The singular value decomposition (SVD) of a matrix \( A \in \mathbb{R}^{m \times n} \) has many important applications. In the SVD we seek real orthogonal \( U \) (\( m \times m \)) and real orthogonal \( V \) (\( n \times n \)) such that:

\[
U^T AV = \text{diag}(\sigma_1, \ldots, \sigma_n)
\]

See Golub and Van Loan (1985). In this paper we analyze a family of methods for computing the SVD that are block generalizations of the parallel Jacobi scheme discussed by Pont, Lick, and Van Loan (1985).

Jacobi procedures proceed by making a (increasingly diagonal) by solving a judiciously chosen sequence of 2-by-2 SVD subproblems. Suppose \( A \) is square and let \( \text{off}(A) \) denote the Frobenius norm of \( A \)'s off-diagonal elements, i.e.,

\[
\text{off}(A) = \sqrt{\sum_{i \neq j} |a_{ij}|^2}
\]

For a given dimension let \( J(\theta, 1, 0) \) denote a Jacobi rotation of \( \theta \) degrees in the \((1,0)\) plane, e.g.,

\[
J(2, \cdot, 0) = \\
\begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\theta) & \sin(\theta) \\
0 & -\sin(\theta) & \cos(\theta)
\end{bmatrix}
\]

Parthiva and Maroju (1960) essentially proposed the following Jacobi SVD procedure for square matrices.

**Algorithm 1**

Given \( A \) (\( m \times n \)) and \( \varepsilon > 0 \), the following algorithm computes \( m \times n \) orthogonal \( U \) and \( V \) such that \( \text{off}(UAV) < \varepsilon \|A\|_F \). A is overwritten with \( UAV \).
\[ u := 1 \]
\[ v := 1 \]
\[
\begin{align*}
\text{for } i = 1 \text{ to } n+1 \\
\text{do } (c_i, r_i) := (c_i, \|r_i\|_2) \\
\text{for } j = 1 \text{ to } n \\
\text{compute cosine sine pairs } (c_{ij}, s_{ij}) \text{ and } (c_{ij}, s_{ij}) \\
\end{align*}
\]
\[
\begin{bmatrix}
1 & c_{ij} & \cdots & c_{ij} \\
-1 & -c_{ij} & \cdots & c_{ij} \\
\end{bmatrix}
\begin{bmatrix}
r_1 \\
r_2 \\
0 \\
\end{bmatrix}
= 
\begin{bmatrix}
a_1 & 0 & \cdots & 0 \\
ar_1 & a_2 & \cdots & 0 \\
0 & a_2 & \cdots & a_n \\
\end{bmatrix}
\]
\[
\text{set } a_{ij} = \sqrt{(a_{ij}^2 - r_{ij}^2)}/2 \\
\text{and update } A
\]
\[
A := \frac{1}{\sqrt{a_{ij}}^2} A \\
U := U_1 \\
V := V_1
\]

There are several ways to solve the 2-by-2 subproblems. See Brent, Luk, and Van Loan (1983).

An important feature of any Jacobi procedure is the order in which the off-diagonal entries are zeroed. Algorithm 1.1 incorporates the "cyclic-by-row" ordering so that the off-diagonal entries are zeroed in row-by-row fashion. Note that cycled entries do not stay zero—though generally become nonzero as a result of subsequent rotations. However, it can be shown that

\[ a_{ij}^2 = \sqrt{(a_{ij}^2 - r_{ij}^2)^2} \]

and thus, \( A \) becomes "more diagonal" after each update. A simple pass through the body of the While-Loop above is called a "quicks". The algorithm usually terminates in 6-10 sweeps for typical values of \( n \) and \( \varepsilon \). Specifically, \( n = 100 \), \( \varepsilon = 10^{-11} \).

Jacobi methods, particularly for the symmetric eigenproblem have a very long history. See Jacobi (1829), Hessen (1939), Hansen (1960), and Schonhage (1964). In subsequent sections we develop a block variant of Algorithm 1.1 that is attractive in certain multiprocessor environments. Block Jacobi methods were first implemented in Hansen (1960). The general form of the block algorithm is given in §4 together with a relevant convergence result. In §3 we discuss the parallel ordering. By sorting the off-diagonal elements according to the parallel ordering a significant amount of concurrency can be introduced. In §4 we discuss two ways that the 2-by-2 block subproblems can be solved and other practical issues associated with the block Jacobi approach.

