Computational and Combinatorial Methods in Systems Theory
C.I. Byrnes and A. Lindquist (editors)

THE BLOCK JACOBI METHOD FOR COMPUTING THE SINGULAR VALUE DECOMPOSITION

Charles F. Van Loan

Department of Computer Science
Cornell University
Ithaca, New York 14853

Jacobi techniques for computing the singular value and symmetric eigenvalue decompositions have achieved recent prominance because of interest in parallel computation. They are ideally suited for certain multiprocessor systems whose processors are connected in nearest-neighbor fashion. If the processors are powerful enough then block Jacobi procedures are attractive because they render a more favorable ratio of computation to communication. We examine two block Jacobi procedures that differ in how the 2-by-2 subproblems are solved.

1. INTRODUCTION

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

$$U^T A V = \text{diag}(\sigma_1,\ldots,\sigma_n)$$

See Golub and Van Loan (1983). In this paper we analyze a family of methods for computing the SVD that are block generalizations of the parallel Jacobi scheme discussed in Brent, Luk, 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 off(A) denote the Frobenius norm of A's off-diagonal elements, i.e.,

$$\text{off}(A) = \text{sqrt}\left(\sum_{i \neq j} |a_{ij}|^2 \right).$$

For a given dimension let $J(i,j,\theta)$ denote a Jacobi rotation of θ degrees in the (i,j) plane, e.g.,

$$J(2,4,\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & 0 & \sin(\theta) \\ 0 & 0 & 1 & 0 \\ 0 & -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

Forsythe and Henrici (1960) essentially proposed the following Jacobi SVD procedure for square matrices:

Algorithm 1.1

Given A (nxn) and eps > 0, the following algorithm computes n-by-n orthogonal U and V such that $\text{off}(U^TAV) \leq \text{eps} \| A \|_F$. A is overwritten with U^TAV.

```
U := I
V := I
Do While( off(A) > eps ‖ A ‖_F )
   For i = 1 to n-1
      For j = i+1 to n
         Compute cosine sine pairs (c_1,s_1) and (c_2,s_2) so

         [ c_1  s_1 ]^T [ a_ii  a_ij ] [ c_2  s_2 ]   [ d_1  0 ]
         [-s_1  c_1 ]   [ a_ji  a_jj ] [-s_2  c_2 ] = [ 0  d_2 ]

         Set J_1 = J(i,j,θ_1) and J_2 = J(i,j,θ_2) and compute
            the updates

                A := J_1^T A J_2 ; U := U J_1 ; V := V J_2
```

There are several ways to solve the 2-by-2 subproblems. See Brent, Luk, and Van Loan (1985).

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 in that the off-diagonal entries are zeroed in row-by-row fashion. Note that zeroed entries do not stay zero--they generally become nonzero as a result of subsequent rotations. However, it can be shown that

$$\text{off}(J_1^T A J_2)^2 = \text{off}(A)^2 - a_{ij}^2 - a_{ji}^2$$

and thus, A becomes "more diagonal" after each update. A single pass through the body of the While Loop above is called a "sweep". The algorithm usually terminates in 6-10 sweeps for typical values of n and eps, e.g., $n = 100$, eps $= 10^{-12}$.

Jacobi methods, particularly for the symmetric eigenproblem have a very long history. See Jacobi (1846), Henrici(1958), Hansen (1962), 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 analyzed in Hansen (1960). The general form of the block algorithm is given in §2 together with a relevant convergence result. In §3 we discuss the parallel ordering. By zeroing 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 SVD PROCEDURE

A block version of Algorithm 1.1 is easy to specify with suitable notation. Assume that $n = kp$ and that we partition A (n-by-n) as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ A_{21} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{bmatrix} \qquad (A_{ij} \in R^{pxp})$$

The case of nonsquare blocks will be covered in §4. Denote the j-th block columns of A, U, and V by A_j , U_j , and V_j . Note that these are n-by-p matrices and

and that $A = [A_1, \ldots, A_k]$, $U = [U_1, \ldots, U_k]$, and $V = [V_1, \ldots, V_k]$. If

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{matrix} p \\ p \end{matrix}$$
$$\quad p \quad p$$

is orthogonal then we let $J(i,j,Q) = (Z_{ij})$ denote the k-by-k block matrix with p-by-p blocks that is the identity everywhere except $Z_{ii} = Q_{11}$, $Z_{ij} = Q_{12}$, $Z_{ji} = Q_{21}$, and $Z_{jj} = Q_{22}$.

