NEW PARALLEL ALGORITHMS FOR STRUCTURAL ANALYSIS AND DESIGN OF AEROSPACE STRUCTURES

By
Dr. Duc T. Nguyen, Principal Investigator
Department of Civil & Environmental Engineering

FINAL REPORT
For the period ending June 30, 1998

Prepared for
NASA Langley Research Center
Attn.: R. Todd Lacks
Grant Officer
Mail Stop 126
Hampton, VA 23681-0001

And

NASA Langley Research Center
Attn.: Olof O. Storaasli
Technical Officer
Mail Stop 240
Hampton, VA 23681-0001

Under
NASA Grant NAG 1-858
ODURF Project No. 184047

Submitted by
Old Dominion University Research Foundation
800 West 46th Street
Norfolk, VA 23508

October 1998
ABSTRACT

Subspace and Lanczos iterations have been developed, well documented, and widely accepted as efficient methods for obtaining p-lowest eigen-pair solutions of large-scale, practical engineering problems. The focus of this paper is to incorporate recent developments in vectorized sparse technologies in conjunctions with Subspace and Lanczos iterative algorithms for computational enhancements. Numerical performance, in terms of accuracy and efficiency of the proposed sparse strategies for Subspace and Lanczos algorithms, is demonstrated by solving for the lowest frequencies and mode shapes of structural problems on the IBM-R6000/590 and Sun-Sparc 20 workstations.

1. INTRODUCTION

The finite element method has been used successfully for the solution of many practical engineering problems in various disciplines, such as structural analysis, fluid mechanics, structural optimization, heat transfer etc. [1-5]. Essential to the finite element solution of these problems is an effective numerical procedure for solving large-scale, sparse systems of linear equations and generalized eigen-equations. These solution phases typically represent the most costly step of the analysis in terms of computational resources.

Subspace and Lanczos iterations have been developed, well documented, and widely accepted as efficient methods for obtaining p-lowest eigen-pair solutions of large-scale, practical engineering problems [6-14]. The focus of this paper is, however, to re-examine these 2 popular eigen-solution algorithms, with the viewpoints to incorporate recent developments in vectorized sparse technologies in conjunctions with Subspace and Lanczos iterative algorithms for computational enhancements. Basic subspace iteration algorithm is reviewed in Section 2. Key steps in Lanczos eigen-solution algorithm is summarized in Section 3. Major computational tasks in Subspace and Lanczos iterative algorithms are identified in Section 4. Computational enhancements using vectorized, sparse strategies are discussed in Section 5. Numerical evaluations of the proposed sparse algorithms, and the developed software are demonstrated in Section 6, through practical finite element models. Finally, conclusions are drawn in Section 7.
2. BASIC SUBSPACE ITERATION ALGORITHM [1, 6-9]

The generalized eigen-equations, in matrix notation, can be expressed as

\[ [K] [\phi] = [M] [\phi] [\lambda] \]  

(1)

In Eq. (1), matrices \([K]\) and \([M]\) represent the structural stiffness and mass, respectively. Matrices \([\lambda]\) and \([\phi]\) represent the eigenvalues and eigenvectors, respectively. The dimension (or degree-of-freedom) of matrices in Eq. (1) is \(N\). For many practical engineering applications, \([K]\) is symmetrical and positive definite. Subspace iteration algorithm can be used effectively to obtain the lowest \(p\) eigen-pair solutions. The algorithm can be conveniently described by the following step-by-step procedures:

<table>
<thead>
<tr>
<th>Table 1: Step-by-step Basic Subspace Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong>: Select the starting iteration vectors ([Y_1]) (\text{where } q &lt; N)</td>
</tr>
</tbody>
</table>
| **Step 2**: Factorize the structural stiffness matrix
  \[ [K] = [L][D][L]^T \]  
  In Eq. (2), \([L]\) is the lower triangular matrix, and \([D]\) is the diagonal matrix |
| **Step 3**: For \(k = 1, 2, \ldots, \text{Maxiter}\), where Maxiter represents the input maximum number of iterations, the following tasks need to be done |
| **Step 4**: Solve \([\Phi_{k+1}]_{N\times q}\) from the following matrix equations
  \[ [K][\Phi_{k+1}]_{N\times q} = [Y_k]_{N\times q} \]  
  (3) |
| **Step 5**: Compute the reduced stiffness matrix
  \[ [K^R]_{q\times q} = [\Phi_{k+1}]_{q\times N}^T [Y_k]_{N\times q} \]  
  (4) |
| **Step 6**: Compute the reduced mass matrix
  \[ [\bar{Y}_{k+1}]_{N\times q} = [M]_{N\times N} [\Phi_{k+1}]_{N\times q} \]  
  \[ [M^R]_{q\times q} = [\Phi]^T_{q\times N} [\bar{Y}_{k+1}]_{N\times q} \]  
  (5) |
| **Step 7**: Solve the reduced eigen-equations
  \[ [K^R]_{q\times q} [Q_{k+1}]_{q\times q} = [M^R]_{q\times q} [Q_{k+1}]_{q\times q} [Q_{k+1}]_{q\times q} \]  
  (7) |
| The eigenvalues \([Q_{k+1}]\) and the associated eigenvectors \([Q_{k+1}]\) need to be arranged in the ascending orders (for example \(Q_1^2 < Q_2^2 < Q_3^2 < \ldots\)) |
| **Step 8**: Find an improved approximation to the eigenvectors
  \[ [\bar{Y}_{k+1}]_{N\times q} = [\bar{Y}_{k+1}]_{N\times q} [Q_{k+1}]_{q\times q} \]  
  (8) |
| **Step 9**: Check for convergence. The iterative process will be stopped if either convergence is achieved, or the maximum number of iteration (\(= \text{Maxiter}\)) is reached (or else, return back to step 3). |
3. LANCZOS ALGORITHM [1,3,10]

Recently, the Lanczos algorithm for the solution of generalized eigenvalue problems has been receiving a lot of attention due to its computational efficiency. The original, generalized eigenvalue equation can be written as:

$$K \phi = \omega^2 M \phi$$  \hspace{1cm} (9)

or

$$K_\sigma \phi = \omega^2_\sigma M \phi$$  \hspace{1cm} (10)

where $K$ and $M$ are structural stiffness matrix and mass matrix, respectively, $K_\sigma = K - \sigma M$, $\sigma$ is the shift value and $\omega^2_\sigma = \omega^2 - \sigma$

Instead of solving Eq. (9), or Eq. (10) directly, the Lanczos algorithm generates a tri-diagonal matrix $T_m$

$$T_m = \begin{bmatrix}
\alpha_1 & \beta_1 & & \\
\beta_1 & \alpha_2 & \beta_2 & \\
& \beta_2 & \ldots & \beta_{m-1} \\
& & \ldots & \beta_m
\end{bmatrix}$$  \hspace{1cm} (11)

through the following three-term recurrence:

$$r_j = \beta_{j-1} q_{j-1} = K_\sigma^{-1} M q_j - \alpha_j q_j - \beta_j q_{j-1}$$  \hspace{1cm} (12)

or in the matrix form:

$$[K_\sigma^{-1} M] Q_m - Q_m T_m = (0, 0, \ldots, r_m) = r_m e^T_m$$  \hspace{1cm} (13)

$$T_m z = \theta z$$  \hspace{1cm} (14)

where $e^T_m = (0, 0, \ldots, 1)$, $Q_m$ is a Nx$m$ orthogonal matrix with columns $q_j = 1, 2, 3 \ldots m$, and $m$ is usually much smaller than $N$. By solving the following reduced eigensystem the eigensolution of Eq. (10) can be obtained as

$$\omega^2_\sigma = \frac{1}{\theta}$$  \hspace{1cm} (15)

$$\phi = Q_m z$$  \hspace{1cm} (16)

For most structural engineering problems, only a few lowest frequencies and the corresponding mode shapes are required, so we have $m << N$ which leads to a significant savings in the number of operations.
A partial restoring orthogonality scheme and a convergence criterion are developed and incorporated into the basic Lanczos algorithm, which is described in a step-by-step procedure, shown in Table 2.

Table 2: Step-by-Step Basic Lanczos Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.   | Factorization: $K_n = L D L^T$  
Form starting vector: $y_0 = 0; q_0 = 0$ |
| 2.   | Compute: $M y_0$ |
| 3.   | Compute:  
$\beta_1 = \sqrt{y_0^T M y_0}; q_1 = \frac{y_0}{\beta_1}$ |
| 4.   | Compute: $P_1 = M q_1$  
Lanczos iteration  
For $j = 1, 2, 3, \ldots$, do  
5.   | $e_j = K_n^{-1} P_j$ |
| 6.   | $\delta_j = e_j - \frac{1}{\beta_j} q_j$ |
| 7.   | $\alpha_j = \delta_j^T M \delta_j = P_j^T \delta_j$ |
| 8.   | $\gamma_j = \delta_j - \alpha_j q_j$ |
| 9.   | $\Lambda_j = M \gamma_j$ |
| 10.  | $\beta_{j+1} = (\gamma_j^T M \gamma_j)^{1/2} = \sqrt{\Lambda_j^T \gamma_j}$  
Reorthogonalization of $q_{j+1}$ |
| 11.  | $q_{j+1} = \frac{\gamma_j}{\beta_{j+1}}; P_{j+1} = \frac{\Lambda_j}{\beta_{j+1}}$ |
| 12.  | IF necessary solve Eq(14): $T x = 0$  
Converged? (IF "No", then return to step 5) |
| 13.  | Eigenvector transformation: $\phi = Q x$ |

Reorthogonalization of Lanczos Vectors

Various reorthogonalization schemes have been developed to increase the efficiency of Lanczos algorithms [10-14]. However, for very large problems where factorization, forward/backward substitution and matrix-vector multiplication are the major operations, the cost of reorthogonalization becomes less important than for small problems, since only a few lowest eigenpairs are desired. In this work, a simple way of reorthogonalization is adopted.

First for any new Lanczos Vector $q_i$, calculate  
$E_i = q_i^T M q_i; (i = 1, 2, \ldots, j-1)$  
(17)

If $E_i > E$, then $q_i$ should be orthogonal to $q_j$ with respect to $M$ where $E$ is a parameter related to the machine parameter $E_0$ such that $1 + E_0 > 1$. Usually, $E$ is taken as:  
$E = \sqrt{E_0}$  
(18)

Eq. (18) is called semi-orthogonality [12] condition.
Convergence Criterion and Error Norm Check

One major advantage of the Lanczos algorithm lies in their ability to terminate the iteration process as soon as the required eigenpairs have converged. In this work, the following error bound for eigenvalues is used (after solving Eq. 14 in step 12)

\[
\text{ERROR} (i) = \left| \frac{\lambda_k - \theta_i}{\theta_i} \right| = \left| \beta_{j-1} \right| \quad \text{where} \quad i = 1, 2, \ldots, j
\]  

(19)

In Eq. (19), \( \lambda_k \) is the k-th exact eigenvalue and \( \theta_i \) is the i-th computed eigenvalue. \( Z^{(i)} \) is the j-th element of vector \( Z^{(i)} \). If \text{ERROR}(i) < RTOL, for \( i = 1, 2, \ldots, p \) (where RTOL is a user's specified tolerance, and \( p \) is the number of eigenpairs to be extracted) then the Lanczos iteration is considered to be converged and the program begins to perform the eigenvector transformation accordingly (see step 13 of Table 1).

4. MAJOR COMPUTATIONAL TASKS IN SUBSPACE ITERATIONS AND LANCZOS ALGORITHM

Careful observations on the subspace iteration, and Lanczos algorithms indicate that the following major computational tasks are required:

**Major task 1:** Matrix factorization (see step 2 of subspace iteration, and step 1 of Lanczos algorithm).

**Major task 2:** Forward and backward equation solutions (see step 4 of subspace iteration, and step 5 of Lanczos algorithm).

**Major task 3:** Matrix-Vector (or Matrix-Matrix) multiplications (see steps 5, 6 & 8 of Subspace iteration, and steps 2, 4, 7, 9, 10 & 13 of Lanczos algorithm).

Computational enhancements in conjunction with the above major tasks will be discussed with great details in the next section.

5. COMPUTATIONAL ENHANCEMENTS FOR SUBSPACE AND LANCZOS ALGORITHMS

It has been pointed out in Section 4 that matrix factorization, forward & backward equation solution, and matrix-vector (or matrix-matrix) multiplications represent the major computational lasts for Subspace iteration, and Lanczos algorithms. Recent developments in Sparse technologies [15] will be fully utilized to improve the computational efficiency of both subspace iteration, and Lanczos algorithms.

**LDL^T Algorithm**
The Choleski (or $U^TU$) factorization is efficient, however its application is limited to the case where the coefficient stiffness matrix $[K]$ is symmetry and positive definite. With negligible additional computational efforts, the LDL$^T$ algorithm can be used for broader applications (where the coefficient matrix can be either positive, or negative definite). In this algorithm, the given matrix $[K]$ in Eq. (1) can be factorized as

$$[K] = [L] [D] [L]^T$$

Where $[L]$ and $[D]$ are lower triangular matrix (with unit values on the diagonal), and diagonal matrix, respectively. For a simple 3X3 symmetrical stiffness matrix, Eq. (20) can be explicitly expressed as

$$\begin{bmatrix}
K_{11} & K_{12} & K_{13} \\
K_{21} & K_{22} & K_{23} \\
K_{31} & K_{32} & K_{33}
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
L_{21} & 1 & 0 \\
L_{31} & L_{32} & 1
\end{bmatrix}
\begin{bmatrix}
D_1 & 0 & 0 \\
0 & D_2 & 0 \\
0 & 0 & D_3
\end{bmatrix}
\begin{bmatrix}
1 & L_{21} & L_{31} \\
L_{21} & 1 & L_{32} \\
0 & 0 & 1
\end{bmatrix}$$

The unknown $L_{ij}$ and $D_i$ can be easily obtained by expressing the equalities between the upper triangular matrix (on the left-hand-side) and its corresponding terms on the right-hand-side of Eq. (21). Since the LDL$^T$ algorithm will be used later on to develop efficient, vectorized sparse algorithm, a pseudo-FORTRAN skeleton code is given in Table 3 (assuming the original given matrix $[K]$ is symmetrical and full).

| 1. C........ Assuming row 1 has been factorized earlier |
| 2. Do 11 I = 2, N |
| 3. Do 22 K = 1, I-1 |
| 4. C............Compute the multiplier (Note : $U$ represents $L^T$) |
| 5. XMULT = U(K,I) / U(K,K) |
| 6. Do 33 J = I, N |
| 7. U(I,J) = U(I,J) - XMULT * U(K,J) |
| 8. 33 CONTINUE |
| 9. U(K,I) = XMULT |
| 10. 22 CONTINUE |
| 11. 11 CONTINUE |

Table 3: Skeleton FORTRAN Code For LDL$^T$  
(assuming the matrix $U$ is completely full)

As an example, implementation of the LDL$^T$ algorithm, shown in Table 3, for a given, simple 3x3 stiffness matrix

$$[K] = \begin{bmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 1
\end{bmatrix}$$

will lead to the following factorized matrix
From Eq. (23), one can readily identify

\[
[U] = \begin{bmatrix}
2 & -\frac{1}{2} & 0 \\
\frac{3}{2} & -\frac{2}{3} & -1 \\
\frac{1}{2} & \frac{1}{3} & 0
\end{bmatrix}
\]  

(23)

\[\begin{bmatrix}
2 & 0 & 0 \\
0 & \frac{3}{2} & 0 \\
0 & 0 & \frac{1}{3}
\end{bmatrix}
\]  

(24)

\[\begin{bmatrix}
1 & -\frac{1}{2} & 0 \\
\frac{1}{2} & -\frac{2}{3} & -1
\end{bmatrix}
\]  

(25)

Storage Schemes for the Coefficient Stiffness Matrix

Successful implementation of a sparse equation solution algorithm depends rather heavily on the reordering method used. While the Reversed Cuthill-McKee (RCM), or Gipspoole-Stockmyer (GS) ... reordering algorithms can be used effectively in conjunction with skyline or variable bandwidth equation solution algorithms [16-18], these reordering algorithms are not suitable for sparse equation solution algorithm. Designing efficient sparse-reordering algorithms is a big task itself, and is outside the scope of this paper. For complete treatments on this subject, the readers are strongly recommended to popular textbooks and articles in the literature [16, 19-20]. In this section, it is assumed that the best available sparse-reordering algorithm, such as Modified Minimum Degree (MMD), or Nested Di-section (ND) [16], has already been applied to the original coefficient matrix [K]. To facilitate the discussions in this section, assuming the resulted matrix [K] (after using MMD, or ND algorithm) takes the following form
For the data shown in Eq. (26), it can be easily shown that the factorized matrix \([U]\) will have the following form:

\[
[U] = \begin{bmatrix}
    x & 0 & 0 & x & 0 \\
    x & 0 & 0 & x & 0 \\
    x & 0 & x & 0 & \\
    x & x & F & \\
    x & x & \\
\end{bmatrix}
\]  

In Eq. (27), the symbols "X" and "F" represent the nonzero values after factorization. However, the symbol "F" also refers to "Fills-in" effect, since the original value of \([K]\) at location has zero entry.

For the same data shown in Eq. (26), if "skyline" equation solution is adopted [21], then the "fills-in" effect will take the following form:

\[
[K_s] = \begin{bmatrix}
    x & 0 & 0 & x & x \\
    x & F & F & x & F \\
    x & F & F & x & F \\
    x & x & F & \\
    x & x & \\
\end{bmatrix}
\]  

On the other hand, if "variable-bandwidth" equation solution is adopted [22], then the "fills-in" effect (on the data shown in Eq. 26) will have the following form:

\[
[K_v] = \begin{bmatrix}
    x & F & F & x & x \\
    x & F & F & x & F \\
    x & F & F & x & F \\
    x & x & F & \\
    x & x & \\
\end{bmatrix}
\]

Thus, for the data shown in Eq. (26), the "sparse" equation solution is the best (in the sense of minimizing the number of arithmetic operations, and the required storage spaces in a sequential computer environment) and the "variable-bandwidth" equation solution is the worst one!
For practical computer implementation, the original stiffness matrix data, such as the one shown in Eq. (26), can be represented by the "sparse formats" as following:

\[
\begin{align*}
\text{ISTARTROW} &= \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 = N + 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} \\
\text{ICOLNUM} &= \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = NCOEF \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \\
\text{DIAG} &= \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = N \end{bmatrix} = \begin{bmatrix} 1.0 \\ 4.0 \\ 6.0 \\ 8.0 \\ 10.0 \\ 12.0 \end{bmatrix} \\
\text{AK} &= \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = NCOEF \end{bmatrix} = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \\ 5.0 \\ 7.0 \end{bmatrix}
\end{align*}
\]

The following definitions are used in Eqs. (30-33):

- \(N\): Size of the original stiffness matrix \([K]\).
- \(NCOEF\): The Number of non-zero, off-diagonal terms of the original stiffness matrix.
- \(\text{ISTATROW}\): Starting location of the first non-zero, off-diagonal term for the \(i\)th row of \([K]\). The dimension for this integer array is \(N + 1\).
- \(\text{ICOLNUM}(j)\): Column numbers associated with each non-zero, off-diagonal terms of \([K]\) (in a row-by-row fashion). The dimension for this integer array is \(NCOEF\).
- \(\text{DIAG}(i)\): Numerical values of the diagonal term of \([K]\). The dimension for this real array is \(N\).
- \(\text{AK}(j)\): Numerical values of the non-zero, off-diagonal terms of \([K]\) (in a row-by-row fashion). The dimension for this real array is \(NCOEF\).
Sparse Symbolic Factorization

The purpose of symbolic factorization is to find the locations of all nonzero (including "fills-in" terms), off-diagonal terms of the factorized matrix \([U]\) (which has NOT been done yet!). Thus, one of the major goals in this phase is to predict the required computer memory for subsequent numerical factorization. The outputs from this symbolic factorization phase will be stored in the following 2 integer arrays (assuming the stiffness matrix data shown in Eq. 26 is used):

\[
J_{\text{STARTROW}} = \begin{bmatrix}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 = N-1
\end{bmatrix} = \begin{bmatrix}
3 \\
4 \\
5 \\
7 \\
8 \\
8
\end{bmatrix}
\]

(34)

\[
J_{\text{COLNUM}} = \begin{bmatrix}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 = NCOEF2
\end{bmatrix} = \begin{bmatrix}
4 \\
6 \\
5 \\
5 \\
5 \\
6 \\
6
\end{bmatrix}
\]

(35)

The following "new" definitions are used in Eqs. (34-35):

- **NCOEF2**: The number of nonzero, off-diagonal terms of the factorized matrix \([U]\)
- **J_{\text{STARTROW}}(i)**: Starting location of the first nonzero, off-diagonal term for the \(i\)th row of the factorized matrix \([U]\). The dimension for this integer array is \(N+1\).
- **J_{\text{COLUMN}}(j)**: Column numbers associated with each nonzero, off-diagonal terms of \([U]\) (in a row-by-row fashion). The dimension for this integer array is \(NCOEF2\). Due to "fills-in" effects, \(NCOEF2 \gg NCOEF\).

The "key" steps involved during the symbolic phase will be described in the following paragraphs:

**Step 1:** Consider each \(i\)th row (of the original stiffness matrix \([K]\).)

**Step 2:** Record the locations (such as column numbers) of the original non-zero, off-diagonal terms
Step 3: Record the locations of the "fills-in" terms due to the contributions of some (not all) appropriated, previous rows (where 1 ≤ j ≤ i-1). Also consider if current ith row will have any immediate contribution to "future" row.

Step 4: Return to Step 1 for next row.

A simple, but highly inefficient way to accomplish Step 3 (of the symbolic phase) will be identifying the nonzero terms associated with the ith column. For example, there will be no "fills-in" terms on row 3 (using the data shown in Eq. 26), due to "no contributions" of the previous rows 1 and 2. This fact can be easily realized by observing that the associated 3rd column of \([K]\) (shown in Eq. 26) has no nonzero terms.

On the other hand, if one considers row 4 in the symbolic phase, then the associated 4th column will have 1 nonzero term (on row 1). Thus, only row 1 (but not rows 2 and 3) may have "fills-in" contribution to row 4. Furthermore, since \(K_{1,6}\) is nonzero (=2), it immediately implies that there will be a "fills-in" terms at location \(U_{4,6}\) of row 4.

A much more efficient way to accomplish step 3 of the symbolic phase is by creating 2 additional integer arrays, as defining in the following paragraphs:

ICHAINL(i): Chained list for the ith row. This array will be efficiently created to identify which previous rows will have contributions to current ith row. The dimension for this integer, temporary array is \(N\).

LOCUPDATE(i): Updated starting location of the ith row.

Using the data shown in Eq. (26), uses of the above 2 arrays in the symbolic phase can be described by the following step-by-step procedure:

Step 0: Initialize arrays:

\[
\text{ICHAINL} = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ N \end{pmatrix} = \{0\} \quad \text{and} \quad \text{LOCUPDATE} = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ N \end{pmatrix} = \{0\}
\]

Step 1: Consider row \(i = 1\)

Step 2: Realizing that the original nonzero terms occur in columns 4 & 6

Step 3: Since the chained list ICHAINL(\(i=1\)) = 0, no other previous rows will have any contributions to row 1

ICHAINL(4) = 1

ICHAINL(1) = 1

(36)

(37)
LOCUPDATE(i = 1) = 1

Equations (36-37) indicate that "future" row i=4 will have to refer to row 1, and row 1 will refer to itself. Eq. (38) states that the updated starting location for row 1 is 1.

**Step 1:** Consider row i=2

**Step 2:** Realizing the original nonzero term(s) only occurs in column 5

**Step 3:** Since ICHAINL (i=2) = 0 no other previous rows will have any contributions to row 2

ICHAINL(5) = 2

ICHAINL(2) = 2

LOCUPDATE(i=2) = 3

Equations (39-40) indicate that "future" row i=5 will have to refer to row 2, and row 2 will refer to itself. Eq. (41) states that the updated starting location for row 2 is 3

**Step 1:** Consider row i=3

**Step 2:** The original nonzero term(s) occurs in column 5

**Step 3:** Since ICHAINL (i=2) = 0 no previous rows will have any contributions to row 3.

The chained list for "future" row i=5 will have to be updated in order to include row 3 into its list:

ICHAINL(3) = 2

ICHAINL(2) = 2

LOCUPDATE(i=3) = 4

Thus Eqs. (39,43,42) state that "future" row i=5 will have to refer to rows 2, row 2 will refer to row 3, and row 3 will refer to row 2. Eq. (44) indicates that the updated starting location for row 3 is 4

**Step 1:** Consider row i=4

**Step 2:** The original nonzero term(s) occurs in column 5
Step 3: Since $ICHAIN(i=4) = 1$, and $ICHAINL(1) = 1$ (please refer to Eqs. 36-37), it implies row #4 will have contributions from row 1 only. The updated starting location of row 1 now will be increased by one, thus

$$LOCUPDATE(1) = LOCUPDATE(1) + 1$$  \hspace{1cm} (45)

Hence,

$$LOCUPDATE(1) = 1 + 1 = 2$$  \hspace{1cm} (46)

Since the updated location of nonzero term in row 1 is at location 2 (see Eq. 46), the column number associated with this nonzero term is column #6 (please refer to Eq. 31). Thus, it is obvious to see that there must be a "fills-in" term in column #6 of (current) row #4. Also, since $K_{1,6} = 2$ (or nonzero), it implies "future" row $i=6$ will have to refer to row 1. Furthermore, since the first nonzero term of row 4 occurs in column 5, it implies that "future" row 5 will also have to refer to row 4 (in additions to refer to rows 2 & 3). The chained list for "future" row 5, therefore, has to be slightly updated (so that row 4 will be included on the list) as following

$$ICHAINL(3) = 2$$  \hspace{1cm} (47)

$$ICHAINL(2) = 2$$  \hspace{1cm} (48)

$$LOCUPDATE(i=3) = 4$$  \hspace{1cm} (49)

Notice that Eq. (48) will override Eq. (43). Thus, Eqs. (39,48,47) clearly show that symbolically factorizing "future" row $i=5$ will have to refer to rows 2, then 4 and then 3, respectively.

