

## **INFORMATION TO USERS**

**This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.**

**The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.**

**In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.**

**Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.**

**ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600**

**UMI<sup>®</sup>**



**LANCZOS ALGORITHM WITH REORTHOGONIZATION FOR SUPERIOR  
SOLUTION IN SOLVING LARGE EIGENVALUE PROBLEM**

**A THESIS**

**Presented to the Department of Mechanical Engineering  
California State University, Long Beach**

**In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science**

**By Montri Ratanajirasut  
B.S., 1998, Kasetsart University in Bangkok**

**December 2002**

UMI Number: 1413260

UMI<sup>®</sup>

---

UMI Microform 1413260

Copyright 2003 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.

---

ProQuest Information and Learning Company

300 North Zeeb Road

P.O. Box 1346

Ann Arbor, MI 48106-1346

WE, THE UNDERSIGNED MEMBERS OF THE COMMITTEE,  
HAVE APPROVED THIS THESIS

LANCZOS ALGORITHM WITH REORTHOGONIZATION FOR SUPERIOR  
SOLUTION IN SOLVING LARGE EIGENVALUE PROBLEM

By

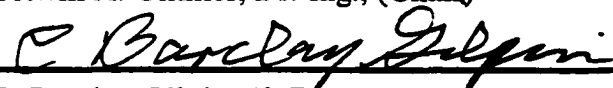
Montri Ratanajirasut

COMMITTEE MEMBERS



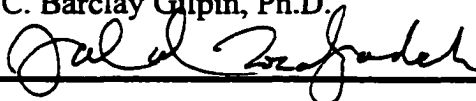
Ortwin A. Ohtmer, Dr.-Ing., (Chair)

Mechanical Engineering



C. Barclay Gilpin, Ph.D.

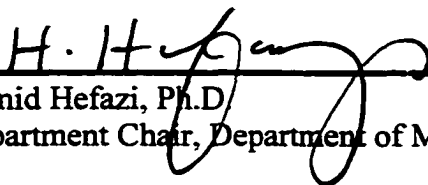
Mechanical Engineering



Torabzadeh Jalal, Ph.D.

Mechanical Engineering

ACCEPTED AND APPROVED ON BEHALF OF THE UNIVERSITY



Hamid Hefazi, Ph.D.

Department Chair, Department of Mechanical and Aerospace Engineering

California State University, Long Beach

December 2002

## ABSTRACT

### LANCZOS ALGORITHM WITH REORTHOGONIZATION FOR SUPERIOR SOLUTION IN SOLVING LARGE EIGENVALUE PROBLEM

By

Montri Ratanajirasut

December 2002

In this thesis, eigenvalue/eigenvector problem is computed by a combination of the most effective eigenproblem solving iteration, Lanczos algorithm, and Rayleigh Ritz analysis. This combination is the fastest method to obtain accurate first few eigenvalues today. The Lanczos algorithm reduces a very large and complex system matrix to the same-sized tridiagonal matrix. The Rayleigh Ritz analysis is used to reverse the eigenvalues of tridiagonal matrix to the eigenvalues of the original system.

Nevertheless, this algorithm has one huge disadvantage when working with a large system or a large number of steps. Unit roundoff error of computer will grow the computation error along the iteration steps that have been computed. The amount of produced error will loosen orthogonality between working vectors as iteration goes on. This is called "Loss of Orthogonality." This error will eventually destroy the accuracy of the result.

The method of reorthogonalization is used to fix this problem. In Lanczos algorithm programming package, written on FORTRAN 90 platform, several subroutines are used to detect loss of orthogonality and perform the

reorthogonalization. The loss of orthogonality is detected when the error bounds of computation grow greater than what is called semi-orthogonality tolerance. The size of semi-orthogonality tolerance is based on computer's precision, unit roundoff error.

With reorthogonalization, accuracy of the computation is controlled. As a result, the precision of the solution is supremely accurate and fast obtained. Furthermore, a few subroutines were written to prevent a critical case when there are 2 very close eigenvalues converged at the same time. This will cause a difficulty to calculate the eigenvector of each eigenvalue.

## ACKNOWLEDGEMENTS

To My family who is the most important driver to push me through everything and makes me the person who I am today.

To all my friends who provide me the special and enormous mental support. It is one of the strongest supports to put me through this thesis.

To my advisor, Dr. Ohtmer who had always been here to give his professionalism when needed. The successful completion of this thesis would have not been found if lack of this great support and advice.



## TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS.....	iii
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
 CHAPTER	
1. FUNDAMENTAL OF EIGENVALUE PROBLEM.....	1
Introduction.....	1
Generalized Eigenvalue/eigenvector Problem and Its Transformation.....	2
Shifting Eigenvalues.....	3
Approximate Solution Techniques for Eigenvalue Problems.....	3
2. THEORETICAL BACKGROUND OF LANZCOS ALGORITHM.....	5
Introduction.....	5
Real Eigenvalue Requirement.....	6
Orthogonality of Eigenvectors.....	6
The Gram-Schmidt Orthogonalization Process.....	8
Solving Linear System of Equations Using Gram-Schmidt...	9
Lanczos Algorithm Derivation.....	11
Simplifying Lanczos Algorithm via Proper Matrix Notation..	17
Rayleigh-Ritz Approximation.....	19
Convergence of Eigenvalues.....	21
3. LOSS OF OTHOGONALITY AND REORTHOGONALIZATION.....	22
Introduction.....	22
Loss-of-Orthogonality Indicator.....	23

CHAPTER	Page
Semi-Orthogonality.....	25
4. INPUT AND OUTPUT AND STRUCTURE OF LANCZOS ALGORITHM PROGRAM.....	26
Input and Output of the Program Package.....	26
Structure of the Lanczos Algorithm Package.....	27
Main Programs.....	30
Subroutines.....	30
Functions.....	44
5. CONCLUSION.....	46
APPENDICES.....	48
A. STRUCTURE FLOWCHART OF LANCZOS PACKAGE'S MAIN PROGRAM, SUBROUTINES AND FUNCTIONS.....	49
B. FORTRAN CODE USING LANCZOS ALGORITHM PACKAGE.....	67
C. DEFINITION OF VARIABLES USED IN LANCZOS ALGORITHM PROGRAM PACKAGE.....	108
D. EXAMPLES OF EIGENVALUE PROBLEM CALCULATION .....	114
REFERENCES.....	119

## LIST OF TABLES

TABLE		Page
1	Definition of the Value of Error Flag.....	32
2	Pointers of Elements in Global Array.....	36

## LIST OF FIGURES

FIGURE		Page
1	Lanczos Algorithm.....	29

**CHAPTER 1**  
**FUNDAMENTAL OF EIGENVALUE PROBLEM**

**Introduction**

Mechanical engineers categorize mechanical engineering problems into two categories, Dynamic and Static system problem. In many mechanical applications, the solution of the problem could be either unique or multiple.

In the case of a system that has multiple solutions or mode shapes, different results of mode shapes occur under different initial conditions, system environments and/or system disturbances. Mechanical engineering problems that deal with multiple results and mode shapes are called “Eigenvalue/Eigenvector problems,” or “Eigenproblems.”

In mechanical engineering problems, Eigenvalue problem is used exclusively in 3 fields of problems; Buckling Analysis, Vibration Analysis and Heat Transfer Analysis.

In Bucking analysis, eigenproblem is used mainly in order to find the critical value that causes a mechanical system to break down by buckling. In Vibration Analysis, it is used mainly in order to find specific critical values or resonance frequencies of each mode shape of a system. In Heat Transfer Analysis, it involves heat capacity and heat conductivity.

The purpose of solving eigenproblems is to find thermal frequencies and corresponding mode shapes, which are eigenvalue and eigenvector respectively.

One eigenproblem might have multiple eigenpairs, which is depending on the size of the system matrices. One eigenpair consists of 1 eigenvalue and its corresponding eigenvector. A standard eigenvalue/eigenvector problem equation can be specified as:

$$[A]\{v\} = \lambda\{v\}$$

when  $[A]$  is a symmetric system matrix

$\{v\}$  is an Eigenvector

$\lambda$  is an eigenvalue

### Generalized Eigenvalue/Eigenvector Problem and Its Transformation

Some mechanical engineering problems are already in the standard eigenproblem form; but many problems are not in the form by nature. Fortunately, generalizing these problems into a form called generalized eigenproblem is possible. Later, these formed equations can be transformed again into the standard eigenproblem form by using matrix factorization technique.

Before starting the detailed derivation, some important properties must be well understood. The Cholesky factorization is described in [10] pages 121-127. The definition of orthogonal vectors is also explained in [10] pages 129-140. Given a dynamic problem in engineering:

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

$[K]$  and  $[M]$  represent Stiffness and Mass-matrices, respectively. The Cholesky factorization applied to the Mass-matrix with  $[C]$ , nonsingular matrix, the result is

$$[M] = [C][C]^T$$

Substituting it into equation (1), obtaining

$$[K]\{v\} = \lambda[C][C]^T \{v\}$$

Then, pre-multiplying by  $[C]^{-1}$  to both sides

$$[C]^{-1}[K]\{v\} = \lambda[C]^{-1}[C][C]^T \{v\}$$

In case of a diagonal mass matrix,  $[M]$ , applying the “lumped” mass matrix approach for large problems,  $[C]$  and  $[C]^{-1}$  are calculated without applying the factorization procedure. The last equation can be rewritten,

$$[\bar{K}]\{\bar{v}\} = \lambda\{\bar{v}\} \quad (2)$$

with  $\{\bar{v}\} = [C]^T \{v\}$  and  $[\bar{K}] = [C]^{-1}[K][C]^{-T}$

### Shifting Eigenvalues

This procedure plays a role of improving the approximation’s precision of the solution. Most of the times, mechanical engineers shift Eigenvalue if rigid body modes are not eliminated by special boundary conditions, the eigenvalues for rigid body modes are 0. To avoid this, shifting is

Let  $[K]_{\sigma} = [K] + \sigma[M]$

with the shift,  $\sigma$ , being a positive number.

Now the shifted eigenvalue problem can be shown below,

$$[K]_{\sigma}\{v\} = \lambda[M]\{v\} \quad (3)$$

or  $[K]\{v\} = (\lambda - \sigma)[M]\{v\}$

where  $\lambda - \sigma = \mu$

### Approximate Solution Techniques for Eigenvalue Problems

In calculation of any mechanical engineering system, it will be too costly or impossible to find an exact solution of a large system, especially with a dynamic system. Because of this reason, mechanical engineers use an efficient approximation method to find a highly accurate solution instead of an exact one.

There are many approximation techniques available to use, such as Static condensation, Discrete Rayleigh-Ritz reduction and Subspace iteration.

Each technique has its own advantages and disadvantages. However, with current technology of high speed and calculation accuracy personal computer, Lanczos Algorithm has 2 big advantages over the others. First, it works in a simple form. Second, it requires a very small number of iterations to pursue fine system's eigenvalues.

With these advantages, mechanical engineers start applying this technique widely in eigenvalue/eigenvector problems especially in finite element method, which deal with a huge number of matrix size.



## CHAPTER 2

### THEORETICAL BACKGROUND OF LANCZOS ALGORITHM

#### Introduction

Even though Lanczos algorithm was originally developed for tridiagonalizing matrices purpose; combining with Rayleigh-Ritz method, the algorithm is a powerful eigenvalue solver.

Once matrices in a generalized eigenproblem form are tridiagonalized, eigenvalues and eigenvectors can be easily calculated by applying Rayleigh-Ritz method.

Lanczos Algorithm has a great advantage due to its simplicity of calculation and spectacularly small number of iterations requirement. However, in the early years of the application of the algorithm, the solution by the approach had disappointed engineers. The precision of the solution was not acceptable. The problem was caused by the computer roundoff error or computer precision. This computer error dramatically propagates the damage of the solution precision every step of iteration. For more information, see [2] on pages 1-13.

Lanczos algorithm is based on the orthogonality property of Lanczos vector series and Ritz vector series calculated with the Mass-Stiffness matrices, see [3]. As the Lanczos iterations are running, the round-off error continues loosening orthogonality of Lanczos vectors and Ritz vectors dramatically. Fortunately, this problem could be

solved by the procedure of reorthogonalization, so the orthogonality of the vectors is recalled. Therefore, precision and reliability of the algorithm are also recalled.

Orthogonality and its recovering are the topics of the following chapter.

### Real Eigenvalue Requirement

Many believe that just because  $[K]$  and  $[M]$  are symmetric; the calculated eigenvalues of the system would be real. Unfortunately, this is not true. The only working criteria to ensure real eigenvalues is that  $(\rho[K] + \tau[M])$  must be positive-definite for some choice of real  $\rho$  and  $\tau$ .

This is the only way to guarantee that the system possesses real eigenvalues. However, it is hard to find out those variables' value. However, both  $[K]$  and  $[M]$  are positive-definite if generated within the Finite Element numerical procedure.

### Orthogonality of Eigenvectors

In an undamped Mechanical system, the characteristic equation can be written as

$$m \ddot{x} = kx$$

or  $\omega^2 mx = kx$

Written for an arbitrary number of degrees of freedom in matrix notation as

$$[K]\{x_i\} = \omega^2 [M]\{x_i\} \tag{4}$$

A discussion of how the characteristic system relates to Lanczos Algorithm is held below.

Assigning index  $i, j$  to the equation (4),

$$[K]\{x_i\} = \omega_i^2 [M]\{x_i\}$$

$$[K]\{x_j\} = \omega_j^2 [M]\{x_j\}$$

Pre-calculating equations above by  $\{x_i\}^T$  and  $\{x_j\}^T$ , they become

$$\{x_j\}^T [K] \{x_i\} = \omega_i^2 \{x_j\}^T [M] \{x_i\} \quad (5)$$

$$\{x_i\}^T [K] \{x_j\} = \omega_j^2 \{x_i\}^T [M] \{x_j\} \quad (6)$$

Multiplying both equations from left by  $[M]^{-1}$

$$\{x_j\}^T [M]^{-1} [K] \{x_i\} = \omega_i^2 \{x_j\}^T \{x_i\} \quad (7)$$

$$\{x_i\}^T [M]^{-1} [K] \{x_j\} = \omega_j^2 \{x_i\}^T \{x_j\} \quad (8)$$

From equation (7) and (8), they become

$$(\omega_i^2 - \omega_j^2) \{x_j\}^T \{x_i\} = 0 \quad (9)$$

Since it is assumed that  $\omega_i \neq \omega_j$ ,

$$\{x_i\}^T \{x_j\} = \delta_{ij} = \begin{cases} (\text{constant}) & \text{for } i=j \\ 0 & \text{for } i \neq j \end{cases} \quad (10)$$

$\delta$  is named Kronecker symbol

It is said the eigenvectors are orthogonal. Applying (10) into (7), becoming

$$\{x_j\}^T [[M]^{-1} [K]] \{x_i\} = \omega_i^2 \{x_j\}^T \{x_i\} \quad (11)$$

or 
$$[[M]^{-1} [K]] \{x_i\} = \omega_i^2 \{x_i\} \quad (12)$$

or 
$$\{x_j\}^T [K] \{x_i\} = \omega_i^2 \{x_j\}^T [M] \{x_i\} \quad (13)$$

comparing (12) and (13),

$$\{x_j\}^T [M] \{x_i\} = \zeta_{ij} \quad (\text{eigenvectors are also mass-orthogonal}) \quad (14)$$

$$\{x_j\}^T [K] \{x_i\} = \omega_i^2 \zeta_{ij} \quad (\text{eigenvectors are also stiffness-orthogonal}) \quad (15)$$

$$\{\hat{\phi}\}^T [M] \{\hat{\phi}\} = [I] \quad \text{Transformation to generalized coordinates} \quad (16)$$

$$\{\hat{\phi}\}^T [K] \{\hat{\phi}\} = [\omega^2] \quad \text{Transformation to generalized coordinates} \quad (17)$$

with

$$\{\hat{\phi}\} = [\phi] \left[ [\phi]^T [M [\phi]] \right]^{-1/2} ; [\omega^2] = \begin{bmatrix} \omega_1^2 & 0 & 0 & 0 & 0 \\ 0 & \omega_2^2 & 0 & 0 & 0 \\ 0 & 0 & . & 0 & 0 \\ 0 & 0 & 0 & . & 0 \\ 0 & 0 & 0 & 0 & \omega_n^2 \end{bmatrix}$$

$$[\phi] = [\{x_1\} \{x_2\} \{x_3\} \dots \{x_n\}]$$

The powerful “Modal-Analysis” in dynamic is based on equation (14) to (17).

### The Gram-Schmidt Orthogonalization Process

This procedure is very useful and widely used for solving a large-not bended linear system of equations. Also, it is used for orthogonalization in Lanczos algorithm. The formulas below are named Gram-Schmidt procedure.

The resulted vectors of this procedure are all orthogonal to each other. Solving a linear system of equation using Gram-Schmidt orthogonalization process is an effective method in solving system of equation with full coefficient matrices. For bended-matrices, the Cholesky-Gauss procedure is more efficient. The procedure transforms the linear system into an upper-diagonal matrix, which can be solved easily. The Gram-Schmidt orthogonalization procedure is specified below.

For :  $1 \leq i \leq m$

$$\{\phi_i\} = a_{1,i} \{\phi_1\} + a_{2,i} \{\phi_2\} + \dots + a_{i-1,i} \{\phi_{i-1}\} + a_{i,i} \{f_i\}$$

With

$$a_{1,i} = \frac{-\left(\{f_i\}^T \{\phi_1\}\right)}{N_i} ; a_{2,i} = \frac{-\left(\{f_i\}^T \{\phi_2\}\right)}{N_i} ; a_{i-1,i} = \frac{-\left(\{f_i\}^T \{\phi_{i-1}\}\right)}{N_i} ; a_{i,i} = \frac{1}{N_i}$$

$$N_i = \left( (\{f_i\}^T \{f_i\}) - (\{f_i\}^T \{\phi_i\})^2 - (\{f_i\}^T \{\phi_2\})^2 - \dots - (\{f_i\}^T \{\phi_{i-1}\})^2 \right)^{\frac{1}{2}} \quad (18)$$

### Solving Linear System of Equation Using Gram-Schmidt

Here is an application of Gram-Schmidt procedure to solve a linear system of equations. For instance, one simple will be solved. Given a linear system of equation

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 5/2 & -1 \\ 0 & -1 & 4/3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \sqrt{2} \begin{Bmatrix} 3 \\ 2 \\ 1 \end{Bmatrix}$$

The equation can be rewritten as

$$\{f_1\}x_1 + \{f_2\}x_2 + \{f_3\}x_3 = \{C\} \quad (19)$$

when

$$\{f_1\} = \begin{Bmatrix} 2 \\ -1 \\ 0 \end{Bmatrix}; \quad \{f_2\} = \begin{Bmatrix} -1 \\ 5/2 \\ -1 \end{Bmatrix}; \quad \{f_3\} = \begin{Bmatrix} 0 \\ -1 \\ 4/3 \end{Bmatrix}$$

and

$$\{C\} = \sqrt{2} \begin{Bmatrix} 3 \\ 2 \\ 1 \end{Bmatrix}$$

Applying Gram-Schmidt orthogonalization to the vector  $f_i$ ,

$i = 1$ ;

$$N_1 = \sqrt{5}; \quad a_{1,1} = \frac{1}{\sqrt{5}}$$

$$\phi_1 = \begin{Bmatrix} 0.894 \\ -0.447 \\ 0 \end{Bmatrix}$$

$i = 2$ ;

$$N_2 = \left( \{f_2\}^T \{f_2\} - (\{f_2\}^T \{\phi_1\})^2 \right)^{1/2} = ((33/4) - 4.046)^{1/2} \\ = 2.050$$

$$a_{1,2} = 0.981$$

$$i = 3; \quad \{\phi_2\} = \begin{Bmatrix} 0.389 \\ 0.781 \\ -0.488 \end{Bmatrix}$$

$$N_3 = 0.726$$

$$a_{1,3} = -0.616 ; a_{2,3} = 1.972$$

$$\phi_3 = \begin{Bmatrix} 0.216 \\ -0.439 \\ 0.874 \end{Bmatrix}$$

Rewriting  $\{f_1\}x_1 + \{f_2\}x_2 + \{f_3\}x_3 = \{C\}$  as

$$N_1 \{\phi_1\}x_1 + \{N_2 \{\phi_2\} + (\{f_2\}^T \{\phi_1\})\{\phi_1\}\}x_2 + \{N_3 \{\phi_3\} + (\{f_3\}^T \{\phi_1\})\{\phi_1\} + (\{f_3\}^T \{\phi_2\})\{\phi_2\}\}x_3 = \{c\}$$

or  $(u_{11} \{\phi_1\})x_1 + (u_{12} \{\phi_1\} + u_{22} \{\phi_2\})x_2 + (u_{13} \{\phi_1\} + u_{23} \{\phi_2\} + u_{33} \{\phi_3\})x_3 = \{c\}$  (20)

with

$$u_{11} = N_1 \quad u_{12} = \{\{f_2\}^T \{\phi_1\}\} \quad u_{13} = \{\{f_3\}^T \{\phi_1\}\}$$

$$u_{22} = N_2 \quad u_{23} = \{\{f_3\}^T \{\phi_2\}\}$$

$$u_{33} = N_3$$

Let  $[Q] = [\{\phi_1\} \{\phi_2\} \{\phi_3\}]$

with

$$[Q]^T [Q] = [I] \tag{21}$$

obtaining,

$$[A] = [Q][U]$$

$$[Q]^T [Q][U]\{x\} = \{c\}$$

$$[U]\{x\} = [Q]^T \{c\} = \{c\}^* \tag{22}$$

Note that [U] is an upper triangular matrix which is easy to solve. Now, use backward-substitution to solve for {X}, which is a vector of  $x_1$ ,  $x_2$  and  $x_3$

$$u_{12} = -2.012$$

$$u_{22} = 2.050$$

$$u_{23} = -1.432$$

$$u_{33} = 0.726$$

The equation of matrices below represents matrices in equation (22) directly.

$$\begin{bmatrix} 2.236 & -2.012 & 0.447 \\ 0 & 2.050 & -1.432 \\ 0 & 0 & 0.726 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{bmatrix} 0.894 & -0.447 & 0 \\ 0.389 & 0.781 & -0.488 \\ 0.216 & 0.439 & 0.874 \end{bmatrix} \begin{Bmatrix} 4.243 \\ 2.828 \\ 1.414 \end{Bmatrix} = \begin{Bmatrix} 2.529 \\ 3.170 \\ 3.394 \end{Bmatrix}$$

solving by backward substitution,

$$x_3 = 4.675$$

$$x_2 = 4.812$$

$$x_1 = 4.527$$

The solution is very close to the actual solution. However, it should be more precise than shown. It is due to round-off error and limitation of 4-decimal calculation.

$$\{X\} = \begin{Bmatrix} 4.525 \\ 4.807 \\ 4.666 \end{Bmatrix}$$

### Lanczos Algorithm Derivation

Lanczos algorithm is used for developing a tridiagonal matrix from the generalized eigenvector system. In general mechanical problems, [K] and [M] are calculated from the mechanical system using the Finite Element Method. Also, shifting of eigenvalues can be possible as in subchapter described earlier.

$$[K_\sigma]\{v\} = \lambda[M]\{v\} \quad (23)$$

with  $[K_\sigma] = [K] - \sigma[M]$

, and  $\sigma$  is the shifting value

With the provided starting vector,  $\{r\}$ , basic iterations such as inverse iteration or power method calculate a sequence of vectors called Krylov sequence below.

$$\{\{r\}, [K_\sigma]^{-1}[M]\{r\}, [[K_\sigma]^{-1}[M]]^2\{r\}, \dots, [[K_\sigma]^{-1}[M]]^j\{r\}\} \quad (24)$$

Similarly, Lanczos iteration constructs a sequence as Krylov sequence as well.

But, the difference is Lanczos iteration uses the successful vector in sequence and Gram-Schmidt orthogonalization process to obtain the best approximation of eigenvector. Instead of the sequence above, Lanczos iteration will compute the sequence below

$$\{\{r\}_0, \{[K_\sigma]^{-1}[M]\{r\}\}_1, \{[[K_\sigma]^{-1}[M]]^2\{r\}\}_2, \dots, \{[[K_\sigma]^{-1}[M]]^j\{r\}\}_j\} \quad (25)$$

If the stiffness matrix cannot be inverted for each step within the Krylov sequence a linear system of equations must be solved.

$$[K_\sigma]\{r\}_1 = [M]\{r\}_0$$

The ongoing elements of this sequence are computed with Gram-Schmidt Orthogonalization process, which is described previously. Note that each vector,  $\{r\}_j$ , after the Mass-Stiffness orthogonalization is named  $\{q\}_j$ . During each step of Lanczos iteration,  $\alpha_i$  and  $\beta_i$  are obtained. These values are components of a tridiagonalized matrix.



The derivation of the Lanczos algorithm will be discussed in more detail.

Assuming that the first  $j$  Lanczos steps have been performed and  $\alpha_i$ ,  $\beta_i$  and Lanczos vectors,  $\{q\}_j$ , with  $i \leq j$ , have been found and the construction of  $\{q\}_{j+1}$  is undertaken.

