NX Nastran

Numerical Methods User's Guide

Proprietary & Restricted Rights Notice

TAUCS Copyright and License

C O N T E N T S

NX Nastran Numerical Methods User's Guide

6
Real Symmetric Eigenvalue Analysis

7
Complex Eigenvalue Analysis

Preface

- About this Book

About this Book

NX Nastran is a general-purpose finite element program which solves a wide variety of engineering problems. This book is intended to help you choose among the different numerical methods and to tune these methods for optimal performance. This guide also provides information about the accuracy, time, and space requirements of these methods.

This edition covers the major numerical methods available in NX Nastran Version 5, including parallel eigenvalue analysis for use in high-performance normal modes analysis, frequency response, and optimization. Further details about configuring and running such jobs can be found in the NX Nastran Parallel Processing Guide.

Introduction

This guide is designed to assist you with method selection and time estimation for the most important numerical modules in NX Nastran. The guide is separated into seven chapters:

These topics are selected because they have the biggest impact on the performance of the software. To obtain the most accurate solutions, you should read this guide carefully. Some of the numerical solutions exhibit different characteristics with different problems. This guide provides you with tools and recommendations for how to select the best solution.

Using This Guide

This guide assumes that you are familiar with the basic structure of NX Nastran, as well as with methods of linear statics and normal modes. A first-time reader of this guide should read Chapter 1 to become familiar with the utility tools and functions. After that, you can move directly to the chapters containing the topic you're trying to apply and tune (see Chapters 2 through 7). Each chapter contains general time estimates and performance analysis information as well as resource estimation formulae for some of the methods described in the chapter.

Since this guide also discusses the theory of numerical methods, it is intended as a stand-alone document except for a few references to the *NX Nastran Quick Reference Guide*.

CHAPTER
1

Utility Tools and Functions

- Utility Tools
- System Cells
- Diagnostic (DIAG) Flags
- Matrix Trailers
- Kernel Functions
- Timing Constants
- Time Estimates
- Storage Requirements
- Performance Analysis
- Parallel Processing

1.1 Utility Tools

In this chapter the following utility tools are described:

- System cells
- DIAG flags
- Matrix trailers
- Kernel functions
- Timing constants

Since these utilities are of a general nature, they are used in the same way on different computers and solution sequences. They are also used to select certain numerical methods and request diagnostics information and timing data. For these reasons, the utility tools are overviewed here before any specific numerical method is discussed.

1.2 System Cells

One of the most important communication utilities in NX Nastran is the SYSTEM common block. Elements of this common block are called system cells. Some of the system cells have names associated with them. In those cases, the system cell can be referred to by this name (commonly called a keyword).

Performance Cells. Some of the system cells related to general performance and timing issues are

BUFFSIZE = SYSTEM(1)
HICORE = SYSTEM(57)
REAL = SYSTEM(81)
IORATE = SYSTEM(84)
BUFFPOOL = SYSTEM(119)

Method Cells. System cells directly related to some numerical methods are

SOLVE = SYSTEM(69) – mixed
MPYAD = SYSTEM(66) – binary
FBSOPT = SYSTEM(70) – decimal

Execution Cells. System cells related to execution types are

SHARED PARALLEL = SYSTEM(107) – mixed
SPARSE = SYSTEM(126) – mixed
DISTRIBUTED PARALLEL = SYSTEM(231) – decimal
USPARSE = SYSTEM(209) – decimal

The binary system cells are organized so that the options are selected by the decimal values of the powers of 2. This organization makes the selection of multiple options possible by adding up the specific option values. The decimal cells use integer numbers. The mixed cells use both decimal and binary values.

The following several system cells are related to machine and solution accuracy:

MCHEPSS = SYSTEM(102)
MCHEPSD = SYSTEM(103)
MCHINF = SYSTEM(100) on LP-64, SYSTEM(98) on ILP-64
MCHUFL = SYSTEM(99) on LP-64, SYSTEM(97) on ILP-64

where MCHEPSS and MCHEPD are the machine epsilons for single- and double-precision, respectively, MCHINF is the exponent of the machine infinity, and MCHUFL is the exponent of machine underflow.

Note that these system cells are crucial to proper numerical behavior; their values should never be changed by the user without a specific recommendation from UGS support.

Setting System Cells

The following are several ways a user can set a system cell to a certain value:

NASTRAN Entry
- NASTRAN SYSTEM (CELL) = value
- NASTRAN KEYWORD = value

DMAP Program
- PUTSYS (value, CELL)
- PARAM //'STSR'/value/ – CELL

The first pair of techniques is used on the NASTRAN entry, and the effect of these techniques is global to the run. The second pair of techniques is used for local settings and can be used anywhere in the DMAP program; PUTSYS is the recommended way.

To read the value of a system cell, use:

VARIABLE = GETSYS (TYPE, CELL)
or
VARIABLE = GETSYS (VARIABLE, CELL)

SPARSE and USPARSE Keywords. The setting of the SPARSE keyword (SYSTEM(126)) is detailed below:

Value	Meaning
1	Enable SPMPYAD T and NT
2	Deselect SPMPYAD NT
3	Force SPMPYAD NT
4	Deselect SPMPYAD T
5	Force SPMPYAD T
6	Deselect SMPMYAD T and NT
7	Force SPMPYAD T and NT
8	Force SPDCMP
16	Force SPFBS

Combinations of values are valid. For example, SPARSE = 24 invokes a sparse run, except for SPMPYAD.

In the table below, the following naming conventions are used:

SPMPYAD	SPARSE matrix multiply
SPDCMP	SPARSE decomposition (symmetric)

The default value for SPARSE is 25.

Another keyword (USPARSE = SYSTEM(209)) is used to control the unsymmetric sparse decomposition and FBS. By setting USPARSE = 0 (the default is 1, meaning on), the user can deselect sparse operation in the unsymmetric decomposition and forward-backward substitution (FBS).

Shared Memory Parallel Keyword. The SMP (or PARALLEL) keyword controls the shared memory (low level) parallel options of various numerical modules.

The setting of the SMP keyword (SYSTEM(107)) is as follows:

Value	Meaning
1 – 1023	No. of Processors
1024	Deselect FBS
2048	Deselect PDCOMP
4096	Deselect MPYAD
8192	Deselect MHOUS
16384	Unused
32768	Deselect READ
262144	Deselect SPDCMP
524288	Deselect SPFBS

Combinations are valid. For example, PARALLEL = 525314 means a parallel run with two CPUs, except with FBS methods.

Module Naming Conventions. In the table above, the following naming conventions are used:

FBS	Forward-backward substitution
PDCOMP	Parallel symmetric decomposition
MHOUS	Parallel modified Householder method
READ	Real eigenvalue module
SPFBS	Sparse FBS

MPYAD	Multiply-Add
SPDCMP	Sparse decomposition

Distributed Parallel Keyword. For distributed memory (high level) parallel processing, the DISTRIBUTED PARALLEL or DMP (SYSTEM (231)) keyword is used. In general, this keyword describes the number of subdivisions or subdomains (in geometry or frequency) used in the solution. Since the value of DMP in the distributed memory parallel execution of NX Nastran defines the number of parallel Nastran jobs spawned on the computer or over the network, its value may not be modified locally in some numerical modules.

1.3 Diagnostic (DIAG) Flags

To request internal diagnostics information from NX Nastran, you can use DIAG flags. The DIAG statement is an Executive Control statement.

DIAG Flags for Numerical Methods. The DIAG flags used in the numerical and performance areas are:

DIAG	8	Print matrix trailers
	12	Diagnostics from complex eigenvalue analysis
	13	Open core length
	16	Diagnostics from real eigenvalue analysis
	19	FBS and Multiply-Add time estimates
	58	Print timing data

For other DIAG flags and solution sequence numbers, see the "Executive Control Statements" in the *NX Nastran Quick Reference Guide*.

Always use DIAG 8, as it helps to trace the evolution of the matrices throughout the NX Nastran run, culminating in the final matrices given to the numerical solution modules.

The module-related DIAGs 12, 16, 19 are useful depending on the particular solution sequence; for example, DIAG 12 for SOL 107 and 111, DIAG 16 for SOL 103, and DIAG 19 for SOL 200 jobs.

DIAG 58 is to be used only at the time of installation and it helps the performance timing of large jobs.

1.4 Matrix Trailers

The matrix trailer is an information record following (i.e., trailing) a matrix containing the main characteristics of the matrix.

Matrix Trailer Content. The matrix trailer of every matrix created during an NX Nastran run is printed by requesting DIAG 8. The format of the basic trailer is as follows:

- Name of matrix
- Number of columns: (COLS)
- Number of rows: (ROWS)
- Matrix form (F)

  = 1 square matrix

  = 2 rectangular

  = 3 diagonal

  = 4 lower triangular

  = 5 upper triangular

  = 6 symmetric

  = 8 identity matrix

  = 10 Cholesky factor

  = 11 partial lower triangular factor

  = 13 sparse symmetric factor

  = 14 sparse Cholesky factor

  = 15 sparse unsymmetric factor
- Matrix type (T)

  = 1 for real, single precision

  = 2 for real, double precision

  = 3 for for complex, single precision

  = 4 for complex, double precision
- Number of nonzero words in the densest column: (NZWDS)
- Density (DENS)

  Calculated as:

$$\frac{\text{number of terms}}{\text{COLS} \bullet \text{ROWS}} \bullet 10{,}000$$

Trailer Extension. In addition, an extension of the trailer is available that contains the following information:

- Number of blocks needed to store the matrix (BLOCKS)
- Average string length (STRL)
- Number of strings in the matrix (NBRSTR)
- Three unused entries (BNDL, NBRBND, ROW1)
- Average bandwidth (BNDAVG)
- Maximum bandwidth (BNDMAX)
- Number of null columns (NULCOL)

This information is useful in making resource estimates. The terms in parentheses match the notation used in the DIAG8 printout of the .f04 file.

The matrices of NX Nastran were previously stored as follows:

The matrix header record was followed by column records and concluded with a trailer record. The columns contained a series of string headers, numerical terms of the string and optionally a string trailer. The strings are consecutive nonzero terms. While this format was not storing zero terms, a must in finite element analysis, it had the disadvantage of storing topological integer information together with numerical real data.

Currently, the following indexed matrix storage scheme is used on most matrices:

Indexed Matrix Structure. An Indexed Matrix is made of three files, the Column, String and Numeric files.

Each file consists of only two GINO Logical Records:

- HEADER RECORD. For the Column file, it contains the Hollerith name of the data block (NAME) plus application defined words. For the String file, it contains the combined data block NAME and the Hollerith word STRING. For the Numeric file, it contains the combined data block NAME and the Hollerith word NUMERIC.
- DATA RECORD. It contains the Column data (see Indexed Matrix Column data Descriptions) for the Column file, the String data for the String file and the numerical terms following each other for the Numeric file.

Indexed Matrix File Structure

Column File	String File	Numeric File
Header Record 0 as written by application (Data block NAME + application defined words)	Data block NAME + "STRING"	Data block NAME + "NUMERIC"
Data Record *6\3 words per Column Entry **Word 1\first 1/2 of 1:** Column Number, negative if the column is null **Word 2\second 1/2 of 1:** Number of Strings in Column **Word 3 and 4\2:** String Relative Pointer to the first String of Column **Word 5 and 6\3:** Relative Pointer to the first Term of Column **Note:** If null column, then word(s) 3 to 4\2 points to the last non null column String Pointer, word(s) 5 to 6\3 points to the last non-null column Numeric Pointer	*2\1 word(s) per String Entry **Word 1\first 1/2 of 1:** Row number of first term in String **Word 2\second 1/2 of 1:** Number of terms in String	All matrix numerical terms following each other in one Logical GINO Record
*n1\n2 words, where • n1 is the number of words on short word machines • n2 is the number of words on long words machines		

1.5 Kernel Functions

The kernel functions are internal numerical and I/O routines commonly used by the functional modules.

Numerical Kernel Functions. To ensure good performance on a variety of computers, the numerical kernels used in NX Nastran are coded into regular and sparse kernels as well as single-level and double-level kernels. The regular (or vector) kernels execute basic linear algebraic operations in a dense vector mode. The sparse kernels deal with vectors described via indices. Double-level kernels are block (or multicolumn) versions of the regular or sparse kernels.

The AXPY kernel executes the following loop:

$$Y(i) = s \bullet X(i) + Y(i)$$

where:

$i = 1, 2, \ldots, n$

s = a scalar

n = the vector length

The sparse version (called AXPl) of this loop is

$$Y(INDEX(i)) = s \bullet X(i) + Y(INDEX(i))$$

where $i = 1, 2, \ldots, n$

In these kernels, X and Y are vectors. INDEX is an array of indices describing the sparsity pattern of the Y vector. A specific NX Nastran kernel used on many occasions is the block AXPY (called XPY2 in NX Nastran).

$$Y(i, j) = S(j) \bullet X(i, j) + Y(i, j)$$

where:

$i = 1, 2, \ldots, n$

$j = 1, 2, \ldots, b$

Here X, Y are blocks of vectors (rectangular matrices), S is an array of scalar multipliers, and b is the number of vectors in the block.

Similar kernels are created by executing a DOT product loop as follows:

$$DOT: \ Sum = \sum_{i=1}^{n} X(i) \bullet Y(i)$$

$$DOT1:\ Sum = \sum_{i=1}^{n} X(i) \bullet Y(i)$$

$$DOT2:\ Sum(j) = \sum_{i=1}^{n} X(i, j) \bullet Y(i, j)$$

where:

b = the block size

$j = 1, 2, \ldots, b$

Indexed versions of the XPY2 and DOT2 kernels also exist.

To increase computational granularity and performance on hierarchic (cache) memory architectures, the heavily used triangular matrix update kernels are organized in a triple loop fashion.

The DFMQ kernel executes the following mathematics:

$$A = A + uv^T$$

where A is a triangular or trapezoidal matrix (a portion of the factor matrix) and u, v are vectors.

The DFMR kernel executes a high rank update of the form

$$A = A + UV^T$$

where now U and V are rectangular matrices. All real, complex, symmetric, and unsymmetric variations of these kernels exist, but their description requires details beyond the scope of this document.

Triple Loop Kernels. Additional triple loop kernels are the triple DOT (DOT3) and SAXPY (XPY3) routines. They are essentially executing matrix-matrix operations. They are also very efficient on cache-machines as well as very amenable to parallel execution.

I/O Kernel Functions. Another category of kernels contains the I/O kernels. The routines in this category are invoked when a data move is requested from the memory to the I/O buffers.

Support Kernels. Additional support kernels frequently used in numerical modules are ZEROC, which zeroes out a vector; MOVEW, which moves one vector into another; and SORTC, which sorts the elements of a vector into the user-requested (ascending or descending) order.

1.6 Timing Constants

Single Loop Kernel Performance. Timing constants are unit (per term) execution times of numerical and I/O kernels. A typical numerical kernel vector performance curve shows unit time T as a function of the loop length. A loop is a structure that executes a series of similar operations. The number of repetitions is called the loop length.

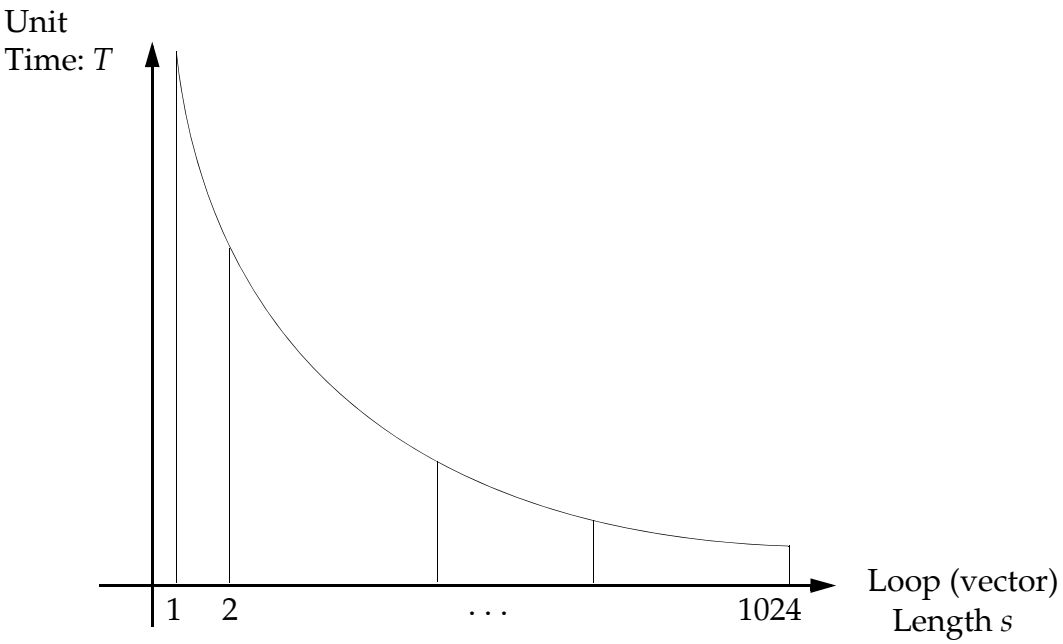


Figure 1-1 Single-Level Kernel Performance Curve

The kernel performance curve can be described mathematically as

$$T = A + \frac{B}{s} \qquad \text{Eq. 1-1}$$

where the constant A is characteristic of the asymptotic performance of the curve since

$$T(s \rightarrow \infty) \rightarrow A \qquad \text{Eq. 1-2}$$

The constant B represents the startup overhead of the loop as

$$T(s = 1) = A + B \qquad \text{Eq. 1-3}$$

These constants for all the NX Nastran numerical kernels can be printed by using DIAG 58.

Sometimes it is impossible to have a good fit for the datapoints given by only one curve. In these cases, two or more segments are provided up to a certain break point in the following format:

X Segments for Kernel Y	
Segment 1	Segment 2
Break Point	Break Point
A1	A2
B1	B2

where X is the number of segments and Y is the name of the particular kernel.

Double Loop Kernel Performance. In the case of the double loop kernels, the unit time is a function of both the inner loop length and the number of columns, which is the outer loop length. The unit time is described by a surface as shown in **Figure 1-2**.