2. A BLOCK JACOBI PROCEDURE

A block version of Algorithm 1.1 is easy to specify with "block notation." Assume that \( n = 2p \) and that we partition \( A \) (a-by-a) as follows:

\[
A = \begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1p} \\
A_{21} & A_{22} & \cdots & A_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
A_{p1} & A_{p2} & \cdots & A_{pp}
\end{bmatrix}
\]

The case of nonsquare blocks will be covered in §4. Denote the \( j \)-th block column of \( A \), \( U \), and \( V \) by \( A_{1j} \), \( U_j \), and \( V_j \). Note that these are a-by-p matrices and
and that \( T = \{ \mathbf{a}_1, \ldots, \mathbf{a}_k \} \), \( U = \{ \mathbf{u}_1, \ldots, \mathbf{u}_p \} \), and \( V = \{ \mathbf{v}_1, \ldots, \mathbf{v}_q \} \). Let

\[
\mathbf{Q} = \begin{bmatrix}
\mathbf{Q}_{11} & \mathbf{Q}_{12} \\
\mathbf{Q}_{21} & \mathbf{Q}_{22}
\end{bmatrix}
\]

\[
\mathbf{P} = \begin{bmatrix}
\mathbf{P}_{11} & \mathbf{P}_{12} \\
\mathbf{P}_{21} & \mathbf{P}_{22}
\end{bmatrix}
\]

be orthogonal, then we let \( \mathbf{Q}_1 \), \( \mathbf{Q}_2 \), and \( \mathbf{P}_1 \), \( \mathbf{P}_2 \) denote the \( p \times k \) block matrix with \( p \times p \) blocks that is the identity everywhere except \( \mathbf{Q}_{11} = \mathbf{Q}_{12} = \mathbf{Q}_{21} = \mathbf{Q}_{22} = \mathbf{1} \).

With this notation we have the following procedure:

**Algorithm 2.2**

Given \( A = (a_{ij}) \), the partitioning (2.1), and \( 0 < \rho \leq 1 \), the following algorithm computes \( n \times n \) orthogonal matrices \( U \) and \( V \) such that

\[
	ext{off}(UAV) \leq \epsilon \| A \|_F
\]

\( A \) is overwritten by \( U^*AV \).

\[
\begin{align*}
\text{if} & \quad 0 < \rho \\
\text{for} & \quad i = 1, \ldots, n \\
\text{do} & \quad \text{off}(A) \leq \| A \|_F \\
\text{for} & \quad j = 1, \ldots, k \\
\quad & \text{compute the SVD } \mathbf{Q}_0 \mathbf{A}_j \mathbf{P}_0 = \mathbf{0} \text{ where } \\
\text{set} & \quad \mathbf{Q}_0 = \mathbf{Q}_{0j} \quad \mathbf{P}_0 = \mathbf{P}_{0j} \\
\text{update } & \quad A = \mathbf{Q}_0^*A_j \mathbf{P}_0 + U_j + V_j, \quad \text{and } \quad A = U_j + V_j
\end{align*}
\]

One pass through the while loop here is referred to as a "block sweep." Corresponding to the real case, \( A \) becomes "more diagonal" after each update. Indeed, it is not hard to show that

\[
\text{off}(A_j)^2 = \text{off}(A)^2 - ∥P_j∥^2_F - ∥Q_j∥^2_F - \text{off}(A_j)_j^2
\]

To set the stage for subsequent analysis, we define Algorithm 2.1 in several ways. First, we take steps to guarantee termination. This can be done by incorporating a threshold. Threshold-based procedures are well known for the regular case for the symmetric eigenvalue problem. In that setting the solving of \( A v = \lambda v \) is stopped if \( |\lambda_v| < \epsilon \), where \( \epsilon \) is the (usually small) threshold parameter. The size of \( \epsilon \) may be fixed or it may vary from sweep to sweep. See Hestenes (1966). In the block situation we pass over the (1,1) subproblem if

\[
|A_{11}| \leq \epsilon \| A_{11} \|_F \leq \epsilon \| A_{1j} \|_F + \epsilon \| A_{j1} \|_F
\]

is small according to the current value of \( \epsilon \).
and that \( A = [A_1, \ldots, A_k] \), \( U = [U_1, \ldots, U_k] \), and \( V = [V_1, \ldots, V_k] \). Let

\[
C = \begin{bmatrix}
Q_{11} & Q_{12} \\
Q_{21} & Q_{22}
\end{bmatrix}
\]

be \( p \times p \) matrices. The SVD of \( C \) is

\[
\text{off}(U^nW^n) \leq \text{eps} \parallel A \parallel_F
\]

where \( A \) is overwritten by \( U^nW^n \).

\[
U := I, \quad V := I
\]

Do while (\text{off}(A) \geq \text{eps} \parallel A \parallel_F)

For \( k = 1 \) to \( m \)

For \( j = 1 \) to \( k \)

Compute the SVD \( U^jA_jV_j = 0 \) where

\[
A_j = \begin{bmatrix}
A_{1j} & A_{2j} \\
A_{3j} & A_{4j}
\end{bmatrix}
\]

Set \( J_j = \text{off}(A_{1j}, A_{2j}) \) and \( J_j = \text{off}(A_{3j}, A_{4j}) \) and perform the updates \( \hat{A} := \hat{A} + J_j U_j \), \( U := U_j \), and \( V := V_j \).