Step 1: Consider row $i=5$

Step 2: The original nonzero term(s) occurs in column 6

Step 3: Since

$$ICHAINL (i=5) = 2$$  \hspace{1cm} (39, repeated)

$$ICHAINL (2) = 4$$  \hspace{1cm} (48, repeated)

$$ICHAINL (4) = 3$$  \hspace{1cm} (47, repeated)

It implies rows #2, then 4, and then 3 "may" have contributions (or "fills-in" effects) on row 5. However, since $K_{5,6}$ is originally a nonzero term, therefore, row 2, 4 and 3 will NOT have any "fills-in" effects on row 5.

Step 1: There is no need to consider the last row $i=N=6$, since there will be no "fills-in" effects on the last row.
It is extremely important to emphasize that upon completion of the symbolic phase, the output array: JCOLUMN(·) has to be re-arranged to make sure that the column numbers in each row should be in the increasing orders!

Sparse Numerical Factorization and Forward Backward Solutions

It is generally safe to say that sparse numerical factorization is more complicated for computer coding implementation than its skyline, or variable bandwidth cases. Main difficulties are due to complex "book-keeping" (or index referring) process. The "key" ideas in this numerical phase are still basically involved the creation and usage of the 2 integer arrays ICHAINL(·) and LOCUPDATE(·), which have been discussed with great details in Section 5. There are two (2) important modifications that need to be done on the symbolic factorization, in order to do the sparse numerical factorization (to facilitate the discussion, please refer to the data shown in Eq. 26):

a) For symbolic factorization purpose, there is no need to have any floating points, arithmetic calculation. Thus, upon completing the symbolic process for row 4, there are practically no needs to consider row 2 and/or row 3 for possible contributions to row 5. Only row 4 needs to be considered for possible contributions (or "fills-in" effects) to row 5 (since row 4, with its "fills-in", is already full).

For numerical factorization purpose, however, all rows 2, then 4, and then 3 will have to be included in the numerical factorization of row 5.

b) For sparse numerical factorization, the basic skeleton FORTRAN code for LDL T, shown in Table 3 of Section 5, can be used in conjunction with the chained list strategies (using arrays ICHAINL and LOCUPDATE) which have been discussed earlier in Section 5. The skeleton FORTRAN code for sparse is shown in Table 4. Comparing Table 4 and Table 3, one immediately sees the "major differences" only occur in the 2 do-loop indexes LDL T, on lines 3 and 6, respectively.

c) Since the sparse forward and backward equation solution phases require much less computational efforts (as compare to factorization phase), their discussions will be omitted in this work.

---

| 1. | Assuming row 1 has been factorized earlier |
| 2. | Do 11 I = 2, N |
| 3. | Do 22 K = Only those previous rows which have contributions to current row I |
| 4. | Compute the multiplier (Note: U represents L T) |
| 5. | XMULT = U(K,I) / U(K,K) |
| 6. | Do 33 J = appropriated column numbers of row # K |
| 7. | U(I,J) = U(I,J) - XMULT * U(K,J) |
| 8. | CONTINUE |
| 9. | U(K,I) = XMULT |
| 10. | CONTINUE |
| 11. | CONTINUE |

Table 4: Pseudo FORTRAN Skeleton Code For Sparse LDL T Factorization

Finding Master (or Super) Degree-of-Freedom (dof)

To simplify the discussion, assuming that upon completion of the symbolic phase, the stiffness matrix $[K]$ will have the following form

$$
[K] =
\begin{bmatrix}
    \ldots & x & x & x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    0 & x & x & x & x & x & x & x & x & x & x & x & x & x & x \\
    0 & 0 & x & x & x & x & x & x & x & x & x & x & x & x & x \\
    0 & 0 & 0 & x & x & x & x & x & x & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & x & x & x & x & x & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x \end{bmatrix}
$$

(50)

In Eq. (50), the stiffness matrix $[K]$ has 14 dof. The symbols "x" and "F" refer to the original nonzero terms, and the nonzero terms due to "fills-in", respectively. It can be seen that rows 1-3 have same nonzero patterns (by referring to the enclosed "rectangular" region, and ignoring the fully populated "triangular" region of rows 1-3). Similarly, rows 4-5 have same nonzero patterns. Rows 7-10 have same nonzero patterns. Finally, rows 11-14 also have same nonzero patterns. Thus, for the data shown in Eq. (50), the "Master" (or "Super") dof can generated as

$$
\text{MASTER} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 = N \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

(51)

According to Eq. (51), then the "master" (or "super") dof are dof # 1 (which is followed by 2 "slave" dof), dof # 4 (which is followed by 1 slave dof), dof # 6 (which has no slave dof!), dof # 7 (which is followed by 3 slave dof), and dof # 11 (which is followed by 3 slave dof).

**Sparse Matrix-Vector Multiplication (With Unrolling Strategies)**

In our developed sparse equation solver, upon obtaining the solutions, the user has the option to compute the relative error norm. For the error norm computation, one needs to have efficient sparse matrix (with unrolling strategies) vector multiplication.
Furthermore, efficient sparse matrix-vector multiplication's are also required in different steps of the subspace and Lanczos algorithms (see Section 4).

To facilitate the discussions, let's consider the coefficient (stiffness) matrix as shown in Fig. 1. This 14 dof matrix is symmetrical, and it has same nonzero patterns as the one considered earlier in Eq. (50). The master/slave dof for this matrix has been discussed and given in Eq. (51). The input data file associated with Fig. 1 follows exactly the same sparse numerical factorization procedures discussed earlier in Section 5 (see Eqs. 30-33). The sparse matrix-vector $[A]\{x\}$ multiplication (with unrolling strategies) can be described by the following step-by-step procedures (please also refer to Fig. 1).

**Step 0.1:** Multiplication's Between the Given Diagonal Terms of $[A]$ and vector $\{x\}$.

**Step 0.2:** Consider the first "master" dof. According to Fig. 1 (and Eq. 51), the first master dof is at row # 1, and this master dof has 2 associated slave dof. In other words, the first 3 rows of Fig. 1 have the same off-diagonal, nonzero patterns.

**Step 1:** The first 3 rows (within a rectangular box) of given matrix $[A]$ (shown in Fig. 1) operate on the given vector $\{x\}$.

**Step 2:** The first 3 columns (within a rectangular box) of the given matrix $[A]$ (shown in Fig. 1) operate on the given vector $\{x\}$.

**Step 3:** The upper and lower triangular portions (right next to the first 3 diagonal terms of the given matrix $[A]$) operate on the given vector $\{x\}$, according to the orders a (9., 9.), then b (1., 2.), and finally c (1., 2.) (as shown in Fig. 1).

**Step 4:** The row number corresponds to the next "master" dof can be easily computed (using the master/slave dof information, provided by Eq. 51).

If the next "master" dof number exceeds $N$ (where $N =$ total number of dof of the given matrix $[A]$, then stop, or else return to Step 0.2 (where the "first" master dof will be replaced by the "second" master dof etc.)
Third Step:
The upper and lower triangular region will finally be processed
(according to the order a, b, and c, respectively)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>1.2</td>
<td>3</td>
<td>4.5</td>
<td>6.7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>102</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.2</td>
<td>103</td>
<td>10</td>
<td>104</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>2.2</td>
<td>105</td>
<td>26</td>
<td>27</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>3.7</td>
<td>106</td>
<td>29</td>
<td>30</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>4.1</td>
<td>11</td>
<td>32</td>
<td>33</td>
<td>37</td>
<td>38</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>5.1</td>
<td>12</td>
<td>35</td>
<td>36</td>
<td>39</td>
<td>40</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>6.1</td>
<td>13</td>
<td>38</td>
<td>39</td>
<td>41</td>
<td>42</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>7.2</td>
<td>14</td>
<td>40</td>
<td>41</td>
<td>45</td>
<td>46</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>8.2</td>
<td>15</td>
<td>43</td>
<td>44</td>
<td>48</td>
<td>49</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
</tbody>
</table>

First Step
- These 3 rows will be processed
  (Dot Product Operations)

Second Step:
- These 3 columns will be processed (SAXPY operations)

Figure 1: Sparse Matrix-Vector Multiplication’s With Unrolling Strategies

Modifications For The Chained List Array ICHAINL(-)

The Chained list strategies discussed earlier in Section 5 need to be modified in
order to include the additional information provided by the MASTER dof (refer to, for
example, Eq. 51). The major modification that need to be done can be accomplished by
simply making sure that the chained list array ICHAINL(-) will be pointing only toward
the Master dof (and not toward the slave dof !)

Sparse Numerical Factorization With Unrolling Strategies

The Vector unrolling, and loop unrolling strategies that have been successfully
introduced earlier by the authors for skyline [21] and variable bandwidth [22] equation
solvers, can also be effectively incorporated into the developed sparse solver (in
conjunction with the Master dof strategies). Referring to the stiffness matrix data shown
in Eq. 50, for example, and assuming the first 10 rows of [U] have already been
completely factorized, thus our objective now is to factorize the current ith (= 11st) row.
By simply observing Eq. (50), one will immediately see that factorizing row # 11 will
required the information from the previously factorized row numbers 1,2,3,6,7,8,9, and
10 (not necessarily to be in the stated increasing row numbers!) in the "conventional"
sparse algorithm. Using "loop-unrolling" sparse algorithm, however, the chained list
array ICHAINL(-) will point only to the "master" dof # 6, # 7 and # 1.
The skeleton FORTRAN code for LDL T (with sparse matrix) shown in Table 4 (refer to
Section 5) should be modified as shown by the pseudo, skeleton FORTRAN code in Table
5. Comparing Table 4 (sparse LDL T factorization) and Table 5 (sparse LDL T
factorization, with unrolling strategies), one still can recognize the many similarities
between the 2 sparse algorithms.
Table 5: Pseudo FORTRAN Skeleton Code For Sparse LDLᵀ Factorization With Unrolling Strategies

6. NUMERICAL EVALUATIONS OF DIFFERENT GENERALIZED EIGEN-SOLVERS

Based upon the discussions in previous sections, practical finite element models (such as Exxon-off-shore Structure, and High Speed Civil Transport Aircraft) are used to evaluate the performance of the developed sparse eigen-solvers. Since the codes have been written in standard FORTRAN language (and without using any library subroutines), it can be ported to different computer platforms (such as SUN-Sparc-20, IBM-R6000/590, Intel Paragon, Cray-C90 etc...) with no (or minimum) changes to the codes. The accuracy of the developed codes for solving generalized eigen equations can be measured by the Relative Error Norm (R.E.N.) which can be computed as:

\[ R.E.N. = \frac{||K\phi - \lambda M\phi||}{||K\phi||} \]  

The basic subspace iteration code, given in Ref. [1], will be used as a based-line reference. This basic subspace iteration code [1] will be compared to the developed basic, "sparse" Subspace iteration, and "sparse" Lanczos codes. Lumped masses have been used in all examples in this section.

Example 1: EXXON Off-Shore Structure

The finite element model for the EXXON model has been used extensively in earlier research works [23-25]. The resulted system of generalized eigen-equations from the EXXON model has 23,155 dof. The number of nonzero terms of the original stiffness
matrix is 809,427. Using the Nested-Dissection (ND) algorithm, the number of nonzero terms (including "fills-in" terms) is 10,826,014. The relative error norm (or R.E.N., defined in Eq. 52) and the wall-clock time are presented and explained in Figures 2-3. It should be noted here that on the IBM-R6000/590 Workstation, vector processing capability is available, whereas the vector processing capability is "not" available on the Sun Sparc-20 workstation.

Example 2: High Speed Civil Transport (HSGT) Aircraft

The finite element model for the HSGT aircraft has been used extensively in earlier research works. The resulted system of linear equations from the HSGT model has 16,152 dof. The number of nonzero terms of the original stiffness matrix is 373,980. Using the Modified Minimum Degree (MMD) algorithm, the number of nonzero terms (including "fills-in" terms) is 2,746,286. The numerical performances of 3 generalized eigen-solvers are presented in Figures 4-5.

![Figure 2: Exxon model](Stretch, IBM-RS600/590 Workstation)

![Figure 3: Exxon Model](USSU31, Sun-Sparc20 Workstation)
7. CONCLUSION

In this paper, basic generalized eigen-solution algorithms are reviewed. Major computational tasks in Subspace iterations, and Lanczos algorithms have been identified. Efficient Sparse technologies have been developed, and fully utilized (such as: sparse symbolic, numerical factorization with unrolling strategies, sparse forward & backward solutions, sparse matrix-vector multiplication's etc...), in conjunction with the basic Subspace Iterations and Lanczos algorithms for efficient solutions of the generalized eigen-equations. Numerical results from practical finite element models have clearly indicated that the proposed "sparse" Subspace iterations, and Lanczos algorithms have offered substantial computational advantages over the traditional "skyline", or "variable bandwidth" strategies.
REFERENCES


--- Because J. Qin tries to re-use memory, error messages and/or wrong answers occur whenever ncoff is less than approx. 2*neq. Although this case rarely occurs in real, practical problems (may only occur for "artificial, small scale" examples, however it is annoying to have this "bugs". These bugs have been fixed in this SPARSEPACK version of eigen-solution.

Two (2) small changes have been made in spamain.f, and spaldln.f which involve mtota and if (ncoff ...)

--- For structures with rigid body motions (such as "floating" structures, without any supports), the "correct error norm" should be close to 1.0 (instead of close to 0.0), because error norm = ||K*Phi - Lamda*M*Phill|| / ||K*Phill|| since Lamda = eigen-value = 0.0 (corespond to rigid body motion), hence error norm = ||K*Phill - 0.0 || / ||K*Phill|| = close to 1.0. For "float structures" we need to apply "shift factor" to avoid "singular" stiffness matrix.

NOTES:

- MREAD < 0 Read K.*
- > 0 Read fort.*
- NREORD = 0 no reordering
- = 3 MMD reordering
- LUMP = 0 CONSISTANT MASS ( or Lump.ne.0)
- = 1 LUMPED MASS
- NEIG = number of desired eigenvalues and corresponding eigenvectors
- N = number of equations
- NCOFF = number of nonzeros coefficients
- ISHIFT = 0 no shift
- .NE.0 perform a shift of value "ishift"
- IBLOCK = 0 : Regular Lanczos
- = 1,2,3 : block Lanczos ( of Block = iblock)
- ITIME = 0 save memory
- = 1 save time

---

```
PROGRAM SPARSEPACK97

EQUATION-EIGEN SOLVER FOR SPARSE POSITIVE DEFINITE SYSTEMS

H. Runesha : runesha@cee.odu.edu
Prof. Duc Nguyen : nguyen@cee.odu.edu

Last update: Dec 20, 1997
--- October 26, 1998
--- Because J. Qin tries to re-use memory, error messages and/or wrong answers occur whenever ncoff is less than approx. 2*neq. Although this case rarely occurs in real, practical problems (may only occur for "artificial, small scale" examples, however it is annoying to have this "bugs". These bugs have been fixed in this SPARSEPACK version of eigen-solution.

Two (2) small changes have been made in spamain.f, and spaldln.f which involve mtota and if (ncoff ...)

--- For structures with rigid body motions (such as "floating" structures, without any supports), the "correct error norm" should be close to 1.0 (instead of close to 0.0), because error norm = ||K*Phi - Lamda*M*Phill|| / ||K*Phill|| since Lamda = eigen-value = 0.0 (correspond to rigid body motion), hence error norm = ||K*Phill - 0.0|| / ||K*Phill|| = close to 1.0. For "float structures" we need to apply "shift factor" to avoid "singular" stiffness matrix.

NOTES:

- MREAD < 0 Read K.*
- > 0 Read fort.*
- NREORD = 0 no reordering
- = 3 MMD reordering
- LUMP = 0 CONSISTANT MASS ( or Lump.ne.0)
- = 1 LUMPED MASS
- NEIG = number of desired eigenvalues and corresponding eigenvectors
- N = number of equations
- NCOFF = number of nonzeros coefficients
- ISHIFT = 0 no shift
- .NE.0 perform a shift of value "ishift"
- IBLOCK = -1 : Subspace iteration
- = 0 : Regular Lanczos
- = 1,2,3 : block Lanczos ( of Block = iblock)
- ITIME = 0 save memory
- = 1 save time

---

implicit real*8 (a-h,o-z)
character*70 title

real*8 a(48 875 248)
mtot = 48 875 220

real*8 a(28 875 248)
mtot = 28 875 220

real*8 a(20 000 000)
mtot = 20 000 000

real*8 a(9 000 000)
mtot = 9 000 000

open(unit=7,file='K.INFO',status='old',form='formatted')
read(7,115) title
read(7,115) nreord,neig,lump,n,n2,NCOFF,itime,ishift,iblock,mread
write(*,*) nreord,neig,lump,n,n2,NCOFF,itime,ishift,iblock,mread
close (7)
```

23
write(23,*) title
write(23,*) ' NEQ = ', n
write(23,*) ' NCOEF = ', ncoff
write(23,*) ' NEIG = ', neig
write(23,*) ' ISHIFT = ', ishift
write(23,*) ' MREAD = ', mread
write(23,*) ' LUMP = ', lump
write(23,*) ' IBLOCK = ', iblock
write(23,*) ' NREORD = ', nreord
write(23,*) ' ITIME = ', itime

if(iblock.eq.0.and.lump.ne.1) then
  write(23,*)'Sorry! The old Lanczos path can not deal with Consistant Mass!'
  stop
endif
if(iblock.gt.3) then
  write(23,*)'Sorry! BLOCK size must less than 4! '
  stop
endif
if(iblock.eq.0.and.ishift.ne.0) then
  write(23,*)'Sorry! No shift is allowed for the Old Lanczos!'
  stop
endif

Read Data
if(mread.le.0) then
  CALL QREAD(n,ncoff,mtot,a,mread,neig,lump,ishift)
  CALL OOCSPA(n,mtot/2,ncoff,a(1),a(mtot/2+4),mread,n1100)
endif

REORDERING
nreord .NE. 0 means MMD reordering is desired ***
mtota = max(4*n,2*ncoff)
mtoti = 3*ncoff + 7*n + 5
if(mtot-mtoti-mtota.lt.0) then
  write(23,*) SPAOOC: increase mtot to: ',mtoti+mtota
  stop
endif
call cputime(time0)
if(nreord.ne.0) then
  IF(LUMP.EQ.1) THEN
    call reord(n,ncoff,nreord,mtota,mtoti,a(1),a(1+mtota),neig)
  ELSE
    if(itime.eq.0) then
      call reord2(n,ncoff,nreord,mtota,mtoti,a(1),a(1+mtota),neig)
    else
      call reord3(n,ncoff,nreord,mtota,mtoti,a(1),a(1+4*ncoff),neig)
    endif
  ENDIF
endif

SOLVER
call sparse(n,ncoff,neig,mtot/2,mtot/2,a(1),a(1+mtot/2))
using 4 bytes for integer .... June 7, 1995 ......
mtoti=mtot/4+1000
mtota=mtot-mtoti
call sparse(n,ncoff,neig,lump,iblock,mtoti,mtota,a(1),a(1+mtoti),
1
  ishift)
call cputime(time1)
timel=time1 - time0
if(neig.ne.0) then
  write(23,*)'***TOTAL CPU FOR EIGENSOLUTION = ', timel
  write(23,*)'*** (This time including norm check & I/O )***'
  write(23,*)'MTOTI = ',mtoti,
  MTOTA = ',mtota
endif
stop
end
**SUBROUTINE OOCSPA(n,mtota,ncoff,a,iq,mread,n1100)**

```
implicit real*8 (a-h,o-z)
real*8 a(1)
integer iq(1)
if (mread.le.0) then
  rewind (15)
  read (15) (iq(i), i=n+2,1+2*n)
  iq(1) = i
  do 10 i=2,n+1
    iq(i)=iq(i-1)+iq(i+n)
  10 rewind (15)
  write(15) (iq(i), i=1,n+1)
else
  if (n.gt.0) stop
endif
return
end
```

**SUBROUTINE QREAD(neq,ncoeff,mtot,a,mread,neig,lump,ishift)**

```
c dec 20 1997 : Runeshah
```

```
if (lump.ne.1 .or. ishift.ne.0) then
  call rreadpierrot(neq,ncoeff,neig,lump,ishift,a(1),a(n+1),
   a(ncoeff+n+1),a(ncoeff+2*n+1),a(ncoeff+3*n+1))
else
  call rread(neq,ncoeff,mtot,a,mread,neig,lump)
endif
return
end
```

**SUBROUTINE RREAD(neq,ncoeff,mtot,a,mread,neig,lump)**

```
if (mread.lt.0) then
  OPEN (unit=54, file='KII.COEFS', form='formatted', status='old')
  OPEN (unit=55, file='K.DIAG', form='formatted', status='old')
  OPEN (unit=56, file='K.RHS', form='formatted', status='old')
  OPEN (unit=57, file='K.DMASS', form='formatted', status='old')
  if(neig.ne.0 and lump.ne.1) OPEN(unit=58,file='K.CMASS',
   form='formatted',status='old')
else
  OPEN (unit=55, file='K.DIAG.B',
   3form='unformatted',status='old')
  OPEN (unit=54, file='KII.COEFS.B',
   3form='unformatted',status='old')
  OPEN (unit=56, file='K.RHS.B',
   3form='unformatted',status='old')
  OPEN (unit=57, file='K.DMASS.B',
   3form='unformatted',status='old')
  if(neig.ne.0 and lump.ne.1) OPEN(unit=58,file='K.CMASS.B',
   3form='unformatted',status='old')
endif
jfile = 12
ISEG = (ncoeff - 1 ) / mtot + 1
DO 100 I = 1,ISEG
  istart = (i-1)*mtot + 1
  iend = min(ncoeff,i*mtot)
  length = iend - istart + 1
  if (mread.lt.0) then
    read(42+jfile,*) (a(k), k=1,length)
    read(42+jfile,9901) (a(k),k=1,length)
    9901 format (6E12.5)
  else
    read(42+jfile) (a(k), k=1,length)
  endif
100 continue
if (mread.lt.0) then
  read(55,*) (a(i),i=1,neq)
```
SUBROUTINE IREAD (neq, ncoeff, rotor, ia, reread, lump)
  integer ia(1)
  if(mread.lt.0) then
    OPEN(unit=52, file='K. PTRS', form='formatted', status='old')
    OPEN(unit=53, file='KII. INDXS', form='formatted', status='old')
  else
    OPEN(unit=52, file='K. PTRS.B', form='unformatted', status='old')
    OPEN(unit=53, file='KII. INDXS.B', form='unformatted', status='old')
  endif
  jfile = 11
  ISEG = ( ncoeff - 1 ) / mtot + 1
  DO 100 I = 1,ISEG
    istart = (i-1)*mtot + 1
    iend = min(ncoeff,i*mtot)
    length = iend - istart + 1
    if(mread.lt.0) then
      read(42+jfile,*) (ia(k),k=1,length)
    else
      read(42+jfile) (ia(k),k=1,length)
    endif
    rewind(jfile)
    write(jfile) (ia(k),k=1,length)
  100 continue
  if(mread.lt.0) then
    read(52,*) (ia(k),k=1,neq)
  else
    read(52) (ia(k),i=k,neq)
  endif
  rewind(15)
  write(15) (ia(k),k=1,neq)
  return
end

SUBROUTINE RREADPIERROT (n, ncoef, neig, lump, ishift, ad, an, b, dm, am)
  implicit real*8 (a-h,o-z)

dimension ad(*), b(*), an(*), dm(*), am(*)