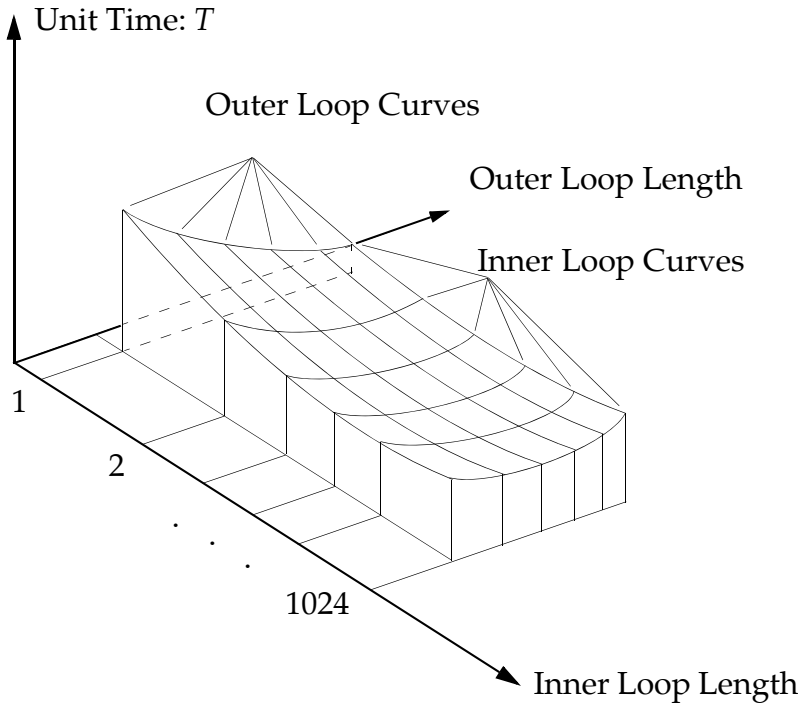


Figure 1-2 Double-Level Kernel Performance Surface

The surface on **Figure 1-2** is built from curves obtained by fixing a certain outer loop length and varying the inner loop length. Intermediate values are found by interpolation. Another set of curves is obtained by fixing the inner loop length and varying the outer loop length.

I/O Kernels. There are also many I/O kernels in NX Nastran.

The unit time for these kernels for string or term operations is

$$T_s = \textit{number of strings} \bullet A + \textit{number of nonzeroes} \bullet B \qquad \text{Eq. 1-4}$$

For column operations (PACK, UNPACK),

$$T_c = T_s + \text{rows} \bullet \textit{columns} \bullet A \qquad \text{Eq. 1-5}$$

and the two values are given for real and complex A and B values.

Triple Loop Kernels. The triple loop kernels are now included in the time estimate (GENTIM) process of NX Nastran.

While difficult to show diagramatically, the timing model for the triple loop kernels can be thought of as families of double loop surfaces as shown in **Figure 1-2**. A family is generated for specified lengths of the outermost loop. Values intermediate to these specified lengths are determined by interpolation.

Many of the numerical kernels are standard BLAS/LAPACK library routines, such as the AXPY kernels (described earlier) and the generalized matrix-multiply GEMM kernels. On certain platforms, vendor specific non-BLAS library routines are used as well. The speed and accuracy of these kernels has a large effect on numerical performance and stability. Therefore, NX Nastran may be linked against external libraries for best performance. Which external libraries are required will vary across hardware platforms, operating systems, and NX Nastran versions. The correct versions of all external libraries must be installed as part of the NX Nastran installation procedure.

1.7 Time Estimates

Calculating time estimates for a numerical operation in NX Nastran is based on analytical and empirical data. The analytical data is an operation count that is typically the number of multiply (add) operations required to execute the operation. In some cases the number of data movements is counted also.

The empirical data is the unit time for a specific kernel function, which is taken from the timing tables obtained by DIAG 58 and explained in "**Timing Constants**" on page 13. These tables are generated on the particular computer on which the program is installed and stored in the database.

The operation count and the execution kernel length are calculated using information contained in the matrix trailers. Sometimes trailer information from the output matrix generated by the particular operation is required in advance. This information is impossible to obtain without executing the operation. The parameters are then estimated in such cases, resulting in less reliable time estimates.

Available Time. Time estimates in most numerical modules are also compared with the available time (TIME entry). Operations are not started or continued if insufficient time is available.

I/O time estimates are based on the amount of data moved (an analytical data item) divided by the IORATE and multiplied by the I/O kernel time. Since the user can overwrite the default value of the IORATE parameter, it is possible to increase or decrease the I/O time calculated, which also results in varying the method selection.

In most numerical modules, NX Nastran offers more than one solution method. You can select the method used. The method automatically selected by NX Nastran is based on time estimates. The estimated (CPU) execution time is calculated by multiplying the number of numerical operations by the unit execution time of the numerical kernel executing the particular operation. In addition, an estimation is given for the (I/O) time required to move information between the memory and secondary storage. After the estimates for the CPU execution time and the I/O time are added together, NX Nastran selects the method that uses the least time.

Matrix Methods. Several methods are offered because each of them is best suited to certain types of matrices. The difference in cost among the methods for specific cases can be an order of magnitude or more. As each matrix is generated, the parameters describing its size and the distribution of nonzero terms are stored in a matrix trailer. (The parameters that define the properties of the matrices were described in "**Matrix Trailers**" on page 8.) For each matrix, these parameters include the number of rows and columns, the form (for example, square or

symmetric), the type (for example, real or complex), the largest number of nonzero words in any column, and the density. Some of the newer methods also record the number of strings in the matrix. Other descriptive parameters may be added in the future.

The only empirical data used in deriving the timing equations is the measurement of the time per operation for the kernels. These measurements are computed at the time of installation on each computer and are stored in the delivery database for later use. After the system is installed, the measurements may be updated if faster hardware options are installed on the computer. The remaining terms in the equations are derived from careful operation counts, which account for both arithmetic and data storage operations.

Timing Equations. Timing equations are derived for all major numerical modules. Conservative upper bounds are the best estimates that can be calculated. At present, these estimates are not used for method selection. Instead, the user is required to input the total amount of available CPU time to solve the total run. The amount of time remaining at the start of the numerical solution modules is compared with the estimate. The run is terminated before the numerical module starts execution if the amount of time remaining is less than the estimate. The goal is to minimize wasted computer resources by terminating expensive operations before they start, rather than terminating them midway before any output is available.

The many types of machine architecture which NX Nastran supports and the great differences in operation between scalar, vector, and parallel computing operations result in a challenge to the numerical method developers to provide correct estimation and method selection. There are a number of diagnostic tools which can be used to print out the estimates and the other parameters affecting computation cost. These tools are generally activated by the DIAG flags described earlier.

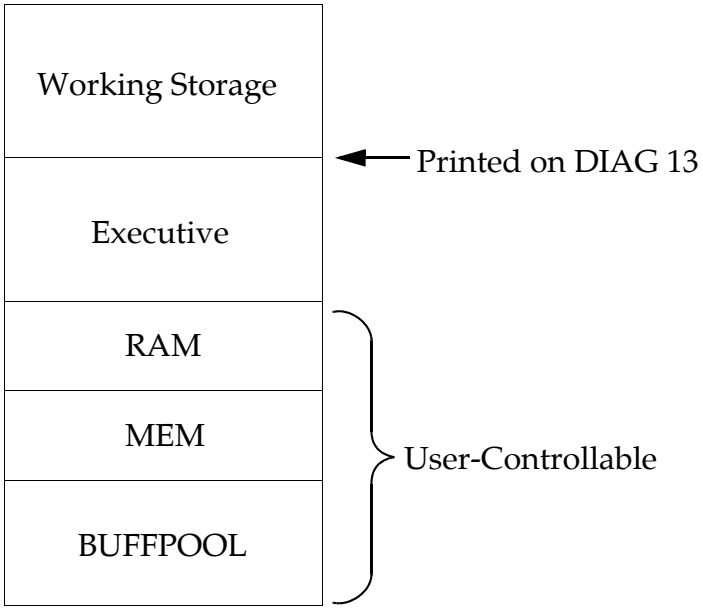
1.8 Storage Requirements

Main storage in NX Nastran is composed of the space used for the code, the space used for the Executive System, and the actual working space used for numerical operations.

Working Space. The actual working space available for a numerical operation can be obtained using DIAG 13.

Disk storage is needed during the execution of an NX Nastran job to store temporary (scratch) files as well as the permanent files containing the solution.

Memory Sections. The Executive System provides the tools needed to optimize the execution using a trade-off between memory and disk usage. The main memory is organized as follows:



RAM, MEM, BUFFPOOL. The RAM area holds database files, while the MEM area holds scratch files. The BUFFPOOL area can act as a buffer memory. The user-selectable sizes of these areas have an effect on the size of the working storage and provide a tool for tuning the performance of an NX Nastran job by finding the best ratios.

A general (module-independent) user fatal message associated with storage requirements is:

UFM 3008:
INSUFFICIENT CORE FOR MODULE XXXX

This message is self explanatory and is typically supported by messages from the module prior to message 3008.

1.9 Performance Analysis

.f04 Event Statistics. The analysis of the performance of an NX Nastran run is performed using the .f04 file.

Disk Usage. The final segment of the .f04 file is the database usage statistics. The part of this output most relevant to numerical modules is the scratch space usage (the numerical modules are large consumers of scratch space). SCR300 is the internal scratch space used during a numerical operation and is released after its completion. The specific SCR300 table shows the largest consumer of internal scratch space, which is usually one of the numerical modules. The output HIWATER BLOCK shows the maximum secondary storage requirement during the execution of that module.

Memory Usage. Another table in this final segment shows the largest memory usage in the run. The output HIWATER MEMORY shows the maximum memory requirement combining working storage and executive system areas, described in "Storage Requirements" on page 18.

1.10 Parallel Processing

Parallel processing in NX Nastran numerical modules is a very specific tool. It is very important in enhancing performance, although its possibilities in NX Nastran and in specific numerical modules are theoretically limited.

The parallelization possibilities in NX Nastran consist of three different categories:

- High level

  Frequency domain

- Medium level

  Geometry domain

- Low level

  Block kernels (high rank updates)

The currently available methods of parallel processing in NX Nastran numerical modules are:

- Shared memory parallel

  Medium, low level

  MPYAD, DCMP, FBS modules

- Distributed memory parallel

  High, medium level

  SOLVIT, DCMP, FBS, READ modules

Details of the various parallel methods are shown in the appropriate Modules' sections throughout.

CHAPTER
2

Matrix Multiply-Add Module

- Multiply-Add Equation
- Theory of Matrix Multiplication
- MPYAD Methods
- DMAP User Interface
- Method Selection/Deselection
- Option Selection
- Diagnostics
- MPYAD Estimates and Requirements

2.1 Multiply-Add Equation

The matrix multiply-add operation is conceptually simple. However, the wide variety of matrix characteristics and type combinations require a multitude of methods.

The matrix multiply-add modules (MPYAD and SMPYAD) evaluate the following matrix equations:

(MPYAD)

$$[D] = \pm[A]^{(T)}[B] \pm [C] \qquad \text{Eq. 2-1}$$

or

(SMPYAD)

$$[G] = [A]^{(T)}[B]^{(T)}[C]^{T}[D]^{T}E \pm F \qquad \text{Eq. 2-2}$$

The matrices must be compatible with respect to the rules of matrix multiplication. The (T) stands for (optional) transpose. The signs of the matrices are also user parameters. In **Eq. 2-2**, any number (between 2 and 5) of input matrices can be present.

The detailed theory of matrix multiply-add operation is described in "**Theory of Matrix Multiplication**" on page 23. Subsequent sections provide comprehensive discussions regarding the selection and use of the various methods.

2.2 Theory of Matrix Multiplication

The matrix multiplication module in NX Nastran evaluates the following matrix equations:

$$D = \pm[A][B] \pm [C] \qquad \text{Eq. 2-3}$$

or

$$D = \pm[A]^T[B] \pm [C]$$

where A, B, C, and D are compatible matrices. The calculation of **Eq. 2-3** is carried out by the following summation:

$$d_{ij} = \pm \sum_{k=1}^{n} a_{ik} b_{kj} \pm c_{ij} \qquad \text{Eq. 2-4}$$

where the elements a, b, c, and d are the elements of the corresponding matrices, and n is the column order of matrix A and the row order of matrix B. The sign of the matrices and the transpose flag are assigned by user-defined parameters.

NX Nastran has four major ways to execute **Eq. 2-3** and performs the selection among the different methods automatically. The selection is based on the density pattern of matrices A and B and the estimated time required for the different kernels.

These methods are able to handle any kind of input matrices (real, complex, single, or double precision) and provide the appropriate result type. Mixed cases are also allowed and are handled properly.

The effective execution of multiplication is accomplished by invoking the NX Nastran kernel functions.

The four methods are summarized in the following table and explained in more detail below.

Method	Combination
1	Dense × Dense
2	Sparse × Dense
3	Sparse × Sparse
4	Dense × Sparse

Method One (Dense x Dense)

Method one consists of several submethods. The submethod designated as method one storage 1 is also known as basic method one.

In basic method one, enough storage is allocated to hold as many non-null columns of matrices B and D as memory allows. The columns of matrix C corresponding to the non-null columns of B are initially read into the location of matrix D (the result). Matrix A is processed on a string-by-string basis. The complete multiplication operation may require more than one pass when all the non-null columns of matrices B and D cannot fit into memory. The number of passes can be calculated as follows:

$$N_p = \frac{N}{N_B} \qquad \text{Eq. 2-5}$$

where:

N = order of problem

N_p = number of passes

N_B = number of non-null columns of B in memory

The basic procedure of method one (storage 1) can be viewed as follows:

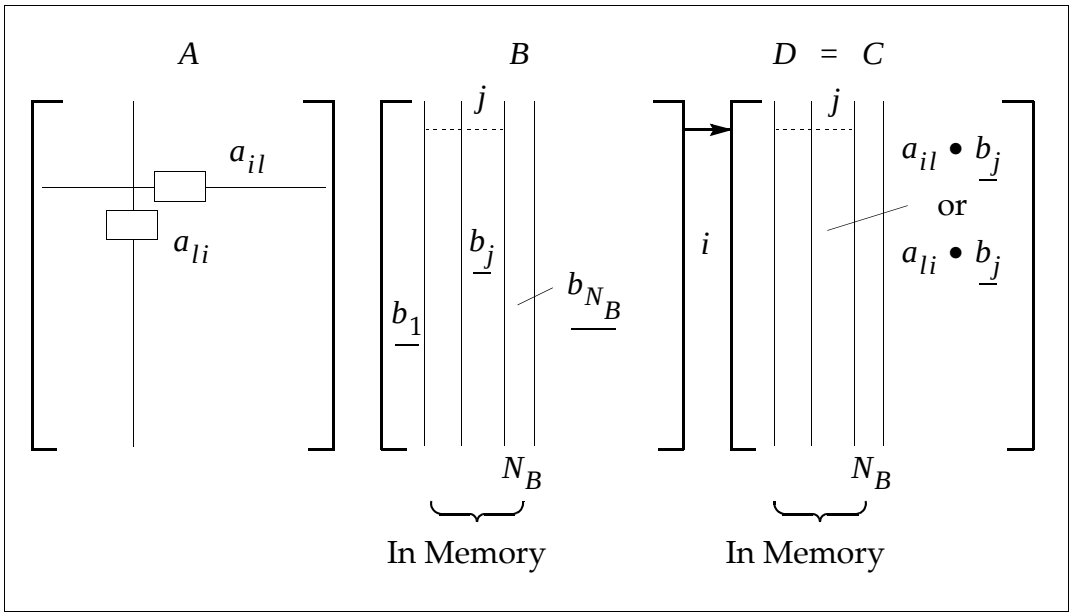


Figure 2-1 Method One

Note: The underlined quantities in Figure 2-1 represent vectors.

For each nonzero element of A, all corresponding terms of B currently in memory are multiplied and accumulated in D. N_B columns of matrix D are calculated at the end of one complete pass of matrix A through the processor. Next, the completed columns of D are packed out, along with any non-null columns of C that correspond to null columns of B skipped in this pass (as they are columns of D). This part is saved, N_B non-null columns of B and the corresponding columns of C are loaded, and the process continues. The effective execution of the multiplication depends on whether or not the transpose flag is set.

Nontranspose:

$$d_{ij} \leftarrow a_{il}\ b_{lj} + d_{ij} \qquad \text{Eq. 2-6}$$

Transpose:

$$d_{ij} \leftarrow a_{li}\ b_{lj} + d_{ij} \qquad \text{Eq. 2-7}$$

The other submethods provide for different handling of matrix A and for carrying out the multiplication operations. The main features of the submethods vary depending on the different ways of handling the strings (series of consecutive nonzero terms in a matrix column). A comparison of the main method and the submethods is shown as follows:

Table 2-1 Nontranspose Cases

Storage A:	Unpacked columns of B and D Processing A string by string A is in the inner loop
Storage B:	Unpacked columns of B and D Processing A string by string A is in the outer loop
Storage C:	Unpacked partial rows of B Processing A string by string A is in the inner loop
Storage D:	Partial rows of B in string format Processing A string by string A is in the outer loop
Storage E:	Unpacked columns of B and D Unpacked columns of A (band only) A is in the outer loop
Storage F:	Partial rows of B in string format Unpacked columns of A A is in the outer loop

Storage 2:	Unpacked non-null columns of B and D Unpacked columns of A Loops are the same as in storage 1, except that the outermost loop is pulled inside the kernel (triple loop kernel)
Storage 3:	Unpacked non-null columns of B and D Unpacked columns of A Loops are the same as in storage 1, except that the outermost loop is pulled inside the kernel (BLAS level 3)

Table 2-2 Transpose Cases

Storage A:	Unpacked columns of B and D Processing A string by string A is in the inner loop
Storage B:	Unpacked columns of B Partial rows of D Processing A string by string A is in the outer loop
Storage C:	Unpacked columns of B and D Unpacked rows of A (band only) A is in the outer loop
Storage D:	Unpacked columns of B Partial rows of D Unpacked rows of A A is in the outer loop
Storage 2:	Unpacked non-null columns of B and D Unpacked columns of A Loops are the same as in storage 1, except that the outermost loop pulled inside the kernel (triple loop kernel)
Storage 3:	Unpacked non-null columns of B and D Unpacked columns of A Loops are the same as in storage 1, except that the outermost loop pulled inside the kernel (BLAS level 3)

The effective execution of the multiplication operation in method one subcases (and other methods with the exception of method three) is accomplished by involving the NX Nastran kernel functions. In method one submethods, except for storage 2, the double loop kernels of DOT2 and XPY2 are used. In storage 2, the triple loop kernels of DOT3 are used.

Depending on whether the length of the current string of A is N or M, the A string is in the inner loop or in the outer loop. This explains the comments: "A is in the inner loop" or "A is in the outer loop" in **Table 2-1**. The selection between the previous two possible usages of the kernel functions depends on the density of matrices A and B. If A is sparse, it is in the inner loop; otherwise, A is in the outer loop.