One pass through the While loop here is referred to as a "block sweep." Corresponding to the scalar case, \( A \) becomes "more diagonal" after each update. Indeed, it is not hard to show that

\[
\text{off}(A_{1j}, A_{2j})^2 = \text{off}(A_{3j}, A_{4j})^2 = \parallel A_{1j} \parallel_F^2 - \parallel A_{4j} \parallel_F^2 = \text{off}(A_{3j})^2 = \text{off}(A_{4j})^2
\]

To set the stage for subsequent analysis, we refine Algorithm 2.1 in several ways. First, we take steps to guarantee termination. This can be done by incorporating a threshold. Threshold Jacobi procedures are well-known in the scalar case for the symmetric eigenvalue problem. In that setting the zeroing of \( A_{1j} \) is skipped if \( \parallel A_{1j} \parallel < \epsilon \) where \( \epsilon \) is the usual (small) threshold parameter. The value of \( \epsilon \) may be fixed or it may vary from sweep to sweep. See Rutishauser (1966). In the block situation we pass over the \((1,3)\) subproblem if

\[
\mu(A_{1j}, A_{2j}) = \text{sqrt}(\parallel A_{1j} \parallel_F^2 + \parallel A_{2j} \parallel_F^2)
\]

is small according to the current value of \( \epsilon \).
The threshold parameters must be suitably related to the termination criteria if convergence is to be ensured and here we wish to make another modification of
Algorithm 2.1. Instead of quitting when \( \text{OPF}(A) \) is small enough, we use its block
analogue:

\[
\text{OPF}(A)^2 = \sum_{i,j} \|A_{ij}\|^2
\]

By terminating when \( \text{OPF}(A) \) is small the final matrix \( A \) will be nearly block
diagonal. The diagonalization process is then completed by comparing the SVD's of the
diagonal blocks (in parallel).

In Algorithm 2.1 the \( 2 \times 2 \) block subproblems are exactly diagonalized. As we
are about to point out, complete diagonalization of the subproblems is unnecessary and
so we may test that if we compute

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
= \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\begin{bmatrix}
V_0 \\
V_0
\end{bmatrix}
\]

then

\[
\|B_{11}\|_F^2 + \|B_{21}\|_F^2 \leq \phi^2 \Lambda(1, 1, 1)^2
\]

for some fixed \( \phi < 1 \). Recommended values for \( \phi \) are discussed in \( \S 6.4 \).

The last feature of Algorithm 2.1 that we wish to relax concerns the ordering, instead of just
considering the low-cycling order, we wish to consider the general ordering

\[
\{1, 2\}, \{1, 3\}, \ldots, \{1, n\}
\]

where \( i_m < i_m \) for \( m = 1 \) to \( r \). Overall we obtain

Algorithm 2.2

Given \( A \) (\( n \times n \)): \( \alpha \geq 0 \), \( N \geq 0 \), \( B \geq 1 \), partitioning (2.1), ordering (2.3), and a threshold \( \tau \) satisfying

\[
\tau \leq \max \|A\|_F / N
\]

the following algorithm computes orthogonal \( V \) and \( T \) such that

\[
\text{OPF}(V^TAV) \leq \alpha \max \|A\|_F
\]

\[
U = I \quad \tau = 1
\]

Do While \( (\text{OPF}(A) > \alpha \max \|A\|_F) \)

For \( m = 1 \) to \( \min(k+1)/2 \)

\[
(i, j) = (i_m, i_m)
\]

If \( \gamma(i, j) \leq \tau \)

then

Compute orthogonal \( U_0 \) and \( V_0 \) such that (2.4) holds.

Let \( J_1 = J(1, j, V_0) \) and \( J_2 = J(1, j, V_0) \) and compute the
updates \( A \leftarrow A_{ij} J_1 J_2 \), \( U \leftarrow U_{ij} \), \( \tau := \tau_{ij} \).
There are several details associated with Algorithm 2.1 that we pursue in the next two sections. These include (a) parallel implementation, (b) the precise procedure for solving the 2p-by-2p subproblem, (c) the application of the resulting orthogonal transformations, (d) what to do if \( A \) is singular, and (e) the relevant condition on \( A \). However, before we take up these more technical matters we consider that the preceding algorithm converges.

**Theorem 2.1**

Algorithm 2.1 terminates after a finite number of block sweeps.

**Proof**

If no subproblems are solved during a particular block sweep then we have

\[
\gamma(A, i, j) < \tau
\]

for all \( i \) and \( j \) that satisfy \( 1 \leq i < j \leq k \). Thus,

\[
\| \text{OFF}(A) \|^2 = \sum_{i=1}^{k} \gamma(A, i, i)^2 \leq k(k-1)\tau^2 \leq \epsilon p^2 \| A \|^2
\]

and termination is achieved.