With this notation we have the following procedure:

Algorithm 2.2.

Given A (n-by-n), the partitioning (2.1), and eps > 0, the following algorithm computes n-by-n orthogonal matrices U and V such that

$$\text{off}(U^T A V) \leq \text{eps} \, \| A \|_F$$

A is overwritten by $U^T A V$.

```
U := I
V := I
Do While (off(A) ≥ eps ||A||_F )
   For i = 1 to k-1
      For j = i+1 to k
         Compute the SVD  U_0^T A_0 V_0 = Σ_0  where
                          A_0 = [ A_ii  A_ij ]
                                [ A_ji  A_jj ]
         Set  J_1 = J(i,j,U_0)  and J_2 = J(i,j,V_0)  and perform the
              updates  A := J_1^T A J_2 , U := UJ_1 , and V := VJ_2
```

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

$$\text{off}(J_1^T A J_2)^2 = \text{off}(A)^2 - \| A_{ij} \|_F^2 - \| A_{ji} \|_F^2 - \text{off}(A_{ii})^2 - \text{off}(A_{jj})^2$$

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_{ij} is skipped if $|a_{ij}| < \tau$ where τ is the (usually small) threshold parameter. The size of τ may be fixed or it may vary from sweep to sweep. See Rutishauser (1966). In the block situation we pass over the (i,j) subproblem if

$$\mu(A,i,j) = \text{sqrt}[\| A_{ij} \|_F^2 + \| A_{ji} \|_F^2 \tag{2.2}$$

is small according to the current value of τ.

and that $A = [A_1, \ldots, A_k]$, $U = [U_1, \ldots, U_k]$, and $V = [V_1, \ldots, V_k]$. If

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{matrix} p \\ p \end{matrix}$$
$$\quad p \quad p$$

is orthogonal then we let $J(i,j,Q) = (Z_{ij})$ denote the k-by-k block matrix with p-by-p blocks that is the identity everywhere except $Z_{ii} = Q_{11}$, $Z_{ij} = Q_{12}$, $Z_{ji} = Q_{21}$, and $Z_{jj} = Q_{22}$.

With this notation we have the following procedure:

Algorithm 2.2.

Given A (n-by-n), the partitioning (2.1), and eps > 0, the following algorithm computes n-by-n orthogonal matrices U and V such that

$$\text{off}(U^TAV) \leq \text{eps} \, \| A \|_F$$

A is overwritten by U^TAV.

```
U := I
V := I
Do While (off(A) ≥ eps ||A||_F )
    For i = 1 to k-1
        For j = i+1 to k
            Compute the SVD  U_0^T A_0 V_0 = Σ_0  where
                                  A_0 = [ A_ii  A_ij ]
                                        [ A_ji  A_jj ]
            Set  J_1 = J(i,j,U_0)  and J_2 = J(i,j,V_0)  and perform the
                 updates  A := J_1^T A J_2 , U := UJ_1 , and V := VJ_2
```

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

$$\text{off}(J_1^TA\,J_2)^2 = \text{off}(A)^2 - \| A_{ij} \|_F^2 - \| A_{ji} \|_F^2 - \text{off}(A_{ii})^2 - \text{off}(A_{jj})^2$$

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_{ij} is skipped if $|a_{ij}| < \tau$ where τ is the (usually small) threshold parameter. The size of τ may be fixed or it may vary from sweep to sweep. See Rutishauser (1966). In the block situation we pass over the (i,j) subproblem if

$$\mu(A,i,j) = \text{sqrt}[\| A_{ij} \|_F^2 + \| A_{ji} \|_F^2 \tag{2.2}$$

is small according to the current value of τ.

The threshold parameter 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 off(A) is small enough, we use its block analogue:

$$OFF(A)^2 = \sum_{i \neq j} \|A_{ij}\|_F^2$$

By terminating when OFF(A) is small the final matrix A will be nearly block diagonal. The diagonalization process is then completed by computing the SVD's of the diagonal blocks (in parallel).

