**On the Schur Decomposition of a₁
Matrix for Parallel Computation.[†]**

Patricia J. Eberlein^{*}
TR 85-689
June 1985

Department of Computer Science
Cornell University
Ithaca, New York 14853

[†] In partial support by NFE 84-10001.

^{*} Visiting Professor, Cornell University, on leave from the University
at Buffalo/SUNY.

On the Schur Decomposition of a Matrix for Parallel Computation

P.J. Eberlein, Visiting Professor, Cornell University, on leave from the University at Buffalo/SUNY

Abstract: An algorithm to solve the eigenproblem for non-symmetric matrices on an $N \times N$ array of mesh connected processors, isomorphic to the architecture described by Brent and Luk for symmetric matrices, is presented. This algorithm is a generalization of the classical Jacobi method, which holds promise for parallel architectures. The rotational parameters for the non-symmetric case are carefully analyzed; many examples of a working program, simulating the parallel architecture are given.

Keywords: Parallel computation, eigenvalues, non-symmetric matrices, mesh-connected processors, Jacobi methods.

On the Schur Decomposition of a Matrix for Parallel

Computation

P.J. Eberlein

1 Introduction

We consider here the problem of finding a parallel method for computing the Schur decomposition of an arbitrary $n \times n$ complex matrix A . In particular, given A , of Frobenius norm one, we wish to determine a unitary matrix U , as a product of a sequence of Jacobi rotations, such that U^*AU is upper triangular. (Note that generally, as many Schur decompositions exist as there are permutations of the diagonal elements.) We define a rotation by $R = (r_{i,j})$, where R is equal to an $n \times n$ identity except for the four matrix elements, $r_{k,k} = r_{m,m} = \cos x$, $r_{k,m} = -e^{i\theta} \sin x$, and $r_{m,k} = e^{-i\theta} \sin x$.

We present here an algorithm, and a working program for solving this problem. The program, as currently implemented, simulates one running on a square array of N^2 mesh-connected processors, where $N=n/2$. This array is isomorphic to that described by Brent and Luk [1]. Like the Brent-Luk model this program will run in $O(n)$ time per sweep. For most matrices, our program exhibits ultimate quadratic convergence similar to the classical Jacobi Method, but for nearly defective matrices, this desirable convergence property may be postponed. We hope that the description of the present algorithm will prove useful to those working in the design of parallel algorithms.

We also discuss some obvious, but infrequently considered differences between the symmetric and non-symmetric problems, as well as the limitation to local information

implicit in parallel computation. After a detailed discussion of the rotation parameters, we briefly describe some heuristics based on experimental and empirical evidence. We believe that further experience and experimentation is necessary before a convergence proof of this, or a similar algorithm, will be found. The reasons for this belief, and some open problems are discussed in the last section of this paper. We also illustrate the properties of the program by numerous examples.

The Jacobi methods should be considered candidates for parallel computation, but for serial computation, the Jacobi method (except for special circumstances), is not recommended.

2 Background

Interest in the generalization of the classical Jacobi method to non-symmetric matrices, using unitary matrices only, is not new; such interest was evidenced in the mid-fifties, when the Jacobi method was still widely used for symmetric (and Hermitian) matrices.

Greenstadt [3] proposed in 1955, (as in the classical Jacobi method) to choose for the pivot pair of the rotation the indices of the maximal below diagonal element (in absolute value) and a rotation to annihilate this element. Of the two possible values for the tangent of this angle, he proposed (again, as for the classical Jacobi method) the smaller angle. He was unable to prove convergence of his proposed algorithm, and indeed, a year or so later Causey [2] published a note pointing out that a 3×3 permutation matrix would cycle. Causey proposed a modification of the Greenstadt procedure which avoids the degeneracy that occurs when diagonal elements of the matrix are equal.

The proof of convergence of the classical Jacobi method uses monotonicity of $\tau = \sum_{i < j} |a_{i,j}|^2$. In the Greenstadt-Causey algorithm for the non-symmetric case, this function is no longer monotone because rotations having pivot pairs (k, m) , where $m > k+1$, tie elements above the diagonal to those below the diagonal which are not equal in the non-symmetric case.

Greenstadt later [4], following discussions with Von Neumann [5], published reports of experiments which show that the minimization of τ at each rotation is empirically ineffective. Note that such a minimization is an entirely different procedure from that involving the annihilation of the maximal below diagonal element. Furthermore, the computation of the rotational parameters in this case involve the knowledge of global matrix information. At any rate, the process leads to a stationary state with $\tau \gg 0$. Greenstadt asserted that the process of annihilating the maximum below diagonal element was more successful.

The algorithm presented in this paper, shows $\tau' = \sum_{i < j} |a_{i,j}|$ monotone over sweeps, suggesting a possible approach for a convergence proof.

Wolf [10] reported the use of only the sub-diagonal pivots to minimize τ , and showed convergence to a matrix having a zero subdiagonal. He also showed experiments indicating the importance of the choice of pivot pattern. Huang [6] describes an inductive proof of a different algorithm based on the decomposition of a 2×2 matrix. His approach is also a global one.