Because of their Mass-Stiffness orthogonality property, all Lanczos vectors must satisfy the condition below.

$$\{q\}_i^T [M] \{q\}_j = \delta_{ij} = \begin{cases} 1 & \text{for with } i = j, \\ 0 & \text{for with } i \neq j \end{cases}$$

with  $\delta_{ij}$  the Kronecker symbol

To calculate  $\{q\}_{j+1}$ , firstly orthogonalization of  $\{v\}_j$  against the  $j$  Lanczos vector, is taken  $\{q\}_j$

$$\{v\}_j = \left\{ [K_\sigma]^{-1} [M] \{r\} \right\}_j \quad (26)$$

Due to the definition

$$\begin{aligned} \{v_j\} &= \left\{ [K_\sigma]^{-1} [M] \{r\} \right\}_j \\ &= \left\{ [K_\sigma]^{-1} [M] \left\{ [K_\sigma]^{-1} [M] \right\}_{j-1} \{r\}_{j-1} \right\}_j \\ &= \left\{ [K_\sigma]^{-1} [M] \right\}_j \{v\}_{j-1} \end{aligned} \quad (27)$$

Similarly,  $v_{j-1}$  is also the vector that was M-orthogonalized against the 1<sup>st</sup>  $j-1$  Lanczos vectors (  $\{q\}_{j-1}, \{q\}_{j-2}, \{q\}_{j-3}, \dots$  ) to obtain  $\{q\}_j$ . Therefore,

$$\{v\}_{j-1} = \sum_{i=1}^j \{v\}_i \{q\}_i \quad (28)$$

From equation (27) and (28),

$$\{v\}_j = \sum_{i=1}^j \{v\}_i \left\{ [K_\sigma]^{-1} [M] \right\}_j \{q\}_i \quad (29)$$

From equation (28) and (29),

$$\{v\}_j = \{r\}_j + \sum_{i=1}^j \{v\}_i \quad (30)$$

Recall (28) and (29), instead of summing up to  $j-1$ , the 2<sup>nd</sup> term of (30) can be written as

$$\{v\}_j = \{v\}_j + \sum_{i=1}^j \{\bar{v}\}_i \{q\}_i$$

Now, it is defined that

$$\{\bar{v}\}_j = [K]^{-1} [M] \{q\}_j \quad (31)$$

In general, it is assumed that this vector contains elements from each of preceded vectors, so

$$\{\bar{v}\}_j = \{r\}_j + \alpha_j \{q\}_j + \beta_{j-1} + \dots \quad (32)$$

$\{r\}_j$  is called pure component of  $\{\bar{v}\}_j$

$\alpha_j$  and  $\beta_j$  are amplitudes of preceding Lanczos vector contained in  $\{\bar{v}\}_j$

multiplying  $\{\bar{v}\}_j$  by  $\{q\}_j^T [M]$ , obtaining

$$\{q\}_j^T [M] \{\bar{v}\}_j = \{q\}_j^T [M] \{r\}_j + \alpha_j \{q\}_j^T [M] \{q\}_j + \beta_j \{q\}_j^T [M] \{q\}_{j-1} + \dots \quad (33)$$

By the definition of Kronecker delta,  $\delta_{ij}$ , all terms except  $\{q\}_j^T [M] \{q\}_j = 1$  are equal to 0, so the equation can be reduced to

$$\{q\}_j^T [M] \{\bar{v}\}_j = \alpha_j \quad (34)$$

or  $\alpha_j = \{q\}_j^T [M] \{\bar{v}\}_j$  (35)

Similarly, multiplying  $\{\bar{v}\}_j$  by  $\{q\}_{j-1}^T [M]$ , it becomes

$$\beta_j = \{q\}_{j-1}^T [M] \{\bar{v}\}_j \quad (36)$$

From equation (31) for  $j$  and  $j-1$ ,

$$\{\bar{r}\}_j = [K]_\sigma^{-1} [M] \{q\}_j,$$

$$\{\bar{r}\}_{j-1} = [K]_\sigma^{-1} [M] \{q\}_{j-1}$$

Substituting  $\{\bar{r}\}_j$  into (35), it becomes

$$\begin{aligned} \beta_j &= \{q\}_{j-1}^T [M] [K]_\sigma^{-1} [M] \{q\}_j \\ &= \{\bar{r}\}_{j-1}^T [M] \{q\}_j \end{aligned}$$

From equation (32), considering step  $(j-1)^{\text{th}}$  and then multiply the equation by

$\{q\}_j^T [M]$  and transpose the result, it becomes

$$\beta_j = \{q\}_j^T [M] \{r\}_{j-1} + \alpha_{j-1} \{q\}_j^T [M] \{q\}_{j-1} + \beta_{j-1} \{q\}_j^T [M] \{q\}_{j-2} + \dots$$

By the definition of Kronecker delta, all terms except 1<sup>st</sup> term on right hand side of the equation are zero. Therefore,

$$\beta_j = \{q\}_j^T [M] \{r\}_{j-1} \quad (36)$$

and,  $q_j$  is the vector from normalizing,  $r_{j-1}$ . Therefore,

$$q_j = \frac{\{r\}_{j-1}}{\|\{r\}_{j-1}\|_M} \quad (37)$$

where  $\|\{r\}_{j-1}\| = (\{r\}_{j-1}^T [M] \{r\}_{j-1})^{1/2}$

Substituting equation (37) into (36),

$$\beta_j = \frac{\{r\}_{j-1}^T [M] \{r\}_{j-1}}{\|\{r\}_{j-1}\|}$$

$$\text{or } \beta_j = (\{r\}_{j-1}^T [M] \{r\}_{j-1})^{1/2} \quad (38)$$

Note: the 3<sup>rd</sup> term of equation (33) will disappear due to the definition of Kronecker delta when applying procedure similarly to the first 2 terms.

As the Lanczos steps go on,  $\alpha_j$  and  $\beta_j$  are found in every step  $j^{\text{th}}$ . Once all  $\alpha_j$  and  $\beta_j$  are found, forming a tridiagonal matrix is taken as the next step. From equation (31) and (32),

$$\{r\}_j = [K]_{\sigma}^{-1} [M] \{q\}_j - \{q\}_j \alpha_j + \{q\}_{j-1} \beta_j \quad (39)$$

It is assumed that our system matrix size is  $m \times m$ . For  $j \leq m$ , rearranging the series of equation (39) in a global matrix, it can be shown as

$$\begin{aligned} [0:0:0:\dots:0:0:r_m] &= [[K]_{\sigma}^{-1} [M]] [Q]_m - [Q]_m [T]_m \\ \{r\}_m \{e\}_m^T &= [[K]_{\sigma}^{-1} [M]] [Q]_m - [Q]_m [T]_m \end{aligned} \quad (40)$$

with  $\{e\}_m^T = \{0,0,0,0,\dots,0,1\}$

where

$$[T]_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & & & & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & & & & & & \\ & \beta_3 & \bullet & \bullet & & & & & & & \\ & & \bullet & \bullet & \bullet & & & & & & \\ & & & \bullet & \bullet & \bullet & & & & & \\ & & & & \bullet & \bullet & \bullet & & & & \\ & & & & & \bullet & \alpha_{m-1} & \beta_m & & & \\ & & & & & & \beta_m & \alpha_m & & & \end{bmatrix} \quad (41)$$

From orthogonality property,  $[Q]_m^T [M] [Q]_m = [I]_m$ , where  $[I]_m$  is a  $m \times m$  unit matrix. Then, multiplying equation (17) by  $[Q]_m^T [M]$ , and obtain

$$0 = [Q]_m^T [M] [K]_{\sigma}^{-1} [M] [Q]_m - [T]_m$$

$$[T]_m = [Q]_m^T [M [K]_\sigma^{-1} [M [Q]_m] \quad (42)$$

Now, applying Rayleigh-Ritz Approximation to obtain eigenvalues of the reduced tridiagonal matrix is possible, which are the reverses of eigenvalues of the original system. The Lanczos algorithm is inefficient when a complete matrix is to be tridiagonalized. Other techniques such as the householder method are significantly more efficient.

If the objective is to calculate only a few eigenvalues and corresponding, eigenvectors of the problem  $[K]\{\phi_i\} = \lambda[M]\{\phi_i\}$ , an iteration based on the Lanczos transformation is very efficient or the most efficient algorithm if integrated with the Rayleigh-Ritz approximation.

#### Simplifying Lanczos Algorithm via Proper Matrix Notation

The standard Lanczos algorithm transforms a symmetric matrix or eigenvalue problem  $[K]\{x\} = \omega^2 [M]\{x\}$  into a symmetric tridiagonal form. The method being described changes the procedure so that a symmetric tridiagonal matrix is created. It is a modified version of the matrix notation in [1]. If

$Y = [\{y_1\}, \{y_2\}, \dots, \{y_n\}]$  is set of mass-orthogonal vectors,  $\{y_i\}$ . The transformation to tridiagonal form may be described by

$$[K] [Y] = [M] [Y] \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \cdot & \cdot & \cdot \\ & & \beta_{n-1} & \alpha_n \end{bmatrix} \quad (43)$$

Multiplying the equation from left by  $[Y]^T$ :

$$[Y]^T [K] [Y] = [Y]^T [M] [Y] \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \cdot & \cdot & \cdot \\ & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

and  $[Y]^T [M] [Y]$  is a unit matrix, so

$$[Y]^T [K] [Y] = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \cdot & \cdot & \cdot \\ & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

The matrix  $[Y]$  can therefore be named a "Transformation matrix." The Lanczos transformation may be written as  $p$  vector equations, which  $p$  is much smaller than  $n$ .  $N$  is the size of matrices  $[K]$  and  $[M]$ . Picking an arbitrary starting vector  $\{x\}$  and calculating  $\alpha = (\{x\}^T [M] \{x\})^{\frac{1}{2}}$ ;  $\{x_1\} = \frac{1}{\alpha} \{x\}$ ; for  $i = 1 \dots p$ , the following equations are calculated. Starting with  $\beta = 0$  solve within the do-loop the linear system of equations:

$$[K] \{\bar{x}_i\} = [M] \{x_i\} = \{l_i\}; \{l_i\} \text{ is loading case I} \quad (44)$$

Calculation of coefficients  $\gamma_i$

$$\alpha_i = \{\bar{x}_i\}^T [M] \{x_i\} = (\{\bar{x}_i\}^T \{l_i\}) \quad (45)$$

Calculation of the following equations within the do-loop if  $i \neq p$

$$\{\bar{x}_i\} = \{\bar{x}_i\} - \alpha_i \{x_i\} - \beta_{i-1} \{x_{i-1}\} \quad (46)$$

$$\beta_i = \left( \{\bar{x}_i\}^T [M] \{\bar{x}_i\} \right)^{\frac{1}{2}} \quad (47)$$

$$\{x_{i+1}\} = \frac{1}{\beta_i} \{\bar{x}_i\} \quad (48)$$

Equation (44) indicates that the “Lanczos Transformation” requires only the solution of a linear system of equations with  $p$  loading cases. The generation of the mass-orthonormalization of the vectors  $\{x_{i+1}\}$  is less time consuming than the standard Gram-Schmidt procedure, since the applied vector  $\{x_i\}$  in (44) is already mass-orthonormal compared to vector  $\{f_i\}$  in (18).  $\beta_i$  in (47) represents  $\{f_i\}^T \{f_i\}$  in (18). The other terms  $\{f_i\}^T \{\phi_j\}$  are zero due to orthogonality.  $\alpha_i$  in (b) represents  $\{f_i\}^T \{\phi_j\}$  in () with  $\{f_i\} = \{\bar{x}_i\}^T$ .  $\{\bar{x}_i\}^T$  is not mass-orthogonal.

### Rayleigh-Ritz Approximation

From a generalized eigenvalue problem,

$$\boxed{[K]_{\sigma} \{y\} = \lambda [M] \{y\}}$$

let  $\{y\} = [X]_m \{s\} = \sum_{i=1}^m \{x\}_i \{s\}_i$

and,  $\theta$  being the eigenvalue

now, defining a residual vector,

$$\{r\} = [K]_{\sigma} \{y\} - \theta [M] \{y\} \tag{49}$$

Rayleigh-Ritz method requires this residual vector to be orthogonal to every starting vector,  $\{x\}_m$ . Thus,

$$\{x\}_m^T \{r\} = \{x\}_m^T [K] \{y\} - \theta \{x\}_m^T [K]_{\sigma} \{y\} = 0$$

, let  $[K]_m = \{x\}_m^T [K] \{x\}_m$  and  $[M]_m = \{x\}_m^T [M] \{x\}_m$

The eigenproblem can be reduced above to

$$[[K]_{\sigma m} - \theta [M]_m] \{s\} = 0$$

Consider a generalized eigenproblem,

$$\begin{aligned}
 [K]_{\sigma} \{x\} &= \lambda [M] \{x\} \\
 \frac{\{x\}}{\lambda} &= [K]_{\sigma}^{-1} [M] \{x\} \\
 \{0\} &= \left[ [K]_{\sigma}^{-1} [M] - \frac{1}{\lambda} [I] \right] \{x\}
 \end{aligned} \tag{50}$$

Then, the residual vector can be calculated by (45),

$$\{r\}_i = [K]_{\sigma}^{-1} [M] \{y\}_i - \theta_i \{y\}_i \tag{51}$$

This residual vector is M-orthogonal to the set of Lanczos vectors, therefore the eigenpair,  $\{\theta_i, \{y\}_i\}$ , is a Ritz pair. Now, recall that  $[Q]_m = [X]_m$

when  $i = 1, 2, \dots, m$ .

$$\begin{aligned}
 \{0\} &= [Q]_m^T [M] \left( [K]_{\sigma}^{-1} [M] [Q]_m \{s\}_i - \theta_i [Q] \{s\}_i \right) \\
 \{0\} &= [[T]_m - \theta_i [Q]] \{s\}_i
 \end{aligned} \tag{52}$$

Comparing equation (47) to (45), eigenvalues and eigenvectors of the generalized eigenproblem can be calculated by

$$\lambda_i = \frac{1}{\theta_i}$$

and,  $\{y\}_i = [Q]_i \{s\}_i$

If the eigenvalues have been shifted, eigenvalues of the original problem can be found by

$$\lambda_i = \frac{1}{\theta_i} + \sigma$$



## Convergence of Eigenvalues

Accuracy of eigenvalue calculation increases as numbers of Lanczos steps are performed. In this thesis, size of Ritz residual vector is used as an indicator to tell whether the eigenvalue being calculated meets the required accuracy or not. The level of accuracy is set as a tolerance. Accuracy is higher when this number is smaller. The accuracy is met when residual vector norm is less than tolerance.

$$\{r\} = [K]_{\sigma}^{-1} [M] \{y\} - \theta \{y\}$$

$$\{\theta_i, \{y\}_i\} \text{ is an eigenpair}$$

Post multiplying (40) with  $\{s\}_m$ , obtaining

$$[K]_{\sigma}^{-1} [M] [Q]_m \{s\}_i - [Q]_m [T]_m \{s\}_i = \{r\}_m \{e\}_m^T \{s\}_i$$

Taking norm,

$$\begin{aligned} \|[K]_{\sigma}^{-1} [M] [Q]_m \{s\}_i - [Q]_m [T]_m \{s\}_i\|_m &= \|\{r\}_m \{e\}_m^T \{s\}_i\|_m \\ &= \|\{r\}_m\| \left| \{e\}_m^T \{s\}_i \right| \\ &= \|\{r\}_m\| |\varsigma_i| \end{aligned} \tag{55}$$

where  $\varsigma_i$  is the last element of  $\{s\}_i$

and  $\|\{r\}_m\|_m$  is  $\beta_{m+1}$ , so

$$\rho_{mi} = \beta_{m+1} |\varsigma_i|$$

and, eigenvalue meets required accuracy when

$$\rho_{mi} \leq \textit{tolerance}$$

## CHAPTER 3

### LOSS OF ORTHOGONIZATION AND REORTHOGONIZATION

#### Introduction

Even though, central processing unit in computer nowadays can calculate with a very high precision, it is still not an exact calculation. The amount of error of each calculation is called a unit roundoff error.

This error, when propagating along on going Lanczos algorithm steps, can cause a huge error to the orthogonality of Lanczos vectors during M-orthogonalization step. Eventually, the error will keep multiplying itself until Lanczos vectors lose orthogonality to each other. As a result, solution's accuracy will never be met.

The ratio of residual vector and  $[K]_{\sigma}^{-1}[M] \{q\}_j$ , can represent sine angle between those 2 vectors. Theoretically, these 2 vectors are orthogonal to each other. Therefore, this ratio should be equal to 1.

$$\begin{aligned} X_j^2 &= \frac{\|\{r\}_j\|_m^2}{\|[K]_{\sigma}^{-1}[M] \{q\}_j\|_m^2} \\ &= \frac{\beta_{j+1}^2}{\beta_{j+1}^2 + \alpha_j^2 + \beta_j^2} \end{aligned} \quad (53)$$

However, in computation this ratio is not exact one. It is affected by the roundoff error. This ratio, which represents orthogonality, decreases when those 2 vectors are less and less orthogonal. When it goes down close to 0, it indicates

that they are almost parallel. In other word, they lose orthogonality. Therefore, residue vector,  $\{r\}_j$ , should be orthogonalized again against  $\{q\}_{j-1}$  and  $\{q\}_j$ . This is called orthogonality repairing or “Reorthogonalization.”

### Loss-of-Orthogonality Indicator

From theorem, Kronecker delta,  $\delta_{ij}$ , is equal to 0 or 1, when  $i = j$  or  $i \neq j$  respectively. However, this condition will be held in exact calculation only. In computations, this delta is affected by roundoff errors. By the fact that when Kronecker delta,  $i \neq j$ , grow far away from 0 along steps of iteration, the error already have grown too much, this number can be a great indicator of calculation precision or amount of loss of orthogonality.

$$\text{Let } [Q]_m = [ \{q\}_1, \{q\}_2, \{q\}_3, \dots, \{q\}_m ]$$

$$\{H\}_m = [Q]_m^T [M] [Q]_m$$

$$\{H\}_m = [ \{h\}_1, \{h\}_2, \{h\}_3, \dots, \{h\}_m ]$$

$$\text{where } \{h\}_i = \begin{bmatrix} \{q\}_i^T [M] \{q\}_1 \\ \{q\}_i^T [M] \{q\}_2 \\ \vdots \\ \{q\}_i^T [M] \{q\}_m \end{bmatrix}$$

where  $\{q\}_i$ ,  $i = 1, 2, \dots, m$ , are Lanczos vectors of step  $i$ .

$\{h\}_i$ ,  $i = 1, 2, \dots, m$ , are loss-of-orthogonality indicators

From equation (32) and (34),

$$\beta_{j+1} q_{j+1} = [K]_{\sigma}^{-1} [M] \{q\}_j - \{\{q\}_j \alpha_j + \{q\}_{j-1} \beta_j\} \quad (54)$$

Elements in  $\{H\}_m$  can be represented as

$$\mu_{ij} = \{q\}_i^T [M] \{q\}_j$$

Pre-multiplying equation (49) by  $\{q\}_i^T [M]$  and rearranging equation,

$$\begin{aligned} & [\{q\}_i^T [M]] [K]_{\sigma}^{-1} [M] \{q\}_j \\ &= \{q\}_i^T [M] \{q\}_{j+1} \beta_{j+1} + \{q\}_j^T [M] \{q\}_j \alpha_j + \{q\}_j^T [M] \{q\}_{j-1} \beta_j \\ &= \beta_{j+1} \mu_{i,j+1} + \alpha_j \mu_{i,j} + \beta_j \mu_{i,j-1} \end{aligned} \quad (55)$$

Similarly,

$$[\{q\}_j^T [M]] [K]_{\sigma}^{-1} [M] \{q\}_i = \beta_{i+1} \mu_{j,i+1} + \alpha_i \mu_{j,i} + \beta_i \mu_{j,i-1} \quad (56)$$

Because  $MK_{\sigma}^{-1}M$  is symmetric, left side of equations (50) and (51) are equal.

Therefore,

$$\beta_{j+1} \mu_{j+1,i} \cong \beta_{i+1} \mu_{j,i+1} + (\alpha_i - \alpha_j) \mu_{j,i} + \beta_i \mu_{j,i-1} - \beta_j \mu_{j-1,i} \quad (57)$$

At the 1<sup>st</sup> step, it is assumed that the value of elements in  $H_m$  has a value of 1 and the unit roundoff error,  $\epsilon$ , when  $i = j$  and  $i \neq j$  respectively.

Rearranging equation (52), it becomes

$$\beta_{j+1} \mu_{j+1,i} \cong (\beta_{i+1} \mu_{j,i+1} + \alpha_i \mu_{j,i} + \beta_i \mu_{j,i-1}) + \beta_i \mu_{j,i-1} - \alpha_j \mu_{j,i} \quad (58)$$

From (34), the series of the parenthesis, when  $i \geq 2$ , can be written as

$[T]_{j-1} \{h\}_{j-}$ . So, equation (53) after rearranging is

$$\beta_{j+1}\mu_{j+1,i} \equiv [T]_{j-1} \{h\}_j + \beta_i\mu_{j,i-1} - \alpha_j\mu_{j,i} \quad (59)$$

Because after each Lanczos step,  $\beta_j$ ,  $\beta_{j-1}$  and  $\alpha_j$  are found, it is easy to calculate loss of orthogonality indicator matrix,  $\{h\}_{j+1}$ .

### Semi-Orthogonality

As mentioned before, restoring orthogonality or reorthogonalization should be conducted when any element in the indicator matrix grows bigger than the projected error limit.

$$|\{q\}_i^T [M] \{q\}_i| > \varepsilon \quad ; i \neq j \quad (60)$$

For faster calculation, applying semi-orthogonality criteria instead of full orthogonality is preferred.

$$|\{q\}_i^T [M] \{q\}_i| > \sqrt{\varepsilon} \quad ; i \neq j \quad (61)$$

Even though precision of semi-orthogonality is not as good as the full one, the solution is still acceptable in any kind of calculation. We do so by using higher limitation of error.

CHAPTER 4  
INPUT AND OUTPUT AND STRUCTURE OF  
LANCZOS ALGORITHM PROGRAM

Input and Output of the Program Package

Lanczos algorithm package in this thesis is a modified version of the listing in the textbook of Dr. Thomas J.R. Hughes, "The Finite Element Method," Prentice-Hall, 1987.

Input of the Program

In order to perform Lanczos Algorithm and solve eigenproblem by this program, data below are required.

1. LANMAX the maximum number of Lanczos steps allowed to perform
2. MAXPRS the maximum number of eigenpairs user allowed to be calculated.
3. N the dimension of system size
4. K the stiffness matrix
5. M the mass matrix

When the program package is initiated, it will prompt to receive the data of N, LANMAX and MAXPRS at the main program "Lanczos." Then, received data will be passed into subroutine LANCZOSCON.

This subroutine will prompt again to receive the data of [K] and [M] matrices. User will input all mass and stiffness matrices right after this point.

Providing a starting vector for Lanczos step is optional and can be input at this subroutine. If starting vector is not received, the subroutine will randomly allocate it. After all, Lanczos loop with reorthogonalization is performed. And, the results are printed.

### Output of the Program

Outputs of the program are eigenvalues and eigenvectors of the user-provided mechanical system. Number of eigenpairs normally is equal to MAXPRS. However, the number of eigenpairs can be less because the outputs are only eigenvalues that are proved to be converged by the program before the maximum steps of Lanczos loop has been reached, LANMAX. When almost converged Ritz values meet convergent tolerance, those Ritz values will be listed in the converged eigenvalue matrix.

Nevertheless, at the end of the last step of Lanczos algorithm, LANMAX<sup>th</sup>, The most updated Ritz values will be listed in the eigenvalue array if the number of converged eigenvalues has not reached the number of the user-wanted eigenpairs, LANMAX. At the end of program, eigenvectors corresponding to the computed eigenvalues will be calculated and shown in the output page. See appendix D for example.

### Structure of the Lanczos Algorithm Program

In his thesis, Lanczos Algorithm programming package was modified from Dr. Thomas's work, [3] on 557-560. This Lanczos algorithm-programming package is

constructed on Fortran 90 compiler. Microsoft FORTRAN Developer Studio was used. The package consists of 1 main program, 5 functions and 24 subroutines.

Lanczos is the main program and called first to initiate all the computation. Function of each FORTRAN function and subroutines will be elaborated later in this chapter. The list of main program, subroutines and functions that are used in the package are listed below.

Main program

LANCZOS

Subroutines

LANCZOSCON  
LANSIM  
STPONE  
OPK  
NEWCOR  
SUBJ  
STORE  
DAXPY

LANDRV  
PURGE  
RITVEC  
OPM  
ORTBND  
DEFLAT  
RAN  
DCOPY

LANSEL  
ANALZT  
K-INPUT  
GIVENS  
QLBOT  
MOVE1  
CSCAL  
ZERO

Functions

ENOUGH  
DDOT

NUMLES  
IDAMAX

GETEPS

On the next page, the Lanczos algorithm is shown. Some subroutines and functions will be called more than once depending on nature of system. The flowchart of the structure of every coding in package and package's structure itself can be seen in appendix A. The codes of each coding elements in Lanczos algorithm package can be seen in appendix B. The meaning of variables used can be seen in appendix C.



## Lanczos Algorithm

Set Initial variable and vectors;  $\{r\}_0$  is the given eigenvector

1.  $\{q\}_0 = 0$
2.  $\beta_1 = (\{r\}_0^T [M] \{r\}_0)^{1/2}$
3.  $\{q\}_1 = \frac{\{r\}_0}{\beta_1}$
4.  $\{p\}_1 = [M] \{q\}_1$

For  $j = 1, 2, 3, \dots$

1.  $\{\bar{r}\}_j = \{K\}_\sigma^{-1} \{p\}_j$
2.  $\{\bar{r}\}_j = \{\bar{r}\}_j - \{q\}_{j-1} \beta_j$
3.  $\alpha_j = \{q\}_j^T [M] \{\bar{r}\}_j = \{p\}_j^T \{r\}_j$
4.  $\{r\}_j = \{\bar{r}\}_j - \{q\}_j \alpha_j$
5.  $\{p\}_j = [M] \{r\}_j$
6.  $\beta_{j+1} = (\{r\}_j^T [M] \{r\}_j)^{1/2} = (\{\bar{p}\}_j^T \{r\}_j)^{1/2}$
7. If MAXPRS has been reached, terminate Lanczos loop
8.  $\{q\}_{j+1} = \frac{1}{\beta_{j+1}} r_j$
9.  $\{p\}_{j+1} = \frac{1}{\beta_{j+1}} \{\bar{p}\}_j$

FIGURE 1. Lanczos Algorithm

## Main Program

Program LANCZOS. The variables that have not been set their values cannot be used to set dimensions of arrays in the same program or subroutine. Therefore, separating main program into 2 parts is needed. First part will receive parameters of the system's size, N, maximum eigenpairs needed by user, MAXPRS, and maximum allowed steps of Lanczos algorithm to perform, LANMAX. Then, LANCZOS passes down these parameters to its continuing subroutine, LANCZOSCON, which will allocate the parameter to appropriate arrays.