In Algorithm 2.1 the 2-by-2 block subproblems are exactly diagonalized. As we are about to point out, complete diagonalization of the subproblems is unnecessary and so we merely insist that if we compute

$$\begin{bmatrix} B_{ii} & B_{ij} \\ B_{ji} & B_{jj} \end{bmatrix} = U_0^T \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{bmatrix} V_0 \tag{2.3}$$

then

$$\|B_{ij}\|_F^2 + \|B_{ji}\|_F^2 \leq \theta^2 \mu(A,i,j)^2 \tag{2.4}$$

for some fixed $\theta < 1$ .Recommended values for θ are discussed in §4.

The last feature of Algorithm 2.1 that we wish to relax concerns the ordering. Instead of just considering the row-cyclic order, we wish to consider the general ordering

$$(i_1,j_1)\ ,\ (i_2,j_2)\ ,\ \ldots,\ (i_r,j_r) \qquad r = k(k-1)/2 \tag{2.5}$$

where $i_m < j_m$ for m = 1 to r . Overall we obtain

Algorithm 2.2

Given A (n-by-n) , eps > 0 , $0 \leq \theta < 1$, partitioning (2.1), ordering (2.5), and a threshold τ satisfying

$$\tau \leq \text{eps}\|A\|_F / k\ ,$$

the following algorithm computes orthogonal U and V such that

$$OFF(U^T A V) \leq \text{eps}\ \|A\|_F$$

```
U := I
V := I
Do While ( OFF(A) > eps||A||_F )
    For m = 1 to k(k-1)/2
        (i,j) = (i_m , j_m)
        If  μ(A,i,j) ≥ τ^2
           then
                Compute orthogonal U_0 and V_0 such that (2.4) holds.
                Let J_1 = J(i,j,U_0) and J_2 = J(i,j,V_0) and compute the
                     updates  A := J_1^T A J_2 , U := UJ_1 , V := VJ_1 .
```

There are several details associated with Algorithm 2.2 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 rectangular, and (e) the value of θ. However, before we take up these very practical matters we confirm that the preceeding algorithm converges.

Theorem 2.1

Algorithm 2.2 terminates after a finite number of block sweeps.

Proof

If no subproblems are solved during a particular block sweep then we have $\mu(A,i,j) < \tau$ for all i and j that satisfy $1 \leq i < j \leq k$. Thus,

$$OFF(A)^2 = \sum_{i<j} \mu(A,i,j)^2 \leq k(k-1)\tau^2/2 \leq eps^2 \, \| A \|_F^2$$

and termination is achieved.

On the other hand, if subproblem (i,j) is solved during a block sweep it is easy to show using the definition (2.2) that the updated matrix satisfies

$$(2.4) \quad OFF(J_1^T A J_2)^2 = OFF(A)^2 - \mu(A,i,j)^2 + \mu(J_1^T A J_2, i, j)^2$$
$$\leq OFF(A)^2 - (1 - \theta^2)\mu(A,i,j)^2 .$$

But if subproblem (i,j) is solved then $\tau \leq \mu(A,i,j)$ and so from (2.4) we have

$$OFF(J_1^T A J_2)^2 \leq OFF(A)^2 - \tau^2(1 - \theta^2)$$

Thus, after s block sweeps OFF of the original A is diminished by

$$s\,\tau\,\mathrm{sqrt}(1 - \theta^2)$$

It follows that the condition $OFF(A) \leq eps \, \| A \|_F$ must eventually be satisfied.

Q.E.D.

See Hansen (1960) for further results pertaining to the convergence of block Jacobi procedures.

3. BLOCK JACOBI SVD WITH PARALLEL ORDERING

The key to speeding up Algorithm 2.2 is to solve nonconflicting subproblems concurrently. For example, if $k = 8$ then the (1,2), (3,4), (5,6), and (7,8) subproblems are nonconflicting in that with 4 processors we could solve the 4 subproblems and perform the necessary updates of A, U, and V at the same time. For general k we may proceed as follows:

Algorithm 3.1

Suppose $A = [A_1, \ldots, A_k]$, $U = [U_1, \ldots, U_k]$, and $V = [V_1, \ldots, V_k]$ where each block column is n-by-p. Assume that k is even and that we have $N = k/2$ processors $P_1, \ldots, P_N$ 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 N, then

$$A := [J(1,2(U(1)) \cdot \cdot \cdot J(k-1,k,U(N))]^T A [V(1,2,V(1)) \cdot \cdot \cdot J(k-1,k,V(N))]$$

$$U := U \text{ diag}(U(1) ,..., U(N))$$

$$V := V \text{ diag}(V(1) ,..., V(N))$$

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

Solve subproblem (2i-1,2i) as in Algorithm 2.2. Let U(i) and V(i) be the resulting orthogonal matrices.

