DEPARTMENT OF CIVIL & ENVIRONMENTAL ENGINEERING
COLLEGE OF ENGINEERING & TECHNOLOGY
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529-0162

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

$$[K]\ [\phi]\ =\ [M]\ [\phi]\ [\lambda] \tag{1}$$

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 1: Step-by step Basic Subspace Algorithm

Step 1: Select the starting iteration vectors $[Y_1]_{N\times q}$ where $q << N$

Step 2: Factorize the structural stiffness matrix

$$[K]\ =\ [L][D][L]^T \tag{2}$$

In Eq. (2), [L] is the lower triangular matrix, and [D] is the diagonal matrix

Step 3: For k = 1,2, ... 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

$$[K][\Phi_{k+1}]_{N\times q}\ =\ [Y_k]_{N\times q} \tag{3}$$

Step 5: Compute the reduced stiffness matrix

$$[K^R{}_{k+1}]_{q\times q}\ =\ [\Phi_{k+1}]^T_{q\times N}[Y_k]_{N\times q} \tag{4}$$

Step 6: Compute the reduced mass matrix

$$[\bar{Y}_{k+1}]_{N\times q}\ =\ [M]_{N\times N}[\Phi_{k+1}]_{N\times q} \tag{5}$$

$$[M^R{}_{k+1}]_{q\times q}\ =\ [\Phi]^T_{q\times N}\ [\bar{Y}_{k+1}]_{N\times q} \tag{6}$$

Step 7: Solve the reduced eigen-equations

$$[K^R{}_{k+1}]_{q\times q}[Q_{k+1}]_{q\times q}\ =\ [M^R{}_{k+1}]_{q\times q}[Q_{k+1}]_{q\times q}[\Omega^2{}_{k+1}]_{q\times q} \tag{7}$$

The eigenvalues $[\Omega^2_{k+1}]$ and the associated eigenvectors $[\ Q_{k+1}\ ]$ need to be arranged in the ascending orders (for example $\Omega^2_1 < \Omega^2_2 < \Omega^2_3 < \ \ldots\ )$

Step 8: Find an improved approximation to the eigenvectors

$$[Y_{k+1}]_{N\times q}\ =\ [\bar{Y}_{k+1}]_{N\times q}[Q_{k+1}]_{q\times q} \tag{8}$$

Step 9: Check for convergence. The iterative process will be stopped if either convergence is achieved, or the maximum number of iteration (= 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:

$$K\,\phi = \omega^2 M \phi \tag{9}$$

or

$$K_\sigma \phi = \omega_\sigma^2 M \phi \tag{10}$$

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

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

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & & \\ & & & \beta_m & \\ & & \beta_m & \alpha_m & \end{bmatrix} \tag{11}$$

through the following three-term recurrence:

$$r_j = \beta_{j-1} q_{j-1} = K_\sigma^{-1} M q_j - \alpha_j q_j - \beta_j q_{j+1} \tag{12}$$

or in the matrix form:

$$[K_\sigma^{-1} M] Q_m - Q_m T_m = \{0.0....r_m\} = r_m e^T_m \tag{13}$$

$$T_{mz} = \theta z \tag{14}$$

where $e^T_m = (0,0,...1)$, Q_m is a Nxm 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

$$\omega_\sigma^2 = \frac{1}{\theta} \tag{15}$$

$$\phi = Q_m z \tag{16}$$

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

Step 1. Factorization : $K_o = L\,D\,L^T$
Form starting vector: $\gamma_o \neq 0$; $q_o = 0$

Step 2. Compute: $M\,\gamma o$

Step 3. Compute :

$$\beta_1 = \sqrt{\gamma_o{}^T M \gamma_o} \quad ; \quad q_1 = \frac{\gamma_o}{\beta_1}$$

Step 4. Compute : $P_1 = Mq_1$
Lanczos iteration
For j = 1,2,3, . . . , do

Step 5. $\epsilon_j = K_o^{-1} P_j$

Step 6. $\delta_j = \epsilon_j - \beta_j q_{j-1}$

Step 7. $\alpha_j = q_j^T M \delta_j = P_j^T \delta_j$

Step 8. $\gamma_j = \delta_j - \alpha_j q_j$

Step 9. $\Lambda_j = M \gamma_j$

Step 10.

$$\beta_{j+1} = (\gamma_j{}^T M \gamma_j)^{\frac{1}{2}} = \sqrt{\Lambda_j{}^T \gamma_j}$$

Reorthogonalization of q_{j+1}

Step 11.

$$q_{j+1} = \frac{\gamma_j}{\beta_{j+1}} \quad ; \quad P_{j+1} = \frac{\Lambda_j}{\beta_{j+1}}$$

Step 12. IF necessary solve Eq(14) : $T_j z = \theta z$
Converged? (If "No", then return to step 5)

Step 13. Eigenvector transformation: $\phi = Q_j z$

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_j, calculate

$$E_i = q'_i M q_j \qquad (i = 1,2,\ldots,j-1) \tag{17}$$

If $E_i > E$, then q_j should be orthogonal to q_i with respect to, M where E is a parameter related to the machine parameter E_o such that $1+E_o > 1$. Usually, E is taken as:

$$E = \sqrt{E_o} \tag{18}$$

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)

$$\text{ERROR}\ (i) \quad = \quad \left|\frac{\lambda_k - \theta_i}{\theta_i}\right| = \left|\beta_{j-1}\frac{Z_j^{(i)}}{\theta_i}\right| \quad \text{where} \quad i=1,2,....j \tag{19}$$

In Eq. (19), λ_K is the k-th exact eigenvalue and θ_i is the i-th computed eigenvalue. $Z_j^{(i)}$ is the j-th element of vector $Z^{(i)}$. If ERROR(i) < RTOL, for i = 1,2 ..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

$$[K] = [L]\,[D]\,[L]^T \tag{20}$$

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

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

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

$$[K] = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \tag{22}$$

will lead to the following factorized matrix

$$[U] = \begin{bmatrix} 2 & -\frac{1}{2} & 0 \\ & \frac{3}{2} & -\frac{2}{3} \\ & & \frac{1}{3} \end{bmatrix} \tag{23}$$

From Eq. (23), one can readily identify

$$[D] = \begin{bmatrix} 2 & 0 & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \tag{24}$$

and

$$[L]^T = \begin{bmatrix} 1 & -\frac{1}{2} & 0 \\ & 1 & -\frac{2}{3} \\ & & 1 \end{bmatrix} \tag{25}$$

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

$$[K] = \begin{bmatrix} 11. & 0. & 0. & 1. & 0. & 2. \\ & 44. & 0. & 0. & 3. & 0. \\ & & 66. & 0. & 4. & 0. \\ & & & 88. & 5. & 0. \\ & & & & 110. & 7. \\ & & & & & 112. \end{bmatrix} \tag{26}$$

For the data shown in Eq. (26), it can be easily shown that the factorized matrix [U] will have the following form:

$$[U] = \begin{bmatrix} x & 0 & 0 & x & 0 & x \\ & x & 0 & 0 & x & 0 \\ & & x & 0 & x & 0 \\ & & & x & x & F \\ & & & & x & x \\ & & & & & x \end{bmatrix} \tag{27}$$

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:

$$[\bar{K}_s] = \begin{bmatrix} x & 0 & 0 & x & 0 & x \\ & x & 0 & F & x & F \\ & & x & F & x & F \\ & & & x & x & F \\ & & & & x & x \\ & & & & & x \end{bmatrix} \tag{28}$$

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:

$$[\bar{K}_s] = \begin{bmatrix} x & F & F & x & F & x \\ & x & F & F & x & F \\ & & x & F & x & F \\ & & & x & x & F \\ & & & & x & x \\ & & & & & x \end{bmatrix} \tag{29}$$

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:

$$\text{ISTARTROW} \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 = N + 1 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{Bmatrix} \tag{30}$$

$$\text{ICOLNUM} \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = NCOEF \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{Bmatrix} \tag{31}$$

$$\text{DIAG} \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = N \end{Bmatrix} = \begin{Bmatrix} 11. \\ 44. \\ 66. \\ 88. \\ 110. \\ 112 \end{Bmatrix} \tag{32}$$

$$\text{AK} \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = NCOEF \end{Bmatrix} = \begin{Bmatrix} 1. \\ 2. \\ 3. \\ 4. \\ 5. \\ 7. \end{Bmatrix} \tag{33}$$

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
- 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.
- 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.
- DIAG(i): Numerical values of the diagonal term of [K]. The dimension for this real array is N.
- 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):

$$\text{JSTARTROW} \left\{ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7=N+1 \end{array} \right\} = \left\{ \begin{array}{c} 1 \\ 3 \\ 4 \\ 5 \\ 7 \\ 8 \\ 8 \end{array} \right\} \tag{34}$$

$$\text{JCOLNUM} \left\{ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7=NCOEF2 \end{array} \right\} = \left\{ \begin{array}{c} 4 \\ 6 \\ 5 \\ 5 \\ 5 \\ 6 \\ 6 \end{array} \right\} \tag{35}$$

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

- •NCOEF2: The number of nonzero, off-diagonal terms of the factorized matrix [U]
- •JSTARTROW(i): Starting location of the first nonzero, off-diagonal term for the ith row of the factorized matrix [U]. The dimension for this integer array is N+1.
- •JCOLUMN(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 > > NCOEF.

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

Step 1: Consider each ith 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 $1 \leq j \leq i-1$) 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 :

$$\text{ICHAINL}\begin{Bmatrix}1\\2\\.\\.\\N\end{Bmatrix} = \{0\} \quad \text{and} \quad \text{LOCUPDATE}\begin{Bmatrix}1\\2\\.\\.\\N\end{Bmatrix} = \{0\}$$

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

$$\text{ICHAINL}(4) = 1 \tag{36}$$

$$\text{ICHAINL}(1) = 1 \tag{37}$$

$$LOCUPDATE(i=1) = 1 \tag{38}$$

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

$$ICHAINL(5) = 2 \tag{39}$$

$$ICHAINL(2) = 2 \tag{40}$$

$$LOCUPDATE(i=2) = 3 \tag{41}$$

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:

$$ICHAINL(3) = 2 \tag{42}$$

$$ICHAINL(2) = 2 \tag{43}$$

$$LOCUPDATE(i=3) = 4 \tag{44}$$

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

$$LOCUPDATE(1) = LOCUPDATE(1) + 1 \tag{45}$$

Hence,

$$LOCUPDATE(1) = 1 + 1 = 2 \text{ (please refer to equation 38)} \tag{46}$$

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

$$ICHAINL(3) = 2 \tag{47}$$

$$ICHAINL(2) = 2 \tag{48}$$

$$LOCUPDATE(i=3) = 4 \tag{49}$$

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	(39, repeated)
ICHAINL (2)	= 4	(48, repeated)
ICHAINL (4)	= 3	(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.c..... 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.c..... 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
7.1              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 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

$$[K] = \begin{bmatrix}
x & x & x & & & x & & & x & x & x & & & x & x \\
 & x & x & & & x & & & x & x & x & & & x & x \\
 & & x & & & x & & & x & x & x & & & x & x \\
 & & & x & x & & x & x & & & & & & x & \\
 & & & & x & & x & x & & & & & & x & \\
 & & & & & x & x & x & F & F & F & & & x & F \\
 & & & & & & x & x & x & x & F & & & x & x \\
 & & & & & & & x & x & x & F & & & x & x \\
 & & & & & & & & x & x & F & & & x & x \\
 & & & & & & & & & x & F & & & x & x \\
 & & & & & & & & & & & x & x & x & x \\
 & & & & & & & & & & & & x & x & x \\
 & & & & & & & & & & & & & x & x \\
 & & & & & & & & & & & & & & x
\end{bmatrix} \tag{50}$$

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

$$\text{MASTER} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14=N \end{Bmatrix} = \begin{Bmatrix} 3 \\ 0 \\ 0 \\ 2 \\ 0 \\ 1 \\ 4 \\ 0 \\ 0 \\ 0 \\ 4 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \tag{51}$$

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.)

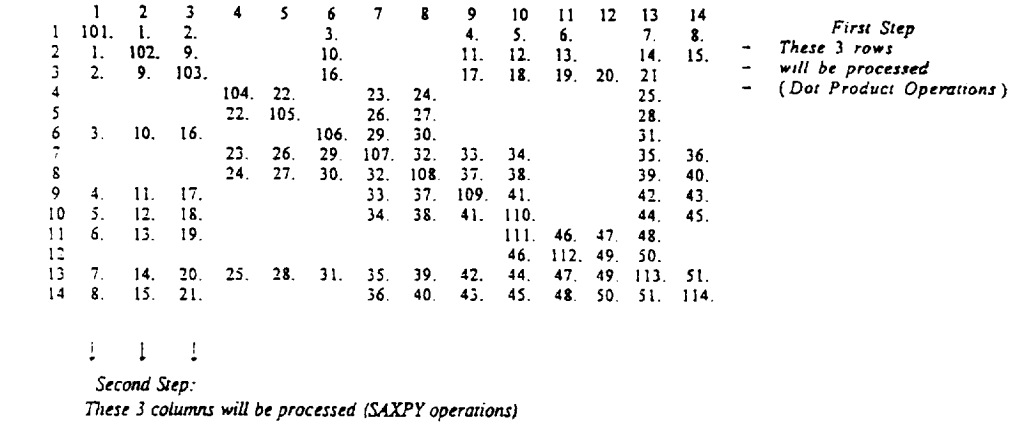


Figure 1: Sparse Matrix-Vector Multiplication's With Unrolling Strategies

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.

```
1.c ..... Assuming row 1 has been factorized earlier
2.        Do 11 I=2, N
3.        Do 22 K=Only those previous "master" rows which have contributions to
                  current row I
4.1c .....Compute the multiplier(s) ( Note: U represents L^T)
4.2       NSLAVE DOF= MASTER (I) - 1
5.1       XMULT = U(K,I) / U(K,K)
5.2       XMUL_m = U(K+m),I)/U(K+m.K+m)
5.3c ..... m=1,2 ... SLAVE DOF
6          Do 33 J = appropriated column numbers of " master " row # K
7.1       U(I,J) = U (I,J) - XMULT * U(K,J)
7.2                        - XMULm *U(K+m,J)
8     33  CONTINUE
9.1       U(K.I) = XMULT
9.2       U(K+m,I) = XMULm
10.   22  CONTINUE
11    11  CONTINUE
```

Table 5 : Pseudo FORTRAN Skeleton Code For Sparse LDL^T 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 :

$$\text{R.E.N.} = \frac{\|K\phi - \lambda M\phi\|}{\|K\phi\|} \tag{52}$$

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, where as 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)

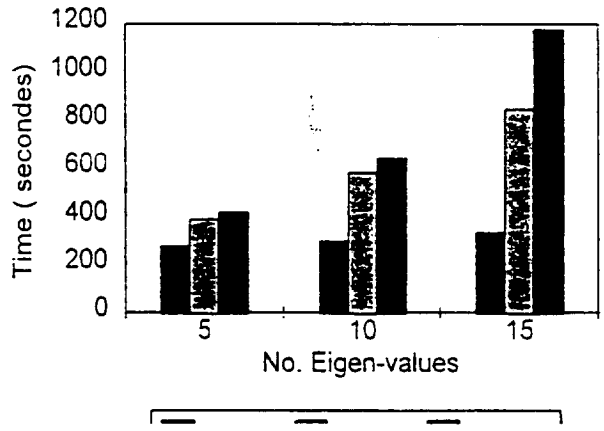


Figure 3: Exxon Model
(USTSU31, Sun-Sparc20 Workstation)

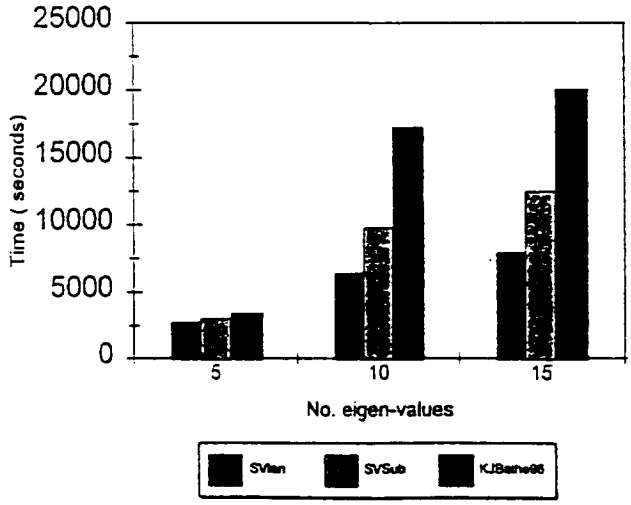


Figure 4: HSCT Aircraft model

(Stretch, IBM-R600/590 Worksation)