On the other hand, if subproblem \((1, j)\) is solved during a block sweep it is easy to show using the definition (2.8) that the updated matrix satisfies

\[
\text{OFF}[(1, 1), (1, j)]^2 \geq \text{OFF}(A)^2 - \gamma(A, 1, j)^2 \leq \text{OFF}(A)^2 - (1 - \theta)^2 \gamma(A, 1, j)^2.
\]

But if subproblem \((1, j)\) is solved then \( 1 \leq j \leq n(A, 1, j) \) and so from (3.4) we have

\[
\text{OFF}[(1, 1), (1, j)]^2 \leq \text{OFF}(A)^2 - \tau^2 (1 - \theta^2)
\]

Thus, after \( k \) block sweeps \( \text{OFF} \) of the original \( A \) is diminished by

\[
\sqrt{\text{OFF}(A)} \leq \epsilon p \| A \|^2,
\]

it follows that the condition \( \text{OFF}(A) \leq \epsilon \text{OPS} \| A \|^2 \) is eventually satisfied.

Q.E.D.

See Hansen (1969) for further results pertaining to the convergence of block Jacobi procedure.

3. BLOCK JACOBI SYM WITH PARALLEL OMLING

The key to speeding up Algorithm 2.1 is to solve nonconflicting subproblems concurrently. For example, if \( k = 8 \) then the (1,1), (3,4), (5,6), and (7,8) subproblems are nonconflicting in that with a procedure we could solve the 4 subproblems and perform the necessary updates of \( \bar{A}, U \), and \( V \) at the same time. For general \( k \) we may proceed as follows:

**Algorithm 3.1**

Suppose \( A = [\Gamma_1, \ldots, \Gamma_k], U = [G_1, \ldots, G_k] \), and \( V = [P_1, \ldots, P_k] \) where each block column is \( m \times q - p \), assume that \( k \) is even and that we have \( k = k/2 \) processors \( P_1, \ldots, P_{k/2} \) and that \( P_i \) contains block columns \( 2i-1 \) and \( 2i \) of \( A, U \), and \( V \). If \( P_i \) carries out the following algorithm for \( i = 1 \) to \( k/2 \), then
\[ A = \begin{bmatrix} (I(1,2)_{(1)} \cdots \cdots (2(k+1-k),_{(2)}) \end{bmatrix}^T \begin{bmatrix} (V(1,2)_{(1)} \cdots \cdots (2(k+1-k),_{(2)}) \end{bmatrix} \]

\[ U = \begin{bmatrix} \text{diag} (U(1), \ldots, U(k)) \end{bmatrix} \]

\[ V = \begin{bmatrix} \text{diag} (V(1), \ldots, V(k)) \end{bmatrix} \]

where \( U(1) \) and \( V(1) \) are the 2p-by-2p orthogonal matrices that solve subproblem \( (2p-1,2p) \).

Solve subproblem \( (2p-1,2p) \) as in Algorithm 2.1. Let \( U(1) \) and \( V(1) \) be the resulting orthogonal matrices.

\[ \begin{bmatrix} A_{2p-1} & A_{2p} \end{bmatrix} = \begin{bmatrix} A_{2p-1} & A_{2p} \end{bmatrix} \begin{bmatrix} V(1) \end{bmatrix} \]

\[ \begin{bmatrix} U_{2p-1} & U_{2p} \end{bmatrix} = \begin{bmatrix} U_{2p-1} & U_{2p} \end{bmatrix} \begin{bmatrix} U(1) \end{bmatrix} \]

\[ \begin{bmatrix} V_{2p-1} & V_{2p} \end{bmatrix} = \begin{bmatrix} V_{2p-1} & V_{2p} \end{bmatrix} \begin{bmatrix} V(1) \end{bmatrix} \]

Transmit \( U(1) \) to every other processor.

Collect \( U(1), \ldots, U(1), \ldots, V(1), \ldots, V(1) \).

\[ \begin{bmatrix} A_{2p-1} & A_{2p} \end{bmatrix} = \begin{bmatrix} \text{diag} (U(1), \ldots, U(1)) \end{bmatrix} \begin{bmatrix} A_{2p-1} & A_{2p} \end{bmatrix} \]

Note that some coordination is required among the processors.

