NEW FAST AND ACCURATE JACOBI SVD ALGORITHM: II. LAPACK WORKING NOTE 170

ZLATKO DRMAČ* AND KREŠIMIR VESELIĆ†

Abstract. This paper presents new implementation of one–sided Jacobi SVD for triangular matrices and its use as the core routine in a new preconditioned Jacobi SVD algorithm, recently proposed by the authors. New pivot strategy exploits the triangular form and uses the fact that the input triangular matrix is the result of rank revealing QR factorization. If used in the preconditioned Jacobi SVD algorithm, described in the first part of this report, 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 machine arithmetic on some systems.

Key words. Jacobi method, singular value decomposition, eigenvalues

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

1. Introduction. Jacobi iteration is one of the time-honored methods to compute 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 the 19'th century computing matador 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 [23] is a very 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 (it may even be used to prove the existence of the spectral decomposition) but also computational, in this aspect it may well be compared with the Gaussian elimination. Thus, upon coming of automatic computation Jacobi method was soon rediscovered by Goldstine, Murray and von Neumann [17] who provided first detailed implementation and error analysis.

In our recent work [16] we reviewed Hestenes variant [22] of the Jacobi method for SVD computation of general matrix A by implicit diagonalization of $H = A^T A$. Briefly, if $V^T HV = \Lambda$, then $AV = U\Sigma$, where $\Sigma = \sqrt{\Lambda}$ is diagonal and U is orthogonal. Orthogonal matrix V is the limit product of Jacobi rotations. In [16] we also have presented a novel approach to the Jacobi SVD method with the goal of making the method more efficient, while retaining superior numerical properties. We have shown that rank revealing QR factorization can serve as versatile preconditioner which enables efficient execution of the iterative part (Jacobi rotations) on a triangular matrix with particular diagonal dominance structure.

The idea of QR iterations as preconditioner for SVD computation is well known, but thus far it was not fully exploited in the context of Jacobi method. It is both simple and powerful: If $AP = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$, and $R^T P_1 = Q_1 R_1$ are rank revealing QR factorizations of A and R^T respectively, then the Hestenes one–sided Jacobi algorithm

*Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia. The work of the author is supported by the Croatian Ministry of Science and Technology under grant 0037120 (*Numerical Analysis and Matrix Theory*), and by the Volkswagen–Stiftung grant *Designing Highly Accurate Algorithms for Eigenvalue and Singular Value Decompositions*.

†Lehrgebiet Mathematische Physik, Fernuniversität Hagen, Postfach 940, D–58084 Hagen, Germany. The work of the author is supported by the Volkswagen–Stiftung grant *Designing Highly Accurate Algorithms for Eigenvalue and Singular Value Decompositions*.

applied to $X = R^T$ or $X = R_1^T$ converges much faster than applied to A. This iterative part on triangular matrix is in [16] 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 V is obtained not from the accumulated product of Jacobi rotations but rather a posteriori, based on the relation $V = X^{-1}X_\infty$. Assembling the SVD of A is straightforward.

In this report we unwrap the black–box and show how it performs in the framework of the new preconditioned Jacobi SVD algorithm [16]. In that sense, this report is a continuation of [16], and it is organized as follows.

In §2 we describe a new pivot strategy for the Hestenes one–sided Jacobi SVD on triangular matrices. It is based on the present knowledge of the asymptotic convergence of Jacobi iterations. We use triangular and scaled diagonal dominance structures to reduce flop count and memory traffic, as well as to achieve faster convergence. It should be stressed here that the global structure of the algorithm in [16] is such that we are completely free to choose pivot strategy (for both serial and parallel computing, open for blocking to enhance efficient use of cache memory) and have all numerical properties described in [16]. The results of detailed numerical testing of mathematical software implementing the new algorithm, as well as modifications prompted by the tests are presented in §3. The material of §3.2.1, 3.2.2 is an example of that part of scientific computing where we have to deal with the peculiarities of the floating point arithmetic. One has to use many little tricks and facts that perhaps are not worth to be wrapped up in forms of propositions, not to mention theorems, but all of them together make a big difference. The results presented in 3.3 carry the main message of [16] and this report: *Our new SVD algorithm is numerically superior to the bidiagonalization based QR (SGESVD) and divide and conquer (SGESDD) algorithms from LAPACK [1], and considerably faster than the previous implementations of the (equally accurate) Jacobi SVD method. Moreover, the new algorithm can compute the SVD faster than SGESVD and not much slower than SGESDD.* These facts open many interesting question about the future development of SVD computation. Comments and discussion are given in §4.

The authors acknowledge generous support by the Volkswagen Science Foundation and the Croatian Ministry of Science and Technology. We 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), I. Slapničar (Split) for their comments, criticisms and many fruitful discussions.

2. One–sided Jacobi on preconditioned triangular matrices. Since in our algorithm Jacobi iterations start with preconditioned matrix with special structure ($n \times n$, triangular with certain diagonal dominance properties), pivot strategy should not blindly control the order of rotations following hardwired rule. Instead, pivoting should be highly adaptive, with optimal use of current structure of the matrix. Recall that the transformation $X^{(k+1)} = X^{(k)}V^{(k)}$ transforms pivot columns p_k, q_k following pivot strategy $k \mapsto (p_k, q_k)$. 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 $\boldsymbol{\varepsilon}$. Else, the rotation is skipped. If $n(n-1)/2$ consecutive rotations with all possible pivot pairs are skipped, the iterations are stopped and numerical convergence is declared.

We have several structural adaptations in mind. First, we will exploit triangular form and avoid trivial computations with zero blocks as long as possible. This will save unnecessary computation before eventual fill in. Secondly, proper pivoting could lead to faster convergence, given the fact that the matrix has been preconditioned. This means that the matrix nearly satisfies certain asymptotic assumptions for higher order convergence (quadratic or even cubic) of some pivot strategies, and that the positions of the largest off–diagonal entries are predictable, see [16]. The question is how to choose pivot strategy to exploit this structure.

Next, since one dot product is necessary to determine if pivot columns should be rotated, dot products not followed by rotation are the overwhelming cost of the later stages as the algorithm approaches numerical convergence. Proper pivot strategy should monitor and use the structure of the matrix to predict (and thus avoid) many unnecessary dot products.

Complete description and full explanation of our new pivot strategy is rather technical, and we will not present it here with all details and subtleties. Instead, we will just sketch the main ideas and discuss relevant issues.

2.1. Review of asymptotic convergence. For the reader's convenience, we first offer a brief review of the main facts about the asymptotic convergence of the Jacobi method. Understanding asymptotic behavior of the Jacobi process helps in devising efficient stoping criteria and it also indicates which mechanisms create higher order convergence and thus helps in constructing better pivot strategies.

For simplicity, we state all results in equivalent form of diagonalization of symmetric matrices. Our symmetric matrix H will always be given implicitly as $H = X^T X$, and the Jacobi rotations are applied to X from the right–hand side in usual way. As basic strategy we take the row–cyclic one, which is periodic and in one full sweep of $n(n-1)/2$ rotations it rotates row–by–row at the pivot positions $(1,2), (1,3), \ldots (1,n)$; $(2,3), \ldots, (2,n)$; $(3,4), \ldots, (3,n); \ldots, (n-2,n)$; $(n-1,n)$. There is a particular reason why we place the row–cyclic pivoting as the basic strategy for future development – the transformation pattern mimics the one of the modified Gram–Schmidt (MGS) orthogonalization, which is a finite (one sweep) algorithm. In fact, in the case of extremely graded matrices, the one–sided Jacobi SVD behaves almost as the MGS.[1]

2.1.1. Quadratic convergence. The symmetric Jacobi algorithm with row–cyclic strategy is quadratically convergent. This is a well known fact, and the proof of the general case of multiple eigenvalues is given by Hari [20]. Using the off-norm $\boldsymbol{\Omega}(\cdot) = \|\cdot - \mathrm{diag}(\cdot)\|_F$, the quadratic convergence is stated as follows:

THEOREM 2.1. *Let $\{H^{(k)} \,:\, k = 0, 1, 2, \ldots\}$ be the sequence of matrices generated by the symmetric Jacobi algorithm with row–cyclic strategy and with the initial matrix $H = H^{(0)}$ with diagonal entries ordered from large to small. Let the eigenvalues of H be*[2] *$\lambda_1 \geq \cdots \geq \lambda_n$, and let the absolute gap in the spectrum be $\boldsymbol{\delta} = \min_{\lambda_i \neq \lambda_j} |\lambda_i - \lambda_j|$. Then $\boldsymbol{\Omega}(H^{(0)}) \leq \dfrac{\boldsymbol{\delta}}{3} \Longrightarrow \boldsymbol{\Omega}(H^{(n(n-1)/2)}) \leq \dfrac{9}{5}\dfrac{\boldsymbol{\Omega}(H^{(0)})^2}{\boldsymbol{\delta}}$.*

For the high relative accuracy, we need small off–diagonal part of the scaled matrices $(H_s^{(k)})_{ij} = (H^{(k)})_{ij}/\sqrt{(H^{(k)})_{ii}(H^{(k)})_{jj}}$. The following theorem from [27] describes the asymptotic behavior of scaled matrices in the Jacobi algorithm.

THEOREM 2.2. *Let $\{H^{(k)} \,:\, k = 0, 1, 2, \ldots\}$ be the sequence of matrices generated by the symmetric Jacobi algorithm with row–cyclic strategy and with the initial positive definite matrix $H = H^{(0)}$. Let the corresponding scaled matrices be $H_s^{(k)}$, $k \geq 0$. If*

[1] That is, symmetric Jacobi becomes similar to Gaussian eliminations.

[2] We always assume that H has at least two different eigenvalues.

H has only simple eigenvalues $\lambda_1 > \cdots > \lambda_n > 0$ and if $H_{11} \geq \cdots \geq H_{nn}$, then

$$\mathbf{\Omega}(H_s^{(0)}) \leq \frac{1}{4}\min\left\{\frac{1}{n}, \boldsymbol{\gamma}\right\} \Longrightarrow \mathbf{\Omega}(H_s^{(n(n-1)/2)}) \leq 0.715\frac{\mathbf{\Omega}(H_s^{(0)})^2}{\boldsymbol{\gamma}},$$

where $\boldsymbol{\gamma} = \min_{i\neq j}\dfrac{|\lambda_i - \lambda_j|}{\lambda_i + \lambda_j}$ is the minimal relative gap in the spectrum.