$$[A_{2i-1} , A_{2i}] := [A_{2i-1} , A_{2i}] V(i)$$

$$[U_{2i-1} , U_{2i}] := [U_{2i-1} , U_{2i}] U(i)$$

$$[V_{2i-1} , V_{2i}] := [V_{2i-1} , V_{2i}] V(i)$$

Transmit U(i) to every other processor.

Collect $U(1) ,..., U(i-1) , U(i+1) ,..., U(N)$.

$$[A_{2i-1} , A_{2i}] := \text{diag}(U(1) ,..., U(N)) [A_{2i} , A_{2i-1}]$$

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 in between applications of the procedure. To illustrate, we return to our example where A is 8-by-8 as a block matrix (k = 8). Partition the set of 28 off-diagonal index pairs into seven "rotation sets" as follows:

I.	(1,2)	(3,4)	(5,6)	(7,8)
II.	(1,4)	(2,6)	(3,8)	(5,7)
III.	(1,6)	(4,8)	(2,7)	(3,5)
IV.	(1,8)	(6,7)	(4,5)	(2,3)
V.	(1,7)	(5,8)	(3,6)	(2,4)
VI.	(1,5)	(3,7)	(2,8)	(4,6)
VII	(1,3)	(2,5)	(4,7)	(6,8)

The four SVD 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 I :	1	3	5	7	
	2	4	6	8	
Round II :	1	2	3	5	
	4	6	8	7	
Round III:	1	4	2	3	etc.
	6	8	7	5	

In the matrix setting we start off with A residing in processors $P_1 ,..., P_4$ as follows:

P_1		P_2		P_3		P_4	
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 reapportion A among the processors as follows:

P_1		P_2		P_3		P_4	
A_{11}	A_{14}	A_{12}	A_{16}	A_{13}	A_{18}	A_{15}	A_{17}
A_{41}	A_{44}	A_{42}	A_{46}	A_{43}	A_{48}	A_{45}	A_{47}
A_{21}	A_{24}	A_{22}	A_{26}	A_{23}	A_{28}	A_{25}	A_{27}
A_{61}	A_{64}	A_{62}	A_{66}	A_{63}	A_{68}	A_{65}	A_{67}
A_{31}	A_{34}	A_{32}	A_{36}	A_{33}	A_{38}	A_{35}	A_{37}
A_{81}	A_{84}	A_{82}	A_{86}	A_{83}	A_{88}	A_{85}	A_{87}
A_{51}	A_{54}	A_{52}	A_{56}	A_{53}	A_{58}	A_{55}	A_{57}
A_{71}	A_{74}	A_{72}	A_{76}	A_{73}	A_{78}	A_{75}	A_{77}

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 (chess tournament round), i.e., subproblems (1,6) , (4,8) , (2,7) , and (3,5) , etc. In the N processor situation the shuffling is most easily described through the n-by-n (n = kp) permutation matrix

$$P = [\, E_1 , E_4 , E_2 , E_6 , E_3 , E_8 , \ldots , E_{k-5} , E_k , E_{k-3} , E_{k-1} \,]$$

where the E_i are block columns of the n-by-n identity:

$$I_n = [\, E_1 , E_2 , \ldots , E_k \,] \qquad E_i \in R^{nxp} , \; n = kp$$

In particular, after each application of Algorithm 3.1 block columns 2i-1 and 2i of the matrix P^TAP are stored in processor P_i . Overall we have

Algorithm 3.2

Suppose $A = [\, A_1 , \ldots , A_k \,]$, $U = [\, U_1 , \ldots , U_k \,]$, and $V = [V_1 , \ldots , V_k \,]$ are given with U = V = I. Assume that each block column has dimension n-by-p and that processor P_i contains block columns 2i-1 and 2i of A , U, and V for i = 1 to

