The Block Jacobi Method for Computing the Singular Value Decomposition

Charles Van Loan
TR 85-680
June 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

The Block Jacobi Method for Computing the Singular Value Decomposition

Charles Van Loan
Department of Computer Science
405 Upson Hall
Cornell University
Ithaca, New York 14853

Abstract

Jacobi techniques for computing the symmetric eigenvalue and singular value decompositions have achieved recent prominance because of interest in parallel computation. They are ideally suited for certain multiprocessor systems having processors that are connected in nearest neighbor fashion. If the processors are reasonably powerful and have significant local memory, then block Jacobi procedures are attractive because they render a more favorable computation to communication ratio. This paper examines some of the practical details associated with two block Jacobi methods for the singular value decomposition. The methods differ in how the 2-by-2 subproblems are solved.

This paper was presented at the 7th International Symposium on the Mathematical Theory of Networks and Systems, Stockholm, Sweden, June 10-14, 1985. The research was partially supported by ONR Contract N0001483-K-0640.

§1. Introduction

The singular value decomposition (SVD) of a matrix $A \in R^{m \times n}$ $(m \geq n)$ has many important applications. In the SVD we seek real orthogonal U (mxm) and V (nxn) such that $U^T AV = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$. See Golub and Van Loan (1983). It is usually assumed that $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ but we will not insist upon this normalization. In this paper we examine a method for computing the SVD that is a block generalization of the parallel Jacobi scheme discussed in Brent, Luk, and Van Loan (1985). Our focus is on the behavior of our algorithm and not upon the details of its parallel implementation. We will document the implementation of our scheme in a multiple array processor environment in another paper.

Jacobi procedures proceed by making A increasingly diagonal by solving a judicious sequence of 2-by-2 SVD subproblems. Suppose A is square and let off(A) denote the Frobenious norm of A's off-diagonal elements, i.e.,

$$\mathrm{off}(A) = \mathrm{sqrt}\,\Big(\sum_{i \neq j} |a_{ij}|^2 \Big).$$

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 $A \in R^{n \times n}$ and eps > 0, the following algorithm computes n-by-n orthogonal matrices U and V such that $\mathrm{off}(U^T AV) \leq \mathrm{eps} \cdot \| A \|_F$. A is overwritten by $U^T AV$.

$U \leftarrow I$
$V \leftarrow I$
Do While ($\mathrm{off}(A) > \mathrm{eps}\cdot\|A\|_F$)
  For i = 1:n-1
    For j = i+1:n
      Compute cosine-sine pairs (c_1,s_1) and (c_2,s_2) such that

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}$$

      Set $J_1 = J(i,j,\theta_1)$ and $J_2 = J(i,j,\theta_2)$ and compute the updates
        $A \leftarrow J_1^T A J_2$, $U \leftarrow UJ_1$, and $V \leftarrow VJ_2$.

There are several ways to solve the 2-by-2 subproblems. For details, 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 elements are zeroed in row-by-row fashion. Note that zeroed entries do not stay zero--they generally become nonzero as the result of subsequent rotations. However, it can be shown that

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

and thus, A becomes "more diagonal" after each update. A single pass through the While-loop is called a *sweep* . The algorithm usually terminates in 6-10 sweeps for typical n and eps , e.g., $n \approx 100$, $\mathrm{eps} \approx 10^{-12}$.

Jacobi methods, particularly for the symmetric eigenvalue problem, have a long history. See Jacobi (1846), Henrici(1958), Hansen (1962) , and Schonhage(1964) as well as the discussion and references in Golub and Van Loan (1983, p.295ff). 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 blocks according to this 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.

We mention that just about everything in this paper carries over to the symmetric eigenvalue problem--just set $U = V$ in the sequel.

§2. A Block Jacobi SVD Procedure

A block version of Algorithm 1.1 is easy to specify with suitable notation. Assume $n = kp$ and that we partition $A \in R^{n\times n}$ as follows:

$$(2.1) \qquad A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ A_{21} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{bmatrix} \qquad A_{ij} \in R^{p\times p}$$

(The case of nonsquare blocks will be discussed 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 that we have $A = [\, A_1, \ldots, A_k \,]$, $U = [\, U_1, \ldots, U_k \,]$, and $V = [\, V_1, \ldots, V_k \,]$. If

$$Q = \begin{array}{cc} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} & \begin{array}{c} p \\ p \end{array} \\ \begin{array}{cc} p & p \end{array} & \end{array}$$

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 2.1

Given $A \in R^{n\times n}$, the partitioning (2.1) and eps > 0 , the following algorithm computes n-by-n orthogonal matrices U and V such that $\mathrm{off}(U^TAV) \le \mathrm{eps}\cdot\|A\|_F$. A is overwritten by U^TAV.