The proofs of the above theorems are tedious, and the quadratic reduction of $\mathbf{\Omega}(H_s^{(k)})$ is given only in the case of simple eigenvalues because the general case is very technical. (Bounding the Jacobi angles when several diagonal entries converge to the same eigenvalue, or to eigenvalues in a cluster, becomes difficult.) To complete the picture of quadratic convergence, we would rather just illustrate the mechanism behind it. The key result is the following proposition from [15].

PROPOSITION 2.3. *Let H be $n \times n$ positive definite matrix with p distinct eigenvalues $\lambda_1 > \cdots > \lambda_p$, and let n_i be the multiplicity of λ_i. Let the gaps between the eigenvalues be defined as $\boldsymbol{\gamma}_i = \min_{\lambda_i \neq \lambda_j}\dfrac{|\lambda_i - \lambda_j|}{\lambda_i + \lambda_j}$, $\boldsymbol{\gamma} = \min_{i=1:p}\boldsymbol{\gamma}_i$. Let the diagonal elements of H be ordered as $H_{11} \geq \cdots \geq H_{nn}$, and let $s_0 = 0$, $s_i = s_{i-1} + n_i$, $i = 1, \ldots, p$. Further, let $H = DH_sD$, where $D = \mathrm{diag}(\sqrt{H_{ii}})_{i=1}^n$.*
If $\|H_s - I\| \leq \boldsymbol{\gamma}/3$ then, for $i = 1, \ldots, p$,

$$\sum_{j=s_{i-1}+1}^{s_i}\left(\frac{H_{jj} - \lambda_i}{H_{jj}}\right)^2 + \sum_{\substack{j,k=s_{i-1}+1\\ k\neq j}}^{s_i}(H_s)_{jk}^2 \leq \frac{4}{\gamma_i^2}\left(\sum_{j=s_{i-1}+1}^{s_i}(\sum_{k=1}^{s_{i-1}}(H_s)_{jk}^2 + \sum_{k=s_i+1}^{n}(H_s)_{jk}^2).\right)^2$$

Briefly, once the scaled off–diagonal norm is smaller than one third of the minimal relative gap between the eigenvalues, diagonal blocks of dimensions $n_i \times n_i$, $i = 1, \ldots, p$ will be quadratically small and clearly visible along the main diagonal. This result is best understood if illustrated by an example.

EXAMPLE 2.1. We take diagonal matrix Σ of order $n = 50$ with the spectrum 20, 15, 10, 7, 5, each singular value with multiplicity 10, and then apply orthogonal transformation $X = U\Sigma V^T$ with random orthogonal matrices U and V. We let the Jacobi SVD algorithm run on X and after each sweep we compute the Gram matrix $H = X^TX$ and display in logarithmic scale the value of $|H_s - I|$. In Figure 2.1 we display the results after the first two sweeps. After the first sweep there is no apparent structure. But after the second sweep the matrix reveals some structure. We used arrows to help the reader in discovering five blocks along the diagonal.[3] It is precisely this special structure of the matrix that works in favor of the Jacobi algorithm and in fact makes the quadratic convergence possible. In fact, this structure also makes the numerical convergence possible – pivot positions that poorly define the rotation are forced toward zero by the linear operator itself, not only by the rotations.

From the preceding description it clearly follows that proper *affiliation* of the diagonal entries of H with the corresponding eigenvalues is an important ingredient of the quadratic convergence. The assumption on the off–diagonal part of H, combined with the Weyl's theorem, implies that the diagonals of H are partitioned by belonging to disjoint intervals around the eigenvalues, where each interval contains the number of

[3]It is instructive to do this experiment in e.g. MATLAB and then to rotate the graph and explore the structure of the matrix after each sweep.

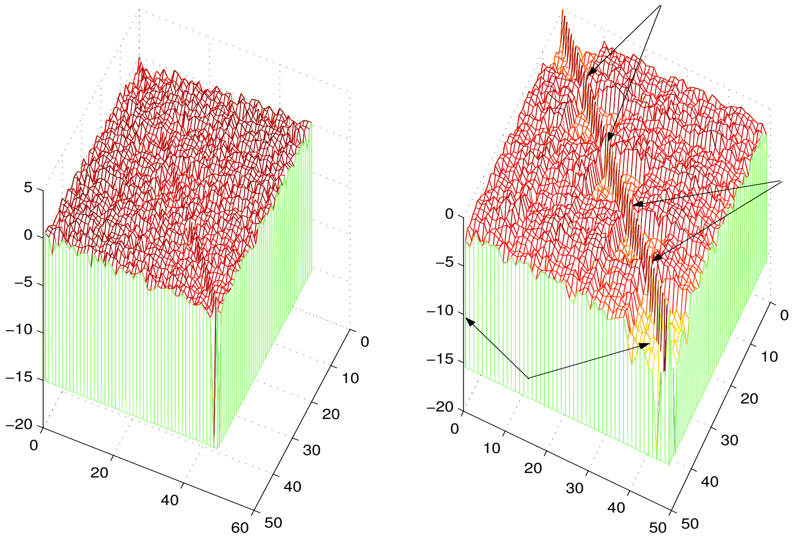

FIG. 2.1. *The structure of the matrix with multiple eigenvalues after the first two sweeps of the Jacobi algorithm.*

affiliated diagonals equal the multiplicity of that eigenvalue. Diagonal entries affiliated with the same eigenvalue should occupy successive positions along the diagonal. *The preconditioner respects this. In the iterative phase, we shall always try to enforce this condition, independent of other modifications we may undertake.*

2.1.2. Cubic convergence. Mascarenhas [26] used the fact that in a quadratically convergent strategy (e.g. row–cyclic) the off–diagonal entries converge to zero at different rates and showed that using special quasi-cyclic strategies Jacobi method can perform cubic asymptotic convergence per quasi-cycle. Here the term quasi–cycle refers to a modified row–cyclic strategy where in a cycle some (slowly convergent) positions are visited more than once. The complexity of the quasi-cycle is about 1.25 times the complexity of classical cycle. Rhee and Hari [29] proved, under certain asymptotic conditions, the global and the cubic (per quasi-cycle) convergence of such modified method.

The cubic convergence is asymptotic, which means that the matrix $X^T X$ needs to be sufficiently close to diagonal form. Since our initial X has been preconditioned twice, we can expect that $X^T X$ is such that the effects of the higher order convergence will be non–negligible. Let the initial matrix be $H = X^T X$ and let us introduce two levels of partitions of H:

$$H = \left(\begin{array}{c||c} H^{[11]} & H^{[12]} \\ \hline H^{[21]} & H^{[22]} \end{array}\right) = \left(\begin{array}{c|c||c|c} H_{[11]} & H_{[12]} & H_{[13]} & H_{[14]} \\ \hline H_{[21]} & H_{[22]} & H_{[23]} & H_{[24]} \\ \hline\hline H_{[31]} & H_{[32]} & H_{[33]} & H_{[34]} \\ \hline H_{[41]} & H_{[42]} & H_{[43]} & H_{[44]} \end{array}\right). \tag{2.1}$$

The quasi–cyclic strategy means, choosing pivot positions in row–cyclic fashion inside the submatrices, respectively, $H_{[33]}$, $H_{[44]}$, $H_{[34]}$, $H_{[33]}$, $H_{[44]}$, $H_{[11]}$, $H_{[22]}$, $H_{[12]}$, $H_{[11]}$, $H_{[22]}$, $H^{[12]}$. Let H' be the matrix computed after such quasi–cycle.

Theorem 2.4. *Let the diagonal entries of H in (2.1) 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*

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

If $\Gamma(H) < 1/4$, then $\Gamma(H') < (49/25)\Gamma(H)^3$. Further, if $\Gamma(H) = \Gamma_1 < 1/4$, then

$$\frac{\boldsymbol{\Omega}(H')}{\boldsymbol{\delta}} \leq 18 \left(\frac{\boldsymbol{\Omega}(H)}{\boldsymbol{\delta}} \right)^3 .$$

Theorem 2.4 states that the convergence is cubic in the $\Gamma(\cdot)$ measure, and in the $\boldsymbol{\Omega}(\cdot)$ measure things look slightly differently. In fact, Rhee and Hari pointed out that the reduction of $\boldsymbol{\Omega}(H)$ 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 [16, §3.]. Intuitively, it is then reasonable strategy to take care of Γ_1 first, thus reducing Γ down to the level which initiates cubic convergence. Briefly, we dare to hope for positive effects of the cubic convergence mechanism very early in the process.*

Here we also note that the presented results on cubic convergence hold for the unscaled off–norm. That the principle can be extended to the scaled off–norm $\boldsymbol{\Omega}(H_s)$ is only a technical matter.

Remark 2.1. It is important to keep in mind that the reality of floating point implementation of the implicit Jacobi SVD looks quite differently from the above description of the convergence. The problem is that the zeros are introduced implicitly, thus the higher order corrections are not added to zeros but to small quantities at the round–off level. (Floating point rotation in general makes the pivot columns at most numerically orthogonal, and not exactly orthogonal.)

2.2. How to exploit triangular form. Strategy of choosing Jacobi rotations is usually not designed to preserve any zero pattern of the input matrix. In fact, the incapability to preserve created zeros is the main reason for the poor performance, as compared with tridiagonalization and bidiagonalization based methods. Strong criticism of this is given in [28, §9.6]. *In our new algorithm, we exploit a possibility to partially use and preserve the zero structure during one–sided transformations, using the fact that the initial matrix is triangular.* We illustrate this in the case of lower triangular matrix.