N is dimension of system matrix.

LANMAX is number of maximum Lanczos steps that program will compute. The Lanczos steps will be stopped if number of step is over this value.

MAXPRS is number of maximum eigenpairs needed by user. The Lanczos steps will be stopped when this number is reached.

## Subroutines

Subroutine LANCZOSCON. This subroutine is continuation of the main program, LANCZOS. It allocates dimensions of W, Y EIG and IW array with suitable LANMAX, MAXPRS and N. After this step, it prompts to receive the data of ENDL and ENDR; call LANDRV to compute eigenvalues and their corresponding eigenvectors; and finally prints the computed eigenpairs.

Subroutine LANDRV. This subroutine is to initialize Lanczos algorithm. It first checks error of received data earlier. These errors are caused by inputting wrong value by nature or impossible case of those values. For example, ENDL is the left end

of eigenvalues' range. It cannot be more than ENDR, which is right end of the range. The list of errors concerned is shown at the end of this subroutine's description.

If any of these errors occur, the computation will be terminated and IERR, error indicator, will be assigned a value. Each value of IERR has its meaning. Different value means different kind of errors. The meaning of IERR is described in the table below as well. This way, small errors that might confuse us when there are calculation errors or miscalculated results is prevented to occur. After all data are checked, subroutine will determine computer precision, unit roundoff error and semi-orthogonality tolerance, by calling GETEP. Then, those values will be put into EPS, EPS1 and REPS.

In Finite Element Method (FEM), Lanczos Algorithm program deals with a very large eigensystem. Million-by-million matrix size is not unusual at all. Therefore, avoiding excessive data transferring between subroutines and within subroutines themselves due to the enormous data being computed is concerned. Therefore, next step of this subroutine is to prepare 1 big matrix that contains all vectors and arrays. The subroutine sets pointers of working vectors and arrays in the main array, W. These pointers indicate the positions of first element of each vectors and arrays. By this strategy, instead of moving all the data in computing vector or array, the program work in the same data spaces lead to by the pointers.

The next step in LANDRV is checking the starting vector. It will check weather the starting vector is provided by user or not. If it has already been provided, the step will be skipped. Otherwise, LANDRV will randomly assign starting vector

using RAN subroutine. Up to this point, all the data have been already input and the program is ready to perform Lanczos Algorithm.

The 1<sup>st</sup> step of Lanczos Algorithm is performed by STPONE. This subroutine calculates first step and prepares Lanczos vector  $q_1$ ,  $\alpha_1$  and  $\beta_2$ . The rest of the Lanczos Algorithm will be performed by LANSEL. Once all the requirements are met of program terminated for some reason, LANDRV will return to subroutine LANCZOSCON.

Input N, LANMAX, MAXPRS, ENDL, ENDR, NW, W and IW

Output EIG, Y, IERR and NEIG

TABLE 1. The Definition of the Value of the Error Flag

Bit value	Represent
1	N < 0
2	LANMAX < 0
3	ENDR < ENDL
4	MAXPRS < 0
5	MAXPRS > LANMAX
6	LANMAX > N
7	NW is too small
-1	J > LANMAX

Note: IERR will be set to negative 1 if maximum Lanczos steps have been reached without any converged eigenvalue.

Subroutine LANSEL. This subroutine performs the rest of Lanczos Algorithm and applies reorthogonalization when error bounds grow higher than set tolerance.

Subroutine PURGE skips itself internally when  $j$  is less than 3, first and second step. Subroutine ANALZT does so as well. Here is the structure of the subroutine LANSEL.

First of all, subroutine LANSEL calls subroutine LANSIM to take a step of Lanczos algorithm. As a result, the new  $\alpha$ ,  $\beta$  and Lanczos vector are identified. After new elements of tridiagonal matrix have been found, ORTBND is called to update the orthogonality errors. This error indicator vector is next to be checked by subroutine PURGE. If any element of this indicator vector is out of limitation error bound, it shows that the algorithm process is losing orthogonality between working vectors.

Inputs: N, LANMAX, MAXPRS, NS, ENDL and ENDR

Outputs: EIG, Y, NEIG and IERR

Internal working variables:

R, NQ, ALF, BET, ALPG, BET2, TAU, OLDTAU, RHO, ETA,  
OLDETA, INFO, S and WORK.

Subroutine LANSIM. This subroutine performs a new Lanczos step. All vectors and arrays are working inside the global array, R. The structure of this subroutine is exactly the same as the structure of Lanczos Algorithm. It will calculate new element of ALF and BET and update running Lanczos vectors. It is called by subroutine LANSEL.

Subroutine LANSIM actually updates already existing the working arrays (set of elements in global array, R) in this subroutine. In other words, the inputs also act as outputs.

Input: NQ, N, J.

Input/Output:

R, ALF, BET, ALPH, BET2, RNM, RNM2

Subroutine PURGE. The main purpose of this subroutine is to monitor loss of orthogonality of the computation. If any occurs, the subroutine will perform appropriate reorthogonalization. It will examine the array ETA, which contains orthogonality bounds,  $h_{j+1}$ . If the largest value of array ETA is less than the semi-orthogonality tolerance,  $\sqrt{\epsilon}$  or REPS, there is no unacceptable loss of semi-orthogonality and the subroutine is skipped.

Oppositely, if the loss of orthogonality is detected, array TAU, which holds  $\tau$  recurrence, will be checked. If  $i^{\text{th}}$  element of array TAU is greater than REPS, it indicates that loss of semi-orthogonality against Ritz vector with index  $i$  has occurred. Then, the corresponding eigenvector of tridiagonal matrix,  $[T]$ , is computed by subroutine GIVEN. And subroutine PURGE orthogonalizes array ETA and OLDETA against this eigenvector.

Next step, subroutine PURGE rechecks the updated array ETA. If there is still loss of semi-orthogonality detected, it will perform full orthogonalization. Subroutine RITVEC is called to perform full orthogonalization, and the value of elements of array TAU, OLDTAU, ETA and OLDETA will be reset to  $\text{EPS1}, \sqrt{n\epsilon}$ . However, if array ETA does not indicate loss of semi-orthogonality in the second check, only loss of semi-orthogonality against Ritz vector will be taken care of. Subroutine PURGE orthogonalizes  $\{q\}_j$  and  $\{r\}_j$  in array Q and R against Ritz vectors in column of Y with

index i. The elements of array TAU and OLDTAU will be reset to EPS1. At the end,  $[M]\{q\}_j$  and  $[M]\{q\}_r$  in QA and RA are recalculated if there is any change in  $\{r\}_j$  and  $\{q\}_j$ .

**Input/Output:**

Almost all the working arrays are both input and output in this subroutine R, Q, RA, QA, T, Y, ALF, BET, S, EIG, ETA, ELDETA, TAU, OLDTAU, WORK, INFO, N, J, NEIG and IBUF.

**Subroutine ANALZT.** This subroutine provides the smallest interval that contains eigenvalues of tridiagonal matrix,  $[T]_j$ . Subroutine only computes eigenvalues of 2 x 2 matrix containing  $\alpha_1$  and  $\alpha_2$  and 2 of  $\beta_2$  in 2<sup>nd</sup> step of Lanczos loop, j = 2. The simple equation to solve eigenvalues of 2 x 2 matrix below is used at this point.

$$\lambda = \frac{\alpha_1 + \alpha_2 \pm \sqrt{4 * \beta^2(2) + (\alpha_1 - \alpha_2)^2}}{2}$$

From the 3<sup>rd</sup> step on, the subroutine will perform 2 phases. Firstly, it updates the data structure, THET and BJ, which contain eigenvalues of  $[T]_{j-1}$  and their residual bounds, respectively. Secondly, it checks convergence of almost converged eigenvalues stored in THET. If any of them has been converged, it will be removed from THET and put into array EIG. In this phase, new element of THET and corresponding BJ will be appended. This subroutine is 1 of the main steps of LANSSEL loop.

**Input:** J, ALF, BET2, EIG, TAU, OLDTAU, RHO, INFO, THET

Output: EIG

Internal variable:

NDST, THET, BJ, NBD, SPREAD, EPS, IP, INC, IS, START,  
PROBE and INDXOK

Subroutine STPONE. This subroutine performs the 1<sup>st</sup> step of Lanczos iteration. It computes  $\alpha_1$ ,  $\beta_1$ ,  $q_1$  and  $p_1$ . From this subroutine on, the global array, W, will be referred as global array, R.

First 6 blocks of elements of R store 6 working vectors. The positions of starting element of each vector are indicated pointers, NQ. Below is the table of structure of first 6 blocks of array R.

The structure of STPONE can be referred from figure 1 when  $j = 1$ .

TABLE. 2 Pointers of Elements  
in Global Array

Pointer	1	NQ1	NQ2	NQ3	NQ4	NQ5
Vector	$r_j$	$q_j$	$q_{j-1}$	$Mr_j$	$Mq_j$	T

Input/Output:

This subroutine mostly updates new value of  $\alpha_1$  and  $\beta_1$ , so most of working arrays are both input and output.

N, ALF, BET, ALPH, BET2, R and NQ

Subroutine RITVEC. This subroutine has 2 main functions. First, it is computing the Ritz vector corresponding to a converged Ritz value or eigenvalue. Second, it is performing a full reorthogonalization.



Computing the Ritz vector is performed every time this subroutine is called. Full reorthogonalization is performed when there is loss of semi-orthogonality against previous Lanczos vector, indicating by logical value EVONLY, which is computed by subroutine PURGE. If logical EVONLY is true, only eigenvector is computed. Otherwise, both functions are performed.

In order to compute the Ritz vector, subroutine RITVEC calls subroutine GIVENS to perform the computation for the wanted eigenvector. Once the vector is obtained, subroutine STORE is called to obtain Lanczos vectors from the secondary storage. Now, the Ritz vector can be calculated by

$$\{Y\}^{(m)} = [Q]_m \{S\}_i^{(m)}, \quad i = 1, \dots, m$$

When logical EVONLY is false, it indicates that the full reorthogonalization is needed. The 2 current Lanczos vectors,  $\{q\}_j$  and  $\{r\}_j$ , in Q and R will be orthogonalized against the previous Lanczos vectors, which have been just called by subroutine STORE.

**Input/ Output:**

Almost all the working arrays are both input and output in this subroutine.

R, Q, RA, QA, T, Y, ALF, BET, EIG, ETA, S, INFO, N, J, NEIG IBUF and EVONLY.

Subroutine K-EINPUT. This subroutine acquires the values of the elements in the system mass and stiffness matrices. It receives data in 2 ways. First, it can receive

data one by one from typing. Second, it can receive data from data files that have already been setup before.

Subroutine OPK. This subroutine solves an equation below. This subroutine is used mainly in subroutine STPONE and LANSIM to perform the M-orthogonization.

$$\{Y\} = [K]_{\sigma}^{-1} \{X\}$$

$\{X\}$  and  $\{Y\}$  are vectors

$[K]_{\sigma}$  is the shifted system stiffness matrix

Subroutine OPM. This subroutine solves an equation below. This subroutine is also used mainly in subroutine STPONE and LANSIM to perform the M-Orthogonization. It is called at almost the same step as OPK.

$$\{Y\} = [M] \{X\}$$

$\{Y\}$  and  $\{X\}$  are vectors

M is the system mass matrix

Subroutine GIVENS. This subroutine is called by RITVEC to compute eigenvector of tridiagonal matrix, T, corresponding to an eigenvalue in THET. Because this is a tridiagonal matrix, computation of eigenvector first starts by assuming a value of the bottom element of the vector. The rest of the elements can be computed by a series of simple equation going backward. For example,

$$\begin{bmatrix} 1 & 5 & 0 & 0 \\ 5 & 2 & 6 & 0 \\ 0 & 6 & 3 & 7 \\ 0 & 0 & 7 & 4 \end{bmatrix} \begin{Bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

Considering, 4<sup>th</sup> column,

$$7 \times v_3 + 4v_4 = 0$$

$$v_3 = \frac{-4}{7}$$

Then, considering 3<sup>rd</sup> column,

$$6 * v_2 + 3 * v_3 + 7v_4 = 0$$

$$6 * v_2 - \frac{12}{7} + 7v_4 = 0$$

by assuming that  $v_4 = 1$ , it becomes

$$v_2 = -\frac{37}{42}$$

This phase is terminated when k<sup>th</sup> element of S is found. Then, the computed elements are scaled to make S(k) equal to 1. After that, the same series of calculation are performed, but starting from the 1<sup>st</sup> element of the vector and going foreword until reaching the k<sup>th</sup> element.

After all, eigenvector S, is normalized and the residual and Rayleigh correction to THETA are computed and put in RES and COR respectively.

Input: K, J, ALF, BET, THET and EPS

Output: S, RES and COR

**Subroutine NEWCOR.** This subroutine's function is to find an eigenvalues of a matrix in a given interval. Two theories are used in this subroutine, Bisection and Newton's method. The Elements of THET with index of INDEX contains the eigenvalues of  $[T]_{j-1}$ . This value is also 1 end of interval. The other end is held by ZETA. ZETA is also the starting value of Newton's method applied the next.

If the interval is considerably broad, subroutine NEWCOR will perform a few steps of Bisection to decrease the interval. The counts of eigenvalues in the interval is computed by subroutine NUMLES. Detail of Bisection and Newton's method can be found in [4] on pages 81-83 and 109-113, respectively.

Once the better approximation of ZETA has been obtained, Newton's method will be performed to refine it for the best approximated ZETA. The convergence of Newton's method is then checked by deflation the eigenvalues exterior to ZETA implicitly.

Input: ALF, BET2, SPREAD, INC, INDX, J and NDST

Input/Output:

ZETA, THET and BJ

Subroutine ORTBND. This subroutine updates the error indication vector of the on going computation, the orthogonality bounds,  $\{h\}_j$  and  $\{\tau\}$  recurrence.

These 2 kinds of indicator vectors have not been mentioned before. In an easy explanation, the program monitors loss of orthogonality at 2 places, loss of orthogonality against previous Lanczos vector and current Ritz vector. The orthogonality bounds are used to monitor loss of orthogonality against previous Lanczos vector.

The reorthogonization against previous Lanczos vector is the full version of reorthogonization. On the other hand, reorthogonization against Ritz value is a shorter process and deals with fewer amounts of data. Therefore, avoiding full

reorthogonalization all the time and conducting orthogonalization against Ritz vector is the way to optimize the computation rate and cost.

Subroutine LANSEL is the only one to call subroutine ORTBND. And, the output of subroutine ORTBND will be used by subroutine PURGE. These outputs will be compared to the semi-orthogonality tolerance to indicate whether there has been loss of orthogonality or not.

Input: ALF, BET, J, EPS1, ETA, OLDETA, TAU, OLDTAU, EIG, INFO, RNM, NEIG and N.

Output: ETA, OLDETA, TAU and OLDTAU.

Subroutine QLBOT. This subroutine computes the last element of the eigenvector of the tridiagonal matrix,  $[T]_j$ , that is corresponding to the eigenvalue stored in THET. It performs QL factorization. The product of sines of the rotation angles is the value of the bottom element. The value of the bottom element is returned as BOT.

Input: ALF, BET2, THET, J

Output: BOT

Subroutine SUBTJ. In some cases, 2 or more eigenvalues of a tridiagonal matrix can be so close that the computer cannot distinguish the difference. This creates a difficulty to compute eigenvectors.

This subroutine solves the problem by working with a sub-matrix  $[T]_{l,m}$  instead of the global,  $[T]_j$ . It calculates an estimate of indices l and m that defines a sub-

matrix for which  $EIG(I)$  is simple.  $INFO(I) = N \times k + j$  stores the 2 indices,  $j$  and  $k$ . The value of  $j$  is the step where  $EIG(I)$  was computed. The value of  $k$  is the index of the biggest element of the eigenvector. If  $INFO(I)$  is less than 0, Ritz vector in  $Y(I)$  has been converged.

Input:  $EIG, INFO, TOL, I, N$  and  $NEIG$

Output:  $K$  the index of the right hand side of subroutine  $GIVENS$

$L$  the index of the last element of the sub-matrix

$M$  the index of the first element of the sub-matrix

Subroutine DEFLAT. This subroutine perform deflation of tridiagonal matrix,  $[T]_j$ , with a converged eigenvalue in  $THET$ . The QR algorithm is used to perform the process. Detail of deflation can be seen in [4] on pages 47-50 and [7] pages 421-424.

Input:  $ALF, BET2, THET, J$

Output:  $ALF, BET2$

Subroutine MOVE1. This subroutine is an important management tool used by subroutine  $ANALZT$ . It inserts a given value,  $T$ , into the  $k^{\text{th}}$  element of array  $Y$ . The elements already existing will be moved by 1 space upward or downward depending on the sign of  $MINC$ .

Input:  $\{Y\}, K, L, MINC, T$

Output:  $\{Y\}$

Subroutine RAN. This subroutine is called by subroutine  $LANDRV$  to randomly select an element of starting vector for the Lanczos loop.

Subroutine CSCAL. This subroutine performs scalar multiplication to a given matrix,  $\{Y\}$ .

$$\{Y\} = a\{Y\}$$

Subroutine DAXPY. This subroutine computes the product of a scalar,  $a$ , and a given vector,  $\{X\}$ . Then, adding the result to another vector,  $\{Y\}$ . The outcome is returned as the vector  $\{Y\}$ .

$$\{Y\} = a\{X\} + \{Y\}$$

Subroutine DCOPY. This subroutine copies the value of vector,  $\{X\}$ , to another vector,  $\{Y\}$ .

$$\{Y\} = \{X\}$$

Subroutine ZERO. This subroutine simply reset every element in a vector,  $\{Y\}$ , to 0.

$$\{Y\} = 0$$

Subroutine STORE. This subroutine acts as storage. It has 2 functions commanded by the value of ISW. When ISW is equal to 1, the subroutine stores the given vector in secondary storage possessing the index of Lanczos step,  $j$ . When the ISW is equal to 2, the vector with the given index will be pulled out from the 2<sup>nd</sup> storage. These vectors are Lanczos vectors of each Lanczos step, which is indicated by the index.

Input: ISW, V, N, J

Output: V

## Functions

Logical function ENOUGH. This Logical function gives logical output indication whether all the user-required eigenvalues have been converged and collected or not. If so, the logical value will be true. The output from this logical function is used by subroutine LANSEL to terminate Lanczos loop when enough eigenvalues have been found.

Another purpose of this logical function is to determine whether an eigenvalue has been found outside the interval [ENDL, ENDR] that is limited by the user or not. If so, subroutine ENOUGH will give output to terminate the Lanczos loop as well.

Input: ENDL, ENDR and MAXPRS

Output: ENOGUH

Function NUMLES. This subroutine counts the eigenvalues exist below or above a given value, ZETA. It performs  $LDL^T$  factorization of the tridiagonal matrix,  $[T]_j$ . For more information, see [4] on pages 123-127. Only D is computed because the program does not use L. The number of eigenvalues above or below ZETA is returned as NUMLES.

Input: ALF, BET2, ZETA, N, INC, EPS

Output: NUMLES

Function GETEPS. This subroutine determines the precision of the computer being used to run the program. It will evaluate roundoff error of the computer. This value is so important in evaluation of loss of orthogonality of the computation. The



roundoff error is computed based on the criterion that roundoff error,  $\epsilon$ , is the smallest value such that  $1+\epsilon > 1$ .

Input: None

Output: EPS

Function DDOT. This function computes the Euclidean inner product of 2 vectors,  $\{X\}$  and  $\{Y\}$ . The result is returned as DOT

$$\text{DOT} = \{X\}^T \{Y\}$$

Function IDAMAX. This function finds the index of the element with the greatest absolute value in a given vector,  $\{Y\}$ .

$$i = \arg (\max_{i=1..n} |\{Y\}_i|)$$

Input:  $\{Y\}$  the given vector

Output:  $i$  the index of the greatest element in absolute value

## CHAPTER 5

### CONCLUSION

Lanczos algorithm was first developed for tridiagonalization. The algorithm is consisting of 2 important numerical procedures, Gram-Schmidt Orthogonalization and Rayleigh-Ritz Approximation.

Gram-Schmidt Orthogonalization is used for calculation of a Krylov sequence, which is a sequence that leads to the resultant eigenvector. Rayleigh-Ritz Approximation is used for constructing the tridiagonalized matrix. The purpose of Lanczos Algorithm is to calculate the 1<sup>st</sup> few eigenpairs of a huge eigenproblem in Finite Element Method (FEM).

Professor Ohtmer, my research advisor and I, strongly believe that it soon becomes a very popular and effective procedure for solving huge eigenproblems. The iteration has 2 outstanding advantages. One is its minimal calculation requirement. Because the smallest eigenvalue is converging a few iterations, not time and cost consuming. The other advantage is its simple iteration procedure. The iteration is not complicated to handle by engineers.

The Lanczos Algorithm applications are involved mainly in 3 different fields in Mechanical Engineering; Buckling Analysis, Natural Frequency Analysis and Thermal Frequency Analysis. The algorithm is well suited for the named applications

because those analysis types require only first few smallest eigenvalues, which is exactly what the Lanczos algorithm provides.

In this Thesis, many Mathematical and Matrix theories have been used besides those 2 important ones. For example, the algorithm requires the use of Orthogonality-measurement between vectors in Krylov sequence, Reorthogonalization, Newton's method, QR factorization, Bisection and Deflation.

The program package was 1<sup>st</sup> developed by Dr. Thomas J.R. Huges and written in his text book, "The Finite Element Method," Printice-Hall, 1987. And, it is modified to be able to apply the algorithm almost automatically. This program package is modified and based on Microsoft Fortran 90 platform.

Input and Output of the program are Mass and Stiffness matrices, system's size and maximum limit of Lanczos iteration and eigenvalue wanted. An initial vector of iteration can be provided by user as an option. Otherwise, the package will generate it. During the iteration, error of the computation is inspected. If an excessive error is detected, subroutines are called to perform a reorthogonization of the running vectors. This way, the accuracy of the calculation is preserved.

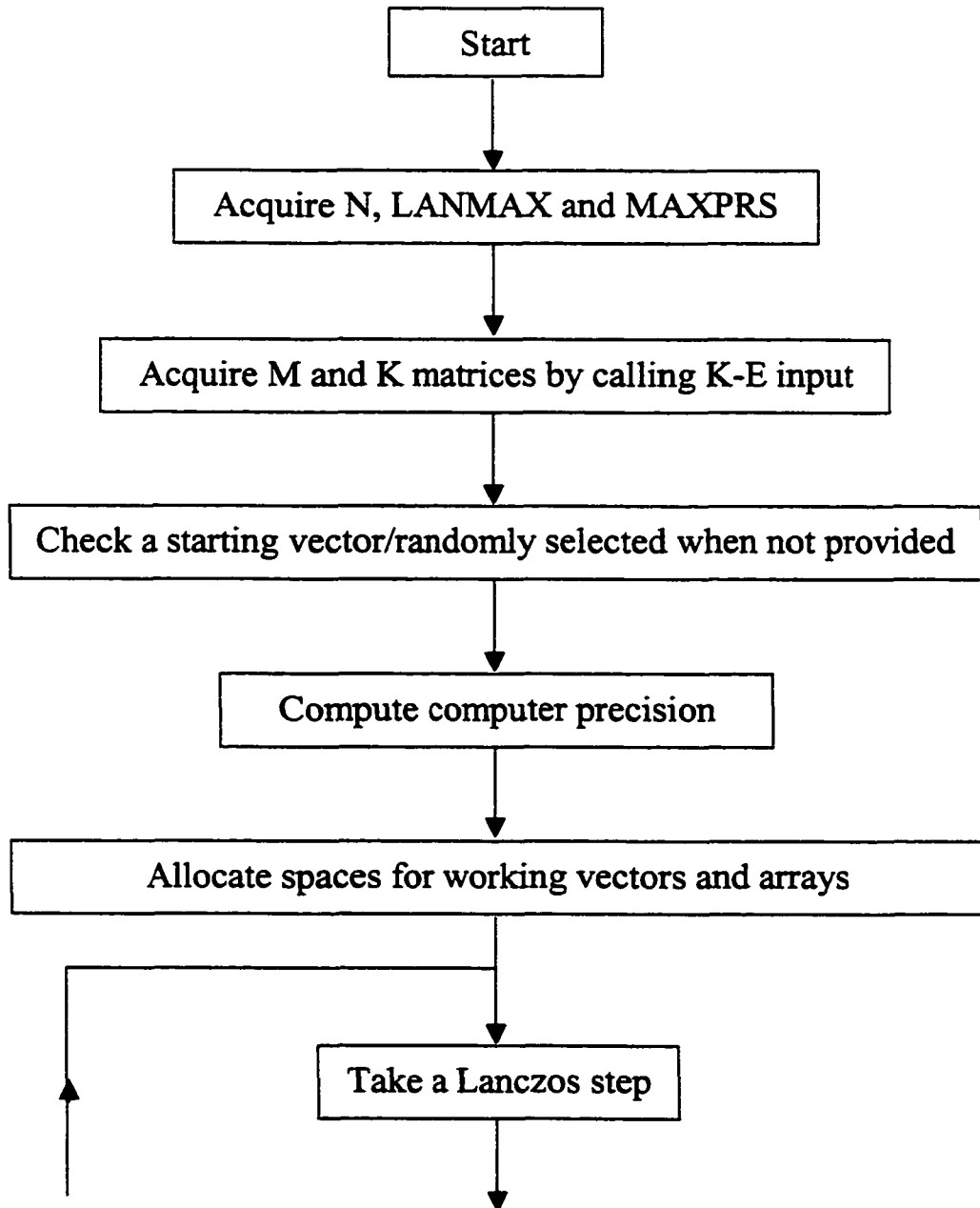
In the computation, positive-definite Mass and Stiffness matrices are required by Lanczos Algorithm to produce real eigenvalues. Fortunately, when delivered from Finite Element Method, these matrices are already positive-definite.