$U \leftarrow I$
$V \leftarrow I$
Do While ($\mathrm{off}(A) > \mathrm{eps}\cdot\|A\|_F$)
    For $i = 1 : k-1$
        For $j = i+1 : k$
            Compute the SVD $U_0{}^TA_0V_0 = \Sigma_0$ where

$$A_0 = \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{bmatrix}$$

            Set $J_1 = J(i,j,U_0)$ and $J_2 = J(i,j,V_0)$ and perform updates $A \leftarrow J_1{}^TA$, $A \leftarrow AJ_2$, $U \leftarrow UJ_1$, and $V \leftarrow VJ_2$) , i.e.,

$$\begin{bmatrix} A_{iq} \\ A_{jq} \end{bmatrix} \leftarrow U_0{}^T \begin{bmatrix} A_{iq} \\ A_{jq} \end{bmatrix} \quad , \quad q = 1{:}k$$

$$[A_i \, , A_j] \leftarrow [A_i \, , A_j]V_0$$
$$[U_i \, , U_j] \leftarrow [U_i \, , U_j]U_0$$
$$[V_i \, , V_j] \leftarrow [V_i \, , V_j]V_0$$

One pass through the While loop is called a *block sweep*. Analogous to the scalar case, A becomes "more diagonal" after each update. Indeed it is not hard to show that

$$\mathrm{off}(J_1{}^TAJ_2)^2 = \mathrm{off}(A)^2 - \|A_{ij}\|_F{}^2 - \|A_{ji}\|_F{}^2 - \mathrm{off}(A_{ii})^2 - \mathrm{off}(A_{jj})^2 .$$

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 eigenproblem. 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

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

is small according to the threshold criteria.

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 we use its block analog:

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

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

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

then

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

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

The last feature of Algorithm 2.1 that we wish to amend concerns the ordering. Instead of just considering the cyclic-by-row scheme we wish to consider the general ordering

$$(2.3) \qquad (i_1,j_1) \ , \ (i_2,j_2) \ ,\dots, \ (i_r,j_r)$$

where $r \equiv k(k-1)/2$ and $i_m < j_m$ for $m = 1 : r$. Overall we obtain

Algorithm 2.2

Given $A \in R^{n\times n}$, eps > 0, $0 \le \theta < 1$, partitioning (2.1), ordering (2.3), and a threshold $\tau \le$ eps $\| A \|_F / k$, the following algorithm computes orthogonal U and V such that $\mathrm{OFF}(U^TAV) \le$ eps $\| A \|_F$.

U ← I
V ← I
Do While (OFF(A) > eps $\| A \|_F$)
  For m = 1:k(k-1)/2
    $(i,j) \leftarrow (i_m\ j_m)$
    If $\mu(A,i,j) \ge \tau$
      then
        Compute orthogonal U_0 and V_0 such that if

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

        then

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

      Let $J_1 = J(i,j,U_0)$ and $J_2 = J(i,j,V_0)$ and perform the updates $A \leftarrow J_1^TAJ_2$, $U \leftarrow UJ_1$, and $V \leftarrow VJ_2$. (See Algorithm 2.1 for details.)

There are several details associated with Algorithm 2.2 that we pursue in the next 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 values for τ and 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 $1 \le i < j \le k$. Thus,

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

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 $J_1^TAJ_2$ matrix satisfies:

$$\begin{aligned} OFF(J_1^TAJ_2)^2 &= OFF(A)^2 - \mu(A,i,j)^2 + \mu(J_1^TAJ_2,\ i,j)^2 \\ &\le OFF(A)^2 - (1-\theta^2)\ \mu(A,i,j)^2 \end{aligned} \tag{2.4}$$

But if subproblem (i,j) is solved then $\tau \le \mu(A,i,j)$ and so from (2.4) we have $OFF(J_1^TAJ_2)^2 \le off(A)^2 - \tau^2(1-\theta^2)$. Thus, after s block sweeps OFF of the original A is diminished by $s\cdot\tau\cdot sqrt(1-\theta^2)$. It follows that the condition $OFF(A) \le eps\cdot\| A \|_F$ must eventually be satisfied. ▌

See Hansen (1960) for further results pertaining to the convergenceof block Jacobi methods.