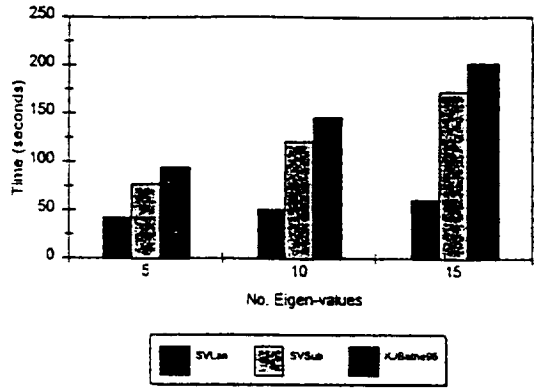
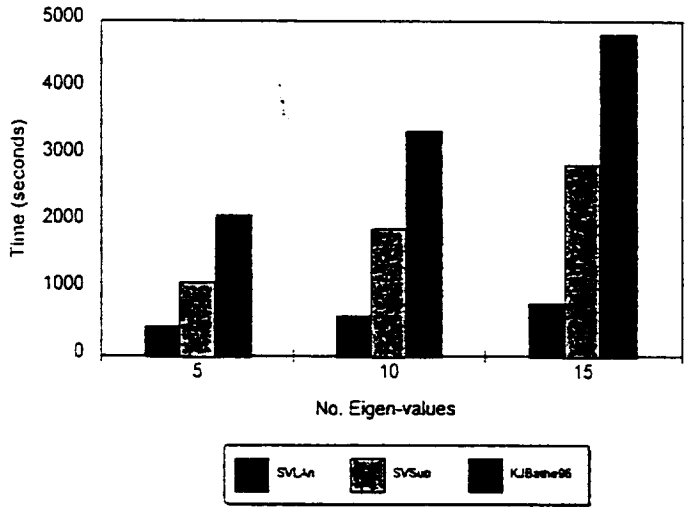


Figure 5: Aircraft model

(Rhino, ODU Sun Sparc-20 Worksation)



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

1. Bathe, J., Finite Element Procedures, Prentice-Hall, Englewood Cliffs, New Jersey (1996)
2. Tong, P., and Rossettos, J.N., Finite Element Method: Basic Technique and Implementation, the MIT Press, Cambridge, Massachusetts, and London, England
3. Hughes, T.J.R., The Finite Element Method, Prentice-Hall, Englewood Cliffs, N.J. (1987)
4. Zinkiewicz, O.C., and Taylor, R.L., The Finite Element Method In Structural and Continuum Mechanics, McGraw-Hill, Vols. 1 and 2 (1989/1990).
5. Hang, E.J., and Arora, J.S., Applied Optimal Design, John Wiley & Sons (1979)
6. Bathe, K.J., "Solution Methods of Large Generalized Eigenvalue Problems in Structural Engineering, "Report UC SESM 71-20, Civil Engineering Department, University of California, Berkeley, 1971.
7. Bathe, K.J., "Convergence of Subspace Iteration," in Formulations and Numerical Algorithms in Finite Element Analysis, MIT Press, Cambridge, Ma, pp.575-598, 1977.
8. Bathe, and Ramaswamy, S, "An Accelerated Subspace Iteration Method," Computer Methods in Applied Mechanics and Engineering, Vol. 23, pp.313-331, 1980.
9. Bathe, and Wilson, E.L., "Eigensolution of Large Structural Systems with Small Bandwidth," ASCE Journal of Engineering Mechanics Division, Vol. 99, pp. 467-479, 1973.
10. Lanczos, "An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operator," Journal of Research of the National Bureau of Standards, 45 (1950), 255-281.
11. Golub, Underwood, R., and Wilkinson, J.H., "The Lanczos Algorithm for the Symmetric Ac = lBc Problem," Tech. Rept. STAN-CS-72-720, Computer Science Department, Stanford University, 1972.
12. Parlett and Scott, D., "The Lanczos Algorithm with Selective Orthogonalization," Mathematics of Computations, 33 No. 145 (1979), 217-238.
13. Nour-Omid, Parlett, B.N., and Tayor, R.L., "Lanczos versus Subspace Iteration for solution of Eigenvalue Problems," International Journal for Numerical Methods in Engineering, 19 (1983), 859-871.
14. Qin, J., Nguyen, D.T., and Zhang, Y., "A Parallel-Vector Lanczos Eigensolver for Structural Vibration problems," in: Proceedings of the Fourth International Conference on Recent Advances in Structural Dynamics, July 15-18, 1991, London, UK.