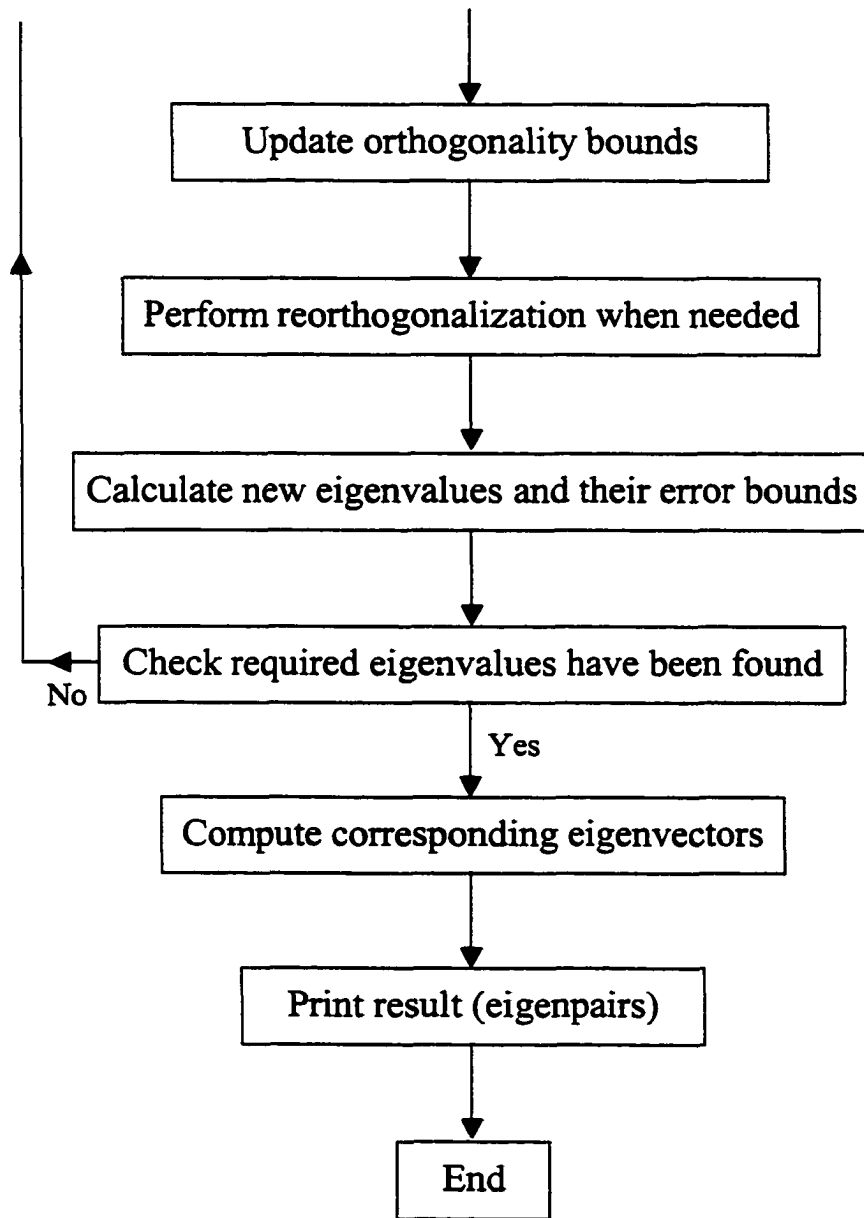
To test the program package, a large system was computed. The time consuming and accuracy were greatly acceptable.

## APPENDICES

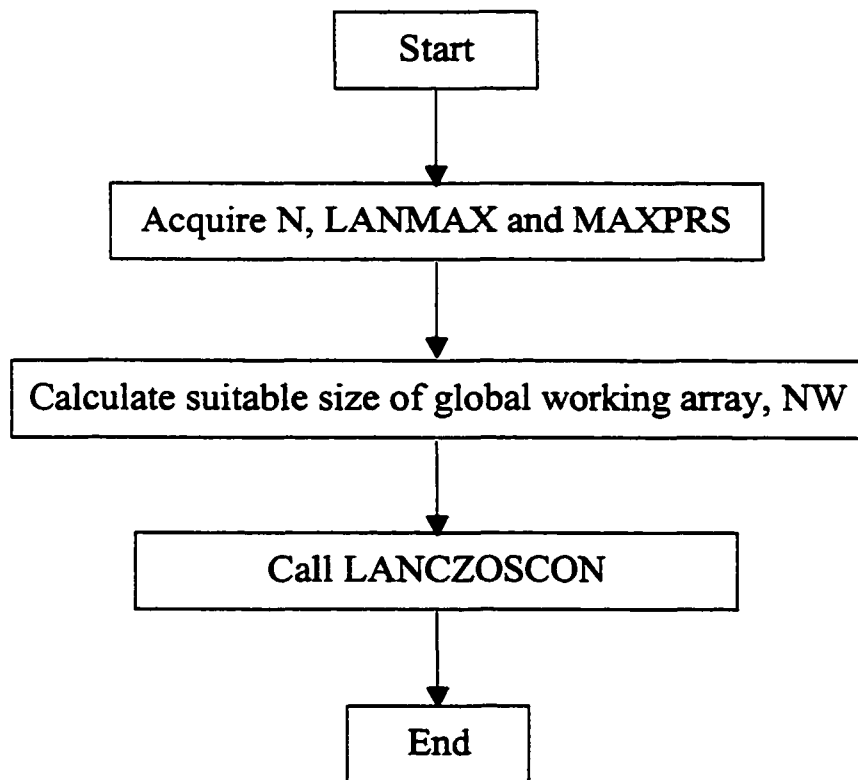
**APPENDIX A**  
**STRUCTURE FLOWCHART OF LANCZOS PACKAGE'S MAIN PROGRAM,**  
**SUBROUTINE AND FUNCTIONS**