N = k/2 . Given eps > 0 the following algorithm computes orthogonal U and V such that $OFF(U^T A V) \leq eps \| A \|_F$.

```
Do While ( OFF(A) > eps||A||_F )
    For rotation_set = 1 to N-1
        Apply Algorithm 3.1
        Perform the updates A := P^T AP , U := UP , and V := VP where P
            is given by (3.1). Note that this involves shuffling A , U,
            and V among the processors.
```

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

4. PRACTICAL DETAILS AND EXPERIENCE

Applying the Orthogonal Transformations

Most of the computational effort in Algorithm 3.1 is spent calculating products of the form ZC and $C^T Z$ where Z is a 2p-by-2p orthogonal matrix and C is 2p-by-n. The obvious method for doing this requires $4np^2$ flops. However, there is an interesting alternative that begins by computing the Householder upper triangularization of Z:

$$H_{2p-1} \cdots H_1 Z = R .$$

Since the upper triangular matrix R is also orthogonal, we must have R = diag(±1). To compute ZC, for example, we sequentially apply the Householder matrices:

```
C := RC
For i = 1 to 2p-1
    C := H_i C
```

If we just count flops this "Householder" approach requires $2(4np^2 + 8p^3)/3$ flops and is thus more economical whenever p < n/4. (This is often the case in multiprocessor environments as it implies k > 4 .) Moreover, representing Z as a product of Householders requires half the space of the conventional representation. This allows for a reduction in the communication costs associated with the transmission 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

$$A_0 = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} p \\ p \end{matrix}$$
$$\quad\; p \quad\; p$$

and must choose U_0 and V_0 such that

$$U_0^T A_0 V_0 = B_0 = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

satisfies

$$(4.1)\quad \| B_{12} \|_F^2 + \| B_{21} \|_F^2 \leq \theta^2 [\, \| A_{12} \|_F^2 + \| A_{21} \|_F^2 \,] = \theta^2 \, \mu(A,i,j)^2$$

for some $\theta < 1$. (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 sweeping until A_0 is sufficiently close to 2-by-2 block diagonal form.

$$\text{Cost} = 50p^3 \text{ flops/per sweep}$$

Method 2. Golub-Reinsch SVD with Bidiagonal Pause

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

$$U_B^T A_0 V_B = \begin{matrix} x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \end{matrix} \qquad (p = 4)$$

where $U_B = H_1 \cdots H_p$ and $V_B = G_1 \cdots G_p$ are products of Householder matrices. Note that the $(p,p+1)$ entry prevents the reduced matrix from being block diagonal. This suggests that if

$$|b| \leq \theta \, \mu(A,i,j)$$

then B_0 is sufficiently close to block diagonal form and we set $U_0 = U_B$ and $V_0 = V_B$. If b is too large then we proceed to the iterative portion of the Golub-Reinsch algorithm terminating as soon as the current $(p,p+1)$ entry is small enough.

$$\text{Cost} = \begin{cases} 18p^3 & \text{if the bidiagonal pause is successful} \\ 80p^3 & \text{otherwise} \end{cases}$$

Method 1 is appealing because it can exploit the fact that the subproblems are increasingly block diagonal as the iteration progresses. On the other hand, Method 2 is attractive because it is cheaper whenever the bidiagonal pause is successful or whenever two or more sweeps are needed by Method 1. 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-2. The subproblems thus have dimension 8-by-4. If Method 2 is applied and the full SVD computed then we obtain

$$\begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that the structure of the reduced matrix conflicts with the aim of block diagonalization--the (1,2) block is nonzero. However, there is a simple solution: interchange rows 4 and 5 with rows 6 and 7. In general, straightforward row and column interchanges following the SVD computation 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 techniques work for Method 1, except that the rectangular subproblems must be made square by adding zero columns (or rows) before the Jacobi algorithm is applied. This manuever is discussed in Brent, Luk, and Van Loan (1985) and is somewhat costly. To be precise, let L and S be the larger and smaller of the subproblem dimension. The Jacobi approach involves $O(L^3)$ work while the Golub-Reinsch procedure requires $O(L^2S)$ flops.

Experimental Observations

To gain perspective on the various options sketched above we ran numerous examples on a VAX 780 in the MATLAB environment. (Machine precision $\cong 10^{-16}$.) We report on one typical 24-by-24 example which we solved using various values of θ, k, and p . In this example $\| A \|_F \cong 15$ and we terminate as soon as $OFF(A) \leq 10^{-15}$.