More recently, Stewart [9] has proposed an interesting method using only sub-diagonal pivots, with the additional condition that the smaller angle never be chosen; he claims that the consistent choice of the larger angle in conjunction with a special pivot ordering will circulate all the below-diagonal elements for potential annihilation. Stewart points out that his method is stationary for matrices having equal diagonal elements and zero first and second sub-diagonals.

3 Local Properties

For any algorithm designed for computation on an $N \times N$ array of processors, (not sharing common memory) we must assume that only local information is available to any processor. For example, in the Brent-Luk algorithm for symmetric matrices ($N=n/2$), or in the Stewart algorithm ($N=n$), a given processor "knows" only the content of the 2×2 matrix presently in its store.

Global properties, such as normality or orthogonality, must be considered unavailable so as not to destroy potential parallelism. It is this restriction to local information which leads us to abandon the norm-reducing algorithms [7], [8], for this purpose at this time. The computation of elements of the commutator, $AA^* - A^*A$, requires non-local information.

A diagonal processor must have some well-defined task to perform when it encounters the 2×2 matrix

$$\begin{pmatrix} a_{k,k} & a_{k,m} \\ a_{m,k} & a_{m,m} \end{pmatrix}.$$

For example, the Greenstadt-Causey procedure would make $a'_{m,k}=0$, using the smaller angle.

The Stewart algorithm would first permute the matrix, and then use the Greenstadt parameters. An important question to ask is: what will a processor do when the matrix

$$\begin{pmatrix} a_{k,k} & a_{k,m} \\ 0 & a_{m,m} \end{pmatrix}$$

is encountered. In the Stewart procedure, a permutation would be used to put the matrix in the form

$$\begin{pmatrix} a_{m,m} & 0 \\ a_{k,m} & a_{k,k} \end{pmatrix},$$

followed by the smaller rotation which sets the new element in the m-k position to zero.

If, in the second case, $a_{k,k} = a_{m,m}$, i.e.,

$$\begin{pmatrix} a & x \\ 0 & a \end{pmatrix}$$

then the permutation and ensuing rotation ($\pm\pi/2$) result in a matrix identical to the original but for signs. This situation causes the algorithm to cycle (unless rounding errors are introduced) for some permutation matrices, despite the fact that these matrices are normal and have distinct eigenvalues. For example, $P_n = (p_{i,j})$, where all $p_{i,j} = 0$ except $p_{n,1} = 1$, and $p_{i,i+1} = 1$, $i = 1 \dots n-1$.

In this paper, we describe in detail the parameters involved in rotations for the non-symmetric case. We also propose an alternative algorithm, with several variations, a compromise, which sets a clearly defined task and goal for each processor, including those cases where the 2×2 matrix is badly behaved despite the fact that the entire matrix defines its eigenvalues and eigenspaces well.

4. On rotation parameters

We consider in detail the choice of parameters for a 2×2 transformation. We have:

$$\begin{pmatrix} \cos x & e^{i\theta} \sin x \\ -e^{-i\theta} \sin x & \cos x \end{pmatrix} \begin{pmatrix} a_{k,k} & a_{k,m} \\ a_{m,k} & a_{m,m} \end{pmatrix} \begin{pmatrix} \cos x & -e^{i\theta} \sin x \\ e^{-i\theta} \sin x & \cos x \end{pmatrix} = \begin{pmatrix} a'_{k,k} & a'_{k,m} \\ a'_{m,k} & a'_{m,m} \end{pmatrix}$$

The transformed elements of interest to us are:

$$(4.1) \quad \begin{aligned} a'_{m,k} &= c^2 e^{-i\theta} [-t(a_{k,k} - a_{m,m}) + e^{i\theta} a_{m,k} - t^2 e^{-i\theta} a_{k,m}], \\ a'_{k,m} &= c^2 e^{i\theta} [-t(a_{k,k} - a_{m,m}) + e^{-i\theta} a_{k,m} - t^2 e^{i\theta} a_{m,k}] \end{aligned}$$

where $t = \tan x$ and $c = \cos x$.

The roots to the quadratic equation which is formed by setting the first quantity in square brackets equal to zero are given by

$$t = \frac{[-d \pm \sqrt{d^2 + a_{k,m} a_{m,k}}]}{e^{-i\theta} a_{k,m}} = \frac{e^{i\theta} a_{m,k}}{[d \pm \sqrt{d^2 + a_{k,m} a_{m,k}}]}$$

where $d = (a_{k,k} - a_{m,m})/2$.

Without loss of generality, we may choose θ so that $t = \tan x$ is real and non-negative. To avoid cancellation, we choose the sign so that $[d \pm \sqrt{d^2 + a_{k,m} a_{m,k}}]$ assumes its largest absolute value.

Calling this value d_{Max} , we may express the two roots of the quadratic equation as shown:

$$(4.2) \quad t_S = e^{i\theta} a_{m,k} / d_{\max} \quad \text{and} \quad t_L = -d_{\max} / e^{-i\theta} a_{k,m}.$$

The subscripts S and L denote the smaller and the larger roots in absolute value. We know these expressions give the smaller and larger "roots" because $|d_{\max}|^2 \geq |a_{k,m} a_{m,k}|$, as we now show.