OPEN(unit=54, file='Kll.COEFS', form='formatted', status='old')
OPEN(unit=55, file='K.DIAG', form='formatted', status='old')
OPEN(unit=56, file='K.RHS', form='formatted', status='old')
OPEN(unit=57, file='K.DMASS', form='formatted', status='old')
if(neig.ne.0.and.lump.ne.1) OPEN(unit=58, file='K.CMASS',
1form='formatted', status='old')

perform also a shift on the diag values

write(54,*) 'ISHIFT =', ishift
read(54,*) (an(i), i=1, ncoeff)
read(55,*) (ad(i), i=1, n)
write(56,*) 'AD before =', (ad(i), i=1, n)
read(57,*) (dm(i), i=1, n)
close(54)
close(55)
close(56)
close(57)
IF (NEIG.NE.0) THEN
OPEN(unit=57, file='K.DMASS', form='formatted', status='old')
read(57,*) (dm(i), i=1, n)
close(57)
rewind(10)
write(10,*) (dm(i), i=1, n)
if(ishift.ne.0) then
doi=1, n
ad(i)=ad(i)+float(ishift)*dm(i)
write(54,*) dm(i), float(ishift)*dm(i), ad(i)
enddo
endif
if(lump.ne.1) then
OPEN(unit=58, file='K.CMASS', form='formatted', status='old')
read(58,*) (am(i), i=1, ncoeff)
close(58)
rewind(17)
write(17,*) (am(i), i=1, ncoeff)
if(ishift.ne.0) then
doi=1, ncoeff
an(i)=an(i)+float(ishift)*am(i)
enddo
endif
ENDIF

write on fort.*

rewind(13)
write(13,*) (ad(i), i=1, n)
write(13,*) 'Inside read AD =', (ad(i), i=1, n)
rewind(12)
write(12,*) (an(i), i=1, ncoeff)
rewind(14)
write(14,*) (b(i), i=1, n)
return
end

subroutine cputime(time)
real tar(2)
real*8 time
time=etime(tar)
time=TSECND()
time=0.01*mclock()
return
end

subroutine sparse(n, ncoeff, neig, lump, iblock, mtoti, mtota, iq, a,
1 ishift)
implicit real*8 (a-h, o-z)
real*8 a(l)
integer iq(1)
   ierr=0
   
c......purpose: sparse gauss version equation solver
   for system of symmetrical equations.
   J. Qin Dec.28, 1992
   H. Runesha June 1996

C*** UDU factorization ********
mtoti=mtoti*2
do 81 i=1,ncoff+n+n
   a(i)=0.0d0
   call cputime(timer0)
   rewind(15)
   read(15)(iq(i),i=1,n+1)
   rewind(11)
   read(11)(iq(i),i=n+2,n+1+ncoff)
   rewind(13)
   read(13)(a(i),i=1,n)
   rewind(12)
   read(12)(a(i),i=2*n+1,2*n+ncoff)
   rewind(14)
   read(14)(a(i),i=1*n,2*n)
   call cputime(timer1)
   call cputime(time00)
   iqmax = mtoti-ncoff-6*n-3
   call symfac(n,iq(1),iq(n+2),iq(n+2+ncoff),iq(ncoff+2*n+3),
   ! a(2*n+ncoff+1),ncof2,iqmax)
   call cputime(time01)
   write(23,*'(neg = ',n
   write(23,*'before fill in, ncoff = ',ncoff
   write(23,*'after fill in, ncof2 = ',ncof2
   write(23,*'Total integer memory used = ',ncoff+ncof2+
   5*n+3
   write(23,*'Total real memory used = ',ncoff+ncof2+4*n
   if(mtoti.lt.(ncoff+3+6*n+ncof2)) then
      ierr=1
   endif
   if(ierr.ne.0) stop
   call cputime(time02)
   call transa(n,n,iq(n+2+ncoff),iq(ncoff+2*n+3),a(ncoff+2*n+1),
   ! a(ncoff+3*n+2))
   call cputime(time03)
   call supnode(n,iq(n-2+ncoff),iq(ncoff+2*n+3),iq(ncoff+4*n+3
   ! +ncof2))
   
C*** if neig.ne.0.and.ishift.ne.0) do shifting: K=K+ishift*M
   
Ccpier
   
   If(iblock.gt.0) then
   IF(NEIG.NE.0.AND.ISHIFT.NE.0) THEN
      rewind(10)
      read(10)(a(2*n+ncoff+i),i=1,n) ! read dmass
      if(lump.ne.1) then
         rewind(17)
         read(17)(a(3*n+ncoff+i),i=1,ncoff) ! read am for consist. mass
         endif
      do i = 1,n
         a(i) = a(i) + float(ishift)* a(2*n+ncoff+i) ! for diagonals
      enddo
      rewind(13)
      write(13)(a(i),i=1,n)
      if(lump.ne.1) then
         do i = 1,ncoff
            a(2*n+i)=a(2*n+i) + float(ishift)*a(3*n+ncoff+i)
         enddo
         rewind(12)
         write(12)(a(2*n+i),i = 1,ncoff)
      endif
      
28
ENDIF

c endif
cpier

call cputime(time031)
c
subroutine Numfa1(n,ia,ja,an,iu,ju,di,un,ip,iup,tl,isupd
call numfaq(n,iq(1),iq(n+2),a(1),a(2*n+1),iq(n+2+ncoff),
 1 iq(ncoff+2*n+3),a(2*n+ncoff+1),a(ncoff+3*n+1),
 1 iq(ncoff+2*n+3+ncoff2),iq(ncoff+3*n+3+ncoff2),
 1 a(3*n+ncoff+ncoff2+1),iq(ncoff+4*n+3+ncoff2)
 1 ,iopf)
call cputime(time04)

IF(NEIG.NE.0) THEN
  if(iblock.eq.-1) then
    write(23,*)'IBLOCK =',iblock

************SUBSPACE iteration***

c DM
  rewind(10)
  read(10) (a(ncof2+ncoff+3*n+i),i=1,n)  ! read dmass
  i00=ncof2+ncoff+3*n+1
  i01=i00+n
  ncmass=1
  if(lump.ne.1) then
    ncmass=ncoff
  AM
  rewind(17)
  read(17) (a(i01-1+i),i=1,ncoff)  ! read am for consist. mass
endif

iiql=min(neig+8,2*neig)
iq=iiq=1
iiq=iiq=min(iiql,n)
n=iiq=n
mtot=mtoti+mtota
i02=101+ncmass
i03=102+i02*iiq
i04=103+i03*iiq
i05=104+i04*iiq
i06=105+i05*iiq
i07=106+n
i08=107+n
i09=108+n
i10=109+n*iiq

if(i10.gt.mtota) write(*,*) 'NOT ENOUGH MEMORY '
c

  call spasubspace(iiq,n,ncoff,neig,lump,mtot,iq(1),iq(n+2)
  +,a(1),a(2*n+1),a(i00),a(i01),iq(n+2+ncoff),iq(ncoff+2*n+3)
  +,a(2*n+ncoff+1),a(ncoff+3*n+1),a(i02),a(i03),a(i04)
  +,a(i05),a(i06),a(i07),a(i08),a(i09),ishift)
c
return

c endif
  c****** end subspace iter.******

  if(iblock.ge.1) then
    if(iblock.gt.6) iblock = 6
C######################################################################
C** Block Lanczos eigensolver:
C** Ig(): 1->n+1 =ia; n+2-> ja; n+2+ncoff -> iu; 2*n+3+ncoff -> ju;
C** total available = mtoti-(2*n+5+ncoff+ncof2)
ileft = (mtoti-(2*n+6+ncoff+ncof2))

isleiz = isize= isize/iblock)*iblock
isize = isize*isize+6 + iblock*iblock*6 + isize*2
write(*,*)'ileft,ineed = ','ileft,ineedc

  if(ineed.lt.ileft) then
    i00 = 2*n+6+ncoff+ncof2
    i01 = i00+isize*isize*2
    i02 = i01+isize*isize*2
    i03 = i02+isize*isize*2
    i04 = i03+iblock*iblock*2
    i05 = i04+iblock*iblock*2
    i06 = i05+iblock*iblock*2
    else
      write(23,*)'Increase mtoti to : ','ineed+2*n+5+ncoff+ncof2
stop
C endif

C** a() : l->n = ad; n->2*n = RHS; 2*n+1-> = an; di; un; t1;
  
  \texttt{f02 = 2*n+ncoff + 1} ! di, unchanged

if(lump.ne.1) then

C*** consistent Mass matrix, using ad,an space for: dmass,am

  \texttt{j00 = 1} ! dmass
  \texttt{j01 = j00 + n} ! am
  \texttt{j02 = j02} ! r(n,iblock)
  \texttt{j03 = j03 + n*iblock} ! p(n,iblock)
  \texttt{j04 = j04 + n*iblock} ! tem(n,iblock)
  \texttt{j05 = j05 + n*iblock} ! Q

else

C** lumped mass matrix:

ibmax = \((2*n+ ncoff)/(3*n)\)

write(*,*)'ibmax, iblocki,isize = ',ibmax, iblock, isize

C****

if(ibmax.ge.iblock) then

  \texttt{j00 = 3*n+ncoff+ncof2+1} ! dmass
  \texttt{j01 = j00 + n} ! am
  \texttt{j02 = j02} ! r(n,iblock)
  \texttt{j03 = j03 + n*iblock} ! p(n,iblock)
  \texttt{j04 = j04 + n*iblock} ! tem(n,iblock)
  \texttt{j05 = j05 + n*iblock} ! Q

else

  \texttt{j00 = 1} ! dmass
  \texttt{j01 = 1+n} ! am
  \texttt{j02 = 2+n} ! r(n,iblock)
  \texttt{j03 = 2+n} ! p(n,iblock)
  \texttt{j04 = j04 + n*iblock} ! tem(n,iblock)
  \texttt{j05 = j05 + n*iblock} ! Q

write(*,*)'2*n+ncoff,j03+n*iblock = ',2*n+ncoff,j03+n*iblock

endif

endif

i00 = j06 + n*(isize+iblock) ! T(isize,isize)
i01 = 100+isize*isize ! B(isize,isize)
i02 = 101+isize*isize ! vec(isize,isize)
i03 = 102+isize*isize ! alfa(iblock,iblock)
i04 = 102+iblock*iblock ! beta(iblock,iblock)
i05 = 102+iblock*iblock ! eig(isize)
i06 = 105+iblock*iblock ! eig(isize)

if(i06+isize.gt.mtota) then

write(23,*)'Increase mtota to : ',i06+isize

stop

endif

C****

write(*,*)'before call blanmain: lump = ',lump

call blanmain(n,isize,iblock,neig,a(j02+n),a(j02),iq(n+2+ncoff),
  1 iq(2*n+3+ncoff) ,a(j03) ,a(j04) ,a(j05) ,a(j06) ,a(i04) ,a(i05) ,a(i06) ,a(i00) ,a(i01) ,a(j01) ,iq(1) ,iq(2+n) ,lump,ncoff,ishift)

return

endif

C********************************************************************

C******

write(*,*)'before call blanmain: lump = ',lump

call blanmain(n,isize,iblock,neig,a(j02+n),a(j02),iq(n+2+ncoff),
  1 iq(2*n+3+ncoff) ,a(j03) ,a(j04) ,a(j05) ,a(j06) ,a(i04) ,a(i05) ,a(i06) ,a(i00) ,a(i01) ,a(j01) ,iq(1) ,iq(2+n) ,lump,ncoff,ishift)

return

endif

C**************** for regular Lanczos:

ax = neig**2
bx = 3*neig
cx = n - ncoff

if(bx*bx-4.0*cx*ax.le.0.0) then

  x1 = 1
  go to 1995

endif

x1 = (-bx+sqrt(bx*bx-4.0*cx*ax))*0.5/ax

1995 lanmax = min(8*neig,neig*x1)

write(23,*)'LANMAX = ',lanmax,x1*neig

lanmax = 4*neig

item = neig*(12 + 16*neig) + n

if(item.gt.ncoff) then

  lanmax = 3*neig
  item = neig*(9 + 9*neig) + n

endif

ij0=2+4*n+ncoff+ncof2

if(ij0+lanmax**2.gt.mtota) then

write(23,*)'Increase Memory(real) to : ',ij0+lanmax**2

stop

endif

if(lanmax.lt.3*neig) then

Pierrot , je fermer ca mais il faut le verifier

write(23,*)'Increase memory for: VEC(lanmax,lanmax) &EIG,ERR,TEM',
1 item, 'memory available now is: ',ncoff
lanmax=4*neig
if(lanmax.gt.n+2) lanmax=n+1
j00=1+4*n+ncoff+1+ncof2
l00=j00+1+ncoff*lanmax**2
if(j00+lanmax**3+lanmax**2+n*lanmax.gt.mtota) then
  write(23,*)'Increase memory mtota to : ',j00+3*lanmax+
lanmax**2
  stop
eendif
write(23,*)'lanmax,j00,ij0,ncoff,ncof2=',lanmax, j00,ij0,ncoff,ncof2
j01 = 2+n
if(ncoff.lt.2*n) then
  j01 = 2*n+3+ncoff+ncof2 + 1
endif
if(lump.eq.1) then
  CALL SP2LAN(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),i(q(n+2+
1ncoff)),i(q(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
2ncof2),a(1+2*n),a(j00+1),a(j00+1+lanmax),
3a(1+j00+2*lanmax),a(1+j00+3*lanmax),ncoff,ncof2,iq(j01),a(ij0) +
  , lump,ishift)
else
  j01 = 2*n+3+ncoff+ncof2 + 1
if(j00+lanmax*3+lanmax**2+n*lanmax+ncoff.gt.mtota) then
  write(23,*)'Increase memory mtota to : ',j00+3*lanmax+
  lanmax**2
  stop
eendif
CALL SP2LAN2(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),i(q(n+2+
1ncoff)),i(q(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
2ncof2),a(1+2*n),a(j00+1),a(j00+1+lanmax),
3a(1+j00+2*lanmax),a(1+j00+3*lanmax),ncoff,ncof2,iq(j01),a(ij0) +
  , a(ij0+lanmax*n), lump, ishift)
else
  j01 = 2*n+3+ncoff+ncof2 + 1
if(j00+lanmax*3+lanmax**2+n*lanmax+ncoff.gt.mtota) then
  write(23,*)'Increase memory mtota to : ',j00+3*lanmax+
  lanmax**2
  stop
eendif
CALL SP2LAN2(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),i(q(n+2+
1ncoff)),i(q(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
2ncof2),a(1+2*n),a(j00+1),a(j00+1+lanmax),
3a(1+j00+2*lanmax),a(1+j00+3*lanmax),ncoff,ncof2,iq(j01),a(ij0) +
  , a(ij0+lanmax*n), lump, ishift, iq(1),iq(n+2))
eendif
return
endif
j01 = 2+n
if(ncoff.lt.n) then
  j01 = 2*n+3+ncoff+ncof2 + 1
endif
if(lump.eq.1) then
  CALL SP2LAN(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),i(q(n+2+
1ncoff)),i(q(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
2ncof2),a(1+2*n),a(j00+1),a(j00+1+lanmax),
3a(1+j00+2*lanmax),a(1+j00+3*lanmax),ncoff,ncof2,iq(j01),a(ij0) +
  , lump, ishift)
else
  j01 = 2*n+3+ncoff+ncof2 + 1
if(j00+lanmax*3+lanmax**2+n*lanmax+ncoff.gt.mtota) then
  write(23,*)'Increase memory mtota to : ',j00+3*lanmax+
  lanmax**2
  stop
eendif
CALL SP2LAN2(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),i(q(n+2+
1ncoff)),i(q(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
2ncof2),a(1+2*n),a(j00+1),a(j00+1+lanmax),
3a(1+j00+2*lanmax),a(1+j00+3*lanmax),ncoff,ncof2,iq(j01),a(ij0) +
  , a(ij0+lanmax*n), lump, ishift, iq(1),iq(n+2))
eendif
return
ELSE
  call fbe(n,iq(n+2+ncoff),iq(ncoff+2*n+3),a(2*n+ncoff+1),
1a(ncoff+3*n+1),a(n+1),a(3*n+ncoff+ncof2+1),iopfb)
call cputime(time05)
call multspa(n,iq(1),iq(n+2),a(2*n+1),a(1),a(3*n+ncoff+
1ncof2+1),a(2*n+ncoff+1))
call cputime(time06)
sum0=0.0d0
sum2=0.0d0
j1=3*n+ncoff+ncof2
j2 = 2*n+ncoff
dmax = 0.0d0
bnorm = 0.0d0
axbn = 0.0d0

do 2000 i=1,n
  j = j2 + i
  bnorm = bnorm + a(n+i)*a(n+i)
axbn = axbn + (a(jj)-a(n+i))**2
if(abs(a(j)).gt.dmax) then
  dmax = abs(a(j))
imax = i
  endif
continue
2000
axbn = sqrt(axbn)
bnorm = axbn/sqrt(bnorm)

write(23,*) 'The Max. Displacement = ',a(jl+imax),' at the', imax,' th D.O.F.'
write(23,*) 'The ABS sum. of the displacements =',sum
write(23,*) 'The sum. of the displacements =',sum2
write(23,*) 'The I I AX--bbAx I =',axbn
write(23,*) 'The I b I =',bnorm
write(23,*) 'Time for error norm check =',time06-time05
write(23,*) 'Time for reading files =',timer1-time00
write(23,*) 'Time for symbolic factorization =',time01-time00
write(23,*) 'Time for reordering =',time03-time02
write(23,*) 'Time for Finding out supernodes =',time031-time03
write(23,*) 'Time for numeric factorization =',time04-time031
write(23,*) 'Time for forward/backward solve =',time05-time04
write(23,*) 'Total time =',time01-time00

loop = 8
write(23,*) 'LOOP UNROLLING LEVEL =',LOOP
write(23,*) 'Mflops for factorization =',float(iopf)*
write(23,*) 'Mflops for forward/backward =',float(iopfb)*

ENDIF

subroutine symfac(n, ia, ja, iu, ju, ip, ncof2, iqmax)
implicit real*8 (a-h,o-z)
integer ia(1),ja(1),iu(1),ju(1),ip(1)
nm = n - 1
nh = n + 1
do i0 i = 1, n
  iu(i) = 0
ip(i) = 0
jp = 1
do 90 i = 1,nm
  jpi = jp
if(jpi.gt.iqmax) then
  write(23,*) 'Symbolic: increase ju() bigger than: ',jpi,
  ** now we only have iqmax = ',iqmax
  stop
endif
  jpp = n + jp - i
  min = nh
  iaa = ia(i)
  iab = ia(i+1) - 1
  if ( iab.lt.iaa ) go to 30
  j0=jpi-iaa
  CDIR$ IVDEP
do 20 j = iaa, iab
    jj = ja(j)
    ju(j0+j) = jj
    iu(jj) = i
    jp=jp+(iab-iaa)+1
    min=ja(iaa)
  20
  last = ip(i)
if( last.eq.0 ) go to 60
  l = last
  30
  lh = l + 1
  iua = iu(l)
  iub = iu(lh) - 1
  if ( 1h.eq.1 ) iub = jpi - 1


```fortran

      iu(i) = i
      do 50 j = iua, iub
        jj = ju(j)
        if ( iu(jj).eq.i ) go to 50
        ju(jj) = jj
      ip = ip + 1
      iu(jj) = i
      if ( min.gt.jj ) min = jj
      50 continue
      if ( ip.eq.jpp ) go to 70
      ip(i) = ip(l)
      ip(l) = i
      go to 90
      ip(min) = i
      ip(l) = i
      if ( l.ne.last ) go to 40
      60 if ( min.eq.nh ) go to 90
      70 1 = ip(min)
      if ( l.eq.0 ) go to 80
      ip(i) = ip(1)
      ip(1) = i
      go to 90
      80 ip(min) = i
      ip(i) = i
      90 iu(i) = \]'pi
      iu(n) = 3P
      iu(nh) = 3P
      ncof2=iu(n)
      return

end

subroutine fbe(n, iu, ju, di, un, b, x, iopfb)
implicit real*8 (a-h,o-z)
real*8 un(1) ,di(1) ,b(1) ,x(1)
integer iu(1) ,ju(1)
  iopfb=0
  nm = n - 1
  do 10 i = 1, n
    x(i) = b(i)
    do 40 k = 1, nm
      iua = iu(k)
      iub = iu(k+1) - 1
      xx = x(k)
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        x(jj) = x(jj) * xx*un(i)
      iopfb=iopfb+2*(iub-iua+1)
      30 continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
        iopfb= iopfb+n
      41 continue
      if ( iub.lt.iua ) go to 30
      CDIR$ IVDEP
      do 20 i = iua, iub
        jj=ju(i)
        20 x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
      continue
      40 continue
      do 41 jj = 1, n
        x(jj) = x(jj)*di(jj)
      33
```
iat(1) = 1
iat(2) = 1
if (m.eq.1) go to 40
do 30 i = 3, mh
30 iat(i) = iat(i) + iat(i-1)
do 60 i = 1, n
iaa = ia(i)
lab = ia(i) - 1
if (lab.lt.iaa) go to 60
CDIR$ IVDEP
do 50 j = ja(jp) + 1
kp = iat(k)
jat(k) = i
50 continue
call tran2(n,m,iat,jat,ia,ja)
return
end
subroutine tran2(n,m, iat,jat,ia,ja)
implicit real*8 (a-h,o-z)
integer ia(1),ja(1),iat(1),jar(1)
mh = m + 1
nh = n + 1
do 10 i = 2, mh
10 iat(i) = 0
iaab = ia(nh) - 1
do 21 jj = 1, n
CDIR$ IVDEP
do 20 i = ia(jj),ia(jj+1)-1
j = ja(i) + 2
iat(j) = iat(j) + 1
20 continue
21 continue
iat(1) = 1
iat(2) = 1
if (m.eq.1) go to 40
do 30 i = 3, mh
30 iat(i) = iat(i) + iat(i-1)
do 60 i = 1, n
iaa = ia(i)
lab = ia(i) - 1
if (lab.lt.iaa) go to 60
CDIR$ IVDEP
do 50 j = ja(jp) + 1
kp = iat(k)
jat(k) = i
50 continue
return
end
C******************************************************************************
C subroutine cputime(time)
C real tar(2)
C real*8 time
C time=etime(tar)
C time=TSECND()
C time=0.01*mclock()
C return
C end
C******************************************************************************
C subroutine numfaq(n,ia,ja,ad,an,iu,ju,di,un,ip,iup,tl,isu,ipf)
imlicit real*8 (a-h,o-z)
real*8 an(1),ad(1),un(1),di(1),tl(1)
integer ia(1),ja(1),iu(1),ju(1),ip(1),iup(1),isup(1)
LOOP = 8
ipf=0
call numfa0(n,ia,ja,an,ad,un,di,tl)
do 10 j = 1, n
10 ip(j) = 0
do 130 i = 1,n
ih = i + 1
iua = iu(i)
iub = iu(ih) - 1

do 20 j = iua, iub
  do (ju(j)) = un(j)
  di(i) = ad(i)
  LN = ip(i)
  if (LN.eq.0) go to 90
  L = ln
  ln = ip(L)
  ik = min(isupd(L),i-L)
  iend = (ik/LOOP)*LOOP
  iuc0 = iup(L)
  iuc00 = iuc0
  iuc(L) = iuc0 + 1
  iud = iu(L+1) - 1
  idif = iud - iuc0
  ipf = ipf+2*ik*(idif+1)+ik
  do 1000 k = L,L+iend-1,LOOP
       j2 = iu(K+2) - 1 - idif
       j3 = iu(K+3) - 1 - idif
       j4 = iu(K+4) - 1 - idif
       j5 = iu(K+5) - 1 - idif
       j6 = iu(K+6) - 1 - idif
       j7 = iu(K+7) - 1 - idif
       j8 = iu(K+8) - 1 - idif
       um = un(iuc0)*di(K)
       um2 = un(j2)*di(K+1)
       um3 = un(j3)*di(K+2)
       um4 = un(j4)*di(K+3)
       um5 = un(j5)*di(K+4)
       um6 = un(j6)*di(K+5)
       um7 = un(j7)*di(K+6)
       um8 = un(j8)*di(K+7)
  1000 continue
  if (ileft.eq.0) go to 1030
  go to (1011,1012,1013,1014,1015,1016,1017,1018) ileft
  go to 1030
  1080 continue
  j2 = iu(Lend+2) - 1 - idif
  j3 = iu(Lend+3) - 1 - idif
  j4 = iu(Lend+4) - 1 - idif
  j5 = iu(Lend+5) - 1 - idif
  j6 = iu(Lend+6) - 1 - idif
  um = un(iuc0)*di(Lend)
  um2 = un(j2)*di(Lend+1)
  um3 = un(j3)*di(Lend+2)
  um4 = un(j4)*di(Lend+3)
  um5 = un(j5)*di(Lend+4)
  um6 = un(j6)*di(Lend+5)
  um7 = un(j7)*di(Lend+6)
  um8 = un(j8)*di(Lend+7)
  1076 continue
  go to 1030
  1070 continue
  j2 = iu(Lend+2) - 1 - idif
  j3 = iu(Lend+3) - 1 - idif
  j4 = iu(Lend+4) - 1 - idif
  j5 = iu(Lend+5) - 1 - idif
  j6 = iu(Lend+6) - 1 - idif
  um = un(iuc0)*di(Lend)
  um2 = un(j2)*di(Lend+1)
  um3 = un(j3)*di(Lend+2)
  um4 = un(j4)*di(Lend+3)
  um5 = un(j5)*di(Lend+4)
  um6 = un(j6)*di(Lend+5)
CDIR$ IVDEP
  do 1066 j = 0, idif
     di(ju(iuc0+j)) = di(ju(iuc0+j)) - um*un(iuc0+j) - um2*un(j2+j)
     1 - um3*un(j3+j) - um4*un(j4+j)
     2 - um5*un(j5+j) - um6*un(j6+j)
  1066 continue
  go to 1030
1015 continue
  j2 = iu(Lend+2) - 1 - idif
  j3 = iu(Lend+3) - 1 - idif
  j4 = iu(Lend+4) - 1 - idif
  j5 = iu(Lend+5) - 1 - idif
  um = un(iuc0)* di(Lend)
  um2 = un(j2) * di(Lend+1)
  um3 = un(j3) * di(Lend+2)
  um4 = un(j4) * di(Lend+3)
1016 continue
  go to 1030
1014 continue
  j2 = iu(Lend+2) - 1 - idif
  j3 = iu(Lend+3) - 1 - idif
  j4 = iu(Lend+4) - 1 - idif
  um = un(iuc0)* di(Lend)
  um2 = un(j2) * di(Lend+1)
  um3 = un(j3) * di(Lend+2)
  um4 = un(j4) * di(Lend+3)
CDIR$ IVDEP
  do 1017 j = 0, idif
     di(ju(iuc0+j)) = di(ju(iuc0+j)) - um*un(iuc0+j) - um2*un(j2+j)
     1 - um3*un(j3+j) - um4*un(j4+j)
     2 - um5*un(j5+j)
  1017 continue
  go to 1030
1013 continue
  j2 = iu(Lend+2) - 1 - idif
  j3 = iu(Lend+3) - 1 - idif
  um = un(iuc0)* di(Lend)
  um2 = un(j2) * di(Lend+1)
  um3 = un(j3) * di(Lend+2)
CDIR$ IVDEP
  do 1018 j = 0, idif
     di(ju(iuc0+j)) = di(ju(iuc0+j)) - um*un(iuc0+j) - um2*un(j2+j)
     1 - um3*un(j3+j)
  1018 continue
  go to 1030
1012 continue
  j2 = iu(Lend+2) - 1 - idif
  um = un(iuc0)* di(Lend)
  um2 = un(j2) * di(Lend+1)
CDIR$ IVDEP
  do 1019 j = 0, idif
     di(ju(iuc0+j)) = di(ju(iuc0+j)) - um*un(iuc0+j) - um2*un(j2+j)
  1019 continue
  go to 1030
1011 continue
  um = un(iuc0)* di(Lend)
CDIR$ IVDEP
  do 1020 j = 0, idif
     di(ju(iuc0+j)) = di(ju(iuc0+j)) - um*un(iuc0+j)
  1020 continue
  go to 1030
1030 continue
  if ( iuc00.eq.iud ) go to 80
  j = ju(iuc00 + 1)
  ip(L) = ip(j)
  ip(j) = L
  80 if ( LN.ne.0 ) go to 50
  um = 1.00/di(1)
  ipf=icpf+1
  if ( iub.lt.iua ) go to 120
CDIR$ IVDEP
  do 100 j = iua, iub
  un(j) = di(ju(j))*um
  if(isupd(1).eq.0) go to 130
  j = ju(iua)
  ip(1) = ip(j)
  100
ip(j) = i
120  iup(i) = iua
130  continue
1900  do 1900  j = 1,n
1900  return
end

subroutine supnode(n, iu, ju, isupd)
integer iu(1), ju(1), isupd(1)
do 10  i = 1,n
10  isupd(i) = 1
if(n.gt.0)  return
do 100  i = 1,n-1
ilen = iu(i+1) - iu(i)
isum = 0
if(isupd(i) .eq. 0)  go to 100
i0 = i+1
k0 = iu(i)
101  continue
k1 = iu(i0)
if(ilen.ne.iu(i0+1)-iu(i0)+i0-1)  go to 100
do 200  jj = k0+i0-1,k0+ilen-1
if(ju(jj).ne.ju(k1))  go to 100
k1 = k1 + 1
200  continue
isum = isum + 1
isupd(i) = isupd(i) + 1
isupd(i+isum) = 0
i0 = i0 + 1
go to 101
100  continue
return
end