15. Nguyen, D.T. Qin, J., Chang, T.P.Y. and Tong, P., "Efficient Sparse Equation Solver With Unrolling Strategies For Computational Mechanics," submitted to Journal "Mathematics and Mechanics of Solids."
16. George and Liu, W.H.,"Computer Solution of Large Sparse Positive Definite Systems" Prentice-hall, Englewood Cliffs, N.J. (1981).
17. Cuthill, and McKee, J., "Reducing The Bandwidth of Sparse Symmetric Matrices," Proceedings of 24th National Conference, Association for Computing Machinery, pp.157-172 (1969).
18. Gibbs, Poole, Stockmeyer, Jr. and P.K., "An Algorithm For Reducing the Bandwidth and Profile of a Sparse Matrix," SIAM Journal on Numerical Analysis, Vol. 13, pp.236-250 (1976).
19. Lewis, Pfyton, B.W. and Pothen, A.,"A Fast Algorithm For Recording Sparse \Matrices For Parallel Factorization," SIAM J. Sci. Statist. Comput., 6, pp. 1146-1173 (1989).
20. Liu, "Reordering Sparse Matrices For Parallel Elimination," Tech. Report 87-01, Computer Science, York University, North York, Ontario, Canada (1987).
21. Storaasli, O.O., Nguyen, D.T., and Agarwal, T.K., "The Parallel Solution of Large-Scale Structural Analysis Problems on Supercomputers," AIAA Journal, Vol. 28, No.7, pp. 1211-1216 (July 1990).
22. Maker, B.N., Qin, J. and Nguyen, D.T., "Performance of NIKE3D with PVSOLVE On Vector and Parallel Computers," Computing Systems In Engineering Journal (1995).
23. Wang, Chang, T.Y.P., and Tong,P., "Nonlinear Deformation Responses of Rubber Components by Finite Element Analysis," Computational Mechanics '95: Theory and Applications Proceedings of the International Conference on Computational Engineering Science. July 30-Aug. 3 '95, Hawaii, USA (Volume 2, pp.3135-3140).
24. Chang, Saleeb, A.F., and Li, G., "Large Strain Analysis of Rubber-like Materials Based On a Perturbed Lagrangian Variational Principle," J. Comput. Mech., Vol. 8, (1991), pp.221-233.
25. Gunderson, "Fatigue Life of TLP Flex-elements," 24th Annual OTC Conference, Houston, Texas, May 4-7, 1992.

```
      PROGRAM SPARSEPACK97
C*********************************************************************
C
C     EQUATION-EIGEN SOLVER FOR SPARSE POSITIVE DEFINITE SYSTEMS
C
C     H. Runesha : runesha@cee.odu.edu
C     Prof. Duc Nguyen : nguyen@cee.odu.edu
C
C     Last update: Dec 20, 1997
C     - J'ai ajoute le shift et le consistant mass
C                  October 26, 1998
C     ---> Because J. Qin tries to re-use memory, error messages and/or
C          wrong answers occur whenever ncoff is less than approx. 2*neq
C          Although this case rarely occurs in real, practical problems
C          (may only occur for "artificial, small scale" examples, however
C          it is annoying to have this "bugs". These bugs have been fixed
C          in this SPARSEPACK version of eigen-solution
C
C          Two (2) small changes have been made in spamain.f, and spaldln.f
C          which involve mtota and if (ncoff ...)
C
C     ---> For structures with rigid body motions (such as "floating" structures,
C          without any supports), the "correct error norm" should be close
C          to 1.0 (instead of close to 0.0), because
C          error norm = ||K*Phi - Lamda*M*Phi|| / ||K*Phi||
C          since Lamda = eigen-value = 0.0 (corespond to rigid body motion),
C          hence  error norm = ||K*Phi - 0.0|| / ||K*Phi|| = close to 1.0
C          For "float structures" we need to apply "shift factor" to avoid
C          "singular" stiffness matrix
C
C*********************************************************************
C
C     NOTES:
C
C         MREAD  < 0 Read K.*
C                > 0 Read fort.*
C         NREORD = 0 no reordering
C                = 3 MMD reordering
C         LUMP   = 0 CONSISTANT MASS ( or Lump.ne.0)
C                = 1 LUMPED MASS
C         NEIG   = number of desired eigenvalues and corresponding
C                  eigenvectors
C         N      = number of equations
C         NCOFF  = number of nonzeros coefficients
C         ISHIFT = 0 no shift
C                = .NE.0 perform a shift of value "ishift"
C         IBLOCK = -1 : Subspace iteration
C                =  0 : Regular Lanczos
C                =  1,2,3 : block Lanczos ( of Block = iblock)
C         ITIME  = 0  save memory
C                = 1  save time
C
C
C
C*********************************************************************

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

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

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

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

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

      open(unit=7,file='K.INFO',status='old',form='formatted')
      read(7,115) title
115   FORMAT(A60)
      read(7,*) nreord,neig,lump,n,n2,NCOFF,itime,ishift,iblock,mread
C     write(*,*) nreord,neig,lump,n,n2,NCOFF,itime,ishift,iblock,mread
C     close (7)
C_____
```

```
      write(23,*)  title
      write(23,*)
      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
      write(23,*)
c_____________

c     if(iblock.eq.0.and.lump.ne.1) then
c     write(23,*)'Sorry! The old Lanczos path can not deal with
c    1 Consistant Mass !'
c     stop
c     endif

      if(iblock.gt.3) then
       write(23,*)'Sorry! BLOCK size must less than 4 ! '
       stop
      endif

c     if(iblock.eq.0.and.ishift.ne.0) then
c       write(23,*)'Sorry! No shift is allowed for the Old Lanczos !'
c       stop
c     endif

c
c     Read Data
c
      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
c
c     REORDERING
c
C     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

c
c     SOLVER
c
c     call sparse(n,ncoff,neig,mtot/2,mtot/2,a(1),a(1+mtot/2))
C     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)
      time1=time1 - time0
      if(neig.ne.0) then
      write(23,*)'***TOTAL CPU FOR EIGENSOLUTION       = ', time1
      write(23,*)'***(This time including norm check & I/O )***'
      write(23,*)'MTOTI = ',mtoti,'     MTOTA = ',mtota
      endif
      stop
      end
c
```

```
c***********************************************************************
c
        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)=1
          do 10 i=2,n+1
10        iq(i)=iq(i-1)+iq(i+n)
        rewind(15)
        write(15)(iq(i),i=1,n+1)
        else
        if(n.gt.0) stop
        endif
        return
        end
c
c***********************************************************************
c
           SUBROUTINE QREAD(neq,ncoeff,mtot,a,mread,neig,lump,ishift)
           real*8 a(1)
           call iread(neq,ncoeff,mtot,a,mread,lump)
c          dec 20 1997 : Runesha
c          if(lump.ne.1.or.ishift.ne.0) then
c          call rreadpierrot(neq,ncoeff,neig,lump,ishift,a(1),a(n+1),
c    +          a(ncoeff+n+1),a(ncoeff+2*n+1),a(ncoeff+3*n+1))
c          else
           call rread(neq,ncoeff,mtot,a,mread,neig,lump)
c          endif
           return
           end
c
c***********************************************************************
c
           SUBROUTINE RREAD(neq,ncoeff,mtot,a,mread,neig,lump)
           real*8 a(1)
         if(mread.lt.0) then
        OPEN(unit=54,file='K11.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')
         else
        OPEN(unit=55,file='K.DIAG.B',
       3form='unformatted',status='old')
        OPEN(unit=54,file='K11.COEFS.B',
       4form='unformatted',status='old')
        OPEN(unit=56,file='K.RHS.B',
       5form='unformatted',status='old')
        OPEN(unit=57,file='K.DMASS.B',
       5form='unformatted',status='old')
        if(neig.ne.0.and.lump.ne.1) OPEN(unit=58,file='K.CMASS.B',
       1form='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)
c          read(42+jfile,9901) (a(k),k=1,length)
9901     format(6E12.5)
         else
           read(42+jfile) (a(k),k=1,length)
         endif
                rewind(jfile)
                write(jfile)(a(k),k=1,length)

100      continue
         if(mread.lt.0) then
             read(55,*)(a(i),i=1,neq)
```

```
c           read(55,9901)(a(i),i=1,neq)
          else
              read(55)(a(i),i=1,neq)
          endif

          rewind(13)
          write(13)(a(i),i=1,neq)
c         write(*,*) 'Inside rread AD=', (a(i),i=1,neq)

           if(mread.lt.0) then
               read(56,*)(a(i),i=1,neq)
c              read(56,9901)(a(i),i=1,neq)
           else
               read(56)(a(i),i=1,neq)
           endif
          rewind(14)
           write(14)(a(i),i=1,neq)
           if(neig.eq.0) return
          if(mread.lt.0) then
              read(57,*)(a(i),i=1,neq)
c             read(57,9901)(a(i),i=1,neq)
          else
              read(57)(a(i),i=1,neq)
          endif
          rewind(10)
           write(10)(a(i),i=1,neq)
         if(neig.ne.0.and.lump.ne.1) then
           if(mread.lt.0) then
               read(58,*)(a(i),i=1,ncoeff)
           else
               read(58)(a(i),i=1,ncoeff)
           endif
          rewind(17)
          write(17)(a(i),i=1,ncoeff)
          endif
         return
         end
c
c*************************************************************************
c

            SUBROUTINE IREAD(neq,ncoeff,mtot,ia,mread,lump)
            integer ia(1)
         if(mread.lt.0) then
         OPEN(unit=52,file='K.PTRS',form='formatted',status='old')
         OPEN(unit=53,file='K11.INDXS',form='formatted',status='old')
         else
         OPEN(unit=52,file='K.PTRS.B',form=
        1'unformatted',status='old')
         OPEN(unit=53,file='K11.INDXS.B',
        2form='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
c
c*************************************************************************
c
          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='K11.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')
c      OPEN(unit=57,file='K.DMASS',form='formatted',status='old')
c      if(neig.ne.0.and.lump.ne.1) OPEN(unit=58,file='K.CMASS',
c     1form='formatted',status='old')

c
c       perform also a shift on the diag values
c
        write(*,*) 'ISHIFT =', ishift
        read(54,*) (an(i),i=1,ncoef)
        read(55,*) (ad(i),i=1,n)
        write(*,*) ' AD before =', (ad(i),i=1,n)
        read(56,*) (b(i),i=1,n)
c       read(57,*) (dm(i),i=1,n)
        close(54)
        close(55)
        close(56)
c       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
        do i=1,n
        ad(i)=ad(i)+float(ishift)*dm(i)
        write(*,*) 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,ncoef)
        close(58)
        rewind(17)
        write(17) (am(i),i=1,ncoef)
         if(ishift.ne.0) then
         do i=1,ncoef
            an(i)=an(i)+float(ishift)*am(i)
         enddo
         endif
        endif
        ENDIF
c
c       write on fort.*
c
        rewind(13)
        write(13) (ad(i),i=1,n)
        write (*,*) ' Inside rread  AD =', (ad(i),i=1,n)
        rewind(12)
        write(12) (an(i),i=1,ncoef)
        rewind(14)
        write(14) (b(i),i=1,n)
        return
        end

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

        subroutine sparse(n,ncoff,neig,lump,iblock,mtoti,mtota,iq,a,
     1                    ishift)
        implicit real*8 (a-h,o-z)
        real*8 a(1)
```

```
      integer iq(1)
      ierr=0
c.....purpose: sparse gauss version equation solver
c.....         for system of symmetrical equations.
c..... J. Qin     Dec.28, 1992
c..... H. Runesha   June 1996

c*********************************************************************
C***    UDU factorization  *******
      mtoti=mtoti*2
      do 81 i=1,ncoff+n+n
81    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),
     1            a(2*n+ncoff+1),ncof2,iqmax)
      call cputime(time01)
      write(23,*)'neq                      =  ',n
      write(23,*)'before fill in, ncoff =  ',ncoff
      write(23,*)'after  fill in, ncof2 =  ',ncof2
      write(23,*)'Total integer memory used  =  ',ncoff+ncof2+
     1            6*n+3
      write(23,*)'Total real    memory used  =  ',ncoff+ncof2+4*n
      if(mtoti.lt.(ncoff+3+6*n+ncof2)) then
        write(23,*) 'The integer array iq should >  ',ncoff+3+6*n
     1              +ncof2
      ierr=1
      endif
      if(mtota.lt.(4*n+ncoff+ncof2)) then
        write(23,*) 'The real array a  should  >   ',4*n+ncoff+ncof2
      ierr=2
      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),
     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
     1             +ncof2))
C
C***  if neig.ne.0.and.ishift.ne.0) do shifting: K~=K+ishift*M
C
cpier
c       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
C
        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
```

```
        ENDIF
c       endif
cpier
        call cputime(time031)
c      subroutine Numfa1(n,ia,ja,ad,an,iu,ju,di,un,ip,iup,t1,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+ncof2),iq(ncoff+3*n+3+ncof2),
     1 a(3*n+ncoff+ncof2+1),iq(ncoff+4*n+3+ncof2)
     1 ,iopf)
        call cputime(time04)
       IF(NEIG.NE.0) THEN
        if(iblock.eq.-1) then
        write(23,*) 'IBLOCK =',iblock

c       ************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
c       AM
        rewind(17)
        read(17)(a(i01-1+i),i=1,ncoff)  ! read am for consist. mass
        endif
       iiq1=min(neig+8,2*neig)
       iiq=iiq+1
       iiq=min(iiq1,n)
c      iiq=n
       mtot=mtoti+mtota
       i02=i01+ncmass
       i03=i02+iiq*iiq
       i04=i03+iiq*iiq
       i05=i04+iiq*iiq
       i06=i05+iiq
       i07=i06+n
       i08=i07+n
       i09=i08+n
       i10=i09+n*iiq
       if(i10.gt.mtota) write(*,*)  'NOT ENOUGH MEMORY '

       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)

       return

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

        if(iblock.ge.1) then
        if(iblock.gt.6) iblock = 6

C#########################################################################
C
C*********** Block Lanczos eigensolver:
C** Iq(): 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))

        isize = min((neig*5*iblock)/iblock,(n/iblock)*iblock)  + 1
        isize=(isize/iblock)*iblock
        ineed = isize*isize*6 + iblock*iblock*6 + isize*2
c       write(*,*)'ileft,ineed = ',ileft,ineed
c       if(ineed.lt.ileft) then
c       i00 = 2*n+6+ncoff+ncof2                 ! T(isize,isize)
c       i01 = i00+isize*isize*2                 ! B(isize,isize)
c       i02 = i01+isize*isize*2                 ! vec(isize,isize)
c       i03 = i02+isize*isize*2                 ! alfa(iblock,iblock)
c       i04 = i02+iblock*iblock*2               ! beta(iblock,iblock)
c       i05 = i02+iblock*iblock*2               ! betai(iblock,iblock)
c       i06 = i05+iblock*iblock*2               ! eig(isize)
c       else
c       write(23,*)'Increase mtoti to  : ',ineed+2*n+5+ncoff+ncof2
c       stop
```

```
c       endif
C** a(): 1->n = ad; n->2*n = RHS; 2*n+1-> = an; di; un; t1;
          j02 = 2*n+ncoff + 1                              ! di, unchanged
       if(lump.ne.1) then
C***  consistant Mass matrix, using ad,an space for: dmass,am
        j00 = 1                                           ! dmass
        j01 = j00 + n                                     ! am
        j03 = 3*n+ncoff+ncof2+1                           ! r(n,iblock)
        j04 = j03 + n*iblock                              ! p(n,iblock)
        j05 = j04 + n*iblock                              ! tem(n,iblock)
        j06 = j05 + n*iblock                              ! Q
        else
C**  lumped mass matrix:
        ibmax = (2*n+ ncoff)/(3*n)
c       write(*,*)'ibmax,iblocki,isize = ',ibmax,iblock,isize
        if(ibmax.ge.iblock) then
          j00 = 3*n+ncoff+ncof2+1                         ! dmass
          j01 = j00 + n                                   ! am
           j03 = 1                                        ! r(n,iblock)
           j04 = j03 + n*iblock                           ! p(n,iblock)
           j05 = j04 + n*iblock                           ! tem(n,iblock)
           j06 = j01 + 1                                  ! Q
        else
           j00 = 1                                        ! dmass
           j01 = 1+n                                      ! am
           j03 = 2+n                                      ! r(n,iblock)
           j04 = 3*n+ncoff+ncof2+1                        ! p(n,iblock)
           j05 = j04 + n*iblock                           ! tem(n,iblock)
           j06 = 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 = i00+isize*isize                 ! B(isize,isize)
         i02 = i01+isize*isize                 ! vec(isize,isize)
         i03 = i02+isize*isize                 ! alfa(iblock,iblock)
         i04 = i02+iblock*iblock               ! beta(iblock,iblock)
         i05 = i02+iblock*iblock               ! betai(iblock,iblock)
         i06 = i05+iblock*iblock               ! eig(isize)
              if(i06+isize.gt.mtota) then
                 write(23,*)'Increase mtota to :  ',i06+isize
                 stop
              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(j00),
     2       a(i04),a(i05),a(i06),a(i02),a(i03),a(j06),
     3       a(i00),a(i01),a(j01),iq(1),iq(2+n),lump,ncoff,ishift)
        return
        endif
C#########################################################################
C
C******    for regular Lanczos:
        ax = neig**2
        bx = 3*neig
        cx = n - ncoff
        if(bx*bx-4.0*cx*ax.le.0.0) then
           ix1 = 1
           go to 1995
        endif
        x1 = (-bx+sqrt(bx*bx-4.0*cx*ax))*0.5/ax
        ix1=x1
1995    lanmax = min(8*neig,neig*ix1)
        write(23,*)'LANMAX = ',lanmax,x1*neig
c       lanmax = 4*neig
c       item = neig*(12 + 16*neig) + n
c       if(item.gt.ncoff) then
c          lanmax = 3*neig
           item   = neig*(9 + 9*neig) + n
c       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
c       Pierrot , je fermer ca mais il faut le verifier
c       write(23,*)'Increase memory for: VEC(lanmax,lanmax)&EIG,ERR,TEM',
```

```
c     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
        ij0=j00+1+3*lanmax+lanmax**2
        if(j00+lanmax*3+lanmax**2+n*lanmax.gt.mtota) then
         write(23,*)'Increase memory mtota to : ',j00+3*lanmax+
      1lanmax**2
        stop
        endif
       write(23,*)'lanmax,j00,ij0,ncoff,ncof2=',lanmax,j00,ij0,ncoff,
      1ncof2
        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),iq(n+2+
      1ncoff),iq(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
c        pierrot a ajouter ceci pour introduire ia ja et am
         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+
      1  lanmax**2+ncoff
         stop
         endif
       CALL SP2LAN2(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),iq(n+2+
      1ncoff),iq(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))
         endif

        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),iq(n+2+
      1ncoff),iq(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
      2ncof2),a(1+2*n),a(1+3*n),a(1+3*n+lanmax),
      3a(1+3*n+2*lanmax),a(1+3*n+3*lanmax),ncoff,ncof2,iq(j01),a(ij0)
      +  ,lump,ishift)
         else
c        pierrot a ajouter ceci pour introduire ia ja et am
         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+
      1  lanmax**2+ncoff
         stop
         endif
       CALL SP2LAN2(n,lanmax,neig,a(ncoff+3*n+1),a(2*n+ncoff+1),iq(n+2+
      1ncoff),iq(ncoff+2*n+3),a(1),a(1+n),a(3*n+1+ncoff+1+
      2ncof2),a(1+2*n),a(1+3*n),a(1+3*n+lanmax),
      3a(1+3*n+2*lanmax),a(1+3*n+3*lanmax),ncoff,ncof2,iq(j01),a(ij0)
      +    , a(ij0+lanmax*n), lump,ishift,iq(1),iq(n+2))
        endif


        return
       ELSE
        call fbe(n,iq(n+2+ncoff),iq(ncoff+2*n+3),a(2*n+ncoff+1),
      1          a(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+
      1 ncof2+1),a(2*n+ncoff+1))
        call cputime(time06)
        sum=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=j1+i
      jj = 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
      sum=sum+abs(a(j))
      sum2=sum2+(a(j))
2000   continue
      axbn = sqrt(axbn)
      bnorm = axbn/sqrt(bnorm)
      write(23,*) 'The Max. Displacement = ',a(j1+imax),'  at the
     1  ',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      || Ax - b ||             =',axbn
      write(23,*)'The    || Ax - b || / ||b||       =',bnorm
      write(23,*)'Time for error norm check         =',time06-time05
      write(23,*)'Time for reading files            =',timer1-timer0
      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
     1            +time05-time02
      write(23,*)'Total Operations in Factorization:',iopf
      write(23,*)'Total Operations in Forw/Back.    :',iopfb
c     write(23,*)'Mflops   for factorization        =',float(iopf)*
c    1            0.000001/(time04-time031)
      loop = 8
      write(23,*)'LOOP UNROLLING LEVEL              =',LOOP
c     write(23,*)'Mflops   for forward/backward     =',float(iopfb)*
c    1            0.000001/(time05-time04)
      ENDIF
      return
      end
      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 10 i = 1, n
       iu(i) = 0
10     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,
     1  '** 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
20     iu(jj) = i
       jp=jp+(iab-iaa)+1
       min=ja(iaa)
30     last = ip(i)
       if( last.eq.0 ) go to 60
       l = last
40     l = ip(l)
       lh = l + 1
       iua = iu(l)
       iub = iu(lh) - 1
       if ( lh.eq. i ) iub = jpi - 1
```

```
        iu(i) = i
CDIR$ IVDEP
        do 50 j = iua, iub
        jj = ju(j)
        if ( iu(jj).eq.i ) go to 50
        ju(jp) = jj
        jp = jp + 1
        iu(jj) = i
        if( min.gt.jj ) min = jj
50      continue
        if ( jp.eq.jpp ) go to 70
        if ( l.ne.last ) go to 40
60      if ( min.eq.nh ) go to 90
70      l = ip(min)
        if ( l.eq.0 ) go to 80
        ip(i) = ip(l)
        ip(l) = i
        go to 90
80      ip(min) = i
        ip(i) = i
90      iu(i) = jpi
        iu(n) = jp
        iu(nh) = jp
        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
10      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)
20      x(jj) = x(jj) - xx*un(i)
        iopfb=iopfb+2*(iub-iua+1)
30      continue
40      continue
        do 41 jj = 1,n
41      x(jj) = x(jj)*di(jj)
        iopfb=iopfb+n
        k = nm
50      iua = iu(k)
        iub = iu(k+1) - 1
        if ( iub.lt.iua ) go to 70
        xx = x(k)
CDIR$ IVDEP
        do 60 i = iua, iub
        jj=ju(i)
60      xx = xx - un(i)*x(jj)
        x(k) =xx
        iopfb=iopfb+2*(iub-iua+1)
70      k = k -1
        if ( k.gt.0 ) go to 50

        return
        end
       subroutine transa(n,m,ia,ja,iat,jat)
       implicit real*8 (a-h,o-z)
       integer ia(1),ja(1),iat(1),jat(1)
       mh = m + 1
       nh = n + 1
       do 10 i = 2, mh
10     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
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)
40      do 60 i = 1, n
        iaa = ia(i)
        iab = ia(i+1) - 1
        if ( iab.lt.iaa ) go to 60
CDIR$ IVDEP
        do 50 jp = iaa, iab
        j = ja(jp) + 1
        k = iat(j)
        jat( k ) = i
50      iat(j) = iat(j) + 1
60      continue
       call tran2(n,m,iat,jat,ia,ja)
        return
        end
        subroutine tran2(n,m,ia,ja,iat,jat)
        implicit real*8 (a-h,o-z)
        integer ia(1),ja(1),iat(1),jat(1)
        mh = m + 1
        nh = n + 1
        do 10 i = 2, mh
10      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
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)
40      do 60 i = 1, n
        iaa = ia(i)
        iab = ia(i+1) - 1
        if ( iab.lt.iaa ) go to 60
CDIR$ IVDEP
        do 50 jp = iaa, iab
        j = ja(jp) + 1
        k = iat(j)
        jat( k ) = i
50      iat(j) = iat(j) + 1
60      continue
        return
        end
c
c*****************************************************************************************
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*****************************************************************************************
c
       subroutine numfaq(n,ia,ja,ad,an,iu,ju,di,un,ip,iup,tl,isupd
      1  ,iopf)
       implicit 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),isupd(1)
       LOOP = 8
       iopf=0
       call numfa0(n,ia,ja,an,iu,ju,di,un)
       do 10 j = 1, n
10     ip(j) = 0
       do 130 i = 1,n
       ih = i + 1
       iua = iu(i)
       iub = iu(ih) - 1
```

```
CDIR$ IVDEP
      do 20 j = iua, iub
20    di(ju(j)) = un(j)
      di(i) = ad(i)
      LN = ip(i)
      if ( LN.eq.0 ) go to 90
50    L = ln
      ln = ip(L)
      ik = min(isupd(L),i-L)
      iend = (ik/LOOP)*LOOP
      iuc0 = iup(L)
      iuc00 = iuc0
      iup(L) = iuc0 + 1
      iud = iu(L+1) - 1
      idif = iud - iuc0
      iopf=iopf+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)
CDIR$ IVDEP
      do 1010 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)
     3                               - um7*un(j7+j)-um8*un(j8+j)
1010  continue
      iuc0 = iu(K+9) - 1 - idif
1000  continue
      ileft = ik - iend
      Lend = L+iend
      if ( ileft.eq.0 ) go to 1030
      go to (1011,1012,1013,1014,1015,1070,1080) 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
      j7 = iu(Lend+7) - 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)
CDIR$ IVDEP
      do 1076 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)
     3                               - um7*un(j7+j)
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)
      um5 = un(j5)  * di(Lend+4)
CDIR$ IVDEP
      do 1016 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)
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)
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
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
90    um = 1.00/di(i)
      iopf=iopf+1
      if ( iub.lt.iua ) go to 120
CDIR$ IVDEP
      do 100 j = iua,iub
100   un(j) = di(ju(j))*um
      if(isupd(i).eq.0) go to 130
      j = ju(iua)
      ip(i) = ip(j)
```

```
      ip(j) = i
120   iup(i) = iua
130   continue
      do 1900 j = 1,n
1900  di(j) = 1.0/di(j)
      return
      end
       subroutine supnode(n,iu,ju,isupd)
       integer iu(1),ju(1),isupd(1)
       do 10 i = 1,n
10     isupd(i) = 1
c      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-i) go to 100
       do 200 jj = k0+i0-i,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
C*************************************************
       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 10 i= 1, n
10     c(i) = ad(i)*b(i)
       do 30 i = 1,n
       iaa = ia(i)
       iab = ia(i+1) - 1
       if( iab.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
 20    di(ju(j)) = 0.0d0
      iaa = ia(i)
      iab = ia(ih) - 1
      if ( iab.lt.iaa ) go to 1000
CDIR$ IVDEP
      do 30 j = iaa, iab
 30    di(ja(j)) = an(j)
CDIR$ IVDEP
      do 100 j = iua,iub
 100   un(j) = di(ju(j))
1000  continue
130   continue
      return
      end
```

```
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine numfa1(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(*)
      dimension 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                                 RR(U)O.