We have defined $d_{\max}$ to be that of the two quantities

$$|d \pm \sqrt{d^2 + a_{k,m} a_{m,k}}|,$$

which is the largest in absolute value. Using the parallelogram law, we obtain

$$\begin{aligned} 2 |d_{\max}|^2 &\geq |d + \sqrt{d^2 + a_{k,m} a_{m,k}}|^2 + |d - \sqrt{d^2 + a_{k,m} a_{m,k}}|^2 \\ &= 2 (|d|^2 + |\sqrt{d^2 + 4 a_{k,m} a_{m,k}}|^2). \end{aligned}$$

Thus,

$$\begin{aligned} |d_{\max}|^2 &\geq |d|^2 + |d^2 + a_{k,m} a_{m,k}| \\ &\geq |d|^2 + ||d|^2 - |a_{k,m} a_{m,k}||. \end{aligned}$$

Now, if $|d|^2 \leq |a_{k,m} a_{m,k}|$, we have immediately that $|d_{\max}|^2 \geq |a_{k,m} a_{m,k}|$. On the other hand, if $|d|^2 > |a_{k,m} a_{m,k}|$, then

$$|d_{\max}|^2 \geq 2|d|^2 - |a_{k,m} a_{m,k}| > |d|^2 > |a_{k,m} a_{m,k}|.$$

We conclude, in any case, that $|d_{\max}|^2 \geq |a_{k,m} a_{m,k}|$. We now know that

$$|t_S| = |a_{m,k}/d_{\max}| \leq |d_{\max}/a_{k,m}| = |t_L|.$$

We discuss how t_S and t_L are related to the inner and outer rotations used by Stewart [9]. In our notation $\cos x = c$ and $e^{-i\theta} \sin x = s$. We do not use a phase angle with $\cos x$, because it enters only as a multiplicative factor in every expression. Stewart defines an inner rotation as the one which is closest to the identity. Hence, up to the lack of the phase t_S corresponds to the inner rotation. Stewart also defines an outer rotation as either i.) 'the other', or ii.) the inner rotation obtained for PAP, where P is a permutation matrix:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

As Stewart points out this amounts to obtaining an inner (smaller) rotation for the matrix

$$PAP = \begin{pmatrix} a_{m,m} & a_{m,k} \\ a_{k,m} & a_{k,k} \end{pmatrix}.$$

Although P is not a rotation, we may consider the larger rotation obtained from t_L , as equivalent to an outer rotation.

We require the eigenvalues of the 2×2 transformation in our notation. We first note that $A(c, e^{-i\theta_S})^T = \lambda(c, e^{-i\theta_S})^T$ is equivalent to the system:

$$\begin{aligned}
 (4.3) \quad & a_{k,k} + a_{k,m} t_1 = \lambda \\
 & a_{m,k} + a_{m,m} t_1 = \lambda t_1.
 \end{aligned}$$

where $t_1 = e^{-i\theta} \sin x / \cos x$. Substitution of t_S into the second of the equations (4.3) yields $\lambda_1 = a_{m,m} + d\text{Max}$, provided $a_{m,k} \neq 0$, and substitution of t_L into the first yields $\lambda_2 = a_{k,k} - d\text{Max}$, provided $a_{k,m} \neq 0$. (We remark that we never attempt a rotation if $a_{m,k} = 0$, and that we cannot use t_L when $a_{k,m} = 0$.) Thus, we have the transformed matrices resulting from the parameters t_S and t_L :

$$\begin{array}{cc}
 \begin{pmatrix} \lambda_1 & a_{k,m}' \\ 0 & \lambda_2 \end{pmatrix} & \begin{pmatrix} \lambda_2 & a_{k,m}' \\ 0 & \lambda_1 \end{pmatrix} \\
 t_S & t_L
 \end{array}$$

In either case, from (4.1) we obtain $a_{k,m}' = e^{i\theta} [e^{-i\theta} a_{k,m} - e^{i\theta} a_{m,k}]$. This result is obtained by noting that, if $\cos x \neq 0$, then $-t(a_{k,k} - a_{m,m}) = e^{i\theta} a_{m,k} - t^2 e^{-i\theta} a_{k,m}$. Substitution of this expression into that for $a_{k,m}'$, and using $c^2(1 + t^2) = 1$, yields the above. In the case that the matrix is Hermitian, $d\text{Max}$ is real, θ is chosen so that $a_{m,k}$ is real, and $a_{k,m}'$ is 0 when $a_{m,k}$ is 0.

5 On the angle size

It is important to note that both t_S and t_L may be ≤ 1 , or similarly, both may be ≥ 1 . For example, let $\epsilon > 0$, and consider either

$$\begin{pmatrix} 3\epsilon & 1/4 - \epsilon \\ 1 + 8\epsilon & -\epsilon \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 3\epsilon & 1 + 8\epsilon \\ 1/4 - \epsilon & -3\epsilon \end{pmatrix}$$

For both matrices, $d_{\max} = \epsilon + 1/2$. In the first case, both $|t_S|$ and $|t_L|$ are ≥ 1 , and in the second, both ≤ 1 , for $\epsilon \geq 0$. Thus, one may not count on near permutations of rows and columns occurring simply because the larger angle is chosen, nor can one count on a rotation being 'small' simply because it is the smaller of the two which will annihilate $a'_{m,k}$.