*******
subroutine multspa(n, ia, ja, an, ad, b, c)
implicit real*8 (a-h,o-z)
real*8 an(1),ad(1),b(1),c(1)
integer ia(1),ja(1)
do 20  i = 1,n
20  c(i) = ad(i)*b(i)
do 30  i = 1,n
iaa = ia(i)
lab = ia(i+1) - 1
if( lab.lt.iaa )  go to 30
u = c(i)
z = b(i)
CDIR$ IVDEP
do 20  k = iaa, iab
j = ja(k)
u = u +an(k) * b(j)
20  c(j) = c(j) + an(k) * z
c(i) = u
30  continue
return
end

**
subroutine numfa0 (n, ia, ja, an, iu, ju, di, un)
implicit real*8 (a-h,o-z)
real*8 an(1),un(1),di(1)
integer ia(1),ja(1),iu(1),ju(1)
do 1000  i = 1,n
ih = i + 1
iua = iu(i)
iub = iu(ih) - 1
if ( iub.lt.iua )  go to 1000
CDIR$ IVDEP
do 20  j = iua, iub
di(ju(j)) = 0.0d0
iaa = ia(i)
lab = ia(ih) - 1
if ( lab.lt.iaa )  go to 1000
CDIR$ IVDEP
do 30  j = iaa, iab
di[ja(j)] = an(j)
CDIR$ IVDEP
do 100  j = iua,iub
1000  continue
return
end
SUBROUTINE NUMFAL(N, IA, JA, AD, AN, IU, JU, UN, IP, IUP, ISUPD, IOPF)
  IMPLICIT REAL*8(A-H,O-Z)
  DIMENSION IA(*), JA(*), AD(*), AN(*), IU(*), JU(*), UN(*), IP(*), IUP(*), ISUPD(*)

  C...... purpose: numerical factorization
  C  pp. 265-267 of text book, CE 795/895
  C input:  ia, ja, an, ad  given matrix A in RR(U)U.
  C     iu, ju  structure of resulting matrix U in
  C     n  order of matrices A and U.
  C output:  un  numerical values of the nonzeros of
  C     di  inverse of the diagonal matrix D.
  C working space:  ip  of dimension N. Chained lists of rows
  C     iup  of dimension N. Auxiliary pointers to
  C     di  is used as the expanded accumulator.

  DO 10 J=1,N
  10 IP(J)=0
  C...... begin of 1-st (nested) loop: outer-most loop, for each i-th row
  DO 130 I=1,N
    IH=I+1
    IUA=IU(I)
    IUB=IU(IH)-1
    IF(IUB.LT.IUA)GO TO 40
    DO 20 J=IUA, IUB
      20 DI(JU(J))=0.
      IAA=IA(I)
      IAB=IA(IH)-1
      IF(IAB.LT.IAA) GO TO 40
      DO 30 J=IAA, IAB
        30 DI(JA(J))=AN(J)
      40 mI {I) =AD(I)
      LAST= IP(I)
      IF(LAST.EQ.0)GO TO 90
      LN=IP(LAST)
      C...... begin of 2-nd (nested) loop: considering all APPROPRIATED previous
      rows (any appropriated rows l---->i-1)
      50 L=LN
      LN=IP(L)
      IUC=IUP(L)
      IUD=IU(L+I)-1
      UM=UN(IUC)*DI(L)
      C...... begin of 3-rd (nested) inner-most loop: considering all APPROPRIATED
      columns (any columns i---->n)
      DO 60 J=IUC,IUD
        JJ=JU(J)
        60 DI(JJ)=DI(JJ)-UN(J)*UM
      J=JU(IUC)
      IF(JJ.EQ.0)GO TO 80
      J=IP(J)
      IF(JJ.EQ.0)GO TO 70
      IP(L)=IP(J)
      IP(J)=L
      GO TO 80
    70 IP(J)=L
    IP(L)=L
    C...... the following go to statement is equivalent to 2-nd nested loop
    C...... for factorization
    C......
    80 IF(L.NE.LAST)GO TO 50
    C......
    90 DI(I)=1.0/DI(I)
    IF(IUB.LT.IUA)GO TO 120
    DO 100 J=IUA, IUB
    100 UN(J)=DI(JU(J))
    J=IU(J)
    IF(JJ.EQ.0)GO TO 110
    IP(J)=IP(JJ)
    IP(JJ)=I
    GO TO 120
    110 IP(J)=I
    IP(I)=I
subroutine numfa2(n, ia, ja, ad, an, iu, ju, di, un, ip, iup, isupd, iopf)
   c...... purpose: numerical factorization
   c this portion of numerical factorization has unrolling level 2
   c Modifications March 30,1995
   implicit real*8(a-h,o-z)
dimension ia(*),ja(*),ad(*),an(*),iu(*),ju(*),di(*),un(*)
dimension ip(*),iup(*),isupd(*)
   DEFINITIONS
   input: ia,_a,an,ad given matrix A in RR(U)U.
iu, 3u structure of resulting matrix U in RR(U)O.
n order of matrices A and U.
output: un numerical values of the nonzeros of
       matrix U in RR(U)O.
di inverse of the diagonal matrix D.
working space: ip of dimension N. Chained lists of rows
       associated with each column.
iup of dimension N. Auxiliary pointers to
       portions of rows.
di is used as the expanded accumulator.
   initialisation du vecteur IP, IP indique si il y a des lignes qui modifie la ligne qu'on factorise
   DO 10 J=1,N
   10 IP(J)=0
   c...... Begin of of 1-st (nested) loop: outer-most loop, for each i-th row
   DO 130 I=1,N
   130 CONTINUE
   return
   end

initialisation de la portion du working array DI necesssaire
DO 20 J=IUA,IUB
   20 DI(JU(J))=0.
   IAA=IA(I)
   IAB=IA(IH)-1
   commenter la ligne suivante
   IF(IAB.LT.IAA) GO TO 40
   copier AN dans working array DI et aussi AD dans DI : A--->U
   DO 30 J=IAA,IAB
   30 DI(JA(J))=AN(J)
   40 DI(I)=AD(I)
   voir si il y a une ligne au dessus qui va modifier la ligne I
   Au depart IP est initialise a )
   LAST=IP(I)
   IF(LAST.EQ.0) GO TO 90
   LN=IP(LAST)
   let LN be the iHEAD of the supernode
   celiminer LN=IP(I)
   c...... begin of 2-nd (nested) loop: consider all APPROPRIATED previous
   c...... rows (any appropriated rows 1--->i-l) which contribute to modify I
   Debut de la partie vecteur
   50 L=LN
   LN=IP(L)
   loop=2
   m= min(i-1,isupd(I))
iend=(m/loop)*loop
   isbegin=1
   isend=isend+1
   keep a copy of IUCL and IUDI of L pour la construction de IP
   IUCL=IUP(isbegin)
iUDI=IU(isbegin+1)-1
IUCI = IUP(is)
UM1 = UN(IUCI) * DI(is)
UM2 = UN(IUC2) * DI(is + 1)

DO 60 J = IUCI, IUD1
   JJ = JU(J)
   60 DI(JJ) = DI(JJ) - UN(J) * UM1 - UN(IUC2 - IUC1 + J) * UM2
   UN(IUC1) = UM1
   un(IUC2) = um2

IUP(is) = IUCI + I
iup(is + 1) = iuc2 + 1

DO 62 J = IUCI, IUDI
   JJ = JU(J)
   62 DI(JJ) = DI(JJ) - UN(J) * UM1
   UN(IUCI) = UM1

IP je ne le fait que pour tout le supernode

IF(IUCL.EQ.IUDL) GO TO 80
if(isupd(i) .eq.0) go to 80
J = JU(IUCL + 1)
JJ = IP(J)
IF(JJ .EQ. 0) GO TO 70
IP(L) = IP(JJ)
IP(JJ) = L
GO TO 80
70 IP(J) = L
IP(L) = L
GO TO 80

IUP(I) = IUA
130 CONTINUE

return

end

subroutine numfa8(n, ia, ja, ad, an, iu, ju, di, un, ip, iup, isupd, iopf)
implicit real*8(a-h,o-z)
dimension ia(*), ja(*), ad(*), an(*), iu(*), ju(*), di(*), un(*), isupd(*)
This subroutine is called by spasolver.f
this portion of numerical factorization has unrolling level 8
Modifications April 24, 1995
modification pour la rapidite october 5, 1995

DEFINITIONS
-----------------
input: ia(n+1), ja(ncoeff), an(ncoeff), ad(n): given matrix A in RR(U)
        iu(n+1), ju(ncoeff2): structure of resulting matrix U in RR(UO).
          n   order of matrices A and U.
output:  un (ncoeff2) numerical values of the nonzeros of
         matrix U in RR(UO).
         di(n) inverse of the diagonal matrix D.
working space: ip of dimension N. Chained lists of rows
               associated with each column is different from the one in symbolic
               iup of dimension N. Auxiliary pointers to portions of rows.
         di(n) is used as the expanded accumulator.

DO 10 J=1,N
   IP(J)=0
10   DO 130 I=1,N
    IH=I+1
    IUA=IU(I)
    IUB=IU(IH)-I
    CV IF(IUB.LT.IUA)GO TO 40
    CDIRS IVDEP
    DO 20 J=IUA,IUB
    DI(J)=0.00
    IAA=IA(I)
    IAB=IA(IH)-I
    CV IF(IAB.LT.IAA)GO TO 40
    CDIRS IVDEP
    DO 30 J=IAA,IAB
    40  DI(J)=AN(J)
    LAST=IP(I)
    IF(LAST.EQ.0)GO TO 90
    LN=IP(LAST)
    DO 50 L=LN
       LN=IP(L)
       m= min(l-1,isupd(l))
       iend=(m/8)*8
       CDIRS IVDEP
       DO 60 IS=ISBEGIN,ISEND,8
          70  DO 68 J=IS+1,IS+8
             JJ=JU(J)
             DI(JJ)=DI(JJ)-UN(J)*UM1
             UM1=UN(IUCL)*DI(IS)
             UM2=UN(IUC2)*DI(IS+1)
             UM3=UN(IUC3)*DI(IS+2)
             UM4=UN(IUC4)*DI(IS+3)
             UM5=UN(IUC5)*DI(IS+4)
             UM6=UN(IUC6)*DI(IS+5)
             UM7=UN(IUC7)*DI(IS+6)
             UM8=UN(IUC8)*DI(IS+7)
             CDIRS IVDEP
             DO 68 J=IUC1,IUCL
                JJ=JU(J)
                DI(JJ)=DI(JJ)-UM1
               UM1=UN(IUCL)*DI(IS)
             CDIRS IVDEP
             DO 68 J=IUC1,IUCL
                JJ=JU(J)
                DI(JJ)=DI(JJ)-UM1
               UM1=UN(IUCL)*DI(IS)
+ -un(iuc3-iucl+j)*um3-un(iuc4-iucl+j)*um4
+ -un(iuc5-iucl+j)*um5-un(iuc6-iucl+j)*um6
+ -un(iuc7-iucl+j)*um7-un(iuc8-iucl+j)*um8

UN(IUCL)=UM1
un(iuc2)=um2
un(iuc3)=um3
un(iuc4)=um4
un(iuc5)=um5
un(iuc6)=um6
un(iuc7)=um7
un(iuc8)=um8

iucl=iu(is+9)-length
iudl=iucl+length-1

endo
c pour loop of level 7, 6, 5, 4, 3, 2, 1
iloop=m-iend
if (iloop.eq.0) go to 77

go to (1, 2, 3, 4, 5, 6, 7) iloop

go to 77
c@@@@@@@@@@@@@@@@

1 is=isend+1
UM1=UN(IUCL)*DI(is)
CDIR$ IVDEP
DO 61 J=IUICL, IUDL
JJ=JU(J)

61 DI(JJ)=DI(JJ)-UN(J)*UM1
UN(IUCL)=UM1
un(IUCL)=um2
un(iuc2)=um2
go to 77
c@@@@@@@@@@@@@@@@

2 is=isend+1
IUC2=IU(is+2)-length
UM1=UN(IUCL)*DI(is)
UM2=UN(IUC2)*DI(is+1)

CDIR$ IVDEP
DO 62 J=IUICL, IUDL
JJ=JU(J)

62 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucl+j)*um2
UN(IUCL)=UM1
un(iuc2)=um2
un(iuc3)=um3
go to 77
c@@@@@@@@@@@@@@@@

3 is=isend+1
IUC2=IU(is+2)-length
IUC3=IU(is+3)-length
UM1=UN(IUCL)*DI(is)
UM2=UN(IUC2)*DI(is+1)
UM3=UN(IUC3)*DI(is+2)

CDIR$ IVDEP
DO 63 J=IUICL, IUDL
JJ=JU(J)

63 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucl+j)*um2
+ -un(iuc3-iucl+j)*um3

UN(IUCL)=UM1
un(iuc2)=um2
un(iuc3)=um3
go to 77
c@@@@@@@@@@@@@@@@

4 is=isend+1
IUC2=IU(is+2)-length
IUC3=IU(is+3)-length
IUC4=IU(is+4)-length
UM1=UN(IUCL)*DI(is)
UM2=UN(IUC2)*DI(is+1)
UM3=UN(IUC3)*DI(is+2)
UM4=UN(IUC4)*DI(is+3)

CDIR$ IVDEP
DO 64 J=IUICL, IUDL
JJ=JU(J)

64 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucl+j)*um2
+ -un(iuc3-iucl+j)*um3-un(iuc4-iucl+j)*um4

UN(IUCL)=UM1
un(iuc2)=um2
un(iuc3)=um3
un(iuc4)=um4
go to 77
c@@@@@@@@@@@@@@@@

5 is=isend+1
IUC2=IU(is+2)-length
IUC3=IU(is+3) -length
IUC4=IU(is+4) -length
IUC5=IU(is+5) -length
UM1=UN(IUC1)*DI(is)
UM2=UN(IUC2)*DI(is+1)
UM3=UN(IUC3)*DI(is+2)
UM4=UN(IUC4)*DI(is+3)
UM5=UN(IUC5)*DI(is+4)

CDIR$ IVDEP
DO 65 J=IUC1, IUDL
JJ=JU(J)
65 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucl+j)*um2
   + -un(iuc3-iucl+j)*um3-un(iuc4-iucl+j)*um4
   + -un(iuc5-iucl+j)*um5
   UN(IUCL)=UM1
   un(iuc2)=um2
   un(iuc3)=um3
   un(iuc4)=um4
   un(iuc5)=um5
   go to 77

CDIR$ IVDEP
DO 66 J=IUC1, IUDL
JJ=JU(J)
66 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucl+j)*um2
   + -un(iuc3-iucl+j)*um3-un(iuc4-iucl+j)*um4
   + -un(iuc5-iucl+j)*um5-un(iuc6-iucl+j)*um6
   UN(IUCL)=UM1
   un(iuc2)=um2
   un(iuc3)=um3
   un(iuc4)=um4
   un(iuc5)=um5
   un(iuc6)=um6
   go to 77

CDIR$ IVDEP
DO 67 J=IUC1, IUDL
JJ=JU(J)
67 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucl+j)*um2
   + -un(iuc3-iucl+j)*um3-un(iuc4-iucl+j)*um4
   + -un(iuc5-iucl+j)*um5-un(iuc6-iucl+j)*um6
   + -un(iuc7-iucl+j)*um7
   UN(IUCL)=UM1
   un(iuc2)=um2
   un(iuc3)=um3
   un(iuc4)=um4
   un(iuc5)=um5
   un(iuc6)=um6
   un(iuc7)=um7
   go to 77

77 continue
   if(iucl.eq.iudl) go to 80
   j=jui(iucl+l)
   go to 77

JJ=IP(J)
IF(JJ.EQ.0)GO TO 70
IP(L)=IP(JJ)
IP(JJ)=L
GO TO 80
70  IP(J)=L
IP(J)=L
80  IF(L.NE.LAST)GO TO 50
90  DI(I)=1.00/DI(I)
IF(JUB.LT.IUA)GO TO 120
CDIR$ IVDEP
DO 100 J=IUA,IUB
  100 UN(J)=DI(JU(J))
     IF(isupd(i).eq.0) go to 130
     J=JU(IUA)
     JJ=IP(J)
     IF(JJ.EQ.0)GO TO ii0
     IP(I) =IP(JJ)
     IP(JJ) =I
     GO TO 120
     ii0 IP(J) =I
     IP(I) =I
     120 IUP(I)=IUA
     130 CONTINUE
RETURN
END

c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subroutine reord(n,ncoff,nreord,mtota,mtoti,a,iq,neig)
real*8 a(1)
integer IQ(1)
if(2*ncoff.gt.mtota) then
write(*,*)'REORD: increase MTOTA to: 2*ncoff = ',2*ncoff
stop
endif
1 3*ncoff+7*n+5.gt.mtoti)
write(*,*)'REORD: increase MTOTI to: 3*ncoff+7*n+5 = '
1
if(nreord.ne.0) then
  call cpu(time0)
call iread0(n,ncoff,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),
  iq(4*n+3),iq(5*n+4+ncoff))
call cpu(time1)
rewind(18)
write(18) (iq(i),i=5*n+4+ncoff,5*n+3+3*ncoff)
call genmmd(n,iq(4*n+3),iq(5*n+4+ncoff),iq(1+n),
  iq(2*n+1),iq(3*n+2),iq(4*n+3),iq(5*n+4+ncoff),iq(5*n+4))
call copyk(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+3),
  iq(5*n+4+ncoff),iq(5*n+4),a(1),a(1+ncoff),a(2*n+1),a(3*n+1),neig)
call copydb(n,iq(1+n),a(1),a(1+n),a(2*n+1),a(3*n+1),neig)
call cputime(time2)
call getnewk(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+3),
  iq(5*n+4+ncoff),iq(5*n+4),a(1),a(1+ncoff),a(2*n+1),a(3*n+1),neig)
call copyk(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+3),
  iq(5*n+4+ncoff),a(1),a(1+ncoff),a(2*n+1),a(3*n+1),neig)
call cpu(time0)
lwriten time0=cpu(time0),time1=cpu(time1),time2=cpu(time2),time0-time2
endif
return
END
C********************************************************************
subroutine iread0(n,ncoff,ia,icol,xls,ls,xadj,ja,adjncy)
integer adjncy(1),ia(1),icol(1),xls(1),xadj(1),ja(1)
rewind(15)
read(15)(ia(i),i=1,n)
rewind(11)
read(11)(ja(i),i=1,ncoff)
cwrite(*,*) 'IREAD0 IA =', (ia(i),i=1,10)
cwrite(*,*) 'IREAD0 JA =', (ja(i),i=1,10)
do 10 i=1,n
icol(i) = 0
Is(i) = 0
xls(i) = 0
xadj(i) = 0
xadj(i+n) = 0
do 11 i = 1,2*ncoff
adjncy(l) = 0
do 12 i = 1,n-1
do 13 j = ia(i),ia(i+1)-1
icol(ja(j)) = icol(ja(j)) + 1
13 continue
icolsum = 0
xadj(i) = 1
ls(i) = icol(1)
do 14 i = 2,n
xadj(i) = ia(i) + ls(i-1)
ls(i) = ls(i-1) + icol(i)
xadj(i+n) = xadj(n) + icol(n)
c write(*,*) 'xadj(i+n) = ',xadj(i+n)
do 15 i = 1,n-1
iadjo = xadj(i) + icol(i) - ia(i)
CDIR$ IVDEP
do 16 j = ia(i),ia(i+1) - 1
jj = ja(j)
adjncy(xadjin+o+j) = jj
adjncy(xadj(jj)+xls(jj)) = i
xls(jj) = xls(jj) + 1
16 continue
15 continue
return
doesubroutine getnewk(n, ia, perm, iu, iup,xadj,adjncy, ju)
integer ia(1),perm(1),iup(1),xadj(1),adjncy(1),ju(1),iu(1)
do 1 i = 1,n
iup(perm(i)) = i
iu(1) = 1
icountt = 0
do 10 i = 1,n
i0 = perm(i)
icount = 0
do 20 j = xadj(i0), xadj(i0+1) - 1
jj = iup(adjncy(j))
if(jj .le. i) go to 20
ju(icountt+1) = jj
icountt = icountt + 1
icount = icount + 1
20 continue
iu(i+1) = iu(i) + icount
10 continue
call transa(n,n,iu,ju,iup,adjncy)
return
doesubroutine copyk(n,ia,perm,ju,ja,un,an,ncoff)
integer ia(1),i(1),ja(1),ju(1),perm(1)
real*8 an(1),un(1)
rewind(11)
read(11)(ja(i),i=1,ncoff)
rewind(12)
read(12)(an(i),i=1,ncoff)
rewind(15)
read(15)(ia(i),i=1,n)
do 200 I = 1,n-1
i0 = perm(I)
do 220 J = iu(I),i(I+1) - 1
J0 = perm(ju(J))
i0 = 10
i00 = j0
if(j0.LT.i0) then
i0 = j0
i00 = 10
endif
CDIR$ IVDEP
do 230 jj = ia(i0),ia(i0+1)-1
if(ja(jj).NE.i00) go to 230
45
un(J) = an(JJ)
go to 220
continue
continue
continue
rewind(ll)
write(ll)(ju(i),i=1,ncoff)
rewind(12)
write(12)(un(i),i=1,ncoff)
rewind(15)
write(15)(iu(i),i=1,n+1)