c                  n              order of matrices A and U.
c     output:      un             numerical values of the nonzeros of
c                                 matrix U in RR(U)O.
c                  di             inverse of the diagonal matrix D.
c     working space:  ip          of dimension N.  Chained lists of rows
c                                 associated with each column.
c                     iup         of dimension N.  Auxiliary pointers to
c                                 portions of rows.
c                     di          is used as the expanded accumulator.
      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
      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 DI(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
c.....                             rows (any appropriated rows 1--->i-1)

   50 L=LN
      LN=IP(L)
      IUC=IUP(L)
      IUD=IU(L+1)-1
      UM=UN(IUC)*DI(L)
c.....begin of 3-rd (nested) inner-most loop: considering all APPROPRIATED
c.....                                        columns (any columns i--->n)
      DO 60 J=IUC,IUD
      JJ=JU(J)
   60 DI(JJ)=DI(JJ)-UN(J)*UM
      UN(IUC)=UM
      IUP(L)=IUC+1
      IF(IUC.EQ.IUD)GO TO 80
      J=JU(IUC+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
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.d0/DI(I)
      IF(IUB.LT.IUA)GO TO 120
      DO 100 J=IUA,IUB
  100 UN(J)=DI(JU(J))
      J=JU(IUA)
      JJ=IP(J)
      IF(JJ.EQ.0)GO TO 110
      IP(I)=IP(JJ)
      IP(JJ)=I
      GO TO 120
  110 IP(J)=I
      IP(I)=I
```

```
  120 IUP(I)=IUA
  130 CONTINUE
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      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(*)
c     DEFINITIONS
c     input:        ia,ja,an,ad     given matrix A in RR(U)U.
c                   iu,ju           structure of resulting matrix U in
c                                   RR(U)O.
c                   n               order of matrices A and U.
c     output:       un              numerical values of the nonzeros of
c                                   matrix U in RR(U)O.
c                   di              inverse of the diagonal matrix D.
c     working space:  ip            of dimension N.  Chained lists of rows
c                                   associated with each column.
c                     iup           of dimension N.  Auxiliary pointers to
c                                   portions of rows.
c                     di            is used as the expanded accumulator.

c     initialisation du vecteur IP, IP indique si il y a des lignes
c     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
c     ih,iuab,iub :ligne suivante, debut ligne i et fin ligne i
      IH=I+1
      IUA=IU(I)
      IUB=IU(IH)-1

      IF(IUB.LT.IUA)GO TO 40
c     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
c     commenter la ligne suivante
      IF(IAB.LT.IAA)GO TO 40
c     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)


c     voir si il y a une ligne au dessus qui va modifier la ligne I
c     Au depart IP est initialise a )
      LAST=IP(I)
      IF(LAST.EQ.0)GO TO 90
      LN=IP(LAST)
c     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-1) which contribute to modify I


c     Debut de la partie vecteur
   50 L=LN
      LN=IP(L)
      loop=2
      m= min(i-1,isupd(l))
      iend=(m/loop)*loop

      isbegin=1
      isend=1+iend-1
c     keep a copy of IUC1 and IUD1 of L pour la construction de IP
      IUCL=IUP(isbegin)
      IUDL=IU(isbegin+1)-1
```

```
      do is= isbegin,isend,2
      IUC1=IUP(is)
      IUD1=IU(is+1)-1
c     verifier si la 2 eme ligne commence au debut de la barriere supenode
      IUC2=IUP(is+1)
      UM1=UN(IUC1)*DI(is)
      UM2=UN(IUC2)*DI(is+1)
c.....begin of 3-rd (nested) inner-most loop: considering all APPROPRIATED
c.....                                        columns (any columns i--->n)
      DO 60 J=IUC1,IUD1
      JJ=JU(J)
   60 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iuc1+j)*um2
      UN(IUC1)=UM1
      un(IUC2)=um2
c     j'update les IUP mais pour les IP je ne le fait que pour tout le supernode
      IUP(is)=IUC1+1
      iup(is+1)=iuc2+1
      enddo

      do is=isend+1,m+1-1
      IUC1=IUP(is)
      IUD1=IU(is+1)-1
      UM1=UN(IUC1)*DI(is)
c.....begin of 3-rd (nested) inner-most loop: considering all APPROPRIATED
c.....                                        columns (any columns i--->n)
      DO 62 J=IUC1,IUD1
      JJ=JU(J)
   62 DI(JJ)=DI(JJ)-UN(J)*UM1
      UN(IUC1)=UM1
c     j'update les IUP mais pour les IP je ne le fait que pour tout le supernode
      IUP(is)=IUC1+1
      enddo


c     j'ai fini les supernode , maintenant j'update le IP
      IF(IUCL.EQ.IUDL)GO TO 80

c     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
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
   90 DI(I)=1.d0/DI(I)
      IF(IUB.LT.IUA)GO TO 120
      DO 100 J=IUA,IUB
  100 UN(J)=DI(JU(J))

c     addition
      if(isupd(i).eq.0) go to 120
c     if(isupd(i).eq.0) go to 130
      J=JU(IUA)
      JJ=IP(J)
      IF(JJ.EQ.0)GO TO 110
      IP(I)=IP(JJ)
      IP(JJ)=I
      GO TO 120
  110 IP(J)=I
      IP(I)=I
  120 IUP(I)=IUA

  130 CONTINUE
       return
       end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      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(*)
      dimension ip(*),iup(*),isupd(*)

c.............
c     purpose: numerical factorization
```

```
c      This subroutine is called by spasolver.f
c      this portion of numerical factorization has unrolling level 8
c      Modifications April 24,1995
c      modification pour la rapidite october 5,1995
c
c     DEFINITIONS
c     -----------
c     input:ia(n+1),ja(ncoef),an(ncoef),ad(n):given matrix A in RR(U)U.
c                  iu(n+1),ju(ncoef2):  structure of resulting matrix U in
c                                       RR(U)O.
c                  n                   order of matrices A and U.
c     output:      un (ncoef2)         numerical values of the nonzeros of
c                                       matrix U in RR(U)O.
c                  di (n)              inverse of the diagonal matrix D.
c     working space:  ip               of dimension N.  Chained lists of rows
c                                      associated with each column. is differnt
c                                      from the one in symbolic
c                     iup              of dimension N.  Auxiliary pointers to
c                                      portions of rows.
c                     di(n)            is used as the expanded accumulator.
c
c.............

      DO 10 J=1,N
   10 IP(J)=0
      DO 130 I=1,N
      IH=I+1
      IUA=IU(I)
      IUB=IU(IH)-1
cv    IF(IUB.LT.IUA)GO TO 40
CDIR$ IVDEP
      DO 20 J=IUA,IUB
   20 DI(JU(J))=0.d0
      IAA=IA(I)
      IAB=IA(IH)-1
cv    IF(IAB.LT.IAA)GO TO 40
CDIR$ IVDEP
      DO 30 J=IAA,IAB
   30 DI(JA(J))=AN(J)
   40 DI(I)=AD(I)
      LAST=IP(I)
      IF(LAST.EQ.0)GO TO 90
      LN=IP(LAST)
cv    loop=8
   50 L=LN
      LN=IP(L)
      m= min(i-1,isupd(l))
c     iend=(m/loop)*loop
      iend=(m/8)*8
c     isbegin=1
      isend=1+iend-1
c     IUCL=IUP(isbegin)
      iucl=iup(l)

      iuc1=iucl
      IUDL=IU(isbegin+1)-1
      iudl=iu(l+1)-1
      iup(l)=iucl + 1
      length=IUDL-IUCL+1
c     do is= isbegin,isend,8
      do is=1,isend,8
      IUC2=IU(is+2)-length
      IUC3=IU(is+3)-length
      IUC4=IU(is+4)-length
      IUC5=IU(is+5)-length
      IUC6=IU(is+6)-length
      IUC7=IU(is+7)-length
      iuc8=IU(is+8)-length
      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)
CDIR$ IVDEP
      DO 68 J=IUCL,IUDL
      JJ=JU(J)
   68 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(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
      enddo
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=IUCL,IUDL
      JJ=JU(J)
   61 DI(JJ)=DI(JJ)-UN(J)*UM1
      UN(IUCL)=UM1
      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=IUCL,IUDL
      JJ=JU(J)
   62 DI(JJ)=DI(JJ)-UN(J)*UM1-un(iuc2-iucL+j)*um2
      UN(IUCL)=UM1
      un(IUC2)=um2
      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=IUCL,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=IUCL,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(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)
CDIR$ IVDEP
      DO 65 J=IUCL,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
c@@@@@@@@@@@@@@
 6    is=isend+1
      IUC2=IU(is+2)-length
      IUC3=IU(is+3)-length
      IUC4=IU(is+4)-length
      IUC5=IU(is+5)-length
      IUC6=IU(is+6)-length
      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)
CDIR$ IVDEP
      DO 66 J=IUCL,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
c@@@@@@@@@@@
 7    is=isend+1
      IUC2=IU(is+2)-length
      IUC3=IU(is+3)-length
      IUC4=IU(is+4)-length
      IUC5=IU(is+5)-length
      IUC6=IU(is+6)-length
      IUC7=IU(is+7)-length
      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)
CDIR$ IVDEP
      DO 67 J=IUCL,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
c@@@@@@@@@@@@@@@@
 77   continue
      if(iucl.eq.iudl) 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
   80 IF(L.NE.LAST)GO TO 50
   90 DI(I)=1.d0/DI(I)
      IF(IUB.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 110
      IP(I)=IP(JJ)
      IP(JJ)=I
      GO TO 120
  110 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
        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(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),
     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),
     3iq(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)
9988    continue
        write(23,*)'CPU to get MD reordering = ',time2-time1
        endif
        return
        end
C*********************************************************
        subroutine iread0(n,ncoff,ia,icol,xls,ls,xadj,ja,adjncy)
        Integer adjncy(1),ia(1),ls(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)

c       write(*,*) 'IREAD0 IA =', (ia(i),i=1,10)
c       write(*,*) 'IREAD0 JA =', (ja(i),i=1,10)
        do 10 i = 1,n
        icol(i) = 0
```

```
        ls(i)    = 0
        xls(i)   = 0
10      xadj(i)  = 0
        xadj(1+n) = 0
        do 11 i = 1,2*ncoff
11      adjncy(i) = 0
        do 12 i = 1,n-1
        do 13 j = ia(i),ia(i+1)-1
13      icol(ja(j)) = icol(ja(j)) + 1
12      continue
        icolsum = 0
        xadj(1) = 1
        ls(1) = icol(1)
        do 14 i = 2,n
        xadj(i) = ia(i) + ls(i-1)
14      ls(i) = ls(i-1) + icol(i)
        xadj(1+n) = xadj(n) + icol(n)
c       write(*,*)'xadj(1+n) = ',xadj(1+n)
        do 15 i = 1,n-1
        iadj0 = xadj(i) + icol(i) - ia(i)
CDIR$ IVDEP
        do 16 j = ia(i), ia(i+1) - 1
        jj = ja(j)
        adjncy(iadj0+j) = jj
        adjncy(xadj(jj)+xls(jj)) = i
        xls(jj) = xls(jj) + 1
16      continue
15      continue
        return
        end
c
c-----------------------------------------------------------------------
c
      subroutine 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
1     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
       end
c
c-----------------------------------------------------------------------
c
      subroutine copyk(n,ia,perm,iu,ju,ja,un,an,ncoff)
      integer ia(1),iu(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),iu(I+1) - 1
                J0 = perm(ju(J))
                ij0  = i0
                ij00 = j0
                if(j0.LT.i0) then
                   ij0 =  j0
                   ij00 = i0
                endif
CDIR$ IVDEP
               do 230 jj = ia(ij0),ia(ij0+1)-1
               if(ja(jj).NE.ij00) go to 230
```

```
                un(J) = an(JJ)
                go to 220
230           continue
220         continue
200    continue
      rewind(11)
      write(11)(ju(i),i=1,ncoff)
      rewind(12)
      write(12)(un(i),i=1,ncoff)
      rewind(15)
      write(15)(iu(i),i=1,n+1)

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

      return
      end
c
c-----------------------------------------------------------------------
c

      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 100 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
c
c-----------------------------------------------------------------------
c

      subroutine transa9(n,m,ia,ja,iat,jat)
      implicit real*8 (a-h,o-z)
      integer ia(1),ja(1),iat(1),jat(1)
      mh = m + 1
      nh = n + 1
      do 10 i = 2, mh
10    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
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)
40    do 60 i = 1, n
      iaa = ia(i)
      iab = ia(i+1) - 1
      if ( iab.lt.iaa ) go to 60
CDIR$ IVDEP
      do 50 jp = iaa, iab
      j = ja(jp) + 1
      k = iat(j)
      jat( k ) = i
50    iat(j) = iat(j) + 1
60    continue
     call tran2(n,m,iat,jat,ia,ja)
      return
      end
      subroutine tran9(n,m,ia,ja,iat,jat)
      implicit real*8 (a-h,o-z)
      integer ia(1),ja(1),iat(1),jat(1)
      mh = m + 1
      nh = n + 1
      do 10 i = 2, mh
10    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
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)
40    do 60 i = 1, n
      iaa = ia(i)
      iab = ia(i+1) - 1
      if ( iab.lt.iaa ) go to 60
CDIR$ IVDEP
      do 50 jp = iaa, iab
      j = ja(jp) + 1
      k = iat(j)
      jat( k ) = i
50    iat(j) = iat(j) + 1
60    continue
      return
      end
     SUBROUTINE  GENMMD ( NEQNS, XADJ, ADJNCY, PERM, INVP,
    1                     DHEAD, QSIZE, LLIST, MARKER,
    1                     NOFSUB,MAXCON, NTERMS )
          INTEGER    ADJNCY(1), DHEAD(1) , INVP(1)  , LLIST(1) ,
    1             MARKER(1), PERM(1)  , QSIZE(1)
          INTEGER    XADJ(1)
          INTEGER    DELTA , EHEAD , I     , MAXINT, MDEG  ,
    1             MDLMT , MDNODE, NEQNS , NEXTMD, NOFSUB,
    1             NUM, TAG
          mquinup = 0
          mqinLm  = 0
          timeup  = 0.0
          timelm  = 0.0
          MAXINT = 10000000*1000000000
          DELTA = 0

          IF  ( NEQNS .LE. 0 )  RETURN
          NOFSUB = 0
          CALL  MMDINT ( NEQNS, XADJ, ADJNCY, DHEAD, INVP, PERM,
    1                    QSIZE, LLIST, MARKER )
          NUM = 1
          NEXTMD = DHEAD(1)
  100     CONTINUE
              IF  ( NEXTMD .LE. 0 )  GO TO 200
                  MDNODE = NEXTMD
                  NEXTMD = INVP(MDNODE)
                  MARKER(MDNODE) = MAXINT
                  INVP(MDNODE) = - NUM
                  NUM = NUM + 1
                  GO TO 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
          MDLMT = MDEG + DELTA
          EHEAD = 0
500       CONTINUE
              MDNODE = DHEAD(MDEG)
              IF  ( MDNODE .GT. 0 )  GO TO 600
                  MDEG = MDEG + 1
                  IF  ( MDEG .GT. MDLMT )  GO TO 900
                      GO TO 500
600           CONTINUE
              NEXTMD = INVP(MDNODE)
              DHEAD(MDEG) = NEXTMD
              IF  ( NEXTMD .GT. 0 )  PERM(NEXTMD) = - MDEG
              INVP(MDNODE) = - NUM
              NOFSUB = NOFSUB + 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, NEQNS
                      IF  ( MARKER(I) .LT. MAXINT )  MARKER(I) = 0
700               CONTINUE
800           CONTINUE
              CALL  MMDELM ( MDNODE, XADJ, ADJNCY, DHEAD, INVP,
     1                       PERM, QSIZE, LLIST, MARKER, MAXINT,
     1                       TAG )
              NUM = NUM + QSIZE(MDNODE)
              LLIST(MDNODE) = EHEAD
              EHEAD = MDNODE
              IF  ( DELTA .GE. 0 )  GO TO 500
900       CONTINUE
          IF  ( NUM .GT. NEQNS )  GO TO 1000
          CALL  MMDUPD ( EHEAD, NEQNS, XADJ, ADJNCY, DELTA, MDEG,
     1                   DHEAD, INVP, PERM, QSIZE, LLIST, MARKER,
     1                   MAXINT, TAG )
          GO TO 300
1000  CONTINUE
      CALL  MMDNUM ( NEQNS, PERM, INVP, QSIZE )
      RETURN
      END
      SUBROUTINE  MMDUPD ( EHEAD, NEQNS, XADJ, ADJNCY, DELTA,
     1                     MDEG, DHEAD, DFORW, DBAKW, QSIZE,
     1                     LLIST, MARKER, MAXINT, TAG )
      INTEGER    ADJNCY(1), DBAKW(1) , DFORW(1) , DHEAD(1) ,
     1           LLIST(1) , MARKER(1), QSIZE(1)
      INTEGER    XADJ(1)
      INTEGER    DEG   , DEG0  , DELTA , EHEAD , ELMNT ,
     1           ENODE , FNODE , I     , IQ2   , ISTOP ,
     1           ISTRT , J     , 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, NEQNS
                  IF  ( MARKER(I) .LT. MAXINT )  MARKER(I) = 0
200           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
                  ENODE = ADJNCY(I)
```

```
                    LINK = - ENODE
                    IF  ( ENODE )  400, 800, 500
  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
  600                       CONTINUE
                            LLIST(ENODE) = Q2HEAD
                            Q2HEAD = ENODE
  700               CONTINUE
  800           CONTINUE
                ENODE = Q2HEAD
                IQ2 = 1
  900           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
 1000                   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
 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
 1200                           CONTINUE
                                IF  ( DBAKW(NODE) .NE. 0 )  GO TO 1400
                                IF  ( DFORW(NODE) .NE. 2 )  GO TO 1300
                                    QSIZE(ENODE) = QSIZE(ENODE) +
     1                                             QSIZE(NODE)
                                    QSIZE(NODE) = 0
                                    MARKER(NODE) = MAXINT
                                    DFORW(NODE) = - ENODE
                                    DBAKW(NODE) = - MAXINT
                                    GO TO 1400
 1300                           CONTINUE
                                IF  ( DBAKW(NODE) .EQ.0 )
     1                              DBAKW(NODE) = - MAXINT
 1400                           CONTINUE
                                GO TO 2100
 1500               CONTINUE
                    ENODE = QXHEAD
                    IQ2 = 0
 1600               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
 1700                           CONTINUE
                                    JSTRT = XADJ(LINK)
```

```
                                    JSTOP = XADJ(LINK+1) - 1
                                    DO  1900  J = JSTRT, JSTOP
                                        NODE = ADJNCY(J)
                                        LINK = - NODE
                                        IF  ( NODE )  1700, 2000, 1800
1800                                    CONTINUE
                                        IF  ( MARKER(NODE) .GE. TAG )
     1                                        GO TO 1900
                                       MARKER(NODE) = TAG
                                       DEG = DEG + QSIZE(NODE)
1900                                CONTINUE
2000                       CONTINUE
2100                   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
2200                   CONTINUE
                       ENODE = LLIST(ENODE)
                       IF  ( IQ2 .EQ. 1 )  GO TO 900
                          GO TO 1600
2300           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     ELMNT , I     , ISTOP , ISTRT , J     ,
     1                 JSTOP , JSTRT , LINK  , MAXINT, MDNODE,
     1                 NABOR , NODE  , NPV   , NQNBRS, NXNODE,
     1                 PVNODE, RLMT  , RLOC  , RNODE , TAG   ,
     1                 XQNBR
           MARKER(MDNODE) = TAG
           ISTRT = XADJ(MDNODE)
           ISTOP = XADJ(MDNODE+1) - 1
           ELMNT = 0
           RLOC = ISTRT
           RLMT = ISTOP
           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) = ELMNT
                       ELMNT = NABOR
 200       CONTINUE
 300       CONTINUE
               IF  ( ELMNT .LE. 0 )  GO TO 1000
                   ADJNCY(RLMT) = - ELMNT
                   LINK = ELMNT
 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
 500                       CONTINUE
                           IF  ( MARKER(NODE) .GE. TAG  .OR.
     1                           DFORW(NODE) .LT. 0 )  GO TO 800
                               MARKER(NODE) = TAG
 600                           CONTINUE
                                   IF  ( RLOC .LT. RLMT )  GO TO 700
                                       LINK = - ADJNCY(RLMT)
                                       RLOC = XADJ(LINK)
                                       RLMT = XADJ(LINK+1) - 1
                                       GO TO 600
```

```
 700                          CONTINUE
                              ADJNCY(RLOC) = NODE
                              RLOC = RLOC + 1
 800                      CONTINUE
 900                  CONTINUE
                      ELMNT = LLIST(ELMNT)
                      GO TO 300
1000      CONTINUE
          IF  ( RLOC .LE. RLMT )  ADJNCY(RLOC) = 0
          LINK = MDNODE
1100      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.
     1                    PVNODE .EQ. (-MAXINT) )  GO TO 1300
                      NXNODE = DFORW(RNODE)
                      IF  ( NXNODE .GT. 0 )  DBAKW(NXNODE) = PVNODE
                      IF  ( PVNODE .GT. 0 )  DFORW(PVNODE) = NXNODE
                      NPV = - PVNODE
                      IF  ( PVNODE .LT. 0 )  DHEAD(NPV) = NXNODE
1300              CONTINUE
                  JSTRT = XADJ(RNODE)
                  JSTOP = XADJ(RNODE+1) - 1
                  XQNBR = 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(XQNBR) = NABOR
                              XQNBR = XQNBR + 1
1400              CONTINUE
1500              CONTINUE
                  NQNBRS = XQNBR - 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
1600              CONTINUE
                  DFORW(RNODE) = NQNBRS + 1
                  DBAKW(RNODE) = 0
                  ADJNCY(XQNBR) = MDNODE
                  XQNBR = XQNBR + 1
                  IF  ( XQNBR .LE. JSTOP )  ADJNCY(XQNBR) = 0
1700          CONTINUE
1800      CONTINUE
          RETURN
      END
      SUBROUTINE  MMDINT ( NEQNS, XADJ, ADJNCY, DHEAD, DFORW,
     1                     DBAKW, QSIZE, LLIST, MARKER )
          INTEGER    ADJNCY(1), DBAKW(1) , DFORW(1) , DHEAD(1) ,
     1               LLIST(1) , MARKER(1), QSIZE(1)
          INTEGER    XADJ(1)
          INTEGER    FNODE , 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 )
          INTEGER    INVP(1)  , PERM(1)  , QSIZE(1)
```

```
      INTEGER    FATHER, NEQNS , NEXTF , NODE  , NQSIZE,
     1           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
  200         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
                  PERM(ROOT) = NUM
                  FATHER = NODE
  400             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(NUM) = NODE
  600 CONTINUE
      RETURN
      END
C*************************************************************************
c     LUMP MASS CASE + shift factor for the regular Lanczos
c
c     H. Runesha  dec 22, 1997
c
C*************************************************************************

      SUBROUTINE SP2LAN(N,LANMAX,NEIG,UN,DI,IU,JU,W,w1,w2,w4,
     $TEM,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/sp2com/nbuf(4)

       rewind(10)
       read(10) (dmass(i),i=1,n)
       iam=0
       nodes=1
       NBLK = 1
       RTOL = 0.0000000001d0
       NNNN= 4*NEIG
       IF(NNNN.GE.LANMAX)NNNN=LANMAX-1
       MN=0
       DO 61 I=1,LANMAX
       EIG(I)=0.0d0
   61  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
       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
       W1(i)=0.0d0
   15  CONTINUE
 1501  CONTINUE
       do 9901 i=1,n
       w2(i)=dmass(i)*w(i)
```

```
9901    continue
            bet=0.0d0
            do 99011 I=1,N
99011      bet=bet+w2(i)*w(i)
            bet=dsqrt(bet)
          do 99012 I=1,N
           W1(i)=w(i)/BET
           w(i)=0.0d0
           Q(i,1) = W1(i)
99012      continue
            do 99013 I=1,n
99013       w2(i)=dmass(i)*W1(i)
c ***************************
        DO 50 JJ=1,LANMAX-1
        DO 1001 I=1,N
1001    W4(I)=W2(I)
C ****************************************
       call fbe(n,iu,ju,di,un,w2,w4,iopfb)
c       write(*,*)'JJ = ',jj
c      call node(nodes,iam,n,nbw,imod,8,stif,z,y,W4,maxa,irow,icolg,
c     1tem0,2,z0)
C ****************************************
        DO 55 I=1,N
        W4(I)=W4(I)-W(I)*BET
55      CONTINUE
        ALF=DDOT(N,W2,1,W4,1)
        DO 60 I=1,N
        W4(I)=W4(I)-ALF*W1(I)
60      CONTINUE
C*********************************************************************
        do 9902 i=1,n
9902    w2(i)=dmass(i)*w4(i)
c ***********************
        BET2=DDOT(N,W4,1,W2,1)
        bet2=dsqrt(bet2)
c       write(*,*)' before call REORTH: JJ = ',jj
        CALL REORTH(N,W2,
      &W4,Q,EPS,LANMAX,TEM,JJ,BET2)
        do 9903 i=1,n
9903    w2(i)=dmass(i)*w4(i)
c ***********************
        BET2=DDOT(N,W4,1,W2,1)
c       write(*,*)' before 100 : JJ,beta = ',jj,beta
        bet2=dsqrt(bet2)
100     CONTINUE
        DBET2=1.0d0/BET2
        DO 110 I=1,N
        W(I)=W1(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
        BET=BET2
c      write(*,*)'JJ,ALF,BET2,iam=',jj,alf,bet2
       IF(JJ.LT.NNNN) GO TO 50
c      if(nodes.gt.1)call mp_sync(nbuf(4))

c      write(*,*) ' ishift =', ishift
       CALL JACOBIPi(N,NEIG,JJ,BET2,W(1),Q,VEC,EIG,
      $ERR,UN,TEM,RTOL,LANMAX,DMASS,
      $W1,N+1,NNNN,NSUC,ncoff,ncof2,iu,ju,di,ishift)
       IF (NSUC.EQ.1) GO TO 2211
50     CONTINUE
120     CONTINUE
       CALL JACOBIPi(N,NEIG,JJ-1,BET2,W(1),Q,VEC,EIG,
      $ERR,UN,TEM,RTOL,LANMAX,DMASS,
      $W1,N+1,NNNN,NSUC,ncoff,ncof2,iu,ju,DI,ishift)
2211    CONTINUE

        RETURN
        END
c
c*********************************************************************
c
        SUBROUTINE REORTH(N,P,R,Q,EPS,LANMAX,HH,NJ,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=1,NJ-1
c     if(abs(hh(k)).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
c
c*********************************************************************
c
      SUBROUTINE JACOBIPi(N,NEIG,NJ,BET2,V,Q,VEC,
     $ EIG,ERR,AN,HH,RTOL,LANMAX,DMASS,
     $ VP1,NP1,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),VP1(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(1000)
      iam=0
      nodes=1
      NSUC=0
      DO 1 I=1,LANMAX
      DO 2 J=1,LANMAX
2     VEC(I,J)=0.0d0
      HH(I)=EIG(I)
      SUB(I)=ERR(I)
      VEC(I,I)=1.0d0
1     CONTINUE
c     if(nodes.gt.1)call mp_sync(nbuf(4))
      CALL TQL2(LANMAX,NJ,EIG,ERR,VEC,IRR,EPS)
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)+abs(vec(j,i))
      ERR(I)=err(i)*0.005d0*ABS(BET2*VEC(NJ,I)/EIG(I))
c     write(*,*)'I,err(i),vec(nj,i)=',i,err(i),vec(nj,i)
20    CONTINUE
      DO 30 I=1,NEIG
C      IF(ERR(I).GT.0.000001.AND.NJ.LT.(LANMAX-1)) GO TO 300
      IF(ERR(I) .GT. RTOL .AND. NJ .LT. (LANMAX-1)) GO TO 300
30    CONTINUE
c     if(nodes.gt.1)call mp_sync(nbuf(4))
      CALL VECTRA(N,NEIG,0,VEC,Q,LANMAX,NJ,V)
c     write(*,*)'IN JACOBQ: VECTRA is done !!!!  '
C     NSUC: =1,succesful
      NSUC=1
c     DO 580 L=1,NEIG
      call cputime(t01)
      ninc=8*nodes
c       do L=1,neig