We now show how the repeated application of Algorithm 3.1 can diagonalize \( A \) if the block columns are redistributed among the processors (as between applications of the procedure). To illustrate, we return to our example where \( A \) is 8-by-8 as a block matrix \( (k = 4) \). Partition the set of 24 off-diagonal index pairs into some "rotation sets" as follows:

I. \((1,2) \ (1,3) \ (1,4) \ (2,3) \ (2,4) \ (3,4) \ (7,8) \)

II. \((1,4) \ (2,4) \ (2,5) \ (3,4) \ (3,5) \ (4,5) \)

IV. \((1,5) \ (1,6) \ (2,5) \ (2,6) \ (3,5) \ (3,6) \)

V. \((1,6) \ (2,6) \ (2,7) \ (3,6) \ (3,7) \ (4,7) \)

VI. \((1,7) \ (2,7) \ (2,8) \ (3,7) \ (3,8) \ (4,8) \)

The four SPD problems specified by each rotation set are nonconflicting. Read left to right, top to bottom, the above is an instance of the "parallel ordering". It is easy to derive by imagining a chess tournament among 8 players in which each player plays every other player exactly once, in between rounds (rotation sets) the players (block columns) move to adjacent tables (processors) in musical chair fashion:

Round 1: 1 3 5 7 2 4 6 8

Round 2: 1 3 5 7 8 6 4 2

Round 3: 1 3 5 7 1 3 5 7 etc.

In the matrix setting we start off with 1 residing in processor \( P_1 \), \ldots, \( P_4 \) as follows:
\[ A_{11} \ A_{12} \ A_{13} \ A_{14} \ A_{15} \ A_{16} \ A_{17} \ A_{18} \\
A_{21} \ A_{22} \ A_{23} \ A_{24} \ A_{25} \ A_{26} \ A_{27} \ A_{28} \\
A_{31} \ A_{32} \ A_{33} \ A_{34} \ A_{35} \ A_{36} \ A_{37} \ A_{38} \\
A_{41} \ A_{42} \ A_{43} \ A_{44} \ A_{45} \ A_{46} \ A_{47} \ A_{48} \\
A_{51} \ A_{52} \ A_{53} \ A_{54} \ A_{55} \ A_{56} \ A_{57} \ A_{58} \\
A_{61} \ A_{62} \ A_{63} \ A_{64} \ A_{65} \ A_{66} \ A_{67} \ A_{68} \\
A_{71} \ A_{72} \ A_{73} \ A_{74} \ A_{75} \ A_{76} \ A_{77} \ A_{78} \\
A_{81} \ A_{82} \ A_{83} \ A_{84} \ A_{85} \ A_{86} \ A_{87} \ A_{88} \\
\]

Subproblems \((1,2), (3,4), (5,6),\) and \((7,8)\) are then solved via Algorithm 3.1.

To get ready for subproblems \((1,4), (2,6), (3,8),\) and \((5,7)\) we recombine \(A\) among the processors as follows:

\[ F_1 \ F_2 \ F_3 \ F_4 \]

\[ A_{11} \ A_{14} \ A_{17} \ A_{16} \ A_{13} \ A_{15} \ A_{12} \ A_{18} \\
A_{21} \ A_{24} \ A_{27} \ A_{26} \ A_{23} \ A_{25} \ A_{22} \ A_{28} \\
A_{31} \ A_{34} \ A_{37} \ A_{36} \ A_{33} \ A_{35} \ A_{32} \ A_{38} \\
A_{41} \ A_{44} \ A_{47} \ A_{46} \ A_{43} \ A_{45} \ A_{42} \ A_{48} \\
A_{51} \ A_{54} \ A_{57} \ A_{56} \ A_{53} \ A_{55} \ A_{52} \ A_{58} \\
A_{61} \ A_{64} \ A_{67} \ A_{66} \ A_{63} \ A_{65} \ A_{62} \ A_{68} \\
A_{71} \ A_{74} \ A_{77} \ A_{76} \ A_{73} \ A_{75} \ A_{72} \ A_{78} \\
A_{81} \ A_{84} \ A_{87} \ A_{86} \ A_{83} \ A_{85} \ A_{82} \ A_{88} \\
\]

Notice that solving the current \((1,2), (3,4), (5,6),\) and \((7,8)\) problems is mathematically equivalent to solving subproblems \((1,4), (2,6), (3,8),\) and \((5,7)\) of the unpermuted \(A\).

Next, we apply Algorithm 3.1 and shuffle the resulting matrix in exactly the same way. We're then set to process the third rotation set (cheese ornamen; round), i.e., subproblems \((1,6), (2,8), (3,7),\) and \((5,3),\) etc. In the \(N\) processor situation the shuffling is most easily described through the \(n\)-by-\(n\) (in \(b\)) permutation matrix

\[ P = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 & \cdots & F_{n-5} & F_{n-4} & F_{n-3} & F_{n-2} \end{bmatrix} \]

where the \(F_i\) are block columns of the \(n\)-by-\(n\) identity:

\[ I_n = \begin{bmatrix} F_1 & F_2 & \cdots & F_n \end{bmatrix} \quad F_i \in \mathbb{R}^{n \times n}, n = 4p \]

In particular, after each application of Algorithm 3.1 block columns \(2i-1\) and \(2i\) of the matrix \(F\) are stored in processor \(F_i\). Overall we have