write(*,*)'PERM =', (perm(i),i=1,10)
write(*,*)'IA =', (IA(I),i=1,10)
write(*,*)'JA =', (JA(i),i=1,10)
write(*,*)'IU =', (IU(I),i=1,10)
write(*,*)'JU =', (JU(i),i=1,10)
write(*,*)'AN =', (an(i),i=1,10)
write(*,*)'UN =', (un(i),i=1,10)

return
end

subroutine copydb(n,perm,diag,b,diag0,b0,neig)
integer perm(1)
real*8 diag(1),b(1),diag0(1),b0(1)
rewind(14)
read(14) (b0(i),i=1,n)
rewind(13)
read(13) (diag0(i),i=1,n)
do i00 i = 1,n
I0 = perm(i)
diag(i) = diag0(I0)
b(i) = b0(I0)
100 continue
rewind(13)
write(13)(diag(i),i=1,n)
rewind(14)
write(14)(b(i),i=1,n)
rewind(18)
write(18)(perm(i),i=1,n)
if(neig.gt.0) then
rewind(10)
read(10)(diag0(i),i=1,n)
do I = 1,n
I0 = perm(i)
diag(i) = diag0(I0)
enddo
rewind(10)
write(10)(diag(i),i=1,n)
endif
return
end

subroutine transa9(n,m, ia,ja,iat,jat)
implicit real*8 (a-h,o-z)
integer ia(1), ja(i), iat(1), jat(i)
mh = m + 1
nh = n + 1
do 10 i = 2, mh
iat(i) = 0
iab = ia(nh) + 1
do 21 jj = 1, n
CDIR$ IVDEP
do 20 i = ia(jj),ia(jj+1)-1
j = ja(i) + 2
iat(j) = iat(j) + 1
20 continue
iat(1) = 1
iat(2) = 1
if ( m.eq.1 ) go to 40
do 30 i = 3, mh
30   iat(i) = iat(i) + iat(i-1)
40   do 60 i = 1, n
40   iaa = ia(i)
40   iab = ia(i+1) - 1
40   if ( iab.lt.iaa ) go to 60
40       CDIR$ IVDEP
40       do 50 jp = iaa, iab
40         j = ja(jp) + 1
40         k = iat(j)
40         iat(k) = i
40       50     iat(j) = iat(j) + 1
40       continue
40       call tran2(n,m,iat,jat,ia,ja)
40       return
40       end
40       subroutine tran9(n,m,ia,ja,iat,jat)
40       implicit real*8(a-h,o-z)
40       integer ja(1),i1,ia(1),iat(1),jat(1)
40       mh = m + 1
40       nh = n + 1
40       do 10 l = 2, mh
40       ia(l) = 0
40       iab = ia(nh) - 1
40       do 21 jj = i, n
40          CDIR$ IVDEP
40          do 20 = ia(jj), ja(jj)+1
40            iat(j) = iat(j) + 1
40          20     continue
40          21 continue
40       iat(1) = 1
40       iat(2) = 1
40       if ( m.eq.1 ) go to 40
40       do 30 i = 3, mh
40       iat(i) = iat(i) + iat(i-1)
40       40   do 60 i = 1, n
40       iaa = ia(i)
40       iab = ia(i+1) - 1
40       if ( iab.lt.iaa ) go to 60
40       CDIR$ IVDEP
40       do 50 jp = iaa, iab
40       j = ja(jp) + 1
40       k = iat(j)
40       iat(k) = i
40       50     iat(j) = iat(j) + 1
40       60 continue
40       return
end
SUBROUTINE GENMMD ( NEQNS, XADJ, ADJNCY, PERM, INVP,
1           DHEAD, QSIZE, LLIST, MARKER, 
1           NOFSUB, MAXCON, NTERMS )
1       INTEGER ADJNCY(1), DHEAD(1), INVP(1), LLIST(1), 
1           MARKER(1), PERM(1), QSIZE(1)
1       INTEGER XADJ(1)
1       INTEGER DELTA, EHEAD, I, MAXINT, MDEG, 
1           MDLMN, MDNODE, NEQNS, NEXTM, NOFSUB,
1           NUM, TAG
1      mqinup = 0
1      mqinLm = 0
1      timeup = 0.0
1      timeLm = 0.0
1       MAXINT = 100000000*1000000000
1       DELTA = 0
1       IF ( NEQNS .LE. 0 ) RETURN
1       NOFSUB = 0
1       CALL MMDINT ( NEQNS, XADJ, ADJNCY, DHEAD, INVP, PERM,
1           QSIZE, LLIST, MARKER )
1       NUM = 1
1       NEXTM = DHEAD(1)
100   CONTINUE
100   IF ( NEXTM .LE. 0 ) GO TO 200
100       MDNODE = NEXTM
100       NEXTM = INVP(MDNODE)
100       MARKER(MDNODE) = MAXINT
100       INVP(MDNODE) = - NUM
100       NUM = NUM + 1
100  200 CONTINUE
IF ( NUM .GT. NEQNS ) GO TO 1000
TAG = 1
DHEAD(1) = 0
MDEG = 2
300 CONTINUE
IF ( DHEAD(MDEG) .GT. 0 ) GO TO 400
MDEG = MDEG + 1
GO TO 300
400 CONTINUE
MDEG = MDEG + DELTA
EHEAD = 0
500 CONTINUE
MDNODE = DHEAD(MDEG)
IF ( MDNODE .GT. 0 ) GO TO 600
MDEG = MDEG + 1
IF ( MDEG .GT. MDEG ) GO TO 900
GO TO 500
600 CONTINUE
NEXTMD = INVP(MDNODE)
DHEAD(MDEG) = NEXTMD
IF ( NEXTMD .GT. 0 ) PERM(NEXTMD) = - MDEG
INVP(MDNODE) = - NUM
NOPSUB = NOPSUB + MDEG + QSIZE(MDNODE) - 2
IF ( NUM-QSIZE(MDNODE) .GT. NEQNS ) GO TO 1000
TAG = TAG + 1
IF ( TAG .LT. MAXINT ) GO TO 800
TAG = 1
DO 700 I = 1, NUM-1
700 IF ( MARKER(I) .LT. MAXINT ) MARKER(I) = 0
CONTINUE
800 CONTINUE
CALL MMDELM ( MDNODE, XADJ, ADJNCY, DHEAD, INVP,
1 PERM, QSIZE, LLIST, MARKER, MAXINT,
1 TAG )
900 CONTINUE
CALL MMDELM ( MDNODE, XADJ, ADJNCY, DHEAD, INVP,
1 PERM, QSIZE, LLIST, MARKER, MAXINT,
1 TAG )
GO TO 300
1000 CONTINUE
CALL MMDELM ( NEQNS, PERM, INVP, QSIZE )
RETURN
END
SUBROUTINE MMDELM ( MDNODE, XADJ, ADJNCY, DHEAD, INVP,
1 PERM, QSIZE, LLIST, MARKER, MAXINT,
1 TAG )
INTEGER XADJ(1)
INTEGER ADJNCY(1), DBAKW(1), DFORW(1), DHEAD(1),
1 LLIST(1), MARKER(1), QSIZE(1)
INTEGER DEG, DEG0, DELTA, EHEAD, ELMNT,
1 ENODE, FNODE, IQ2, ISTOP,
1 ISTOP, J, JSTRT, JSTOP, JSTRT, LINK,
1 MAXINT, MDEG, MDEG0, MTAG, NABOR,
1 NEQNS, NODE, Q2HEAD, QXHEAD, TAG
MDEG0 = MDEG + DELTA
ELMNT = EHEAD
100 CONTINUE
IF ( ELMNT .LE. 0 ) RETURN
MTAG = TAG + MDEG0
IF ( MTAG .LT. MAXINT ) GO TO 300
TAG = 1
DO 200 I = 1, NUM-1
200 IF ( MARKER(I) .LT. MAXINT ) MARKER(I) = 0
CONTINUE
MTAG = TAG + MDEG0
300 CONTINUE
Q2HEAD = 0
QXHEAD = 0
DEG0 = 0
LINK = ELMNT
400 CONTINUE
ISTRT = XADJ(LINK)
ISTOP = XADJ(LINK+1) - 1
DO 700 I = ISTRT, ISTOP
700 ENODE = ADJNCY(I)
48
LINK = - ENODE
IF ( ENODE ) 400, 800, 500
CONTINUE
IF ( QSIZE(ENODE) .EQ. 0 ) GO TO 700
DEG0 = DEG0 + QSIZE(ENODE)
MARKER(ENODE) = MTAG
IF ( DBAKW(ENODE) .NE. 0 ) GO TO 700
IF ( DFORW(ENODE) .EQ. 2 ) GO TO 600
LLIST(ENODE) = QXHEAD
QXHEAD = ENODE
GO TO 700
CONTINUE
LLIST(ENODE) = Q2HEAD
Q2HEAD = ENODE

700 CONTINUE
CONTINUE = Q2HEAD
IQ2 = 1
CONTINUE
CONTINUE
IF ( ENODE .LE. 0 ) GO TO 1500
IF ( DBAKW(ENODE) .NE. 0 ) GO TO 2200
TAG = TAG + 1
DEG = DEG0
ISTRT = XADJ(ENODE)
NABOR = ADJNCY(ISTRT)
IF ( NABOR .EQ. ELMNT ) NABOR = ADJNCY(ISTRT+1)
LINK = NABOR
IF ( DFORW(NABOR) .LT. 0 ) GO TO 1000
DEG = DEG + QSIZE(NABOR)
GO TO 2100
CONTINUE
ISTRT = XADJ(LINK)
ISTOP = XADJ(LINK+1) - 1
DO 1400 I = ISTRT, ISTOP
NODE = ADJNCY(I)
LINK = - NODE
IF ( NODE .EQ. ENODE ) GO TO 1400
IF ( NODE ) 1000, 2100, 1100
CONTINUE
IF ( QSIZE(NODE) .EQ. 0 ) GO TO 1400
IF ( MARKER(NODE) .GE. TAG ) GO TO 1200
MARKER(NODE) = TAG
DEG = DEG + QSIZE(NODE)
GO TO 1400
CONTINUE
IF ( DBAKW(NODE) .NE. 0 ) GO TO 1400
IF ( DFORW(NODE) .NE. 2 ) GO TO 1300
QSIZE(NODE) = 0
MARKER(NODE) = MAXINT
DFORW(NODE) = - ENODE
DBAKW(NODE) = - MAXINT
GO TO 1400
CONTINUE
IF ( DBAKW(NODE) .EQ. 0 )
DBAKW(NODE) = - MAXINT
GO TO 2100
CONTINUE
CONTINUE = QXHEAD
IQ2 = 0
CONTINUE
IF ( ENODE .LE. 0 ) GO TO 2300
IF ( DBAKW(ENODE) .NE. 0 ) GO TO 2200
TAG = TAG + 1
DEG = DEG0
ISTRT = XADJ(ENODE)
ISTOP = XADJ(ENODE+1) - 1
DO 2000 I = ISTRT, ISTOP
NABOR = ADJNCY(I)
IF ( NABOR .EQ. 0 ) GO TO 2100
IF ( MARKER(NABOR) .GE. TAG ) GO TO 2000
MARKER(NABOR) = TAG
LINK = NABOR
IF ( DFORW(NABOR) .LT. 0 ) GO TO 1700
DEG = DEG + QSIZE(NABOR)
GO TO 2000
CONTINUE
JSTRT = XADJ(LINK)
JSTOP = XADJ(LINK+1) - 1
DO 1900 J = JSTRT, JSTOP
NODE = ADJNCY(J)
LINK = - NODE
IF ( NODE ) 1700, 2000, 1800
CONTINUE
IF ( MARKER(NODE) .GE. TAG )
1 GO TO 1900
MARKER(NODE) = TAG
DEG = DEG + QSIZE(NODE)
CONTINUE
1900 CONTINUE
DEG = DEG - QSIZE(ENODE) + 1
FNODE = DHEAD(DEG)
DFORW(ENODE) = FNODE
DBAKW(ENODE) = - DEG
IF ( FNODE .GT. 0 ) DBAKW(FNODE) = ENODE
DHEAD(DEG) = ENODE
IF ( DEG .LT. MDEG ) MDEG = DEG
CONTINUE
2000 CONTINUE
ENODE = LLIST(ENODE)
IF ( IQ2 .EQ. 1 ) GO TO 900
GO TO 1600
2100 CONTINUE
TAG = MTAG
ELMNT = LLIST(ELMNT)
GO TO 100
END
SUBROUTINE MMDELM ( MDNODE, XADJ, ADJNCY, DHEAD, DFORW,
1 DBAKW, QSIZE, LLIST, MARKER, MAXINT,
1 TAG )
INTEGER ADJNCY(1), DBAKW(1), DFORW(1), DHEAD(1),
1 LLIST(1), MARKER(1), QSIZE(1)
INTEGER XADJ(1)
INTEGER ELMT, I, ISTOP, ISTRT, J,
1 JSTOP, JSTRT, LINK, MAXINT, MDNODE,
1 NABOR, NODE, NPV, NQNBR, NXXNODE,
1 PVNODE, RLMT, RLOC, RNODE, TAG,
1 XQNBR
MARKER(MDNODE) = TAG
ISTRT = XADJ(MDNODE)
JSTOP = XADJ(MDNODE+1) - 1
ELMNT = 0
RLMT = ISTRT
RLOC = ISTRT
DO 200 I = ISTRT, ISTOP
NABOR = ADJNCY(I)
IF ( NABOR .EQ. 0 ) GO TO 300
IF ( MARKER(NABOR) .GE. TAG ) GO TO 200
MARKER(NABOR) = TAG
IF ( DFORW(NABOR) .LT. 0 ) GO TO 100
ADJNCY(RLOC) = NABOR
RLOC = RLOC + 1
GO TO 200
100 CONTINUE
LLIST(NABOR) = ELMT
ELMNT = NABOR
200 CONTINUE
CONTINUE
300 CONTINUE
IF ( ELMNT .LE. 0 ) GO TO 1000
ADJNCY(RLMT) = - ELMT
LINK = ELMT
CONTINUE
400 CONTINUE
JSTRT = XADJ(LINK)
JSTOP = XADJ(LINK+1) - 1
DO 800 J = JSTRT, JSTOP
NODE = ADJNCY(J)
LINK = - NODE
IF ( NODE ) 400, 900, 500
CONTINUE
IF ( MARKER(NODE) .GE. TAG ) OR.
1 DFORW(NODE) .LT. 0 ) GO TO 800
MARKER(NODE) = TAG
CONTINUE
500 CONTINUE
IF ( RLOC .LT. RLMT ) GO TO 700
LINK = - ADJNCY(RLMT)
RLOC = XADJ(LINK)
RLMT = XADJ(LINK+1) - 1
GO TO 600
600 CONTINUE
CONTINUE
ADJNCY(RLOC) = NODE
RLOC = RLOC + 1

CONTINUE

ELMNT = LLIST(ELMNT)
GO TO 300

CONTINUE
IF ( RLOC .LE. RLMT ) ADJNCY(RLOC) = 0
LINK = MDNODE
CONTINUE
ISTRT = XADJ(LINK)
ISTOP = XADJ(LINK+1) - 1
DO 1700 I = ISTRT, ISTOP
RNODE = ADJNCY(I)
LINK = - RNODE
IF ( RNODE ) 1100, 1800, 1200
1200 CONTINUE
PVNODE = DBAKW(RNODE)
IF ( PVNODE .EQ. 0 .OR.
PVNODE .EQ. (-MAXINT) ) GO TO 1300
NXNOD = DFORW(RNODE)
IF ( NXNODE .GT. 0 ) DBAKW(NXNODE) = PVNODE
IF ( NXNODE .LT. 0 ) DFORW(PVNODE) = NXNODE
NPV = - PVNODE
IF ( PVNODE .LT. 0 ) DHEAD(NPV) = NXNODE

CONTINUE JSTRT = XADJ(RNODE)
JSTOP = XADJ(RNODE+1) - 1
XQNB = JSTRT
DO 1400 J = JSTRT, JSTOP
NABOR = ADJNCY(J)
IF ( NABOR .EQ. 0 ) GO TO 1500
IF ( MARKER(NABOR) .GE. TAG ) GO TO 1400
ADJNCY(XQNB) = NABOR
XQNB = XQNB + 1

CONTINUE
CONTINUE
NQNBRS = XQNB - JSTRT
IF ( NQNBRS .GT. 0 ) GO TO 1600
QSIZE(MDNODE) = QSIZE(MDNODE) + QSIZE(RNODE)
QSIZE(RNODE) = 0
MARKER(RNODE) = MAXINT
DFORW(RNODE) = - MDNODE
DBAKW(RNODE) = - MAXINT
GO TO 1700

CONTINUE
DFORW(RNODE) = NQNBRS + 1
DBAKW(RNODE) = 0
ADJNCY(XQNB) = MDNODE
XQNB = XQNB + 1
IF ( XQNB .LE. JSTOP ) ADJNCY(XQNB) = 0

CONTINUE
1800 CONTINUE
RETURN

END
SUBROUTINE MMDINT ( NEQNS, XADJ, ADJNCY, DHEAD, DFORW, DBAKW, QSIZE, LLIST, MARKER 
1 INTEGER ADJNCY(1), DBAKW(1), DFORW(1), DHEAD(1), LLIST(1), MARKER(1), QSIZE(1)
1 INTEGER NODE, NDEG, NEQNS, NODE
DO 100 NODE = 1, NEQNS
DHEAD(NODE) = 0
QSIZE(NODE) = 1
MARKER(NODE) = 0
LLIST(NODE) = 0

100 CONTINUE
DO 200 NODE = 1, NEQNS
NDEG = XADJ(NODE+1) - XADJ(NODE) + 1
FNODE = DHEAD(NDEG)
DFORW(NODE) = FNODE
DHEAD(NDEG) = NODE
IF ( FNODE .GT. 0 ) DBAKW(FNODE) = NODE
DBAKW(NODE) = - NDEG

200 CONTINUE
RETURN

END
SUBROUTINE MMDNUM ( NEQNS, PERM, INVP, QSIZE 
1 INTEGER INVP(1), PERM(1), QSIZE(1)
INTEGER FATHER, NEQNS, NEXTF, NODE, NQSIZE, 
    NUM, ROOT
DO 100 NODE = 1, NEQNS 
    NQSIZE = QSIZE(NODE) 
    IF ( NQSIZE .LE. 0 ) PERM(NODE) = INVP(NODE) 
    IF ( NQSIZE .GT. 0 ) PERM(NODE) = - INVP(NODE) 
100 CONTINUE 
DO 500 NODE = 1, NEQNS 
    IF ( PERM(NODE) .GT. 0 ) GO TO 500 
    FATHER = NODE 
    CONTINUE 
    IF ( PERM(FATHER) .GT. 0 ) GO TO 300 
    FATHER = - PERM(FATHER) 
    GO TO 200 
300 CONTINUE 
    ROOT = FATHER 
    NUM = PERM(ROOT) + 1 
    INVP(NODE) = - NUM 
    FATHER = NODE 
    CONTINUE 
    NEXTF = - PERM(FATHER) 
    IF ( NEXTF .LE. 0 ) GO TO 500 
    PERM(FATHER) = - ROOT 
    FATHER = NEXTF 
    GO TO 400 
500 CONTINUE 
DO 600 NODE = 1, NEQNS 
    NUM = - INVP(NODE) 
    INVP(NODE) = NUM 
    PERM(Root) = NUM 
700 CONTINUE 
RETURN 
END

***********

LUMP MASS CASE + shift factor for the regular Lanczos

H. Runesha dec 22, 1997

***********

SUBROUTINE SP2LAN(N, LANMAX, NEIG, UN, DI, IU, JU, W, w1, w2, w4, 
STEM, EIG, ERR, VEC, ncoff, ncof2, dmass, Q, lump, ishift) 

IMPLICIT REAL*8 (A-H, O-Z)
REAL*8 W(1), VEC(LANMAX, LANMAX), EIG(1), W4(1), W2(1), Q(N, LANMAX)
REAL*8 TEM(1), ERR(1), W1(1), dmass(1), UN(1), DI(1)
INTEGER IU(1), JU(1)
COMMON /QIN/ QLAN, EPSOO
COMMON /SP2 COM/NBUF(4)
REWIND (10)
READ (10) (dmass(i), i=1, n)
iam=0
nodes=1
NBLO = 1
RTOL = 0.0000000001d0
NNNN= 4*NEIG
IF (NNNN.GE. LANMAX) NNNN=LANMAX-1
MN=0
DO 61 I=1, LANMAX 
61 SIG(I) = 0.0d0 
ERR(I) = 0.0d0 
EPS=GETEPS(IBETA, IT, IRND) 
EPSOO=EPS 
EPS=dsqrt (EPSOO) 
EPSb=EPS* 5.0d0 
wnor=0.0d0 
k=1
DO 14 I=1, n 
14 w(i)=1.0d0 
if (k.gt.100) k = 1
wnor=wnor+w(i)*w(i)
wnor=1.0d0/dsqrtn(wnor)
DO 15 I=1, N 
15 W1(i)=w(i)*wnor 
W1(i)=0.0d0 
15 CONTINUE 
DO 9901 I=1, n 
9901 w2(i)=dmass(i)*w(i)
52
bet=0.0d0
bet=bet+w2(i)*w(i)
bet=dsqrt(bet)
W1(i)=w(i)/bet
Q(i,1)=W1(i)

continue
do 99013 I=1,n
w2(i)=dmass(i)*W1(i)
do 9901 w2(i)=dmass(i)*w(i)
bet2=ddot(N, W4, i, W2, i)
bet2=dsqrt(bet2)
c write(*,*) ' before REORTH: JJ = ',jj,bet2
call REORTH(N, W2, Q, EPS, LANMAX, JJ, BET2)
do 9903 i=1,n
w2(i)=dmass(i)*w4(i)
bet2=ddot(N, W4, i, W2, i)
bet2=dsqrt(bet2)
c write(*,*) ' before i00 : JJ, beta = ',jj,bet2
RETURN
END

SUBROUTINE REORTH(N, P, R, Q, EPS, LANMAX, JJ, BET2)
implicit real*8 (a-h,o-z)
REAL*8 P(1),R(1),Q(N,LANMAX)
REAL*8 HH(1)
DO 10 K=1,NJ-1
HH(K)=0.0d0
DO 20 I=1,N
HH(K)=HH(K)+P(I)*Q(I,K)
20 CONTINUE
10 CONTINUE
HNORM=HH(K)
DO 40 K=I,NJ-I
if(abs(hk(hk)).lt.1.0e-10) go to 40
DO 30 I=1,N
R(I)=R(I)-HH(K)*Q(I,K)
30 CONTINUE
40 CONTINUE
RETURN
END

SUBROUTINE JACOBIPi (N, NEIG, NJ, BET2, V, Q, VEC,
$ EIG, ERR, AN, HH, RTOL, LANMAX, DMASS,
$ VPI, NPI, NNNN, NSUC, ncoff, ncof2, ia, ja, ad, ishift)
implicit real*8 (a-h,o-z)
REAL*8 DMASS (1) , Q (N, LANMAX) , VEC (LANMAX, LANMAX)
$ , EIG (1), ERR (1), HH (1), VPI (1), V (1), AN (1), ad (1)
integer ia (1), ja (1)
common/sp2com/nbuf(4)
COMMON /QIN/ QLAN, EPS
COMMON /NQIN/ NQL
COMMON /QJA/ SUB (I000)
iam=0
nodes=1
NSUC=0
DO 1 I=1,LANMAX
VEC(I,J)=0.0d0
1 CONTINUE
HH(I)=EIG(I)
SUB(I)=ERR(I)
VEC(I,I)=1.0d0
DO 2 J=I,LANMAX
VEC (I, J) =0.0d0
2 CONTINUE
if (nodes.gt.1) call mp_sync(nbuf(4))
CALL TQL2 (LANMAX, NJ, EIG, ERR, VEC, IRR, EPS)
write(*,*) 'IN JACOBQ: TQL2 is done ! ! ! ! ',
DO 20 I=1,NJ
err(i)=0.0d0
do 21 j=1,nj
err(i)=err(i)+abs(vec(j,i))
21 CONTINUE
ERR(I)=err(i)*0.005d0*ABS(BET2*VEC(NJ, I)/EIG(I))
write(*,*)'I,err(i),vec(nj,i)=', i,err(i),vec(nj,i)
DO 30 I=1,NEIG
IF(ERR (I) .GT. 0.000001.AND.NJ.LT. (LANMAX-I)
30 CONTINUE
if (nodes.gt.1) call mp_sync(nbuf(4))
CALL VECTRA(N,NEIG, 0,VEC,Q, LANMAX,NJ,V)
write(*,*) 'IN JACOBQ: VECTRA is done ! ! ! ! ',
NSUC=1,successful
NSUC=1
DO 580 L=1,NEIG
call cputime(t01)
ninc=8*nodes
do L=1,neig
do i-n-nadd+1,n
q(i,L)=0.0d0
enddo
c enddo
c write(*,*)'NADD = ',nadd,n
endif
rewind(15)
read(15)(ia(i),i=1,1+n)
rewind(11)
read(11)(ja(i),i=1,ncoff)
rewind(12)
read(12)(an(i),i=1,ncoff)
rewind(13)
read(13)(ad(i),i=1,n)
c write(*,*)'***PI** AD= ', (ad(i),i=1,n)
DO 510 J=1,N
WRITE(*,*) (q(i,j),i=1,n)