To begin with, the asymptotic rate of convergence is quadratic. Consider the $k = 6$, $p = 4$ situation with Method 2. Tabulating OFF(A) we find

Sweep	$\theta = 10^{-15}$	$\theta = .25$	$\theta = .50$	$\theta = .75$
1	$.97 \times 10^{1}$	$.11 \times 10^{1}$	$.12 \times 10^{1}$	$.14 \times 10^{1}$
2	$.22 \times 10^{1}$	$.28 \times 10^{1}$	$.41 \times 10^{1}$	$.61 \times 10^{0}$
3	$.19 \times 10^{0}$	$.34 \times 10^{0}$	$.10 \times 10^{0}$	$.13 \times 10^{0}$
4	$.32 \times 10^{-3}$	$.48 \times 10^{-3}$	$.27 \times 10^{-1}$	$.67 \times 10^{-1}$
5	$.36 \times 10^{-8}$	$.70 \times 10^{-6}$	$.23 \times 10^{-4}$	$.49 \times 10^{-4}$
6	conv	conv	$.24 \times 10^{-11}$	$.72 \times 10^{-10}$
7	conv	conv	conv	conv

In general, we find that the number of block sweeps is a mildly increasing function of θ. Indeed, we have found that the number of block sweeps is usually minimal so long as $\theta \in (0,.25]$. Here we report on the number of block sweeps that are necessary for various k, p, and θ. (Again, Method 2 was used.)

k	p	$\theta = 10^{-15}$	$\theta = .25$	$\theta = .50$	$\theta = .75$
3	8	4	4	5	5
4	6	5	5	6	7
6	4	6	6	7	7
8	3	6	6	8	8
12	2	6	7	8	10

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 times that the bidiagonal pause is successful rapidly decreases as the iteration proceeds. For example, in the $(k,p,\theta) = (4,6,.5)$ situation, the bidiagonal pause 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 thus far reported on have all been with Method 2, they essentially apply when the subproblems are solved using Method 1, the Jacobi approach. Thus, how one solves the subproblem doesn't really matter from the standpoint of block sweeps. Moreover, for $\theta = .25$ we find that two sweeps in Method 1 almost always suffice to solve the subproblem. Based on flop counts we see from above that the two methods are equally efficient. Method 1, being simpler to implement, may be preferable in situautions when program memory in the processors is limited.

Acknowledgement

The author wishes to thank Clare Chu for discussing the computations presented in this paper.

References

[1] Brent, R, F. Luk, and C. Van Loan, Computation of the singular value decomposition using mesh-connected processors, J. VLSI and Computer Systems, 1, (1985), 242-270.

[2] Forsythe, G. and P. Henrici, The cyclic Jacobi method for computing the principal values of a complex matrix, Trans. Amer. Math. Soc., 94 (1960), 1-23.

[3] Golub, G.H. and C. Van Loan, Matrix Computations, (Johns Hopkins University Press, 1983).

[4] Hansen, E.R. , On Jacobi methods and block Jacobi methods for computing matrix eigenvalues, Ph.D. thesis, Stanford University, 1960.

[5] Hansen, E.R., On quasi-cyclic Jacobi methods, ACM, 9(1962), 118-135.

[6] Henrici, P., On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing the eigenvalues of Hermitian matrices, SIAM J. Applied Math., 6 (1958) , 144-162.

[7] Jacobi, C. , Uber ein leiches vehfahren die in der theorie der sacular storungen vorkommendern gleichungen numerisch aufzulosen, Crelle's Journal, 30 (1846), 51-94.

[8] Rutishauser, H., The Jacobi method for real symmetric matrices, Numer.Math., 16 (1966), 205-223.

[9] Schonhage, A, On the quadratic convergence of the Jacobi process, Numer. Math., 6 (1964), 410-412.