Algorithm 3.1

Suppose \(A = \{ A_1, \ldots, A_8 \}, U = \{ U_1, \ldots, U_4 \},\) and \(V = \{ V_1, \ldots, V_4 \}\) are given with \(U = V = 1\). Assume that each block column has dimension \(n\)-by-\(p\) and that processor \(F_i\) contains block columns \(2i-1\) and \(2i\) of \(A, U,\) and \(V \) for \(i = 1\) to
N = \sqrt{2} \cdot \text{Given } \epsilon > 0 \text{ the following algorithm computes orthogonal } U \text{ and } V \text{ such that } \|U^T A V - \text{diag}(\epsilon)\|_F \leq \epsilon. \\

\text{Do while } (\|U^T A V - \text{diag}(\epsilon)\|_F > \epsilon) \\
\text{Apply Algorithm J.1} \\
\text{before the update } A := \text{trunc}_p U^T A V, \quad L := U^T L, \quad V := V D \text{ where } p \text{ is given by (J.1). Note that this involves shifting } A, U, \\text{and } V \text{ among the processors.} \\

With this brief development of the parallel block Jacobi SVD algorithm we are ready to look at some important practical details.

1. PRACTICAL DETAILS AND EXPERIENCE

Applying the Orthogonal Transformations

Most of the computational effort in Algorithm J.1 is spent calculating products of the form $Z C$ and $C Z$, where $Z$ is a $2p$-by-$2p$ orthogonal matrix and $C$ is $2p$-by-$4n$. The obvious method for doing this requires $8np^2$ flops. However, there is an interesting alternative that begins by computing the Householder upper triangularization of $L$:

$H_{2p-1} \cdots H_2 H_1 \cdot Y_{2p-1} \cdots Y_2 Y_1 = X$

Since the upper triangular matrix $X$ is also orthogonal, we must have $X = \text{diag}(y)$. To compute $X C$, for example, we sequentially apply the Householder matrices:

$C := X C$ \\
$C := H_1 C$ \\

If we just count flops this "Householder" approach requires $2((4n^2 + 8p^3)/3)$ flops and is thus more economical whenever $p < n^{1/4}$. (This is often the case in multi-processor environments as in implies $k = n^{1/4}$.) Moreover, representing $Z$ as a product of Householder matrices requires half the space of the conventional representation. This allows for a reduction in the communication costs associated with the transposition of an orthogonal matrix from one processor to another. Thus, the Householder alternative has certain advantages as it does in conventional settings. (See Golub and Van Loan (1983), pp. 41-42.)

Solving the Subproblems

In the typical subproblem we are presented with a submatrix

$A_0 = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

and must choose $U_0$ and $V_0$ such that

$p$
\[ \begin{bmatrix} a_1^T & a_2^T \end{bmatrix} u_0 \begin{bmatrix} \eta_1 & \eta_2 \\ \eta_1 & \eta_2 \end{bmatrix} = u_0 = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} \]

satisfies
\[ (4.1) \| v_{12} \|_2^2 + \| v_{21} \|_2^2 \leq \delta^2 \left( \| v_{12} \|_2^2 + \| v_{21} \|_2^2 \right) = \delta^2 \mu(s,1,1)^2 \]

for some \( \delta < 1 \). (See Theorem 2.1.) We study two distinct approaches to this problem.

**Method 1: Partial SVD via Row Cyclic Jacobi**

Use the row-cyclic Jacobi procedure (Algorithm 1.1) to compute \( u_0 \) and \( v_0 \) such that \( (4.1) \) holds. That is, keep swapping until \( a_0 \) is sufficiently close to 2-by-2 block diagonal form.

**Cost = 50n^3 flop/iter w/ swap**

**Method 2: Golub-Kahan SVD with Bidiagonal Pause**

Recall that the Golub-Kahan algorithm begins with a bidiagonalization of the matrix. After \( p \) steps of this initial reduction we have

\[ \begin{bmatrix} a \end{bmatrix} = \begin{bmatrix} x & x & x & x & x & x & x & x & x & x \end{bmatrix} \]

\[ (s = 4) \]

where \( u_a = U_0 \cdots U_p \) and \( v = V_0 \cdots V_p \) are products of Householder matrices. Note that the \((p,p+1)\) entry prevents the reduced matrix from being block diagonal.

This suggests that if

\[ \| b \| \leq \delta s(n,1,1) \]

then \( a_0 \) is sufficiently close to block diagonal form and we set \( u_0 = U_p \) and \( v_0 = V_p \). If \( b \) is too large then we proceed to the iterative portion of the Golub-Kahan algorithms terminating as soon as the current \((p,p+1)\) entry is small enough.