In the scheme (2.2), X is the array in the memory occupied by the iterates in the Jacobi SVD algorithm. (That is, the input $n \times n$ matrix X and all matrices computed by application of right–handed Jacobi rotations to X are denoted by X.) Let the columns of X be partitioned in four blocks X_1, X_2, X_3, X_4, of dimensions $n \times n_i$, respectively, where each n_i is approximately $n/4$.

$$X = \left(\begin{array}{cc|cc||cc|cc} \blacksquare & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \blacksquare & \blacksquare & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \blacksquare & \blacksquare & \blacksquare & 0 & 0 & 0 & 0 & 0 \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 & 0 & 0 & 0 \\ \hline\hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 & 0 & 0 \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 & 0 \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{array} \right) \equiv (X_1, X_2, X_3, X_4)\,. \tag{2.2}$$

We will say that rotations are applied in the column space of X_i if we implicitly transform $X_i^T X_i$, following some pivot strategy. Further, for two blocks of columns Y_i and Y_j of X, rotating in $Y_i \leftrightarrow Y_j$ means implicit (one sided) transformations inside the off–diagonal blocks in the matrix $\begin{pmatrix} Y_i^T Y_i & \boxed{Y_i^T Y_j} \\ \boxed{Y_j^T Y_i} & Y_j^T Y_j \end{pmatrix}$, following some pivot strategy. In other words, each pivot pair has one column from Y_i and one from Y_j. Here Y_k can be some X_i or e.g. (X_1, X_2).

Consider the most natural greedy approach. Rotating in the column space of 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, and for simplicity pivot pairs can be chosen in the row–cyclic manner. In the same way, transformation of the columns of X_3 by any strategy is computation in a subspace of dimension of approximately $n/2$, as well as the transformation of the columns of the sub–array (X_3, X_4). It is trivial to see that the savings in floating point operations and the effects of better use of memory are nontrivial. Repeated transformations of the columns of X_3 (and independently X_4) are still in the lower dimensional subspace, thus very efficient. Savings in the transformations of the columns of X_2 are modest, but not worthless. (Note that a similar schema can be used for the accumulated product of the Jacobi rotations.)

This greedy approach leads to another nontrivial improvement of the convergence. Indeed, rotating, respectively, in spaces of X_3, X_4, $X_3 \leftrightarrow X_4$, X_3, X_4, X_1, X_2, $X_1 \leftrightarrow X_2$, X_1, X_2, and finally in $(X_1, X_2) \leftrightarrow (X_3, X_4)$ leads, as discussed in §2.1.2, to *cubically* convergent process.

REMARK 2.2. Our goal is to initiate synergy of the positive effects of preconditioning, improved memory traffic, lower flop count (due to triangular form) and higher order convergence. Because of the preconditioning, we expect that with some partition (2.1) $H = X^T X$ is a good candidate for cubic reduction of the off–diagonal entries. However, the block partition (2.2) is motivated by flop count savings and regardless of the almost diagonal structure of $X^T X$ and the distribution of its eigenvalues, which is important for the cubic convergence. A more careful partitioning of X would try to estimate the distribution of its singular values, match the column norms and the corresponding singular values (by column permutations if needed), and then partition so that multiple or clustered singular values are approximated from single column block. This can be done, but we choose not to change the initial partition too much. We combine different devices and therefore reasonable trade–offs are unavoidable.

Obviously, X becomes dense after rotating a full sweep, and this block–quasi–cyclic strategy cannot use any zero structure of X in the next sweep. *However, there is another, more subtle, structure which can be exploited and which becomes clearly visible if we study the Gram matrix $X^T X$.*

2.3. How to adapt to the nearly band structure?. Now, X is dense, but $X_c = X\mathrm{diag}(1/\|X(:,i)\|)_{i=1}^n$ is close to orthogonal and the largest off-diagonal entries of $X_c^T X_c$ are located close to the diagonal. *That is, we expect that $X^T X$ is an s.d.d. matrix in the sense of [3], and that its off–diagonal mass is distributed close to the diagonal. In fact, $X^T X$ is also expected to be shifted quasi–definite matrix, see [16].* This is typical non–pathological situation. Pathological case occurs if the singular values are multiple or poorly separated, that is, the spectrum is composed of many clusters of singular values.

The scheme (2.3) gives the main idea how pivot strategy can dynamically adapt to the structure of X^TX. The basic strategy is row–cyclic, but if in a row i certain number of consecutive rotations is skipped (because the stopping criterion is satisfied), the control of the row–cycling moves to the next row. This means that the remaining pivot positions of the i–th row are not tested against the tolerance. This strategy

$$
X^TX = \left(\begin{array}{cccc||cccc}
\blacksquare & \boxplus & \boxminus & \boxminus & \otimes & \otimes & \otimes & \otimes \\
 & \blacksquare & \boxminus & \boxplus & \boxminus & \boxminus & \otimes & \otimes \\
 & & \blacksquare & \boxminus & \boxminus & \otimes & \otimes & \otimes \\
 & & & \blacksquare & \boxplus & \boxminus & \boxminus & \otimes \\
\hline\hline
 & & & & \blacksquare & \boxplus & \boxminus & \boxplus \\
 & & & & & \blacksquare & \boxplus & \boxminus \\
 & & & & & & \blacksquare & \boxplus \\
 & & & & & & & \blacksquare
\end{array}\right), \quad
\boxed{\begin{array}{ll}
\boxplus & \text{rotated} \\
\boxminus & \text{rotation skipped after test} \\
\otimes & \text{dot product test abandoned}
\end{array}}
\tag{2.3}
$$

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

is motivated by the following reasons. First, it is very likely that the $\otimes$–positions in the scheme (2.3) will pass the tolerance check, so we save unnecessary dot products. (See [16].) Secondly, even if the $\otimes$–positions would 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. Certainly, this modification of the row–cyclic strategy may bring no substantial savings if the initial matrix has close singular values. But, 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 returns the control to a convergent strategy after at most 3 or 4 modified sweeps. In a non–pathological case the expected total number of rotations in 3 or 4 sweeps of this predict–and–skip strategy is $O(n)$.

There are more good reasons to avoid even to probe the pivots at the $\otimes$ positions. 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 strategy is not suitable for the implicit Jacobi SVD algorithm because checking the pivot against the threshold value is extremely expensive – some $O(n)$ flops per pivot. *Our predict–and–skip strategy produces the positive effect of variable threshold, but avoids dot products.*

Our next rationale is related to the convergence of the scaled matrices $H_s^{(k)}$ – it shows why *it does not pay off to annihilate small pivots early in the process.* Note that in general the sequence $\|H_s^{(\ell)} - I\|_F$, $\ell \geq 0$, does not converge monotonically. The transition from $H_s^{(\ell)}$ to $H_s^{(\ell+1)}$ introduces two zeros at the pivot positions, but the transformation is not orthogonal. Let us analyze one step. To simplify notation, consider the transformation $H' = V^THV$, where V is the Jacobi rotation that annihilates the position (p,q). Let the corresponding scalings be $H = DH_sD$, $H' = D'H_s'D'$. Then $H_s' = (D')^{-1}V^TDH_sDV(D')^{-1}$. The matrix $S = (D')^{-1}V^TD$ differs from the identity only at the pivot positions where it takes the form

$$
\hat{S} = \begin{pmatrix} \sqrt{\frac{H_{pp}}{H'_{pp}}}\cos\phi & -\sqrt{\frac{H_{qq}}{H'_{pp}}}\sin\phi \\ \sqrt{\frac{H_{pp}}{H'_{qq}}}\sin\phi & \sqrt{\frac{H_{qq}}{H'_{qq}}}\cos\phi \end{pmatrix} = \begin{pmatrix} s_{pp} & s_{pq} \\ s_{qp} & s_{qq} \end{pmatrix}. \tag{2.4}
$$

Let $\Delta = \|H_s' - I\|_F^2 - \|H_s - I\|_F^2$, and let y_p and y_q be the p–th and the q–th row of the $(n-2)\times(n-2)$ matrix obtained from H_s by deleting its pivot rows and columns. Direct calculation, similar as in [14], [27] gives

$$\frac{\Delta}{2} = \|y_p\|^2(s_{pp}^2 + s_{qp}^2 - 1) + \|y_q\|^2(s_{pq}^2 + s_{qq}^2 - 1) + 2y_p^T y_q(s_{pp}s_{pq} + s_{qp}s_{qq}) - \frac{H_{pq}^2}{H_{pp}H_{qq}}.$$

Using the formulas for the Jacobi rotation and the definition of S, we obtain

$$s_{pp}^2 + s_{qp}^2 - 1 = \frac{\frac{H_{pq}^2}{H_{pp}H_{qq}}}{1 - \frac{H_{pq}^2}{H_{pp}H_{qq}}} = s_{pq}^2 + s_{qq}^2 - 1,$$

$$s_{pp}s_{pq} + s_{qp}s_{qq} = \frac{1}{2}\sqrt{H_{pp}H_{qq}}\left(\frac{1}{H_{qq}'} - \frac{1}{H_{pp}'}\right)\sin 2\phi = -\frac{\frac{H_{pq}}{\sqrt{H_{pp}H_{qq}}}}{1 - \frac{H_{pq}^2}{H_{pp}H_{qq}}}.$$

Proposition 2.5. *Let $H^{(\ell+1)}$ be computed from $H^{(\ell)}$ by rotating at pivot position (p,q). Then the difference $\|H_s^{(\ell+1)} - I\|_F^2 - \|H_s^{(\ell)} - I\|_F^2$ equals*
$2\left(\frac{(H_s^{(\ell)})_{pq}^2}{1-(H_s^{(\ell)})_{pq}^2}\sum{}'((H_s^{(\ell)})_{pk}^2 + (H_s^{(\ell)})_{qk}^2) - \frac{2(H_s^{(\ell)})_{pq}}{1-(H_s^{(\ell)})_{pq}^2}\sum{}'(H_s^{(\ell)})_{pk}(H_s^{(\ell)})_{qk} - (H_s^{(\ell)})_{pq}^2\right)$,
where the sums $\sum'$ are $\sum_{\substack{k=1\\k\neq p,q}}^n$.

The next corollary gives a necessary condition for the growth of $\mathbf{\Omega}(H_s^{(\ell)})$.

Corollary 2.6. *Let in one step of the Jacobi algorithm $\mathbf{\Omega}(H_s^{(\ell+1)}) > \mathbf{\Omega}(H_s^{(\ell)})$, and let $\omega = \sum_{\substack{k=1\\k\neq p,q}}^{n}((H_s^{(\ell)})_{pk}^2 + (H_s^{(\ell)})_{qk}^2) + (H_s^{(\ell)})_{pq}^2 < 1$, where (p,q) is the pivot position. Then $\left|(H_s^{(\ell)})_{pq}\right| \leq \frac{2}{1-\omega}|\sum_{\substack{k=1\\k\neq p,q}}^{n}(H_s^{(\ell)})_{pk}(H_s^{(\ell)})_{qk}|$, which means that the pivot was quadratically small. The increase of the scaled off norm is also at most quadratic.*

Fairly simple proof is omitted. *The corollary suggests that only too small pivot (relative to the remaining entries in pivot rows) can lead to an increase of the scaled off–norm.* But even if the rotation does not increase the scaled off–norm, too small pivot cannot make the reduction substantial. That is exactly what we can expect in the $\otimes$–positions in (2.3).