DO 580 L=1,N
CALL MULTSPA(N,IA,JA,AN,AD,Q,V)
CALL MULMEIKO(N,...)
IF(NINC.GT.8) CALL MP_SYNC(NBUF(4))
CALL SP2MUL(IAM,NINC,N,icolg,maxa,stif,q(1,L),vpl,v)
CALL MULTSPA(N,IA,JA,AN,AD,Q(L,L),V)
WRITE(*,*) 'Q(L,L)= ',L,(Q(I,L),I=1,N)
NORMCHECK2
DO 590 I=I,N
VNORM=VNORM+V(I)*V(I)
WNORM=0.0D0
DO 2239 I=1,N
VPL(I)=DMASS(I)*Q(I,L)
CONTINUE
WRITE(*,*) 'I, VPL= ',I,(VPL(I),I=1,N)
RNORM=0.0D0
RT=1.0D0/EIG(L)
DO 600 I=1,N
ERM=MAX(ABS(V(I)),ABS(RT*VPL(I)))/RT
ERROR=ABS(V(I)-RT*VPL(I))/ERM
IF(ERROR.GT.0.95D0.AND.ERM.GT.0.00001D0) THEN
WRITE(*,*) 'I, VPL= ',I,(VPL(I),I=1,N)
ENDIF
V(I)=V(I)-RT*VPL(I)
CONTINUE
WRITE(*,*) 'HI= ',(HH(I),I=1,N)
CALL CPU_TIME(T02)
T02=T02-T01
WRITE(23,*)'*** K, EIG, HERTZ, ERROR, NORM *** IAM'
IF(IShift.EQ.0) THEN
DO 700 K=1,NEIG
HERTZ=1.0/(2.0*3.1415927*DSQRT(EIG(K)))
ERROR=1.0/(2.0*3.1415927*DSQRT(DABS(EIG(K))))
WRITE(23,701)K,1.0/EIG(K),HERTZ,ERROR(K),HH(K)
FORMAT(2X,15,2X,4E15.7)
CONTINUE
ELSE
DO 705 K=1,NEIG
EIG(K)=1./EIG(K)-FLOAT(ISHIFT)
HERTZ=DSQRT(EIG(K))/(2.0*3.1415927)
ERROR=DSQRT(DABS(EIG(K)))/(2.0*3.1415927)
WRITE(23,701)K,1.0/EIG(K),HERTZ,ERROR(K),HH(K)
CONTINUE
ENDIF
NSUC=1
IF(IAM.EQ.0) WRITE(23,*) 'JACOBIQ: Steps in IAM = ',NJ,IAM
RETURN
300 NNNN=MINT(LANMAX-1,NJ+3*(NEIG-I)+4)
NSUC=0
DO 304 I=1,LANMAX
EIG(I)=HH(I)
SUBROUTINE TQL2 (NM, N, D, E, Z, IERR, MACHEP)
implicit real*8 (a-h,o-z)
REAL* 8 MACHEP
REAL*8 D(NM), E(NM), Z(NM, NM)
C **** MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC
IQL0=ICOUNQ
IERR=0
C WRITE(*,*) '*** TQL2 BEGIN *******'
IF(N.EQ.1) GO TO 1001
CD 100 I=2, N
C 100 E(I-1)=E(I)
E(N)=E(N-1)
F=0.0d0
B=0.00
E(N)=0.0d0
DO 240 L=I, N-1
J=0
H=MACHEP*(ABS(D(L))+ABS(E(L)))
IF(B.LT.H) B=H
C ****** LOOK FOR SMALL SUB-SIAGONAL ELEMENT ***********
DO 110 M=L, N
IF(ABS(E(M)) .LE.B) GO TO 120
C**** E(N))IS ALWAYS ZERO,SO THERE IS NO EXIT THROUGH
C THE BOTTOM OF THE LOOP ***************
110 CONTINUE
120 IF(M.EQ.L) GO TO 220
130 IF(J.EQ.30) GO TO 1000
J=J+1
C ******* FORM SHIFT ********
LI=L+L-1
G=D(L)
P=(D(LI)-G)/(2.0d0*E(L))
R=DSQRT(P*P+1.0d0)
D(L)=E(L)/(P+SIGN(R, P))
H=G-D(L)
DO 140 I=LI, N
140 D(I)=D(I)-H
COUNQ=ICOUNQ+(N+I-L1)
F=F+H
C***** QL TRANSFORMATION *******
P=D(M)
C=I.0d0
S=0.0d0
MML=M- L
C**** FOR I=M-1 STEP -1 UNTIL L DO -- ********
DO 200 II=M-1, MML
I=M- II
G=C*E(I)
H=C*P
IF(ABS(P).LT.ABS(E(I))) GO TO 150
C=E(I)/P
R=DSQRT(C*C+1.0d0)
E(I+1)=S*P*R
S=C/R
C=I.0/R
GO TO 160
150 C=P/E(I)
R=DSQRT(C*C+1.0d0)
E(I+1)=S*E(I)*R
S=1.0d0/R
C=C*S
160 P=C*D(I)-S*G
D(I+1)=H+S*(C*G+S*D(I))
C******** FORM VECTOR *********
DO 180 K=1, N
H=Z(K, I)
Z(K, I+1)=S*Z(K, I)+C*H
Z(K, I)=C*Z(K, I)-S*H
180 CONTINUE
ICOUNQ=ICOUNQ+6*N
C 200 CONTINUE
**ORDER EIGENVALUES AND EIGENVECTORS  **********

DO 300 II=2,N
  I=II-1
  K=I
  P=D(I)
C
DO 260 J=II,N
  IF(D(J).LE.P) GO TO 260
  K=J
  P=D(J)
C
CONTINUE
C
IF (K.EQ.I) GO TO 300
  D(K)=D(I)
  D(I)=P
C
DO 280 J=1,N
  P=Z(J,I)
  Z(J,I)=Z(J,K)
  Z(J,K)=P
C
CONTINUE
C
GO TO 1001
C
** SET ERROR -- NO CONVERGENCE TO AN EIGENVALUE
C AFTER 30 ITERATIONS ***************

1000  IERR=L
1001  CONTINUE
  WRITE(23,*)'VECTRA: lanmax,nj,neig = ',lanmax,nj,neig
  DO 20 i = l,n
    do 30 j=1,neig
      R(j) =0.0d0
      do 40 k=1,nj
        R(j ) =R (j ) +q(i, k ) *vec(k, j)
        do 50 kk=1,neig
          q(i,kk)=R(kk)
        50  CONTINUE
        WRITE(23,*)'**** VECTRA END ,ilen,iseg= *****'
      30  continue
    40  continue
  20  CONTINUE
  WRITE(23,*)'TQL2 END,IERR=',IERR,';at ',N,'-th Lan step.'
  RETURN
END

FUNCTION GETEPS(IBET, IT, IRND)
  implicit real*8 (a-h,o-z)
  a = 1.0d0
  if(((a+1.0d0)-a)-1.0d0.eq.0.0d0) go to 10
  b=1.0d0
  if ((a+b)-a.eq.0.0d0) go to 20
  ibeta=int(real((a+b)-a))
  beta=float(ibeta)
  it=0
  b=1.0d0
  if(((b+1.0d0)-b)-1.0d0.eq.0.0d0) go to 30
  irnd=0
  betaml=beta-1.0d0
  if ((a+betaml)-a.ne.0.0d0) irnd=1
  betain=1.0d0/beta
  a=1.0d0
  do 40 i=1,irnd+3
a=a*betain
continue
if((1.0d0+a)-1.0d0.ne.0.0d0) go to 60
a=a*beta
go to 50
eps=a
if((ibeta.eq.2).or.(irnd.eq.0)) go to 70
a=(a*(1.0d0+a))/(1.0d0+1.0d0)
a=(1.0d0+a)-1.0d0.ne.0.0d0)eps=a
geteps=eps
return
dend
c*************** Block Lanczos Driver ********_____________
c SUBROUTINE SP2LAN (N, LANMAX, NEIG, UN, DI, IU, JU, W, W1, W2, W4,
c STEM, EIG, ERR, VEC, NCOFF, NCOF2, DMASS, Q)
c implicit real*8 (a-h, o-z)
c REAL*8 W(1), VEC((LANMAX, LANMAX), EIG(1), W4(1), W2(1), Q(N, LANMAX))
c REAL*8 TEM(1), ERR(1), W1(1), DMASs(1), UN(1), DI(1)
c INTEGER IU(1), JU(1)
c C subroutine blanmain(n, isize, iblock, neig, un, di, iu, ju, r, p, tem, dmass, 
1 beta, beta, eig, vec, alfa, q, t, b, am, ia, ja, lump, ncoff, ishift)
c implicit real*8 (a-h, o-z)
c real*8 T(isize, isize), un(1), di(1), am(1), eig(isize)
c real*8 dmass(1), q(n, isize+iblock), r(n, iblock), tem(n, iblock)
c real*8 alfa(iblock, iblock), beta(iblock, iblock), vec(isize, isize)
c real*8 b(isize, isize), beta(iblock, iblock), p(n, iblock)
c integer ia(1), ja(1), iu(1), ju(1)
c C
C***** This is the driver for block Lanczos eigensolver *****
C***** 1. Initialization:
C
C     do i = 1, isize
C     do j = 1, isize
C          T(j,i) = 0.0d0
C          B(j,i) = 0.0d0
C     enddo
C     endif
C     if(lump.eq.1) then
C        rewind(10)
C        read(10) (dmass(i), i=1,n)
C        write(23,*)'Lumped Mass is used ! '
C else
C        write(23,*)'Consistant Mass is used ! '
C        rewind(10)
C        read(10) (dmass(i), i=1,n)
C        rewind(17)
C        read(17) (am(i), i=1,ncoff)
C        rewima
C        read(15) (ia(i), i=1,1+n)
C        read(11)
C        read(11) (ja(i), i=1, ncoff)
C     endif
C     stop
C***** Do the shifting: K~ = K + ishift*M
C     lanmax=isize/iblock
C     write(23,*)'n, isize, iblock, ishift= ', n, isize, iblock, ishift
C
DO j = 1, iblock
  DO i = 1, n
    q(i, j) = 0.0d0
    r(i, j) = 0.0d0
    p(i, j) = 0.0d0
  ENDDO
ENDDO

xmul = float(((n+1)**3)/3)
xmul = n
xmul = dsqrt(3.0d0)/((xmul*dsqrt(xmul))
xmul = dsqrt(2.0d0)/float(n)
xmul = 1.0d0
write(23,*)'xmul = ',xmul
DO i = 1, n
  r(i, 1) = xmul
  r(i, 1) = float(i)*xmul
  IF(iblock.GE.2) THEN
    IF((i/2)*2.eq.i) THEN
      xmul = n-i+1
    ELSE
      xmul = i+n-1
    ENDIF
  ELSE
    xmul = n-i+1
  ENDIF
  r(i, 1) = xmul
  IF(iblock.eq.3) THEN
    r(i, 3) = 1.0d0 + sin(3.1415926*i/4)
  ENDIF
ENDIF
WRITE(*,*) 'r(i, 1) = ', i, r(i, 1)
R(n, 2) = 1.0d0
!
CALL STEPF(n, iblock, TEM, Q, R(1, 1), BETA, IA, JA, AM, DM, LUMP)
!
CALL STEPEI(n, iblock, P(1, 1), Q(1, 1), IA, JA, AM, DM, LUMP)
!
! Just for the 6x6 ex.
! Just for the 6x6 ex.
!
! Now, the main DO LOOP starts here: J = 1, 2, 3, .... <= LANMAX
!
DO J = 1, LANMAX
!
J01 = (J-1)*iblock+1
J02 = J*iblock+1
J03 = (J-2)*iblock+1
!
DO K = 1, iblock
  DO I = 1, iblock
    BETA(i, k) = BETA(i, k)
  ENDDO
ENDO
!
CALL STEPA(n, iblock, R(1, 1), P(1, 1), IU, JU, DI, UN)
!
IF(J.ne.1) CALL STEPB(n, iblock, R(1, 1), Q(1, J03), BETA)
!
CALL STEPC(n, iblock, P(1, 1), R(1, 1), ALFA)
!
IF(J.EQ.LANMAX) GO TO 1000
!
CALL STEPD(n, iblock, Q(1, J01), R(1, 1), ALFA)
DO K = 1, iblock
  WRITE(23,*)'J, ALFA(col.) = ', J, (ALFA(i, k), i=1, iblock)
ENDDO
!
CALL STEPEI(n, iblock, P(1, 1), Q(1, J02), IA, JA, AM, DM, LUMP)
!
CALL STEPF(n, iblock, TEM, Q, R(1, 1), BETA, IA, JA, AM, DM, LUMP)
!
CALL STEPEI(n, iblock, P(1, 1), Q(1, J02), IA, JA, AM, DM, LUMP)
1000 CONTINUE
!
CALL ASSEM(ISIZE, IBLOCK, T, ALFA, BETA, J)
!
ENDDO
!
CALL JACOBI(T, B, VEC, TEM(1, 1), TEM(1, 2), ISIZE, TOL, NSMAX, IFPR, ISIFT)
**C** write(23,*)'***** Max. Element in the K-th VEC vectors *****'
do L = 1, isize
qmax=0.0d0
iqmax=0
do i = 1, isize
if(dabs(VEC(i,L)).gt.qmax) then
  qmax=dabs(VEC(i,L))
i qmax = i
endif
endo
c write(23,*) 'K, ivmax,Vmax = ',L, iqmax,qmax
do L = l,isize
qmax=0.0d0
iqmax=0
do i = l,n
if(dabs(q(i,L)).gt.qmax) then
  qmax=dabs(q(i,L))
iqmax=i
endif
endo
c write(23,*) 'L, iqmax,Qmax = ' , L, iqmax, qmax
enddo
call VECTRA(N,NEIG,IBLOCK,VEC,Q,isize,isize,R(1,1))
c
C***** This subroutine finds: r that Kr = p in Step a. ******
do i = l,iblock
call fbe(n,iu,ju,di,un,p(l,i),r(l,i),iopfb)
endo
c returnend
C subroutine stepb(n,iblock,r,q,beta)
** This subroutine finds r(head)=r(bar) q*beta(transpo.) i/% Step b. ****
** beta stored as upper-trangular matrix
do i = l,iblock
  do j = i,iblock
    do k = l,n
      r(k,i)=r(k,i) - q(k,j) * beta(i,j)
    enddo
  enddo
enddo
c returnend
C subroutine stepc(n,iblock,p,r,alfa)
** This subroutine finds: alfa = P(transp.)*R  in Step c. *****
do j = 1,iblock
  alfa(i,j)=0.0d0
  do k = 1,n
    alfa(i,j)=alfa(i,j) + p(k,i) * r(k,j)
  enddo
  alfa(j,i) = alfa(i,j)
enddo
c return
C subroutine stepd(n, iblock, q, r, alfa)
  implicit real*8 (a-h,o-z)
  real*8 r(n, iblock), q(n, iblock), alfa(iblock, iblock)
C**** This subroutine finds: R = R(head) - Q*alfa in Step d. ******
do j = 1, iblock
  do i = 1, iblock
    do k = 1, n
      r(k, j) = r(k, j) - q(k, i) * alfa(i, j)
    enddo
  enddo
enddo
return
end

C subroutine stepe(n, iblock, p, r, ia, ja, am, dmass, lump)
  implicit real*8 (a-h,o-z)
  real*8 r(n, iblock), p(n, iblock), am(1), dmass(1)
  integer ia(1), ja(1)
C**** This subroutine finds: P(bar) = DMASS*R in Step e. ******
  if(lump .eq. 1) then
    C****** lumped Mass matrix ************
do j = 1, iblock
      do i = 1, n
        p(i, j) = dmass(i) * r(i, j)
      enddo
  enddo
  else
    C****** consistant Mass matrix ************
do j = 1, iblock
      call multspa(n, ia, ja, am, dmass, r(1, j), p(1, j))
  enddo
endif
return
end

C subroutine stepf(n, iblock, tem, q, r, beta, ia, ja, am, dmass, lump, JB)
  implicit real*8 (a-h,o-z)
  real*8 q(n, iblock), r(n, iblock), beta(iblock, iblock)
  integer ia(1), ja(1)
C************* This is for Steps f, g and h. ********************
C************* This is similar to that of page 5 of 5. ********************
C************* iblock = block size; n = size of K & M; ********************
C************* Looking for: q*beta = r, where only r is known !******
C************* and qMq = delta ********************
c write(*,*)'in stepf: LUMP = ', lump
  j0 = (JB-1)*iblock
  do i = 1, iblock
    do j = 1, n
      tem(j, i) = 0.0d0
      q(j, i+j0) = r(j, i)
    enddo
  enddo
  do j = 1, iblock
    beta(j, i) = 0.0d0
  enddo
  if(lump .eq. 1) then
    C****** lumped Mass matrix ************
do j = 1, iblock
      do i = 1, n
        tem(i, j) = dmass(i) * r(i, j)
      enddo
  enddo
  else
    C****** consistant Mass matrix ************
do j = 1, iblock
      call multspa(n, ia, ja, am, dmass, r(1, j), tem(1, j))
**enddo**

**endf**

****** now: tem = Mr already !**

**write(*)'stepf: finished tem=Mr !'**

**do k = 1, iblock**

**C**

**do j = 1, k-1**

**C***********

**do i = l,n**

beta(j,k)=beta(j,k) + q(i,j+j0)*tem(i,k)

**enddo**

**C**

**beta(k,j) = 0.0d0**

**write(23,*)'beta(j,k) = ',beta(j,k)**

**do i = l,n**

**write(*,*)'q(i,k),q(i,j) = ',q(i,k+j0),q(i,j+j0)**

q(i,k+j0)= q(i,k+j0) - beta(j,k)*q(i,j+j0)

**enddo**

**C***********

**enddo**

**C**

**if(lump.eq.l) then**

**C***** lumped Mass matrix *************

**do i = l,n**

tem(i,k) = dmass(i) * q(i,k+j0)

**enddo**

**C**

**C***** consistant Mass matrix *************

**write(*,*) f:dmass=', (dmass(i),i=l,n)**

**write(*,*) f:am = ',(am(i),i=1,6)**

**write(*,*) f:q = ',(q(i,k+j0),i=1,4)**

**call multspa(n,ia,ja,am,dmass,q(_,k+j0),tem(l,k))**

**write(*,*) f:tem=Mq=', (tem(i,k),i=1,4)**

**endif**

**beta(k,k) = 0.0d0**

**do i = l,n**

beta(k,k) = beta(k,k) + q(i,k+j0)*tem(i,k)

**enddo**

**C**

**beta(k,k) -- dsqrt (beta (k, k))**

**xmult=1.0d0/beta (k,k)**

**write(23,*)'beta (k, k) = ',k,beta (k,k)**

**do i = l,n**

q(i,k+j0) = xmult * q(i,k+j0)

**enddo**

**write(*,*)'stepf: finished beta !'**

**C**

**C*** The following is for REORTHOgonalization/normalization ******

**C***** JB is the block # of the current block. *****

**do m = (JB-l)*iblock+l, JB*iblock**

**if(n.gt.l) goto 1995**

**C**** DO: temp(i,1) = M*q(1,m)**

**C======**

**if(lump.eq.l) then**

**do j = 1,n**

tem(j,1) = dmass(j) * q(j,m)

**enddo**

**else**

**call multspa(n,ia,ja,am,dmass,q(1,m),tem(1,1))**

**endif**

**write(*,*)'stepf: finished tem=Mq !  m = ',m**

**C======**

**do i = 1,m-1**

tem(i,2) = 0.0d0

**do j = 1,n**

tem(i,2)=tem(i,2) + q(j,i)*tem(j,1)

**enddo**
In REOR: i, m, tem(i, 2) = i, m, tem(i, 2)
enddo

C+++++
do i = 1, m-1
do j = 1, n
q(j, m) = q(j, m) - tem(i, 2) * q(j, i)
enddo
enddo

C+++++
1995 continue
if(lump.eq.1) then
   do j = 1, n
      tem(j, 1) = dmass(j) * q(j, m)
   enddo
else
   call multspa(n, ia, ja, am, dmass, q(1, m), tem(1, 1))
endif

qnorm = 0.0d0
do i = 1, n
   qnorm = qnorm + q(i, m) * tem(i, 1)
enddo
qnorm = 1.0d0/dsqrt(qnorm)
c write(*,*) 'i, q(i, m), tem(i, 1) = ', i, q(i, m), tem(i, 1)
enddo

C+++++
endo
c return
end

C subroutine stepei(n, iblock, p, q, ia, ja, am, dmass, lump)
implicit real*8 (a-h, o-z)
real*8 q(n, iblock), p(n, iblock), am(1), dmass(1)
integer ia(1), ja(1)
C**** This subroutine finds: P = DMASS*Q in Step e & i. *******
if(lump.eq.1) then
C****** lumped Mass matrix ************
do j = 1, iblock
   do i = 1, n
      p(i, j) = dmass(i) * q(i, j)
   enddo
C+C else
C****** consistant Mass matrix ************
c write(*,*) 'ei: dmass=', (dmass(i), i=1, n)
c write(*,*) 'ei: am = ', (am(i), i=1, 6)
do j = 1, iblock
   call multspa(n, ia, ja, am, dmass, q(1, j), p(1, j))
enddo
C+C endif
C return
C end

C subroutine assemt(isize, iblock, t, alfa, beta, JJ)
implicit real*8 (a-h, o-z)
real*8 T(isize, isize), alfa(iblock, iblock)
real*8 beta(iblock, iblock)
C***** This subroutine puts: alfa & beta into T matrix *******
J0 = (JJ-1)*iblock
JL = JJ*iblock
if(J1.LT.isize) THEN
   do J = 1, iblock
      do I = 1, iblock
         T(J0+J, J0+I) = alfa(I, J)
         T(J0+I, J0+J) = beta(I, J)
      enddo
   enddo
C write(*,*)'j0+i, j0+j, T = ', (T(j0+i, j0+j), j=1, j0+j)
c write(*,*)'j0+j, j1+1, T = ', (T(j0+j, j1+1), j=1, j1+1)
enddo
 C********* if( J1.GE.the size of T) do not put beta in !!!! ********
 C     if( Ji.GE.the size of T)
 C     do J = 1.iblock
 C     do I = 1.iblock
 C     T(J0+I,J0+J) = alfa(I,J)
 C     endif
 C     endif
 C
 C return
 C end

 subroutine check(n,neig,isize,ncoff,ia,ja,an,ad,q,hh,eig,
 1  V,VPI,am,dmass,lump,ishift)
 implicit real*8 (a-h,o-z)
 real*8 hh(1) ,q(n,1),V(1),VPI(1),EIG(1),an(1),ad(1)
 real*8 am(1),dmass(1)
 integer ia(1), ja(1)

 C**** Error Norm Check: ||Ax-wMx||/||Ax|| ***********
 call cpup time(t01)
 rewind(15)
 read(15) (ia(i),i=1,1+n)
 rewind(11)
 read(11) (ja(i),i=1,ncoff)
 rewind(12)
 read(12) (an(i),i=1,ncoff)
 rewind(13)
 read(13) (ad(i),i=1,n)

 write(*,*)'lump,n, isize, ncoff = ',lump,n,isize,ncoff
 write(*,*)'check: eig = ',(eig(i),i=1,neig)
 write(*,*)'check: ia = ',(ia(i),i=1,n+1)
 write(*,*)'check: ja = ',(ja(i),i=1,ncoff)
 write(*,*)'check: ad = ',(ad(i),i=1,n)
 write(*,*)'check: an = ',(an(i),i=1,ncoff)
 write(*,*)'check: dmass = ',(dmass(i),i=1,n)
 do k = 1,neig
 write(*,*)'check: k, Q = ',(q(i,k),i=1,n)
 enddo

 DO 580 L=1,neig
 qmax=0.0d0
 iymax=0
 do i = 1,n
 if(dabs(q(i,L)).gt.qmax) then
  qmax=dabs(q(i,L))
  iymax=i
 endif
 enddo

 WRITE(23,'(L9,i9,5F3.1)') qmax,iymax
 call multspa(n,ia, ja,an,ad,q(1,L),V)
 VNORM=0.0d0
 DO 590 I=1,N
 VNORM=VNORM+V(I)**2
 if(lump.eq.1) then
  DO 2239 I=1,N
   vpi(I)=dmass(I)*q(i,L)
  2239 continue
 C else
 C***** Consistent Mass matrix **************
 call multspa(n,ia, ja,am,dmass,q(1,L),VPI)
 C endif

 C RT=1.0d0/EIG(L) - float(ishift)
 DO 600 I=1,N
 V(I)=V(I)-RT*VPI(I)
 600 continue
 do 601 i = 1,n
 601 VNORM=VNORM+V(I)**2
 VNORM=DSQRT(VNORM)
 VNORM=DSQRT(VNORM)

 C write(23,*)'L,wnorm,Vnorm = ',L,wnorm,vnorm
 HH(L)=VNORM/VNORM
 if(dabs(RT).lt.1.0D-14)HH(L) = DABS(RT)
SUBROUTINE JACOBI (A,B,X,EIGV,D,N,RTCL,NSMAX,IFPR,IOUT)