## Lanczos package's structure



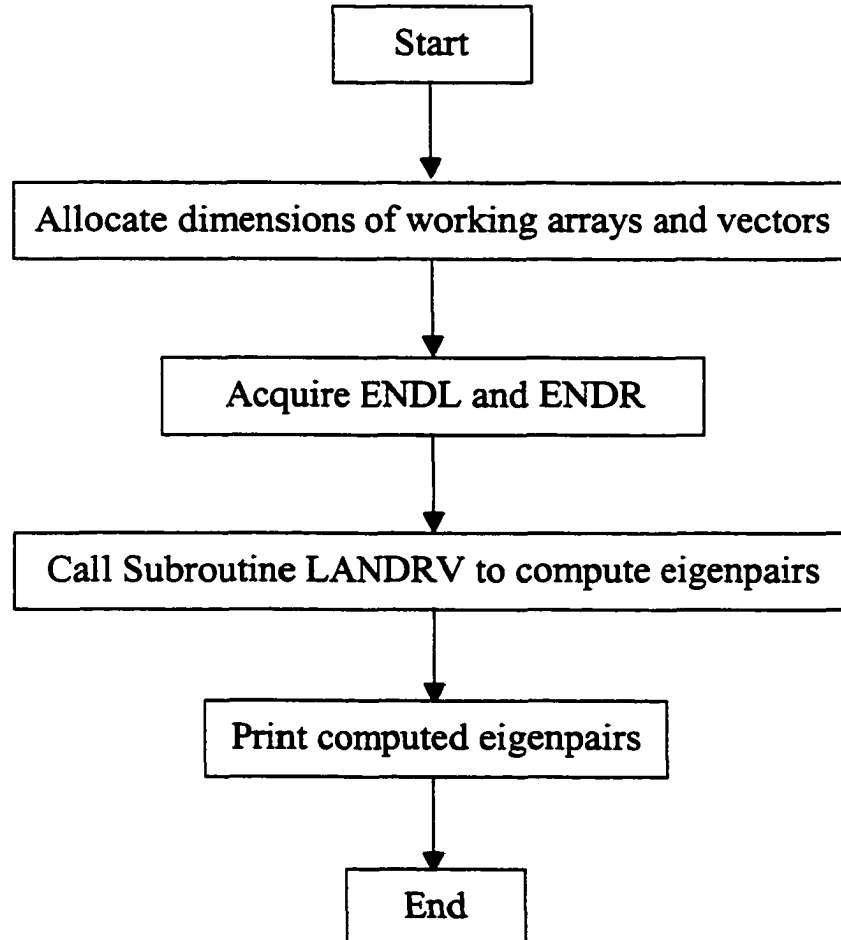


## Main Program LANCZOS

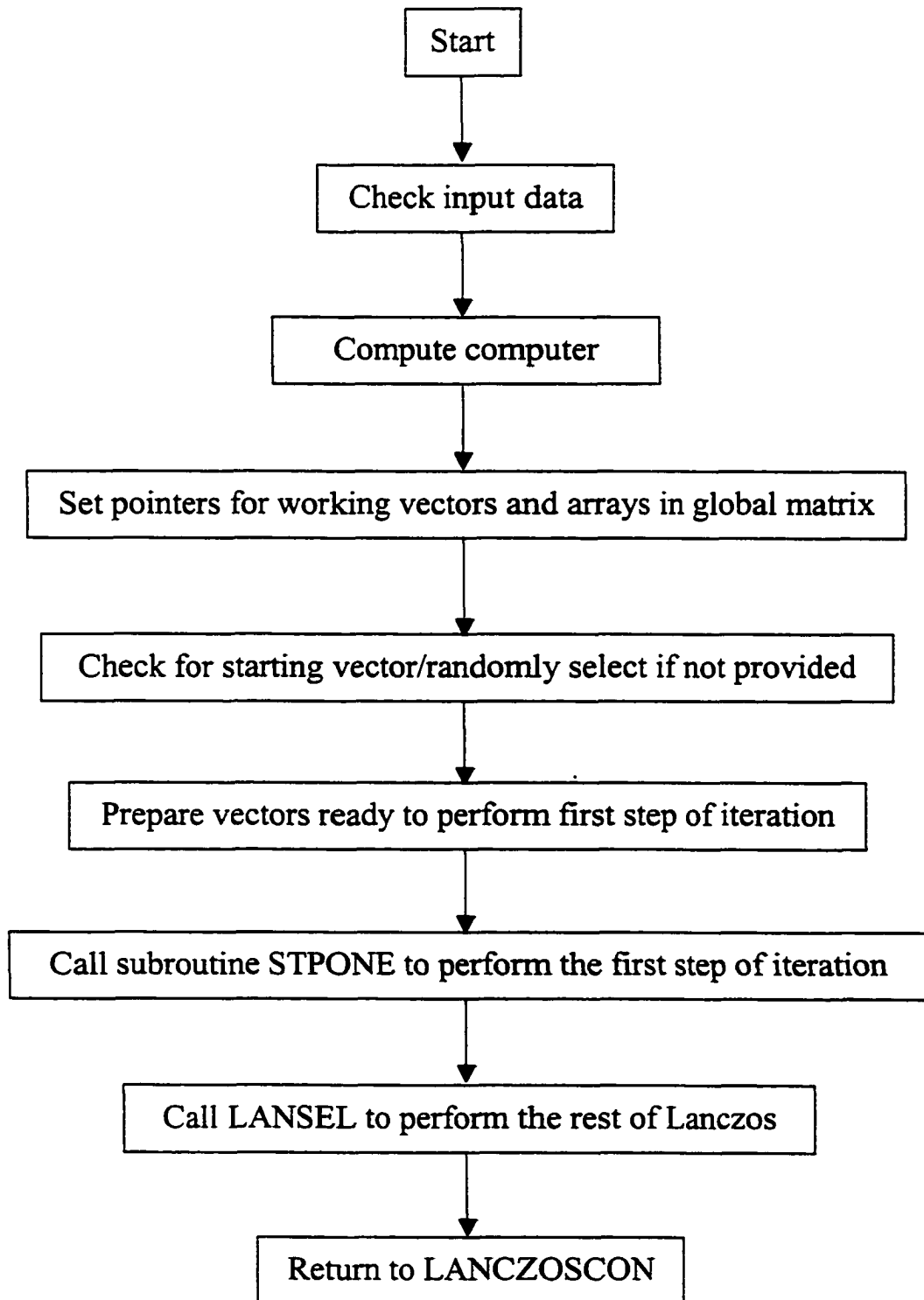




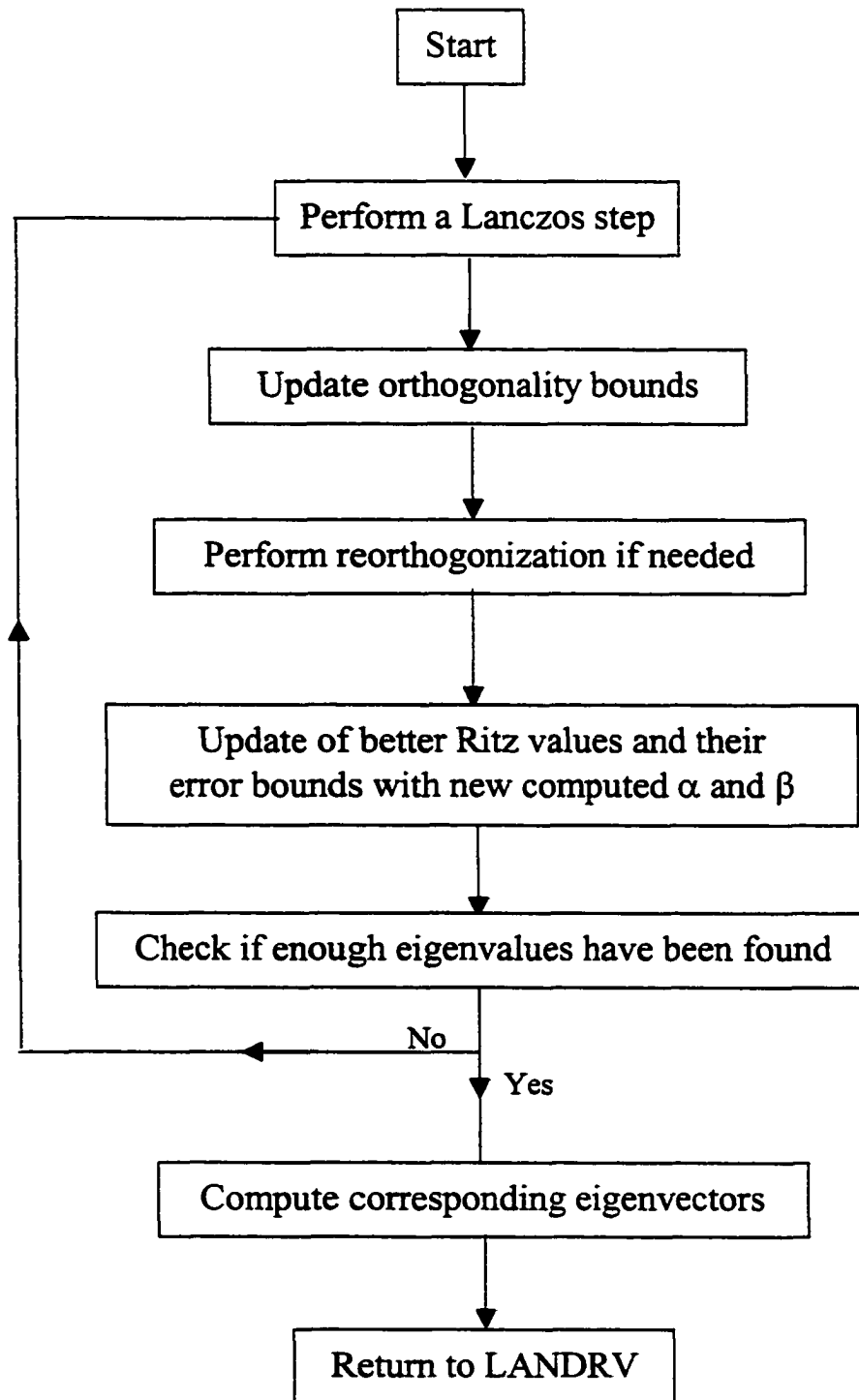
Subroutine LANCZOSCON



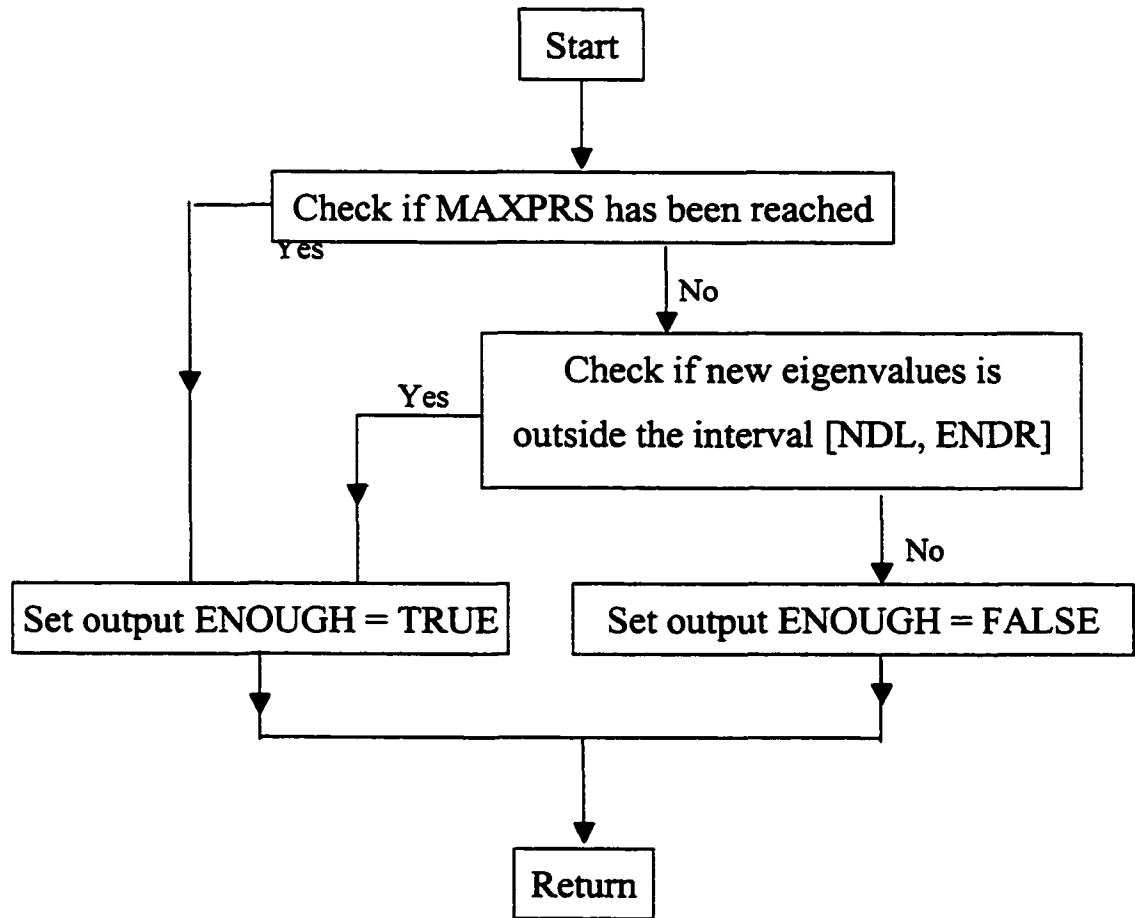
Subroutine LANDRV



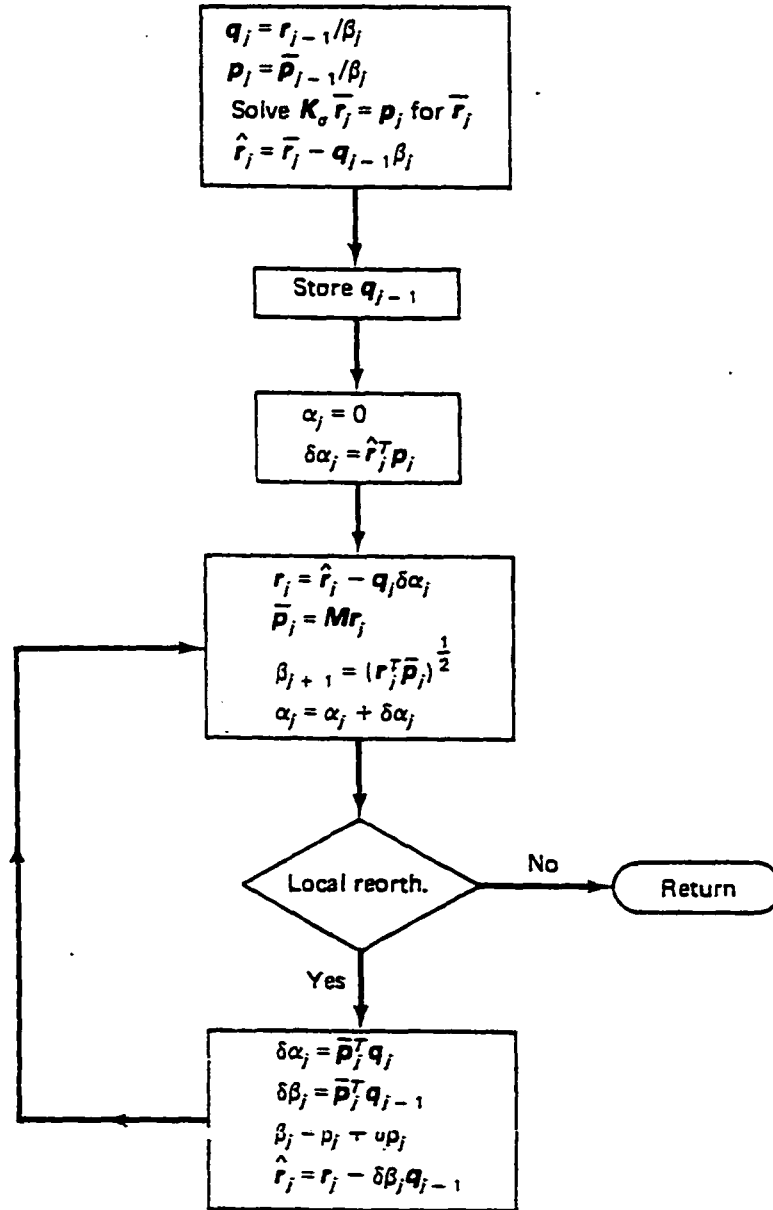
Subroutine LANSEL



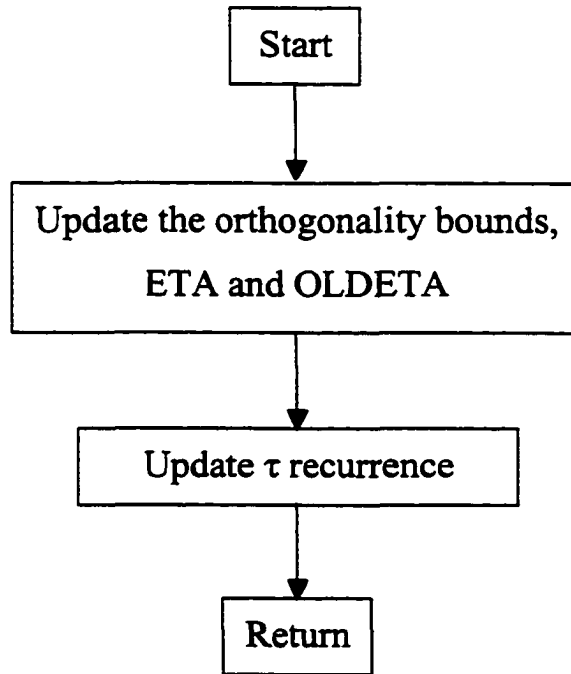
# Logical function ENOUGH



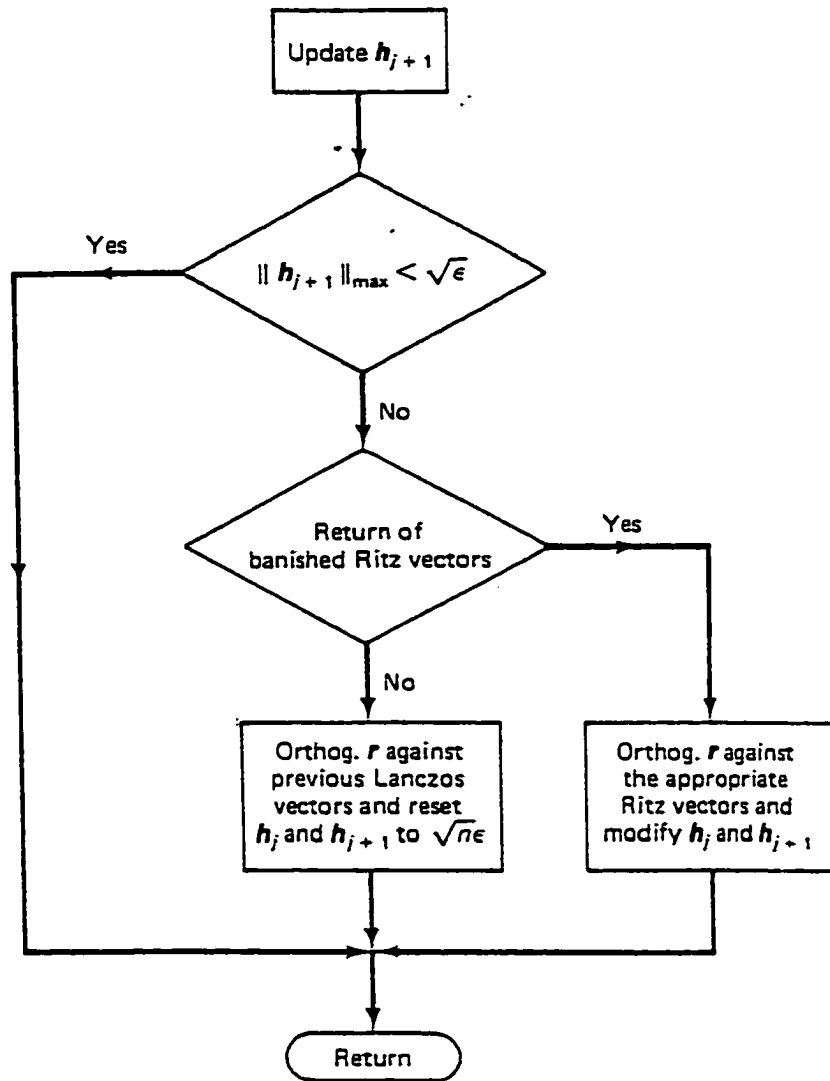
Subroutine LANSIM



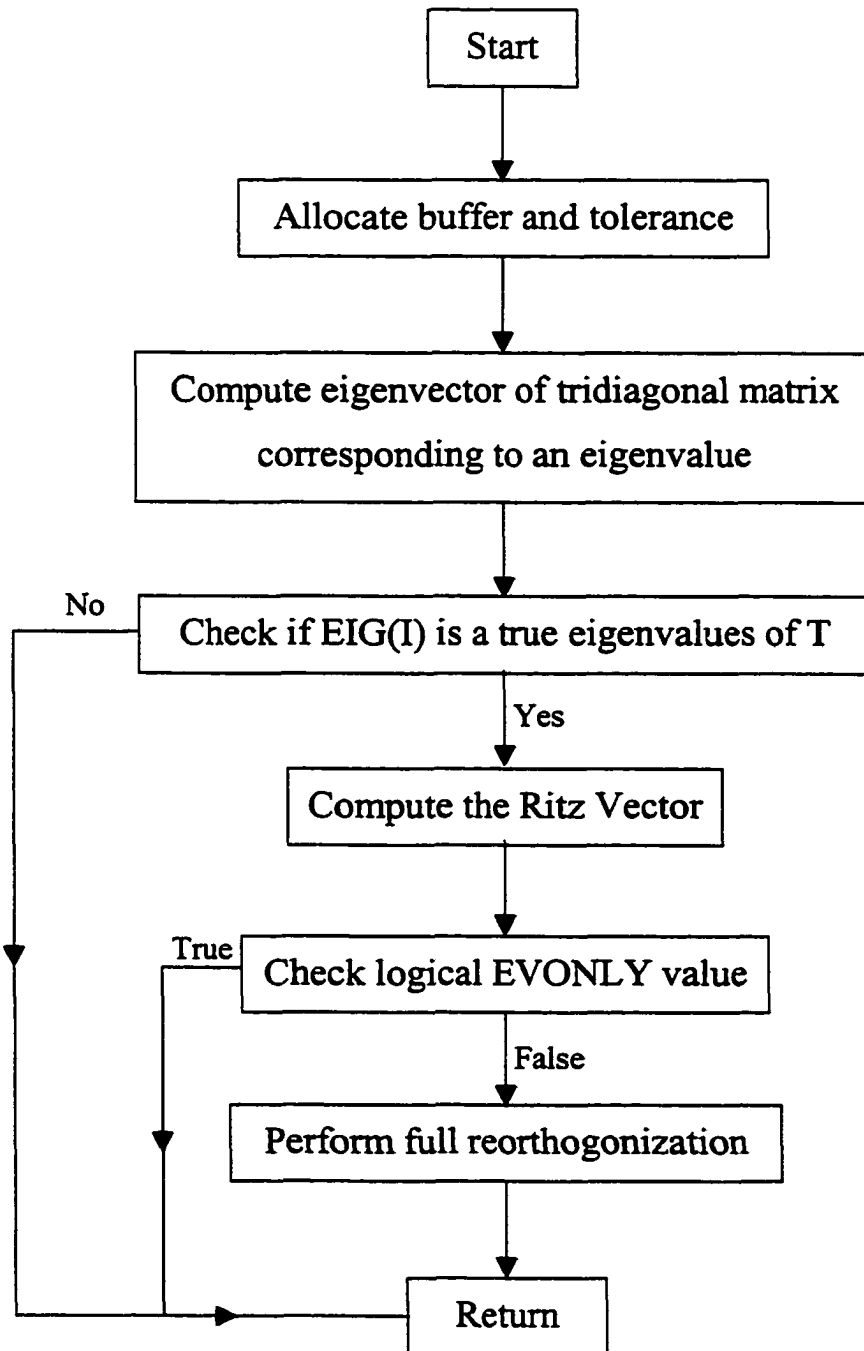
Subroutine ORTBND



# Subroutine PURGE

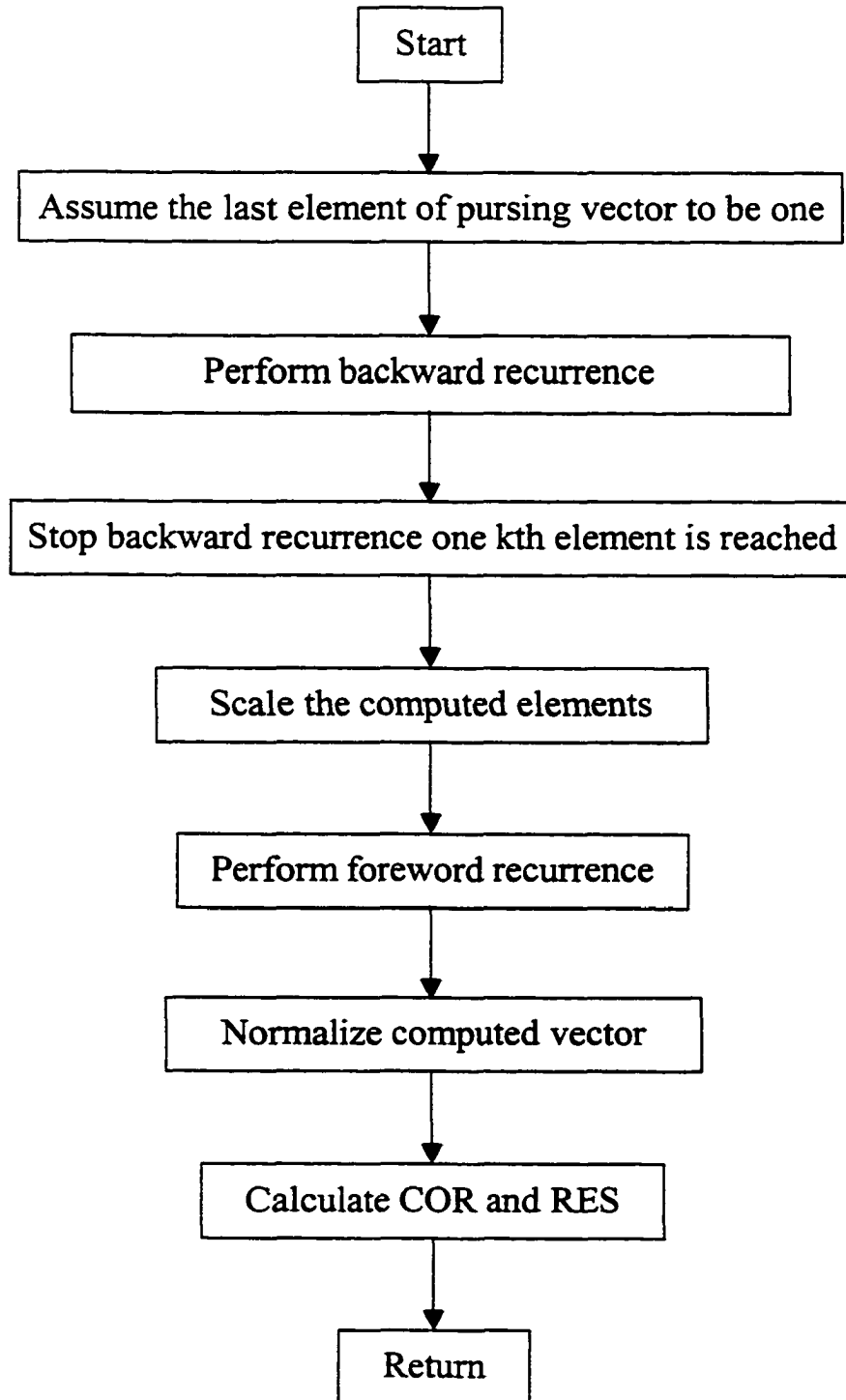


Subroutine RITVEC

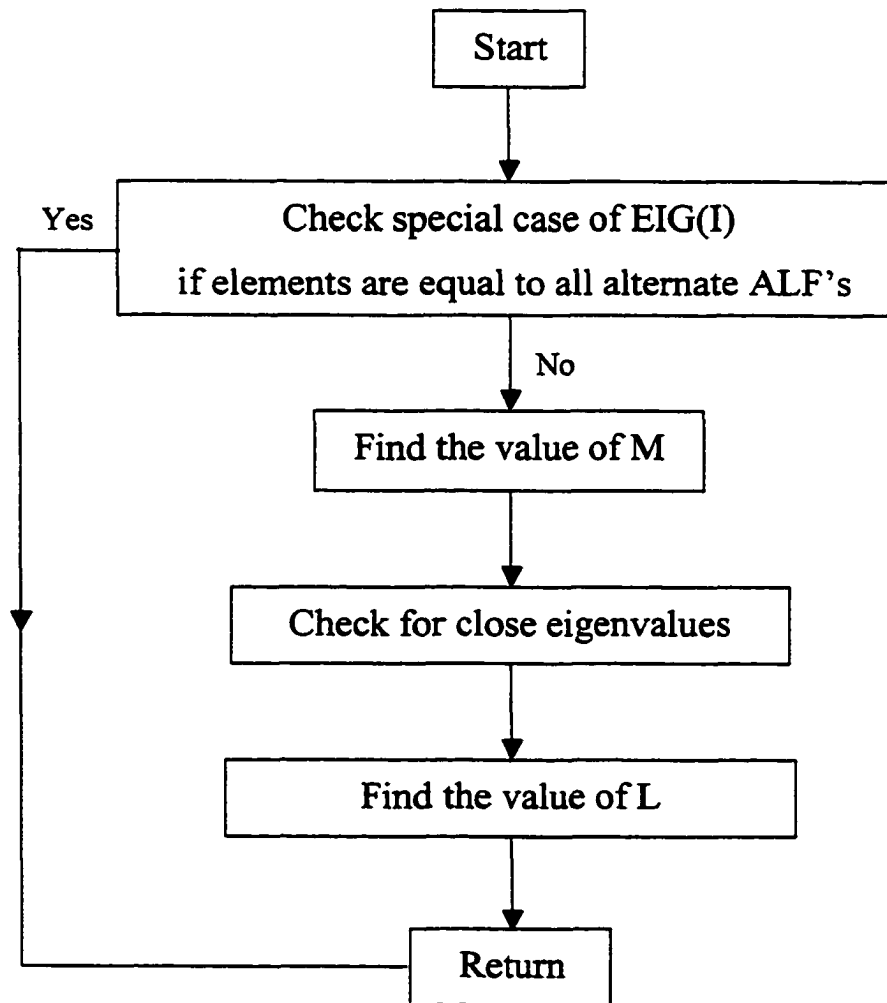




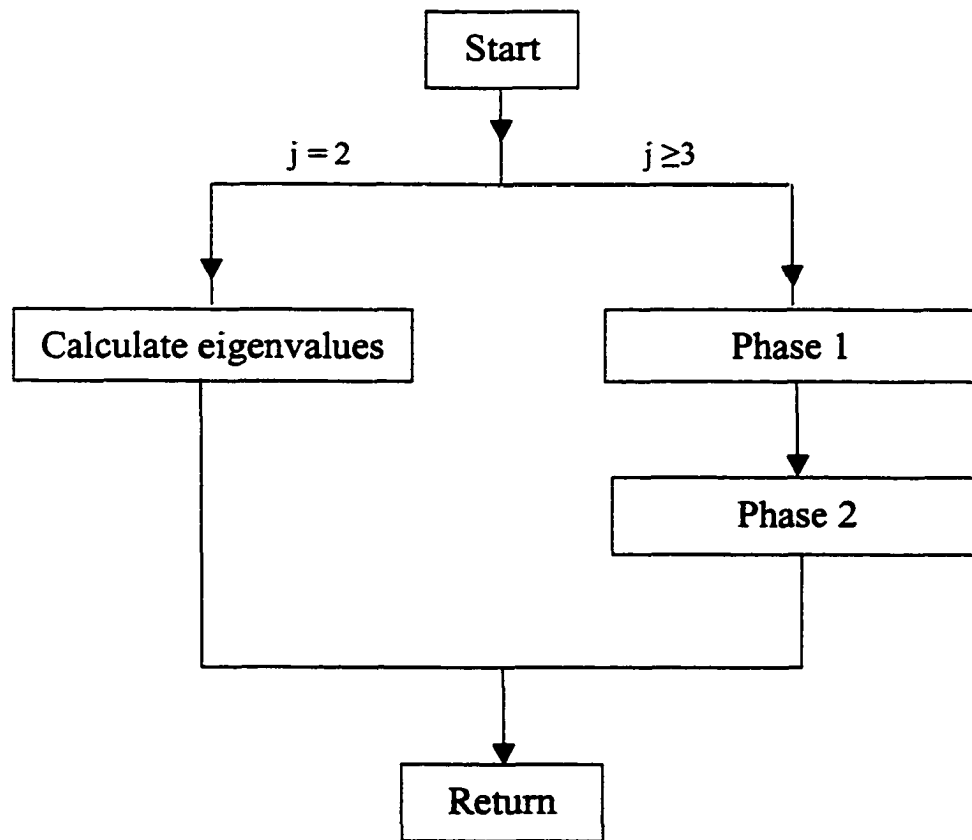
Subroutine GIVENS



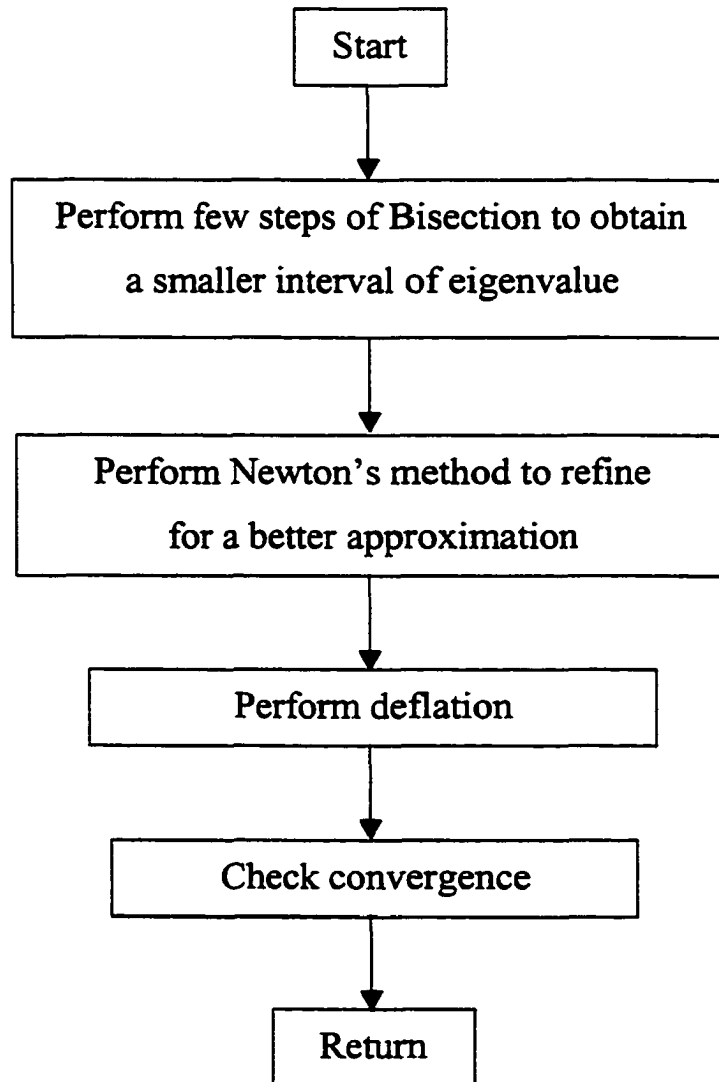
Subroutine SUBTJ



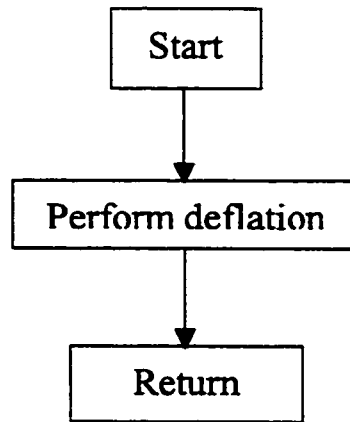
Subroutine ANALZT



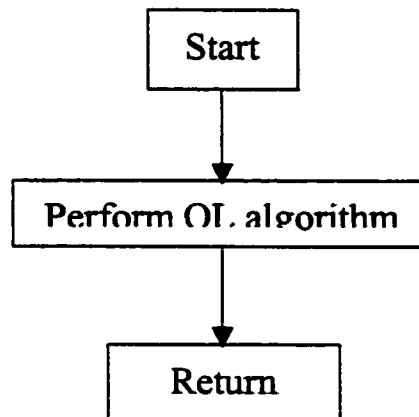
Subroutine NEWCOR



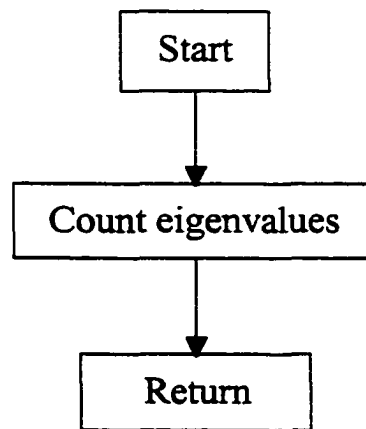
Subroutine DEFLAT



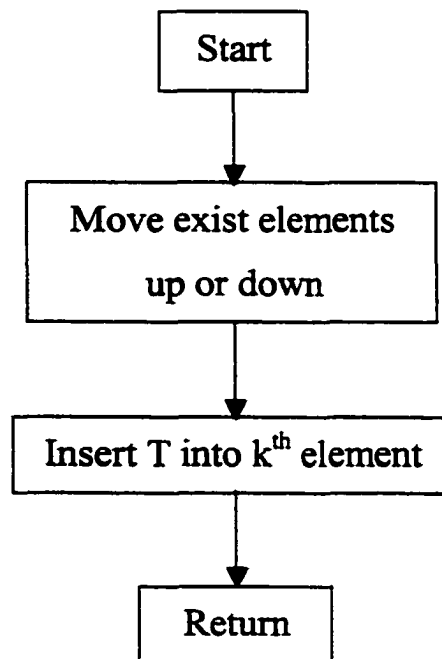
Subroutine QLBOT



Subroutine NEWCOR



Subroutine MOVE1



**APPENDIX B**  
**FORTRAN CODE USING IN**  
**LANCZOS ALGORITHM PACKAGE**

## MAIN PROGRAM

```
PROGRAM LANZCOS

COMMON /LANMAX/LANMAX
COMMON /NW/NW
COMMON /N/N
COMMON /MAXPRS/MAXPRS

PRINT *,'INPUT A DIAMENSION OF METRIX'

N=5

!READ (*,*) N

PRINT *,'INPUT THE UPPER LIMIT OF NUMBER OF LANZCOS STEPS'

LANMAX = 5

!READ (*,*) LANMAX

PRINT *,'INPUT THE UPPER LIMIT TO THE NUMBER OF WANTED EIGENPAIRS'

MAXPRS = 5

!READ (*,*) MAXPRS

NW = 6*N + 2*MAXPRS + 10*LANMAX

PRINT *,'THE LENGTH OF THE W ARRAY IS',NW
PRINT *,'

CALL LANZCOSCON

END PROGRAM
```



## SUBROUTINES

SUBROUTINE LANZCOSCON

```
REAL ENDL, ENDR
INTEGER IERR, NEIG
REAL W(NW), Y(N, MAXPRS), EIG(MAXPRS), IW(LANMAX)
COMMON /LANMAX/ LANMAX
COMMON /NW/ NW
COMMON /N/ N
COMMON /MAXPRS/ MAXPRS
COMMON /STOREFIRST/ STOREFIRST
LOGICAL STOREFIRST
COMMON /IDATA/ A, B, NEIG

DO 40 I = 1, MAXPRS

    EIG(I) = 0

40 CONTINUE

STOREFIRST = .TRUE.

PRINT *, 'INPUT THE LEFT END OF THE INTERVAL OF EIGENVALUES'
READ (*, *) ENDL

PRINT *, 'INPUT THE RIGHT END OF THE INTERVAL OF EIGENVALUES'
READ (*, *) ENDR

CALL INPUTKEM(N)

CALL LANDRV (N, LANMAX, MAXPRS, ENDL, ENDR, NW, W, IW, EIG, Y, IERR)

DO 10 I = 1, NEIG
    DO 20 J = 1, N
        PRINT *, Y(J, I)
    20 CONTINUE
    PRINT *, "
10 CONTINUE

DO 30 I = NEIG, 1, -1

    EIG(I) = 1/EIG(I)
    PRINT *, EIG(I)

30 CONTINUE
END
```

SUBROUTINE LANDRV (N, LANMAX, MAXPRS, ENDL, ENDR, NW, W, IW, EIG, Y, IERR)

! INPUTS

!N DIMENSION OF THE EIGENPROBLEM  
!LANMAX UPPER LIMIT TO THE NUMBER OF LANCZOS STEPS  
!MAXPRS UPPER LIMIT TO THE NUMBER OF WANTED EIGENPAIRS  
!ENDL LEFT END OF THE INTERVAL CONTAINING THE WANTED EIGENVALUES  
!ENDR RIGHT END OF THE INTERVAL CONTAINING THE WANTED EIGENVALUES  
!NW LENGTH OF THE WORK ARRAY W  
!W WORK ARRAY OF LENGTH NW  
!IW WORK ARRAY OF LENGTH MAXPRS

! OUTPUTS

! EIG ARRAY OF LENGTH MAXPRS TO HOLD THE CONVERGED RITZ VALUES  
! Y ARRAY OF LENGTH MAXPRS\*N TO HOLD THE CONVERGED RITZ VECTORS  
! IERR ERROR FLAG  
! NEIG TOTAL NUMBER OF CONVERGED EIGENPAIRS

! SUBROUTINES : DDOT, GETEPS, LANSEL, OPM, RAN, STPONE

IMPLICIT DOUBLE PRECISION (A-H, O-Z)  
REAL NQ(5), Y(N,MAXPRS), EIG(MAXPRS), W(NW), IW(MAXPRS), K  
INTEGER\*2 IRANDOM1, IRANDOM2  
INTEGER NW, STARTINGVEC  
REAL ENDL, ENDR

! COMMON IDATA

!ETGL INNER MOST EIGENVALUE CONVERGED FROM LEFT END OF TRANSFORMED SPECTRUM  
!EIGR INNER MOST EIGENVALUE CONVERGED FROM RIGHT END OF TRANSFORMED  
SPECTRUM  
!NEIG1 TOTAL NUMBER OF CONVEGED EIGENVALUES

! COMMON RDATA

!RNM NORM OF THE RESIDUAL VECTOR IN R (NQ (1))  
!RNM2 SQUARE OF RNM  
!SPREAD WIDTH OF THE INTERVAL CONTAINING THE UNCONVERGED EIGENVALUES  
!TOL TOLERANCE FOR CONVERGENCE OF THE EIGENVALUES  
!EPS COMPUTER PRECISION  
!EPS1 EPS\*SQRT (N)  
!REPS SQUARE ROOT OF EPS

COMMON /IDATA/EIGL, EIGR, NEIG  
COMMON /RDATA/ RNM, RNM2, SPREAD, TOL  
COMMON /PRECISION/ EPS, EPS1, REPS  
COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN,  
ONE28, TWO56, FIVE12, ORTFAC, OVRFLW  
DATA ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28, TWO56,  
FIVE12, ORTFAC, OVRFLW / 0.0D0, 0.1D0, 0.125D0, 0.25D0, 0.5D0, 1D0, 2.0D0, 4.0D0, 10.0D0, 128.0D0,  
256.0D0, 512.0D0, 4.0D0, 1.7014D+38 /

! CHECK INPUT DATA

```

IERR = 0
MT = 6*N + 2*MAXPRS + 10*LANMAX
IF ( N .LE. 0 )      IERR = IERR + 1
IF ( LANMAX .LE. 0 ) IERR = IERR + 2
IF ( ENDR .LE. ENDL ) IERR = IERR + 4

IF ( MAXPRS .LE. 0 )      IERR = IERR + 8
IF ( MAXPRS .GT. LANMAX ) IERR = IERR + 16
IF ( LANMAX .GT. N )      IERR = IERR + 32
IF ( MT .GT. NW )         IERR = IERR + 64
IF ( IERR .GT. 0 ) RETURN

K = GETEPS(IBETA, IT, IRND)
EPS = K
REPS = SQRT (EPS)
EPS1 = EPS*SQRT (FLOAT (N))

!      SET POINTERS AND INITIALIZE

M1 = 1      + 6*N
M2 = MAXPRS + M1
M3 = MAXPRS + M2
M4 = LANMAX + M3
M5 = LANMAX + M4
M6 = LANMAX + M5
M7 = LANMAX + M6
M8 = LANMAX + M7
M9 = LANMAX + M8
M10 = LANMAX + M9
M11 = LANMAX + M10

NS = NW - MT + 2*LANMAX
NQ(1) = N + 1
DO 20 I = 2, 5
NQ(I) = NQ(I-1) + N
20 CONTINUE
DO 25 I = 1, NW
W(I) = 0
25 CONTINUE

PRINT *, 'DO YOU WANT TO INPUT YOUR STARTING VECTOR? 1 FOR YES OR 2 FOR NO'
READ *, STARTINGVEC

IF (STARTINGVEC .EQ. 1) THEN
PRINT *, 'INPUT YOUR STARTING VECTOR'
DO 27 I = 1, N

READ *, W(I)

27 CONTINUE
GO TO 50
END IF

! GET RANDOM STARTING VECTOR

IRAND = N + LANMAX + MAXPRS + NW
IRANDOM1 = IRAND

```

```

DO 40 I = 1, N
  IRANDOM2 = I
  W(I) = RAN(IRANDOM1, IRANDOM2)
40  CONTINUE

```

```

50 CALL OPM (W, W(NQ(3)), N)

```

```

RNM2 = DDOT(N, W, W(NQ(3)))