§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 four processors we could solve the four 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 $A = [\, A_1, \dots, A_k \,]$, $U = [\, U_1, \dots, U_k \,]$, and $V = [\, V_1, \dots, V_k \,]$, where $n = kp$ and each block column is in $R^{n \times p}$. Assume that k is even, that we have $N = k/2$ processors $P_1, \dots, P_N$, and that P_i contains the block columns A_{2i-1}, A_{2i}, U_{2i-1}, U_{2i}, V_{2i-1}, and V_{2i}. If for $i = 1$ to N, P_i executes the following algorithm, then

$$
\begin{aligned}
A &\leftarrow [J(1,2,U(1)) \cdots J(k-1,k,U(N)) \,]^T A [J(k-1,k,V(1)) \cdots J(1,2,V(N)) \,] \\
U &\leftarrow U \,\mathrm{diag}(\, U(1), \dots, U(N)\,) \\
V &\leftarrow V \,\mathrm{diag}(\, V(1), \dots, V(N)\,)
\end{aligned}
$$

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.

$$
\begin{aligned}
[A_{2i-1}, A_{2i}\,] &\leftarrow [\, A_{2i-1}, A_{2i}\,] V(i) \\
[\, V_{2i-1}, V_{2i}\,] &\leftarrow [\, V_{2i-1}, V_{2i}\,] V(i) \\
[\, U_{2i-1}, U_{2i}\,] &\leftarrow [\, U_{2i-1}, U_{2i}\,] U(i)
\end{aligned}
$$

Broadcast U(i) to every other processor.
Collect $U(1), \cdots, U(i-1), U(i+1), \cdots, U(N)$.

$$[\, A_{2i-1}, A_{2i}\,] \leftarrow \mathrm{diag}(U(1), \dots, U(N)\,)^T [\, A_{2i-1}, A_{2i}\,] .$$

Note that some coordination is required among the processors. This very important detail will be dealt with in another paper.

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 the applications of the procedure. To illustrate we return to our example where A is an 8-by-8 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 can be easily derived 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
	6	8	7	5

etc.

In the matrix setting we start off with A residing in processors P_1, P_2, P_3, and 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 solved via Algorithm 3.1. To get ready for subproblems (1,4) , (2,6) , (3,8) , and (5,7) we reapportion A among the processorsas 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 equivalent to solving the (1,4) , (2,6) , (3,8) , and (5,7) subproblems of the unpermuted A--precisiely what we are supposed to do.

Next, we apply Algorithm 3.1 and shuffle the resulting matrix exactly as before. 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

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

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

$$I_n = [\, E_1 , E_2 , \dots , E_k \,] \qquad E_i \in R^{n \times p} , \quad n = kp$$

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 $A = [\, A_1 ,\dots, A_k \,]$, $U = [\, U_1 ,\dots, U_k \,] = I_n$ and $V = [\, V_1 ,\dots, V_k \,] = I_n$ are given where $n = kp$ and each block column is nxp. Assume that we have $N = k/2$ processors and that processor i contains block columns 2i-1 and 2i of A , U , and V . Given eps > 0 the following algorithm computes orthogonal U and V such that $OFF(U^TAV) \le eps \, \| A \|_F$.

Do While ($OFF(A) > eps \, \| A \|_F$)
- For rotation.set = 1 : N-1
  - Apply Algorithm 3.1.
  - Perform the updates $A \leftarrow (P^TA)P$, $U \leftarrow UP$, and $V \leftarrow VP$ where P is given by (3.1). (Notice that this implies a reshuffling of the matrices A , U , and V among the processors.)

Readers familiar with the one-sided Hestenes SVD algorithm may be confused. Brent and Luk(1985) discuss the implementation of that procedure on a linear array. We are using a linear array to carry out the 2-sided Jacobi SVD procedure. It would be possible to implement Algorithm 3.2 on a quadratic array, but for expository reasons we have assumed a linear array of 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^TZ where $Z \in R^{2p \times 2p}$ is orthogonal and $C \in R^{2p \times n}$. 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 R is orthogonal, $R = \mathrm{diag}(\pm 1)$. Then, to compute ZC (for example) we sequentially apply the Householder matrices:

$$\begin{array}{l} C \leftarrow RC \\ \text{For } i = 1{:}2p-1 \\ \quad \lfloor C \leftarrow H_i C \end{array}$$

If we just count flops this "Householder" approach requires $^2/_3\,(4np^2 + 8p^3)$ flops and is thus more economical when $p < n/4$. (This is often the case in multiprocessor environments since it implies $k > 4$.) Moreover, representating Z as a product of Householders requires half the space of the conventional representation . This allows for a reduction in 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. 41-2).)

Solving the Subproblems

In the typical subproblem we are presented with a submatrix

$$A_0 = \begin{array}{cc} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} & \begin{array}{c} p \\ p \end{array} \\ \begin{array}{cc} p & \;\; p \end{array} & \end{array}$$

and must chooseorthogonal U_0 and V_0 such that

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

satisfies

(4.1) $\| B_{12} \|_F^2 + \| B_{21} \|_F^2 \;\le\; \theta^2 [\, \| A_{12} \|_F^2 + \| A_{21} \|_F^2 \,] \equiv \theta^2 \mu(A,i,j)^2$.

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 for some θ . That is, keep sweeping until A_0 is sufficiently close to 2-by-2 block diagonal form.