PROGRAM TO SOLVE THE GENERALIZED EIGENPROBLEM
USING THE GENERALIZED JACOBI ITERATION

-- INPUT VARIABLES --
A(N,N) = Stiffness matrix (assumed positive definite)
B(N,N) = Mass matrix (assumed positive definite)
X(N,N) = Vector storing eigenvectors on solution exit
EIGV(N) = Vector storing eigenvalues on solution exit
D(N) = Working vector
N = Order of matrices A and B
RTCL = Convergence tolerance (usually set to 10.**-12)
NSMAX = Maximum number of sweeps allowed
(usually set to 15)
IFPR = Flag for printing during iteration
EQ.0 No printing
EQ.1 Intermediate results are printed
IOUT = Output device number
ISHIFT: \( K = K + \text{ishift} \times M \)

-- OUTPUT --
A(N,N) = Diagonalized stiffness matrix
B(N,N) = Diagonalized mass matrix
X(N,N) = EIGENVECTORS stored columnwise
EIGV(N) = Eigenvalues

SUBROUTINE JACOBI (A,B,X,EIGV,D,N,RTCL,NSMAX,IFPR,IOUT)

Parameter (RTCL=10.**-12,NSMAX=15,N=2,IFPR=0,IOUT=6)
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 A(N,N),B(N,N),X(N,N),EIGV(N),D(N),BB
NSMAX = 15
IOUT = 6
IFPR = 0
RTCL = 1.0D-14

This program is used in single precision arithmetic on cdc equipment and double precision arithmetic on ibm or univac machines. Activate, deactivate or adjust above card for single or double precision arithmetic

INITIALIZE EIGENVALUE AND EIGENVECTOR MATRICES

DO I = 1,N
DO J = 1,N
B(I,J) = 0.0D0
ENDDO
DO J = 1,I-1
A(I,J) = A(J,I)
ENDDO
B(I,I) = 1.0D0
ENDDO
DO 10 I=1,N
IF (A(I,I).GT.0.D0 .AND. B(I,I).GT.0.D0) GO TO 4
WRITE (IOUT,2020)
STOP
4 D(I)=A(I,I)/B(I,I)
10 EIGV(I)=D(I)
DO 30 I=1,N
DO 20 J=1,N
X(I,J)=0.0D0
30 X(I,I)=1.0D0

INitalize SWEEP COUNTER AND BEGIN ITERATION

IF (N.EQ.1) RETURN

NSWEEP=0
NR=N-1

NSWEEP=NSWEEP+1
IF (IFPR.EQ.1) WRITE (IOUT,2000)NSWEEP

CHECK IF PRESENT OFF-DIAGONAL ELEMENT IS LARGE ENOUGH TO REQUIRE
ZEROING

EPS=(.01**NSWEEP)**2
DO 210 J=1,NR
JJ=J+1
DO 210 K=JJ,N

EPTOLA=(A(J,K)*A(J,K))/(A(J,J)*A(K,K))
EPTOLB=(B(J,K)*B(J,K))/(B(J,J)*B(K,K))

IF ((EPTOLA.LT.EPS).AND.(EPTOLB.LT.EPS)) GO TO 210

AKK=A(K,K)*B(J,K)-B(K,K)*A(J,K)
AJJ=A(J,J)*B(J,K)-B(J,J)*A(J,K)
AB=A(J,J)*B(K,K)-A(K,K)*B(J,J)
CHECK=(AB*AB+4.0D0*AKK*AJJ)/4.0D0

IIF (CHECK) 50,60,60

WRITE (IOUT,2020)
STOP

SQCH=DSQRT(CHECK)
D1=AB/2.0D0+SQCH
D2=AB/2.0D0-SQCH
DEN=D1

IF (DABS(D2).GT.DABS(D1)) DEN=D2
IF (DEN) 80,70,80

CA=0.
CG=-A(J,K)/A(K,K)
GO TO 90

CA=AKK/DEN
CG=-AJJ/DEN

PERFORM THE GENERAL ROTATION TO ZERO THE PRESENT OFF DIAGONAL ELEMENT

IF (N-2) 100,190,100

JPl=J+1
JMI=J-1
KPl=K+1
KM1=K-1

IF (JMI-1) 130,110,110

DO 120 I=I,JMI
AJ=A(I,J)
BJ=B(I,J)
AK=A(I,K)
BK=B(I,K)

A(I,J)=AJ+CG*AK
B(I,J)=BJ+CG*BK
A(I,K)=AK+CA*AJ
B(I,K)=BK+CA*BJ

120 IF (KPI-N) 140,140,160
140 DO 150 I=KPI,N
AJ=A(J,I)
BJ=B(J,I)
AK=A(I,K)
BK=B(I,K)

A(J,I)=AJ+CG*AK
B(J,I)=BJ+CG*BK
A(K,I)=AK+CA*AJ

150 B(I,K)=BK+CA*BJ

160 IF (JPl-KM1) 170,170,190

170 DO 180 I=JPl,KM1
AJ=A(J,I)
BJ=B(J,I)
AK=A(I,K)
BK=B(I,K)

A(J,I)=AJ+CG*AK
B(J,I)=BJ+CG*BK
A(I,K)=AK+CA*AJ

180 B(I,K)=BK+CA*BJ

190 A(K,K)=AK*2.0D0*CA*A(J,K)+CA*CA*A(J,J)
B(K, K) = B(K, K) + 2.0D0 * CA * B(J, K) + CA * CA * B(J, J)
A(J, J) = A(J, J) + 2.0D0 * CG * A(J, K) + CG * CG * AK
B(J, J) = B(J, J) + 2.0D0 * CG * B(J, K) + CG * CG * BK
A(J, K) = 0.0D0
B(J, K) = 0.0D0

UPDATE THE EIGENVECTOR MATRIX AFTER EACH ROTATION

DO 200 I = 1, N
  X(J) = X(I, J)
  X(K) = X(I, K)
  X(I, J) = X(J) + CG * X(K)
  X(I, K) = X(K) + CA * X(J)
200 CONTINUE

UPDATE THE EIGENVALUE AFTER EACH SWEEP

DO 220 I = 1, N
  IF (A(I, I) .GT. 0.0D0 .AND. B(I, I) .GT. 0.0D0) GO TO 220
  WRITE (IOUT, 2020)
  STOP
220 EIGV(I) = A(I, I) / B(I, I)
  IF (IFPR .EQ. 0) GO TO 230
  WRITE (*, 2030)
  WRITE (*, 2010) (EIGV(I), I = 1, N)

CHECK FOR CONVERGENCE

230 DO 240 I = 1, N
  TOL = RTOL * D(I)
  DIF = DABS (EIGV(I) - D(I))
  IF (DIF .GT. TOL) GO TO 280
240 CONTINUE

CHECK ALL OFF-DIAGONAL ELEMENTS TO SEE IF ANOTHER SWEEP IS REQUIRED

EPS = RTOL * EPS
DO 250 J = 1, NR
  JJ = J + 1
  DO 250 K = JJ, N
    EPSA = (A(J, K) * A(J, K)) / (A(J, J) + A(K, K))
    EPSB = (B(J, K) * B(J, K)) / (B(J, J) + B(K, K))
    IF ((EPSA .LT. EPS) .AND. (EPSB .LT. EPS)) GO TO 250
    GO TO 280
250 CONTINUE

FILL OUT BOTTOM TRIANGLE OF RESULTANT MATRICES AND SCALE EIGENVECTORS

DO 260 I = 1, N
  DO 260 J = I, N
    A(J, I) = A(I, J)
 260 B(J, I) = B(I, J)
  DO 270 J = 1, N
    BB = DSQRT (B(J, J))
    DO 270 K = I, N
      X(K, J) = X(K, J) / BB
 270 CONTINUE

RETURN

WRITE(*,*) 'THE EIGENVECTORS ARE :'
WRITE(*,*) ((X(I, J), J = 1, N), I = 1, N)

Reorder the eigenvalues by decreasing order, by Jiangning Qin, Sep. 1995
DO I = 1, N
  DO J = I + 1, N
    IF (EIGV(J) .GT. EIGV(I)) THEN
      BB = EIGV(J)
      EIGV(J) = EIGV(I)
      EIGV(I) = BB
      DO K = 1, N
        BB = X(K, J)
        X(K, J) = X(K, I)
        X(K, I) = BB
      END DO
    END IF
  END DO
END DO
WRITE(23,*) 'THE EIGENVALUES in JACOBI ARE :'
do i = 1, n
WRITE(23,*)'** I ** ','i',' ',EIGV(i)',' **'
enddo
return
STOP

UPDATE D MATRIX AMD START NEW SWEEP, IF ALLOWED

280 DO 290 I=1,N
290 D(I)=EIGV(I)
   IF(NSWEEP.LT.NSMAX) GO TO 40
   GO TO 255
2000 FORMAT(27HOSWEEP NUMBER IN *JACOBI* = ,I4)
2010 FORMAT(1HO,6E20.12)
2020 FORMAT(25H0***ERROR SOLUTION STOP /
   1    IX, 30H MATRICES NOT POSITIVE DEFINITE )
2030 FORMAT(36H0CURRENT EIGENVALUES IN *JACOBI* ARE,/)
SUBROUTINE JACOBI2 (A,B,X,EIGV,N,D,rtol,nsmax)
modified by runesha Nov 27, 1995

PROGRAM TO SOLVE THE GENERALIZED EIGENPROBLEM
USING THE GENERALIZED JACOBI ITERATION

-- INPUT VARIABLES --
A(N,N) = Stiffness matrix (assumed positive definite)
B(N,N) = Mass matrix (assumed positive definite)
X(N,N) = Vector storing eigenvectors on solution exit
EIGV(N) = Vector storing eigenvalues on solution exit
D(N) = Working vector
N = order of matrices A and B
RTOL = Convergence tolerance (usually set to 10.**-12)
NSMAX = Maximum number of sweeps allowed
(usually set to 15)

-- OUTPUT --
A(N,N) = Diagonalized stiffness matrix
B(N,N) = Diagonalized mass matrix
X(N,N) = Eigenvectors stored columnwise
EIGV(N) = Eigenvalues

Parameter (RTOL=10.**-12,NSMAX=15,IFPR=0,IOUT=6)
Parameter (RTOL=0.00000000001,NSMAX=15,IFPR=0,IOUT=6)
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(N,1),B(N,1),X(N,1),EIGV(1),D(1)

ABS(X)=DABS(X)
SQRT(X)=DSQRT(X)

This program is used in single precision arithmetic on
CDC equipment and double precision arithmetic on IBM
or UNIVAC machines. Activate, deactivate or adjust above
Card for single or double precision arithmetic

INITIALIZE EIGENVALUE AND EIGENVECTOR MATRICES

DO 10 I=1,N
IF (A(I,I).GT.0. .AND. B(I,I).LT.0.) GO TO 4
WRITE (*,2020)
STOP
4 D(I)=A(I,I)/B(I,I)
10 EIGV(I)=D(I)

DO 30 J=I,N
DO 20 K=J,N
20 X(I,J)=0.d0
30 X(I,J)=1
C...... IF (N.EQ.I) RETURN
IF (N.EQ.I) STOP

INITIALIZE SWEEP COUNTER AND BEGIN ITERATION

NSWEEP=0
NR=N-1
40 NSWEEP=NSWEEP+1
IF (IFPR.EQ.I) WRITE (*,2000)NSWEEP

CHECK IF PRESENT OFF-DIAGONAL ELEMENT IS LARGE ENOUGH TO REQUIRE
ZEROING

EPS=(.01**NSWEEP)**2
DO 210 J=1,NR
JJ=J+1
DO 210 K=J,N
210 EPTOLAF=A(J,J)*A(J,K))/A(J,J)*A(K,K))
EPTOLBF=B(J,K)*B(J,K))/B(J,J)*B(K,K))
IF ((EPTOLAF.EQ.EPS).AND.(EPTOLBF.EQ.EPS)) GO TO 210
AKK=A(K,K)*B(J,K)-B(K,K)*A(J,K)
AJJ=A(J,K)*B(J,K)-B(J,J)*A(J,K)
AB=A(J,J)*B(K,K)-A(K,K)*B(J,J)

DO 210 J=1,NR
CHECK = (AB*AB + .4*AKK*AJJ) / .4.
IF (CHECK) 50, 60, 60

50 WRITE (*, 2020)
STOP
60 SQCH = DSQRT (CHECK)
   D1 = AB / 2. + SQCH
   D2 = AB / 2. - SQCH
   DEN = D1
   IF (DABS (D2) .GT. DABS (D1)) DEN = D2
   IF (DEN) 80, 70, 80
70 CA = 0.
    CG = -A (J, K) / A(K, K)
    GO TO 90
80 CA = AKK / DEN
    CG = -AJJ / DEN

PERFORM THE GENERAL ROTATION TO ZERO THE PRESENT OFF DIAGONAL ELEMENT

90 IF (N-2) 100, 190, 100
100 JP1 = J + 1
    JM1 = J - 1
    KP1 = K + 1
    KM1 = K - 1
    IF (JM1) 130, 110, 110
110 DO 120 I = 1, JM1
   AJ = A(I, J)
   BJ = B(I, J)
   AK = A(I, K)
   BK = B(I, K)
   A(I, J) = AJ + CG*AK
   B(I, J) = BJ + CG*BK
   A(I, K) = AK + CA*AJ
120 B(I, K) = BK + CA*BJ
130 IF (KP1 - N) 140, 140, 160
140 DO 150 I = KP1, N
   AJ = A(J, I)
   BJ = B(J, I)
   AK = A(K, I)
   BK = B(K, I)
   A(J, I) = AJ + CG*AK
   B(J, I) = BJ + CG*BK
   A(K, I) = AK + CA*AJ
150 B(K, I) = BK + CA*BJ
160 IF (JP1 - KM1) 170, 170, 190
170 DO 180 I = JP1, KM1
   AJ = A(J, I)
   BJ = B(J, I)
   AK = A(K, I)
   BK = B(K, I)
   A(J, I) = AJ + CG*AK
   B(J, I) = BJ + CG*BK
   A(K, I) = AK + CA*AJ
180 B(K, I) = BK + CA*BJ
190 AK = A(K, K)
    BK = B(K, K)
    A(K, K) = AK + 2.*CA*A(J, K) + CA*CA*A(J, J)
    B(K, K) = BK + 2.*CA*B(K, J) + CA*CA*B(J, J)
    A(J, J) = A(J, J) + 2.*CG*A(J, K) + CG*CG*AK
    B(J, J) = B(J, J) + 2.*CG*B(J, K) + CG*CG*BK
    A(J, K) = 0.d0
    B(J, K) = 0.d0

UPDATE THE EIGENVECTOR MATRIX AFTER EACH ROTATION

DO 200 I = 1, N
   XJ = X(I, J)
   XK = X(I, K)
   X(I, J) = XJ + CG* XK
   X(I, K) = XK + CA*XJ
200 CONTINUE

UPDATE THE EIGENVALUE AFTER EACH SWEEP

DO 220 I = 1, N
   IF (A(I, I) .GT. 0. .AND. B(I, I) .GT. 0.) GO TO 220
   WRITE (*)
   STOP
220 EIGV(I) = A(I, I) / B(I, I)
   IF (IFPR.EQ.0) GO TO 230
   WRITE (IOUT, 2030)
WRITE(IOUT,2010) (EIGV(I),I=1,N)

CHECK FOR CONVERGENCE

230 DO 240 I=1,N
    TOL=RTOL*D(I)
    DIF=DABS(EIGV(I)-D(I))
    IF(DIF.GT.TOL) GO TO 280
240 CONTINUE

CHECK ALL OFF-DIAGONAL ELEMENTS TO SEE IF ANOTHER SWEEP IS REQUIRED

EPS=RTOL**2
DO 250 J=1, N
    JJ=J+1
    DO 250 K=JJ,N
        EPSA=(A(J,K)*A(J,K))/(A(J,J)+A(K,K))
        EPSB=(B(J,K)*B(J,K))/(B(J,J)+B(K,K))
        IF((EPSA.LT.EPS) .AND. (EPSB.LT.EPS))GO TO 250
250 CONTINUE

FILL OUT BOTTOM TRIANGLE OF RESULTANT MATRICES AND SCALE EIGENVECTORS

255 DO 260 I=1,N
    DO 260 J=1,N
        A(J,I)=A(I,J)
    260 B(J,I)=B(I,J)
    DO 270 J=I,N
        BB=DSQRT(B(J,J))
        DO 270 K=I,N
            X(K,J)=X(K,J)/BB
270 CONTINUE

EIGENVECTORS

280 DO 290 I=1,N
    D(I)=EIGV(I)
    IF(NSWEEP.LT.NSMAX) GO TO 40
290 CONTINUE

UPDATE D MATRIX AND START NEW SWEEP IF ALLOWED

280 DO 290 I=1,N
    D(I)=EIGV(I)
    IF(NSWEEP.LT.NSMAX) GO TO 40
290 GO TO 255
2000 FORMAT(27HOSWEEP NUMBER IN *JACOBI* = ,I4)
2010 FORMAT(IHO,6E20.12)
2020 FORMAT(25H0***ERROR SOLUTION STOP /
1 30H MATRICES NOT POSITIVE DEFINITE )
2030 FORMAT(36H0CURRENT EIGENVALUES IN *JACOBI* ARE,/) END

subroutine matmat3(a,b,n,iq,temp)
xx(n*iq)=xx(n*iq)*phi(iq,iq)
a = a * b
implicit real*8(a-h,o-z)
dimension a(1), b(iq,1), temp(1)
iqq=iq-mod(iq,8)
do 10 i=1,n
do 20 j=1,iq
    temp(j) = a((j-1)*n+i)
do 30 k=1,iq
    sum=0.
d0  do 1 l=iq,iqq, 8
        sum=sum + temp(l)*b(l,1)*b(l+1,1)*b(l+2,1)*b(l+3,1)+temp(l+4,1)*b(l+5,1)+temp(l+6,1)*b(l+7,1)
    enddo
do 40 l=iq+1,iq
    sum = sum + temp(l)*b(l,1)
do 10 continue
end

return
This version does not have any loop enrolling

subroutine multspa2(n, ip, indx, coef, diag, rhs, temp)
subroutine multsp(n, ia, ja, an, ad, b, c, isupd, iptrs)
implicit real*8(a-h,o-z)
real et(2)
dimension rhs(l), coef(l), indx(l)
dimension diag(l), temp(l), ip(l)
nccoef = ip(n+1)-1

c  ett1=etime(et)
ttl=et(l)
do i=1,n
temp(i)=diag(i)*rhs(i)
enddo
do 10 i=1,n
ifirst=ip(i)
ilast=ip(i+1)-1
suml=temp(i)
10       do k=ifirst,ilast
kk= indx(k)
suml=suml+coef(k)*rhs(kk)
temp(kk)=temp(kk)+coef(k)*rhs(i)
enddo
temp(i)=suml
continue

c  ett2=etime(et)
tt2=et(tt2-ttl)
c  time2=(tt2-ttl)
c  noper= ncoef*4+n
write(*,*) 'Time in Matrix-by-Vector = ',time2
return
end

subroutine normcheck (iq, n, neig, ncoef, ia, ja, an, ad, din, am, eigv, 
+ xx, v, vpl, lump, err, ishift)
implicit real*8 (a-h,o-z) 
real*8 an(l), ad(1), eigv(1), xx(1), dm(1) 
+ err(l), v(1), vpl(l), am(l)
integer ia(l), ja(l)
real t01,t02,t03
c..... Purpose : Calculate ||Kx-eigv.Mx||/||K.x||
call cputime(t01)
c
read(15) (ia(i),i=1,1+n)
c
read(11) (ja(i),i=1,nccoef)
c
read(12) (an(i),i=1,nccoef)
c
read(13) (ad(i),i=1,n)
do 100 k=1,neig
call multspa2(n, ia, ja, an, ad, xx((k-1)*n+1), v)
VNORM=0.0d0
do 110 i=1,n
VNORM=VNORM+V(I)*V(I)
110        VNORM=VNORM+V(I)*V(I)
if (lump.eq.1) then
******** Lumped Mass matrix **********
do 120 i=1,n
vpl(I)=dm(I)*xx((k-1)*n+i)
120      continue
else
******** consistent Mass matrix **********
call multspa2(n, ia, ja, an, ad, xx((k-1)*n+1), vpl)
endif

RT= eigv(k)
DO 600 I=1,N
V(I)=V(I)-rt*VPL(I)
600 continue
do 601 i = 1, n

100 CONTINUE
call cputime(t02)
t03=t02 - t01
write(23,*) ' Time normcheck = ', t03
WRITE(23,*) + '***** # **,*** EIGV *** , *** Hertz *** , ** ERROR NORM ** ' 

if(ishift.ne.0) then
do i=1,neig
   eigv(i)=eigv(i)-float(ishift)
endo
endif

DO 900 K=1,NEIG
   hertz=dsqrt(eigv(k))/(2.0*3.1415927)
   hertz=dsqrt(dabs(eigv(k)))/(2.0*3.1415927)
WRITE(23,901)K,EIGV(K),HERTZ,ERR(K)
901 FORMAT(2X,I5,2X,4E15.7)
CONTINUE
return
end

subroutine spasubspace(iq,n,ncoef,neig,lumas,mtot,ik,jk,dk
+ ,ak,am,iu,ju,di,un,xkk,xmm,phi,eigv,eigv0,tolj
+ ,temp,xx,ishift)

c developed by runesha October 20,1995
implicit real*8(a-h,o-z)
real*8 ak(i),dk(i),am(1),un(1),di(1),xx(1),xkk(iq,1),
+ xmm(iq,1),temp(1),eigv(1),eigv0(1),tolj(1),phi(iq,1)
integer ik(1),jk(1),iu(1),ju(1),itertot
real t1,t2,t3,t4,t5,t6,t7
real t1,t7

call cputime(t1)
to limit the number of iterations
itertot= 100
ICONV=0
toler=0.000000000001
nsmax=12
Initialization
do i=1,iq
   eigv0(i)=0.
endo
c call cputime(t2)
c
write(23,*) ' NEQ = ',n
write(23,*) ' NCOEF= ',ncoef
write(23,*) ' LUMAS= ',lumas
C write(23,*) 'NEIG = ',neig
C write(23,*) 'IQ = ',iq
C write(23,*) 'MAX NUMBER OF ITER ALLOWED = ',itertot
C write(23,*) 'TOLER IN JACOBI = ',toler
memory=2*ncoef+2*ncoef2+7*n +3*iq*iq +3*iq+n*iq
C
C write (23,*) 'Total memory = ' ,memory
C
C call cputime(t3)
C
C .... STARTING ITERATION VECTORS.
C..
The following is the basic just 1 on the diag not even Kii/Mii
C
C ii=n*iq
C
C do i=!, ii
C     xx(i)=0.d0
C   enddo
C
C do i= i, iq
C   xx (i+n*(i-l)) =l.d0
C enddo
C
C KJ Bathe style starting vector Iteration
nd=n/iq
C
C if(lumas.ne.1) go to 4
C   j=0
C   do 6 i=1,n
C      xx(i)=dm(i)
C      if(dm(i).gt.0) j=j+1
C   6        temp(i)=dm(i)/dk(i)
C     if(iq.le.j) go to 16
C     write(*,*) 'IQ CANNOT BE LARGER THAN 3 DIAG NON ZERO ' stop
C 4   do 11 i=1,n4
C C 11    xx(i)=dm(i)
C 16   do 20 i=n+l,n*iq
C     xx(i)=0
C     i=n-nd
C   do 30 j=2,iq
C      rt=0
C      do 40 i=1,1
C         if(temp(i).lt.rt) go to 40
C      rt=temp(i)
C     ij=i
C   40 continue
C   do 50 i=1,n
C      if (temp(i).le.rt) go to 50
C     rt=temp(i)
C     ij=i
C   50 continue
C   tolj(j)=float(ij)
C     temp(ij)=0.
C     l=1-nd
C 30   xx((j-1)*n+ij)=1.
C write(*,*) 'Degrees of freedom excited by unit starting
C write(*,*) (tolj(j), j=2,iq)
C
C A random vector is added to the last vector
pi=3.141592654
C
C xxx=0.5
C iloc=(iq-1)*n
doi=1,n
C xxx=(pi+xxx)**5
C ix=int (xxx
C xxx=xxx-float (ix)
C
C 60 xx(iloc+k)=xx(iloc+k)+xxx
C
C call cputime(t4)
C
C write(*,*) 'STARTING ITERATION VECTORS'
C do j=1,iq
C write(*,*) (xx((j-1)*n+i), i=1,n)
C enddo
C.... End Iteration vectors
C
C.... BEGIN SUBSPACE ITERATIONS
C call cputime(t5)
C n1=n-mod(n,8)
C iter=0
iter=iter+1

calculate the projection of K and M

do 110 i=1,iq
    call fbe(n,iu,ju,di,un,xx((j-1)*n+1),temp,iopfb)
do 130 i=j,iq
    sum=0.d0
do k=1,nl,8
        sum=sum+xx((i-1)*n+k)*temp(k)
        +xx((i-1)*n+k+1)*temp(k+1)
        +xx((i-1)*n+k+2)*temp(k+2)
        +xx((i-1)*n+k+3)*temp(k+3)
        +xx((i-1)*n+k+4)*temp(k+4)
        +xx((i-1)*n+k+5)*temp(k+5)
        +xx((i-1)*n+k+6)*temp(k+6)
        +xx((i-1)*n+k+7)*temp(k+7)
    enddo
do 140 k=nl+1,n
    sum=sum+xx((i-1)*n+k)*temp(k)
do 150 k=1,n
    xx((j-1)*n+k)=temp(k)
do 110 continue

if (lumas.ne.1) then
    do 160 j=1,iq
        call multspa2(n,ik,jk,am,dm,xx((j-1)*n+1),temp)
do 180 i=j,iq
        sum=0.d0
do k=1,nl,8
        sum=sum+xx((i-1)*n+k)*temp(k)
        +xx((i-1)*n+k+1)*temp(k+1)
        +xx((i-1)*n+k+2)*temp(k+2)
        +xx((i-1)*n+k+3)*temp(k+3)
        +xx((i-1)*n+k+4)*temp(k+4)
        +xx((i-1)*n+k+5)*temp(k+5)
        +xx((i-1)*n+k+6)*temp(k+6)
        +xx((i-1)*n+k+7)*temp(k+7)
    enddo
do 190 k=nl+1,n
    sum=sum+xx((i-1)*n+k)*temp(k)
do 180 xmm(i,j)=sum
if (iconv.gt.0) go to 160
do 200 k=1,n
    xx((j-1)*n+k)=temp(k)
do 160 continue
else
    do 162 j=1,iq
        temp(i)=xx((j-1)*n+i)*dm(i)
do 182 i=j,iq
        sum=0.d0
do k=1,nl,8
        sum=sum+xx((i-1)*n+k)*temp(k)
        +xx((i-1)*n+k+1)*temp(k+1)
        +xx((i-1)*n+k+2)*temp(k+2)
        +xx((i-1)*n+k+3)*temp(k+3)
        +xx((i-1)*n+k+4)*temp(k+4)
        +xx((i-1)*n+k+5)*temp(k+5)
        +xx((i-1)*n+k+6)*temp(k+6)
        +xx((i-1)*n+k+7)*temp(k+7)
    enddo
do 192 k=nl+1,n
    sum=sum+xx((i-1)*n+k)*temp(k)
do 182 xmm(i,j)=sum
if (iconv.gt.0) go to 162
do 202 k=1,n
    xx((j-1)*n+k)=temp(k)
do 162 continue
endif

do 131 i=1,iq-l
    do 132 j=i+1,iq
        xmm(i,j)=xmm(i,j-i)
xkk(i,j)=xkk(j,i)
do 131 continue

write(*,'(A)') 'REDUCED STIFFNESS MATRIX'
do i=1,iq
    do j=1,iq
        write(*,'(A,i6)') (xkk(i,j),j=1,iq)
    enddo
write(*,'(A)') 'REDUCED MASS MATRIX'
do i=1,iq
    do j=1,iq
        write(*,'(A,i6)') (xmm(i,j),j=1,iq)
    enddo

solve for eigensystem of subspace operations
call jacobi2(xkk, xmm, phi, eigv, iq, temp, toler, nsmax)
cwrite(*,*) 'AFTER JACOBI'
cwrite(*,*) 'PHI'
cwrite(*,*) (phi(i,j), i=1,iq), j=1,iq)
cwrite(*,*) 'EIGV'
cwrite(*,*) (eigv(i), i=1,iq)

c ARRANGE EIGENVALUES AND EIGENVECTORS IN ASCENDING ORDER
do i=1,iq
do j=i+1,iq
if(eigv(i) .gt. eigv(j)) then
eigvt=eigv(j)
eigv(j)=eigv(i)
eigv(i)=eigvt
do 220 l=1,iq
phi(i,j)=phi(l,j)
dphi(i,j)=phi(l,i)
220 phi(l,i)=phi(l,i)
enddo
enddo