```

```

CALL STPONE (N, W(M3), W(M4), W(M5), W(M6), W, NQ(1))
CALL LANSEL (N, LANMAX, MAXPRS, NS, ENDL, ENDR, W, W(M3), W(M4), W(M5), W(M6), EIG,
W(M1), W(M2), W(M10), W(M11), W(M7), W(M8), IW, Y, W(M9), NQ, IERR)

```

```

RETURN
END

```

```

SUBROUTINE LANSEL (N,LANMAX, MAXPRS,NS, ENDL, ENDR ,R ,ALF ,BET, ALPH, BET2, EIG, TAU,
OLDTAU, RHO, WORK, ETA, OLDETA, INFO, Y, S, NQ, IERR)

```

```

!      INPUTS
!
!      N          DIMENSION OF THE EIGENPROBELM
!      LANMAX    UPPER LIMIT TO THE NUMBER OF LANCZOS STEPS
!      MAXPRS    UPPER LIMIT TO THE NUMBER OF WANTED EIGENPAIRS
!      NS        LENGTH OF THE ARRAY S
!      ENDL      LEFT END OF THE INTERVAL CONTAINING THE WANTED EIGHNVALUES
!      ENDR      RIGHT END OF THE INTERVAL CONTAINING THE WANTED
EIGHNVALUES
!
!      WORK SPACE
!
!      R          HOLDS 6 VECTORS OF LENGTH N. SEE THE TEXT FOR DETAILS
!      NQ (5)     CONTAINS THE POINTERS TO THE BEGINNING OF EACH VERTOR IN R.
!      ALF        ARRAY OF LENGTH LANMAX TO HOLD DIAGONAL OF THE
TRIDIAGONAL T
!      BET        ARRAY OF ELNGHT LANMAX TO HOLD OFF-DIAGONAL OF T
!      ALPH       DIAGONAL OF THE DEFLATED TRIDIAGONAL
!      BET2       SQUARE OF THE OFF-DIAGONALS OF THE DEFLATED TRIDIAGONAL
!      TAU        ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J
!      OLDTAU    ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J-1
!      RHO        WORKING ARRAY USED IN DEFLAT ()
!      ETA        ORTHOGONALITY ESTIMATE OF LANCZOS VECTORS AT STEP J
!      OLDETA    ORTHOGONALITY ESTIMATE OF LANCZOS VECTORS AT STEP J-1
!      INFO       INFORMATION ARRAY ABOUT EIGENVECTORS OF T
!      S          ARRAY FOR COMPUTING THE EIGENVECTORS OF THE TRIDIAGONAL
!      WORK       WORKING ARRAT TO HOLD SQUARES OF ARRAY BETA
!
!      OUTPUTS
!
!      EIG        ARRAY OF LENGTH MAXPRS TO HOLD THE CONVERGED RITZ VALUES
!      Y          ARRAT OF LENGTH MAXPRS*N TO HOLD THE CONVERGED RITZ
VALUES
!      NEIG       NUMBER OF COMPUTED EIGENPAIRS
!      IERR       ERROR FLAG

```

```

!
!       SUBROUTINES : ANALZT, ENOUGH, LANSIM, ORTBND, PURGE, RITVEC

IMPLICIT DOUBLE PRECISION (A-H, O-Z)

REAL R(NW), Y(N,MAXPRS), EIG(N), TAU(N), OLDTAU(N) , S(N, MAXPRS), NQ(5), REALS(N,
MAXPRS)
REAL ETA(LANMAX), OLDETA(LANMAX), RHO(LANMAX), WORK(LANMAX), ALPH(N), BET2(N)
REAL ENDL, ENDR, EIGL, EIGR, ALF(N), BET(N)
INTEGER INFO(MAXPRS)

LOGICAL LASTEP, ENOUGH

COMMON /IDATA/EIGL, EIGR, NEIG
COMMON /RDATA/RNM, RNM2, SPREAD, TOL
COMMON /PRECISION/ EPS, EPS1, REPS
COMMON /NW/NW

JJ      = 1
ETA(1) =EPS1
EIGL   = ENDL - (ENDR-ENDL)
EIGR = ENDR + (ENDR - ENDL)
NEIG = 0

!       LANCZOS LOOP

DO 10 J = 2, LANMAX+1
    NBUF = NS/J
    RNM = SQRT(RNM2)

!       RESTORE THE ORTHOGONALITY STATE WHEN NEEDED

CALL PURGE(R, R(NQ(1)), R(NQ(3)), R(NQ(4)), R(NQ(5)), Y, ALF, BET, S, EIG, ETA, OLDETA, TAU,
OLDTAU, WORK, INFO, N, J-1, NBUF, IERR)

IF (IERR .GT. 0) RETURN

!       UPDATE THE RITZ VALUES

CALL ANALZT (JJ, ALPH, BET2, EIG, TAU, OLDTAU, RHO, INFO)

IF ( ENOUGH(ENDL, ENDR, MAXPRS)) GO TO 30

IF (RNM .LT. REPS*SPREAD) GO TO 20

IF (J .EQ. LANMAX+1) GOTO 20

JJ = JJ +1

!       TAKE A LANCZOS STEP

CALL LANSIM(R, ALF(J), BET(J), ALPH(JJ), BET2(JJ), RNM, RNM2, NQ, N, J)

!       UPDATE THE ORTHOGONALITY BOUNDS

CALL ORTBND (ALF, BET, J, EPS1, ETA, OLDETA, TAU, OLDTAU, EIG, INFO, RNM, NEIG, N)

10      CONTINUE
20      J = J - 1

```

```

!      COMPUTE THE REMAINING RITZ VECTORS
30          DO 50 I=1, NEIG
      M = 1
      DO 40 K = 1, NEIG
          IF (INFO(K) .GT. 0) INFO(K)=-INFO(K)
          M = MIN(ABS(INFO(K)), M)
40          CONTINUE

      IF ( M .EQ. 0) THEN

CALL RITVEC(R, R(NQ(1)), R(NQ(3)), R(NQ(4)), R(NQ(5)), Y, ALF, BET, EIG, S, INFO, N, J, NEIG,
NBUF, .TRUE., WORK, IERR)
      IF (IERR .GT. 0) THEN
          RETURN
      ENDIF
      ELSE
          GO TO 60
      END IF

50 CONTINUE
60 CONTINUE

IF(J .EQ. LANMAX) THEN
IERR = -1
END IF
RETURN
END

```

```

SUBROUTINE LANSIM(R, ALF , BET , ALPH , BET2, RNM, RNM2, NQ , N, J )

```

```

!      THIS ROUTINE PERFORMS A SINGLE STEP OF THE LANCZOS ALGORITHM,
!      FOLLOWED BY A STEP OF LOCAL REORTHOGONALIZATION IF NEEDED.

```

```

!      INPUT/OUTPUT

```

```

!      R      AN ARRAY CONTAINING [R (J) , Q (J) , Q (J-1) , P (J) , MR (J) ]
!      ALF    THE NEW DIAGONAL OF T
!      BET    THE NEW OFF-DIAGONAL OF T
!      ALPH   THE NEW DIAGONAL OF THE DEFLATED T
!      BET2   THE NEW OFF-DIAGONAL SQUARED OF THE DEFLATED T
!      RNM    NORM OF R (J)
!      RNM2   RNM**2
!      NQ (5) LOCATION POINTERS FOR THE ARRAY R
!      N      DIMENSION OF THE EIGENPROBLEM
!      J      CURRENT LANCZOS STEP

```

```

!      SUBROUTINES : DAXPY, DCOPY, DDOT, DSCAL, OPK, OPM, STORE

```

```

IMPLICIT DOUBLE PRECISION (A-H, Q-Z)
REAL NQ(5) , R(NW), T, BET, BET2, ALF, ALPH, DALF, DBET

```

```

COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN
, ONE28, TWO56, FIVE12, ORTFAC, OVRFLW
COMMON /NW/NW

```

```

!      SWAP Q(J) AND Q(J-1)

NTMP = NQ(2)
NQ(2) = NQ(1)
NQ(1) = NTMP

!      Q = R/BETA

ONE = 1.0D0
T = ONE/RNM
      CALL DCOPY (N, R, R(NQ(1)))
      CALL DSCAL (N, T, R(NQ(1)))

!      P = PBAR/BETA

CALL DCOPY (N, R(NQ(3)), R(NQ(4)))
CALL DSCAL (N, T, R(NQ(4)))

!      R = ( K INVERSE ) * Q

CALL OPK (R(NQ(4)), R, N)

!      R = R - Q(J - 1) *BETA

T = -RNM
CALL DAXPY (N, T, R(NQ(2)), R)

!      STORE Q(J - 1)

CALL STORE ( R(NQ(2)), N, J-1, 1)

!      START LOCAL REORTHOGONALIZATION

BET = RNM
BET2 = RNM2
!BET = ZERO
ALF = ZERO

!      ALF = ( R TRANSPOSE ) * P

DALF = DDOT (N, R, R (NQ(4)))

DO 10 I = 1, 2
      CALL DAXPY (N, -DALF, R (NQ(1)), R)
      ALF = ALF + DALF
      CALL OPM (R, R(NQ(3)), N)
      RNM2 = DDOT (N, R, R(NQ(3)))
      ALPH = ALF

      IF (RNM2*ORTFAC.GT.(ALF**2 + BET2).OR.I.EQ.2) RETURN

!      REPEAT LOCAL REORTHOGONALIZATION WHEN WARRANTED

      DALF = DDOT (N, R(NQ(1)), R(NQ(3)))
      DBET = DDOT (N, R(NQ(2)), R(NQ(3)))
      CALL DAXPY (N, -DBET, R(NQ(2)), R)
      BET = BET + DBET
      BET2 = BET**2
10 CONTINUE; END

```

SUBROUTINE PURGE (R, Q, RA, QA, T, Y, ALF, BET, S, EIG, ETA, OLDETA, TAU, OLDTAU, WORK,  
INFO, N, J, NBUF, IERR)

! THIS ROUTINE EXAMINES ETA, OLDETA, TAU AND OLDTAU TO DECIDE  
! WHICH FORM OF REORTHOGONALIZATION IF ANY SHOULD BE PERFORMED.

! INPUT/OUTPUT

! R THE RESIDUAL VECTOR TO BECOME THE NEXT LANCZOS VECTOR  
! Q THE CURRENT LANCZOS VECTOR  
! RA THE PRODUCT OF THE MASS MATRIX AND R  
! QA THE PRODUCT OF THE MASS MATRIX AND Q  
! T A TEMPORARY VECTOR TO HOLD THE PREVIOUS LANCZOS VECTORS  
! Y CONTAINS THE COMPUTED RITZ VECTORS  
! ALF THE NEW DIAGONAL OF T  
! BET THE NEW OFF-DIAGONAL OF T  
! S VECTOR FOR COMPUTING EIGENVECTORS OF T(J)  
! EIG OLDS THE CONVERGED RITX VALUES  
! ETA STATE OF ORIHOGONALITY BETWEEN R AND PREVIOUS LANCZOS VECTORS  
! OLDETA STATE OF ORIHOGONALITY BETWEEN Q AND PREVIOUS LANCZOS VECTORS  
! TAU STATE OF ORIHOGONALITY BETWEEN R AND PREVIOUS RITZ VECTORS  
! OLDTAU STATE OF ORIHOGONALITY BETWEEN Q AND PREVIOUS RITZ VECTORS  
! WORK WORKING ARRAY EXPLAINED IN LANSSEL  
! INFO INFORMATION ABOUT THE EIGENVECTORS OF T(J)  
! N DIMENSION OF THE EIGENPROBLEM  
! J CURRENT LANCZOS STEP  
! NEIG NUMBER OF RITZ VALUES  
! NBUF NUMBER OF VECTORS IN S  
!  
! SUBROUTINES : DAXPY, DZERO, DDOT, GIVENS, IDAMAX, OPM, RITVEC, SUBTJ

IMPLICIT DOUBLE PRECISION (A-H, O-Z)

REAL R(N), Q(N), RA(N), QA(N), T(N), Y(N, MAXPRS), ALF(LANMAX), BET(LANMAX), S(N, MAXPRS)  
REAL EIG (MAXPRS), ETA(LANMAX), OLDETA(LANMAX), TAU (MAXPRS), OLDTAU (MAXPRS),  
WORK(LANMAX)  
INTEGER INFO(MAXPRS)  
LOGICAL ORTHO

COMMON/IDATA/EIGL, EIGR, NEIG  
COMMON/RDATA/RNM, RNM2, SPREAD, DUMMY  
COMMON/PRECISION/ EPS, EPS1, REPS  
COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,  
TWO56, FIVE12, ORTFAC, OVRFLW  
COMMON /LANMAX/LANMAX  
COMMON /MAXPRS/MAXPRS

ORTHO = .FALSE.  
TOL = REPS\*SPREAD  
REPSOJ = SQRT(EPS/FLOAT(J))

!CHECK ORTHOGONALITY

K = IDAMAX (J-2, ETA, 1)  
IF (ABS (ETA(K)) .GT. REPSOJ) THEN

DO 10 I=1, NEIG  
IF (INFO(I) .LT. 0 .AND. ABS(TAU(I)) .GT. REPSOJ) THEN  
CALL DZERO (J, S, 1)  
CALL SUBTJ (ALF, BET, EIG, INFO, TOL, I, N, NEIG, J, L, K, M, WORK, IERR)



```

IF (IERR .GT. 0) RETURN
CALL GIVENS (K-M+1, L-M+1, ALF(M), BET(M), EIG(I), EPS, S, RESID, RAYCOR, IERR)

IF (IERR .GT. 0) RETURN

ZETA = -DDOT (J, ETA, S)
CALL DAXPY (J, ZETA, S, ETA)
ZETA = -DDOT (J, OLDETA, S)
CALL DAXPY (J, ZETA, S, OLDETA)

END IF
10 CONTINUE

! CHECK AGAIN

K = IDAMAX (J - 2, ETA, 1)
IF (ABS(ETA(K)) .GT. REPSOJ) THEN

! GRAM-SCHMID NEEDED

ORTHO = .TRUE.
CALL RITVEC(R, Q, RA, QA, T, Y, ALF, BET, EIG, S, INFO, N, J, NEIG, NBUF, .FALSE., WORK, IERR)

IF (IERR.GT.0) RETURN

DO 20 I = 1, NEIG
    TAU(I) = EPSI
    OLDTAU(I) = EPSI
20 CONTINUE

DO 30 I = 1, J-1
    ETA(I) = EPSI
    OLDETA(I) = EPSI
30 CONTINUE

ELSE

! REMOVE COMPONENTS OF A RITZ VECTOR

ORTHO = .TRUE.
DO 40 I = 1, NEIG
    IF (ABS(TAU(I)).GT. REPSOJ) THEN

        TAU(I) = EPSI
        OLDTAU(I) = EPSI
        ZETA = DDOT(N, RA, Y(1,I))
        CALL DAXPY(N, -ZETA, Y(1,I), R)
        ZETA = DDOT(N, QA, Y(1,I))
        CALL DAXPY(N, -ZETA, Y(1,I), Q)

    END IF
40 CONTINUE

END IF
END IF
END IF

IF (ORTHO) THEN

    CALL OPM(Q, QA, N)
    QNORM = SQRT(DDOT(N, Q, QA))

```

```
CALL DSCAL(N, ONE/QNORM, Q)
CALL DSCAL(N, ONE/QNORM, QA)
```

```
BET(J) = BET(J) * QNORM
ZETA = DDOT(N, R, QA)
ALF(J) = ALF(J) + ZETA
```

```
CALL DAXPY(N, -ZETA, Q, R)
CALL OPM(R, RA, N)
RNM2 = DDOT(N, R, RA)
RNM = SQRT(RNM2)
```

```
END IF
```

```
RETURN
END
```

SUBROUTINE ANALZT(J, ALF, BET2, EIG, TAU, OLDTAU, RHO, INFO)

! THIS ROUTINE UPDATES SOME EIGENVALUES OF A TRIDIAGONAL T(J) USING  
! THE EIGENVALUES OF T(J-1).

! INPUTS

! J ORDER OF THE TRIDIAGONAL T.  
! ALF DIAGONAL OF T.  
! BET2 SQUARES OF THE OFFDIAGONAL TERMS, BET2(1) = ZERO  
! EIG ARRAY OF CONVERGED EIGENVALUES  
! TAU ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J  
! OLDTAU ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J-1  
! RHO WORKING ARRAY USED IN DEFLAT  
! INFO INFORMATION ARRAY ABOUT EIGENVECTORS OF T

! INTERNAL VARIABLES

! THET EXTERIOR EIGENVALUES OF T. NEARLY CONVERGED  
! RITZ VALUES. THET(1)=LEFTMOST, THET(NDST)=RIGHTMOST.  
! NDST SIZE OF THET AND BJ  
! BJ ERROR BOUND ON THET  
! BJ(I) IS SET TO -1 IF THET(I) DISAPPEARS.  
! NBD CONTAINS L AND R IN THE TEXT.  
! SPREAD THET(NDST) - THET(1)  
! EPS PRECISION OF ARITHMETIC OPERATIONS  
! IP IP = 1 FOR UPDATING LEFT END, IP = 2 FOR THE RIGHT END.  
! INC INC = 1 FOR UPDATING LEFT END, INC = -1 FOR THE RIGHT END.  
! IS STARTING INDEX (EITHER 1 OR NDST)  
! START LEFT BOUND ON EIGENVALUES (INC=1), RIGHT BOUND (INC=-1)  
! PROBE THE OUTER END OF THE NEXT SUBINTERVAL TO BE UPDATED.  
! INDXOK TRUE, IF THERE ARE I-INC RITZ VALUES EXTERIOR TO THE  
! NEW THET(I).

! SUBROUTINES : DEFLAT, MOVE1, NEWCOR, NUMLES

IMPLICIT DOUBLE PRECISION(A-H,O-Z)  
PARAMETER (NMAX = 128)

REAL ALF(LANMAX), BET2(LANMAX), TAU(LANMAX), OLDTAU(LANMAX), RHO(LANMAX),  
EIG(MAXPRS)  
REAL THET(NMAX), BJ(NMAX), EIGL, EIGR, INFO(MAXPRS)

LOGICAL INSERT, INDXOK, APPEND

COMMON /IDATA/ EIGL, EIGR, NEIG  
COMMON /RDATA/ RNM, RNM2, SPREAD, TOL  
COMMON /PRECISION/ EPS, EPS1, REPS  
COMMON /ATDATA/ THET, BJ, WINDOW, NBD(2), NDST  
COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,  
TWO56, FIVE12, ORTFAC, OVRFLW  
COMMON /LANMAX/LANMAX  
COMMON /MAXPRS/MAXPRS

IF (J.LE. 1) RETURN  
IF (J.EQ. 2) THEN

NDST = 16  
WINDOW = FOURTH/SQRT (REPS)

```

      THET(1) = (ALF(1) + ALF(2) - SQRT (FOUR*BET2(2) + (ALF(1) - ALF(2))**2))*HALF
      THET(NDST) = ALF(1) + ALF(2) - THET(1)
      BJ(1) = SQRT(RNM2/(ONE + BET2(2)/(THET(1) - ALF(1))**2))
      BJ(NDST) = SQRT(RNM2/(ONE + BET2(2)/(THET(NDST) - ALF(1))**2))
      NBD(1) = 1
      NBD(2) = NDST
      SPREAD = THET(NDST) - THET(1)
      RETURN
END IF

SPREAD = THET(NDST) - THET(1)
TOL = TWO*REPS*SPREAD
W = WINDOW*TOL

!!!   BEGIN PHASE 1,

!     LOOP FOR LEFT END, THEN RIGHT

DO 100 IP = 1,2
      INC = 3 - 2*IP
      IS = (NDST - 1)*IP - (NDST - 2)
      I = IS
      INSERT = .FALSE.
      START = (THET(I) + ALF(J) - INC*SQRT(BET2(J)*FOUR + (ALF(J) -
      THET(I))**2))*HALF
      PROBE = THET(I) - INC*BJ(I)
      INDXOK = NUMLES(ALF,BET2,PROBE,J,INC,EPS) .EQ. 0

      DO 50 IDUMMY = 1,NDST

          IF (I-NBD(IP) .EQ. INC) GO TO 100

!           EXAMINE I-TH SUBINTERVAL

              IF (INDXOK) THEN
                  IF (INSERT) THEN
                      START = THET(I)
                      THET(I) = START + INC*MIN(B**2/ABS(START-THET(I-INC)),B)
                  ELSE
                      IF (INT(SIGN(ONE,PROBE-START)) .EQ. INC) START = PROBE
                      END IF
              END IF

!           CHECK FOR DISJOINT SUBINTERVALS

              IF (I .EQ. NBD(IP)) THEN
                  PROBE=THET(I)+INC*(THET(NBD(2))-THET(NBD(1)))/(FOUR*J)
              ELSE
                  PROBE = THET(I+INC) - INC*BJ(I+INC)
              END IF
              IF (INT(SIGN(ONE,PROBE-THET(I))) .EQ. INC) THEN

!                   CHECK FOR AN EXTRA RITZ VALUE

                      K = NUMLES(ALF,BET2,PROBE,J,INC,EPS)

!                   IF (K .LT. ABS(I-IS+INC)) THEN
                          THET(I) DISAPPEARS

                              BJ(I) = -ONE
                              ELSE

```

```

!      RECORD INDXOK FOR NEXT LOOP. USE REFINED C BOUNDS.

                                IF (.NOT. INSERT) THEN
                                    B = BJ(I)
                                    INDXOK = (K .LE. ABS(I-IS+INC))
                                    BND = MIN(B**2/ABS(PROBE-THET(I)),B)
                                IF (INDXOK .AND. BND .LT. ABS(THET(I)-START))
THEN
                                    START = THET(I) - INC*BND
                                    END IF
                                END IF
                            END IF
                        END IF
                    ELSE
!      PREPARE FOR AN INTRUDING RITZ VALUE

                                IF ((IS .EQ. NBD(IP) .OR. BJ(NBD(IP) - INC) .LT. W) .AND. NBD(2) - NBD(1) .GT. 1)
NBD(IP) = NBD(IP) + INC
                                    CALL MOVE1 (THET,I,NBD(IP) ,-INC,PROBE)
                                    CALL MOVE1 (BJ,I,NBD(IP), -INC,TWO*TOL)
                                    INSERT = .TRUE.
                                    INDXOK = .TRUE.
                                END IF

                                IF (BJ(I) .GT. TOL) THEN
!      USE NEWTON ITERATION TO FIND NEW THET(I)

                                    CALL NEWCOR(ALF,BET2,START,THET,BJ,INC,I,J,NDST)

                                    END IF

                                IF (BJ(I) .LT. 0) THEN
!      THET(I) DISAPPEARS

                                    CALL MOVE1(THET,NBD(IP),I,INC,ZERO)
                                    CALL MOVE1(BJ,NBD(IP),I,INC,ZERO)
                                    NBD(IP) = NBD(IP) - INC
                                    INSERT = .FALSE.
                                    INDXOK = .TRUE.

                                    I = I - INC
                                END IF
                            I = I + INC
50      CONTINUE

!!!     END OF PHASE 1

100     CONTINUE

!!!     BEGIN PHASE 2.

!      APPEND MORE RITZ VALUES AND CHECK FOR CONVERGED RITZ VALUES.

DO 200 IP = 1,2
        INC = 3 - 2*IP
        IS = (NDST-1)*IP - (NDST-2)
        I = IS

```

```

DO 150 IDUMMY = 1,J
  NREM = J - NBD(1) - ((NDST+1) - NBD(2))
  IF ((I-NBD(IP))*INC .GT. 0) GO TO 200
  APPEND = I .EQ. NBD(IP) .AND. (BJ(I) .LT. W .OR. (J .EQ. 4 .AND. NBD(IP)
.EQ. IS)) .AND. NREM .GT. 0
  IF (APPEND) THEN
    START = THET(I) + FLOAT(INC)*BJ(I)
    PROBE = INC*(THET(NBD(2)) - THET(NBD(1)))/NREM
  END IF
  IF (BJ(I) .LE. TOL) THEN

!     APPLY QR ALGOR. TO DEFLATE

        CALL DEFLAT(ALF,BET2,THET(I),J)

!     INSERT THET(I) INTO EIG

        NEIG = NEIG + 1
        IF (IP .EQ. 1) THEN
          EIGL = MAX(EIGL, THET(I))
        ELSE
          EIGR = MIN(EIGR, THET(I))
        END IF
        EIG(NEIG) = THET(I)
        INFO(NEIG) = 0

!     REMOVE STABILIZED RITZ VALUES

        CALL MOVE1(THET,NBD(IP), I,INC,ZERO)
        CALL MOVE1(BJ,NBD(IP),I,INC,ZERO)
        NBD(IP) = NBD(IP) - INC
        I = I - INC
        END IF
        IF (APPEND .AND. NBD(2) - NBD(1) .GT. 1) THEN
          T = START + PROBE
          NBD(IP) = NBD(IP) + INC
          IK = ABS(IS - NBD(IP))
          DO 110 IDUM = 1,J
            IF (NUMLES(ALF,BET2,T,J,INC,EPS) .NE. IK) GO TO 120
            T = T + PROBE
110          CONTINUE
120          THET(NBD(IP)) = T
          START = T - PROBE
          CALL NEWCOR(ALF,BET2,START,THET,BJ,INC,NBD(IP),J,NDST)

        END IF
        IF (J .GT. NDST .AND. I .EQ. NBD(IP) .AND. I .NE. IS .AND. BJ(I) .GT. BJ(I-
INC) .AND. BJ(I-INC) .GT. W) NBD(IP) = NBD(IP) - INC
        I = I + INC
150 CONTINUE
200CONTINUE

!     RE-ESTABLISH AND END MARKER, IF NECESSARY, AT EARLY STAGE

DO 300 IP = 1,2
  INC = 3 - 2*IP
  IS = (NDST - 1)*IP - (NDST - 2)
  IF (NBD(IP) .EQ. IS - INC) THEN
    THET(IS) = THET(NBD(3 - IP))
    BJ(IS) = BJ(NBD(3 - IP))

```

```

                NBD(IP) = IS
                NBD(3 - IP) = NBD(3 - IP) + INC
                END IF
300    CONTINUE

RETURN
END

```

```

SUBROUTINE STPONE (N,ALF,BET,ALPH,BET2,R, NQ)

```

```

!      THIS ROUTINE PERFORMS THE FIRST STEP OF THE LANCZOS ALGORITHM.
!      IT PERFORMS A STEP OF LOCAL REORTHOGONALIZATION IF NEEDED.