**Cost = 180n^3 if the bidiagonal pause is successful**

**Cost = 80n^3 otherwise**

**Method 1 is appealing because it can exploit the fact that the subproblems are incrementally block diagonal as the iteration progresses. On the other hand, Method 2 is attractive because it is whenever the block diagonal pause is successful.**

Furthermore, Method 2 can handle rectangular problems more gracefully as we now show.
Handling Rectangular Problems

Up to this point we have assumed that \( A \) is a square block matrix with square blocks. It is possible to relax these restrictions. To illustrate, suppose that each block is 4-by-4. The subproblem thus have dimension 8-by-8. If Method 2 is applied and the full SVD computed, we obtain:

\[
\begin{align*}
\sigma_1 & = 0 \\
\sigma_2 & = 0 \\
0 & = \sigma_3 \\
0 & = \sigma_4 \\
0 & = \sigma_5 \\
0 & = \sigma_6 \\
0 & = \sigma_7 \\
0 & = \sigma_8
\end{align*}
\]

Note that the structure of the reduced matrix conflicts with the aim of block elimination—the (1,2) block is square. However, there is a simple solution: interchange rows 4 and 5 with rows 6 and 7. In general, straightforwards row and column interchanges following the SVD computations are sufficient to make Method 2 work on problems having rectangular blocks. (The blocks need not have the same dimension as they did in our example.)

The same technique works for Method 1, except that the rectangular subproblem must be made square by adding zero columns (or rows) before the SVD algorithm is applied. This scheme is discussed in Warm, Lok, and Van Loan (1988) and is somewhat costly. To be precise, let \( L \) and \( S \) be the larger and smaller of the subproblem dimensions. The SVD approach involves \( 4L^3 \) work while the Golub-Reinsch procedure requires \( O(L^2) \) time.

Experimental Observations

To gain perspective on the various options sketched above we ran numerous examples on a VAX 750 in the MATLAB environment. (Machine precision \( 10^{-16} \).) We report on a typical 24-by-24 example which we solved using various values of \( L \), \( K \), and \( M \). In this example \( A \) is 20, 15 and we terminate as soon as \( \text{SV} \text{(2)} \leq \text{SV} \text{(16)} \).

To begin with, the asymptotic rate of convergence is quadratic. Consider the \( k = 6, p = 4 \) situation with Method 2. Tabulating \( \text{SV}(2) \) we find:

<table>
<thead>
<tr>
<th>( \text{Seq} )</th>
<th>( \text{SV} \text{(1)} )</th>
<th>( \text{SV} \text{(2)} )</th>
<th>( \text{SV} \text{(3)} )</th>
<th>( \text{SV} \text{(4)} )</th>
<th>( \text{SV} \text{(5)} )</th>
<th>( \text{SV} \text{(6)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 0.97 \times 10^4 )</td>
<td>( 0.12 \times 10^5 )</td>
<td>( 0.14 \times 10^5 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( 0.71 \times 10^4 )</td>
<td>( 0.43 \times 10^5 )</td>
<td>( 0.61 \times 10^5 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( 0.19 \times 10^4 )</td>
<td>( 0.10 \times 10^6 )</td>
<td>( 0.13 \times 10^6 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( 0.32 \times 10^3 )</td>
<td>( 0.27 \times 10^6 )</td>
<td>( 0.67 \times 10^6 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( 0.36 \times 10^2 )</td>
<td>( 0.15 \times 10^6 )</td>
<td>( 0.49 \times 10^6 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>conv</td>
<td>conv</td>
<td>( 0.72 \times 10^7 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>conv</td>
<td>conv</td>
<td>conv</td>
<td>conv</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In general, we find that the number of block sweeps is a mildly increasing function of \( \theta \). Indeed, we have found that the number of block sweeps is usually minimal as long as \( \theta \leq 0.20 \). Here we report on the number of block sweeps that are necessary for various \( k \), \( p \), and \( M \). (With Method 2 was used.)
Another obvious fact is revealed by the table: the number of sweeps increases with $k$. It is conjectured that the number of block sweeps increases as the logarithm of $k$.

We mention that when Method 2 is used to solve the subproblems then the number of Gauss-Seidel sweeps is successful rapidly decreases as the iteration proceeds. For example, in the $(6,6,5)$ situation, the diagonal sweeps was successful 83% of the time during the first block sweep, 16% of the time during the second block sweep, and never again thereafter.

Although the results that we have that have not been mentioned have all been with Method 2, they essentially apply when the subproblems are solved using Method 1, the Jacobi approach. Thus, the fact that the algorithm doesn’t really need to start from the standpoint of block sweeps. Moreover, for $k = 5$ we find that the two sweeps in Method 1 almost always suffice to solve the subproblems. Based on flip counts we see from above that the two methods are equally efficient. Method 1, being simpler to implement, may be preferable in situations where program memory in the processors is limited.

Acknowledgement

The author wishes to thank Clara Che for discussing the computations presented in this paper.

References


STABILITY CRITERIA FOR INTERVAL MATRICES

Xu Daxi
Mianyang Teachers' College
Mianyang, Sichuan,
China

This paper presents several easy-to-check criteria for the stability and complete unstability of interval matrices, and extends the results obtained by Neimn (1984) and Xu (1985).

