalization is not rank revealing unless enhanced with pivoting, which would make it more expensive.) In fact, in cases of low numerical rank, we can even outperform SGESDD. If $m \gg n$, both methods start with the QR factorization, which is the most expensive part of the computation, and in that case the efficiency of all three methods is about the same. Also, if only the standard absolute accuracy is required, then Jacobi iterations can be controlled by a loosened stopping criterion, thus allowing satisfactory approximation with less computational effort.

However, our algorithm is not yet fully optimized and several issues are the subject of our current and future work. The most expensive part of our method are BLAS 1 Jacobi rotations, which means that there is more potential for optimization. One simple improvement will be possible once optimized combinations of AXPY and DOT, and combination of two linked AXPY operations are available in single calls.

Nontrivial improvement will be obtained by using block rotations. We expect that the fast scaled block rotations designed by Hari [21] will fully exploit the potential of our approach and considerably improve the performance, especially in difficult cases of tightly clustered singular values. Namely, in such cases the preconditioner does not perform well, which means that one or two full sweeps of rotations are needed to compensate the lack of proper preconditioning. Such heavy computation on a BLAS 1 level is clearly seen in the numerical results presented in section 4 (see, e.g., the crosses above 1.5 and 2 on Figure 4.4).

Further, the overhead of column pivoting in BLAS 3 optimized QR factorization (SGEQP3 from LAPACK) is still too big. Any future improvement of the column-pivoted QR factorization contributes to the efficiency of our algorithm. Other issues include new rank revealing QR factorization and using shifts in the second QR (and third in some cases) factorization if only classical absolute error bounds are required. Our ultimate goal is high performance LAPACK-style software. The question is whether or not we will be able to reach the efficiency of SGESDD. Time will tell.

Acknowledgments. The authors acknowledge generous support by the Volkswagen Science Foundation and the Croatian Ministry of Science and Technology. They are also indebted to P. Arbenz (Zürich), J. Barlow (State College), J. Demmel (Berkeley), F. Dopico (Madrid), V. Hari (Zagreb), W. Kahan (Berkeley), J. Moro (Madrid), B. Parlett (Berkeley), and I. Slapničar (Split) for their comments, criticisms, and many fruitful discussions. Special thanks go to the anonymous referees for their substantial and constructive suggestions.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenny, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, 2nd ed., SIAM, Philadelphia, 1995.

[2] J. Barlow and J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, SIAM J. Numer. Anal., 27 (1990), pp. 762–791.

[3] C. H. Bischof and G. Quintana-Orti, *Algorithm* 782: *Codes for rank-revealing QR factorizations of dense matrices*, ACM Trans. Math. Software, 24 (1998), pp. 254–257.

[4] C. H. Bischof and G. Quintana-Orti, *Computing rank-revealing QR factorizations of dense*

matrices, ACM Trans. Math. Software, 24 (1998), pp. 226–253.
[5] N. N. Chan and K.-H. Li, *Diagonal elements and eigenvalues of a real symmetric matrix*, J. Math. Anal. Appl., 91 (1983), pp. 562–566.
[6] S. Chandrasekaran and I. C. F. Ipsen, *Analysis of a QR algorithm for computing singular values*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 520–535.
[7] P. I. Davies and N. J. Higham, *Numerically stable generation of correlation matrices and their factors*, BIT, 40 (2000), pp. 640–651.
[8] J. Demmel and K. Veselić, *Jacobi's method is more accurate than QR*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1204–1245.
[9] P. P. M. de Rijk, *A one-sided Jacobi algorithm for computing the singular value decomposition on a vector computer*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 359–371.
[10] Z. Drmač, *Computing the Singular and the Generalized Singular Values*, Ph.D. thesis, Lehrgebiet Mathematische Physik, Fernuniversität Hagen, Germany, 1994.
[11] Z. Drmač, *Implementation of Jacobi rotations for accurate singular value computation in floating point arithmetic*, SIAM J. Sci. Comput., 18 (1997), pp. 1200–1222.
[12] Z. Drmač, *A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm*, IMA J. Numer. Anal., 19 (1999), pp. 191–213.
[13] Z. Drmač and Z. Bujanović, *On the failure of rank revealing QR factorization software—a case study*, ACM Trans. Math. Software, to appear.
[14] Z. Drmač and K. Veselić, *New fast and accurate Jacobi SVD algorithm.* I, SIAM J. Matrix Anal. Appl., 29 (2008), pp. 1322–1342.
[15] K. V. Fernando and B. N. Parlett, *Implicit Cholesky algorithms for singular values and vectors of triangular matrices*, Numer. Linear Algebra Appl., 2 (1995), pp. 507–531.
[16] A. George, K. Ikramov, and A. B. Kucherov, *Some properties of symmetric quasi-definite matrices*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1318–1323.
[17] H. H. Goldstine, H. H. Murray, and J. von Neumann, *The Jacobi method for real symmetric matrices*, J. Assoc. Comput. Mach., 6 (1959), pp. 59–96. (Also in Collected Works, Vol. V, J. von Neumann, ed., Pergamon Press, New York, 1973, pp. 573–610.)
[18] K. Goto, http://www.cs.utexas.edu/users/kgoto/, 2004.
[19] K. Goto and R. van de Geijn, *On Reducing TLB Misses in Matrix Multiplication*, Tech. report TR-2002-55, FLAME Working Note 9, Department of Computer Science, The University of Texas at Austin, Austin, TX, 2002.
[20] V. Hari, *On sharp quadratic convergence bounds for the serial Jacobi methods*, Numer. Math., 60 (1991), pp. 375–406.
[21] V. Hari, *Accelerating the SVD block-Jacobi method*, Computing, 75 (2005), pp. 27–53.
[22] V. Hari, *Convergence of a block-oriented quasi-cyclic Jacobi method*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 349–369.
[23] V. Hari and K. Veselić, *On Jacobi methods for singular value decompositions*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 741–754.
[24] M. R. Hestenes, *Inversion of matrices by biorthogonalization and related results*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 51–90.
[25] C. G. J. Jacobi, *Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen*, Crelle's Journal für reine und angew. Math., 30 (1846), pp. 51–95.
[26] W. F. Mascarenhas, *On the convergence of the Jacobi method for arbitrary orderings*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1197–1209.
[27] R. Mathias and G. W. Stewart, *A block QR algorithm for singular value decomposition*, Linear Algebra Appl., 182 (1993), pp. 91–100.
[28] N. H. Rhee and V. Hari, *On the global and cubic convergence of a quasi-cyclic Jacobi method*, Numer. Math., 66 (1993), pp. 97–122.
[29] G. W. Stewart, *The efficient generation of random orthogonal matrices with an application to condition estimators*, SIAM J. Numer. Anal., 17 (1980), pp. 403–409.
[30] G. W. Stewart, *Perturbation Theory for the Singular Value Decomposition*, Tech. report UMIACS-TR-90-124, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 1990.
[31] G. W. Stewart, *A Gap-Revealing Matrix Decomposition*, Tech. report TR-3771, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 1997.
[32] G. W. Stewart, *The QLP Approximation to the Singular Value Decomposition*, Tech. report TR-97-75, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 1997.
[33] K. Veselić and V. Hari, *A note on a one-sided Jacobi algorithm*, Numer. Math., 56 (1989), pp. 627–633.