Method Two (Sparse x Dense)

In the nontranspose case of this method, a single term of B and one full column of the partially formed D are in the main memory. The remaining main memory is filled with as many columns of A as possible. These columns of matrix A are in string format. This method is effective when A is large and sparse; otherwise, too many passes of B are required. The number of passes in B is calculated from **Eq. 2-5**.

The method can be graphically represented as follows:

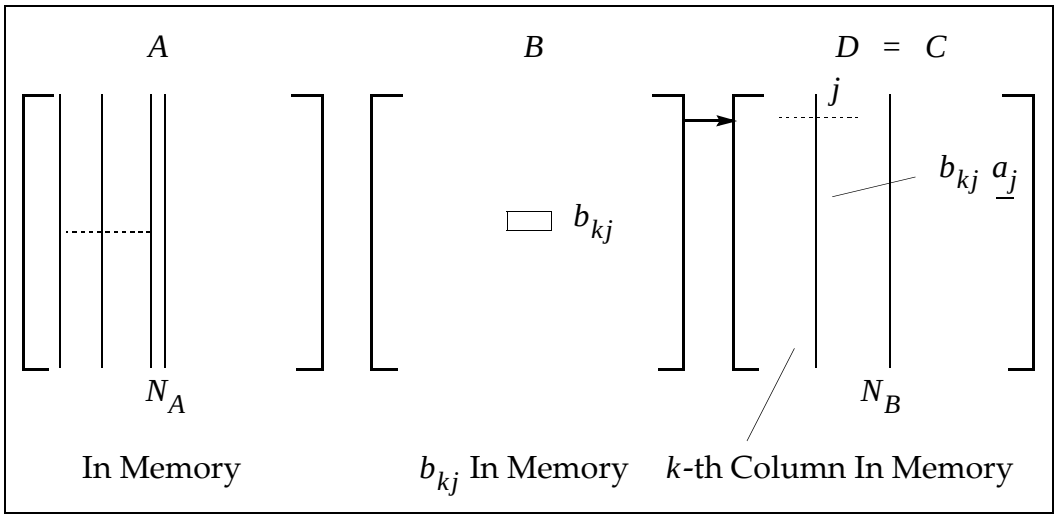


Figure 2-2 Method Two

When b_{kj} is in memory, the k-th column of A is processed against it and the result is accumulated into the k-th column of D. In the transpose case, one column of B is held in memory while D holds only a single term at a time. This method provides an alternative means of transposing matrix A by using the identity matrix B and the zero matrix C when the transpose module of NX Nastran is inefficient.

Method Three (Sparse x Sparse)

In method three, the transpose and nontranspose cases are essentially the same except for the initial transpose of matrix A in the transpose case. In both cases, matrix A is stored in the same way as in method two, i.e., in string format. Matrix B is processed on an element-by-element basis, and the products of each b_{kj} term are calculated using the corresponding terms of A in memory. However, in method three, the results and storage are different from method two. In method three, "storage bins" are established for the columns of matrix D. The number of these bins is based on the anticipated density of matrix D and is calculated as follows:

$$Number\ of\ bins = N\rho \qquad \text{Eq. 2-8}$$

where:

N = order of the problem

ρ = density of D (estimated)

The size of the bins is calculated as follows:

$$Size\ of\ bins = \frac{N}{Number\ of\ bins} = \frac{1}{\rho} \qquad \text{Eq. 2-9}$$

This manner of storing the results takes advantage of the sparsity of matrix B. However, it requires sorting in the bins before packing out the columns of D.

Method Four (Dense x Sparse)

This method has two distinct branches depending on the transpose flag.

Nontranspose Case

First, matrix B is transposed and written into the results file. This operation is performed with the assumption that B is sparse. As many columns of A as possible are unpacked into memory and the columns of B^T (rows of B) are interpreted on a term-by-term basis.

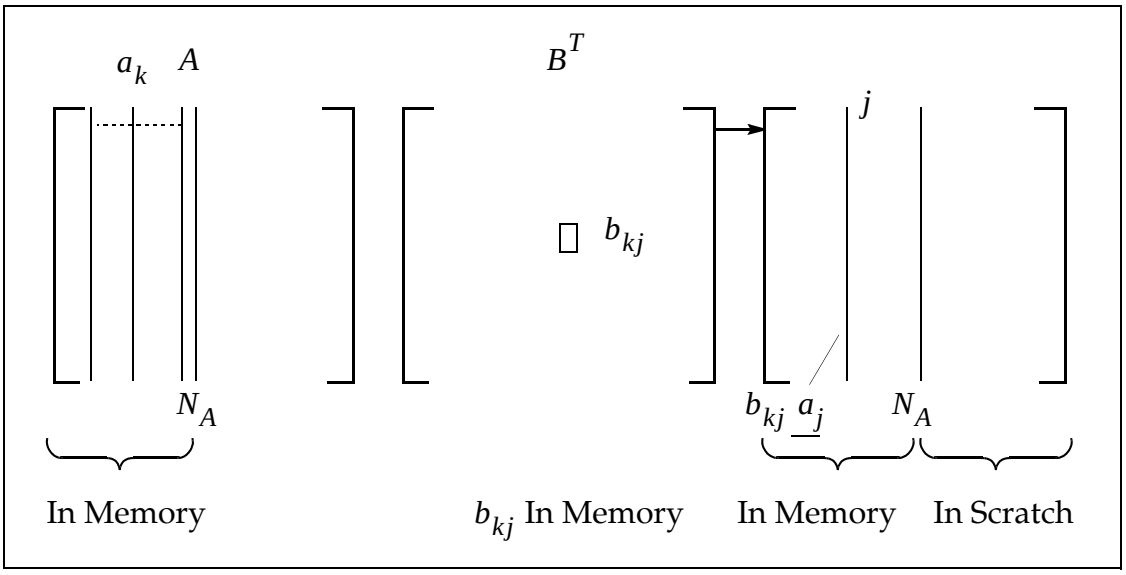


Figure 2-3 Nontranspose Method Four

For each nonzero term of B^T, the scalar product with the columns of A is formed and written into the scratch file. When all columns of A and rows of B are processed, the scratch file contains one column for each nonzero term in B. Therefore, a final pass must be made to generate matrix D.

Transpose Case

In this case, a set of rows of A^T (columns of A) are stored in unpacked form. These rows build a segment. Matrix B is processed on a string-by-string basis, providing the dot products of the strings and the columns of the segments.

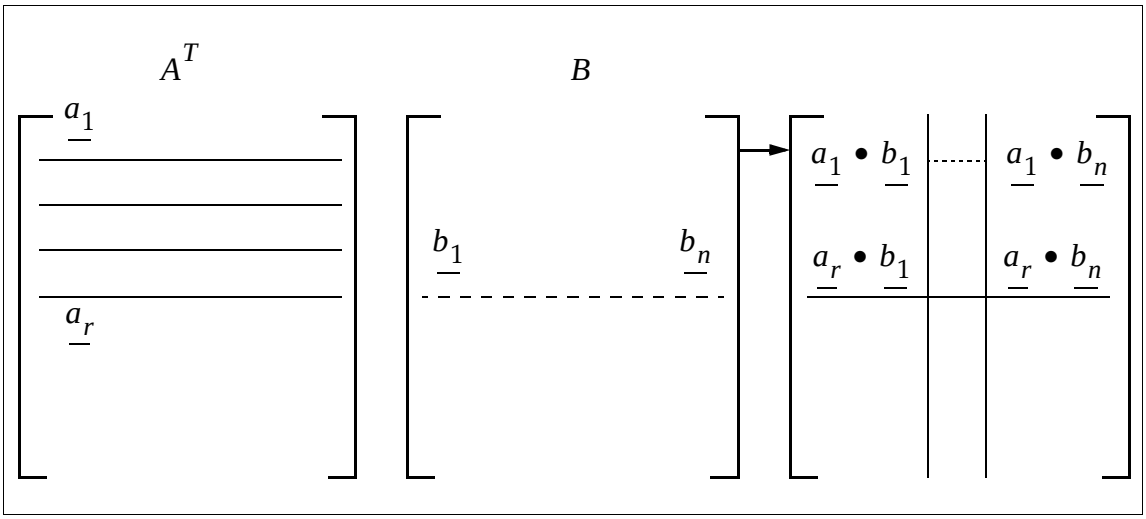


Figure 2-4 Transpose Method Four

The results are sequentially written into the scratch file continuously. The structure of this file is as follows:

CoI 1	CoI 2	...	CoI n	CoI 1	...	CoI n EOF
Seg 1	Seg 1		Seg 1	Seg 2		Seg n

The number of segments is (see **Figure 2-4**):

$$k = \frac{n}{r} \qquad \text{Eq. 2-10}$$

Finally, the product matrix must be assembled from the scratch file, and matrix C, if present, must be added.

Sparse Method

The sparse multiply method is similar to regular method three. When the transpose case is requested, matrix A is transposed prior to the numerical operations. This step is typically not very expensive since A is sparse when this method is selected.

The significance of this method is that matrix A is stored in a new sparse form. Specifically, all nonzero terms of a column are stored in a contiguous real memory region, and the row indices are in a separate integer array. The sparse kernel AXPI is used for the numerical work. The scheme of this method is shown on the following figure.

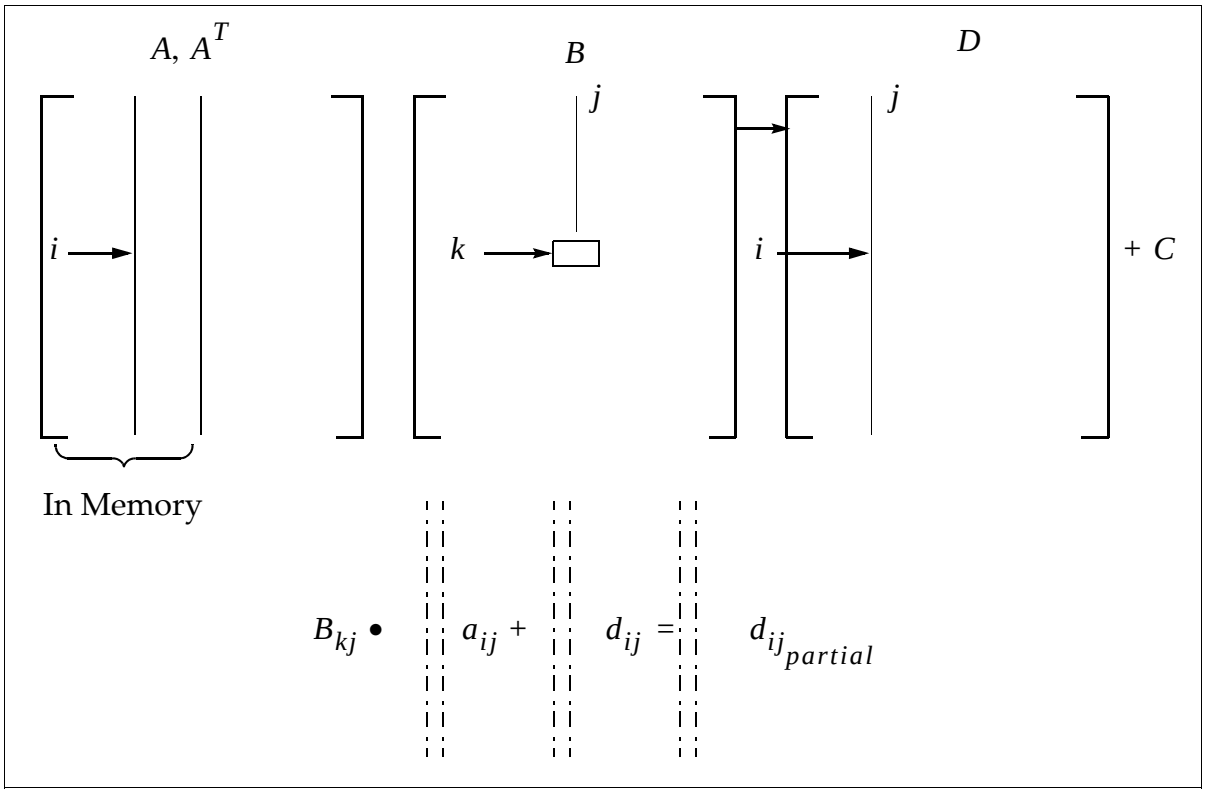


Figure 2-5 Sparse Method

From the figure, it is clear that matrix B is processed on an element-by-element basis. When all the nonzero terms of A do not fit into memory at once, multiple passes are executed on matrix B, and only partial results are obtained in each pass. These results are summed up in a final pass.

The sparse method is advantageous when both matrices are sparse.

Triple Multiply Method

In NX Nastran a triple multiplication operation involving only two matrices occurs in several places. The most common is the modal mass matrix calculation of $\phi^T M \phi$. Note that in this multiplication operation the matrix in the middle is symmetric. Therefore, the result is also symmetric. No symmetry or squareness is required for the matrices on the sides. Historically, this operation was performed by two consecutive matrix multiplications which did not take advantage of the symmetry or the fact that the first and third matrices are the same.

The operation in matrix form is

$$C = [A]^T[B][A] \pm D \qquad \text{Eq. 2-11}$$

where:

A = order of $n \times m$ (n rows $\times$ m columns)

B = $n \times n$ symmetric matrix

C = $m \times m$ symmetric matrix

D = $m \times m$ symmetric matrix

Any element c_{lk} of matrix C can be calculated as follows:

$$c_{lk} = \sum_{j=1}^{n} \left(a_{jk} \sum_{i=1}^{n} a_{il}\, b_{ij} \right) + d_{lk} \qquad \text{Eq. 2-12}$$

where:

l = the row index, $1 \le l \le m$

k = the column index, $1 \le k \le m$

It can be proven by symmetry that

$$\sum_{j=1}^{n} \left(a_{jl} \sum_{i=1}^{n} a_{ik}\, b_{ij} \right) = \sum_{j=1}^{n} \left(a_{jk} \sum_{i=1}^{n} a_{il}\, b_{ij} \right) \qquad \text{Eq. 2-13}$$

Based on the equality in **Eq. 2-13**, a significant amount of work can be saved by calculating only one-half of matrix C.

When designing the storage requirements, advantage is taken of the fact that matrix B is only needed once to calculate the internal sums. Based on this observation, it is not necessary to have this matrix in the main memory. Matrix B can be transferred through the main memory using only one string at a time.

The main memory is equally distributed among A^T, $A^T B$, and three vector buffers. One of the vector buffers must be a full column in length. Therefore, the structure of the main memory is as follows:

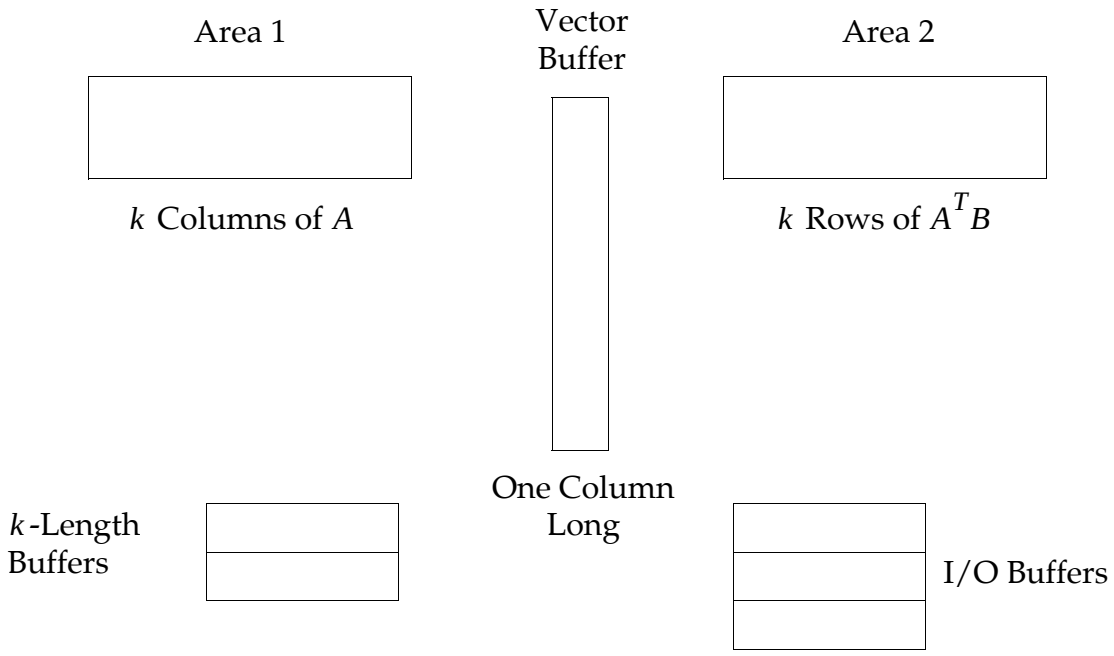


Figure 2-6 Memory Distribution for Triple Multiply

Three I/O buffers must be reserved for the three simultaneously open files of A, B, and the scratch file containing partial results. Therefore, the final main memory size is

$$n_z = 2k \bullet n + n + 3 \bullet \text{BUFFSIZE} + 2k \qquad \text{Eq. 2-14}$$

From **Eq. 2-14**, the number of A columns fitting into memory can be calculated as follows:

$$k = \frac{n_z - 3 \bullet \text{BUFFSIZE} - n}{2n + 2} \qquad \text{Eq. 2-15}$$

The number of passes is calculated as follows:

$$p = \begin{cases} \frac{m}{k} \text{ if } \frac{m}{k} = \text{integer} \\ \text{int}\left(\frac{m}{k} + 1\right) \text{ if } \frac{m}{k} \neq \text{integer} \end{cases} \qquad \text{Eq. 2-16}$$

which is equal to the number of times the triple multiply operation reads through the matrix B.

The number of times the triple multiply operation reads through the matrix A can be approximated as follows:

$$p_A = \frac{p}{2} \quad \text{Eq. 2-17}$$

The triple multiply method is implemented with a block spill logic where the result of the matrix C is generated in $k \times k$ blocks.

The above triple multiply is called the triple multiply with sparse middle matrix. The triple multiply with dense middle matrix is introduced below.

When both matrices A and B are dense, it is more efficient to apply dense multiply with BLAS, which is applied in the triple multiply with dense middle matrix. For the triple multiply with dense middle matrix, the main memory is equally distributed among A, B, $A^T B$, two vector buffers and three (or four if matrix D exists) I/O buffers. The stucture of main memory is similar to Figure 2-6, except that the vector buffer for B is replaced by k columns of B. The final main memory size is

$$n_z = 3k \bullet n + 3 \bullet \text{BUFFSIZE} + 2k \quad \text{Eq. 2-18}$$

and the number of A columns fitting into memory can be calculated as follows:

$$k = \frac{n_z - 3 \bullet \text{BUFFSIZE}}{3n + 2} \quad \text{Eq. 2-19}$$

The estimation of the numbers of passes with the triple multiply with dense middle matrix operation through the matrices A and B is the same as that of the triple multiply with sparse middle matrix.