```

```

!      INPUT/OUTPUT

```

```

!      N          DIMENSION OF THE EIGENPROBLEM
!      ALF        THE NEW DIAGONAL OF T
!      BET        THE NEW OFF-DIAGONAL OF T
!      ALPH       THE NEW DIAGONAL OF THE DEFLATED T
!      BET2       THE NEW OFF-DIAGONAL SQUARED OF THE DEFLATED T
!      R          AN ARRAY CONTAINING [R (J), Q(J), Q(J-1), P(J), MR(J)]
!      NQ(5)      LOCATION POINTERS FOR THE ARRAY R

```

```

!      SUBROUTINES: DAXPY, DCOPY, DDOT, DSCAL, OPK, OPM

```

```

IMPLICIT DOUBLE PRECISION (A-H, O-Z)

```

```

REAL R(NW), ALF(LANMAX), BET(LANMAX), ALPH(LANMAX), BET2(LANMAX)

```

```

REAL NQ(5), T, DALF

```

```

COMMON /LANMAX/LANMAX

```

```

COMMON /NW/NW

```

```

COMMON /RDATA / RNM,RNM2,SPREAD,TOL

```

```

COMMON /PRECISION/ EPS,EPS1,REPS

```

```

COMMON / LANCON / ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,
TWO56, FIVE12, ORTFAC, OVRFLW

```

```

!      MODIFY R TO SATISFY THE CORRECT CONDITIONS FOR SINGULAR M

```

```

T = ONE / SQRT (RNM2)

```

```

CALL DCOPY (N, R(NQ(3)), R(NQ(4)))

```

```

CALL DSCAL (N, T, R(NQ(4)))

```

```

CALL DCOPY (N, R, R(NQ(1)))

```

```

CALL DSCAL (N, T, R(NQ(1)))

```

```

CALL OPK (R(NQ(4)),R ,N)

```

```

CALL OPM(R, R(NQ(3)), N)

```

```

RNM2 = DDOT (N, R(NQ(1)), R(NQ(3)))

```

```

RNM = SQRT(RNM2)

```

```

CALL DCOPY(N, R(NQ(1)), R(NQ(2)))

```

```

BET2(1) = ZERO

```

```

ALF(1) = ZERO

```

```

DALF = RNM2

```

```

BET(1) = 0

```

```

CALL DAXPY (N, -DALF, R(NQ(1)), R)
CALL OPM (R, R(NQ(3)), N)
RNM2 = DDOT (N, R, R(NQ(3)))
RNM = SQRT(RNM2)

BET(2) = RNM
BET2(2) = RNM2
T= ONE/BET(2)

ALF(1) = ALF(1) + DALF
ALPH(1) = ALF(1)

RETURN
END
SUBROUTINE RITVEC(R,Q,RA,QA,T,Y,ALF,BET,EIG, S,INFO,N,J,NEIG,NBUF,EVONLY,WORK,IERR)

!      THIS ROUTINE COMPUTES SOME RITZ VECTORS AND PERFORMS A
!      REORTHOGONALIZATION OF THE LANCZOS VECTORS.

!      INPUT/ OUTPUT

!      R      THE RESIDUAL VECTOR TO BECOME THE NEXT LANCZOS VECTOR
!      Q      THE CURRENT LANCZOS VECTOR
!      RA     THE PRODUCT OF THE MASS MZTRIX AND R
!      QA     THE PRODUCT OF THE MASS MZTRIX AND Q
!      T      A TEMPORARY VECTOR TO HOLD THE PREVIOUS LANCZOS VECTORS
!      Y      CONTAINS THE COMPUTED RITZ VECTORS
!      ALF    THE NEW DIAGONAL OF T
!      BET    THE NEW OFF-DIAGONAL OF T
!      EIG    HOLDS THE CONVERGED RITZ VALUES
!      S      VECTOR FOR COMPUTING EIGENVECTORS OF T(J)
!      INFO   INFORMATION ABOUT THE EIGENVECTORS OF T(J)
!      N      DIMENSION OF THE EIGENPROBLEM
!      J      CURRENT LANCZOS STEP
!      NEIG   NUMBER OF RITZ VALUES
!      NBUF   NUMBER OF VECTORS IN S
!      EVONLY IF .TRUE. NO REORTHO. IS PERFORMED, COMPUTES ONLY RITZ VECTORS

!      SUBROUTINES: DAXPY,DDOT,DZERO,GIVENS,IDAMAX,NUMLES,STORE

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
REAL R(N),Q(N),RA(N),QA(N),T(N),Y(N,LANMAX),ALF(LANMAX),BET(LANMAX)
REAL EIG(MAXPRS),WORK(LANMAX), RESID, RAYCOR, INFO(MAXPRS),S(N,MAXPRS), SI
LOGICAL EVONLY

COMMON /RDATA/ RNM,RMN2,SPREAD,DUMMY
COMMON / PRECISION/ EPS,EPS1,REPS
COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,
TWO56, FIVE12, ORTFAC, OVRFLW
COMMON /LANMAX/LANMAX
COMMON /MAXPRS/MAXPRS

!      EPS14 = EPS**(1/4)
!      TOL14 = EPS**(1/4)*SPREAD
!      TOL34 = EPS**(3/4)*SPREAD

EPS14 = SQRT(REPS)
TOL14 = EPS14*SPREAD
TOL34 = REPS*TOL14

```



```

IBUF = 0
TOL = REPS*SPREAD
RNEPS = RNM*EPS

NBUF = NEIG

!      COMPUTE EIGENVECTORS OF T AND PUT IN THE BUFFER.
DO 30 I=NEIG,1,-1
      IF (INFO(I) .GE. 0 .AND. IBUF .LT. NBUF) THEN
          IF (EVONLY .AND. INFO(I) .NE. 0) GO TO 30
          IBUF = IBUF + 1
          CALL DZERO(N,Y(1,I),1)

!      EIG(I) ISOLATED

          M = 1
          L = J
          K = M
          CALL DZERO(J, S(1,IBUF),1)
          CALL GIVENS(K-M+1, L-M+1,ALF(M),BET(M),EIG(I),EPS1, S(M,IBUF), RESID,RAYCOR,IERR)
          IF (IERR .GT. 0) GO TO 900

!      CHECK THAT EIG(I) IS AN EIGENVALUE OF T

          DO 5 IDUM = M, L
              WORK(IDUM) = BET(IDUM)*BET(IDUM)
5          CONTINUE
          ZETA = EIG(I)*(ONE - EPS14)
          NUL = NUMLES(ALF(M),WORK(M), ZETA, L-M+1,1,EPS)
          ZETA = EIG(I)*(ONE + EPS14)
          NUR = NUMLES(ALF(M),WORK(M), ZETA, L-M+1,1,EPS)

!      EIG(I) IS NOT AN EIGENVALUE OF T, GOODBYE.

          IF (NUR .EQ. NUL) THEN
              IERR = IERR + 1024
              GO TO 900
          END IF

25          INFO(I) = N*IDAMAX(J,S(1,IBUF),1) + J-1
          END IF
30      CONTINUE

!      COMPUTE THE RITZ VECTORS AND PERFORM G-S ORTHOGONALIZATION.
DO 50 I=1, J-1

!      RETRIEVING THE LANCZOS VECTOR AND PUT IT IN T
      CALL STORE(T,N,I,2)
      KBUF = 0
      DO 40 K=NEIG,1,-1
          IF (INFO(K) .GE. 0 .AND. KBUF .LT. IBUF) THEN
              KBUF = KBUF + 1
              SI = S(1,KBUF)

                  IF (ABS(SI) .GT. EPS) CALL DAXPY(N, SI, T, Y(1,K))

      END IF
50      CONTINUE

```

```

40    CONTINUE
      IF (.NOT. EVONLY) THEN
          ZETOLD = -DDOT(N, QA, T)
      IF (ABS(ZETOLD) .GT. EPS) CALL DAXPY(N,ZETOLD, T, Q)
          ZETA = -DDOT(N, RA, T)
      IF (ABS(ZETA) .GT. RNEPS) CALL DAXPY(N,ZETA, T, R)
      END IF
50    CONTINUE

!      ADD IN CONTRIBUTION OF QJ TO Y
KBUF = 0
DO 60 I=NEIG,1,-1
    IF (INFO(I) .GE. 0 .AND. KBUF .LT. IBUF) THEN
        KBUF = KBUF + 1
        IF (ABS(S(J,KBUF)) .LT. EPS1) INFO(I) = -INFO(I)
        CALL DAXPY(N, S(I,KBUF), Q, Y(1,I))
    END IF
60    CONTINUE
900   RETURN
END
SUBROUTINE INPUTKEM(N)

!      THIS PROGRAM TAKES DATA OF THE ELEMENTS IN MASS AND STIFFNESS MATRICES
AND
!      PUT THEM IN SEPERATED FILES

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
REAL KE(N,N), M(N,N)
INTEGER STIFF, MASS

! INPUT
!      N   SYSTEM SIZE

! OUTPUT
!      MASS AND STIFFNESS MATRICES

PRINT *, 'DO YOU WANT TO INPUT STIFF METRIX NOW? 1 FOR YES OR 2 FOR NO'
READ *, STIFF
IF (STIFF .EQ. 1) THEN
    OPEN (13, FILE='KEDATA', STATUS='NEW')
    DO 5 I=1, N
        PRINT *, "INPUT DATA OF ", I, " LINE OF KE"
        DO 10 J=1, N
            READ (*, *) KE(I, J)
            WRITE (13, *) KE(I, J)
        10 CONTINUE
    5 CONTINUE

    REWIND 13

    CLOSE(13, STATUS='KEEP')

    END IF

PRINT *, 'DO YOU WANT TO INPUT MASS METRIX NOW? 1 FOR YES OR 2 FOR NO'
READ *, MASS

    IF (MASS .EQ. 1) THEN

```

```

      OPEN (14,FILE='MDATA',STATUS='NEW')
      DO 15 I=1,N
        PRINT *, "INPUT DATA OF ",I," LINE OF M"
        DO 20 J=1,N
          READ (*,*) M(I,J)
          WRITE (14,*) M(I,J)
        20 CONTINUE
      15 CONTINUE

      REWIND 14

      CLOSE(14, STATUS='KEEP')
      END IF

      PRINT *, '      KE      M'

      DO 25 I=1,N
        PRINT *, (KE(I,J), J=1,N), '      ', (M(I,J), J=1,N)
      25 CONTINUE

      RETURN
      END

      SUBROUTINE OPK( X, Y, N)

      !      THIS PROGRAM PERFORMS SOLVING THE LINEAR SYSTEM OF EQUATION  $Ky=x$ 
      !      K IS RECEIVED FROM STORED FILE

      INTEGER N,C,U
      REAL X(N), Y(N), KE(N,N)

      !      INPUT VECTOR X

      !      OUTPUT VECTOR Y

      OPEN (13,FILE='KEDATA',STATUS='OLD')

      DO 5 I=1,N
        DO 10 J=1,N
          READ(13,*) A,B,KE(I,J)
        10 CONTINUE
      5 CONTINUE

      REWIND 13

      CLOSE(13, STATUS='KEEP')

      100 FORMAT(I3,I3,F5.3)

      U = 1
      C=1
      I=1
      DO WHILE (I .LE. N)
        Y(C)= 0

          DO 20 J=1,N
            Y(C) = KE(I,J)*X(U) + Y(C)
          20 CONTINUE
        I=I+1
        C=C+1
      END DO

```

```

        U=U+1
        20 CONTINUE
        C=C+1
        I=I+1
        U=1
END DO

```

```

RETURN
END

```

```

SUBROUTINE OPM(X,Y,N)

```

```

!THIS PROGRAM PERFORMS SOLVING THE LINEAR SYSTEM OF EQUATION  $y = Mx$ 
!M IS RECEIVED FROM STORED FILE

```

```

INTEGER N,C,U
REAL X(N), Y(N), M(N,N)

```

```

!INPUT VECTOR X

```

```

!OUTPUT VECTOR Y

```

```

OPEN (14,FILE='MDATA',STATUS='OLD')

```

```

DO 5 I=1,N
DO 10 J=1,N
READ(14,*) A,B,M(I,J)
100 FORMAT(I3,I3,F5.3)

```

```

10 CONTINUE
5 CONTINUE

```

```

REWIND 14
CLOSE(14, STATUS='KEEP')

```

```

U = 1
C=1
I=1
DO WHILE (I .LE. N)
  Y(C)= 0

```

```

      DO 20 J=1,N
        Y(C) = M(I,J)*X(U) + Y(C)
        U=U+1
      20 CONTINUE
      C=C+1
      I=I+1
      U=1

```

```

END DO

```

```

RETURN
END

```

```

SUBROUTINE GIVENS(K,J,ALF,BET,THET,EPS,S,RES,COR,IERR)

!      GIVENS RECURRENCE FOR COMPUTING IEIGENVECTORS OF A TRIDIAGONAL T.
!
!      INPUTS
!
!      K      INDEX OF THE RIGHT HAND SIDE E(K)
!      J      DIMENSION OF THE TRIDIAGONAL MATRIX
!      ALF(J) DIAGONALS OF T
!      BET(J) OFF=DIAGONALS OF T
!      THET   EIGENVALUE OF T
!      EPS    COMPUTER PRECISION
!
!      OUTPUTS
!
!      S(J)   COMPUTED EIGENVECTOR
!      RES    NORM OF THE RESIDUAL
!      COR    RAYLEIGH CORRECTION FOR THET
!
!      SUBROUTINES: DSCAL

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
REAL ALF(J), BET(J), S(J), THET, SUM2, F, RES, COR

!      OVRFLW IS THE MACHINE OVERFLOW THRESHOLD

COMMON / LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO,FOUR, TEN, ONE28,
TWO56, FIVE12, ORTFAC, OVRFLW

!      BACKWARD RECURRENCE

BIG = SQRT(OVRFLW/FLOAT(J+1))
RES = ZERO

S(J) = ONE
SUM2 = ONE
IF (K .LT. J) THEN
    S(J-1) = -(ALF(J) - THET)*S(J)/BET(J)
    SUM2 = SUM2 + (S(J-1)**2)
END IF

DO 10 I = J-1, K+1, -1
    S(I-1) = -((ALF(I) - THET)*S(I) + BET(I+1)*S(I+1))/BET(I)

!      SCALE TO AVOID OVERFLOW

    IF (ABS(S(I-1)) .GT. BIG) THEN
        F = ONE/S(I-1)
        S(I-1) = ONE
        CALL DSCAL(J-I+1, F, S(I))
        SUM2 = (SUM2*F)*F
    END IF
    SUM2 = SUM2 + S(I-1) ** 2
10  CONTINUE

!      STOP EXECUTION GRACEFULLY IF S(K) IS EXACTLY ZERO

IF (S(K) .EQ. ZERO) THEN
    IERR = IERR + 256
    RETURN

```

```

END IF
F = ONE/S(K)
S(K) = ONE
SUM2 = (SUM2*F) * F

CALL DSCAL (J-K, F, S(K+1))

IF (K .LE. 1) GO TO 30

!      FORWARD RECURRENCE

      X = ZERO
      S(1) = ONE
      SUM1 = ONE
      IF (K.GT.2) THEN
      S(2) = -((ALF(1)-THET)*S(1))/BET(2)
      SUM1=SUM1 + S(2) ** 2

      DO 20 I = 2,K-2
          S(I+1) = -((ALF(I) - THET)*S(I) + BET(I) * S(I-1))/BET(I+1)
          IF (ABS(S(I+1)) .GT. BIG) THEN
          F = ONE/S(I+1)
          S(I+1) = ONE
          CALL DSCAL(I,F,S)
          SUM1 = (SUM1*F)*F
          END IF
          SUM1= SUM1 + S(I+1) **2
20      CONTINUE
      X = BET(K-1) * S(K-2)
END IF
X = -(X + (ALF(K-1) - THET)*S(K-1))/BET(K)

!      MATCH X WITH S(K)

IF (X .EQ. ZERO) THEN
      RES = ONE
RETURN
END IF

F = S(K)/X
CALL DSCAL(K-1, F, S)
SUM2 = SUM2 + SUM1 *F**2
RES = BET(K) * S(K-1)

!      NORMALIZE S

30      F = ONE/SQRT (SUM2)

CALL DSCAL (J, F, S)
RES = RES*F + (ALF(K) - THET) *S(K)
IF ( K .LT. J) RES = RES + BET (K+1)*S(K+1)
COR = S(K)*RES
RES = ABS(RES)

RETURN
END

```

```

SUBROUTINE NEWCOR(ALF,BET2,ZETA,THET,BJ,INC,INDX,J,NDST)

!      COMPUTES EXTERIOR EIGENVALUES OF A TRIDIAGONAL USING A
!      COMBINATION OF BISECTIONS AND NEWTON'S METHOD

!      INPUT

!      ALF   DIAGONAL OF T
!      BET2  SQUARES OF THE OFFDIAGONAL TERMS, BET2(1) = ZERO
!      SPREAD THET(NDST) - THET(1)
!      INC   INC = 1 FOR UPDATING LEFT END, INC = -1 FOR THE RIGHTEND.
!      INDX  INDEX OF TO-BE-UPDATED THET.
!      J     ORDER OF THE TRIDIAGONAL T.
!      NDST  SIZE OF THET AND BJ

!      INPUT/OUTPUT

!      ZETA  EXTERIOR BOUND FOR EIGENVALUE OF T IN THET[INDX]
!      THET  EXTERIOR EIGENVALUES OF T, NEARLY CONVERGED RITZ VALUES,
!            THET (1) = LEFTMOST, THET(NDST) = RIGHTMOST.
!      BJ    ERROR BOUDN ON THET
!
!      SUBROUTINES : NUMLES, QLBOT

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER (MAXBIS = 15, MAXNEW = 40)
REAL ALF(LANMAX),BET2(LANMAX),THET(NDST), BJ(NDST)

COMMON /RDATA/ RNM, RNM2,SPREAD,TOL
COMMON /PRECISION/ EPS, EPS1, REPS
COMMON /LANCON /ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,
TWO56, FIVE12, ORTFAC, OVRFLW
COMMON /LANMAX/LANMAX
COMMON /MAXPRS/MAXPRS
COMMON /NW/NW
COMMON /N/N

ZOLD = ZETA
IF (J .EQ. 1) THEN
    ZETA = THET(INDX)
    THET(INDX) = ALF(J)
    BJ(INDX) = SQRT(RNM2)
    RETURN
END IF

!      PERFORM BISECTION FOR AN IMPROVED ZETA

IS = ((NDST + 1) - (NDST - 1) * INC)/2
FACT = FIVE12*FLOAT(J)*LOG(FLOAT(J))
WIDTH = (THET(INDX) - ZETA)*HALF
IF (WIDTH .EQ. ZERO) THEN
    WIDTH = BJ(INDX)*HALF
END IF
IOLD = ABS(IS - INDX)
DO 10 IDUMMY = 1, MAXBIS
    IF (ABS(WIDTH)*FACT .LE. ABS(THET((NDST + 1) - IS) - THET(INDX))) GO TO 20
    ZNEW = ZETA + WIDTH
    INEW = NUMLES(ALF, BET2, ZNEW, J, INC, EPS)
    WIDTH = WIDTH*HALF
    IF (INEW .EQ. IOLD) THEN

```

```

                ZETA = ZNEW
            END IF

10      CONTINUE
20      CONTINUE

DO 50 IDUMMY = 1, MAXNEW
    U = ALF(1) - ZETA
    IF (U .EQ. ZERO) U = TENTH*EPS*BET2(2)
    RAT = ONE/U
    SUM = RAT
    DO 30 I = 2,J
        H = BET2(I)/U
        U = ALF(I) - ZETA - H
        IF (U .EQ. ZERO) U = TENTH*EPS*(H + BET2(I))
        RAT = (ONE + H*RAT)/U
        SUM = SUM + RAT
30      CONTINUE
        BOT2 = U*SUM

!          DEFLATION

        DO 40 I = IS, INDX-INC,INC
            DEL = ZETA - THET(I)
            IF (ABS(DEL) .LT. EPS*ABS(ZETA)) THEN
                DEL = EPS*ABS(ZETA)
            END IF
            SUM = SUM + ONE/DEL
40      CONTINUE

!          CHECK FOR CONVERGENCE

        ZNEW = ZETA + ONE/SUM
        ZETA = ZNEW
        IF (SPREAD + TENTH/SUM .EQ. SPREAD .OR. FLOAT(INC)/SUM .LT. ZERO) THEN
            GO TO 60
        END IF
50      CONTINUE
60      CONTINUE

CALL QLBOT(ALF,BET2,ZETA,BOT2,J)

ZNEW = THET(INDX)
THET(INDX) = ZETA
ZETA = ZNEW
BJ(INDX) = SQRT(RNM2*BOT2)

RETURN
END

```



```

SUBROUTINE ORTBND (ALF, BET, J, EPS1, ETA, OLDETA, TAU, OLDTAU, EIG, INFO, RNM, NEIG, N)

!      UPDATE THE ETA AND TAU RECURRENCES.

!      INPUTS

!      ALF (J)      DIAGONAL OF THE TRIDIAGONAL T
!      BET (J)      OFF-DIAGONAL OF T
!      J            DIMENSION OF T
!      EPS1        ROUNDOFF ESTIMATE FOR DOT PRODUCT OF TWO UNIT VECTORS
!      ETA (J)      ORTHOGONALITY ESTIMATE OF LANCZOS VECTORS AT STEP J
!      OLDETA (J)   ORTHOGONALITY ESTIMATE OF LANCZOS VECTORS AT STEP J-1
!      TAU (NEIG)   ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J
!      OLDTAU (NEIG) ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J-1
!      EIG (NEIG)   ARRAY OF CONVERGED EIGENVALUES
!      INFO (NEIG)  INFORMATION ARRAY ABOUT EIGENVECTORS OF T
!      RNM          NORM OF THE NEXT RESIDUAL VECTOR
!      NEIG         NUMBER OF CONVERGED EIGENVALUES
!      N           DIMENSION OF THE EIGENPROBLEM
!
!      OUTPUTS
!
!      ETA (J)      ORTHOGONALITY ESTIMATE OF LANCZOS VECTORS AT STEP J+1
!      OLDETA (J)   ORTHOGONALITY ESTIMATE OF LANCZOS VECTORS AT STEP J
!      TAU (NEIG)   ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J+1
!      OLDTAU (NEIG) ORTHOGONALITY ESTIMATE OF RITZ VECTORS AT STEP J

IMPLICIT DOUBLE PRECISION (A-H, O-Z)
REAL ALF(LANMAX), BET(LANMAX), ETA(LANMAX), OLDETA(LANMAX), TAU(LANMAX),
OLDTAU(LANMAX)
REAL EIG(MAXPRS), INFO(MAXPRS)
COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,
TWO56, FIVE12, ORTFAC, OVRFLW
COMMON /MAXPRS/MAXPRS
COMMON /LANMAX/LANMAX

IF (J.GT. 1) THEN
  OLDETA(1) = ( BET(2)*ETA(2) + (ALF(1) - ALF(J))*ETA(1) - BET(J)*OLDETA(1) ) / RNM
  J1 = J - 1
  IF (J.GT. 2) THEN
    DO 100 K=2, J1
      OLDETA (K) = (BET (K+1)*ETA (K+1) + (ALF (K) - ALF (J) ) * ETA (K) + BET
(K)*ETA (K-1) - BET (J)*OLDETA(K) ) / RNM
    100 CONTINUE
  END IF

  DO 200 K=1, J1
    T = OLDETA(K)
    OLDETA (K) = ETA (K)
    ETA (K) = T
  200 CONTINUE

END IF

ETA (J) = EPS1*MAX(BET(2)/RNM, ONE)

!      UPDATE THE TAU RECURRENCE.

DO 300 I=1, NEIG

```

```
IF ( INFO(I) .NE. 0 ) THEN
  T = TAU(I)
  TAU (I) = (EIG(I) - ALF(J))*TAU(I) - BET(J)*OLDTAU(I)
  OLDTAU(I) = T
END IF
300  CONTINUE

RETURN
END
```

```

SUBROUTINE QLBOT(ALF,BET2,THET,BOT,J)

!      COMPUTES THE BOTTOM ELEMENT OF THE NORMALIZED EIGENVECTOR OF A
!      TRIDIAGONAL MATRIX CORRESPONDING TO EIGENVALES THET.
!

!      INPUTS

!      ALF(J)  DIAGONALS OF T
!      BET2(J) SQUARE OF THE OFF-DIAGONALS OF T
!      THET   EIGENVALUE OF T
!      J      DIMENSION OF THE TRIDIAGONAL MATRIX
!

!      OUTPUTS

!      BOT    BOTTOM ELEMENT OF THE NORMALIZED EIGENVECTOR

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
REAL ALF(LANMAX),BET2(LANMAX)

COMMON /LANCON /ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO,FOUR, TEN, ONE28, TWO56,
FIVE12, ORTFAC, OVRFLW
COMMON /LANMAX/LANMAX

BOT = ONE
C = ONE
S = ZERO
G = ALF(J) - THET
P = G**2

DO 100 I = J-1, 1, -1
    B = BET2(I + 1)
    R = P + B
    OLDC = C
    C = P/R
    S = B/R
    OLDG = G
    A = ALF(I)
    G = C*(A - THET) - S*OLDG
    IF (C .EQ. ZERO) THEN
        P = OLDC*B
    ELSE
        P = G**2/C
    END IF
    BOT = BOT*S
100 CONTINUE

RETURN
END

```

SUBROUTINE SUBTJ (ALF, BET, EIG, INFO, TOL, I, N, NEIG, J, L, K, M, U, IERR)

! THIS ROUTINE SCANS BACK THROUGH THE CONVERGED EIGENVALUES  
! FOR COPIES OF EIG(I) TO DETERMINE THE SUBMATRIX T(M, L) AND  
! THE RIGHT HAND SIDE E(K) FOR THE GIVENS RECURRENCE.