INTRODUCTION

In 1966, Moore gave the concept of interval matrices, and described in detail various computations possible with interval matrices. An interval matrix $N(F; Q)$ is a set of real matrices $N(F; Q) = \{X = (x_{ij}) | P_{ij}(t) = a_{ij} + b_{ij}t, \ldots \}$ (Neimn, 1984). The set $N(F; Q)$ is called stable if every $X \in N(F; Q)$ is stable. And the set $N(F; Q)$ is called completely unstable if every $X \in N(F; Q)$ is unstable. These are important properties since the equilibrium state of the linear dynamical system

$$\dot{x}(t) = Ax(t) \quad X(t) \in \mathbb{R}^n \quad (1)$$

is absolutely stable if and only if $A$ is stable. In situations where $A$ is known only to the extent of $A \in N(F; Q)$ (due to component tolerances, measurement errors, etc.), one can guarantee the stability of (1) only by knowing that $N(F; Q)$ is stable. In 1983, Masia gave the necessary and sufficient condition for the stability of interval matrices, but this condition is inconvenient (Neimn, 1984) and false (Harish et al., 1984). Recently Neimn (1984) and Xu (1985) presented respectively the simple sufficient conditions for the stability of interval matrices. This paper extends further their results and gives the criteria for completely unstable of interval matrices.

MAIN RESULTS

Theorem 1 (Xu, 1985). Let the matrix $N(a_{ij})$ be defined as

$$\bar{a}_{ij} = \min\{a_{ij}, b_{ij}\} \quad (2)$$

or

$$\bar{a}_{ij} = \max\{a_{ij}, b_{ij}\} \quad (3)$$

If $a_{ij}$, which are the leading principal minors of order $k$ of the matrix $N$, satisfy

$$(-1)^k \bar{a}_{ij} > 0 \quad k=1, 2, \ldots, n. \quad (4)$$
then the set \( X(t; c) \) is stable.

The conditions (2) and (4) in Theorem 1 concentrate all results by Heinen (1984). Utilizing scalar Lyapunov function, we shall extend further these results.

Consider the linear system (1), for \( t \in [t_0, t_1] \) assume \( X(t) \geq 0 \). Let a Lyapunov function of the system (1) be

\[
\dot{V}_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i(t)x_j(t),
\]

where \( a_{ij}(1:x_1, \ldots, x_n) \) are some positive numbers and \( x_i(t) \) is the \( i \)th component of \( X(t) \). We calculate the derivative \( \dot{V}_1 \) along the solution of the system (1) as

\[
\dot{V}_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i(t)x_j(t) \frac{dx_i}{dt} + \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i(t)x_j(t) \frac{dx_j}{dt}.
\]

Let \( \max_{j \in \{1, \ldots, n\}} \frac{\partial a_{ij}}{\partial x_j} a_{ij} \leq -\delta_i \), then

\[
\dot{V}_1 \leq \sum_{i=1}^{n} \delta_i x_i(t)^2 \leq \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i(t)x_j(t),
\]

and

\[
V_1 \leq V_1(t_0),
\]

where \( V_1(0_0) \), \( t_0 \).

Hence for arbitrary \( t \in [t_0, t_1] \) we have

\[
\dot{V}_1(t) \leq -\delta_i(t_0),
\]

where \( V_1(t) \geq V_1(t_0) \) and \( \delta_i(t_0) \) is constant.

For a segment of the solution \( X(t) \) in some other time varying interval, we can by a linear transformation change the segment to positive and repeat the process. For example, assume that the solution \( X(t) \) of (1) has \( k \) negative components for \( t \in (t_1, t_2) \), let

\[
x_{i_1} < 0 \quad i_1=1, \ldots, i_1, x_{i_1} > 0 \quad i_1, \ldots, i_2
\]

where \( i_{i_1} = i_{i_1} < i_2 < \ldots < i_k \) are and \( \xi \) and let

\[
\Delta X = [\Delta x_{i_1}, \ldots, \Delta x_{i_2}],
\]

be an \( n \times n \) diagonal matrix, then \( \Delta X(0) = 0 \) for \( t \in (t_1, t_2) \) and satisfy

\[
\Delta X(t) = (T \Delta X(t)) + (A \Delta X(t)).
\]

Let \( \Delta X = \Delta x_{i_1}, \Delta x_{i_2}, \ldots, \Delta x_{i_2} \), taking \( \Delta X = \Delta x_{i_1}, \Delta x_{i_2}, \ldots, \Delta x_{i_2} \), we obtain as before

\[
\dot{\Delta X} = \Delta X(0) \leq C_0e^{-\delta_0(t_0-t_2)}
\]

where \( \Delta X(0) = \Delta x_{i_1} \) replaces \( A \) of (1), \( C_0 \) is constant and

\[
\dot{\Delta X} = \Delta X(0) \leq C_0 e^{-\delta_0(t_0-t_2)}.
\]