c       do i=n-nadd+1,n
c       q(i,L)=0.0d0
c       enddo
c       enddo
c     write(*,*)'NADD = ',nadd,n
c     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)
```

```
c       write(*,*) '_____NORMCHECK2'
c       write(*,*)
c       write(*,*) 'lanmax,n,neig'
c       write(*,*) lanmax,n,neig
c       write(*,*) 'IA =', (ia(i),i=1,1+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,iq)
c        do 510 j=1,neig
c510     write(*,*) (q(i,j),i=1,n)


        DO 580 L=1,neig
c       call multspa(n,ia,ja,an,ad,q,v)
c       call mulmeiko(n,,,,)
c      if(ninc.gt.8)call mp_sync(nbuf(4))
c       call sp2mul(iam,ninc,n,icolg,maxa,stif,q(1,L),vp1,v)
        call multspa(n,ia,ja,an,ad,q(1,L),v)
c       write(*,*)'Q(i,L) = ',L,(q(i,L),i=1,n)
        VNORM=0.0d0
        DO 590 I=1,N
590     VNORM=VNORM+V(I)*V(I)
        WNORM=0.0d0
        do 2239 i=1,n
        vp1(I)=dmass(i)*Q(i,L)
2239    continue
c        write(*,*) 'I, vp1=', i, (vp1(ii),ii=1,n)

             RT=1.0d0/EIG(L)
        DO 600 I=1,N
c       erm=max(abs(v(i)),abs(rt*vp1(i)))
c       error=abs(v(i)-rt*vp1(i))/erm
c       if(error.gt.0.95d0.and.erm.gt.0.00001d0)then
c       write(*,*)'L,I,V(I),,=',L,I,v(i),rt*vp1(i),error,q(i,L)
c       endif
         V(I)=V(I)-RT*VP1(I)
600     continue
c       write(*,*) 'V(I)=', (v(i),i=1,n)

        do 601 i = 1,n
601     WNORM=WNORM+V(I)*V(I)
c       if(iam.eq.0)write(*,*)'VNORM, WNORM = ',vnorm,wnorm,L
        VNORM=DSQRT(VNORM)
        WNORM=DSQRT(WNORM)
        HH(L)=WNORM/VNORM
580     CONTINUE
c       write(*,*) ' HH =' ,(hh(i),i=1,neig)

        call cputime(t02)
        t02=t02 - t01
        WRITE(23,*)'*** K, * EIG*,*HERTZ *,* ERROR *,* NORM *** iam'

        if(ishift.eq.0) then
        DO 700 K=1,NEIG
c       HERTZ=1.0/(2.0*3.1415927*DSQRT(EIG(K)))
        HERTZ=1.0/(2.0*3.1415927*DSQRT(DABS(EIG(K))))
        WRITE(23,701)K,1.0/EIG(K),HERTZ,ERR(K),HH(K)
701     FORMAT(2X,I5,2X,4E15.7)
700     CONTINUE
        else
        DO 705 K=1,NEIG
        EIG(K)=1./EIG(K)-float(ishift)
c       HERTZ=DSQRT(EIG(K))/(2.0*3.1415927)
        HERTZ=DSQRT(DABS(EIG(K)))/(2.0*3.1415927)
c       WRITE(23,701)K,1.0/EIG(K),HERTZ,ERR(K),HH(K)
        WRITE(23,701)K,EIG(K),HERTZ,ERR(K),HH(K)
705     CONTINUE
        endif

        NSUC=1
        if(iam.eq.0)write(23,*) 'JACOBIQ: Steps in IAM = ',nj,iam
        RETURN
300     NNNN=MIN0(LANMAX-1,NJ+3*(NEIG-I)+4)
        NSUC=0
        DO 304 I=1,LANMAX
        EIG(I)=HH(I)
```

```
304     ERR(I)=SUB(I)
        RETURN
        END
C
C*********************************************************************
C

        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 SPECIFING
C       THE RELATIVE PRECISION OF FLOATING POINT ARITHEMETIC
        IQL0=ICOUNQ
        IERR=0
c       WRITE(*,*)'*** TQL2 BEGIN  *******'
        IF(N.EQ.1) GO TO 1001
CDO 100 I=2,N
C100     E(I-1)=E(I)
        E(N)=E(N-1)
        F=0.0d0
        B=0.00
        E(N)=0.0d0
        DO 240 L=1,N
        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  ********
        L1=L+1
        G=D(L)
        P=(D(L1)-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=L1,N
140     D(I)=D(I)-H
        ICOUNQ=ICOUNQ+(N+1-L1)
        F=F+H
C*****  QL TRANSFORMATION  *******
        P=D(M)
        C=1.0d0
        S=0.0d0
        MML=M-L
C****   FOR I=M-1 STEP -1 UNTIL L DO -- *******
        DO 200 II=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=1.0/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+1)
        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
```

```
C
        ICOUNQ=ICOUNQ+15*MML+13
        E(L)=S*P
        D(L)=C*P
        IF(ABS(E(L)).GT.B) GO TO 130
220     D(L)=D(L)+F
240     CONTINUE
C***  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)
260      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
280     CONTINUE
C
300     CONTINUE
C
        GO TO 1001
C    ** SET ERROR -- NO CONVERGENCE TO AN EIGENVALUE
C       AFTER 30 ITERATIONS     ****************
1000    IERR=L
1001    CONTINUE
        WRITE(23,*)'TQL2 END,IERR=',IERR,';at ',N,'-th Lan step.'
        RETURN
        END
        SUBROUTINE VECTRA(N,NEIG,IBLOCK,VEC,Q,LANMAX,NJ,R)
       implicit real*8 (a-h,o-z)
        REAL*8 Q(N,IBLOCK+LANMAX),VEC(lanmax,lanmax),R(1)
c       write(23,*)'VECTRA: lanmax,nj,neig = ',lanmax,nj,neig,iblock
        DO 20 i = 1,n
        do 30 j=1,neig
        R(j)=0.0d0
        do 40 k=1,nj
40        R(j)=R(j)+q(i,k)*vec(k,j)
30       continue
         do 50 kk=1,neig
50       q(i,kk)=R(kk)
20      CONTINUE
        WRITE(23,*)'**** VECTRA  END ,ilen,iseg= *****'
        RETURN
        END
C
C*********************************************************************
C

        FUNCTION GETEPS(IBET,IT,IRND)
       implicit real*8 (a-h,o-z)
         a = 1.0d0
10      a = a + a
        if(((a+1.0d0)-a)-1.0d0.eq.0.0d0) go to 10
        b=1.0d0
20      b=b+b
        if ((a+b)-a.eq.0.0d0) go to 20
        ibeta=int(real((a+b)-a))
        beta=float(ibeta)
        it=0
        b=1.0d0
30      it=it+1
        b=b*beta
        if(((b+1.0d0)-b)-1.0d0.eq.0.0d0)go to 30
        irnd=0
        betam1=beta-1.0d0
        if((a+betam1)-a.ne.0.0d0)irnd=1
        betain=1.0d0/beta
        a=1.0d0
        do 40 i=1,it+3
```

```
        a=a*betain
40      continue
50      if((1.0d0+a)-1.0d0.ne.0.0d0)go to 60
        a=a*beta
        go to 50
60      eps=a
        if((ibeta.eq.2).or.(irnd.eq.0))go to 70
        a=(a*(1.0d0+a))/(1.0d0+1.0d0)
        if((1.0d0+a)-1.0d0.ne.0.0d0)eps=a
70      geteps=eps
         return
        end
c
c*********************************************************************
c

        real*8 function ddot(n,w,k,w2,j)
        implicit real*8 (a-h,o-z)
        real*8 w(1),w2(1)
        dd=0.0d0
        do 1 i=1,n
1       dd=dd+w(i)*w2(i)
        ddot=dd
        return
        end
c
c*********************************************************************
c

C*************  Block Lanczos Driver     ********************************
C*********************************************************************
c      SUBROUTINE SP2LAN(N,LANMAX,NEIG,UN,DI,IU,JU,W,w1,w2,w4,
c     $TEM,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,betai,eig,vec,alfa,q,t,b,am,ia,ja,lump,ncoff,ishift)
        implicit real*8 (a-h,o-z)
        real*8 T(isize,isize),un(1),di(1),am(1),eig(isize)
        real*8 dmass(1),q(n,isize+iblock),r(n,iblock),tem(n,iblock)
        real*8 alfa(iblock,iblock),beta(iblock,iblock),vec(isize,isize)
        real*8 b(isize,isize),betai(iblock,iblock),p(n,iblock)
        integer ia(1),ja(1),iu(1),ju(1)
C
C
C*****   This is the driver for block Lanczos eigensolver   *****
C*****   1.   Initialization:
C
        do i = 1,isize
           do j = 1,isize
           T(j,i) = 0.0d0
           B(j,i) = 0.0d0
           enddo
        enddo
        if(lump.eq.1) then
                rewind(10)
                read(10) (dmass(i),i=1,n)
                write(23,*)'Lumped Mass is used ! '
        else
                write(23,*)'Consistant Mass is used ! '
                rewind(10)
                read(10) (dmass(i),i=1,n)
                rewind(17)
                read(17)(am(i),i=1,ncoff)
                rewind(15)
                read(15)(ia(i),i=1,1+n)
                rewind(11)
                read(11)(ja(i),i=1,ncoff)

c               stop
        endif
C****   Do the shifting: K~ = K + ishift*M

        lanmax=isize/iblock

       write(23,*)'n,isize,iblock,ishift= ',n,isize,iblock,ishift
```

```
        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
C
        xmul=float(((n+1)**3)/3)
        xmul=n
        xmul=dsqrt(3.0d0)/(xmul*dsqrt(xmul))
c       xmul=dsqrt(2.0d0)/float(n)
c       xmul=1.0d0                              !  just for the 6x6 ex.
        write(23,*)'xmul = ',xmul
        do i = 1,n
c         r(i,1) = xmul
          r(i,1)= float(i)*xmul
        if(iblock.GE.2) then
        if((i/2)*2.eq.i)then
             xmul1=n-i+1
          else
            xmul1=i-n-1
        endif
        r(i,2)= xmul1 * xmul
        if(iblock.eq.3) then
          r(i,3)=1.0d-03 + sin(3.1415926*i/4)
        endif
        endif
c         write(*,*)'r(i,1) = ',i,r(i,1)
        enddo
C
c       r(n,2)=1.0d0                            !  just for the 6x6 ex.
C
        call stepf(n,iblock,tem,q,r(1,1),beta,ia,ja,am,dmass,lump,
       11)
C
        call stepei(n,iblock,p(1,1),q(1,1),ia,ja,am,dmass,lump)
C
C***    Now, the main DO LOOP starts here:  J = 1,2,3,....<= lanmax
C
        DO J = 1,lanmax

C
           j01=(J-1)*iblock+1
           j02=J*iblock+1
           j03=(J-2)*iblock+1
          do k = 1,iblock
               do i = 1,iblock
                 betai(i,k) = beta(i,k)
               enddo
          enddo

          call stepa(n,iblock,r(1,1),p(1,1),iu,ju,di,un)
C
          if(J.ne.1) call stepb(n,iblock,r(1,1),q(1,J03),betai)
C
          call stepc(n,iblock,p(1,1),r(1,1),alfa)
C
          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
C
c         call stepei(n,iblock,p(1,1),r(1,1),ia,ja,am,dmass,lump)
C
          call stepf(n,iblock,tem,q,r(1,1),beta,ia,ja,am,dmass
     1              ,lump,J+1)
C
          call stepei(n,iblock,p(1,1),q(1,J02),ia,ja,am,dmass,lump)
C
1000      continue

          call assemt(isize,iblock,T,alfa,beta,J)
C
        ENDDO
C
        call JACOBI (T,B,VEC,TEM(1,1),TEM(1,2),ISIZE,TOL,NSMAX,IFPR,
     1              ISHIFT)
```

```
C
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))
           iqmax=i
        endif
        enddo
c       write(23,*)'K,ivmax,Vmax = ',L,iqmax,qmax
        enddo
c       write(23,*)'before VECTRA* Max. Element in the L-th Q vectors *'
        do L = 1,isize
        qmax=0.0d0
        iqmax=0
        do i = 1,n
        if(dabs(q(i,L)).gt.qmax) then
           qmax=dabs(q(i,L))
           iqmax=i
        endif
        enddo
c       write(23,*)'L,iqmax,Qmax = ',L,iqmax,qmax
        enddo
        call VECTRA(N,NEIG,IBLOCK,VEC,Q,isize,isize,R(1,1))
c       do k = 1,neig
c       write(*,*)'k,Q= ',K,(q(i,k),i=1,n)
c       enddo
C
        call check(n,neig,isize,ncoff,ia,ja,un,di,q,p(1,1),tem(1,1),
     1 tem(1,2),r(1,1),am,dmass,lump,ishift)

C
          return
          end

       subroutine stepa(n,iblock,r,p,iu,ju,di,un)
       implicit real*8 (a-h,o-z)
       real*8 un(1),di(1),r(n,iblock),p(n,iblock)
       integer iu(1),ju(1)
C***** This subroutine finds: r that Kr = p in Step a.  *******
       do i = 1,iblock
          call fbe(n,iu,ju,di,un,p(1,i),r(1,i),iopfb)
       enddo
       return
       end
C*****************************************************************
C
       subroutine stepb(n,iblock,r,q,beta)
       implicit real*8 (a-h,o-z)
       real*8 r(n,iblock),q(n,iblock),beta(iblock,iblock)
C** This subroutine finds r(head)=r(bar) - q*beta(transpo.) in Step b. ****
C**                  beta stored as upper-trangular matrix
       do i = 1,iblock
          do j = i,iblock
             do k = 1,n
                r(k,i)=r(k,i) - q(k,j) * beta(i,j)
             enddo
          enddo
       enddo
       return
       end
C*********************************************************************
C
       subroutine stepc(n,iblock,p,r,alfa)
       implicit real*8 (a-h,o-z)
       real*8 r(n,iblock),p(n,iblock),alfa(iblock,iblock)
C**** This subroutine finds: alfa = P(transp.)*R   in Step c.   ******
       do j = 1,iblock
          do i = 1,j
             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
       enddo
       return
```

```
      end
C*********************************************************************
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*********************************************************************
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
C
       else
c      write(*,*)'e:dmass=',(dmass(i),i=1,n)
c      write(*,*)'e:am = ',(am(i),i=1,6)
C******  consistant Mass matrix   *************
         do j = 1,iblock
             call multspa(n,ia,ja,am,dmass,r(1,j),p(1,j))
         enddo
C
       endif
       return
       end
C*********************************************************************
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,1),am(1),dmass(1),r(n,iblock)
      real*8 beta(iblock,iblock),tem(n,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
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
c
         do j = 1,iblock
            beta(j,i)  = 0.0d0
         enddo
       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
C
        else
C
C******  consistant Mass matrix   *************
         do j = 1,iblock
             call multspa(n,ia,ja,am,dmass,r(1,j),tem(1,j))
```

```
          enddo
C
        endif
C ****    now: tem = Mr   already !
c       write(*,*)'stepf: finished tem=Mr !'
        do k = 1, iblock
C*
          do j = 1, k-1
C*********
            do i = 1,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)
C
            do i = 1,n
c           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.1) then
C******   lumped Mass matrix   *************
           do i = 1,n
              tem(i,k) = dmass(i) * q(i,k+j0)
           enddo
C
        else
C
C******  consistant Mass matrix   *************
c
c       write(*,*)'f:dmass=',(dmass(i),i=1,n)
c       write(*,*)'f:am = ',(am(i),i=1,6)
c       write(*,*)'f:q = ',(q(i,k+j0),i=1,4)
c       write(*,*)'f:tem= ',(tem(i,k),i=1,4)
            call multspa(n,ia,ja,am,dmass,q(1,k+j0),tem(1,k))
c       write(*,*)'f:tem=Mq= ',(tem(i,k),i=1,4)
C
        endif
          beta(k,k) = 0.0d0
                  do i = 1,n
                    beta(k,k) = beta(k,k) + q(i,k+j0)*tem(i,k)
                  enddo
C
                  beta(k,k) = dsqrt(beta(k,k))
C
                     xmult=1.0d0/beta(k,k)
                     write(23,*)'beta(k,k) = ',k,beta(k,k)
                     do i = 1,n
                       q(i,k+j0) =  xmult * q(i,k+j0)
                     enddo
        enddo
c       write(*,*)'stepf: finished beta !'
C*
C***      The following is for  REORTHOGONALIZATION/normalization  ******
C
C*******  JB is the block # of the current block.      *****
C
        do m = (JB-1)*iblock+1, JB*iblock
c       if(n.gt.1) goto 1995

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

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

       enddo
C
C
        return
        end
C*********************************************************************
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
         enddo
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
       endif
       return
       end
C*********************************************************************
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  *******
C
      J0 = (JJ-1)*iblock
      J1 = JJ*iblock
      if(J1.LT.isize) THEN
      do J = 1,iblock
         do I = 1,iblock
         T(J0+I,J0+J) = alfa(I,J)
         T(J0+J,J1+I) = beta(I,J)
c     write(*,*)'j0+i,j0+j,T = ',j0+i,j0+j,T(j0+i,j0+j)
c     write(*,*)'j0+j,j1+i,T = ',j0+j,j1+i,T(j0+j,j1+i)
         enddo
```

```
      enddo
      ELSE
C**********  if( J1.GE.the size of T) do not put beta in !!!!  ********
      do J = 1,iblock
         do I = 1,iblock
         T(J0+I,J0+J) = alfa(I,J)
         enddo
      enddo
      ENDIF
C
      return
      end
C*****************************************************************
C
       subroutine check(n,neig,isize,ncoff,ia,ja,an,ad,q,hh,eig,
     1 V,VP1,am,dmass,lump,ishift)
       implicit real*8 (a-h,o-z)
       real*8 hh(1),q(n,1),V(1),VP1(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||     *********
c      call cputime(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)
c      write(*,*)'lump,n,isize,ncoff = ',lump,n,isize,ncoff
c      write(*,*)'check: eig = ',(eig(i),i=1,neig)
c      write(*,*)'check: ia = ',(ia(i),i=1,n+1)
c      write(*,*)'check: ja = ',(ja(i),i=1,ncoff)
c      write(*,*)'check: ad = ',(ad(i),i=1,n)
c      write(*,*)'check: an = ',(an(i),i=1,ncoff)
c      write(*,*)'check: dmass = ',(dmass(i),i=1,n)
c      do k = 1,neig
c      write(*,*)'check: k,Q = ',(q(i,k),i=1,n)
c      enddo
c      write(23,*)'Check: **** Max. Element in the L-th Q vectors ****'
       DO 580 L=1,neig
       qmax=0.0d0
       iqmax=0
       do i = 1,n
       if(dabs(q(i,L)).gt.qmax) then
          qmax=dabs(q(i,L))
          iqmax=i
       endif
       enddo
c      write(23,*)'L,iqmax,Qmax = ',L,iqmax,qmax
       call multspa(n,ia,ja,an,ad,q(1,L),v)
       VNORM=0.0d0
       DO 590 I=1,N
590    VNORM=VNORM+V(I)*V(I)
       WNORM=0.0d0
       if(lump.eq.1) then
C***** Lumped Mass matrix        **************
C
       do 2239 i=1,n
       vp1(I)=dmass(i)*Q(i,L)
2239   continue
C
       else
C******  consistant Mass matrix   *************
         call multspa(n,ia,ja,am,dmass,q(1,L),VP1)
C
       endif
C
            RT=1.0d0/EIG(L) - float(ishift)
       DO 600 I=1,N
        V(I)=V(I)-RT*VP1(I)
600    continue
       do 601 i = 1,n
601    WNORM=WNORM+V(I)*V(I)
       VNORM=DSQRT(VNORM)
       WNORM=DSQRT(WNORM)
c      write(23,*)'L,wnorm,Vnorm = ',L,wnorm,vnorm
       HH(L)=WNORM/VNORM
       if(dabs(RT).lt.1.0D-14)HH(L) = DABS(RT)
```

```
580     CONTINUE
c       call cputime(t02)
        t02=t02 - t01
        WRITE(23,*)'*** K, * EIG*,*HERTZ *,* ERROR  NORM *** '
        DO 700 K=1,NEIG
        HERTZ=1.0/(2.0*3.1415927)*DSQRT(max(0.0D0,1.0D0/EIG(K)-ishift))
        WRITE(23,701)K,max(0.0D0,1.0D0/EIG(K)-ishift),HERTZ,HH(K)
701     FORMAT(2X,I5,2X,4E15.7)
700     CONTINUE
        return
        end
C*****************************************************************
C
c       SUBROUTINE JACOBI (A,B,X,EIGV,D,N,RTCL,NSMAX,IFPR,IOUT)
c     .....................................................................
c     .                                                                   .
c     .   PROGRAM TO SOLVE THE GENERALIZED EIGENPROBLEM                   .
c     .   USING THE GENERALIZED JACOBI ITERATION                          .
c     .                                                                   .
c     .   -- INPUT VARIABLES --                                           .
c     .                                                                   .
c     .      A(N,N)     =Stiffness matrix (assumed positive definite)     .
c     .      B(N,N)     =Mass matrix (assumed positive definite )         .
c     .      X(N,N)     =Vector storing eigenvectors on solution exit     .
c     .      EIGV(N)    =Vector storing eigenvalues on solution exit      .
c     .      D(N)       =Working vector                                   .
c     .      N          =order of matrices A and B                        .
c     .      RTOL       =Convergence tolerance(usually set to 10.**-12    .
c     .      NSMAX      =Maximum number of sweeps allowed                 .
c     .                  (usually set to 15)                              .
c     .      IFPR       =flag for printing during iteration               .
c     .          EQ.0    No printing                                      .
c     .          EQ.1    Intermeddiate results are printed                .
c     .      IOUT       =output device number                             .
C     *      ISHIFT:    K~=K+ishift*M
c     .                                                                   .
c     .   -- OUTPUT --                                                    .
c     .                                                                   .
c     .      A(N,N)     =Diagonalized stiffness matrix                    .
c     .      B(N,N)     =Diagonalized mass matrix                         .
c     .      X(N,N)     =EIGENVECTors stored columnwise                   .
c     .      EIGV(N)    =Eigenvalues                                      .
c     .                                                                   .
c     .....................................................................
      SUBROUTINE JACOBI (A,B,X,EIGV,D,N,RTCL,NSMAX,IFPR,ISHIFT)
c     Parameter (RTOL=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
      RTOL = 1.0D-14
c     .....................................................................
c     .   This program is used in single precision arithmetic on          .
c     .   cdc equipement and double precision  arithmetic on ibm          .
c     .   or univac machines. activate,deactivate or adjust above         .
c     .   card for single or double precision arithmetic                  .
c     .....................................................................

c
c     INITIALIZE EIGENVALUE AND EIGENVECTOR MATRICES
c
      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
   20    X(I,J)=0.d0
```

```
   30 X(I,I)=1.0D0
ccccccccccccccccccccccccccc      IF (N.EQ.1) RETURN
      IF (N.EQ.1) STOP
c
c     INITIAALIZE SWEEP COUNTER AND BEGIN ITERATION
c
      NSWEEP=0
      NR=N-1
   40 NSWEEP=NSWEEP+1
      IF (IFPR.EQ.1) WRITE (IOUT,2000)NSWEEP
c
c     CHECK IF PRESENT OFF-DIAGONAL ELEMENT IS LARGE ENOUGH TO REQUIRE
c     ZEROING
c
      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
      IF (CHECK)50,60,60
   50 WRITE(IOUT,2020)
      STOP
   60 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
   70 CA=0.
      CG=-A(J,K)/A(K,K)
      GO TO 90
   80 CA=AKK/DEN
      CG=-AJJ/DEN
c
c     PERFORM THEGENERAL ROTATION TO ZERO THE PRESENT OFF DIAGGONAL ELEMENT
c
   90 IF(N-2)100,190,100
  100 JP1=J+1
      JM1=J-1
      KP1=K+1
      KM1=K-1
      IF(JM1-1)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(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 AK=A(K,K)
      BK=B(K,K)
      A(K,K)=AK+2.0D0*CA*A(J,K)+CA*CA*A(J,J)
```

```
      B(K,K)=BK+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
C
C     UPDATE THE EIGENVECTOR MATRIX AFTER EACH ROTATION
C
      DO 200 I=1,N
      XJ=X(I,J)
      XK=X(I,K)
      X(I,J)=XJ+CG*XK
  200 X(I,K)=XK+CA*XJ
  210 CONTINUE
C
C     UPDATE THE EIGENVALUE AFTER EACH SWEEP
C
      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)
C
C     CHECK FOR CONVERGENCE
C
  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

C
C     CHECK ALL OFF-DIAGONAL ELEMENTA TO SEE IF ANOTHER SWEEP IS REQUIRED
C
      EPS=RTOL**2
      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
C
C     FILL OUT BOTTOM TRIANGLE OF RESULTANT MATRICES AND SCALE EIGENVECTORS
C
  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=1,N
      BB=DSQRT(B(J,J))
      DO 270 K=1,N
  270 X(K,J)=X(K,J)/BB
CCCCCCCCCCCCCCCCCC      RETURN


c     write(*,*) 'THE EIGENVECTORS ARE :'
c     write(*,*) ((x(i,j),j=1,n),i=1,n)
C
C**   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
             enddo
           endif
          enddo
      enddo
      write(23,*) 'THE EIGENVALUES in JACOBI ARE :'
      do i = 1,n
```

```
      WRITE(23,*)'** I ** ',i,'    ',EIGV(I),'    **'
      enddo
      return
c     STOP
c     ..................................................................
c
c     UPDATE D MATRIX AMD START NEW SWEEP,IF ALLOWED
c
  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     1X, 30H MATRICES NOT POSITIVE DEFINITE )
 2030 FORMAT(36HOCURRENT EIGENVALUES IN *JACOBI* ARE,/)
      END
c     ..................................................................
```

```
      SUBROUTINE JACOBI2 (A,B,X,EIGV,N,D,rtol,nsmax)
c     modified by runesha Nov 27, 1995
c     ..................................................................
c     .                                                                .
c     .   PROGRAM TO SOLVE THE GENERALIZED EIGENPROBLEM                .
c     .   USING THE GENERALIZED JACOBI ITERATION                       .
c     .                                                                .
c     .   -- INPUT VARIABLES --                                        .
c     .                                                                .
c     .      A(N,N)     =Stiffness matrix (assumed positive definite)  .
c     .      B(N,N)     =Mass matrix (assumed positive definite )      .
c     .      X(N,N)     =Vector storing eigenvectors on solution exit  .
c     .      EIGV(N)    =Vector storing eigenvalues on solution exit   .
c     .      D(N)       =Working vector                                .
c     .      N          =order of matrices A and B                     .
c     .      RTOL       =Convergence tolerance(usually set to 10.**-12 .
c     .      NSMAX      =Maximum number of sweeps allowed              .
c     .                  (usually set to 15)                           .
c     .                                                                .
c     .   -- OUTPUT --                                                 .
c     .                                                                .
c     .      A(N,N)     =Diagonalized stiffness matrix                 .
c     .      B(N,N)     =Diagonalized mass matrix                      .
c     .      X(N,N)     =Eigenvectors stored columnwise                .
c     .      EIGV(N)    =Eigenvalues                                   .
c     .                                                                .
c     ..................................................................

c     Parameter (RTOL=10.**-12,NSMAX=15,IFPR=0,IOUT=6)
c     Parameter (RTOL=0.000000000001,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)
c     ..................................................................
c     .   This program is used in single precision arithmetic on       .
c     .   cdc equipement and double precision  arithmetic on ibm       .
c     .   or univac machines. activate,deactivate or adjust above      .
c     .   card for single or double precision arithmetic               .
c     ..................................................................

c
c     INITIALIZE EIGENVALUE AND EIGENVECTOR MATRICES
c
      DO 10 I=1,N
      IF (A(I,I).GT.0. .AND. B(I,I).GT.0.) GO TO 4
      WRITE (*,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
   20    X(I,J)=0.d0
   30 X(I,I)=1.
c.....IF (N.EQ.1) RETURN
      IF (N.EQ.1) STOP
c
c     INITIAALIZE SWEEP COUNTER AND BEGIN ITERATION
c
      NSWEEP=0
      NR=N-1
   40 NSWEEP=NSWEEP+1
c     IF (IFPR.EQ.1) WRITE (*,2000)NSWEEP
c
c     CHECK IF PRESENT OFF-DIAGONAL ELEMENT IS LARGE ENOUGH TO REQUIRE
c     ZEROING
c
      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.*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
C
C     PERFORM THEGENERAL ROTATION TO ZERO THE PRESENT OFF DIAGGONAL ELEMENT
C
   90 IF(N-2)100,190,100
  100 JP1=J+1
      JM1=J-1
      KP1=K+1
      KM1=K-1
      IF(JM1-1)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(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 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(J,K)+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
C
C     UPDATE THE EIGENVECTOR MATRIX AFTER EACH ROTATION
C
      DO 200 I=1,N
      XJ=X(I,J)
      XK=X(I,K)
      X(I,J)=XJ+CG*XK
  200 X(I,K)=XK+CA*XJ
  210 CONTINUE
C
C     UPDATE THE EIGENVALUE AFTER EACH SWEEP
C
      DO 220 I=1,N
      IF (A(I,I).GT.0. .AND.B(I,I).GT.0.)GO TO 220
      WRITE(*,2020)
      STOP
  220 EIGV(I)=A(I,I)/B(I,I)
C     IF (IFPR.EQ.0)GO TO 230
C     WRITE(IOUT,2030)
```