Cost $\approx 50p^3$ flops per sweep

Method 2. (Golub-Reinsch SVD with Bidiagonalization 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 $V_B = G_1 \cdots G_p$ (products of Householders). Note that the (p,p+1) entry is all that prevents the reduced matrix from being block diagonal. This suggests that if $|b| \le \theta\mu(A,i,j)$ then A_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, we complete the bidiagonalization and proceed with the iterative portion of the Golub-Reinsch algorithm terminating as soon as the absolute value of the current (p,p+1) entry is less than $\theta\mu(A,i,j)$.

$$\text{Cost} \approx 18\ p^3 \quad \text{(if bidiagonal pause successful)}$$
$$\approx 80\ p^3 \quad \text{(otherwise)}$$

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 appealing because it is much 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.

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 each block is 4-by-2. The subproblems are thus 8-by-4. If Method 2 is applied and the full SVD is computed then we obtain

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

Note that structure of the reduced matrix conflicts with the aim of block diagonalization--the (1,2) block is nonzero. However, there is a simple remedy : interchange rows 4 and 5 with rows 6 and 7. In general, straight forward row and column interchanges following the SVD computation are sufficient to make Method 2 work on problems with rectangular blocks. (The blocks need not be all of the same dimension.)

The same techniques work with Method 1, except that the rectangular subproblems must be "made square" by adding zero rows (or columns) before the Jacobi procedure can be applied. This manuever is discussed in Brent, Luk, and Van Loan (1985) and is somewhat costly. Let L and S be the larger and smaller of the subproblem's dimensions. The Jacobi approach will involve $O(L^3)$ flops whereas the Golub-Reinsch procedure will require $O(L^2S)$ flops.

Experimental Observations

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

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

Sweep	$\theta = 10^{-15}$	$\theta = .25$	$\theta = .50$	$\theta = .75$
1	$.97\times10^{1}$	$.11\times10^{1}$	$.12\times10^{1}$	$.14\times10^{1}$
2	$.22\times10^{1}$	$.28\times10^{1}$	$.41\times10^{1}$	$.61\times10^{0}$
3	$.19\times10^{0}$	$.34\times10^{0}$	$.10\times10^{0}$	$.13\times10^{0}$
4	$.32\times10^{-3}$	$.48\times10^{-3}$	$.27\times10^{-1}$	$.67\times10^{-1}$
5	$.36\times10^{-8}$	$.70\times10^{-6}$	$.23\times10^{-4}$	$.49\times10^{-4}$
6	conv	conv	$.24\times10^{-11}$	$.72\times10^{-10}$
7			conv	conv

In general, we find that the number of block sweeps is a very 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 necessary for various k , p , and θ.

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.

We mention that when Method 2 is used to solve the subproblems, 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 reported thus far have all been with Method 2, they essentially apply when the subproblems are solved via the Jacobi approach (Method 1). 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 the flop counts given above we see that the two subproblem methods are equally efficient. Method 1, however, is quite a bit simpler to implement and may be preferable in situations when there is limited program memory in the processors.

Acknowledgement

The author wishes to thank Clare Chu for performing some of the computations discussed in this paper.

References

R. Brent and F. Luk (1985), "The Solution of Singular Value and Symmetric Eigenproblems on Multiprocessor Arrays", SIAM J. Scientific and Statistical Computing, 6, 69-84.

R. Brent, F. Luk, and C. Van Loan (1985), "Computation of the Singular Value Decomposition Using Mesh Connected Processors", J. VLSI and Computer Systems 1, 242-270.

G. Forsythe and P. Henrici (1960) , "The Cyclic Jacobi Method for Computing the Principal Values of a Complex Matrix", Trans. Amer. Math. Soc., 94, 1-23.

G.H. Golub and C. Van Loan (1983), *Matrix Computations* , Johns Hopkins University Press, Baltimore Md.

E. Hansen (1960) , "On Jacobi Methods and Block-Jacobi Methods for Computing Matrix Eigenvalues, Ph.D. Thesis, Stanford University, Stanford, California.

E. Hansen (1962), "On Quasicyclic Jacobi Methods", ACM J., 9, 118-135.

P. Henrici (1958), " On the Speed of Convergence of Cyclic and Quasicyclic Jacobi Methods for Computing the Eigenvalues of Hermitian Matrices", SIAM J. Applied Math., 6, 144-62.

C.G.J. Jacobi (1846) ,"Uber ein Leiches Vehfahren Die in der theorie der Sacular-storungen Vorkommendern Gleichungen Numerisch Aufzulosen," Crelle's Journal , 30 , 51-94.

H. Rutishauser (1966), " The Jacobi Method for Real Symmetric Matrices", Numer. Math., 16, 205-223.

A. Schonhage (1964), "On the Quadratic Convergence of the Jacobi Process", Numer. Math., 6 , 410-412.