We remark in passing that the only rotation which will annihilate $a'_{m,k}$ for the matrix

$$\begin{pmatrix} a & 0 \\ 1 & a \end{pmatrix}$$

is one having an angle of $\pm \pi/2$.

6 Choice of angle

In this section, it must be remembered that the rotation is an $n \times n$ rotation. We assume that the pivot pair (k,m) is given and is known to the appropriate diagonal processors. This

processor determines the rotational parameters, and sends them to the off-diagonal processors which compute the new matrix elements:

$$\begin{aligned}
 (6.1) \quad a_{k,i}' &= c a_{k,i} + e^{i\theta} s a_{m,i} & a_{i,k}' &= c a_{i,k} + e^{-i\theta} a_{i,m} \\
 a_{m,i}' &= -e^{-i\theta} s a_{m,i} + c a_{m,i} & a_{i,m}' &= -e^{i\theta} s a_{i,k} + c a_{i,m}
 \end{aligned}$$

where $i = 1 \dots n, i \neq k, m$.

From (6.1) we see that pivots on the subdiagonal, $k = m-1$, behave differently from those below the subdiagonal, $k < m-1$, as is illustrated in the following diagram.

We first discuss the conditions under which we will use t_S and t_L . We will do nothing if $|a_{m,k}| \leq \text{eps}$, a machine dependent constant. (Throughout we assume that the original matrix has been scaled to norm one.)

For pivots below the sub-diagonal, where $k < m-1$, we wish to use 'small' rotations, so as not to introduce too much influence from the elements above the diagonal. Realizing that t_S can be larger than 1, we check for this condition and set $\tan 2x = t_S$ and find $\tan x$ accordingly. When $k = m-1$, we may use t_L .

We have found experimentally that having a relatively large diagonal element between smaller elements can cause cycling. Roughly, this situation may be seen by considering matrices of the following type:

$$\begin{pmatrix} \epsilon & X & X & X \\ X & \epsilon & X & X \\ X & X & X & X \\ X & X & X & \epsilon \end{pmatrix}$$

where we consider ϵ to represent small, but not necessarily equal elements, and X to be a relatively large element. Rotations in the (1,3) and (2,4) planes will tend to be small since $d_{\max}$ will be 'large'. On the other hand, a rotation in the (1,4) plane can be large, since $d_{\max}$ will be 'small'. A large rotation will produce a large influence from the 1st row and 4th column in the elements below the diagonal. If the diagonal can be reordered, so that the large element, X , appears in the (1,1) position, then the rotations in the (1,3) and (1,4) planes will all be appropriately 'small'. Of course, the rotation in the (2,4) plane can still be too large. This situation appears to make convergence slow when all the eigenvalues are nearly equal, and matrix has nearly parallel eigenvectors.

We use rotations with the larger angle t_L to force an ordering of the diagonal elements with respect to size, to avoid this pitfall. On the other hand, in the case of nearly equal diagonal elements, too frequent interchanges occurring from the use of t_L will cause cycling. We discuss the question as to what orderings may be imposed upon the diagonal.

We have tried ordering the diagonal elements by absolute value, which was successful provided that the use of t_L was restricted to the first few (3-4) sweeps. Since we have used mostly real matrices, we have decided upon an ordering by real parts. Perhaps additional conditions upon the imaginary parts should be added, but we have not done this here.

Let x and y represent the real parts of two adjacent diagonal elements. We allow an interchange (i.e. t_L) when $x > \alpha_1 y$ and $|x - y| > \alpha_2 n$. In one implementation, we use $\alpha_1 = 2.5$, and $\alpha_2 = 0.05$. In another implementation, we have successfully used the following heuristic: relax the above conditions by setting $\alpha_1 = 1$, and $\alpha_2 = .001$, but allow the larger rotations to take place only once every 1 sweeps (when sweep number mod 1 = 2).

We can now describe the algorithm used as follows:

If $|a_{m,k}| \leq \text{eps}$, a machine dependent constant, the rotation is skipped

else

1. If $k=m-1$, and $\text{real}(a_{k,k} - d_{\text{Max}}) > \alpha_1 \text{real}(a_{m,m} + d_{\text{Max}})$, and $|\text{real}(a_{k,k} - d_{\text{Max}}) - \text{real}(a_{m,m} + d_{\text{Max}})| > \alpha_2 n$, and the sweep number is appropriate then t_L is chosen to order the diagonal elements. Otherwise, t_S is chosen, subject to 3 below.

2. If $k \neq m-1$, then t_S is chosen, again, subject to 3 below.
3. If in either 1 or 2 above we find that t_S is $\geq \alpha_3$, where $\alpha_3 < 1$, we set $\tan 2x = t_S$ and find $\tan x$ accordingly. Note that θ is left unchanged.