To

```
c      WRITE(IOUT,2010) (EIGV(I),I=1,N)
c
c      CHECK FOR CONVERGENCE
c
  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

c
c      CHECK ALL OFF-DIAGONAL ELEMENTA TO SEE IF ANOTHER SWEEP IS REQUIRED
c
      EPS=RTOL**2
      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
c
c      FILL OUT BOTTOM TRIANGLE OF RESULTANT MATRICES AND SCALE EIGENVECTORS
c
  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=1,N
      BB=DSQRT(B(J,J))
      DO 270 K=1,N
  270 X(K,J)=X(K,J)/BB


*      write(*,*) 'THE EIGENVECTORS ARE :'
*     write(*,*) ((x(i,j),j=1,n),i=1,n)
      return
c     ..................................................................
c
c      UPDATE D MATRIX AMD START NEW SWEEP,IF ALLOWED
c
  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       30H MATRICES NOT POSITIVE DEFINITE )
 2030 FORMAT(36HOCURRENT EIGENVALUES IN *JACOBI* ARE,/)
      END

c     ..................................................................


       subroutine matmat3(a,b,n,iq,temp)
c      xx(n*iq)=xx(n*iq)*phi(iq,iq)
c      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
  20   temp(j) = a((j-1)*n+i)
       do 30 k=1,iq
       sum=0.d0
       do 1=1,iqq,8
       sum=sum + temp(1)*b(1,k)+temp(1+1)*b(1+1,k)
     + + temp(1+2)*b(1+2,k)+temp(1+3)*b(1+3,k)+temp(1+4)*b(1+4,k)
     + + temp(1+5)*b(1+5,k)+temp(1+6)*b(1+6,k)+temp(1+7)*b(1+7,k)
       enddo
       do 40  1=iqq+1,iq
  40   sum = sum + temp(1)*b(1,k)
  30   a((k-1)*n+i)=sum
  10   continue

       return
       end
```

```
c     This version does not have any loop enrolling
      subroutine multspa2(n,ip,indx,coef,diag,rhs,temp)
c     subroutine  multsp(n,ia,ja,an,ad,b,c,isupd,iptrs)
      implicit real*8(a-h,o-z)
      real et(2)
      dimension rhs(1),coef(1),indx(1)
      dimension diag(1),temp(1),ip(1)
      ncoef = ip(n+1)-1
c     ett1=etime(et)
      tt1=et(1)
      do i=1,n
      temp(i)=diag(i)*rhs(i)
      enddo
      do 10 i=1,n
      ifirst=ip(i)
      ilast=ip(i+1)-1
      sum1=temp(i)
C     DIR$ IVDEP
      do k=ifirst,ilast
      kk=indx(k)
      sum1=sum1+coef(k)*rhs(kk)
      temp(kk)=temp(kk)+coef(k)*rhs(i)
      enddo
      temp(i)=sum1
 10   continue
c     ett2=etime(et)
c     tt2=et(1)
c     time2=(tt2-tt1)
c     noper= ncoef*4+n
c     write(*,*) 'Time in Matrix-by-Vector     =    ',time2
      return
      end


      subroutine normcheck (iq,n,neig,ncoef,ia,ja,an,ad,dm,am,eigv,
     +     xx,v,vp1,lump,err,ishift)
      implicit real*8 (a-h,o-z)
      real*8 an(1),ad(1),eigv(1),xx(1),dm(1)
     +     ,err(1),v(1),vp1(1),am(1)
      integer ia(1),ja(1)
      real t01,t02,t03
c.....Purpose : Calculate ||Kx-eigv.Mx||/||K.x||
      call cputime(t01)

c      rewind(15)
c      read(15) (ia(i),i=1,1+n)
c      rewind(11)
c      read(11) (ja(i),i=1,ncoef)
c      rewind(12)
c      read(12) (an(i),i=1,ncoef)
c      rewind(13)
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
 110  VNORM=VNORM+V(I)*V(I)
      WNORM=0.0d0
      if(lump.eq.1) then
C***** Lumped Mass matrix        **************
      do 120 i=1,n
      vp1(I)=dm(i)*xx((k-1)*n+i)
 120  continue
      else
C******  consistant Mass matrix   *************
      call multspa2(n,ia,ja,am,dm,xx((k-1)*n+1),VP1)
      endif

      RT= eigv(k)
      DO 600 I=1,N
      V(I)=V(I)-rt*VP1(I)
600   continue
      do 601 i = 1,n
601   WNORM=WNORM+V(I)*V(I)
      VNORM=DSQRT(VNORM)
      WNORM=DSQRT(WNORM)
      err(k)=WNORM/VNORM
100   CONTINUE
```

```
        call cputime(t02)
        t03=t02 - t01

        write(23,*) ' Time normcheck =  ',t03
        WRITE(23,*)
     +    '***** # **,*** EIGV *** , *** HERTZ ***, ** ERROR NORM ** '
c----
        if(ishift.ne.0) then
          do i=1,neig
          eigv(i)=eigv(i)-float(ishift)
          enddo
        endif

c----

        DO 900 K=1,NEIG
c       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)
900     CONTINUE
        return
        end


        subroutine spasubspace(iq,n,ncoef,neig,lumas,mtot,ik,jk,dk
     +    ,ak,dm,am,iu,ju,di,un,xkk,xmm,phi,eigv,eigv0,tolj
     +    ,temp,xx,ishift)
c       developped by runesha October 20,1995

        implicit real*8(a-h,o-z)
        real*8 ak(1),dk(1),dm(1),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
c       real t1,t2,t3,t4,t5,t6,t7
        real t1,t7

c.....................................................................
c.....Purpose  : This subroutine computes NEIG eigenvalues and eigenvectors
c                of an NxN matrix
c.....This subroutine is called by the main program
c
c.....INPUT    :-ik(n+1),jk(ncoef),ak(ncoef),dk(n) : given upper triangular
c                stiffness matrix in RR(U)U
c               -ik(n+1),jk(ncoef),am(ncoef),dm(n) : given upper triangular
c                mass matrix in RR(U)U
c               -NEIG   : number of desired eigenvectors and eigenvalues
c               -NCOEF  : number  of upper triangular non zero coefficients
c                         diagonal values not included.
c               -LUMAS  : = 0 Case of lumped mass
c                         = 1 case of consistent mass
c.....OUTPUT   :

c.....Working space:
c
c.....................................................................


        call cputime(t1)
c       To limit the number of iterations
        itertot= 100
        ICONV=0
        toler=0.000000000001
        nsmax=12
c       Initialization
        do i=1,iq
        eigv0(i)=0.
        enddo
c       call cputime(t2)

c       write(23,*) '    ************************************************'
c       write(23,*) '    *                                              *'
        write(23,*) '    *   OUTPUT VECTOR-SPARSE SUBSPACE ITERATION    *'
c       write(23,*) '    *                                              *'
c       write(23,*) '    ************************************************'
c       write(23,*)
        write(23,*) 'NEQ  = ',n
        write(23,*) 'NCOEF= ',ncoef
c       write(23,*) 'LUMAS= ',lumas
```

```
c     write(23,*) 'NEIG = ',neig
      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     write(23,*) 'Total memory  = ',memory

c     call cputime(t3)
c.... STARTING ITERATION VECTORS.
c.. The following is the basic just 1 on the diag not even Kii/Mii
cp    ii=n*iq
cp    do i=1,ii
cp    xx(i)=0.d0
cp    enddo
cp    do i=1,iq
cp    xx(i+n*(i-1))=1.d0
cp    enddo

c      KJ Bathe style starting vector Iteration
      nd=n/iq
c     if(lumas.ne.1) go to 4
      j=0
      do 6 i=1,n
      xx(i)=dm(i)
      if(dm(i).gt.0) j=j+1
 6    temp(i)=dm(i)/dk(i)
      if(iq.le.j) go to 16
      write(*,*) ' IQ CANNOT BE LARGER THAN 3 DIAG NON ZERO '
      stop
c 4   do 11 i=1,n4
c     xx(i)=dm(i)
c 11  temp(i)=dm(i)/dk(i)
 16   do 20 i=n+1,n*iq
 20   xx(i)=0
      l=n-nd
      do 30 j=2,iq
       rt=0
      do 40 i=1,l
      if(temp(i).lt.rt) go to 40
      rt=temp(i)
      ij=i
 40   continue
      do 50 i=1,n
      if (temp(i).le.rt) go to 50
      rt=temp(i)
      ij=i
 50   continue
      tolj(j)=float(ij)
      temp(ij)=0.
      l=l-nd
 30   xx((j-1)*n+ij)=1.
c     write(*,*) 'Degrees of freedom excited by unit starting
c    +    iteration vector are:'
c     write(*,*) (tolj(j),j=2,iq)

c     A random vector is added to the last vector
      pi=3.141592654
      xxx=0.5
      iloc=(iq-1)*n
      do 60 k=1,n
      xxx=(pi+xxx)**5
      ix=int(xxx)
      xxx=xxx-float(ix)
 60   xx(iloc+k)=xx(iloc+k)+xxx


c     call cputime(t4)

c     write(*,*) 'STARTING ITERATION VECTORS'
c     do j=1,iq
c     write(*,*) (xx((j-1)*n+i),i=1,n)
c     enddo
c.... End Iteratin vectors


c.... BEGIN SUBSPACE ITERATIONS
c     call cputime(t5)
      n1=n-mod(n,8)
      iter=0
```

```
 10   iter=iter+1

c     CALCULATE THE PROJECTION OF K AND M
      do 110 j=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,n1,8
      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=n1+1,n
 140  sum =sum + xx((i-1)*n+k)*temp(k)
 130  xkk(i,j)=sum
      do 150 k=1,n
 150  xx((j-1)*n+k)=temp(k)
 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,n1,8
      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=n1+1,n
 190  sum =sum + xx((i-1)*n+k)*temp(k)
 180  xmm(i,j)= sum
      if (iconv.gt.0) go to 160
      do 200 k=1,n
 200  xx((j-1)*n+k)=temp(k)
 160  continue
      ELSE
      do 162 j=1,iq
      do i=1,n
      temp(i)=xx((j-1)*n+i)*dm(i)
      enddo
      do 182 i=j,iq
      sum=0.d0
      do k=1,n1,8
      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=n1+1,n
 192  sum =sum + xx((i-1)*n+k)*temp(k)
 182  xmm(i,j)= sum
      if (iconv.gt.0) go to 162
      do 202 k=1,n
 202  xx((j-1)*n+k)=temp(k)
 162  continue
      ENDIF


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

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

c
c     SOLVE FOR EIGENSYSTEM OF SUBSPACE OPERATIONS
```

```
      call jacobi2(xkk,xmm,phi,eigv,iq,temp,toler,nsmax)
c     write(*,*) 'AFTER JACOBI'
c     write(*,*) 'PHI '
c     write(*,*) ((phi(i,j),i=1,iq),j=1,iq)
c     write(*,*) 'EIGV'
c     write(*,*) (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
          phit=phi(l,j)
          phi(l,j)=phi(l,i)
 220      phi(l,i)=phit
      endif
      enddo
      enddo
c.....................

c     write(*,*) 'AFTER ARRANGING IN ASCENDING ORDER '
c     write(*,*) 'PHI '
c     write(*,*) ((phi(i,j),i=1,iq),j=1,iq)
c     write(*,*) 'EIGV'
c     write(*,*) (eigv(i),i=1,iq)


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

c     CHECK FOR CONVERGENCE OF EIGENVALUES

c       call cputime(t6)
       if (iconv.gt.0) go to 500
       do 400 I=1,iq
        diff=dabs(eigv(i)-eigv0(i))
 400    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.itertot) 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
 430    eigv0(i)=eigv(i)
        go to 10
c.......End of iteration
 500    continue
        call cputime(t7)

c500    write(23,*) ' RESULTS FOR EIGENSOLUTION'
        write(23,*) ' RESULTS FOR EIGENSOLUTION'
c       write(23,*) '------------------------------'
c       write(23,*)
        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       do 510 j=1,neig
c510    write(23,*) (xx((j-1)*n+i),i=1,n)
        write(23,*)
        write(23,*) 'TOLERANCE CHECK ON EIGENVALUES '
        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 '
c         write(23,*)
c         write(23,*) ' Time Init + Start. Iter. Vect.  =',(t2-t1)
c     +                 +(t4-t3)
c         write(23,*) ' Time subspace Iter.             =', t6-t5
          write(23,*) ' Time subspace Iter.             =', t7-t1
c
c.....ERROR NORM CALCULATION : ||Kx-eigv.Mx||/||Kx||

       call normcheck(iq,n,neig,ncoef,ik,jk,ak,dk,dm,am,
      +      eigv,xx,temp,eigv0,lumas,tolj,ishift)

       return
       end

c*******************************************************************************
```