Parallel Multiply Method

The parallel multiply method, which is only used when parallel processing is requested, is basically a parallel version of method one designed to solve the CPU-intensive multiplication of dense matrices.

The storage structure of this method is the same as that of method one. However, the columns of matrix B are distributed among the processors. Consequently, although B is stored in the main (shared) memory, the processors access different portions of it.

At the beginning of each pass, subtasks are created. These subtasks wait for the columns of matrix A to be brought into memory. Once a column of A is in the main memory, all subtasks process this column with their portion of matrix B. When all of the subtasks are finished, the main processor brings in a new column of A and the process continues. The parallel method is advantageous when matrix B is very dense and multiple processors are available.

Algorithms 1TC and 1NTE (see "MPYAD Methods" on page 36) are executed in parallel when parallel MPYAD is requested. The parallel methods are represented in Figure 2-7.

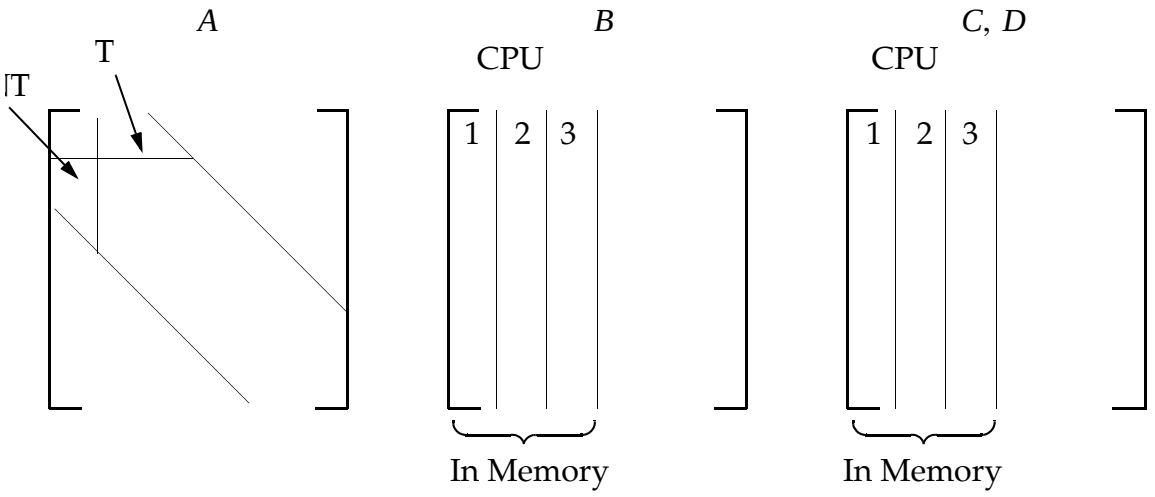


Figure 2-7 Parallel Multiply Method

The triple loop kernels used in Method 1 Storage 2 are also parallelized by some machine vendors providing another way to execute parallel MPYAD.

2.3 MPYAD Methods

The methods in the MPYAD module are divided into six main categories: four methods for different density combinations, one method for parallel execution, and one method for sparse operations. These methods are summarized in the following table:

Method	Combination
1	Dense × Dense
2	Sparse × Dense
3	Sparse × Sparse
4	Dense × Sparse
P	Dense × Dense
S	Sparse × Sparse

There are no fixed density boundaries set for these methods. Method selection is a complex topic. Within method one, there are also ten submethods for the special handling of the matrices.

Methods two and three are only selected in special cases. In most cases, the sparse method replaces both methods two and three.

The parallel multiply method is aimed at shared memory parallel computers. It does not run on distributed memory parallel computers.

The method 1 submethods (*A*-*F*) are automatically deselected in some cases. One example is when the A and B matrices are the same; another is when any of the matrices are non-machine precision.

MPYAD Method Identifiers. For selection and deselection purposes, identifiers are assigned to certain methods. However, these identifiers are bit oriented, and in some cases their decimal equivalents are used:

Method	Bit	Decimal
1NT	0	1
1T	1	2
2NT	2	4
2T	3	8
3NT	4	16

Method	Bit	Decimal
3T	5	32
4NT	6	64
4T	7	128
1NTA	8	256
1NTB	9	512
1NTC	10	1024
1NTD	11	2048
1NTE	12	4096
1NTF	13	8192
1TA	14	16384
1TB	15	32768
1TC	16	65536
1TD	17	131072
Deselect	20	1048576
DIAG	21	2097152
	22	4194304
1NT2	23	8388608
1T2	24	16777216
AutoS2	25	33554432
1NT3	26	67108864
1T3	27	134217728

In the above table, T represents transpose, NT indicates nontranspose, and A, B, C, D, E, F, 2 are submethod names when they appear as the last character. For example, 1NTD is a method one, nontranspose case, D submethod operation.

Bit 21 (with the corresponding decimal value of 2097152) is reserved for submethod diagnostics.

2.4 DMAP User Interface

The DMAP call for the MPYAD module executing the operation in **Eq. 2-1** is

```
MPYAD          A,B,C/D/T/SIGNAB/SIGNC/PREC/FORM
```

where:

T = 0,1: Non-transpose or transpose (see "**Multiply-Add Equation**" on page 22)

PREC = 0,1,2: Machine, single, double, etc. (see "**Matrix Trailers**" on page 8)

FORM = 0,1,2: Auto, square, rectangular, etc. (see "**Matrix Trailers**" on page 8)

An alternative MPYAD call is to use the SMPYAD module as follows

```
SMPYAD   A,B,C,D,E,F/G/N/SIGNG/SIGNF/PREC/TA/TB/TC/TD/FORM
```

where:

N = number of matrices given

TA,TB,TC,TD = transpose flags

FORM = as above

This module executes the operation in **Eq. 2-2**.

2.5 Method Selection/Deselection

MPYAD automatically selects the method with the lowest estimate of combined cpu and I/O time from a subset of the available methods. Also, those methods that are inappropriate for the user's specific problem are automatically deselected. The user can override the automatic selection and deselection process by manual selection, subject to certain limitations. The details of both automatic and user selection and deselection criteria are described below.

Automatic Selection

By default, methods 1 (all submethods), 3, 4, and Sparse are available for automatic selection. Methods 2 and P are excluded from automatic selection unless bit 25 of System Cell 66 has been set (decimal value 33554432) or all of the default methods have been deselected. Also, if all of the default methods have been deselected, method 2 will be used, provided it was not deselected. If all methods have been deselected, a fatal error occurs.

Automatic Deselection

If a method is determined to be inappropriate for the user's problem, it will be automatically deselected. Except in those cases noted below, an automatically deselected method will be unavailable for either automatic selection or manual user selection. Note that any method that has insufficient memory to execute the user's problem will be deselected. The other automatic deselection criteria are described below for each method. In this discussion "mpass*I*" stands for the number of passes required by method *I*. ρ_A and ρ_B represent the densities of the A matrix and B matrix, respectively. Also, unless the method name is qualified by NT (non-transpose) or T (transpose), the criteria applies to both.

Method 1 – All Submethods	If method S is not deselected and mpass1 is greater than $5 \times$ mpass3 and ρ_A and ρ_B are less than 10%. If MPYAD was called by the transpose module.
Method 1 – Storage A-F	If the type of matrix A is real and either B or C is complex. If the type of matrix A is complex and both B and C are real.
Method 1NT – Storage A, D, and F	Unless explicitly user selected.
Method 1T – Storage B and C	1TB unless 1TA deselected. 1TD unless 1TC is deselected.
Method 2	If method S is not deselected, unless user selected.

	If the C matrix is non-null and the type of C is not machine precision or the A, B, and C matrices are not all real or all complex.
	If MPYAD was called by the transpose module.
Method 3	If method S is not deselected, unless user selected.
	If matrix A is real and the B and/or C matrix is complex.
Method 3NT	If the requested type of the D matrix is real and any of the input matrices are complex, or if the requested type of D is complex and all of the input matrices are real.
Method 4	If methods 1, 2, 3, and S are not deselected and ρ_A greater than 10%, unless method 4 has been user selected.
	If matrix C is non-null and its type is not equal to machine precision or the B and C matrices are not both real or not both complex.
Method 4T	If more than 100 "front-end" (R4) passes or more than 10 "back-end" (S4) passes.
Method Sparse	If the type of matrix B is not machine precision.
	If matrix C is non-null and A and C are not both real or both complex.
	If matrix A is complex and matrix B is real.
Method Parallel NT	Unless 1NTE is not deselected.
	If the number of columns per pass for 1NTE is less than the number of available processors.
Method Parallel T	Unless 1TC is available.
	If the number of columns per pass for 1TC is less than the number of available processors.

User-Specified Deselection

For method deselection, the following is required:

Main Methods – Deselection

SYSTEM(66) = decimal value of method

Submethods – Deselection

SYSTEM(66) = decimal value of submethod

Sparse Method – Deselection

SYSTEM(126) = 0: Deselect all sparse methods
SYSTEM(126) = 2: Deselect sparse NT only
SYSTEM(126) = 4: Deselect sparse T only

where SYSTEM (126) is equivalent to the SPARSE keyword.

Parallel Method – Deselection

SYSTEM(107) = 0: Deselect all parallel modules
SYSTEM(107) = 4096 + ncpu: Deselect parallel MPYAD only

where ncpu = number of CPUs and SYSTEM (107) is equivalent to the PARALLEL. keyword.

The philosophy of method selection is to deselect all the methods except for the one being selected.

Triple Loop Method – Deselection

SYSTEM(252) = 0 or > 100: Do not use triple loops in 1T,1NT

User-Specified Selection

For method selection, the following is required:

Main Methods – Selection

SYSTEM(66) = 255 – decimal value of method identifier

Main or Submethods – Selection

SYSTEM(66) = 1048576 + bit value of method or submethod identifier

Triple Multiply Method – Selection

SYSTEM(129) = 0 : Default, automatic selection

SYSTEM(129) = 1 : Two Multiply, i.e. pre-MSC.Nastran Version 67 method

SYSTEM(129) = 2 : Triple multiply for sparse middle matrix

SYSTEM(129) = 3 : Triple multiply for dense middle matrix

Sparse Method – Selection

SYSTEM(126) = 1: Auto selection (This is the default.)
SYSTEM(126) = 3: Force sparse NT method
SYSTEM(126) = 5: Force sparse T method
SYSTEM(126) = 7: Force either T or NT sparse

Parallel Method – Selection

SYSTEM(107) > 0 and
SYSTEM(66) = 1048592(T) or 1048588(NT)

2.6 Option Selection

The following table shows the type combination options that are supported (R stands for real, C for complex). These options are automatically selected based on matrix trailer information. When the user selects one particular method with an option not supported with that method, an alternate method is chosen by MPYAD unless all of them are deselected.

Method	R • R + R	C • C + C	R • C + R	R • C + C
1T	YES	YES	YES	YES
1NT	YES	YES	YES	YES
2T	YES	YES	YES	YES
2NT	YES	YES	NO	NO
3T	YES	YES	NO	NO
3NT	YES	YES	NO	NO
4T	YES	YES	NO	YES
4NT	YES	YES	NO	NO
S	YES	YES	NO	NO
P	YES	YES	NO	NO

2.7 Diagnostics

The MPYAD module outputs diagnostic information in two categories: performance diagnostics and error diagnostics.

Performance Diagnostics

DIAG 19 Output. The following performance diagnostics is received by setting DIAG 19.

```
M   MATRIX A      Trailer(COLS ROWS FORM TYPE NZ DENS)     METHOD 1 Passes = XX CPU = XX I/O = XX Total = XX
P   MATRIX B      Trailer(COLS ROWS FORM TYPE NZ DENS)     METHOD 2 Passes = XX CPU = XX I/O = XX Total = XX
Y   MATRIX C      Trailer(COLS ROWS FORM TYPE NZ DENS)     METHOD 3 Passes = XX CPU = XX I/O = XX Total = XX
A   Working Memory = XX SYSTEM (66) = XX                   METHOD 4 Passes = XX CPU = XX I/O = XX Total = XX
D   Transpose Flag = XX SYSTEM (126) = XX                  METHOD S Passes = XX CPU = XX I/O = XX Total = XX
```

Figure 2-8 Excerpt from the DIAG19 Output

In the above figure, "passes" means the number of partitions needed to create the result matrix.

To prevent creating huge .f04 files when many MPYAD operations are executed, NX Nastran has a machine-dependent time limit stored in SYSTEM(20). When a time estimate is below this value, it is not printed. To print all time estimates, the user should set SYSTEM(20) = 0.

Most of the diagnostics information mentioned in the above table is self explanatory. Notice the presence of the MPYAD keyword (SYSTEM(66)) used to verify the method selection/deselection operation.

Whenever a method is deselected, its time estimate is set to 999999.

Submethod Diagnostics

For special diagnostics on the submethods, the user must add 2097152 to the value of SYSTEM(66) (i.e. turn on bit 21). The format of this diagnostic is shown in **Table 2-3**. The first column heading indicates the selected submethod, the DESELECT column contains either YES or NO for each submethod, and the last four columns contain the appropriate times.

Table 2-3 Method One Submethods

NEW1 = B	DESELECT	NCPP	PASSES	KERNEL	CPU	I/O	TOTAL
A	YES	x	x	x	x	x	x
B	NO	x	x	x	x	x	x
C	NO	x	x	x	x	x	x
D	YES	x	x	x	x	x	x

Table 2-3 Method One Submethods (continued)

NEW1 = B	DESELECT	NCPP	PASSES	KERNEL	CPU	I/O	TOTAL
E	YES	x	x	x	x	x	x
F	YES	x	x	x	x	x	x
1	YES	x	x	x	x	x	x
2	YES	x	x	x	x	x	x

where:

NCPP = number of columns per pass
NEW1 = B indicates that submethod B is chosen

Error Diagnostics

Error messages are abbreviated as follows:

UFM	User Fatal Message
SFM	System Fatal Message
UWM	User Warning Messages
SWM	System Warning Messages
UIM	User Information Messages
SIM	System Information Messages

The following error-related messages may be received from MPYAD:

UFM 3055:
AN ATTEMPT TO MULTIPLY NONCONFORMABLE MATRICES.

The message is given if the number of columns of A is not equal to the number of rows in B, the number of rows of C is not equal to the number of rows of A, or the number of columns of C is not equal to the number of columns of B. This message is also given when MPYAD is called from another module.

SFM 5423:
ATTEMPT TO MULTIPLY INCOMPATIBLEMATRICES.

The cause for this message is the same as for UFM 3055. However, this message is more elaborate and prints the trailers for all matrices involved. This message comes from the MPYAD module.

UFM 6199:
INSUFFICIENT CORE AVAILABLE FOR MATRIX MULTIPLY.

This message results while using the sparse multiply method when the storage estimate based on the trailer information is exceeded during the actual execution of the operation.

2.8 MPYAD Estimates and Requirements

The CPU time estimate for the sparse multiply-add method is based on the following input matrix characteristics:

$$[A] : m \bullet n, \quad [B] : n \bullet p, \quad [C] : m \bullet p, \quad \rho_A \qquad \text{Eq. 2-20}$$

Computation time (sec):

$$m \bullet n \bullet p \bullet \rho_A \bullet M \qquad \text{Eq. 2-21}$$

Data move time (sec):

$$n \bullet p \bullet \rho_p \bullet P \bullet n_{\text{pass}} + m \bullet n \bullet \rho_A \bullet P_* + (2)m \bullet p \bullet \rho_{C,\,(D)} \bullet P_* \qquad \text{Eq. 2-22}$$

ρ_* = density of $*$ matrix

P_* = one of either P_s, P, or P_i depending on the particular methods used

$$n_{\text{pass}} = \frac{m \bullet n \bullet \rho_A}{W/(\text{IPREC} + 1)}$$

W = workspace available in words

IPREC= machine precision (1 for short-word machines, 2 for long-word machines)

Note: M, P, and P_* are defined in the Glossary of Terms.

The minimum storage requirements are as follows:

Disk: $m \bullet n \bullet \rho_A + n \bullet p \bullet \rho_B + (2)m \bullet p \bullet \rho_D$

Memory: $2(n + m) \bullet \text{IPREC}$

CHAPTER 3

Matrix Decomposition

- Decomposition Process
- Theory of Decomposition
- User Interface
- Method Selection
- Option Selection
- Diagnostics
- Decomposition Estimates and Requirements
- References

3.1 Decomposition Process

The decomposition operation is the first step in solving large linear systems of equations.

For symmetric matrices:

$$[A] = [L][D][L]^T \quad \text{Eq. 3-1}$$

where:

$[A]$ = system matrix

$[L]$ = lower triangular factor

$[D]$ = diagonal matrix

or

$$[A] = [C][C]^T \quad \text{Eq. 3-2}$$

where:

$[A]$ = system matrix

$[C]$ = Cholesky factor

For unsymmetric matrices:

$$[A] = [L][U] \quad \text{Eq. 3-3}$$

where:

$[A]$ = system matrix

$[L]$ = lower triangular factor

$[U]$ = monic upper triangular factor

3.2 Theory of Decomposition

Symmetric Decomposition Method

The symmetric decomposition algorithm of NX Nastran is a sparse algorithm. This algorithm relies on fill-in reducing sequencing and sparse numerical kernels. The specific implementation also allows for the indefiniteness of the input matrix. This method is based on Duff, et al., 1982.

The factor has a specific storage scheme that can be interpreted only by the sparse FBS method.