2.4. Cache–aware pivot strategy. From the early days of scientific computing it was clear that memory references play significant role in the run time. Having to read from (or even first manually replace) the magnetic tape to access matrix elements needed for operations but at the moment not in computer memory certainly kills the performance of any matrix algorithm. Nowadays, we do not use magnetic tapes in matrix computations, but even for matrices that fit into the main memory the penalty of having to perform operations on data not in fast cache memory is still severe.

Since the CPU speed is not matched by the memory speed, the patterns of memory traffic during the run time are of the utmost importance. Well designed algorithm will use the cache memory wisely and avoid various sorts of cache misses. The basic principle to improve temporal and spatial data locality has led to development of a bag of tricks and tips for cache efficient matrix computations. Of course, optimal performance on a particular machine is obtained by an additional profiling with given tools and technical data of the cache system.

Row cyclic pivoting in one–sided Jacobi SVD algorithm is an embarrassing example of violation of main principles of good cache–aware matrix computation. Fairly simple reasoning reveals that the processor is very often stalled due to cache misses. Fortunately, a simple idea of tiling taken from the bag of tricks changes that.

Introduce a parameter b (block size expressed in number of columns) and partition the columns of X. This introduces $\lceil n/b \rceil \times \lceil n/b \rceil$ block partition in $H = X^T X$ and the new strategy is to visit all blocks in the usual row–cyclic fashion, and inside each block all positions are visited row by row. This strategy is equivalent (in the sense of equivalence of pivot strategies in convergence theory of Jacobi processes) to the row–cyclic strategy: both strategies compute the same matrix after $\binom{n}{2}$ rotations.

This block–cyclic strategy can be modified following the lines of §2.2, 2.3 and we omit the technical details which can be found in the software source code. Let us just mention one out of many additional options. At the beginning of the r–th block row, after rotating in the diagonal block (r, r) we allow the possibility to transform next k diagonal blocks before entering the block $(r, r + 1)$. The parameter k is small integer (typically $0, 1, 2$) depending on X, n, b and cache parameters. It influences convergence rate (this is easily seen, take e.g. $k = 1$) and memory access patterns. The proof of global convergence of this strategy should be only a technical matter.