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

c      SUBROUTINE SP2LAN2(N,LANMAX,NEIG,UN,DI,IU,JU,W,w1,w2,w4,
c     $TEM,EIG,ERR,VEC,ncoff,ncof2,dmass,Q,lump,ishift
c     + am,ia,ja)
       SUBROUTINE SP2LAN2(N,LANMAX,NEIG,UN,DI,IU,JU,W,w1,w2,w4,
      $TEM,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(1),w1(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)
c      if(neig.ne.0.and.lump.ne.1) then
       rewind(17)
       read(17) (am(i),i=1,ncoff)
c      write(*,*) (am(i),i=1,ncoff)
c      endif


c      j'ai besoin de IA et JA pour le multiplication de la mass
c      les valeurs ont ete detruites pendant la factorization
c      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)
c
c       write(*,*) '_____________________________________'
c       write(*,*)
c       write(*,*) 'Inside Spalan '
c       write(*,*) ' Lump =', lump
c       write(*,*) 'N,LANMAX,NEIG,ncoff,ncof2'
c       write(*,*)  N,LANMAX,NEIG,ncoff,ncof2
c       write(*,*) 'DM =',(dmass(i),i=1,n)
c       write(*,*) 'AM =', (am(i), i=1,ncoff)
c       write(*,*) 'IA =', (ia(i),i=1,1+n)
c       write(*,*) 'JA =', (ja(i),i=1,ncoff)
c       write(*,*)
c       write(*,*) '_____________________________________'