Mathematical Algorithm

Permute and partition A as follows:

$$PAP^T = \begin{bmatrix} E & C^T \\ C & B \end{bmatrix} \quad \text{Eq. 3-4}$$

where the assumption is that the inverse of the s by s submatrix E exists. lf A is indefinite, appropriate pivoting is required to ensure the existence of the inverse. This requirement is fulfilled by the presence of the P permutation matrices in the above equation. The order of *E* is either 1 or 2. Then the elimination of E can be shown as

$$PAP^T = \begin{bmatrix} I_s & 0 \\ CE^{-1} & I_{n-1} \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B - CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I_s & E^{-1}C^T \\ 0 & I_{n-1} \end{bmatrix} \quad \text{Eq. 3-5}$$

Take $A_2 = B - CE^{-1}C^T$, permute and partition again to obtain the following:

$$PA_2P^T = \begin{bmatrix} E_2 & C_2^T \\ C_2 & B_2 \end{bmatrix} \quad \text{Eq. 3-6}$$

and continue the process until

$$O(B_k) = 1 \text{ or } 2$$

The final factored form of

$$PAP^T = LDL^T \quad \text{Eq. 3-7}$$

is then given by building the following:

$$L = \begin{bmatrix} I_1 & 0 & 0 & . & 0 \\ C_1 E_1^{-1} & I_2 & 0 & . & 0 \\ : & C_2\, E_2^{-1} & I_3 & . & 0 \\ : & : & C_3 E_3^{-1} & . & 0 \\ : & : & : & . & I_k \end{bmatrix} \quad \text{Eq. 3-8}$$

and

$$D = \begin{bmatrix} E_1 & 0 & 0 & . \\ 0 & E_2 & 0 & . \\ 0 & 0 & E_3 & . \\ 0 & 0 & 0 & B_k - C_k\, E_k^{-1} C_k^T \end{bmatrix} \quad \text{Eq. 3-9}$$

where D is built from 1 by 1 and 2 by 2 diagonal blocks. The I_k identity submatrices are also of order 1 or 2 and the submatrices are rectangular with one or two columns. The rows of the $C_i\, E_i^{-1}$ matrices extend to the bottom of the *L* matrix.

The most important step is the proper selection of the E partition. This issue is addressed later in this guide.

The module consists of two distinct phases: the symbolic phase and the numeric phase.

Symbolic Phase

This phase first reads the input matrix A and creates the following information: one vector of length NZ (where NZ is the number of nonzero terms of the upper half of the input matrix A) which contains the column indices, and another vector of the same length which contains the row indices of the nonzero terms of the upper triangular half of the matrix A. Both of these vectors contain integers. Another responsibility of this phase is to eliminate the zero rows and columns of the input matrix.

The selection of the E partition (i.e., the general elimination process) can be executed in a variety sequences. The performance of the elimination using different sequences is obviously different. To find an effective elimination sequence, a symbolic decomposition is also executed in the preface. An important criterion is to minimize the fill-in (off-diagonal nonzeros) created in each step of the elimination process, thereby reducing the numerical work and the I/O requirements.

The reordering methods may be prefaced by a compression process based on grid-dof connection or the so-called supernodal amalgamation principle (both available with any reordering). Both contribute by making the reordering more efficient in time and profile.

Several different reordering algorithms are available in the symbolic phase: multiple minimum degree algorithm (MMD), Metis, EXTREME, MLV (multilevel vertex partitioning), etc. These are described in the references.

Each of the methods can be selected by setting system cell 206 to the desired option; see "**User Interface**" on page 58 for details.

In each of the above methods, the elimination of a chosen variable is performed based on severing all the edges connected to it and connecting those nodes which were connected through the eliminated variable. Severing these edges leaves a reduced graph where the same process can continue until the reduced graph is a single node only. Then using this term as the root, create a so-called assembly tree of the matrix. The final elimination sequence is obtained by traversing the assembly tree. Note that this sequence may be changed due to numerical reasons.

Finally, the elimination sequence is stored into an array of length 3 • N (where N is the order of the A matrix). Note that at this phase, the actual terms of the matrix are not needed.

Numeric Phase

The mathematical decomposition process was described previously except for the details of the pivot selection for numerical stability. The strategy applied is a variant of the Bunch-Parlett method (1971) and is implemented as follows.

Let us assume that the next potential pivot row is the *j*-th. The diagonal entry of that row is tested against all the other terms k = (j + 1, …, n) as follows:

$$|a_{jj}| > t|a_{jk}|$$

where *t* is based on an input parameter. If the inequality is true, then the decomposition process uses *s* = 1 (1 by 1 pivot) and a_{jj} as the pivot term (E matrix). If the inequality is not true and the a_{jl} term is the largest in the pivot row, then the following pivotal matrix is tested for stability:

$$E_2 = \begin{bmatrix} a_{jj} & a_{jl} \\ a_{lj} & a_{ll} \end{bmatrix}$$

If

$$\max_i \sum_{j=1}^{2} |e_{ij}| < \frac{|a_{lm}|}{t}$$

is satisfied, then E_2 is the pivot. Here a_{lm} is the largest term in row l and e_{ij} are the terms of E_2^{-1}.

If both of the above pivots fail, then a search is performed in the remaining possible pivot rows $(k > j)$ for pivots. When one is found, that particular row is permuted into the *j*-th row position (by putting ones into the proper locations of the P matrix), and the numerical elimination proceeds.

The numerical work is executed primarily by vector kernels. The triangular kernels DFMR and DFMQ are also used in addition to the conventional AXPY kernel.

Numerical Reliability of Symmetric Decomposition

The numerical reliability of the matrix decomposition process is monitored via the matrix/factor diagonal ratio as follows:

$$\frac{a_{ii}}{d_i} \qquad \text{Eq. 3-10}$$

where a_{ii} is the original diagonal term of the matrix and d_i is the corresponding diagonal term of the factor. The maximum value of these ratios is used to indicate how well-conditioned the original matrix was. The higher this ratio, the closer the matrix is to singularity. As shown by the algorithm of the decomposition, small d_i values are the cause of numerical instability. Hence, in the case of unusually high matrix/factor diagonal ratios, the user should practice extreme care in evaluating the results.

In NX Nastran, a common source of high ratios are mechanisms. A mechanism is a part of the structure that may move independently from the rest of the structure as a rigid body.

For example, when unconstrained directions allow the entire model to move (a mechanism) a high ratio occurs at the last grid point in the internal sequence. Another possible cause of high ratios is connecting flexible elements to stiff elements. Finally, missing elements can also cause high ratios.

In general, ratios below 10^7 are usually acceptable; however, the safety limit is approximately 10^3.

Unsymmetric Decomposition

The sparse unsymmetric decomposition algorithm is another variation of the Gaussian elimination as follows:

For $j = 1, ..., n$ (loop on all columns)

For $i = 1, ..., n$ (loop on all rows)

If $i \geq j$, then (elements of lower triangular factor)

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}$$

If $i < j$, then

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}}{l_{ii}}$$

Note that the diagonal entries of *U* are equal to 1 by construction. For numerical reliability, the above elimination order is modified when necessary. The pivoting step is based on the following criterion:

$$a_{kk} \geq \max_{i} |a_{ik}| \bullet t \qquad \text{Eq. 3-11}$$

Thus, the a_{kk} term is accepted as a possible pivot term if it is larger than the maximum taken in the *k*-th column multiplied by a ratio of *t* (which is based on a user-specified threshold parameter; see **Eq. 3-12**).

From the computer science aspect, the sparse unsymmetric decomposition is similar to the symmetric decomposition, using indexed vector operations and frontal logic which are not discussed here in detail. The sparse unsymmetric decomposition also has a distinct symbolic and numeric phase similar to symmetric sparse decomposition.

Partial Decomposition

The sparse symmetric decomposition method may be used to decompose only a certain partition specified by a partitioning vector (PARTVEC input data block for DCMP). In this case the following decomposition is obtained:

$$A = \begin{bmatrix} A_{oo} & A_{oa} \\ A_{ao} & A_{aa} \end{bmatrix} = \begin{bmatrix} L_{oo} & \\ L_{ao} & I \end{bmatrix} \begin{bmatrix} D_{oo} & \\ & \bar{A}_{aa} \end{bmatrix} \begin{bmatrix} L_{oo}^T & L_{ao}^T \\ & I \end{bmatrix}$$

where

$$\bar{A}_{aa} = A_{aa} - L_{ao} D_{oo} L_{ao}^T$$

The results of L_{oo}, D_{oo}, L_{ao} are stored in the LD output data block and the $\bar{A}_{aa}$ is in the LSCM data block (see "**User Interface**" on page 58).

Distributed Decomposition

A distributed decomposition method based on domain decomposition is available using the DISDCMP module. This method is used, for example, in the GDSTAT parallel SOL 101 sequence. Essentially, each processor performs a partial decomposition, producing a decomposition of the local interior matrix A_{oo} and a global Schur complement (assembled from the local Schur complements). A conceptually similar distributed decomposition method is used in the eigensolver; see "**Geometric Domain Decomposition-Based Distributed Parallel Lanczos Method**" on page 159 for details. The DISFBS module is used to perform the corresponding distributed forward-backward substitution.

Diagonal Scaling Option

When the input matrix is sparse and has diagonal entries of widely varying magnitude, the factor in symmetric decomposition may have more deferred pivots, therefore, obtain an excessively large front size and spend much more time for factorization. Diagonal scaling would achieve a smaller front size and improve performance of symmetric decomposition.

Diagonal scaling prescales the input matrix A into $\tilde{A}$:

$$\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

where D is a diagonal matrix. Each entry of D is the magnitude of the corresponding diagonal term in A . The decomposition of $\tilde{A}$ is:

$$\tilde{A} = \tilde{L}\tilde{D}\tilde{L}^T$$

and the decompositon of A is

$$A = L\left(D^{\frac{1}{2}}\tilde{D}D^{\frac{1}{2}}\right)L^T$$

where

$$L = D^{\frac{1}{2}} \tilde{L} D^{-\frac{1}{2}}$$

3.3 User Interface

DCMP Matrix decomposition with extended diagnostics

Decompose a square matrix [A] into upper and lower triangular factors [U] and [L] and diagonal matrix [D]. DCMP also provides extended diagnostics.

$[A] = [L][U]$ for unsymmetric [A]

$[A] = [L][D][L]^T$ for symmetric [A]

Format:

```
DCMP        USET,SIL,EQEXIN,A,,PARTVEC/
            LD,U,LSCM/
            S,N,KSYM/CHOLSKY/BAILOUT/MAXRATIO/SETNAME/F1/DECOMP/
            DEBUG/THRESH/S,N,MINDIAG/S,N,DET/S,N,POWER/S,N,SING/
            S,N,NBRCHG/S,N,ERR/LMTROWS $
```

Input Data Blocks:

USET	Degree-of-freedom set membership table.
SIL	Scalar index list.
EQEXIN	Equivalence between external and internal numbers.
A	A square matrix (real or complex, symmetric or unsymmetric).
PARTVEC	Partitioning vector specified when A is a partition of SETNAME. Its rowsize is indicated by SETNAME. A is the zero-th partition from PARTVEC. In the partial decomposition case it defines A_{oo}.

Output Data Blocks:

LD	Nonstandard lower triangular factor [L] and diagonal matrix [D] or Cholesky Factor. [LD] also contains [L_{ao}] for partial decomposition.
U	Upper triangular factor or high ratios matrix. If A is unsymmetric, U is the nonstandard upper triangular factor of [A] or the Cholesky factor. If A is symmetric and the value of system cell 166 includes the value of 8, U contains the "high ratio terms of the factor diagonal ratios."
LSCM	Resequencing matrix based on internal resequencing of A.

Parameters:

KSYM	Input/output-integer-default=1. See "**Method Selection**" on page 61
CHOLSKY	Input-integer-default=0. See "**Method Selection**" on page 61
BAILOUT	Input-integer-default=0. If BAILOUT≥0, then the module exits with error message if factor to diagonal ratio exceeds MAXRATIO. If BAILOUT≤–1, then the module continues with warning message if factor to diagonal ratio exceeds MAXRATIO.
MAXRATIO	Input-real-default=1.E5. See the BAILOUT and ERR parameter.
SETNAME	Input-character-default='H'. One or two letters indicating the set membership of [A].
F1	Input-real-default = 0.0. Tolerance for suppressing numbers of small magnitude. Matrix elements with magnitudes less than F1 will be set to zero.
DECOMP	Input-integer-default=–1. See "**Option Selection**" on page 62.
DEBUG	Input-integer-default=–1. See "**Option Selection**" on page 62.
THRESH	Input-integer-default=–6. See "**Option Selection**" on page 62.
MINDIAG	Output-real double precision-default=0.0D0.
DET	Output-complex-default=(0.0,0.0).
POWER	Output-integer-default=0.
SIGN	Output-integer-default=0. See "**Option Selection**" on page 62.
NBRCHG	Output-integer-default=0. See READ module.
ERR	Output-integer-default=–1. If BAILOUT=–1, this parameter always remains at zero. If BAILOUT=0 and the factor to diagonal ratio is negative or greater than MAXRATIO, ERR is reset to –1.
LMTROWS	Input-integer-default=0. Number of Lagrange multipliers appended to the A matrix. These rows are excluded from the internal reordering in the DCMP module.

DISDCMP Distributed decomposition

Performs distributed decomposition which includes the parallel elimination of boundary nodes and summation of global Schur complement.

Format:

```
DISDCMP   USET,SIL,EQEXIN,SCHUR,,EQMAP/
          LBB,DSFDSC,SCHURS/
          HLPMETH/////////////// $
```

Input Data Blocks:

USET	Degree-of-freedom set membership table.
SIL	Scalar index list.
EQEXIN	Equivalence between external and internal numbers.
SCHUR	Local Schur complement matrix in sparse factor format.
EQMAP	Table of degree-of-freedom global-to-local maps for domain decomposition.

Output Data Blocks:

LBB	Distributed boundary matrix factor in sparse factor format (contains the local panels of the fronts).
DSFDSC	Distributed boundary matrix factor.
SCHURS	Sum of all Schur matrices from all processors.

Parameters:

HLPMETH	Input-integer-default=1. Processing option. >0 Summation only. =0 Complete boundary decomposition.

3.4 Method Selection

To select decomposition methods in DCMP, the following parameters are used:

KSYM
- 1 Use symmetric decomposition (default).
- 0 Use unsymmetric decomposition.
- –1 Use decomposition consistent with form of [A]. KSYM will be reset to 0 or 1 consistent with actual decomposition type.
- 3 Use symmetric partial decomposition.

CHOLSKY If KSYM=1 or KSYM=–1 and [A] is symmetric then:
- 1 Use Cholesky decomposition.
- 0 Use standard decomposition (default).

If KSYM=3, then CHOLSKY is set to the number of degrees of freedom in the o-set.

3.5 Option Selection

Minimum Front Option

To increase performance of the sparse symmetric decomposition, the user may set SYSTEM(198), the MINFRONT keyword to a value greater than 1. The appropriate value is problem and machine dependent. Its meaning is to restrict the sparse strategy to above a certain minimum front size (characteristically 8 – 16).

Reordering Options

The various reordering options are selected via SYSTEM(206) as follows. Note that the EXTREME method has two submethods; BEND and AMF. If EXTREME is used when SYSTEM(206) = 0 (default), then either BEND or AMF is automatically selected by the software depending on the size of the model. If SYSTEM(206) = 4, then BEND is used.

SYSTEM(206) DCMPSEQ	Method
0	(Default) EXTREME for 3D, Metis or MMD for 2D
1	MMD – definite matrices
2	MMD – indefinite matrices
3	No sequencing
4	EXTREME
8	Metis
9	Better of Metis and MMD
32	MLV
64	Turn on Supernodal Compression Scheme

Compression Options

The supernodal compression scheme currently is available only with EXTREME and Metis. Supernodal compression scheme with EXTREME is selected by SYSTEM(206) = 68 (64 + 4), and similarly supernodal compression scheme with Metis is chosen by SYSTEM(206) = 72 (64 + 8).

The grid-based compression scheme is automatically executed when the datablocks defining the grid-DOF connections (USET,SIL) are available to the module.

Non-Sparse SDCOMP Options

If bit 4 of SPARSE = SYSTEM(126) is set to 0, then sparse symmetric decomposition is deactivated, and the old non-sparse symmetric decomposition is used instead. If bit 1 of SYSTEM(166) is set and the sparse decomposition fails because of insufficient memory, the old non-sparse decomposition will be attempted.

TAUCS provides another option for sparse decomposition of symmetric positive definite matriices. The functionality is Multifrontal Supernodal Cholesky Factorization. It uses the BLAS to factor. For better performance, an efficient BLAS is necessary. TAUCS performs factorization in-core and requires larger memory than sparse Cholesky decomposition. Without sufficient memory, it will fail and fall back to sparse Cholesky decomposition. More detail about TAUCS can be found at http://www.tau.ac.il/~stoledo/taucs/.

If the SPARSE setting allows sparse symmetric decomposition, Cholesky decomposition and TAUCS decomposition are further controlled by SPCHOL = SYSTEM(424) as in the following table:

SYSTEM(424) SPCHOL	Cholesky Decomposition
0	Old non-sparse Cholesky only
1	Attempt sparse; if sparse fails, fall back to old non-sparse
2	Sparse only, do not fall back
4	TAUCS; if fails, fall back to option 0
5	TAUCS; if fails, fall back to option 1

Non-Sparse UDCOMP Option

If SYSTEM(209) = 0 is set, the old non-sparse unsymmetric Gaussian elimination option is used.

Perturbation Options

If DEBUG (= SYSTEM(60)) is set with the old non-sparse decomposition, then an $\varepsilon = 10^{-\text{DEBUG}}$ replaces the zero diagonal terms. If DEBUG is not set or the sparse decomposition is used, then the perturbation is $\varepsilon = 10^{-10}$.

If SYSTEM(69)=16 with sparse decomposition, then a 1.0 is placed in the diagonal position for all null columns of the input matrix.

High Rank Options

An additional performance improvement is possible, with the high rank update in the sparse decomposition methods. The rank of update in the various sparse decomposition methods is set as:

SYSTEM	Sparse Decomposition
(205)	Symmetric, real
(219)	Symmetric, complex
(220)	Unsymmetric, real
(221)	Unsymmetric, complex

The defaults of these cells are set automatically, since they are machine dependent.

Diagnostic Options

These options of the sparse symmetric decomposition are requested as follows:

SYSTEM	Action
(69) = 1	Stop if null column is found
(69) = 4	Stop if zero diagonal term is found
(69) = 16	Place 1.0 on diagonal of null columns and continue
(69) = 32	Terminate on zero diagonal term
(69) = 64	Stop after diagnostic phase

(166) =	1	Fall back due to insufficient memory
	2	Provides internal diagnostics
	4	Overwrites MAXRATIO by 1.0
	8	Provides MAXRATIO vector in U
	16	Reserved for Siemens PLM internal use
	32	Reserved for Siemens PLM internal use
	64	UWM or UFM if sparse Cholesky fails
	128	Diagonal scaling
(294)>0		Print debugging information from symbolic phase

The THRESH parameter is used to control the pivoting for the unsymmetric and the sparse (which also pivots) decompositions. The pivot threshold is

$$t = 10^{THRESH} \qquad \text{Eq. 3-12}$$

In the unsymmetric case, pivoting occurs if a factor diagonal value is less than t. In the symmetric case of the sparse decomposition, pivoting occurs when the ratio for the factor diagonal value to the largest term in the pivot row is less than t.