! INPUTS

! EIG(NEIG) LIST OF THE CONVERGED EIGENVALUES  
! INFO(NEIG) INFORMATION ABOUT THE EIGENVECTORS  
! TOL TOLERANCE FOR FINDING COPIES OF EIG(I)  
! I INDEX OF THE EIGENVALUE CONSIDERED  
! N DIMENSION OF THE EIGENPROBLEM  
! NEIG NUMBER OF EIGENVALUES IN EIG

! OUTPUTS

! K INDEX OF THE RIGHT HAND SIDE FOR GIVENS  
! L INDEX OF THE LAST ELEMENT OF THE SUBMATRIX  
! M INDEX OF THE FIRST ELEMENT OF THE SUBMATRIX

IMPLICIT DOUBLE PRECISION (A-H, O-Z)  
REAL ALF(LANMAX), BET(LANMAX), EIG(MAXPRS), U(LANMAX)  
INTEGER INFO(MAXPRS)

COMMON /LANCON/ ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28, TWO56,  
FIVE12, ORTFAC, OVRFLW  
COMMON /RDATA/ RNM, RNM2, SPREAD, DUMMY  
COMMON /PRECISION/ EPS  
COMMON /LANMAX/LANMAX  
COMMON /MAXPRS/MAXPRS

L = J  
EIGI = EIG(I)

! CHECK SOPECIAL CASE OF EIGI = ALL ALTERNATE ALF'S

DO 10 JM = J, 1, -2  
IF (ABS (ALF(JM) - EIGI) .GE. EPS\*SPREAD) GO TO 20  
10 CONTINUE

! FALSE RITZ VALUE ACCEPTED, STOP EXECUTION GRACEFULLY.

IERR = IERR + 2048  
RETURN

! RUN RECURRENCE UNTIL "PIVOT = DIAGONAL " TO FIND M

20 U(JM) = (ALF(JM) - EIGI)\*(ONE - TEN\*EPS)  
UMIN = ABS (U(JM) )  
IUMIN = JM  
DO 30 M = JM-1, 1, -1  
U(M) = ALF(M)-EIGI-BET(M+1)\*\*2/U(M+1)  
IF (ABS (U(M)) .LT. TOL\*EIGHTH) GO TO 40  
IF (ABS (U(M)) .LE. UMIN) THEN  
UMIN = ABS(U(M))  
IUMIN = M  
ENDIF  
30 CONTINUE

```

!      TEMPORARILY
M = IUMIN
!      NOW SET K
40     IF (M .EQ. 1) THEN
        K = M
        ELSE
        K = M+1
        ENDIF
!      NOW CHECK FOR CLOSE EIGENVALUES
IF (M .GT. 1) THEN
    DO 50 I1 = I-1, 1, -1
    IF (ABS(EIGI-EIG(I1)) .LT. ONE28*EPS*SPREAD) GO TO 60
50     CONTINUE
        RETURN
60     KPREV = MOD(ABS(INFO(I1)), N)
        IF (KPREV .LE. K) K = KPREV+1
    ENDIF
!      FIND L
DO 200 I2 = I+1, NEIG
    IF (ABS (EIGI - EIG(I2)) .LT. TOL) GO TO 210
200    CONTINUE
210    IF (I2 .LE. NEIG) THEN
        L = ABS(INFO(I2))/N - 1
        IF (INFO(I2) .EQ. 0) L = MOD (ABS(INFO(I)), N)
    ENDIF
RETURN
END

```

```

SUBROUTINE DEFLAT(ALF,BET2,THET,J)

!      THIS ROUTINE PERFORMS DEFLATION OF T USING SHIFT THET.

!      INPUTS

!      ALF(J)          DIAGONALS OF T
!      BET2(J)         SQUARES OF OFF-DIAGONALS OF T
!      THET           EIGENVALUE OF T TO BE DEFLATED EXPLICITLY
!      J              DIMENSION OF THE TRIDIAGONAL MATRIX

!      OUTPUTS

!      ALF(J - 1)     MODIFIED DIAGONALS OF T
!      BET2(J - 1)    MODIFIED OFF-DIAGONALS OF T

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER (MAXITR = 3)
REAL ALF(J),BET2(J), THET

COMMON /RDATA /RNM,RNM2,SPREAD,TOL
COMMON /PRECISION/EPS,EPS1,REPS
COMMON /LANCON /ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28,
TWO56, FIVE12, ORTFAC, OVRFLW

!      PWK ALGORITHM

DO 200 LOOP = 1,MAXITR
      C = ONE
      S = ZERO
      G = ALF(1) - THET
      P = G**2
      DO 100 I = 1, J-1
        B = BET2(I + 1)
        R = P + B
        BET2(I) = S * R
        OLDC = C
        C = P/R
        S = B/R
        OLDG = G
        A = ALF(I + 1)
        G = C*(A - THET) - S*OLDG
        ALF(I) = OLDG + (A - G)
        IF (C .EQ. ZERO) THEN
          P = OLDC * B
        ELSE
          P = G*(G/C)
        END IF
      100 CONTINUE

      BET2(J) = S * P
      ALF(J) = G + THET
      IF (BET2(J) .LE. EPS*SPREAD) THEN
        GO TO 300
      END IF
      200 CONTINUE
      300 J = J - 1

RETURN
END

```

```

SUBROUTINE MOVE1(Y,K,L,MINC,T)
!   MOVES THE CONTENT OF Y TO OPEN A SPACE FOR T.
!
!   INPUT/OUTPUT
!
!   Y       THE ARRAY TO TO BE REORGANIZED
!   K       THE POSITION IN Y OF THE NEW ELEMENT T
!   L       END OF THE DATA IN Y
!   MINC    THE INCREMENT + 1 OR -1
!   T       THE NEW ELEMENT TO BE INSERTED

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
REAL Y(L)

DO 100 I = L, K - MINC, MINC
    Y(I) = Y(I + MINC)
100  CONTINUE
Y(K) = T

RETURN
END

```

```

SUBROUTINE STORE(V, N, JI, ISW)

REAL TEMPV(N,N),V(N)
INTEGER JI, ISW, N
LOGICAL STOREFIRST
COMMON /STOREFIRST/STOREFIRST

IF (STOREFIRST) THEN

OPEN (15, FILE='TEMPV', STATUS='NEW')

DO 50 I=1,N
DO 100 J=1,N
WRITE (15,*) 0
100 CONTINUE
50 CONTINUE

REWIND 15

STOREFIRST = .FALSE.
ELSE

GO TO 1500

END IF
1500 OPEN (15, FILE='TEMPV',STATUS='OLD')

DO 5 I=1,N
DO 10 J=1,N

READ (15,*) TEMPV (I,J)

10 CONTINUE
5 CONTINUE
REWIND 15

IF (ISW .EQ. 1) THEN
DO 15 I=1,N
TEMPV (JI,I) =V(I)
15 CONTINUE

DO 500 I=1,N
DO 1000 J=1,N
WRITE(15,*) TEMPV(I,J)
1000 CONTINUE
500 CONTINUE
REWIND 15
END IF

IF (ISW .EQ. 2) THEN
DO 20 I=1,N
V(I) = TEMPV(JI,I)
20 CONTINUE
REWIND 15
END IF

CLOSE(15, STATUS='KEEP')
RETURN
END

```



```

FUNCTION RAN(RANDOM1, RANDOM2)
!THIS PROGRAM RANDOMLY SELECT A NUMBER TO PUT INTO STARTING VECTOR
!INPUT RANDOM1, RANDOM2
!OUTPUT RAN

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
REAL H1, H2, H3
INTEGER*2 RANDOM1, RANDOM2
CALL RANDOM_NUMBER (H1)
CALL RANDOM_NUMBER (H2)
CALL RANDOM_NUMBER (H3)

10 IF (H1*H2 .LT. 0.5) THEN

20 IF (H3 .GE. 1) THEN
GO TO 30
ELSE
H3 = H3*10
GOTO 20
END IF
END IF

30 IF (H3 .GT. 5) THEN

RAN= RANDOM1

ELSE

RAN = RANDOM2

END IF
END

```

SUBROUTINE DSCAL(N,B,Y)

! THIS PROGRAM PERFORMS SCALAR MULTIPLICATION OF MATRIX  
! INPUT ORIGINAL MATRIX Y AND SCALAR B  
! OUTPUT MULTIPLICATED MATRIX Y

REAL Y(N),B

DO 5 I=1,N  
Y(I) = B\*Y(I)  
5 CONTINUE

RETURN  
END

SUBROUTINE DAXPY(N, A, X, Y)

! THIS PROGRAM PERFORMS  $Y=AX+Y$   
! INPUT MATRIX Y, X AND SCALAR A  
! OUTPUT CALCULATED Y

IMPLICIT DOUBLE PRECISION (A-H, O-Z)  
INTEGER N  
REAL A, Y(N), X(N)

DO 5 I=1,N  
Y(I) = A\*X(I) + Y(I)  
5 CONTINUE

RETURN  
END

SUBROUTINE DCOPY(N,X,Y)

! THIS PROGRAM PERFORMS COPYING X AND PUT INTO Y  
! INPUT MATRIX X  
! OUTPUT MATRIX Y

IMPLICIT DOUBLE PRECISION (A-H, O-Z)

INTEGER N  
REAL X(N), Y(N)

DO 5 I=1,N  
Y(I) = X(I)  
5 CONTINUE

RETURN  
END

SUBROUTINE DZERO(N,Y,C)

!THIS PROGRAM RESET THE ELEMENTS OF A VECTOR Y TO ZERO  
!INPUT Y  
!OUTPUT ZERO MATRIX Y

IMPLICIT DOUBLE PRECISION (A-H, O-Z)

REAL Y(N), C

DO 5 I=1,N  
Y(I) = 0  
5 CONTINUE

RETURN  
END

## FUNCTION

LOGICAL FUNCTION ENOUGH (ENDL, ENDR, MAXPRS)

! EXMAINE IF ENOUGH EIGENVALUES HAVE CONVERGED

! INPUT

! ENDL LEFT END OF THE INTERVAL CONTAINING THE WANTED EIGENVALUES  
! ENDR RIGHT END OF THE INTERVAL CONTAINING THE WANTED

EIGENVALUES

! MAXPRS UPPER LIMIT TO THE NUMBER OF WANTED EIGENPAIRS

IMPLICIT DOUBLE PRECISION (A-H, O-Z)  
PARAMETER (NMAX = 128)

COMMON/ATDATA/THET(NMAX) , BJ(NMAX) , WINDOW, NBD(2) , NDST  
COMMON/IDATA/EIGL, EIGR, NEIG

ENOUGH = .TRUE.  
IF ( NEIG .GT. MAXPRS ) RETURN

ENOUGH = (THET (1) .GT. ENDL .AND. THET (NDST) .LT. ENDR) .AND. (( EIGL .GT. ENDL .AND.  
EIGL .LT. ENDR ) .OR. ( EIGR .GT. ENDL .AND. EIGR .LT. ENDR ))

RETURN  
END

```

INTEGER FUNCTION NUMLES(ALF,BET2, ZETA, N, INC, EPS)

!      ROUTINE TO PERFORM THE SPECTRUM SLICING OF A TRIDIAGONAL MATRIX.

!      IF INC = 1, NUMLES RETURNS THE NUMBER OF EIGENVALUES BELOW ZETA.
!      IF INC = -1, NUMLES RETURNS THE NUMBER OF EIGENVALUES ABOVE ZETA.

!      INPUTS

!      ALF(N)          DIAGONALS OF T
!      BET2(N)         SQUARE OF THE OFF-DIAGONALS OF T
!      ZETA            THE SHIFT TO BE APPLIED TO T
!      N              DIMENSION OF THE TRIDIAGONAL MATRIX
!      INC             INDEX TO INDICATE ABOVE OR BELOW
!      EPS            COMPUTER PRECISION

!      OUTPUTS

!      NUMLES         THE NUMBER OF EIGENVALUES ABOVE/BELOW ZETA.

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
REAL ALF(LANMAX), BET2(LANMAX), SAVE

COMMON /LANCON /ZERO, TENTH, EIGHT, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28, TWO56,
FIVE12, ORTFAC, OVRFLW
COMMON /LANMAX/LANMAX

SAVE = BET2(1)
BET2(1) = ZERO
DEL = ONE
K = 0
DO 10 J = 1,N
    DEL = (ALF(J) - ZETA) - BET2(J)/DEL
    IF (DEL .EQ. ZERO) DEL = EPS*BET2(J + 1) * INC
    IF (DEL .LT. ZERO) K = K + 1
10  CONTINUE

NUMLES = K
IF (INC .LT. 0) NUMLES = N - K
BET2(1) = SAVE

RETURN
END

```

FUNCTION GETEPS (IBETA, IT, IRND)

! THIS PROGRAM DETERMINES THE COMPUTER PRECISION AND COMPUTES UNIT ROUND OFF ERROR

! OUTPUT IBETA, IT, IRND AND COMPUTER PRECISION EPS

IMPLICIT DOUBLE PRECISION (A-H, O-Z)

COMMON / LANCON / ZERO, TENTH, EIGHTH, FOURTH, HALF, ONE, TWO, FOUR, TEN, ONE28, TWO56, FIVE12, ORTFAC, OVRFLW

! DETERMINE IBETA, BETA

```
A = ONE
10   A = A + A
IF ((( A + ONE) - A) - ONE .EQ. ZERO) GO TO 10
B = ONE
20   B = B + B
IF (( A + B) - A .EQ. ZERO) GO TO 20
IBETA = INT (SNGL ((A + B) - A))
BETA = FLOAT (IBETA)
```

! DETERMINE IT, IRND

```
IT = 0
B = ONE
30   IT = IT + 1
B = B * BETA
IF (((B + ONE) - B) - ONE .EQ. ZERO) GO TO 30
IRND = 0
BETAM1 = BETA - ONE
IF ((A + BETAM1) - A .NE. ZERO) IRND = 1
```

! DETERMINE EPS

```
BETA1N = ONE / BETA
A = ONE
DO 40 I = 1, IT + 3
A = A * BETA1N
40   CONTINUE

50   IF ((ONE + A) - ONE .NE. ZERO) 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*(ONE + A)) / (ONE + ONE)
IF ((ONE + A) - ONE .NE. ZERO) EPS = A
70   GETEPS = EPS
```

```
RETURN
END
```

```
FUNCTION DDOT(N,X,Y)
```

```
!THIS FUNCTION PERFORMS EUCLIDEAN INNER PRODUCT OF MATIRX X AND Y  
!INPUT MATRIX X AND Y  
!OUTPUT DDOT
```

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)  
INTEGER N  
REAL X(N), Y(N)
```

```
DDOT=0  
DO 5 I=1,N  
DDOT = DDOT + X(I)*Y(I)  
5 CONTINUE
```

```
RETURN  
END
```

```
FUNCTION IDAMAX(N,V,C)
```

```
!THIS PROGRAMS FINDS THE INDEX OF THE ELEMENT OF A VECTOR V WITH MAXIMUM  
ABSOLUTE VALUE  
!INPUT VECTOR V  
!OUTPUT INDEX IDAMAX
```

```
IMPLICIT DOUBLE PRECISION(A-H,O-Z)  
INTEGER N  
REAL V(N)
```

```
IDAMAX = 1  
MAXI= ABS(V(1))
```

```
DO 5 I=2,N  
IF (ABS(V(I)) .GT. MAXI) THEN  
MAXI = ABS(V(I))  
IDAMAX = I
```

```
END IF  
5 CONTINUE
```

```
RETURN  
END
```

**APPENDIX C**  
**DEFINITION OF VARIABLE USED IN LANCZOS**  
**ALGORITHM PROGRAM PACKAGE**

<b>ALF</b>	An array containing elements of diagonal of [T]
<b>ALPH</b>	An array containing elements of diagonal of deflated tridiagonal matrix
<b>BET</b>	An array containing elements of off-diagonal of [T]
<b>BET2</b>	An array containing elements of squares of the Off-diagonal terms, BET2(1) = 0
<b>BJ</b>	An array containing element of error bounds of elements in THET, BJ(I) is set to -1 if THET(I) has been converged
<b>BOT</b>	A value containing the bottom element of the normalized eigenvector
<b>COR</b>	A variable containing the Rayleigh correction for THET
<b>EIG</b>	An array containing elements of converged eigenvalues
<b>ENDL</b>	A value containing the left end of the Interval containing the wanted eigenvalues
<b>ENDR</b>	A value containing the right end of the Interval containing the wanted eigenvalues
<b>EPS</b>	The computer precision
<b>EPS1</b>	A value containing roundoff estimate for dot-product of two unit vectors
<b>ETA</b>	An array containing elements of orthogonality estimate of Lanczos vectors at step J
<b>EVONLY</b>	A logical variable used in controlling subroutine RITVEC, computed by subroutine PURGE. When true, no reorthogonalization is performed. When false, it is perform as well as computing Ritz vector



<b>I</b>	A value used by subroutine SUBTJ representing the index of the eigenvalue being considered
<b>IERR</b>	The error indicator
<b>INC</b>	A variable used in subroutine ANALZT, INC = 1 means updating left end, INC = 2 means updating right end
<b>INDX</b>	A value used in subroutine ANALZT indication the index of to-be-updated THET
<b>INDXOK</b>	A logical variable used in subroutine ANALZT, the value is true if there are (I-Inc) Ritz values exterior to the new THET(I)
<b>INFO</b>	An array containing elements of convergence information of eigenvalues of T
<b>IP</b>	A variable used in subroutine ANALZT, IP = 1 means updating left end, IP = 2 means updating right end
<b>IS</b>	A management variable used in subroutine ANALZT representing the starting index (1 or NDST)
<b>IW</b>	An working array of length MAXPRS
<b>J</b>	Ranking of the current step of Lanczos algorithm
<b>K</b>	A value used by subroutine SUBTJ representing the index of the right hand side in subroutine
<b>L</b>	A value used by subroutine SUBTJ representing the index of the last element of the sub-matrix

L	A value used in subroutine MOVE1 representing the end of the data in vector Y
LANMAX	A value containing the upper limit of the number of Lanczos step
M	A value used by subroutine SUBTJ representing the index of the first element of the sub-matrix
MAXPRS	A value containing the upper limit of the number of wanted eigenpairs
MINC	A value used in subroutine MOVE1 representing the increment +1 or -1
NDST	A number of size of THET and BJ
N	A value containing the size of the pursuing eigenproblem
NBD	A value containing L and R in the text
NBUF	A value representing the number of vectors in array S
NEIG	A value indicating number of converged eigenvalues
NS	A value containing the length of the array containing eigenvectors, S
NQ	An array containing the pointers of the beginning position of running arrays and vectors to global vector R or W
NUMLES	A variable representing the number of eigenvalues above or below ZETA
NW	A value containing the size of the global working array
OLDETA	An array containing elements of orthogonality estimate of Lanczos vectors at step J -1

OLDTUA	An array containing elements of Orthogonality estimate of Ritz vectors at step J-1
PROBE	A variable used in subroutine ANALZT representing the outer end of the next subinterval to be updated
SPREAD	A working variable that is equal to THET(NDST)-THEY(1)
START	A value containing the eigenvalue bound, left bound when INC=1 and right bound when INC=-1
Q	An array containing elements of the current Lanczos vector
QA	An array containing elements of the product of mass matrix and Lanczos vector
R	A global array passed down from global array W; in some subroutine and functions, it will represent only first N elements as the residual vector, $\{r\}_j$
RA	An array containing elements of the product of mass matrix and residual vector
RES	A variable containing the norm of the residual of eigenvector
RHO	A working array that is used in subroutine DEFLAT
RNM	A value containing norm of $\{r\}_j$
RNM2	A value containing $RNM^2$
S	A three-dimension array containing Computed eigenvectors

<b>T</b>	<b>A value used in subroutine MOVE1 representing the value of new element</b>
<b>T</b>	<b>A temporary array used by subroutine RITVEC to hold previous Lanczos vector</b>
<b>TAU</b>	<b>An array containing elements of orthogonality estimate of Ritz vectors at step J</b>
<b>THET</b>	<b>An array containing exterior eigenvalues of T, which are nearly converged Ritz values. THET(1) = current most left interval. THET(NDST)= current most right interval.</b>
<b>TOL</b>	<b>A value used by subroutine SUBTJ representing the tolerance of two eigenvalues considered duplicate</b>
<b>W</b>	<b>A global working array</b>
<b>WORK</b>	<b>An array containing elements of squares of array BET</b>
<b>Y</b>	<b>A Three-dimension array containing converged Ritz vector</b>
<b>ZETA</b>	<b>A value used in subroutine NEWCOR representing exterior bound of eigenvalue of T in THET(INDX)</b>

**APPENDIX D**  
**EXAMPLES OF EIGENVALUE PROBLEM**  
**CALCULATION**

## EXAMPLE I

The dimension of concerned problem is 6 X 6.

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

The first few eigenvalues of  $[K]\{\phi\} = \lambda[M]\{\phi\}$  can be calculated by Lanczos package program. Two files containing values of  $[M]$  and  $[K]^{-1}$  are stored in Lanczos Package subfolder named MDATA and KDATA consecutively.  $[K]^{-1}$  is obtained from another matrix calculation package.

The input data required by Lanczos package are number of system's dimension, maximum limit of Lanczos steps and eigenvalues wanted. The initial vector is optional to be given by user. As shown below, the first three calculated eigenvalues of the system are 0.1980623, 0.7532398, 1.554958, 2.4398140. These answers are acceptable accurate values comparing to the exact ones. The interface window shown below is interfacing sentence of Lanczos Package.



## EXAMPLE II

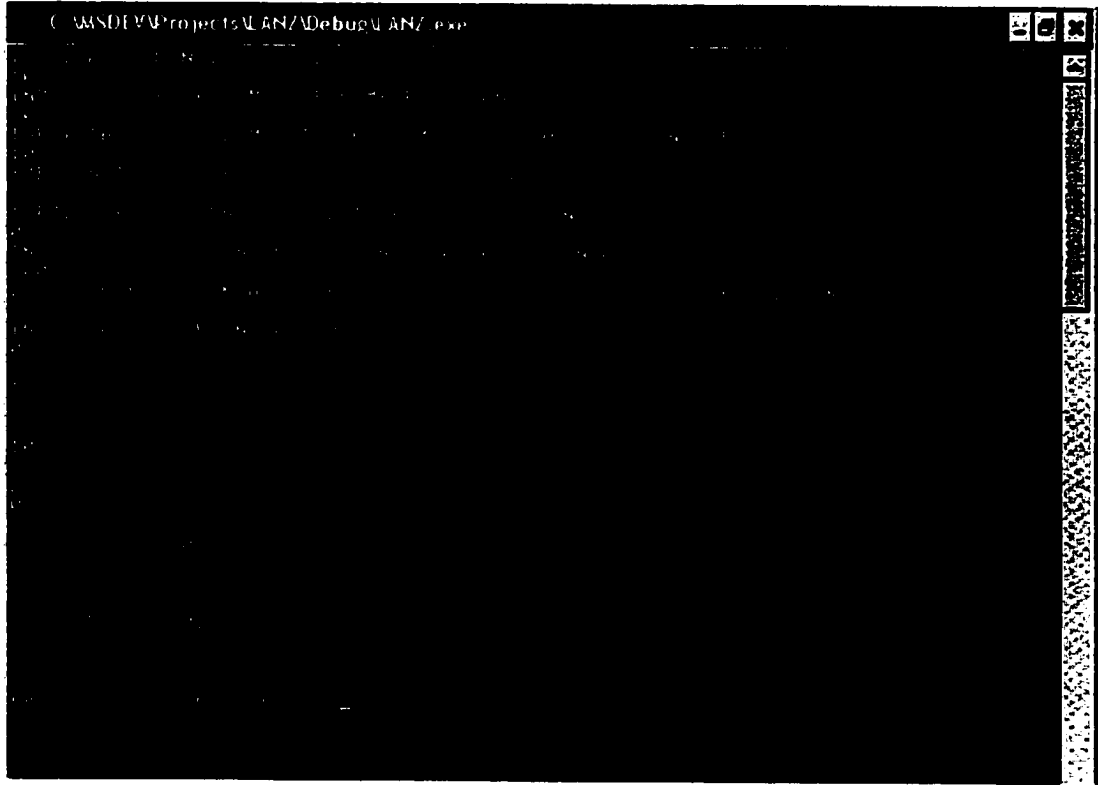
The dimension of concerned problem is 10 X 10.

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

And [M] is a identical matrix with 10 X 10 dimension.

The first few eigenvalues of  $[K]\{\phi\} = \lambda[M]\{\phi\}$  can be calculated by Lanczos package program. The calculated eigenvalues are 0.08101405 and 0.3175191. Again, the results are acceptably accurate compared to the exact solution.





## REFERENCES

## REFERENCES

- [1] Bathe, Klaus-Jurgen. 1996. Finite Element Procedures in Engineering Analysis. Prentice-Hall.
- [2] Miller, Web and Wrathall Celia. 1980. Software for Roundoff Analysis of Matrix Algorithms. Academic Press.
- [3] Huges, Thomas J.R. 1987. The Finite Element Method-Linear Static and Dynamic Finite Element Analysis, Prentice-Hall, pp. 605-629.
- [4] Gourlay, A. R. 1973. Computational Methods for Matrix Eigenproblems, John Wiley.
- [5] Rice, John R. 1981. Matrix Computations and Mathematical software. McGraw-Hill.
- [6] Carey, F. Graham and Oden, Tinsley. 1984. Finite Elements-Computation Aspects. Volume III. Prentice-Hall.
- [7] Esfandiari, Ramin S. 1996. Applied Mathematics for Engineers, 2<sup>nd</sup> edition. McGraw-Hill-college custom series.
- [8] Schwar, James and Best, Chales. 1986 Applied FORTRAN for Engineering and Science. Science Research Associates.
- [9] Thomsom, William T. 1993. Theory of Vibration with Applications, 4<sup>th</sup> edition. Prentice-Hall.
- [10] Kraus, Allan D. 1987. Matrices for Engineer. Hemisphere.
- [11] Kerrigan, Jim. 1993. Migrating to FORTRAN 90. O'Reilly.
- [12] Alan Jennings. 1977. Matrix Computation for Engineer and Scientists. John Wiley.