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

       iam=0
       nodes=1
       NBLK = 1
       RTOL = 0.0000000001d0
       NNNN= 4*NEIG
       IF(NNNN.GE.LANMAX)NNNN=LANMAX-1
       MN=0
       DO 61 I=1,LANMAX
       EIG(I)=0.0d0
61     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
       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
       W1(i)=0.0d0
15     CONTINUE
```

```
1501  CONTINUE
c     do 9901 i=1,n
c     w2(i)=dmass(i)*w(i)
c9901 continue
      call multspa2(n,ia,ja,am,dmass,w,w2)
      bet=0.0d0
      do 99011 I=1,N
99011 bet=bet+w2(i)*w(i)
      bet=dsqrt(bet)
      do 99012 I=1,N
      W1(i)=w(i)/BET
      w(i)=0.0d0
      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 JJ=1,LANMAX-1
      DO 1001 I=1,N
1001  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,W2,1,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,W4,1,W2,1)
      bet2=dsqrt(bet2)
c     write(*,*)' 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,W4,1,W2,1)
c     write(*,*)' before 100 : JJ,beta = ',jj,beta
      bet2=dsqrt(bet2)
100   CONTINUE
      DBET2=1.0d0/BET2
      DO 110 I=1,N
      W(I)=W1(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
      BET=BET2
c     write(*,*)'JJ,ALF,BET2,iam=',jj,alf,bet2
      IF(JJ.LT.NNNN) GO TO 50
c     if(nodes.gt.1)call mp_sync(nbuf(4))

c     write(*,*) ' 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.1) GO TO 2211
50    CONTINUE
120    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
c
c*************************************************************************
c

       SUBROUTINE JACOBIP2(N,NEIG,NJ,BET2,V,Q,VEC,
     $ EIG,ERR,AN,HH,RTOL,LANMAX,DMASS,AM,
     $ VP1,NP1,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),VP1(1),V(1),AN(1),ad(1),AM(1)
      integer ia(1),ja(1)
      common/sp2com/nbuf(4)
      COMMON /QIN/ QLAN,EPS
      COMMON /NQIN/ NQL
      COMMON /QJA/ SUB(1000)
      iam=0
      nodes=1
      NSUC=0
      DO 1 I=1,LANMAX
      DO 2 J=1,LANMAX
2     VEC(I,J)=0.0d0
      HH(I)=EIG(I)
      SUB(I)=ERR(I)
      VEC(I,I)=1.0d0
1     CONTINUE
c     if(nodes.gt.1)call mp_sync(nbuf(4))
      CALL TQL2(LANMAX,NJ,EIG,ERR,VEC,IRR,EPS)
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)+abs(vec(j,i))
      ERR(I)=err(i)*0.005d0*ABS(BET2*VEC(NJ,I)/EIG(I))
c     write(*,*)'I,err(i),vec(nj,i)=',i,err(i),vec(nj,i)
20    CONTINUE
      DO 30 I=1,NEIG
C      IF(ERR(I).GT.0.000001.AND.NJ.LT.(LANMAX-1)) GO TO 300
      IF(ERR(I) .GT. RTOL .AND. NJ .LT. (LANMAX-1)) GO TO 300
30    CONTINUE
c     if(nodes.gt.1)call mp_sync(nbuf(4))
      CALL VECTRA(N,NEIG,0,VEC,Q,LANMAX,NJ,V)
c     write(*,*)'IN JACOBQ: VECTRA is done !!!!  '
C     NSUC: =1,succesful
      NSUC=1
c     DO 580 L=1,NEIG
      call cputime(t01)
      ninc=8*nodes
c       do L=1,neig
c       do i=n-nadd+1,n
c       q(i,L)=0.0d0
c       enddo
c       enddo
c     write(*,*)'NADD = ',nadd,n
c     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)


c     write(*,*) '_____NORMCHECK2'
c     write(*,*)
c     write(*,*) 'lanmax,n,neig'
c     write(*,*) lanmax,n,neig
c     write(*,*) 'IA =', (ia(i),i=1,1+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,iq)
c      do 510 j=1,neig
c510   write(*,*) (q(i,j),i=1,n)


      DO 580 L=1,neig
c     call multspa(n,ia,ja,an,ad,q,v)
c     call mulmeiko(n,,,,)
c     if(ninc.gt.8)call mp_sync(nbuf(4))
c     call sp2mul(iam,ninc,n,icolg,maxa,stif,q(1,L),vp1,v)
      call multspa(n,ia,ja,an,ad,q(1,L),v)
c     write(*,*)'Q(i,L) = ',L,(q(i,L),i=1,n)
      VNORM=0.0d0
      DO 590 I=1,N
```

```
590     VNORM=VNORM+V(I)*V(I)
        WNORM=0.0d0
c       do 2239 i=1,n
c       vp1(I)=dmass(i)*Q(i,L)
c2239    continue
         call multspa(n,ia,ja,am,dmass,q(1,L),vp1)
c        write(*,*) 'I, vp1=', i, (vp1(ii),ii=1,n)

              RT=1.0d0/EIG(L)
        DO 600 I=1,N
c       erm=max(abs(v(i)),abs(rt*vp1(i)))
c       error=abs(v(i)-rt*vp1(i))/erm
c       if(error.gt.0.95d0.and.erm.gt.0.00001d0)then
c       write(*,*)'L,I,V(I),,=',L,I,v(i),rt*vp1(i),error,q(i,L)
c       endif
         V(I)=V(I)-RT*VP1(I)
600     continue
c       write(*,*) 'V(I)=', (v(i),i=1,n)

        do 601 i = 1,n
601     WNORM=WNORM+V(I)*V(I)
c       if(iam.eq.0)write(*,*)'VNORM, WNORM = ',vnorm,wnorm,L
        VNORM=DSQRT(VNORM)
        WNORM=DSQRT(WNORM)
        HH(L)=WNORM/VNORM
580     CONTINUE
c       write(*,*) ' HH =' ,(hh(i),i=1,neig)

        call cputime(t02)
        t02=t02 - t01
        WRITE(23,*)'*** K, * EIG*,*HERTZ *,* ERROR *,* NORM *** iam'

        if(ishift.eq.0) then
        DO 700 K=1,NEIG
c       HERTZ=1.0/(2.0*3.1415927*DSQRT(EIG(K)))
        HERTZ=1.0/(2.0*3.1415927*DSQRT(DABS(EIG(K))))
        WRITE(23,701)K,1.0/EIG(K),HERTZ,ERR(K),HH(K)
701     FORMAT(2X,I5,2X,4E15.7)
700     CONTINUE
        else
        DO 705 K=1,NEIG
        EIG(K)=1./EIG(K)-float(ishift)
c       HERTZ=DSQRT(EIG(K))/(2.0*3.1415927)
        HERTZ=DSQRT(DABS(EIG(K)))/(2.0*3.1415927)
c       WRITE(23,701)K,1.0/EIG(K),HERTZ,ERR(K),HH(K)
        WRITE(23,701)K,EIG(K),HERTZ,ERR(K),HH(K)
705     CONTINUE
        endif

        NSUC=1
        if(iam.eq.0)write(23,*) 'JACOBIQ: Steps in IAM = ',nj,iam
        RETURN
300     NNNN=MIN0(LANMAX-1,NJ+3*(NEIG-I)+4)
        NSUC=0
        DO 304 I=1,LANMAX
        EIG(I)=HH(I)
304     ERR(I)=SUB(I)
        RETURN
        END
c
c*****************************************************************
c


        subroutine REORD2(n,ncoff,nreord,mtota,mtoti,a,iq,neig)
        real*8 a(1)
c
c       This is an additional routine to reorder as well the
c       consistant mass
c
c
c       note personel
c       si je met un IF LUM en dehors, il faut commenter celui-ci
c
        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(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),
     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)

c       pierrot
c       IF (LUMP.NE.1) THEN
        call copyk2(n,iq(1),iq(1+n),iq(2*n+1),iq(5*n+4),iq(5*n+4+
c    4            ncoff),a(1+3*ncoff),a(1+2*ncoff),ncoff)
     4            ncoff),a(1),a(1+ncoff),ncoff)
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-time1
        endif
        return
        end
C
C**********************************************************************
c
      subroutine copyk2(n,ia,perm,iu,ju,ja,un,am,ncoff)
      integer ia(1),iu(1),ja(1),ju(1),perm(1)
      real*8 am(1),un(1)

      do i=1,ncoff
      un(i)=0.
      am(i)=0.
      enddo

c     rewind(11)
c     read(11)(ja(i),i=1,ncoff)
      rewind(17)
      read(17)(am(i),i=1,ncoff)
c     rewind(15)
c     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))
                ij0  = i0
                ij00 = j0
                if(j0.LT.i0) then
                   ij0 =  j0
                   ij00 = i0
                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)
```

```
c      write(*,*) ' PERM =', (perm(i),i=1,10)
c      write(*,*) ' IA   =', (IA(I),i=1,10)
c      write(*,*) ' JA   =', (JA(i),i=1,10)
c      write(*,*) ' IU   =', (IU(I),i=1,10)
c      write(*,*) ' JU   =', (JU(i),i=1,10)
c      write(*,*) ' AM   =', (am(i),i=1,10)
c      write(*,*) ' UN   =', (un(i),i=1,10)
       return
       end
c
c-----------------------------------------------------------------------
c

       subroutine REORD3(n,ncoff,nreord,mtota,mtoti,a,iq,neig)
       real*8 a(1)
c
c      This is an additional routine to reorder as well the
c      consistant mass
c
c
c      note personel
c      si je met un IF LUM en dehors, il faut commenter celui-ci
c
       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(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),
     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)

c      pierrot
c      IF (LUMP.NE.1) THEN
       call copyk3(n,iq(1),iq(1+n),iq(2*n+1),iq(5*n+4),iq(5*n+4+
     4           ncoff),a(1+3*ncoff),a(1+2*ncoff),ncoff)
c    4           ncoff),a(1),a(1+ncoff),ncoff)
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-time1
       endif
       return
       end
c
C*********************************************************************
c
       subroutine copyk3(n,ia,perm,iu,ju,ja,un,am,ncoff)
       integer ia(1),iu(1),ja(1),ju(1),perm(1)
       real*8 am(1),un(1)

c      do i=1,ncoff
c      un(i)=0.
```

```
c      am(i)=0.
c      enddo

c      rewind(11)
c      read(11)(ja(i),i=1,ncoff)
       rewind(17)
       read(17)(am(i),i=1,ncoff)
c      rewind(15)
c      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))
                 ij0  = i0
                 ij00 = j0
                 if(j0.LT.i0) then
                    ij0 =  j0
                    ij00 = i0
                 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)


c      write(*,*) ' PERM =', (perm(i),i=1,10)
c      write(*,*) ' IA   =', (IA(I),i=1,10)
c      write(*,*) ' JA   =', (JA(i),i=1,10)
c      write(*,*) ' IU   =', (IU(I),i=1,10)
c      write(*,*) ' JU   =', (JU(i),i=1,10)
c      write(*,*) ' AM   =', (am(i),i=1,10)
c      write(*,*) ' UN   =', (un(i),i=1,10)
       return
       end
c
c-----------------------------------------------------------------------
c
```

```
#
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=
#FFLAGS=  -Mperf
#FFLAGS= -O4 -Knoieee -Mvect
#DEBUG    = -g

.f.o:
        $(F77) $(DEBUG) $(FFLAGS) -c $<

$(EXEC):$(OBJ)
        $(F77) -o $(EXEC) $(OBJ) $(FFLAGS)

source:
        cat $(SOURCE) > source.f
```

```
1234567890
1234567890
     1875.0000000000     87890.625000001     65.917968750000     7.2115384615384D-03
    0.56250000000000     750.00000000000     3750.0000000000     175781.25000000
     131.83593750000     1.4423076923077D-02     1.1250000000000     1500.0000000000
     3749.9999999999     175781.25000000     131.83593750000     1.4423076923077D-02
     1.1250000000000     1500.0000000000     3749.9999999999     175781.25000000
     131.83593750000     1.4423076923077D-02     1.1250000000000     1500.0000000000
     3749.9999999999     175781.25000000     131.83593750000     1.4423076923077D-02
     1.1250000000000     1500.0000000000     3749.9999999999     175781.25000000
     131.83593750000     1.4423076923077D-02     1.1250000000000     1500.0000000000
     3749.9999999999     175781.24999999     131.83593749999     1.4423076923077D-02
     1.1250000000000     1500.0000000000     3750.0117188232     175782.89796981
     131.83717347736     1.4423121995474D-02     1.1250035156470     1500.0046875293
     3750.0117188233     175782.89796982     131.83717347737     1.4423121995474D-02
     1.1250035156470     1500.0046875293     3750.0000000000     175781.25000000
     131.83593750000     1.4423076923077D-02     1.1250000000000     1500.0000000000
     1874.9999999999     87890.624999994     65.917968749995     7.2115384615382D-03
    0.56249999999998     749.99999999998
    -----------------
    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    5.8663999999999D-09  0.  0.
     0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08  0.
     0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732763335000D-08    1.1732763335000D-08    1.1732763335000D-08
     0.  0.  0.    1.1732763335000D-08    1.1732763335000D-08    1.1732763335000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    5.8664000000001D-09    5.8664000000001D-09    5.8664000000001D-09
     0.  0.  0.
    ----------
    66*1.0
       Testing Agoura Hill's small eigen data !!
     3  10  1  66  66  114  0  +1000000.0  0  -1
    nreord, neig, lump, neq, neq, ncoef, itime, ishift, iblock, mread
     1  3  3  1  2  2  1  3  3  1  2  2  1  2  2  1  2  2  1  2  2  1  2
     2  1  2  2  1  2  2  1  2  2  1  3  3  1  2  2  1  3  3  1  2  2  1
     2  2  1  3  3  1  2  2  0  1  1  0  0  0
       5.8663999999999D-09    5.8663999999999D-09    5.8663999999999D-09  0.  0.
     0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08  0.
     0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    1.1732763335000D-08    1.1732763335000D-08    1.1732763335000D-08
     0.  0.  0.    1.1732763335000D-08    1.1732763335000D-08    1.1732763335000D-08
     0.  0.  0.    1.1732800000000D-08    1.1732800000000D-08    1.1732800000000D-08
     0.  0.  0.    5.8664000000001D-09    5.8664000000001D-09    5.8664000000001D-09
     0.  0.  0.
      -1875.0000000000     7031.2500000001    -87890.625000001     7031.2500000001
      -5.2734375000000    -65.917968750000    -5.2734375000000    -7.2115384615384D-03
       5.2734375000000    0.28125000000000    -7031.2500000001     375.00000000000
      -1875.0000000000    -8.7311491370201D-11    -87890.624999999     7031.2500000000
       8.5265128291212D-14    -65.917968750000    -5.2734374999999    -7.2115384615383D-03
       5.2734374999999    0.28125000000000    -7031.2500000000     375.00000000000
      -1875.0000000000    -87890.624999999     7031.2500000000    -65.917968750000
      -5.2734374999999    -7.2115384615383D-03     5.2734374999999    0.28125000000000
      -7031.2500000000     375.00000000000    -1875.0000000000    -87890.624999999
       7031.2500000000    -65.917968750000    -5.2734374999999    -7.2115384615383D-03
       5.2734374999999    0.28125000000000    -7031.2500000000     375.00000000000
      -1875.0000000000    -87890.624999999     7031.2500000000    -65.917968750000
      -5.2734374999999    -7.2115384615383D-03     5.2734374999999    0.28125000000000
      -7031.2500000000     375.00000000000    -1875.0000000000    -87890.624999999
       7031.2500000000    -65.917968750000    -5.2734374999999    -7.2115384615383D-03
       5.2734374999999    0.28125000000000    -7031.2500000000     375.00000000000
      -1874.9999999999    -2.9103830456734D-10    -87890.624999994     7031.2499999997
       2.2737367544323D-13    -65.917968749995    -5.2734374999997    -7.2115384615382D-03
       5.2734374999997    0.28124999999999    -7031.2499999997     375.00000000000
      -1875.0117188232     8.7891449307790D-02    -87892.272969819     7031.3378914490
      -6.5918587040414D-05    -65.919204727364    -5.2735034185868    -7.2115835339355D-03
```

```
   5.2735034185868    0.28125175782348    -7031.3378914490     375.00234376465
  -1875.0000000000    -8.7891448725713D-02    -87890.625000005     7031.2500000003
   6.5918586585667D-05    -65.917968750004    -5.2734375000002    -7.2115384615385D-03
   5.2734375000002    0.28125000000000    -7031.2500000003     375.00000000001
  -1874.9999999999    -5.8207660913467D-10    -87890.624999994     7031.2499999997
   4.5474735088646D-13    -65.917968749995    -5.2734374999997    -7.2115384615382D-03
   5.2734374999997    0.28124999999999    -7031.2499999997     375.00000000000
  -7031.2499999997     5.2734374999997
------------
114*0.0
 7  6  8  12  5  9  11  10  9  11  8  12  13  12  14  18  11  15  17  16  15
 17  14  18  19  20  24  21  23  22  21  23  20  24  25  26  30  27  29  28
 27  29  26  30  31  32  36  33  35  34  33  35  32  36  37  38  42  39  41
 40  39  41  38  42  43  42  44  48  41  45  47  46  45  47  44  48  49  48
 50  54  47  51  53  52  51  53  50  54  55  54  56  60  53  57  59  58  57
 59  56  60  61  60  62  66  59  63  65  64  63  65  62  66  66  65
```