Several boundary questions arise which must be answered. For example, what should be done when $d_{\max} = 0$, that is when we have

$$\begin{pmatrix} a_{k,k} & 0 \\ a_{m,k} & a_{k,k} \end{pmatrix}$$

Both t_S and t_L are undefined. This situation occurs when $(k,m) = (1,3)$, for example, in the matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

or when $(k,m) = (1,2)$ or $(2,3)$ in the matrix

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

In the first case no single rotation will cause a decrease in $\tau = \sum_{i>j} |a_{i,j}|^2$. In the second case, a rotation using $x = \pm \pi/2$, results in a matrix with the same 0-1 pattern as the first but for signs. In the latter case, we arbitrarily choose $x = \pi/4$, which equalizes locally the 'mass' above and below the diagonal. In the former case, we set $t_S = 1$, and let rule 3 occur.

If $d \neq 0$ and t_L is called for, but $a_{k,m} = 0$ so that t_L is undefined, ($dMax \neq 0$), we use t_S . (Letting $x = \pi/2$ is a temptation for $k=m-1$, but can lead to cycling.) Thus, we have

4. If t_S is undefined, that is, $dMax = 0$ then set $t = 1$ and $\theta = 0$. If $dMax \neq 0$, $a_{k,m} = 0$, and t_L is called for, we use t_S which is defined.

Because the pivot pair (k,m) determines the choice of θ and x , we require that index information be sent to, or computed by, all processors when the four matrix elements are transmitted. Note that when the matrix is not Hermitian, $a_{m,k}$ must be distinguished from $a_{k,m}$ as must the k -th and m -th rows and columns.

7 The Array of Processors

We assume the same architecture as do Brent and Luk [1], that is a square array of N^2 ($N=n/2$) processors, $P_{i,j}$, each holding 4 (complex) matrix elements. Diagonal, horizontal, and vertical communication connections identical to those used by Brent and Luk are also assumed. If n is odd, a row and column of zeros is appended, so that n may be assumed to be even. We illustrate, using the diagram from the Brent-Luk paper [1], for $n=8$.

Brent and Luk use a permutation pairing which is initially

$$(1,2), (3,4), (5,6), \dots \quad (n-1,n).$$

At each time interval matrix elements migrate to neighboring cells. The pairings return to the original configuration after $n-1$ time intervals. The permutations reached from one time interval to the next may be described by the mapping:

$$(7.1) \quad 1 \rightarrow 1, 2 \rightarrow 4, 4 \rightarrow 6, \dots, n \rightarrow n-1, n-1 \rightarrow n-3, \dots, 7 \rightarrow 5, 5 \rightarrow 3, 3 \rightarrow 2.$$

We have implementations using this ordering, but have also experimented with others.

Any other initial pairing may be used, provided that the matrix elements migrate according to (7.1). In order that the element 1 be paired with elements $n, n-1, n-2, \dots, 3, 2$, in that order, we may assume the initial pairing:

$$(1, n), (2, n-1), (3, n-2) \dots (n/2, n/2 + 1).$$

The mapping defined by (7.1) now becomes:

$$(7.2) \quad 1 \rightarrow 1, 2 \rightarrow n, n \rightarrow n-1, n-1 \rightarrow n-2 \dots 6 \rightarrow 5, 5 \rightarrow 4, 4 \rightarrow 3, 3 \rightarrow 2.$$

We illustrate the permutations which take place for $n=8$, assuming the Brent-Luk migration of elements, but the above initial ordering:

$$\begin{array}{cccc} (1,8) & (2,7) & (3,6) & (4,5) \\ (1,7) & (8,6) & (2,5) & (3,4) \\ (1,6) & (7,5) & (8,4) & (2,3) \\ (1,5) & (6,4) & (7,3) & (8,2) \\ (1,4) & (5,3) & (6,2) & (7,8) \\ (1,3) & (4,2) & (5,8) & (6,7) \\ (1,2) & (3,8) & (4,7) & (5,6) \end{array}$$

Such an initialization requires that the input matrix be permuted before the algorithm begins and be permuted back again after the decomposition is complete. For example, a 4x4 matrix would originally be permuted to

$$\begin{pmatrix} a_{1,1} & a_{1,4} & a_{1,2} & a_{1,3} \\ a_{4,1} & a_{4,4} & a_{4,2} & a_{4,3} \\ a_{2,1} & a_{2,4} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,4} & a_{3,2} & a_{3,3} \end{pmatrix}.$$

After the decomposition, but before permutation back again, the matrix will appear (if convergence occurs) as

$$\begin{pmatrix} a_{1,1} & a_{1,4} & a_{1,2} & a_{1,3} \\ 0 & a_{4,4} & 0 & 0 \\ 0 & a_{2,4} & a_{2,2} & a_{2,3} \\ 0 & a_{3,4} & 0 & a_{3,3} \end{pmatrix}.$$

We mention again an important difference between the symmetric and non-symmetric cases. When the matrix to be decomposed is not symmetric, it is imperative that each processor differentiate between pivot indices k and m . Hence, as matrix elements pass from one processor to the next, the indices of these elements must either be carried along or the pivot elements calculated so that all processors are able to distinguish between $a_{m,k}$ and $a_{k,m}$, as well as between $a_{k,k}$ and $a_{m,m}$.