The default value of THRESH is 6 for the sparse decomposition and 10 for the unsymmetric decomposition. The latter may also be defined by SYSTEM(91) for the unsymmetric case.

In the case where DCMP is called from eigenvalue analysis, the THRESH parameter may be set by SYSTEM(89).

The shared memory parallel execution of the sparse symmetric decomposition can be selected by

SYSTEM(126) = 8 and SYSTEM(107) > 1

and deselected by turning off either one of these system cells.

3.6 Diagnostics

The diagnostics given by the decomposition module are organized into numerical, performance, statistical, and error diagnostics.

Numerical Diagnostics

These diagnostics are related to the accuracy of the solution.

Singularity

Causes of Singularity. A matrix is singular if its inverse cannot be calculated. The SING parameter is set to –1 if the matrix is singular. Singularity is a relative issue with respect to stiffness ratios. However, some independent general reasons for singularity include:

- Degree of freedom without stiffness
- 2-D problem, normal rotation unconstrained
- 3-D problem, rotational DOFs at solids unconstrained
- Planar joint in space structure

Singularity Test

To avoid singularity, the user can enter the AUTOSPC keyword. If AUTOSPC is set to YES, the singular degrees of freedom are automatically constrained out. A degree of freedom is considered singular if

$$\frac{A_{ij}}{A_{max}} \leq \varepsilon \qquad \text{Eq. 3-13}$$

where A_{ij} is the term in the *i*-th row and the *j*-th column of A matrix, and A_{max} is the largest term in $[A_{elem}]$.

The default for ε is 10^{-8} and can be changed by the keyword EPZERO. The SPC entries constraining the singular DOFs are generated by setting the SPCGEN keyword to YES.

Parameter EPPRT (Bulk Data) (default = 10^{-8}) is used to set a threshold below which all potential singularities are listed. If EPPRT is greater than EPZERO, then the printing of singularities with a ratio of exactly zero is suppressed.

Ill-Conditioning

Causes of Ill-Conditioning. The ill-conditioning of a matrix can be caused by any of the following reasons:

- Low stiffness in rotation
- Large mass
- Very stiff beam
- Mechanisms

MAXRATIO Parameter. Ill-conditioning is diagnosed by the MAXRATIO diagnostics parameter which is defined as

$$\text{MAXRATIO} = \frac{A_{ii}}{D_{ii}} \qquad \text{Eq. 3-14}$$

where D_{ii} is the *i*-th diagonal term in the D matrix.

The maximum MAXRATIO of the decomposition is printed under User Information Message 4158 or 4698 when its value is greater than 10^7. This limit can be changed by setting the keyword MAXRATIO. Setting SYSTEM(60) = -999 will result in printing only the highest and lowest.

In the case of 2×2 pivoting, this ratio is not calculated.

Negative Terms on Factor Diagonal

STURM Number. The NBRCHG parameter (DMAP call) gives the number of negative terms on the factor diagonal (also called the STURM number). This diagnostics information message is important when decomposition is used in an eigenvalue module. In this case, the number of negative terms provides the number of negative eigenvalues of the matrix. Since the matrix decomposed in this case is usually a shifted matrix, the NBRCHG gives the number of eigenvalues to the left of the shift. User Information Messages 4158 and 5010 in the eigenvalue modules print the value of NBRCHG.

Performance Diagnostics

For symmetric decomposition, the following message (UIM 4157) appears:

MATRIX SIZE	NUMBER OF NONZEROES
NUMBER OF ZERO COLUMNS	NUMBER OF ZERO DIAGONALS
CPU TIME ESTIMATE	I/O TIME ESTIMATE
EST. MEMORY REQUIRED	MEMORY AVAILABLE
EST. INTEGER WORDS IN FACTOR	EST. NONZERO TERMS IN FACTOR
EST. MAX FRONT SIZE	RANK OF UPDATE

The integer words in factor are the row and column index information, and the real words are the actual terms of the factor matrix [L]. In the sparse methods, the integer and real words are stored in two separate records.

The 4157 message is followed by UIM 6439 for the sparse symmetric decomposition as follows:

UIM 6439 (DFMSA) ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE SYMMETRIC DECOMPOSITION

SPARSE DECOMP MEMORY REQUIRED	MAXIMUM FRONT SIZE
INTEGER WORDS IN FACTOR	NONZERO TERMS IN FACTOR

For unsymmetric sparse decomposition, UIM 4216 provides the following information.

MATRIX SIZE	NUMBER OF NONZEROES
NUMBER OF ZERO COLUMNS	NUMBER OF ZERO DIAGONAL TERMS
CPU TIME ESTIMATE	I/O TIME ESTIMATE
ESTIMATED MEMORY REQUIREMENT	MEMORY AVAILABLE
EST. INTEGER WORDS IN FACTOR	EST. NONZERO TERMS
ESTIMATED MAXIMUM FRONT SIZE	RANK OF UPDATE

This message is also followed by a UIM 6439 which gives actual values for the estimates in UIM 4216.

Statistical Diagnostics

The following messages are self-explanatory.

UIM 4158:
STATISTICS FOR SYMMETRIC (PARALLEL AND/OR SPARSE)
DECOMPOSITION OF DATA BLOCK XX
NUMBER OF NEGATIVE TERMS ON FACTOR DIAGONAL.
MAXIMUM RATIO OF MATRIX DIAGONAL TO FACTOR DIAGONAL.

UIM 4367:
STATISTICS FOR UNSYMMETRIC DECOMPOSITION OF DATA BLOCK XX FOLLOW.
NUMBER OF PIVOT OPERATIONS = XX.

UWM 5221:
STATISTICS FOR DECOMPOSITION OF MATRIX XX.
THE FOLLOWING DEGREES OF FREEDOM HAVE NULL COLUMNS.

UWM 4698:
STATISTICS FOR DECOMPOSITION OF MATRIX XX.
THE FOLLOWING DEGREES OF FREEDOM HAVE FACTOR DIAGONAL RATIOS GREATER THAN MAXRATIO OR HAVE NEGATIVE TERMS ON THE FACTOR DIAGONAL.

Error Diagnostics

The following are messages from sparse decomposition and are described as follows:

SFM 4370:
DECOMPOSITION REQUIRES THAT PRECISION OF DATA BLOCK XX EQUAL SYSTEM PRECISION.

This is a general limitation of the module.

UFM 3057:
MATRIX XX IS NOT POSITIVE DEFINITE.

A Cholesky option was requested by the user on a non-positive definite matrix.

SFM 4218:
UNSYMMETRIC DECOMPOSITION IS ABORTED DUE TO INSUFFICIENT MEMORY.

The preface of the unsymmetric decomposition module needs more memory to execute.

SFM 4255:
UNSYMMETRIC DECOMPOSITION OF DATA BLOCK XX FAILS AT ROW XX. UNABLE TO PIVOT.

A singular matrix was given to the module.

UW(F)M 6136 (DFMSA):
INSUFFICIENT CORE FOR SYMBOLIC (NUMERIC) PHASE OF SPARSE DECOMPOSITION.
USER ACTION: INCREASE CORE BY XX WORDS.

UFM 6133 (DFMSD):
SINGULAR MATRIX IN SPARSE DECOMPOSITION.
USER ACTION: CHECK MODEL

UFM 6134 (DFMN):
MATRIX IS NOT POSITIVE DEFINITE IN SPARSE DECOMPOSITION AT ROW = XX:
USER ACTION: CHECK MODEL

Sparse Cholesky decomposition failed because the matrix is not positive definite.

SWM 6731 (SDCBOD):
ROW XX OF LOWER TRIANGULAR FACTOR HAS DIAGONAL TERM = 0 (OR .LT. 0 IF CHOLESKY)

Non-sparse decomposition failed, usually because a Cholesky decomposition was requested and the matrix is not positive definite.

UFM 6137 (DFMSD):
INPUT MATRIX IS RANK DEFICIENT, RANK = XX.
USER ACTION: CHECK MODEL

3.7 Decomposition Estimates and Requirements

The CPU time estimate for the sparse symmetric decomposition method includes:

Computation time (sec):

$$\frac{1}{2} \cdot N \cdot N_{front}^2 \cdot M \qquad \text{Eq. 3-15}$$

Data move time (sec):

$$\frac{1}{2} \cdot N \cdot N_{front} \cdot P_s + 2 \cdot P_s \cdot N_z \qquad \text{Eq. 3-16}$$

where:

N = order of problem

N_{front} = average number of connected DOFs

N_z = approximate number of nonzeroes in the upper triangle of the original matrix $N^2 \bullet \rho/2 + N$

ρ = density of the original matrix

Note: M and P_s are defined in the Glossary of Terms.

Storage Requirements. The minimum storage requirements are as follows:

Disk: $(1 + NWPT)N_{front} \cdot N + N$

Memory: $(6 + 2 \cdot NWPT) \cdot N$

where:

$NWPT$ = number of words per term

1 for 32 bit word real arithmetics
2 for 32 bit word complex arithmetics
2 for 64 bit word real arithmetics
4 for 64 bit word complex arithmetics

The CPU time estimate for the sparse unsymmetric decomposition method is

Computation time (sec):

$$N \cdot N_{front}^{2} M \qquad \text{Eq. 3-17}$$

Data move time (sec):

$$N \cdot N_{front} P_s + 4 P_s N_z \qquad \text{Eq. 3-18}$$

3.8 References

Bunch, J. R.; Parlett, B. N. ***Direct Methods for Solving Symmetric Indefinite Systems of Linear Equations.*** Society for Industrial and Applied Mathematics Journal of Numerical Analysis, Volume 8, 1971.

Duff, I. S.; Reid, J. K. ***The Multifrontal Solution of Indefinite Sparse Symmetric Linear Systems.*** Harwell Report CSS122, England, 1982.

George, A.; Liu, J. W. ***Computer Solutions of Large Sparse Positive Definite Systems.*** Prentice Hall, 1981.

Goehlich, D.; Komzsik, L. ***Decomposition of Finite Element Matrices on Parallel Computers***. Proc. of the ASME Int. Computers in Engineering Conf., 1987.

Golub, G. H.; Van Loan, C. F. ***Matrix Computations***. John Hopkins University Press, 1983.

Hendrickson, B., Rothberg, E. I**mproving the Runtime and Quality of Nested Dissection Ordering**. Silicon Graphics, Inc., Mountain View, CA, April 11, 1996.

Karypis, G., Kumar, V. ***METIS©, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 3.0.3***, University of Minnesota, Department of Computer Sciences/Army HPC Research Center, Minneapolis, MN, November 5, 1997. (http://www.cs.umn.edu/~karypis)

Pissanetzsky, S. ***Sparse Matrix Technology.*** Academic Press, 1984.

Shamsian, S.; Komzsik, L. ***Sparse Matrix Methods in MSC/NASTRAN***. The MSC 1990 World Users Conf. Proc., Vol. II, Paper No. 64, March, 1990.

Rothberg, E. **Ordering Sparse Matrices Using Approximate Minimum Local Fill**, Silicon Graphics, Inc., Mountain View, CA, April 11, 1996.

CHAPTER

4 Direct Solution of Linear Systems

- Solution Process
- Theory of Forward-Backward Substitution
- User Interface
- Method Selection
- Option Selection
- Diagnostics
- FBS Estimates and Requirements

4.1 Solution Process

Solution of linear systems is an important and time-consuming component of NX Nastran runs. Mathematically, the solution process represents a right-handed

$$[A][X] = [B] \quad \text{Eq. 4-1}$$

or a left-handed

$$[X]^T[A] = [B]^T \quad \text{Eq. 4-2}$$

direct solution.

The iterative solution directly solves **Eq. 4-1** and is described in "**Iterative Solution of Systems of Linear Equations**" on page 89.

The direct solution method contains two distinct parts, a forward and a backward substitution, hence the name forward-backward substitution.

The forward-backward substitution is a follow-up operation to the decomposition process (see "**Matrix Decomposition**" on page 49).

The right-handed direct solution step is executed using the triangular factors calculated in the decomposition as a forward step of

$$[L][Y] = [B] \quad \text{Eq. 4-3}$$

The backward step in the case of symmetric decomposition uses the intermediate result $[Y]$ as follows:

$$[L]^T[X] = [D]^{-1}[Y] \quad \text{Eq. 4-4}$$

In the case of unsymmetric decomposition, the backward step is

$$[U][X] = [Y] \quad \text{Eq. 4-5}$$

The left-handed forward step to solve **Eq. 4-2** is

$$W^T L^T = B^T \quad \text{Eq. 4-6}$$

and

$$X^T L = W^T \quad \text{Eq. 4-7}$$

In **Eq. 4-6** and **Eq. 4-7**, the decomposition is assumed to be a Cholesky method. In the LDL^T case, **Eq. 4-7** is modified as

$$X^T L = W^T D^{-1} \quad \text{Eq. 4-8}$$

The theory covering the solution of systems of linear equations is described in the following section.

4.2 Theory of Forward-Backward Substitution

Right-Handed Method

The elements of matrices $[Y]$ and $[X]$ of **Eq. 4-3** and **Eq. 4-4** are given by

$$y_{ij} = b_{ij} - \sum_{k=1}^{i-1} l_{ik}\, y_{kj} \qquad \text{Eq. 4-9}$$

and

$$x_{ij} = \frac{y_{ij}}{d_i} - \sum_{k=i+1}^{n} l_{ki}\, x_{kj} \qquad \text{Eq. 4-10}$$

NX Nastran uses a sparse implementation of the above algorithm, as well as special modifications for the parallel, unsymmetric, and left-handed cases.

Left-Handed Method

The elements of **Eq. 4-6** and **Eq. 4-7** are:

$$w_{ik} = \frac{b_{ik} - \sum_{j=1}^{i-1} w_{jk}\, l_{ij}}{l_{ii}} \qquad \text{Eq. 4-11}$$

and

$$z_{ik} = \frac{w_{ik} - \sum_{j=i+1}^{n} w_{jk}\, l_{ij}}{l_{ii}}$$

In the above equations, the A matrix is assumed to have been decomposed with the Cholesky method.

Sparse Method

The sparse option of FBS executes the forward-backward substitution from the factor of the sparse (multifrontal) decomposition. Therefore, one must consider the pivoting performed by the permutation matrix P for the symmetric case, or the permutations P and Q for the unsymmetric case, and then solve the following matrix equations. For the symmetric case,

$$LDL^{T}PX = PB \quad \text{Eq. 4-12}$$

For the unsymmetric case,

$$PLUQX = PB \quad \text{Eq. 4-13}$$

The usual forward pass solves the following equation for the symmetric case

$$LY = PB \quad \text{Eq. 4-14}$$

For the unsymmetric case,

$$PLY = PB \quad \text{Eq. 4-15}$$

for Y. The backward pass gives X from the following equation for the symmetric case

$$DL^{T}(PX) = Y \quad \text{Eq. 4-16}$$

For the unsymmetric case,

$$UQX = Y \quad \text{Eq. 4-17}$$

The storage requirements are real arrays for the right-hand side B and the result X, an integer vector of length N holding the permutation information, and workspace for the factor.

Note that only real right-hand sides are supported in the sparse Cholesky case.

Parallel Method

A shared memory, parallel execution of the sparse method is also available. The parallelization is either on the factor side of the equations, based on special shared memory parallel kernels, or on the right-hand side when multiple loads exists.

Also, the DISFBS module is available for distributed forward-backward substitution based on domain decomposition. The distributed method uses the factor and Schur complement matrices produced by the DISDCMP module. Conceptually, this is similar to the distributed linear solution used in the parallel Lanczos method; see "**Geometric Domain Decomposition-Based Distributed Parallel Lanczos Method**" on page 159.

4.3 User Interface

To solve the matrix equation $[A][X] = \pm[B]$ (right-handed solution) or $[X]^T[A] = [B]^T$ (left-handed solution) using the triangular factors computed by DCMP.

```
FBS       LD,U,B/X/KSYM/SIGN/FBTYP $
```

Input Data Blocks:

LD	Lower triangular factor/diagonal, or Cholesky factor.
U	Upper triangular factor. Purged unless [A] is unsymmetric.
B	Rectangular matrix.

Output Data Block:

X	Rectangular matrix having the same dimensions as [B].

Parameters:

KSYM	Input-integer-default = –1. See "**Method Selection**" on page 82.
SIGN	Input-integer-default = 1. See "**Method Selection**" on page 82.
FBTYP	Input-integer-default = 0. See "**Option Selection**" on page 83.

To perform a distributed forward-backward substitution using the factors computed by DISDCMP.

```
DISFBS    LBB,DSFDSC,EQMAP,UABAR/UA,PABAR,LOO/HLPMETH $
```

Input Data Blocks:

LBB	Distributed boundary sparse factor matrix (contains the local panels of the fronts).
DSFDSC	Table description of boundary sparse factor matrix.
EQMAP	Table of degree-of-freedom global-to-local maps for domain decomposition.
UABAR	Local updated rectangular ("loads") matrix.

Output Data Blocks:

UA	Global boundary solution for distributed decomposition.
PABAR	Summed up updated rectangular ("loads") matrix for distributed decomposition.
LOO	Merged boundary sparse factor matrix for distributed decomposition.

Parameters:

HLPMETH Input-integer-default = 0. Processing option.

>0 Summation only.

=0 Complete distributed forward-backward substitution (default).

=4 Summation operation and merging of distributed sparse boundary factor matrix.

Remark: LBB and DSFDSC may be purged.

4.4 Method Selection

FBS Method Selection

The FBS method selection is executed via the following parameters:

KSYM Symmetry flag.

- –1 choose symmetric if [U] is purged, otherwise unsymmetric (default).
- 0 matrix [A] is unsymmetric.
- 1 matrix [A] is symmetric.
- 2 perform left-handed solution. See "**Option Selection**" on page 83.

SIGN Sign of [B].

- 1 solve [A] [X] = [B] (default).
- –1 solve [A] [X] = –[B].

Additionally, if the factor matrix is symmetric and non-sparse, the system cell FBSOPT = SYSTEM(70) may be used to select a submethod. By default, the method minimizing the sum of CPU and I/O time estimates is chosen. The value of FBSOPT is interpreted as follows:

SYSTEM(70) FBSOPT	Non-Sparse Submethod
-2	Method 1A
-1	Method 1
0	Automatic selection (default)
+1	Method 2

4.5 Option Selection

Right-handed FBS Options

The user can control the flow of the forward-backward substitution from within source code via SYSTEM(74), or by using the FBTYP parameter (DMAP call). If the FBS module is called from DMAP, then SYSTEM(74) is overriden by the FBTYP parameter. The default value is 0, which indicates a full execution of both passes of the solution. Set FBTYP to +1 for a forward pass only; set FBTYP to –1 for a backwards only partial execution. These options are useful in eigenvalue modules.