C ......................
cwrite (*, *) 'AFTER ARRANGING IN ASCENDING ORDER '
cwrite (*, *) 'PHI '
cwrite(*,*) (phi(i, j), i=l, iq), j=l, iq)
cwrite (*, *) 'EIGV'
cwrite(*,*) (eigv(i), i=l, iq)

C IMPROVED APPROXIMATION TO THE EIGENVECTORS
Xk+l =Xk+1 * Qk+l
call matmat3(xx,phi,n, iq, temp)
cwrite(*,*) 'IMPROVED EIGENVECTORS '
cwrite(*,*) (xx(i), i=1,n*iq)

C CHECK FOR CONVERGENCE OF EIGENVALUES

c call cpu(time(t6)
if (iconv.gt.0) go to 500
do 400 I=1,iq
diff=dabs(eigv(i)-eigv0(i))
tolj(i)=diff/eigv(i)
do 410 i=1,neig
if(tolj(i).gt.toler) go to 440
410 continue
iconv=1
go to 10
440 if(iter.lt.iterot) go to 420
iconv=2
write(*,*) 'The number of iterations is larger than the allowed
+ iter', iter, itertot
go to 10
420 do 430 I=1,iq
eigv0(i)=eigv(i)
go to 10
430 continue

C END OF ITERATION

c call cpu(time(t7)
c500 write(23,*) ' RESULTS FOR EIGENSOLUTION'
c500 write(23,*) ' RESULTS FOR EIGENSOLUTION'
c write(23,*) '-----------------------------'
c write(23,*) 'Number of iterations =',iter-1
c write(23,*)
c write(23,*) 'Eigenvalues '
c write(23,*) (eigv(i), i=1,neig)
c write(23,*)
c write(23,*) 'Eigenvectors'
c write(23,*)
c do 510 j=1,neig
c510 write(23,*) (xx((j-1)*n+i), i=1,n)
c write(23,*)
c write(23,*) 'TOLERANCE CHECK ON EIGENVALUES '
c write(23,*) ' # #', ' EIGV ', ' TOLJ'

if(ishift.ne.0) then
do i=1,neig
write(23,*) ' ', i , ' ',eigv(i)-float(ishift), ' ', tolj(i)
enddo
else
   do i=1, neig
      write(23,*), ' ', i, ', ', eigv(i), ', ', tolj(i)
   enddo
endif
write(23,*)
write(23,*) ' Timing'
write(23,*) ' Time Init + Start. Iter. Vect. =', (t2-t1)
write(23,*) ' Time subspace Iter. =', t6-t5
write(23,*) ' Time subspace Iter. =', t7-t1

C..... ERROR NORM CALCULATION : |Kx-eigv.Mx|/||Kx||
call normcheck(iq,n,neig,ncoef,ik,jk,ak,dk,am,
   eigv,xx,temp,eigv0,umas,tolj,ishift)
return
end

******************************************************************************
c **********************************************************************
  c CONSISTANT MASS CASE + SHIFT FOR REGULAR Lanczos
  c
  c H. Runesha dec 22, 1997
  c**********************************************************************

SUBROUTINE SP2LAN2(N,LANMAX,NEIG,UN,DI,LU,W,W1,W2,W4,$
STEM,EIG,ERR,VEC,ncoff,ncof2,dmass,Q,lump,ishift
+am,ia,ja)

SUBROUTINE SP2LAN2(N,LANMAX,NEIG,UN,DI,LU, W4,W1,W2,W4,$
STEM,EIG,ERR,VEC,ncoff,ncof2,dmass,Q,$
+am,lump,ishift,ia,ja)

 implicit real*8 (a-h,o-z)
 REAL*8 W(1),VEC(lanmax,lanmax) ,EIG(1) ,W4(1) ,W2(1) ,Q(N,LANMAX)
 REAL*8 TEM(1) ,ERR(l) ,wl(1) ,dmass(1) ,UN(1) ,DI(1) ,am(1)
 INTEGER IU(1) ,JU(1) ,ia(1) ,ja(1)
 COMMON /QIN/ QLAN,EPSOO
 common/sp2com/nbuf(4)

 rewind (10)
 read(10) (dmass(i) ,i=1,n)
 if(neig.ne. 0 .and. lump.ne. 1) then
 rewind (17)
 read(17) (am(i) ,i=1,ncoff)
 write(*,*) (am(i) ,i=1,ncoff)
 endif

 j'ai besoin de IA et JA pour le multiplication de la mass
 les valeurs ont ete detruites pendant la factorization
 essayer de ne pas les relire pendant le normcheck
 rewind(15)
 read(15)(ia(i),i=1,1+n)
 rewind(11)
 read(11)(ja(i),i=1,ncoff)

 write(*,*) '$$$ UN =', (un(i),i=1,15)

 iam=0
 nodes=1
 NBLK = 1
 RTOL = 0.000000001d0
 NNNN= 4*NEIG
 IF(NNNN.GE.LANMAX)NNNN=LANMAX-1
 MN=0
 DO 61 I=1,LANMAX
 EIG(I)=0.0d0
 ERR(I)=0.0d0
 EPS=GETEPS(IBETA, IT, IRI) EPS=EPS
 EPS0=EPS
 EPS0=EPS*5.0d0
 wnor=0.0d0
 k=1
 do 14 i=1,n
 w(i)=1.0d0
 c if(k.gt.100) k = 1
 14 wnor=wnor+w(i)*w(i)
 wnor=1.0d0/dsqrt (wnor)
 DO 15 I=1,N
 W(I)=w(i)*wnor
 W(I)=0.0d0
 CONTINUE
CONTINUE
  do 9901 i=1,n
  w2(i)=dmass(i)*w(i)
  continue
  call multspa2(n,ia,ja,am,dmass,w,w2)
  bet=0.0d0
  do 9901 I=1,N
  bet=bet+w2(i)*w(i)
  bet=dsqrt(bet)
  do 9901 I=1,N
  w(i)=w(i)/bet
  Q(i,1)=W1(i)
  99012 continue
  c do 99013 I=1,n
  c99013 w2(i)=dmass(i)*w1(i)
  call multspa2(n,ia,ja,am,dmass,w1,w2)
  do 50 J=1,LANMAX-1
  do 1001 I=1,N
  W4(I)=W2(I)
  call fbe(n,iu,ju,di,un,w2,w4,iopfb)
  do 55 I=1,N
  W4(I)=W4(I)-W(I)*BET
  55 CONTINUE
  ALF=DDOT(N,W2,W4,1)
  DO 60 I=1,N
  W4(I)=W4(I)-ALF*W1(I)
  60 CONTINUE
  c do 9902 I=1,n
  c9902 w2(i)=dmass(i)*w4(i)
  call multspa2(n,ia,ja,am,dmass,w4,w2)
  BET2=DDOT(N,W4,W2,1)
  bet2=dsqrt(bet2)
  write(*,'(a20,3x,f12.8)') before call REORTH: JJ = ',jj
  CALL REORTH(N,W2,
  &W4,Q,EPS,LANMAX,TEM,JJ,BET2)
  c do 9903 I=1,n
  c9903 w2(i)=dmass(i)*w4(i)
  call multspa2(n,ia,ja,am,dmass,w4,w2)
  BET2=DDOT(N,W4,W2,1)
  write(*,'(a20,3x,f12.8)') before 100 : JJ,beta = ',jj,bet
  bet2=dsqrt(bet2)
  100 CONTINUE
  DBET2=1.0d0/BET2
  DO 110 I=1,N
  W(I)=W(I)
  W1(I)=W4(I)*DBET2
  Q(I,JJ+1)=W1(I)
  W2(I)=W2(I)*DBET2
  110 CONTINUE
  EIG(JJ)=ALF
  ERR(JJ)=BET2
  write(*,'(a20,3x,f12.8,2x,3x,f12.8)') JJ, ALF,BET2,iain=' ,jj, alf,bet2
  IF (JJ.LT.NNNN) GO TO 50
  if (nodes.gt.l) call mp_sync(nbuf(4))
  write(*,'(a20,3x,f12.8,2x,5x,a2)') ishift = ', ishift
  CALL JACOBIP2(N,NEIG,JJ,BET2,W(1),Q,VEC,EIG,
  &ERR,UN,TEM,RTOL,LANMAX,DMASS,AM,
  &W1,N+1,NNNN,NSUC,ncoff,ncof2,iu,ju,di,ishift)
  IF (NSUC.EQ.I) GO TO 2211
  CONTINUE
  CALL JACOBIP2(N,NEIG,JJ-1,BET2,W(1),Q,VEC,EIG,
  &ERR,UN,TEM,RTOL,LANMAX,DMASS,AM,
  &W1,N+1,NNNN,NSUC,ncoff,ncof2,iu,ju,di,ishift)
  2211 CONTINUE
  RETURN
END
REAL*8 DMAS(1),Q(N,LANMAX),VEC(lanmax,lanmax)
$ 
integer ia(1),ja(1)
common/sp2com/nbuf(4)
COMM/NQIN/QLAN, EPS
COMMON /QPJA/ SUB(1000)

iam=0
nodes=1
NSUC=0
DO 1 I=1,LANMAX
DO 2 J=1,LANMAX
VEC(I,J)=0.0d0
HH(I)=EIG(I)
SUB(I)=ERR(I)
VEC(I,I)=1.0d0
CONTINUE
1 
if(nodes.gt.1)call mp_sync(nbuf(4))
call TQL2(LANMAX,NJ,EIG,ERR,VEC,IRR,EPS)
c 
c write(*,*),IN JACOBQ: TQL2 is done !!!!
DO 20 I=1,NJ
ERR(I)=0.0d0
do 21 J=1,NJ
21 
ERR(I)=ERR(I)*0.005d0*ABS(BET2*VEC(NJ,I)/EIG(I))
c 
c write(*,*),I,ERR(I),VEC(NJ,I)
CONTINUE

DO 30 I=1,NEIG
IF(ERR(I) .GT.0.000001 .AND. NJ .LT. (LANMAX-I)) GO TO 300
IF(ERR(I) .GT. RTOL .AND. NJ .LT. (LANMAX-I)) GO TO 300
30 
if(nodes.gt.1)call mp_sync(nbuf(4))
call VECTRA(N,NEIG,0,VEC,Q,LANMAX,NJ,V)
c 
c write(*,*),IN JACOBQ: VECTRA is done !!!!
c NSUC=1,successful
NSUC=1
c 
DO 580 L=1,NEIG
nc=9*nodes
do L=1,neig
do i=n-nadd+l,n
c 
c write(*,*),NADD = ',nadd,n
c endif
rewind(15)
read(15)(ia(i),i=1,l+n)
rewind(11)
read(11)(ja(i),i=1,ncoff)
rewind(12)
read(12)(an(i),i=1,ncoff)
rewind(13)
read(13)(ad(i),i=1,n)
c 
c write(*,*),***PI** AD= ',ad(i),i=1,n)
c 
c write(*,*),
write(*,*),lanmax,n,neig'
c 
c write(*,*),lanmax,n,neig
c 
c write(*,*),IA = ',(ia(i),i=1,l+n)
c write(*,*),JA = ',(ja(i),i=1,ncoff)
c write(*,*),AN = ',(an(i),i=1,ncoff)
c write(*,*),AD = ',(ad(i),i=1,n)
c write(*,*),DM = ',(dmass(i),i=1,n)
c write(*,*),EIG= ',(eig(i),i=1,lq)
c do 510 j=1,neig
c510 
write(*,*), (q(i,j),i=1,n)
c 
DO 580 L=1,neig
c call multspa(n,ia,ja,an,ad,q,v)
c call mulmeiko(n,)/
c 
c if(ninc.gt.8)call mp_sync(nbuf(4))
c call sp2mul(iam,ninc,n,icolg,maxa,stif,q(1,L),vpl,v)
c call multspa(n,ia,ja,an,ad,q(1,L),v)
c 
c write(*,*),Q(i,L) = ',L,(q(i,L),i=1,n)
VNORM=0.0d0

DO 590 I=1,N
VNORM = VNORM + V(I) * V(I)

WU1NORM = 0.0D0

do 2239 i = 1, n

vpl(i) = dmass(i) * Q(i, L)

continue

call multiSpa(n, ia, ja, am, dmass, q(1, L), vpl)

write(*, *) 'I, vpl=', i, (vpl(ii), ii = 1, n)

RT = 1.0D0 / EIG(L)

DO 600 I = 1, N

err = max(abs(v(i)), abs(rt * vpl(i)))

error = abs(v(i)) / err

if(error .gt. 0.95D0 .and. err .gt. 0.00001D0) then

write(*, *) 'V(I) = V(I) - RT*VPL(I)

continue

write(*, *) 'V(I) = ', (v(i), i = 1, n)

WNORM = WNORM + V(I) * V(I)

continue

write(*,*) 'V(I) = ', (v(i), i = 1, n)

WNORM = DSQRT(WNORM)

HH(L) = WNORM / VNORM

continue

write(*,*) 'HH = ', (hh(i), i = 1, neig)

call cputime(t02)

t2 = t02 - t01

WRITE(23, *) '*** K, * EIG*, *HERTZ*, *ERROR*, *NORM*** iam'

if(ishift .eq. 0) then

DO 700 K = 1, NEIG

HERT2 = 1.0 / (2.0 * 3.1415927 * DSQRT(EIG(K)))

write(23, 701) K, 1.0 / EIG(K), HERTZ, ERR(K), HH(K)

CONTINUE

else

DO 705 K = 1, NEIG

EIG(K) = EIG(K) / float(ishift)

HERT2 = DSQRT(EIG(K)) / (2.0 * 3.1415927)

write(23, 701) K, 1.0 / EIG(K), HERTZ, ERR(K), HH(K)

CONTINUE

endif

NSUC = 1

if(ishift .eq. 0) write(23, *) 'JACOBIQ: Steps in IAM = ', nj, iam

RETURN

300 NNNN = MIN0 (LANMAX - I, NJ + 3 * (NEIG - I) + 4)

NSUC = 0

DO 304 I = 1, LANMAX

EIG(I) = HH(I)

ERR(I) = SUB(I)

RETURN

END

c**********************************************************************************
c

subroutine REORD2(n, ncioff, nreord, mtota, mtoti, a, iq, neig)
real*8 a(1)

This is an additional routine to reorder as well the consistant mass

note personnel
si je met un IF LUM en dehors, il faut commenter celui-ci

integer IQ(1)

if(2 * ncioff .gt. mtota) then

write(*, *) 'REORD: increase MTOTA to: 2*ncioff = ', 2 * ncioff

stop

endif

if(3 * ncioff + 7 * n + 5 .gt. mtoti) then
write(*,*!) 'REORD: increase MTOTI to: 3*ncoff+7*n+5 = ',
1 3*ncoff+7*n+5
stop
endif
if(nreord.ne.0) then
  call cputime(time0)
call iread0(n,ncoff,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),
1 iq(4*n+3),iq(5*n+4),iq(5*n+4+ncoff))
call cputime(time1)
  rewind(18)
  write(18) (iq(i),i=5*n+4+ncoff,5*n+3+3*ncoff)
call genmmd(n,iq(4*n+3),iq(5*n+4+ncoff),iq(1+n),
1 iq(6*n+5+3*ncoff),iq(2*n+1),
3 iq(3*n+2),iq(1),iq(5*n+4+3*ncoff),ncofsub,maxcon,nterms)
  rewind(18)
call cputime(timel)
call getnewk(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+3),
3 iq(5*n+4+ncoff),iq(5*n+4))
call copyk(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+4+
4 ncoff),a(1),a(1+ncoff),ncoff)
c pierrot
  c IF (LUMP.NE.1) THEN
  call copyk2(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+4+
4 ncoff),a(1),a(1+ncoff),a(2*n+1),a(3*n+1),nofsub,maxcon,nterms)
  c ENDIF
  c pierrot
  call copydb(n,iq(1+n),a(1),a(1+n),a(2*n+1),a(3*n+1),neig)
9988 continue
write(23,*) 'CPU to get MD reordering = ',time2-timel
  end
  return
end
C
C**************************************************************************
C subroutine copyk2(n,ia,perm,ju,ja,un,am,ncoff)
integer ia(1),ju(1),ja(1),perm(1)
real*8 am(1),un(1)
do i=1,ncoff
  un(i)=0.
am(i)=0.
endo
  rewind(11)
  read(11) (ja(i),i=1,ncoff)
  rewind(17)
  read(17) (am(i),i=1,ncoff)
  rewind(15)
  c
  read(15) (ia(i),i=1,n)
do 200 I = 1,n-1
    I0 = perm(i)
do 220 J = ia(I0),ia(I0+1) - 1
      J0 = perm(ju(J))
i0 = I0
i100 = J0
    if(J0.LT.I0) then
      i0 = J0
i100 = 10
    endif
    CDIR$ IVDEP
do 230 jj = ia(i0),ia(i0+1)-1
    if(ja(jj).NE.ia(jj)) go to 230
      un(J) = am(JJ)
go to 220
      continue
230 continue
220 continue
200 continue
write(17) (un(i),i=1,ncoff)
subroutine REORD3(n, ncoff, nreord, mtota, mtoti, a, iq, neig)
real*8 a(1)
This is an additional routine to reorder as well the
constant mass

note personnel
si je met un IF LUM en dehors, il faut commenter celui-ci

integer IQ(1)
if(2*ncoff.gt.mtota) then
write(*,*)'REORD: increase MTOTA to: 2*ncoff = ',2*ncoff
stop
endif
if(3*ncoff+7*n+5.gt.mtoti) then
write(*,*)'REORD: increase MTOTI to: 3*ncoff+7*n+5 = ',
1 3*ncoff+7*n+5
stop
endif
if(nreord.ne.0) then
call cputime(time0)
call iread0(n,ncoff,iq(1),iq(l+n),iq(2*n+l),iq(3*n+2),
1 iq(4*n+3),iq(5*n+4+ncoff))
call cputime(time1)
rewind(18)
write(18) (iq(i),i=5*n+4+ncoff,5*n+3+3*ncoff)
call genmmd(n, iq(4*n+3),iq(5*n+4+ncoff),iq(1+n),
4 iq(6*n+5+3*ncoff),iq(2*n+1),
3 iq(3*n+2), iq(1),iq(5*n+4+3*ncoff),nofsub,maxcon,nterms)
rewind(18)
read(18) (iq(i),i=5*n+4+ncoff,5*n+3+3*ncoff)
call cputime(time2)
call getnewk(n,iq(1),iq(1+n),iq(2*n+1),iq(3*n+2),iq(4*n+3),
3 iq(5*n+4+ncoff),iq(5*n+4))
call copyk(n,iq(1),iq(1+n),iq(2*n+1),iq(5*n+4),iq(5*n+4+
4 ncoff),a(1),a(1+ncoff),ncoff)
call copydb(n,iq(1+n),a(1),a(1+n),a(2*n+1),a(3*n+1),neig)
continue
write(23,*)'CPU to get MD reordering = ',time2-time1
endif
return
end

subroutine copyk3(n,ia,perm,iu,ju,ja,un,am,ncoff)
integer ia(l),iu(l),ja(l),ju(l),perm(l)
real*8 am(l),un(l)
do i=1,ncoff
un(i)=0.

am(i) = 0.
enddo
rewind(11)
read(11) (ja(i), i=1, ncoff)
rewind(17)
read(17) (am(i), i=1, ncoff)
rewind(15)
read(15) (ia(i), i=1, n)

    do 200 I = 1, n-1
        I0 = perm(i)
        do 220 J = iu(I), iu(I+1) - 1
            J0 = perm(ju(J))
            if (j0 .LT. i0) then
                j00 = j0
                endif
                CDIR$ IVDEP
                do 230 jj = ia(ij0), ia(ij0+1) - 1
                    if (ja(jj) .NE. ij00) go to 230
                    un(J) = am(JJ)
                    go to 220
                230       continue
                220       continue
                200       continue
                rewind(17)
                write(17) (un(i), i=1, ncoff)

        write(*,*) ' PERM =', (perm(i), i=1, 10)
        write(*,*) ' IA =', (IA(I), i=1, 10)
        write(*,*) ' JA =', (JA(i), i=1, 10)
        write(*,*) ' IU =', (IU(I), i=1, 10)
        write(*,*) ' JU =', (JU(i), i=1, 10)
        write(*,*) ' AM =', (am(i), i=1, 10)
        write(*,*) ' UN =', (un(i), i=1, 10)
        return
    end
EXEC = a.out
OBJ = spamain.o spaldln.o reord.o meikolan.o blanlib.o \
    jacobi2.o matmat3.o multspa2.o normcheck.o spasubspaceN.o \
    meikolan2.o reord2.o reord3.o
SOURCE = spamain.f spaldln.f reord.f meikolan.f blanlib.f \
    jacobi2.f matmat3.f multspa2.f normcheck.f spasubspaceN.f \
    meikolan2.f reord2.f reord3.f
F77 = f77
#F77 = /usr/local/lang/f77
#FFLAGS= -Mperf
#FFLAGS= -04 -Knoieee -Mvect
#DEBUG = -g
.f.o:  
    $(F77) $(DEBUG) $(FFLAGS) -c $<
$(EXEC): $(OBJ)
    $(F77) -o $(EXEC) $(OBJ) $(FFLAGS)
source:
    cat $(SOURCE) > source.f
1234567890
61
51
41
31
21
11
01
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
749.9999999998
0.562499999999

3 1
2
1 2
2 1
1

66.1.0

Testing Agoura Hill’s small eigen data!!

1234567890
61
51
41
31
21
11
01
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.

5.8663999999999D-09
5.8663999999999D-09
0.0.