8 Examples

We implemented a similar algorithm some years ago in Fortran, while in Oxford,

(reported at the Gatlinberg conference in 1971), but never published it because we had no proof of convergence. Interest at that time in parallel computation was not strong. More recently, the program was rewritten in Pascal, and run on a Vax 11/780 as well as on a MacIntosh. Versions in C and Fortran are currently being implemented. A number of differences exist between the earlier and the later implementations. First, in the original, the pivots were chosen in an ordering used for Givens rotations in a QR (or RQ) decomposition. Also, the ordering of the diagonal was by absolute value rather than real part as described in S 6.

In the current implementations, the pivots are chosen to simulate an $N \times N$ array of processors as indicated in S 7. The use of the larger angles for ordering of the real parts is described in S 6. The algorithm works in a satisfactory manner although, for matrices far from normality, not as well as Jacobi. As the examples show, the speed of convergence is about that of the Jacobi method for normal matrices. Convergence is considerably slower for matrices which have a large distance from normality. This is not surprising, since other methods such as QR or the norm-reducing methods also exhibit similar behavior for such matrices.

We illustrate the current implementation with a series of matrices which depend on n . But first, we illustrate its performance on the 5 matrices of order five used by Stewart [9]. These matrices have the form $A = U(D + \alpha F)U^H$, where U is unitary (to 4 decimal places), $D = \text{diag}(1,2,3,4,5)$, and F is strictly upper triangular. The parameter α takes on values .01, .1, 1., and 10. A fifth example uses $D = \text{diag}(1,2,3,3,4)$ and $\alpha = .1$. We summarize the convergence of our implementation for these matrices of increasing non-normality.

matrix α	.01	.1	1	10	.1 (2cd Diagonal
Sweep	tau	tau	tau	tau	tau
0	4.9e+00	4.8e+00	5.7e+00	3.4e+01	3.8e+00
1	2.7e+00	2.6e+00	3.3e+00	1.9e+01	9.0e-01
2	8.5e-02	1.5e-01	7.6e-01	7.6e+00	1.7e-01
3	1.0e-04	8.6e-03	6.5e-02	5.0e+00	1.2e-03
4	2.6e-08	8.0e-05	5.6e-04	3.1e+00	6.2e-05
5	4.1e-13	1.2e-08	3.6e-06	1.7e+00	4.6e-08
6	---	9.9e-14	3.9e-09	8.8e-01	1.0e-12
7	---	---	4.2e-14	3.7e-01	...
8	---	---	---	1.7e-01	
9	---	---	---	6.3e-02	
10	---	---	---	8.6e-03	
11	---	---	---	2.0e-04	
12	---	---	---	2.5e-07	
13	---	---	---	2.8e-12	

Among defective matrices, is one with two equal pairs of complex conjugate eigenvalues, originally introduced by Semendaev. Its convergence behavior, similar to the Stewart matrix $\alpha = 10$, is shown below.

9.2e+00 sweep * i	1	The eigenvalues obtained are: 1.49999997752345e+00-3.57071417095666e+00 1.50000002247655e+00-3.57071425758619e+00 1.49999998847807e+00 3.57071416622975e+00 1.50000001152193e+00 3.57071426231310e+00 1.00000000000000e+00-9.29676259302225e-16
6.8e+00 sweep * i	2	
2.7e+00 sweep * is	3	
1.4e+00 sweep * is	4	
1.6e+00 sweep * is	5	
2.2e-01 sweep * is	6	
2.7e-02 sweep * is	7	
1.5e-03 sweep * is	8	The loss of accuracy after the 8th decimal is marked and typical for this kind of defective matrix.
1.8e-05 sweep * is	9	
1.6e-08 sweep * is	10	
2.4e-12 sweep * is	11	

We compare the convergence for matrices known as B_n or the Frank, matrices for $n = 3, 4, \dots, 9$. These matrices are known to have poorly conditioned smaller eigenvalues; as n increases, the rate of convergence is seen to decrease. For these small n , no loss of

accuracy was noted.

n	3	4	5	6	7	8	9
sweep\tau							
1	1.1+00	3.3+00	5.0+00	8.5+00	1.9+01	2.6+01	3.5+01
2	9.4-02	6.1-01	2.9+00	6.8+00	1.1+01	1.2+01	2.6+01
3	1.1-03	1.2-01	9.1-01	1.5+00	4.3+00	4.5+00	1.7+01
4	8.2-08	5.0-03	2.3-01	7.1-01	1.4+00	1.5+00	8.9+01
5	0.0-16	1.1-06	3.0-02	2.1-01	5.8-01	4.3-01	4.4+01
6	---	6.2-11	1.0-03	2.2-01	1.9-01	1.8-01	1.7+01
7	---	3.7-21	5.3-06	2.8-02	6.7-02	7.4-02	7.5-01
8	---	---	1.0-13	9.5-03	8.1-03	1.5-02	4.8-01
9	---	---	---	1.0-04	2.9-04	3.2-02	3.3-01
10	---	---	---	1.6-06	3.5-06	5.7-04	1.4-01
11	---	---	---	3.5-10	8.4-09	7.7-05	9.1-02
12	---	---	---	4.6-15	4.9-13	1.5-05	5.4-02
13	---	---	---	---	---	2.2-07	2.2-02
14	---	---	---	---	---	4.4-09	1.1-03
15	---	---	---	---	---	1.0-14	1.3-04
....	...	...	...	...	...	...	
20	---	---	---	---	---	---	1.6-14