If the forward-backward substitution is called from source code with a symmetric non-sparse factor matrix, then SYSTEM(73) = +1 can be set to indicate the presence of a Cholesky factor $[L][L]^T$. The default is 0, which indicates the regular $[L][D][L]^T$ factor. The factor type is determined automatically in the FBS module, so this system cell has no effect when called from DMAP. To summarize this:

SYSTEM	Value	Action
(73)	+1	Cholesky factor: $[L][L]^T[X] = [B]$
	else	Regular factor: $[L][D][L]^T[X] = [B]$ (default)
(74)	+1	Solve $[L][Y] = [B]$ or $[L][D][Y] = [B]$
or FBTYP	1	Solve $[L]^T[X] = [Y]$
	0	Full FBS (default)

Left-handed FBS Option

The left-handed FBS obtains the solution by rows, as opposed to columns. Since the solution is packed out via the usual GINO facilities, the solution rows are stored as columns of the solution (X) matrix. The user may input a transposed or untransposed right-hand side (B) matrix, except for sparse Cholesky factors, in which case only SYSTEM(72) = 1 is supported. To summarize this:

SYSTEM	Value	Action
(72)	=1	$B = X^T A$
	≠ 1	$B^T = X^T A$

The FBS options controlled by cells 73 and 74 apply to the left-handed solution similarly.

Parallel FBS Solution

The user needs FBSOPT = –2 and SYSTEM(107) = PARALLEL > 1 for the right-hand side shared memory parallel execution. For the factor side parallel, only PARALLEL > 1 is needed.

The parallel method can be deselected for FBS by setting PARALLEL = 1024 + ncpu where ncpu = number of CPUs. This setting leaves the other parallel methods enabled.

4.6 Diagnostics

For the direct solution of systems of linear equations, diagnostics can also be classified as follows: numerical diagnostics, performance messages, and error diagnostics.

Numerical Diagnostics

UIM 5293:

FOR DATA BLOCK .... LOADSEQ EPSILON EXTERNAL WORK

....

....

Performance Messages

UIM 4234:
UFBS TIME ESTIMATE TO FORM XX TYPE = X CPU = X I/0 = X TOTAL = X PASSES = X

UIM 4153:
FBS METHOD X TIME ESTIMATE TO FORM XX CPU = X I/0 = X TOTAL = X PASSES = X

These messages are printed only when the CPU time estimate is greater than the value of SYSTEM(20) (default = machine-dependent). To force the printing of these messages, the user must set SYSTEM(20) = 0.

Error Diagnostics

FBS 1(2, 3 OR 4) FATAL ERROR 20: USER FATAL MESSAGE

This error should not occur under normal circumstances. The cause for the error is that information from the GINO control block is incompatible with information from the buffer. This occurs in methods 1 or 1A only.

SFM FBSUB LOGIC ERROR 10: USER FATAL MESSAGE

This error is similar to the previous one from method 2.

SFM 6069 (LFBSS):
SYMMETRIC LEFT HANDED FBS IS CALLED TO SOLVE A COMPLEX SYSTEM WITH CHOLESKY FACTOR.

This option is not supported.

SFM 6070 (LFBSS):
ERROR IN READING THE FACTOR IN SYMMETRIC LEFT HANDED FBS

These messages are from the left-handed method. They are either from symmetric, unsymmetric, or from the timing phase of either one. The cause for this error is similar to the cause for Error 20.

SFM 6072 (LFBSU):
INCORRECT PIVOTING INSTRUCTIONS IN UNSYMMETRIC FACTOR DURING A LEFT HANDED FBS.

This message is similar to Error 20 since it indicates inconsistency in a data block.

SFM 6073 (LFBSU):
ERROR IN READING THE FACTOR IN UNSYMMETRIC LEFT HANDED FBS

Similar causes as Error 20.

SFM 6201 (FBSQCK):
SPARSE FBS CANNOT BE EXECUTED WHEN THE FACTOR IS REAL AND THE RIGHT HAND SIDE IS COMPLEX.

Recommendation: Do not select the sparse decomposition and FBS methods under these circumstances.

UFM 6138 (DFMSB):
INSUFFICIENT CORE FOR SPARSE FBS.
USER ACTION: INCREASE CORE BY XX WORDS.

4.7 FBS Estimates and Requirements

Sparse FBS Estimates

The CPU time for sparse FBS is:

$$2\ N_{RHS} \cdot NZ_{FAC} \cdot M \qquad \text{Eq. 4-18}$$

Data move time (sec):

$$2\ N \cdot N_{RHS} \cdot P + 2\ NZ_{FAC} \cdot P_s \cdot N_{pass} \qquad \text{Eq. 4-19}$$

where:

N_{RHS} = number of right-hand sides

NZ_{FAC} = number of nonzeroes in the factor

N_{pass} = number of passes

The minimum storage requirements of FBS are:

Disk: $NZ_{FAC} \cdot \text{IPREC} + 2\ N \cdot N_{RHS} \cdot \text{IPREC}$

Memory: $1/2 \cdot N_{front}^2 \cdot \text{IPREC} + 2\ N \cdot \text{IPREC}$

where:

N_{front} = average front size

IPREC = machine precision (1 for short-word machines, 2 for long-word machines)

Note: M, P and P_s are defined in the Glossary of Terms.

4.8 References

Komzsik, L. ***Parallel Processing in Finite Element Analysis***. Finite Element News, England, June, 1986.

Komzsik, L. ***Parallel Static Solution in Finite Element Analysis***. The MSC 1987 World Users Conf. Proc., Vol. I, Paper No. 17, March, 1987.

CHAPTER

5

Iterative Solution of Systems of Linear Equations

- Iterative Solutions
- Theory of the Conjugate Gradient Method
- Preconditioning Methods
- User Interface
- Iterative Method Selection
- Option Selection
- Global Iterative Solution Diagnostics
- Global Iterative Solver Estimates and Requirements
- Element Iterative Solver Memory Requirements
- References

5.1 Iterative Solutions

The previous chapter discussed the solution of linear systems using the direct method, which consists of matrix decomposition followed by a forward-backward substitution. In this chapter an alternative method, the iterative solution, is described.

There are two types of iterative solution; global and element. The global type uses the assembled matrices while the element type uses the element matrices. The element type is limited in the problems it can solve but is more efficient with solve times as much as 6X faster.

Solution Sequences. The iterative solver can be used in the following solution sequences:

Global	Element
Linear statics (SOLs 1, 101)	Linear statics (SOLs 1, 101)
Nonlinear statics (SOL 106)	
Direct frequency response (SOLs 8, 108)	
Modal frequency response (SOLs 11, 111)	

Parallel Execution. In addition to the above sequential versions, the global iterative solver is implemented for parallel execution on distributed memory machines in SOL 1 and SOL 101 (linear statics).

Methods

Global:

For symmetric positive definite systems, several different versions of the preconditioned conjugate gradient method are available in NX Nastran. For unsymmetric or indefinite systems, the preconditioned bi-conjugate gradient or the preconditioned conjugate residual method is used.

Element:

In the element based version, the preconditioner consists of two parts:

(i) The first part approximates the matrix very well in a lower dimensional subspace. This lower dimensional subspace is generated using the element and node geometry and the types of degrees of freedom in the model. The lower dimensional subspace is meant to provide a very coarse approximation to the low frequency eigenvector subspace.

(ii) The second part is a simple approximation to the matrix in the orthogonal complement of the above low dimensional subspace.

The preconditioner makes a suitable trade-off between the cost of factoring the preconditioner and the estimated number of conjugate gradient iterations required to solve the problem within a reasonable tolerance.

5.2 Theory of the Conjugate Gradient Method

The conjugate gradient method minimizes the error function

$$F(x) = \frac{1}{2} x^T A x - x^T b \qquad \text{Eq. 5-1}$$

The first derivative (gradient) of this function is

$$\frac{dF}{dx} = Ax - b = -r \qquad \text{Eq. 5-2}$$

which is the negative of the residual. An iterative procedure is obtained by calculating consecutive versions of the approximate solution as follows:

$$x_{i+1} = x_i + \alpha_i p_i \qquad \text{Eq. 5-3}$$

and

$$r_{i+1} = r_i - \alpha_i A p_i \qquad \text{Eq. 5-4}$$

where the direction p and the distance α are computed to minimize the above error function.

Computationally efficient forms to calculate these quantities are

$$\alpha_i = \frac{r_i^T r_i}{p_i^T A p_i}$$
$$p_i = r_i + \beta_i p_{i-1} \qquad \text{Eq. 5-5}$$

where

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

See Hageman and Young, 1981 [1] for more details. The algorithmic formulation is fairly straightforward.

Convergence Control

Convergence is achieved inside the iteration loop when the following inequality holds:

$$\frac{\|r_n\|}{\|b\|} < \varepsilon \qquad \text{Eq. 5-6}$$

where r_n denotes the residual vector after n iterations, b denotes the initial right-hand side vector, and ε is a user-defined parameter (default: 1.E-06). If this ratio does not become smaller than ε within the maximum number of iterations, ε is compared to an energy norm (**Eq. 5-7**) after the iteration loop is finished. The solution is accepted if the following inequality holds:

$$\frac{\left|x^T r\right|}{\left|b^T x\right|} < \varepsilon \qquad \text{Eq. 5-7}$$

where x is the solution vector, r is the final residual vector, and b is the initial right-hand side vector as before. Experience has shown that the convergence criterion inside the iteration loop (**Eq. 5-6**) is far more conservative than the convergence criterion outside the iteration loop (**Eq. 5-7**).

Based on user selection, an alternate convergence criterion is available for Jacobi, Cholesky and RIC preconditioning:

$$\Delta x_n < \varepsilon \left|1 - f_n\right| \qquad \text{Eq. 5-8}$$

where:

$$\Delta x_n = \left\|x_{n+1} - x_n\right\|$$

$$f_n = \frac{\Delta x_n}{\Delta x_{n-1}}$$

See Conca, 1992 [2] for details.

Block Conjugate Gradient Method (BIC)

The BIC method is the most recent and most recommended method for the iterative solution of linear systems in NX Nastran. It was introduced several years ago and has gone through major improvements with regard to memory management and spill logic since then.

BIC is a block version of the conjugate gradient method described above where 'block' refers to two characteristics of the method:

1. The efficient solution of the linear system with a block of right-hand sides where the following inequality is expected to hold for $m < 10$ (m = the number of right-hand sides)

$$\sum_{i=1}^{m} t_i > t_m$$

where t_i refers to the time needed to solve the linear system with just the i-th right-hand side and t_m refers to the time needed to solve the linear system with the complete block of m right-hand sides.

The method will still work for $m \geq 10$; however, the inequality may not hold.

2. Both the matrix and the preconditioner are stored in block-structured form to improve the performance.

The block of direction vectors is updated in every iteration as

$$P_k = [T_k^T A T_k]^{-1} T_k^T R_k \quad \text{Eq. 5-9}$$

and the blocks of residual and solution vectors are updated as

$$R_{k+1} = R_k - A T_k P_k \quad \text{Eq. 5-10}$$

$$X_{k+1} = X_k - T_k P_k \quad \text{Eq. 5-11}$$

where all matrices are $n \times m$ size except for T_k, which is obtained by the concatenation of the two matrices as

$$T_k = [X_k - X_{k-1} \,|\, R_k] \quad \text{Eq. 5-12}$$

and it is of size $n \times 2m$. Convergence is achieved when the following inequality holds

$$max\left(\max_i \frac{\|X_{n+1} - X_n\|}{\|X_{n+1}\|} , \max_i \frac{\|R_{n+1}\|}{\|F\|} \right) < \varepsilon \quad \text{Eq. 5-13}$$

where i is the column index, $i \leq m$, R is the rectangular matrix of the r residual vectors, F is the set of right-hand sides, and X is the set of solution vectors. See Babikov, 1995 [3] for more details.

The algorithmic formulation of this method employed in the iterative solution module is fairly straightforward, except for the very difficult issue of performing the linear solve of order $2m$ in **Eq. 5-9**. However, the details of that calculation are beyond the scope of this guide.

Real and Complex BIC

The BIC methods differ for real and complex linear systems of equations, and therefore are described separately in the following paragraphs. The parameters are described in "User Interface" on page 101.

The real BIC method follows the basic algorithm shown below. Here, B is the preconditioner, F is the block of right-hand side vectors, X is the block of updated solution vectors, and R is the block of updated residual vectors.

$$X_0 = X_{-1} = 0$$

$$R_0 = R_{-1} = -F$$

Loop on $k = 1, 2, \ldots,$ ITSMAX

$$W_k = B^{-1} R_k$$

$$V_k = A W_k$$

$$T_k = [X_k - X_{k-1} \mid W_k]$$

$$P_k = [R_k - R_{k-1} \mid V_k]$$

$$G_k = (P_k^T T_k)^{-1}$$

$$H_k = G_k T_k^T R_k$$

$$X_k = X_{k-1} - T_k H_k$$

$$R_k = R_{k-1} - P_k H_k$$

IF (error < ITSEP S) then converged

End loop on k

The memory management and the spill logic are rather involved, and only the basic steps are listed below:

1. Execute a bandwidth-reducing permutation
2. Symbolic phase

Find the best memory usage given a certain amount of memory and an IPAD value (see "**User Interface**" on page 101 for details on IPAD). The following logical steps are traversed in this order:

a. Check if matrix and factor fit in core; if yes, go to step e.
b. Check if factor fits in core with matrix being out-of-core; if yes, go to step e.
c. Check if memory is sufficient when factor and matrix are both out-of-core; if yes, go to step e.
d. Decrease padding level: IPAD = IPAD – 1; go back to step a.
e. Set up memory accordingly.

3. The numeric phase of the preconditioner calculation makes use of the following techniques:
   a. Calculation of preconditioner in double precision; storage of final preconditioner in single precision.
   b. Restarts with global shift regularization if incomplete factorization fails.
   c. Post truncation for well-conditioned problems.
4. Using the techniques described above, more memory may be available than was predicted by the symbolic phase. Unless both matrix and preconditioner are in core, as much as possible is read from the scratch files and saved in memory.

The memory management and spill logic for the complex BIC methods are different.

For the complex case, there are two different BIC methods which are selected via the IPAD value. For IPAD < 5, the complex BIC algorithm is similar to the real BIC method. However, for IPAD ≥ 5, a very different strategy is used, which is described below.

The solution of the system

$$AX = F$$

is based on its equivalent representation

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

where matrices A_{11} and A_{12} ($= A_{21}^T$, since A is symmetric) have fully zero imaginary parts and A_{22} is truly complex. The solution is found using the Schur complement.

There is no spill logic for the first complex method (IPAD < 5): if the matrix and preconditioner do not both fit in core, then a fully out-of-core approach is used.

Spill logic for the second complex BIC method is available. It determines whether only A_{11}, only A_{22}, or both A_{11} and A_{22} can be kept in core. A_{12} is always kept in core since it usually has very few nonzero terms.

The second complex BIC method is recommended and is also the default for complex symmetric linear systems.

5.3 Preconditioning Methods

The use of a preconditioner is recommended to reduce the number of iterations. The disadvantage of this process is that the preconditioning calculation increases the amount of work in each iteration. With the use of a preconditioner matrix, the original problem is now written as

$$P^{-1}Ax = P^{-1}b \qquad \text{Eq. 5-14}$$

where P is the preconditioning matrix. In this form, the preconditioner is applied in every iteration step. This is called stepwise preconditioning. The Jacobi and Cholesky preconditioners are stepwise. If $P = A$ is chosen, then the problem is trivial:

$$P^{-1}Ax = Ix = A^{-1}b \qquad \text{Eq. 5-15}$$

Of course, the cost of this preconditioning is equivalent to the direct solution of the system.

The following four major stepwise preconditioning strategies are supported in NX Nastran.

Jacobi (J). For the Jacobi method, the P matrix is a diagonal matrix containing the diagonal terms of the A matrix. The preconditioning step in every iteration is a simple division of the current residual vector by these terms.

Cholesky (C). In the Cholesky method, the selection of the preconditioner matrix is

$$P = CC^{T} \cong A \qquad \text{Eq. 5-16}$$

where C is an incomplete Cholesky factor of A. Despite wide acceptance, the use of standard Cholesky preconditioning in the conjugate gradient method is only proven to be very good for finite difference discretization of partial differential equations. For example, the conjugate gradient method has convergence problems when used in finite element problems with high Poisson ratios.

Reduced Incomplete Cholesky (RIC). In the RIC method the preconditioner matrix is calculated based on elemental information as

$$P = CC^{T} \cong B = f(A, A_{elem}) \qquad \text{Eq. 5-17}$$

where A_{elem} is the elemental matrices used to assemble A.

This method significantly increases the performance of the iterative solver for shell problems. See Efrat, 1986 [4] for details.

Block Incomplete Cholesky (BIC). Another disadvantage of the Cholesky method is the calculation cost for the incomplete or reduced factor and the memory requirement for parts of the Cholesky factor. A specific block sparse implementation reduces these disadvantages. This proprietary method uses a global shift regularization strategy, a post-truncation technique for well-conditioned problems, and allow a certain level of fill-in based on delicate numerical considerations. The implementation of the block conjugate gradient method accelerates convergence for problems involving multiple loads. Moreover, a band reordering method is used for the matrix in the symbolic phase of the iterative solver. See Babikov, 1995 for details.

Scaling

Another approach for preconditioning is to use the preconditioner as a transformation. Then

$$P^{-1}AP^{-1}Px = P^{-1}b \qquad \text{Eq. 5-18}$$

is transformed into

$$\bar{A}\,\bar{x} = \bar{b} \qquad \text{Eq. 5-19}$$

In this case the solution of the transformed system has to be converted back to the original solution as follows:

$$x = P^{-1}\bar{x} \qquad \text{Eq. 5-20}$$

An example of this transformation approach is diagonal scaling. Diagonal scaling is a useful tool for matrices whose terms differ significantly in magnitude. In this method, the following transformation is performed:

$$D^{-1}A \bullet D^{-1} \bullet \widehat{x} = D^{-1} \bullet b \qquad \text{Eq. 5-21}$$

where $\widehat{x}$ is an intermediate result, such as

$$D = diag(\ldots, \sqrt{a_{ii}}, \ldots) \text{ and } x = D^{-1} \bullet \widehat{x} \qquad \text{Eq. 5-22}$$

The diagonal terms of the scaled matrix are unity as a result of the diagonal scaling. This scaling makes the Jacobi preconditioning step trivial. The other (Cholesky type) preconditioning methods may be combined with scaling.