Algorithm 1 Example of tiling for a modified 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
```

2.5. How to stop?. The predict–and–skip idea can be transferred to checking the numerical convergence which occurs after $n(n-1)/2$ consecutive rotations have been skipped. Note that this means computation of $n(n-1)/2$ dot products not followed by rotations, an $O(n^3)$ cost that is not negligible. Of course, all dot products must be computed in order to declare numerical convergence, there is no possibility in saving flops by skipping some of them. However, if the dot products are computed as composite BLAS 3 [11] operation xSYRK, information needed to check the convergence is gathered much faster. For that, we need to predict that such an empty sweep is ahead and then we can interrupt the pivot strategy by the call to xSYRK. If the predictor is right, this means drastic reduction of the cost of the last sweep. In the case of false alarm, more rotations are needed, but we can use the information on the structure of $X^T X$. If extra rotations are non–overlapping, they will lead to numerical convergence. If not, the control is returned to the main pivot strategy until the predictor issues next interrupt. This schema can be constructed in many different ways (with two or more collaborating predictors) and we omit the details which will be given elsewhere, together with the details of the predictor based on convergence theory of Jacobi processes.

We finally note that the tolerance in the pivot strategy can be modified to give better performance in the case that only the singular values are needed, and with the classical absolute error bounds, see [16].

3. Numerical testing. In this section we present mathematical software that implements the new algorithm described in [16, Algorithm 4] and in this report. We give the results of preliminary testing of the algorithm with respect to numerical accuracy and efficiency (run time compared with existing algorithms). Our goal is numerically reliable software implemented to reach reasonable fraction of the efficiency of the less accurate bidiagonalization based methods. In other words, 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.

High performance matrix computation is usually achieved by using machine optimized BLAS library. It is our impression that the developers of these libraries are mainly interested in gaining high flop count per second at any price, and that some numerical properties can be silently lost in the release–after–release–after–release fixes and improvements. It seems that the potentials of the processors with respect to numerical accuracy are sometimes sacrificed in favor of potential for optimization for speed. The criticism of such tendencies in hardware and software developments is best expressed by Kahan [24]. Memory leaks are sometimes considered acceptable risk for gaining higher execution speed. Bugs in such libraries are considered as unlucky accidents to be resolved in the next releases.

Here we assume that we can trust that the compiler, the implemented machine arithmetic and the BLAS library are bug free. One should be aware that this is not always warranted, and that implementing and testing a theoretically completed algorithm into mathematical software is far from being a routine software engineering task, even if we use building blocks from the BLAS and the LAPACK libraries. Sometimes, it requires tedious and frustrating work even to get commercial compiler suite to link properly. The criticism expressed in the 1973. review paper of F. Bauer [4] applies today as well.

3.1. Why and how to test an SVD algorithm. Why do we need to test software implementation of a numerical algorithm if we have numerical analysis proving its accuracy? Once we are sure that there are no pure programming errors in the software and if we assume that the arithmetic is implemented according to the model (standard) used in the analysis, everything should run as predicted by the theory.

The problem is that the assumptions of the developed theory are not always met in the practice. Typical example is expectation that implemented arithmetic behaves following simplified model which ignores (perhaps of really low probability but not impossible) occurrences of underflow (flush to zero or denormalized) in the error analysis. Good starting point in the study of these problems is [8]. Further, there is a dangerous possibility of a gap between numerical properties of standard implementation of some elementary matrix factorization (used in the theoretical analysis) and its highly optimized version used to get optimal performance.

Carefully designed testing of the software is also test of the theory. It shows how sharp are the theoretical bounds and it also gives new insights in the cases of input matrices which are on the boundaries of the theoretical assumptions. Good test cases (accidentally found or cleverly designed) 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, before giving any test result, in §3.2 we describe modification of the algorithm, prompted by numerical tests of an early version of the code.

Finally, testing different algorithms can help in deciding which one is the method of choice in particular situations, with respect to numerical properties (accuracy, stability) and efficiency (speed, memory requirements).

We test single precision (32 bit representation, $\boldsymbol{\varepsilon} \approx 5.3 \cdot 10^{-8}$) implementation because lower precision computation increases the probability of encountering unforeseen numerical difficulties related to finite precision arithmetic. Underflow threshold is denoted by $\boldsymbol{\nu}$ and overflow by $\boldsymbol{\omega}$. It is always assumed that the nonzero entries of input matrix are normalized floating point numbers.

3.1.1. Measuring error – distance to what. 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

$$\begin{aligned}
\mathbf{r} &= \mathit{computed}\Big(\frac{\|A - \tilde{U}\tilde{\Sigma}\tilde{V}^T\|_F}{\|A\|_F}\Big) \ \ \text{(should be at most } f(m,n)\boldsymbol{\varepsilon},\ f \text{ moderate);} \\
\mathbf{o}_U &= \mathit{computed}(\max_{i,j} |(\tilde{U}^T\tilde{U})_{ij} - \delta_{ij}|) \ \ \text{(should be at most } O(m\boldsymbol{\varepsilon})\text{);} \\
\mathbf{o}_V &= \mathit{computed}(\max_{i,j} |(\tilde{V}^T\tilde{V})_{ij} - \delta_{ij}|) \ \ \text{(should be at most } O(n\boldsymbol{\varepsilon})\text{).}
\end{aligned}$$

These measures are useful to test the correctness of the code (bug free in the usual sense) 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. On the other hand, the column–wise backward error (that is, residual bounds for each column of A) cannot be easily checked. In fact, we have

$$\mathit{computed}(\tilde{U}\tilde{\Sigma}\tilde{V}^T) = \tilde{U}\tilde{\Sigma}\tilde{V}^T + E, \quad |E| \leq O(n\boldsymbol{\varepsilon})|\tilde{U}| \cdot \tilde{\Sigma} \cdot |\tilde{V}|^T, \tag{3.1}$$

which means that small columns of $\tilde{U}\tilde{\Sigma}\tilde{V}^T$ are not necessarily computed to sufficient accuracy, so even if the backward error in some column is small, we are not able to detect that fact by using computed residuals. Note that even double precision computation of the residual may not be enough if the condition number of A is large.

One difficulty in testing a new SVD software on a large set of pseudo–random 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 pseudo–random numerically orthogonal $\overline{U}$, diagonal $\overline{\Sigma}$ and numerically orthogonal $\overline{V}$ and then define $A = \mathit{computed}(\overline{U}\ \overline{\Sigma}\ \overline{V}^T)$. But as the relation (3.1) shows, 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. We should also keep in mind that working in higher precision generates different numerical process in finite precision environment and that double precision procedure also needs to be tested. 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) $\kappa(A) < 1/\varepsilon \approx 10^8$. Our choice of the reference procedure is classical one–sided Jacobi SVD with de Rijk's [7] 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]

$$\mathbf{e}_i = \frac{|\tilde{\sigma}_i - \hat{\sigma}_i|}{\hat{\sigma}_i},\quad i = 1,\ldots,n,\quad \mathbf{e} = \max_{i=1:n} \mathbf{e}_i. \tag{3.2}$$

We should not forget that the double precision and the single precision procedure do not have the same matrix on input, as the single precision version A_{single} of a double precision matrix A_{double} is generally from an ε–neighborhood of A_{double}.

3.1.2. Test matrices. Our primary target are 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 that case we will have experimental evidence that both the theory and the numerical testing are done properly.

We use the algorithm of Stewart [30] to generate random orthogonal matrices distributed uniformly according 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], [6]. 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 LAPACK testing library (LAPACK/TESTING/MATGEN/), and with each of them we generate certain number

[4]During the testing we have accidentally found an example of serious failure of 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 is by definition $0/0 = 0$.

of samples (test matrices). In this way, we have an automated generator of pseudo–random matrices with certain relevant parameters varying systematically in given range, for instance $\kappa(S)(=\kappa(B))$ is set to take the values 10, 10^2, $10^3, \ldots, 10^8$.

3.1.3. Run time comparisons. We also show run times of the new algorithm and compare them with the SVD procedures from LAPACK. Timing matrix computations software is also a tricky business because of the variety of computing environments and machine optimized libraries which are improving at an impressive rate. An illustration of this development are the following lines taken from the literature:

White [32] describes the classical symmetric Jacobi with the greedy pivoting: "One code for the IBM 650 for matrices up to order 26 is in use at the General Electric research laboratory in Schenectady. It requires 7 to 8 hours to complete the solution of a 26th order matrix." White continues with the description of the row–cyclic strategy and reports:[6] "The number of iterations is between 6 and 10 and the time varies from about 1 minute for a 8×8 matrix to 25 minutes for a 43×43 matrix."