We have also run P_n , the permutation matrices of order n defined below, P_n^1 and P_n^{10} also defined below, the Forsythe matrices, F_n and several sets of random matrices.

The permutation matrices illustrated here are defined by $P_n = (p_{i,j})$, where $p_{i,j} = 0$ except $p_{i,i+1} = 1$, for $i=1 \dots n-1$, and $p_{n,1} = 1$. The rate of convergence remains almost constant for increasing values of n . We give the "effective" number of sweeps, which is the number of rotations divided by $n(n-1)/2$. We illustrate for $n=5, \dots, 12$

sweep\ n	5	6	7	8	9	10	11	12
0	1.0e1	1.0e1	1.0e1	1.0e1	1.0e1	1.0e1	1.0e1	1.0e1
1	3.29e1	4.47e1	5.32e1	6.29e1	7.26e1	8.01e1	9.21e1	9.97e1
2	1.85e1	2.31e1	3.86e1	5.95e1	7.51e1	5.17e1	8.31e1	9.86e1
3	7.6e-2	4.77e-1	1.52e1	2.49e1	4.12e1	1.34e1	4.25e1	4.57e1
4	2.9e-5	2.5e-3	5.7e-2	1.7e-1	7.8e-1	3.8e-2	4.7e-1	6.2e-1
5	7.9e-16	7.1e-9	7.7e-6	5.1e-4	5.4e-3	7.8e-6	2.3e-3	3.5e-3
6	...	2.0e-16	2.0e-14	5.2e-10	6.8e-8	1.4e-13	3.7e-8	5.6e-8
7	...	...	4.7e-16	4.7e-16	1.1e-15	...	4.2e-16	2.8e-16
eff. *swps	4.5	5.27	5.52	5.96	6.17	5.51	6.04	6.21

We summarize the number of sweeps for higher values of

n	13	14	15	16	17	18	19	20
eff. *swps	6.41	6.63	6.39	6.97	6.79	6.92	7.26	7.02

For $n = 25, 30$, and 40 , the effective number of sweeps was $7.14, 7.37$, and 8.14 . We believe that the slow increase in the effective number of sweeps is caused by rotations which are too large taking place during the first few sweeps, bringing in to the below diagonal the effects of the above diagonal elements. Further research may eliminate this effect. The above results are for unnormalized matrices.

Similar experiments were performed for matrices identical to the above permutation matrices except that $p_{n,1}$ was set to 0.1 and to 10.0 . These matrices were normalized. The results we give here are for the Brent-Luk ordering. The chart lists the effective number of sweeps to convergence.

n	5	8	11	14	17	21	24	25
a(n,1)=.1	8.4	12.7	13.3	16.2	18.9	18.0	20.4	18.1
a(n,1)=10	8.2	11.3	14.9	15.9	20.7	21.7	22.2	22.2

We ran a set of random matrices using the Brent-Luk ordering and also the second ordering described in S 7. We summarize these results.

Brent-Luk ordering			Second ordering		
n	*sweeps	eff.*sweeps	*sweeps	*rotations	eff.*sweeps
3	6	5.33	6	16	5.33
4	6	6	6	31	5.17
5	9	8	8	71	7.10
6	12	11.07	8	113	7.53
7	11	9.38	9	172	8.19
8	12	10.96	10	258	9.21
9	14	11.92	13	428	11.89
10	19	17.62	17	692	15.38
11	19	15.67	16	818	14.87
12	19	16.95	16	932	14.12
13	22	18.94	16	1133	14.53
14	18	15.81	18	1478	16.24
15	24	21.04	18	1764	16.8
16	21	18.23	18	1889	15.74
17	25	22.39	24	2949	21.68
18	24	21.13	20	27.81	18.18
19	19	17.26	20	3087	18.05
20	26	22.11	23	3776	19.87

We have no interpretation of these uneven results; we do believe further efforts should be made to investigate the choices of pivot ordering and the choices of the constants α_1 , α_2 , and α_3 . The number of sweeps in these examples is unacceptable for other than small n.