Computational and Combinatorial Methods in Systems Theory
C.I. Byrnes and A. Lindquist (editors)

STABILITY CRITERIA FOR INTERVAL MATRICES

Xu Daoyi

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 Heinen (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 $n\times n$ interval matrix $N(P,Q)$ is a set of real matrices

$$N(P,Q)=(A=(a_{ij}):p_{ij}\le a_{ij}\le q_{ij},\ i,j=1,2,\ldots,n)$$

(Neinen,1984). The set $N(P,Q)$ is called stable if every $A\in N(P,Q)$ is stable. And the set $N(P,Q)$ is called completely unstable if every $A\in N(P,Q)$ is unstable. These are important properties since the equilibrium state of the linear dynamical system

$$\dot{X}(t)=AX(t) \qquad X(t_o)=X_o \tag{1}$$

is absolutely stable if and only if A is stable. In situations where A is known only to the extent of $A\in N(P,Q)$ (due to component tolerances, measurement errors,etc.),one can guarantee the stability of (1) only by knowing that $N(P,Q)$ is stable. In 1983, Białas gave the necessary and sufficient condition for the stabllity of interval matrices, but this condition is inconvenient (Neinen,1984) and false (Barmish et al, 1984). Recently Heinen (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 $M=(m_{ij})$ be defined as

$$m_{ii}=q_{ii}<0,\ m_{ij}=\max(|p_{ij}|,|q_{ij}|)\quad (i\neq j) \tag{2}$$

or

$$m_{ii}=2q_{ii}<0,\ m_{ij}=\max(|p_{ij}+p_{ji}|,|q_{ij}+q_{ji}|)\quad (i\neq j)^{i,j=1,\ldots,n.} \tag{3}$$

If Δ_k, which are the leading principal minors of order k of the matrix M, satisfy

$$(-1)^k\Delta_k>0 \quad k=1,2,\ldots,n, \tag{4}$$

then the set $N(P,Q)$ 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 $X(t)\geqslant 0$. Let a Lyapunov function of the system (1) be

$$V_1=\sum_{i=1}^{n} d_i^{(1)} x_i(t),$$

where $d_i^{(1)}(i=1,\ldots,n)$ are some positive numbers and $x_i(t)$ is the ith component of $X(t)$. We calculate the derivative $\dot{V}_1$ along the solution of the system (1) as

$$\dot{V}_1=\sum_{i=1}^{n} d_i^{(1)}\dot{x}_i=\sum_{i=1}^{n} d_i^{(1)}\sum_{j=1}^{n} a_{ij}x_j=\sum_{j=1}^{n}\sum_{i=1}^{n} d_i^{(1)} d_j^{(1)^{-1}} a_{ij}\, d_j^{(1)} x_j$$

Let $\max_j \sum_{i=1}^{n} d_i^{(1)} d_j^{(1)^{-1}} a_{ij}=-\delta_1$, then

$$\dot{V}_1\leqslant -\delta_1 \sum_{j=1}^{n} d_j^{(1)} x_j = -\delta_1 V_1$$

and

$$V_1\leqslant V_{10} e^{-\delta_1(t-t_0)}, \text{ where } V_{10}=V_1(t_0).$$

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

$$\|X(t)\|\leqslant C_1 e^{-\delta_1(t-t_0)}$$

where $\|X(t)\|=\sum_{i=1}^{n}|x_i(t)|$ and $C_1=\dfrac{V_{10}}{\min\{d_i\}}$ 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

$$x_i<0 \quad i=r_1,\ldots,r_k; \qquad x_i\geqslant 0 \quad i\neq r_1,\ldots,r_k$$

where $1\leqslant r_1<r_2<\cdots r_k\leqslant n$ and $1\leqslant i\leqslant n$. And let

$$T=\operatorname{diag}(1,\ldots,1,\overset{r_1}{-1},1,\ldots,1,\overset{r_k}{-1},1,\ldots,1)$$

be an $n\times n$ diagonal matrix, then $TX(t)\geqslant 0$ for $t\in[t_1,t_2]$ and satisfy

$$T\dot{X}(t)=(TAT^{-1})TX(t). \tag{5}$$

Let $TX=(x_1^{(2)},x_2^{(2)},\ldots,x_n^{(2)})^T$, taking $V_2=\sum_{i=1}^{n} d_i^{(2)} x_i^{(2)}$ $(d_i^{(2)}>0, i=1,\ldots,n)$, we obtain as before

$$\|X\|=\|TX\|\leqslant C_2 e^{-\delta_2(t-t_0)} \quad \text{for } t\in[t_1,t_2]$$

where $TAT^{-1}\triangleq B=(b_{ij})$ replaces A of (1), C_2 is constant and

$$-\delta_2=\max_j \sum_{i=1}^{n} d_i^{(2)} d_j^{(2)^{-1}} b_{ij}.$$