Kogbetliantz [25] describes the performance of the new Jacobi–like SVD: "The results of numerical computations performed with the aid of IBMs new electronic data processing machine type 701 agree completely with the conclusion of this study of convergence. The type 701 performs with extreme rapidity: a 32×32 matrix is diagonalized in 19 minutes and the numerical results are printed in four minutes, the total time being 23 minutes."

We use the clock provided by the operating system and simply record the elapsed time for each competing procedure. 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). The LAPACK library is compiled from the source code and linked with the BLAS library from the Intel's MKL 6.1.

3.2. Underflow and overflow – problems and solutions by perturbation theory. In error analysis of matrix algorithms we usually assume that the data are so scaled that no overflow and underflow exceptions occur during the computation. So, for instance, LAPACK's driver routine SGESVD computes $\alpha = \max_{i,j}|A_{ij}|$ and scales A with $(1/\alpha)\sqrt{\boldsymbol{\nu}}/\boldsymbol{\varepsilon}$ (if $\alpha < \sqrt{\boldsymbol{\nu}}/\boldsymbol{\varepsilon}$) or $(1/\alpha)\boldsymbol{\varepsilon}\sqrt{\boldsymbol{\omega}}$ (if $\alpha > \boldsymbol{\varepsilon}\sqrt{\boldsymbol{\omega}}$).

Since our implementation of Jacobi rotations uses the column norms and the cosines of the angles between the columns, we can compute the singular values almost in the entire range $(\boldsymbol{\nu},\boldsymbol{\omega})$ of floating point numbers. This means that we need to insure that A is scaled so that its maximal column is not larger than $\boldsymbol{\omega}/\sqrt{n}$. (Note that largest value of any column norm of an $m\times n$ A is $\sqrt{m}\boldsymbol{\omega}$.) However, since we use computational routines from other libraries (such as BLAS and LAPACK) which may implicitly use elements of $A^T A$, we choose to be on the safe side and scale A to make sure that its maximal column is not larger than $\sqrt{\boldsymbol{\omega}}/\sqrt{n}$ in Euclidean norm. If the spectrum of singular values spreads over the full range of normalized numbers (which can be detected for instance by discovering columns with very small (near $\boldsymbol{\nu}$) as well as very large (near $\boldsymbol{\omega}$) Euclidean length) and if all of them are wanted to high relative accuracy then enforcing $\sqrt{\boldsymbol{\omega}}/\sqrt{n}$ as the maximum column norm wipes out smallest singular values. In that case $\boldsymbol{\omega}/\sqrt{n}$ is better choice, provided that no computational routine in linked libraries squares the column norms. In any case, the scaling factor c can also be modified to represent the closest exact power of the base of the arithmetic.

[6]The implementation was fixed point code NY CRV3 for matrices of size at most 68 on an IBM 704. In our terminology one iteration means one full sweep of the row–cyclic pivoting.

An extension of this is to use diagonal scalings – we can think of $A = A_0 D_0$ with diagonal D_0 and such that A cannot be stored because of underflow/overflow, but both A_0 and D_0 can. Fast scaled rotations work on the pair A_0, D_0 and deliver the result in factored form.

Underflow problems can be dangerous, tricky and not removable by scaling. In the case of Jacobi algorithm, underflow can cause non–convergence and loss of accuracy. A catastrophic instance of this is flushing Jacobi rotation to identity, even if its action is needed and substantial. Solution of this problem, independent of the mode the denormals are treated (gentle or flush to zero underflow) is given in [13], and we present it here for the readers convenience.

3.2.1. What if rotation underflows?. Recall that in each step the Jacobi rotation implicitly diagonalizes the Gram matrix

$$G = \begin{pmatrix} a & c \\ c & b \end{pmatrix} = \begin{pmatrix} x^T \\ y^T \end{pmatrix} \begin{pmatrix} x & y \end{pmatrix}, \quad x, y \in \mathbf{R}^m \setminus \{0\}. \tag{3.3}$$

The rotation is defined by

$$J = \frac{1}{\sqrt{1+t^2}} \begin{pmatrix} 1 & t \\ -t & 1 \end{pmatrix}, \text{ where } t = \frac{\operatorname{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}, \text{ and } \zeta = \frac{b-a}{2c}. \tag{3.4}$$

In practice, one first computes the parameters a, b, c, and then carefully implements the formulas (3.4). We use slightly modified strategy for the following reasons:

• The range of the computational routine should be the largest possible. This means that singular values should be computed with theoretical accuracy if they are between the underflow and the overflow thresholds. Since the computation of the Gram matrix (3.3) squares the column norms, using the formulae (3.3), (3.4) in general reduces the computation range to the interval $(\sqrt{underflow}, \sqrt{overflow})$. We find this an unnecessary restriction that should be removed. One obvious way to do that is to use higher precision (single to double, double to extended) to compute G. We prefer more intrinsic modification of the formula for ζ, namely

$$\zeta = \frac{\sqrt{\dfrac{b}{a}} - \sqrt{\dfrac{a}{b}}}{2\dfrac{c}{\sqrt{ab}}} = \frac{\dfrac{\|y\|_2}{\|x\|_2} - \dfrac{\|x\|_2}{\|y\|_2}}{2\cos\angle(x, y)}. \tag{3.5}$$

Since $\sigma_{\min} \leq \|x\|, \|y\| \leq \sigma_{\max}$, the column norms will not overflow (underflow) unless $\sigma_{\max}$ ($\sigma_{\min}$) overflows (underflows).

• One sided algorithm introduces the zeros implicitly by orthogonalizing the pivot columns. This means that inaccuracies in the rotation parameters (due to roundoff) do not affect the accuracy of a single floating point rotation – the constructed rotation is numerically orthogonal and its application is stable. But, to reach the eventual convergence, each rotation must increase toward $\pi/2$ the acute angle between pivot columns. The formula (3.5) clearly reveals potential source of problem. If e.g. $\|x\| \gg \|y\|$ so that $\|x\|/\|y\|$ overflows, then $\tilde{\zeta} = computed(\zeta)$ overflows and the computed t is flushed to zero (thus, the computed rotation is identity) or denormalized (thus, with loss of relative accuracy). Note that denormalized nonzero $\tilde{t} = computed(t)$ is possible if it is first computed in higher precision arithmetic and then rounded back to working precision. From the formulas

$$\zeta = \frac{\sqrt{\frac{b}{a}} - \sqrt{\frac{a}{b}}}{2\frac{c}{\sqrt{ab}}} = \frac{\frac{\|y\|_2}{\|x\|_2} - \frac{\|x\|_2}{\|y\|_2}}{2\cos\angle(x,y)}, \quad t = \frac{\operatorname{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$$

we concluded that, for $\|x\| \gg \|y\|$, $\tilde{\zeta}$ or $\tilde{\zeta}^2$ may overflow, thus causing $\tilde{t}$ to underflow, independent of the value of $\cos\angle(x,y)$. Here we do not mean only the trivial mistake to use the straightforward formula for t as function of ζ in cases of big ζ (e.g. if $\tilde{\zeta}^2$ overflows, but $\tilde{\zeta}$ does not) – the correct formula is then $\tilde{t} = \mathit{computed}(0.5/\tilde{\zeta})$.

If $\tilde{\zeta}$ overflows, the problem is subtle. For, even if we go over to the higher precision to compute the rotation, its satisfactory working precision representation may not exist. If we keep it in higher precision, then its application to the working precision data would be inefficient because of the type conversion.

Important thing to realize here is that floating point matrix representation of the rotation can be identity even if the rotation itself performs nontrivial action. To solve this problem, we first analyze the geometry behind the rotation as $\|x\|/\|y\|$ grows toward infinity. In fact, useful insight into the geometric structure of the transformation in such situation is already possible for $\|x\| > \|y\|/\sqrt{\varepsilon}$. In that case

$$\cot 2\phi = \frac{\frac{\|y\|}{\|x\|} - \frac{\|x\|}{\|y\|}}{2\frac{y^T x}{\|x\|\|y\|}} \approx \frac{-\frac{\|x\|}{\|y\|}}{2\frac{y^T x}{\|x\|\|y\|}}, \quad |\cot 2\phi| \geq \frac{1}{2\sqrt{\varepsilon}}$$

and, with relative error of order ε, $\tan\phi \approx \dfrac{1}{2}\dfrac{1}{\cot 2\phi} \approx -\dfrac{y^T x}{\|x\|^2} = -\dfrac{y^T x}{\|x\|\|y\|}\dfrac{\|y\|}{\|x\|}$. Thus, $|\tan\phi| \leq \sqrt{\varepsilon}$ and $\cos\phi$ will be computed as one. Note that in fact the transformation $y' = y + x\tan\phi \approx y - x\dfrac{y^T x}{\|x\|^2}$ performs the Gram–Schmidt orthogonalization of y against x and that the transformation $x' = x - y\tan\phi$ changes x only by

$$|\tan\phi|\|y\| \approx \frac{|y^T x|}{\|x\|\|y\|}\frac{\|y\|^2}{\|x\|} \leq \frac{\|y\|^2}{\|x\|^2}\|x\| \leq \varepsilon\|x\|.$$

This modification can be rigorously analyzed and it works very well in practice. See [13] for more details on this and other issues related to floating point Jacobi rotation.

Now we discuss a different sort of problems – denormalized values are just annoying small numbers which have no influence on the result (they are added to larger values, or can be treated as tiny perturbation) but do have influence on the execution time if not properly handled by the computer.

3.2.2. Just annoying underflows. Denormalized numbers are a clever way to fill the interval around zero free of normalized numbers. From the mathematical point of view this is much more than a neat idea, see [8]. Unfortunately, the implementation of this extension (and some other exceptions) on modern processors and operating systems is not very efficient and it is possible that many denormalized numbers, which in the algorithm may be as good as zeros, drastically slowdown the computation. Depending on the computational task, the *set to zero* option may solve the problem, provided it exists and works properly. It also helps to scale the matrix with maximal factor which will not cause overflow. (For a discussion on how to deal with floating point exceptions in numerical algebra see [8], [9].) However, if the mechanism that creates denormalized numbers is invariant under scaling, the problem persists.