The Forsythe matrices, $F_n = (f_{i,j})$ are defined by $f_{i,j} = 0$, except for $f_{i,i+1} = 1$, $i=1..n-1$, and $f_{n,1} = 10^{-n}$. These matrices are the ultimate perturbation of the above-mentioned permutation matrices. The eigenvalues of the Forsythe matrices are the n -th roots of unity multiplied by one tenth. As the examples show, the rate of convergence is most unsatisfactory, even for very small n . However, this behavior may be considered as a function of the matrix rather than the method. Note that $f_{n,1}$ is the only element below the diagonal which is not equal to zero. We show these examples through $n=7$.

n	3	4	5	6	7
sweep\tau					
1	9.9-01	1.9+00	2.8+00	3.8+00	5.5+00
2	1.5-01	1.0+00	1.4+00	1.1+00	3.0+00
3	9.3-02	5.3-01	6.1-01	4.6-01	...
4	3.1-02	2.7-01	...	3.2-01	...
5	1.0-02	2.0-01	...	...	5.9-01
6	2.6-03	9.9-02	...	...	...
7	2.4-03	4.8-02	7.4-02	...	...
8	1.9-06	2.5-02	...	8.2-02	...
9	2.7-10	1.4-02	...	...	...
10	9.4-20	1.4-02	...	...	...
11	...	7.0-03	...	...	...
12	...	1.7-03	8.0-03	...	8.4-02
13	...	1.0-04	...	...	...
14	...	4.6-07	...	...	...
15	...	9.0-12	1.9-04	...	...
16	...	...	7.6-06	...	...
17	...	...	3.7-08	...	...
18	...	...	7.0-12	5.9-03	...
...	...	...	...	...	...
21	...	...	...	3.4-04	8.7-03
22	...	...	...	1.7-06	5.2-03
23	...	...	...	5.5-08	2.4-03
24	...	...	...	5.0-12	7.3-04
25	...	...	...	...	1.0-04
26	...	...	...	...	6.2-06
27	...	...	...	...	1.2-08
28	...	...	...	...	3.6-12

The computed eigenvalues for $n = 7$ are illustrative of the accuracy obtained.

```
i = 1  0.1000000000005 -0.0000000000004
i = 2  0.062348980194  0.078183148250
i = 3  0.062348980187 -0.078183148250
i = 4 -0.022252093396 -0.097492791223
i = 5 -0.022252093397  0.097492791227
i = 6 -0.090096886795 -0.043388373915
i = 7 -0.090096886797  0.043388373914
```

The case for $n=8$ took 38 sweeps, stopping when the maximum below diagonal element was $3e-12$.

9 Comments

The Jacobi algorithm we present here is suggested as an alternative possibility for parallel computation on a square grid of mesh-connected processors. It is not recommended for serial computation.

While we are encouraged that the algorithm works as it does, we believe we have left many questions unanswered. We suggest that the distance of a pivot pair from the diagonal should control the size of the rotation allowed, i.e., the program presented here prevents rotations larger than $\pi/4$ if the pivot pair is at a distance of two or more from the diagonal. Perhaps the prevention of larger rotations for pivots far from the diagonal would speed the slow convergence displayed in early sweeps for badly non-normal matrices.

One suspects that the sorting of the diagonal elements may cause slow convergence. It is tempting to adapt the ideas presented here to the architecture introduced by Stewart, because in that setting the sorting would take at most time $O(n)$. Certainly more experimentation would be useful. The choice of α_1 , α_2 , and α_3 has been ad hoc.

However, the fact that the program does converge for very badly conditioned matrices is encouraging, as is the quadratic convergence exhibited, and the nearly constant number of sweeps for normal matrices.

We gratefully acknowledge the competent programming and editorial help of Kris Eberlein, as well as the financial assistance of the Visiting Professorships for Women program of the National Science Foundation.

Ithaca, N.Y., April 1985

P. J. Eberlein
Visiting Professor, Cornell University
on leave from The University at Buffalo/SUNY

References

- [1.] R. Brent and Franklin Luk, The Solution of Singular-Value and symmetric Eigenvalue Problems on Multiprocessor arrays, SIAM J. Sci. Stat Comput, v. 6, #1, 1985, pp. 69-84.
- [2.] R. Causey, Computing Eigenvalues of non-Hermitian Matrices by the Methods of Jacobi, SIAM J. App. Math., v. 6, 1958, pp. 172-181.
- [3.] John Greenstadt, A Method for Finding Roots of Arbitrary Matrices, MTAC, V.. 9, #50, 1955, pp. 47-52.
- [4.] John Greenstadt, Some Numerical Experiments in Triangularizing Matrices, Num. Math v.4, 1962, pp.187-195.
- [5.] H. H. Goldstein, A Footnote to a Recent Paper, JACM, v. 7, pp.78-79.
- [6.] C. P. Huang, A Jacobi-Type Method for triangularizing an Arbitrary Matrix, SIAM J. Num. Anal. v. 12, 1975, pp. 566-670.
- [7.] P. J. Eberlein, Solution to the Complex Eigenproblem by a Norm-reducing Jacobi-type Method, Num. Math., v. 14, 1970, pp. 232-245.
- [8.] A. Sameh, On Jacobi and Jacobi-type Algorithms for a Parallel Computer, Math. Comp. v. 25, 1971, pp. 579-590.
- [9.] G. W. Stewart, A Jacobi-like Algorithm for Computing the Schur Decomposition of a Non-Hermitian Matrix, Technical Report 1321, Dept. Comp. Sci., U of Md., 1983.