Let us first describe the genesis of the unwanted denormalized numbers in our algorithm. Their first occurrence is possible in the QR factorization of A, even if all non–zero entries of A are deep in the range of the normalized numbers. The easiest illustrative example is sparse A where many Householder reflections hit a very sparse vector. (Think of a 1000×1000 lower triangular matrix (plus some random very sparse upper triangular) and consider application of a series of Householder reflections to its last column. In the same way, the reconstruction of the Q factor can have the same problem.) Once we have computed $\tilde{R}$ with some denormalized entries we transpose it and compute the QR factorization again. The same mechanism generates new denormals and there it is! The second QR factorization can be extremely slow if the arithmetic with denormalized numbers is not well implemented. (The first QR factorization too, but this kind of problem with general dense A occurs with lower probability.) From the point of view of numerical accuracy, these denormalized numbers 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 of the initial A are normalized numbers.

To make things worse, there is another mechanism that produces denormals – the QR iteration which we used as preconditioner. If the matrix A is such that at some index k it holds $\sigma_k \gg \sigma_{k+1}$ and if the RRQR factorization $AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ is good, then we expect separation to be seen in R. Thus, the second QR factorization on R^T may produce even more denormalized numbers.

An finally, the Jacobi rotations of the columns of lower triangular factor are the last victims of the small villains. The dot product needed to compute the rotation angle 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 may generate additional denormalized matrix entries, especially if sparse vectors are involved.

The problems described above are very likely to appear if the columns of A are heavily graded or if A is large and sparse with most nonzero entries in the lower triangle. In that case the convergence of the Jacobi iterations is swift, the total number of rotation is very small and the run time is unacceptably long. We have found this situation rather frustrating – the worst performance in cases of swift convergence!

How to deal with this problem? The first natural idea would be to set small off–diagonal matrix entries to zero, but it is immediately clear that this is not the solution of the problem. First, inspecting individual entries after each transformation in QR factorization in order to set denormalized numbers to zero kills the performance of highly optimized blocked code. Secondly, the denormals would keep reappearing because they actually grow in places of zero entries. We choose quite the opposite approach – using artificial perturbation we destroy all zeros and increase small entries.

Let X be $n \times n$ lower triangular and nonsingular transposed upper triangular factor from the QR factorization. Our 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 Jacobi iterations; *(iii)* small enough so that it does not interfere with the preconditioner and that it does not prevent the use of the lower triangular structure; *(iv)* small enough so that it does not preclude stable a posteriori computation of the right singular vectors. Clearly, the set of active constrains depends on the moment in the algorithm and the task to be completed with given X. We list the relevant cases and discuss the structure of perturbation that fits our needs. We use ζ to denote appropriate threshold value used in construction of $\delta X \equiv \delta X_\zeta$, for instance $\zeta = \sqrt{\boldsymbol{\nu}/\boldsymbol{\varepsilon}}$, or $\zeta = \sqrt{\boldsymbol{\nu}}$, or $\zeta = \boldsymbol{\varepsilon}/n$.

Case 1. *In the next step we need QR factorization of X (with or without column pivoting) or we apply the right–handed Jacobi SVD algorithm but without computation of the right singular vectors.* Here X is the result of the first or the second QR factorization with pivoting, thus with known nontrivial structure. (Some of the ideas apply to general lower triangular matrix with no other known structure.) We first describe the lower triangle of δX_ζ. Each X_{ij} with the property $|X_{ij}| < \zeta|X_{jj}|$ is replaced with $\mathrm{sign}(X_{ij})\zeta|X_{jj}|$, 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 $(\delta X_\zeta)_{ji} = -\mathrm{sign}(X_{ij})\zeta|X_{ii}|$. (Here ζ can be replaced with a random number of the order of magnitude of ζ.) Note that $\|\delta X_\zeta(:,j)\| \leq \sqrt{n}\zeta|X_{jj}|$ which means that computations with X and $X + \delta X_\zeta$ are almost indistinguishable from the backward and forward error points of view.

Case 2. *In the next step we compute the SVD of X using right–handed Jacobi rotations and the right singular vectors are computed a posteriori from matrix equation.* In this case we also have to preserve the stability of the matrix equation solved for the right singular vectors. This means that the perturbation δX_ζ has to be also row–wise small. This is achieved by taking $(\delta X_\zeta)_{ij} = -X_{ij} + \mathrm{sign}(X_{ij})\zeta\min\{|X_{ii}|, |X_{jj}|\}$, $(\delta X_\zeta)_{ji} = -\mathrm{sign}(X_{ij})\zeta\min\{|X_{ii}|, |X_{jj}|\}$ for all $i > j$. It is interesting that after introducing δX_ζ we simply ignore it and treat $\tilde{X} = X + \delta X_\zeta$ as lower triangular. This is because the pivot strategy tailored for lower triangular matrix is important for the overall performance and losing its effects is not an option. To justify this manipulation with δX_ζ, we first note that the upper triangle of δX_ζ contains tiny row–wise relative error in X. Indeed, for $i < j$ we have

$$\frac{|(\delta X_\zeta)_{ij}|}{\|X(i,:)\|} \leq \zeta\frac{|X_{jj}|}{\|X(i,:)\|} \leq \zeta \ \ (\text{since } |X_{jj}| \leq |X_{ii}|)$$

where in case of graded matrix $|X_{jj}|/\|X(i,:)\| \ll 1$. Consider now any $(k, n-k)$ 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, $\|\tilde{X}_{12}(i,:)\| \leq \sqrt{n-k}\zeta\|X(i,:)\|$ for all $i = 1,\ldots,k$. Suppose we are to rotate last $n-k$ columns of $\tilde{X}$, following some pivot strategy, but decide to think of $\tilde{X}$ as if it was $\tilde{X}_{\llcorner}$ (that is, we do not transform $\tilde{X}_{12}$). The computed matrix is represented by the backward perturbation analysis as

$$\begin{aligned} \tilde{X}' &= \begin{pmatrix} \tilde{X}_{11} & 0 \\ \tilde{X}_{21} & \tilde{X}_{22} + \delta\tilde{X}_{22} \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & \hat{W} \end{pmatrix} + \begin{pmatrix} 0 & \tilde{X}_{12} \\ 0 & 0 \end{pmatrix} = \\ &= \begin{pmatrix} \tilde{X}_{11} & \tilde{X}_{12}\hat{W}^T \\ \tilde{X}_{21} & \tilde{X}_{22} + \delta\tilde{X}_{22} \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & \hat{W} \end{pmatrix}, \ \hat{W}^T\hat{W} = I, \\ \text{where} \quad & \|\delta\tilde{X}_{22}(i,:)\| \leq \varepsilon_J\|\tilde{X}_{22}(i,:)\|, \ \ i = 1,\ldots,n-k, \end{aligned}$$

which means that ignoring $\tilde{X}_{12}$ is equivalent to replacing it with $\tilde{X}_{12}\hat{W}^T$. Since right–handed orthogonal transformation does not change the Euclidean lengths in the row space of the involved matrices, the row–wise backward stability is preserved.

REMARK 3.1. Adding artificial perturbation to avoid underflows can be applied for instance in computing the orthogonal Q from stored reflections by multiplying perturbation of the identity by the reflections constructed during the QR factorization or bidiagonalization. Note that, also in this case, the perturbation is constructed, applied and ignored in exploiting the special structure of the unperturbed matrix.

3.3. Test results. Our new algorithm is implemented in a LAPACK–style routine SGEPVD, which is in an early stage of the development. We have done no serious profiling in order to optimize it for a particular architecture. The most consuming part 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.

3.3.1. 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 §3.1.2, 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 the 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 in eight groups with fixed $\kappa(B)$.[7] Finally, we choose the row and the column dimensions, m and n, and the test procedure is ready.

Before we go over to the comparison of the new procedure SGEPVD[8] with SGESVD and SGESDD from LAPACK, we should point out that SGEPVD computes the SVD to higher accuracy and it also provides an estimate of the maximal relative error by computing an approximation of $\|B^\dagger\|$. Also, after computing the singular vectors, SGEPVD computes their norms using doubly accumulated dot products and explicitly normalizes those with Euclidean length deviating from unity by more than $3\boldsymbol{\varepsilon}$. Note that SGEPVD returns the singular vectors numerically orthogonal up to $m\boldsymbol{\varepsilon}$ which, as in other algorithms dealing with numerical orthogonality, for large m and n may not look satisfactory in single precision ($\boldsymbol{\varepsilon} \approx 10^{-7}$ and e.g. $m = n = 4000$).

We show only two out of many tests performed during code development. Or first test ran with $m = 1500$, $n = 1300$, and with one pseudo–random matrix in each class. Compare the maximal relative errors in computed singular values for all 1024 test cases, shown on Figure 3.1. 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 3.1. The best caption for this figure is the title of [10]. Note that SGESVD returns much better results than SGESDD. To our best 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. [31]).

The timings for this example are shown on Figure 3.2. We immediately note that the new Jacobi SVD algorithm is not at all much slower than the bidiagonalization based fast methods. In fact, it outperforms the QR algorithm and it is on average less than twice slower than the divide and conquer algorithm. The worst case performance for SGEPVD is on matrices with weak column scaling and singular spectrum composed of many tight clusters (examples above the 1.5 mark on Figure 3.2). In all other cases the time of SGEPVD is on average 1.5 times the time of SGESDD. Here,

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

[8]PVD is the acronym for Principal Value Decomposition, an old name of the SVD.

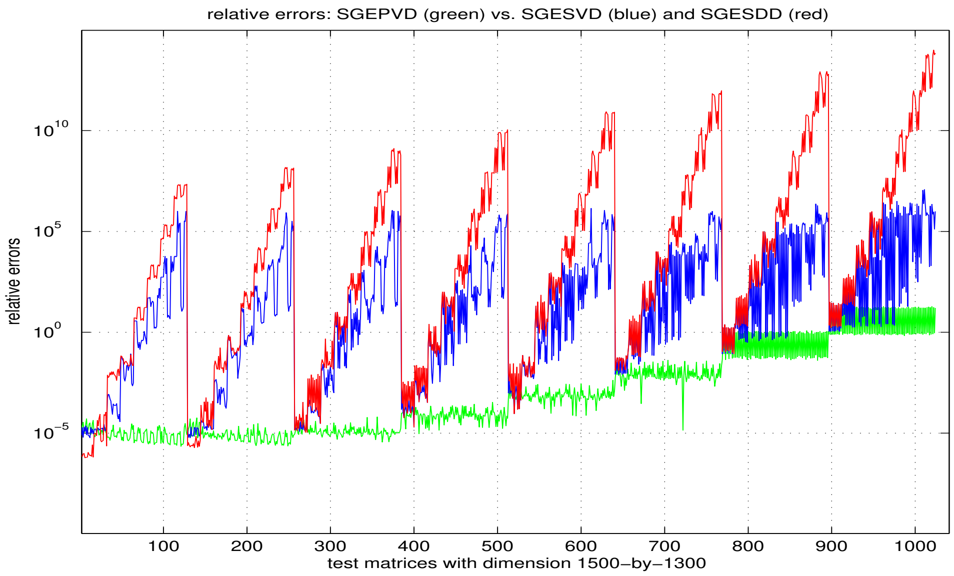

FIG. 3.1. Maximal relative errors for 1500×1300 matrices. The top curve (worst case) describes the accuracy of SGESDD. The middle curve represents the errors of SGESVD, and the lowest curve (smallest relative errors) belongs to SGEPVD.

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 result and additional information are computed in reasonable time. Thus, for a fair comparison one should consider both Figure 3.1 and Figure 3.2 before deciding which algorithm is better choice for a particular application.

In the second test we have 500×350 matrices, with two examples in each of 1024 classes. The results are on Figure 3.3 and Figure 3.4.

REMARK 3.2. Note few outliers above mark 2 on Figures 3.2, 3.4. They correspond to matrices on which SGEPVD actually performed very well, with low number of rotations and swift convergence. (We have checked that 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 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 $\zeta = \boldsymbol{\varepsilon}/n$ the run time reflects the actual flop count. If *set to zero* underflow is in function, this kind of problems disappears.

REMARK 3.3. We have noted that using double accumulated dot product in preparation of Jacobi rotation reduces the total number of rotations, especially in case of multiple or tightly clustered singular values. Unfortunately, in the MKL BLAS library DSDOT performs poorly in comparison to SDOT and the saving in number of rotations does not reduce the total run time.

REMARK 3.4. 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 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

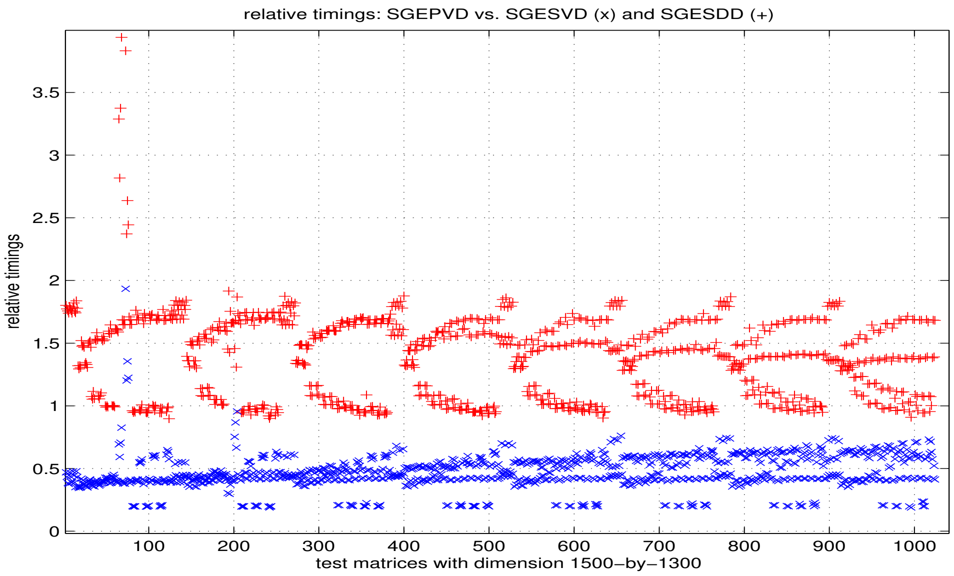

FIG. 3.2. Computing full SVD: Relative timings for 1500×1300 matrices on a Pentium 4 machine with Intel MKL 6.1 library. The crosses denote $time_{SGEPVD}/time_{SGESVD}$ and the pluses are $time_{SGEPVD}/time_{SGESDD}$.

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.

3.3.2. 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 [10], [12]. In the previous section we showed that it computes the full SVD with efficiency comparable with fast bidiagonalization–based approaches.

If only the singular values are needed, reaching similar level of relative efficiency seems to be mission impossible. For, we need Jacobi–based algorithm that computes Σ in time comparable to the time needed to bidiagonalize the matrix!

We immediately note that in the case $m \gg n$ both methods start with the QR factorization which is the most expensive part of the computation. So, this is one example where Jacobi–based approach can be competitive. Further, our approach should have some advantage in case of matrices of low numerical rank, because we start with a rank revealing factorization. (Bidiagonalization is not rank revealing unless enhanced with pivoting which would make it more expensive.) Also, if only the standard absolute error bound is required, then Jacobi iterations can be controlled by a loosened stopping criterion, thus allowing satisfactory approximation with less computational effort.

The test matrices of the form $A = BD$ are generated as in §3.3.1, 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 §3.3.1. 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). See Figures 3.5, 3.6. (We note that SGEPVD computes Σ and also an estimate of the scaled condition number.)

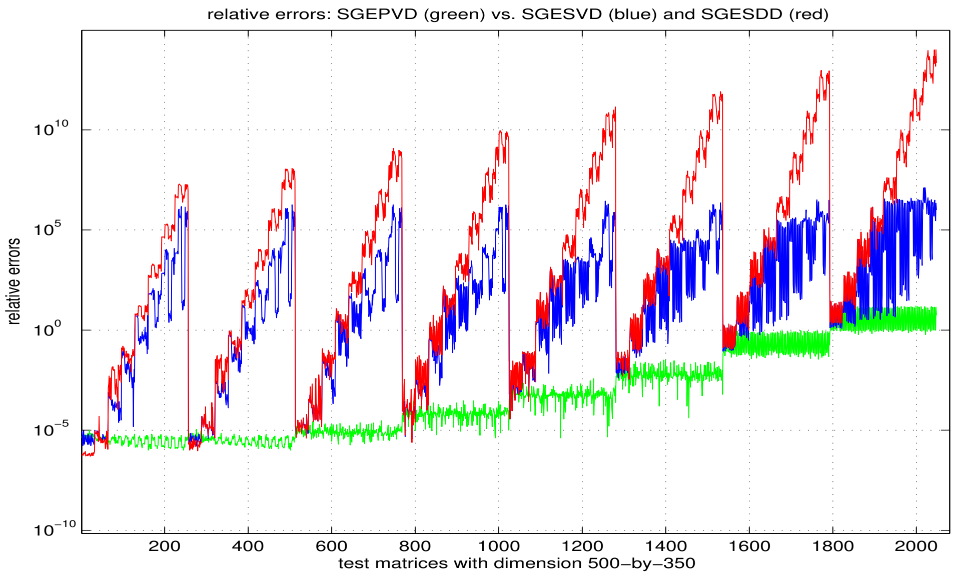

FIG. 3.3. Maximal relative errors for 500×350 matrices. The top curve (worst case) describes the accuracy of SGESDD. The middle curve represents the errors of SGESVD, and the lowest curve (smallest relative errors) belongs to SGEPVD.

4. Conclusion and future work. The most important message of the results in this report is that the question of the ultimate dense non–structured full SVD method is still open. Wanted is the most efficient algorithm capable of computing the SVD to optimal (numerically feasible) accuracy warranted by the data. In the first stage of our program we have shown that a new preconditioned Jacobi–type SVD algorithm can be competitive with fast bidiagonalization based methods without trading accuracy for speed. Moreover, if only the absolute accuracy suffices, our algorithm can take advantage of the rank revealing property of the preconditioner and thus run very efficiently. Because of that, our quest for ultimate dense non–structured SVD algorithm will follow the Jacobi idea.

Another approach could be to try to improve the accuracy of the bidiagonalization. An improvement suggested by Barlow preprocesses the bidiagonalization using QR factorization with complete pivoting, the bidiagonalization itself must use pivoting and in some cases Givens rotations instead of reflections. This in fact works very well in practice, and it is only partially understood, see Barlow [2]. Hence, it improves the accuracy of the bidiagonalization and in many cases it enables bidiagonalization based SVD methods to deliver the decomposition as accurately as the Jacobi SVD method. However, these modifications increase the total time of any bidiagonalization based algorithm by (i) the time of the QR factorization with complete pivoting; (ii) the time of multiplication by the orthogonal factor of this initial QR factorization, to get the left singular vectors; (iii) the additional time of pivoting and replacing reflections by rotations in the bidiagonalization. The question is how these improvements of accuracy impact the efficiency.

On the other hand, the most expensive part of our method are BLAS 1 Jacobi rotations which means that there is more potential for optimization. This and other relevant issues will be explored in the second stage of our program. One simple

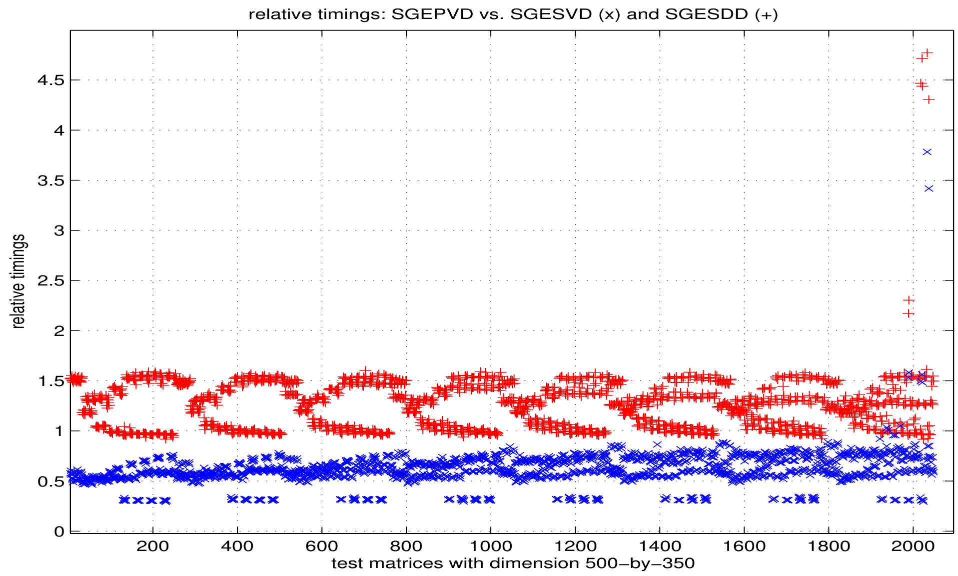

FIG. 3.4. Computing full SVD: Relative timings for 500×350 matrices on a Pentium 4 machine with Intel MKL 6.1 library. The crosses denote $time_{SGEPVD}/time_{SGESVD}$ and the pluses are $time_{SGEPVD}/time_{SGESDD}$.

improvement will be possible once optimized combinations of AXPY and DOT, and combination of two linked AXPY's 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] and currently under further development and implementation at the Department of Mathematics in Zagreb will fully exploit the potential of our approach. 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.

The second stage will complete with the release of a high performance LAPACK–style software. The question is whether or not we will be able to reach the efficiency of SGESDD. Time will tell.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenny, S. Ostrouchov, and D. Sorensen, *LAPACK users' guide, second edition*, SIAM, Philadelphia, PA, 1992.

[2] J. Barlow, *More accurate bidiagonal reduction for computing the singular value decomposition*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 761–798.

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

[4] F. Bauer, *Software and software engineering*, SIAM Review, 15 (1973), pp. 469–480.

[5] N. N. Chan and Kim-Hung Li, *Diagonal elements and eigenvalues of a real symmetric matrix*, SIAM J. Matrix Anal. Appl., 91 (1983), pp. 562–566.

[6] P. I. Davies and N. J. Higham, *Numerically stable generation of correlation matrices and their factors*, BIT, 40 (2000), pp. 640–651.

[7] P. P. M. de Rijk, *A one–sided Jacobi algorithm for computing the singular value decomposition on a vector computer*, SIAM J. Sci. Stat. Comp., 10 (1989), pp. 359–371.

[8] J. Demmel, *Underflow and the reliability of numerical software*, SIAM J. Sci. Stat. Comp.,

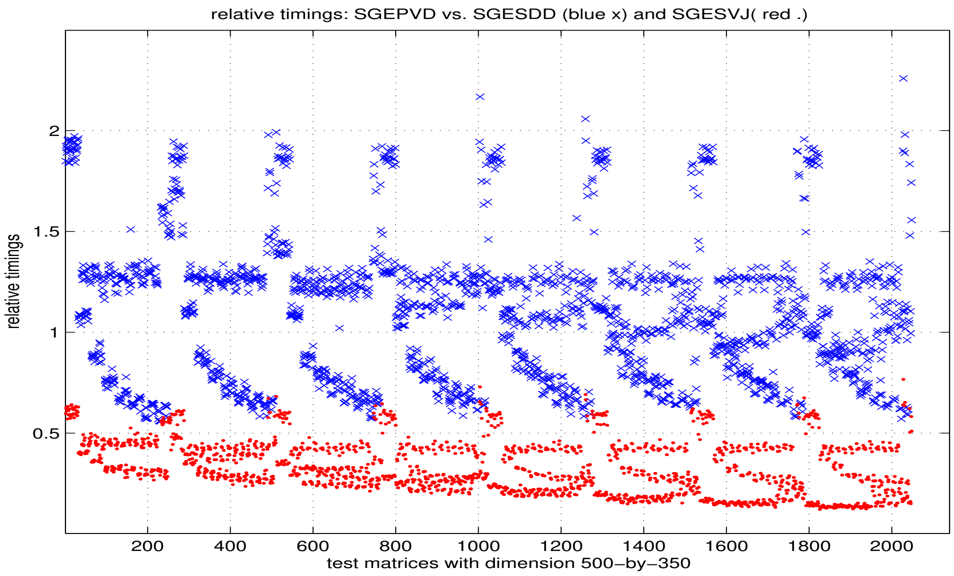

FIG. 3.5. Computing only Σ: Relative timings for 500×350 matrices on a Pentium 4 machine with Intel MKL 6.1 library. The blue crosses denote $time_{SGEPVD}/time_{SGESVD}$ and the red dots are $time_{SGEPVD}/time_{SGESVJ}$.

(1984), pp. 887–919.

[9] J. Demmel and X. Li, *Faster numerical algorithms via exception handling*. Technical report UT–CS–93–192 Department of Computer Science, University of Tennessee, Knoxville (LAPACK Working Note 59), 1993.

[10] J. Demmel and K. Veselić, *Jacobi's method is more accurate than QR*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1204–1245.

[11] J. J. Dongarra, J. J. Du Croz, I. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., (1990), pp. 1–17.

[12] Z. Drmač, *Computing the Singular and the Generalized Singular Values*, PhD thesis, Lehrgebiet Mathematische Physik, Fernuniversität Hagen, 1994.

[13] ———, *Implementation of Jacobi rotations for accurate singular value computation in floating point arithmetic*, SIAM J. Sci. Comp., 18 (1997), pp. 1200–1222.

[14] Z. Drmač and V. Hari, *On the quadratic convergence of the J–symmetric Jacobi method*, Numer. Math., 64 (1993), pp. 147–180.

[15] ———, *Relative residual bounds for the eigenvalues of a Hermitian semidefinite matrix*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 21–29.

[16] Z. Drmač and K. Veselić, *New fast and accurate Jacobi SVD algorithm: I.*, tech. report, Department of Mathematics, University of Zagreb, Croatia, June 2005.

[17] H. H. Goldstine, H. H. Murray, and J. von Neumann, *The Jacobi method for real symmetric matrices*, J. Assoc. Comp. Mach., 6 (1959), pp. 59–96. (Also in J. von Neumann,*Collected Works*, vol. V, pages 573-610, Pergamon Press, New York,1973).

[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, Department of Computer Science, The University of Texas at Austin, Austin, TX 78712, 2002. (FLAME Working Note 9.).

[20] V. Hari, *On sharp quadratic convergence bounds for the serial Jacobi methods*, Numer. Math., 60 (1991), pp. 375–406.

[21] ———, *Fast scaled block Jacobi rotations*, Computing, to appear, (2005).

[22] M. R. Hestenes, *Inversion of matrices by biorthogonalization and related results*, J. SIAM, 6 (1958), pp. 51–90.

[23] 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.

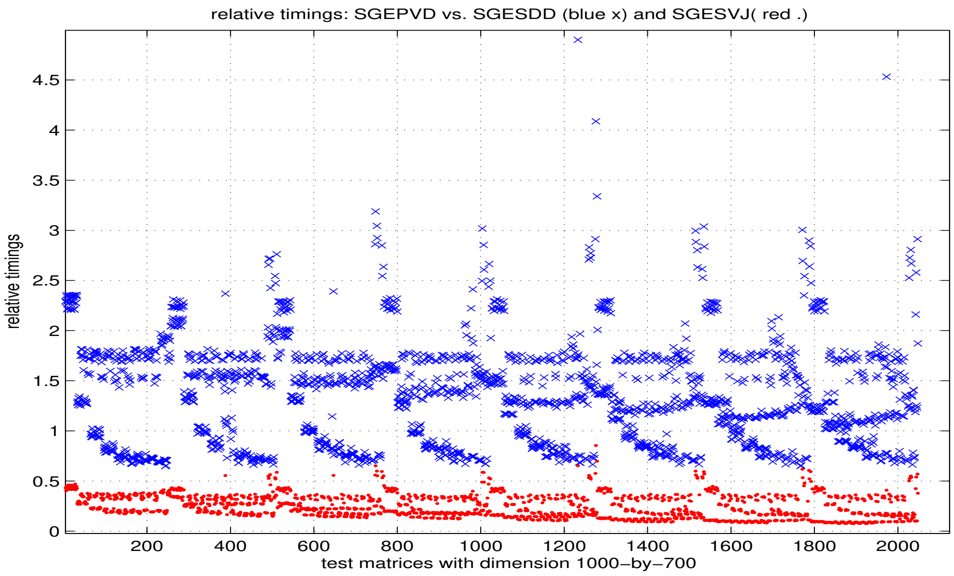

FIG. 3.6. Computing only Σ: Relative timings for 1000×700 matrices on a Pentium 4 machine with Intel MKL 6.1 library. The blue crosses denote $time_{SGEPVD}/time_{SGESVD}$ and the red dots are $time_{SGEPVD}/time_{SGESVJ}$.

[24] W. KAHAN, *The baleful effect of computer benchmarks upon applied mathematics, physics and chemistry*, tech. report, 1995.

[25] E. G. KOGBETLIANTZ, *Solution of linear equations by diagonalization of coefficient matrix*, Quart. Appl. Math., 13 (1955), pp. 123–132.

[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] J. MATEJAŠ, *Quadratic convergence of scaled matrices in Jacobi method*, Numer. Math., 87 (2000), pp. 171–199.

[28] B. N. PARLETT, *The Symmetric Eigenvalue Problem, Classics In Applied Mathematics 20*, SIAM, Philadelphia, PA, 1998.

[29] 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.

[30] 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.

[31] ———, *Perturbation theory for the singular value decomposition*, Technical Report UMIACS–TR–90–124, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, 1990.

[32] P. WHITE, *The computation of eigenvalues and eigenvectors of a matrix*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 393–437.