Numerical Reliability of Equation Solutions

The accuracy of the solution of the linear equation systems in NX Nastran is evaluated with the following residual vector:

$$r = b - Ax \quad \text{Eq. 5-23}$$

This residual vector is calculated for each solution vector in static analysis. Then a scalar value is calculated as follows:

$$\varepsilon = \frac{x^T r}{x^T b} \quad \text{Eq. 5-24}$$

The magnitude of this error ratio indicates the numerical accuracy of the solution vector x.

5.4 User Interface

Format for global non-p-version solution:

```
SOLVIT    A,B,XS,PC,USET,KGG,GM,SIL,EQEXIN,EDT,CASECC,EQMAP/
          X,R,PC1,EPSSE/
          SIGN/ITSOPT/ITSEPS/ITSMAX/IPAD/IEXT/ADPTINDX/
          NSKIP/MSGLVL/PREFONLY/S,N,ITSERR/SEID $
```

Format for global p-version solution:

```
SOLVIT    A,B,XS,PS,USET,USET0,SIL0,SIL,EQEXIN,EDT,CASECC,
          EQMAP/
          X,R,PG,EPSSE/
          SIGN/ITSOPT/ITSEPS/ITSMAX/IPAD/IEXT/ADPTINDX/
          NSKIP/MSGLVL/PREFONLY/S,N,ITSERR/SEID $
```

Input Data Blocks:

A	Square matrix (real or complex, symmetric or unsymmetric).
B	Rectangular matrix (real or complex), the right-hand side.
XS	Optional starting vector, same type as B (may be purged).
PC	Optional stepwise preconditioner, same type as A (may be purged).
USET	Degree-of-freedom set membership table. See Remark 3.
KGG	Stiffness matrix - g-set. See Remark 3.
GM	Multipoint constraint transformation matrix. See Remark 3.
USET0	USET table from previous adaptivity index in p-version analysis.
SIL	Scalar index list.
SIL0	SIL table from previous adaptivity index in p-version analysis.
EQEXIN	Equivalence table between external and internal grid/scalar identification numbers. Required for p-version preconditioning only.
EDT	Table of Bulk Data entry images related to element deformation, aerodynamics, p-element analysis, divergence analysis, and the iterative solver.
CASECC	Table of Case Control command images. Required if SMETHOD Case Control command is used and NSIP=-1.
EQMAP	Table of degree-of-freedom global-to-local maps for domain decomposition.

Output Data Blocks:

X	Solution matrix. Rectangular matrix having the same dimensions and type as [B].
R	Residual matrix. Rectangular matrix having the same dimensions and type as [B], the residual [R] = [B] - [A][X].
PC1	Updated stepwise preconditioner matrix. See Remark 6.
EPSSE	Table of epsilon and external work.

Parameters:

SIGN Input-integer-default = 0. Sign flag for [B].

0 : + [B]

1 : - [B]

ITSOPT Input-integer-default = 0. Preconditioner flag. See "**Option Selection**" on page 108.

0 Choose optimal method based on type of problem:

ITSOPT	Type of problem
6	p-version and real [A] and [B]
10	complex [A] and/or [B]
11	non p-version and real [A] and [B]

1 Jacobi preconditioning (default) for real, complex, symmetric and unsymmetric A.

2 Incomplete Cholesky preconditioning or user-given preconditioner.

3 Reduced incomplete Cholesky preconditioning. preconditioner (available for real symmetric A only).

4 User supplied for real, complex, symmetric A.

5 Incomplete geometric, Jacobi hierarchic for real symmetric A.

6 Complete geometric, Jacobi hierarchic for real symmetric A.

7 Complete geometric, incomplete hierarchic for real symmetric A.

10 Block incomplete Cholesky for well-conditioned real symmetric A (default for real A).

11 Block incomplete Cholesky for well-conditioned complex symmetric A (default for complex A).

<0 Same as above with diagonal scaling.

ITSEPS Input-real-default = 1.0E-6. Convergence parameter epsilon.

ITSMAX Input-integer-default = 0. Maximum number of iterations. The default value implies N/4 (N = dimension of [A]).

IPAD Input-integer-default = 0 (see table below). Padding level for reduced or block incomplete Cholesky factorization (0, 1, 2, ...). See Remarks 1 and 2. See also "**Option Selection**" on page 108.

IPAD default	Method	ITSOPT	Model type	Type of [A]
0	reduced incomplete Cholesky	3	all	real
2	block incomplete Cholesky	10,11	3-D	real
3	block incomplete Cholesky	10,11	2-D or mixed	real
5	block incomplete Cholesky	10,11	all	complex

IEXT Input-integer-default = 0. Extraction level in reduced or block incomplete Cholesky factorization. See Remarks 1 and 2. See also "**Option Selection**" on page 108.

IEXT default	Reduced	Block
0	0 solid bodies, no rigid elements.	Requires USET and SIL
1	1 shells only	Heuristic block structure (default)
2	2 mixed including rigid elements	n/a

ADPTINDX Input-integer-default=0. P-version analysis adaptivity index. See Remark 7.

NSKIP Input-integer-default=1. Record number of current subcase in CASECC and used only if the SMETHOD command selects the ITER Bulk Data entry which specifies values for the desired iteration parameters. If NSKIP=-1 then CASECC is not required and the values are taken from the module specification of the values.

MSGLVL Input-integer-default=0. Message level output. See "**Option Selection**" on page 108.

0 minimal; i.e., UIM 6447 (default).

1 UIM 6447, convergence ratios, and residual norms

PREFONLY Input-integer-default=0. Preface execution only. If set to -1 then SOLVIT is terminated after the preface information is computed and printed.

ITSERR Output-integer-default=0. Iterative solver return code.

1 no convergence

2 insufficent memory

SEID Input-integer-default=0. Superelement identification number.

Remarks:

1. If ITSOPT = 3, the IPAD level is recommended to be 0, 1, or 2 (IEXT = 0) and should be increased when IEXT is increased.
2. The amount of memory needed for ITSOPT = 3, 10, and 11 increases with the increase of the parameters IPAD and IEXT.
3. For ITSOPT = 1 or 2, the input data blocks USET, KGG, and GM may be purged. For ITSOPT = 3, USET must be specified. KGG and GM are necessary only if IEXT = 2.
4. If the message " *** USER FATAL MESSAGE 6288 (SITDRV): UNABLE TO CONVERGE WITH ITERATIVE METHOD" is issued, then results will still be printed but may be inaccurate.
5. The system cell SYSTEM (69) is equivalent to the SOLVE keyword and controls some special options for the module:

SOLVE	Action
2	Suppresses the user information message at each iteration.
8	Use alternative convergence criterion (less conservative than default).

6. If data block PC1 is specified, the CPU time will increase slightly.
7. If SOLVIT is to be used for p-element analysis and ADPTINDX>1, then XS and PC must be the solution matrix and pre-conditioner from the previous adaptivity p-level. Also, the USET and SIL from the previous p-level are specified for U and KGG and the USET and SIL from the current p-level are specified for GM and SIL.
8. For frequency response analysis with ITSOPT=10 or 11 (block incomplete Cholesky), IEXT=0 is not available and IEXT=1 is used automatically.

Examples:

1. Solve [A][X]=[B] with Jacobi pre-conditioning with convergence established at 1.E-4 and maximum allowed iterations at 55 specified for the module parameters.

```
SOLVIT   A,B,,,,,,,,,/X,,//1/1.E-4////-1 $
```

2. Same as 1 except parameters are obtained from the SMETHOD command and ITER entry.

```
SOLVIT   A,B,,,,,,,,EDT,CASECC/X,, $
```

3. Same as 2 except for p-version analysis.

```
DBVIEW SIL0    = SILS   (WHERE PVALID=PVALOLD) $
DBVIEW UL0     = UL     (WHERE PVALID=PVALOLD) $
DBVIEW USET0   = USET   (WHERE PVALID=PVALOLD) $
DBVIEW PRECON0 = PRECON (WHERE PVALID=PVALOLD) $
SOLVIT         KLL,PLI,UL0,PRECON0,USET,USET0,SIL0,SILS,
               EQEXINS,EDT,CASES/
               UL,RUL,PRECON///////ADPTINDX/NSKIP $
```

Format for element based solution:

```
SOLVIT  KELM,PG,KDICT,SIL,ECT,BGPDT,CSTM,EDT,CASECC,USETB,RG,MPT,
        YGB,SLT,MDICT,MELM,EPT/UGV1,QG1,/V,Y,ISIGN/V,Y,IOPT/
        S,N,ITSEPS/V,Y,ITSMAX/V,Y,IPAD/V,Y,IEXT//NSKIP/V,Y,IMSGFL/
        V,Y,IDEBUG/V,Y,IERROR $
```

Input Data Blocks:

KELM	Element stiffness matrix.
PG	Load vector in g set.
KDICT	Element stiffness dictionary.
SIL	Scalar index list.
ECT	Element connectivity table.
BGPDT	Basic grid point data table.
CSTM	Coordinate system transformation matrix.
EDT	Element data table.
CASECC	Case control command images.
USETB	Degree-of-freedom set membership table.
RG	Constraint matrix in g set.
MPT	Material property table.
YGB	Specified non-zero displacements in g set.
SLT	Static load table.
MDICT	Mass dictionary.
MELM	Element mass matrix.
EPT	Element property table.

Output Data Blocks:

UGV1	Displacements - g set.
QG1	SPC forces - g set.

Parameters:

ITSEPS	Input-real-default = 1.0E-6. Convergence parameter epsilon.
ITSMAX	Input-integer-default = 1000. Maximum number of iterations.
ITSERR	Output-integer-default = 0. Iterative solver return code.

5.5 Iterative Method Selection

The NASTRAN keyword ITER (equivalent to SYSTEM(216)) is used to select the iterative solution by ITER = YES. The default is ITER = NO. The NASTRAN keyword ELEMITER (equivalent to SYSTEM(399)) is also required to select the element iterative solution by ELEMITER=YES.

The defaults for the iterative solver can be changed via the Bulk Data entry ITER, which needs to be selected in the Case Control Section as

SMETHOD = SID

The Bulk Data entry ITER uses a free parameter format as follows:

```
ITER      SID
          PRECOND=CHAR CONV=CHAR
          MSGFLG=CHAR ITSEPS=REAL ITSMAX=INT
          IPAD=INT IEXT=INT PREFONLY=INT
```

where CHAR = character type and INT = integer.

Note that the order and existence of the parameters is not mandatory. For example, the following Bulk Data entry selected by SMETHOD=10 in the Case Control Section is valid:

```
ITER      10
          ITSEPS=1.0E-04 PRECOND=J
```

This entry chooses a Jacobi preconditioner with 1.0E-04 accuracy.

Continuation lines must start in column 9, aligning with SID. Embedded spaces and commas are allowed and the conventional continuation entry markers (+xx, +xx) are obsolete.

All integer and real parameters correspond to the appropriate SOLVIT parameters. The parameters used for method and option selections are also described in "**Option Selection**" on page 108.

5.6 Option Selection

Preconditioner Options

The global iterative solver preconditioner options are controlled via the IOPT and the PRECOND parameters as shown below:

PRECOND	IOPT	Preconditioner	Type
J(S)	1(–1)	Jacobi	real, complex, symmetric, unsymmetric
C(S)	2(–2)	Incomplete Cholesky	real, complex, symmetric, unsymmetric
RIC(S)	3(–3)	Reduced Incomplete Cholesky	real, symmetric
USER	4	User given	real, complex, symmetric

The scaling option is chosen by adding an 'S' to PRECOND or by setting IOPT negative. For example, PRECOND=CS or IOPT=–2 means incomplete Cholesky with scaling.

The option PRECOND=USER can be used in the following ways:

- For direct frequency response (SOLs 8, 108) it will result in using the direct method for the first frequency. The factor from this decomposition will then be used for the subsequent frequencies as the preconditioner with the iterative solver. If the iterative solver fails or takes too long time to converge, then the direct method will be used for the next frequency and the new factor will be used as the preconditioner for the subsequent frequencies.
- In cases where several linear systems of the same size need to be solved, and where the system matrices differ only slightly, this option can be used with a DMAP ALTER. A possible scenario is to use the direct method to solve the first linear system, and then use the SOLVIT module for the solution of the subsequent systems. Specify the output data block from the decomposition containing the factor as the 4th input data block (= user given preconditioner) to the SOLVIT module. The following lines of DMAP show a simple example of the solution of two linear systems:

$$A\ X = F$$

and

```
B Y = G
```

where A and B do not differ too much

```
DECOMP    A/L, /$
FBS       L, , F/X/$
SOLVIT    B,G,,L,,,,,,,/Y,,//4//////-1 $
```

The hierarchic (for p-version elements) preconditioning strategies are chosen as:

PRECOND	IOPT	Strategy	Type
PBCJ	5	Incomplete geometric, Jacobi hierarchic	real, symmetric
PBDJ	6	Complete geometric, Jacobi hierarchic (default for p-version problems)	real, symmetric
PBDC	7	Complete geometric, Incomplete hierarchic	real, symmetric

The block incomplete Cholesky (BIC) technique options are:

PRECOND	IOPT	Technique	Type
BIC	11	Well conditioned problems (default for real problems)	real, symmetric
BICCMPLX	10	Complex problems (default for complex problems)	complex symmetric

For the distributed parallel SOLVIT only Jacobi preconditioning is available.

Convergence Criterion Options

The convergence criterion options of SOLVIT are selected by CONV as follows:

CONV	Criterion
AR	Equation Eq. 5-6
GE	Equation Eq. 5-8
AREX	Equations Eq. 5-6 and Eq. 5-7
GEEX	Equations Eq. 5-8 and Eq. 5-7

For the BIC method the convergence criterion is automatically set to **Eq. 5-13**.

The default is determined according to preconditioner and solution type. For SOL 108, **Eq. 5-8** is used; otherwise, **Eq. 5-6** applies. This default is overruled by choosing Jacobi (which selects **Eq. 5-6**) or incomplete Cholesky preconditioning (**Eq. 5-8**). Furthermore, the external convergence criterion (**Eq. 5-7**) is used unless the preconditioner is Jacobi with scaling (IOPT=-1) or incomplete Cholesky with scaling (IOPT=-2).

Diagnostic Output Options

The diagnostic output from the iterative solver is controlled by MSGFLG or MSGLVL as follows:

MSGFLG	MSGLVL	Action
no (default)	0	Only minimal output (i.e., UIM 6447 [see "**Option Selection**" on page 108], information on whether convergence was achieved). In case of multiple loads, only if one or more loads did not converge, a final output is given for each right hand side.
yes	1	For 1 RHS: Output above + convergence ratio and norm of residual for each iteration are given. For > 1 RHS: Minimal output + final output is given for each RHS.

The parameter PREFONLY (default=0) can be set to –1, which will terminate the iterative solver after the preface, giving some helpful information in UIM 4157 (.f04 file) such as matrix size and optimal memory requirements for best performance.

Element Iterative Solver Options

The parameter ELITASPC performs the autospc in the element iterative solver (default is NO). This parameter is set using a bulk PARAM card, e.g. PARAM,ELITASPC,YES. Normally the element iterative solver does not perform an autospc function as it is usually not necessary. For solid elements, the rotational dofs are eliminated directly. If K6ROT is specified for linear shell elements, there is no issue either. But for CQUAD8 and CTRIA6 elements and possibly other special cases, the autospc function is required. The drawback of this option is that it requires the assembly of the KGG matrix which is used in the autospc and this can have a significant impact on performance. This parameter will also generate the rigid body mass properties.

In-core Frequency Response Options

The in-core frequency response is available for modal frequency response(SOL 111). USER is selected as the preconditioner option. The in-core frequency method is selected by setting SYSTEM(462)=1and iter=yes. It will be deactivated if memory is insufficient.

Incomplete Cholesky Density Options

The density of the incomplete factor is controlled by the IPAD parameter as follows:

IPAD Default	Method	ITSOPT	Model Type	Type of [A]
0	reduced incomplete Cholesky	3	all	real
2	block incomplete Cholesky	10,11	3-D	real
3	block incomplete Cholesky	10,11	2-D or mixed	real
5	block incomplete Cholesky	10,11	all	complex

Extraction Level Options for Incomplete Cholesky

IEXT Default	Reduced	Block
0	solid bodies, no rigid elements.	Requires USET and SIL
1	shells only	Heuristic block structure (default)
2	mixed including rigid elements	n/a

Recommendations

The recommended preconditioner options are as follows:

- Real symmetric positive definite systems

  Sequential : PRECOND = BIC

  Parallel : PRECOND = J

- Complex symmetric positive definite systems

  Sequential : PRECOND = BICCMPLX

- p-version analysis

  Sequential : PRECOND = PBDJ

- Direct frequency response (SOLs 8, 108)

Sequential : PRECOND = USER

- Unsymmetric or indefinite systems

  Sequential : PRECOND = J

Note: Except for SOL 108, all of the above recommended options are also the defaults.

In the case where it is known in advance that a particular model needs to be run many times (e.g. because the model needs to be modified in between, the load vector changes, or for some other reason), it is highly recommended to determine the best parameter setting for the iterative solver. To do this, one can simply vary the IPAD value from 1, ... , 4, while monitoring the EST OPTIMAL MEMORY (see Performance Diagnostics), the size of the SCRATCH FILE (see Performance Diagnostics), and the CPU time spent in SOLVIT. The defaults have been selected to give the best performance for most problems, but since the iterative solver is very sensitive to the conditioning of the matrix, they may not be best for all problems.

To find the EST OPTIMAL MEMORY without having to go through a complete run, one can set PREFONLY=–1 on the ITER Bulk Data entry. This option will give the desired information and cause NX Nastran to exit after the symbolic phase of the iterative solver.

The distributed memory parallel execution of the SOLVIT module takes advantage of the fact that the matrix multiply operation (the most time-consuming part of certain iterative strategies) is also easily executed while the system matrix resides in parts on the local memories. This is also the method of the iterative option of the STATICS supermodule.

Examples:

1. Solve [A][X]=[B] with Jacobi pre-conditioning with convergence established at 1.E-4 and maximum allowed iterations of 55 specified for the module parameters.

```
SOLVIT   A,B,,,,,,,,,/X,,//1/1.E-4/55///-1 $
```

2. Same as 1 except parameters are obtained from the SMETHOD command and ITER entry.

```
SOLVIT   A,B,,,,,,,,EDT,CASECC/X,, $
```

5.7 Global Iterative Solution Diagnostics

There are several different types of diagnostics given by the iterative solver, including:

- Accuracy diagnostics (.f06 file)
- Performance and memory diagnostics (.f04 file)

Accuracy Diagnostics

In the .f06 output file of an NX Nastran run, the following accuracy diagnostics and convergence messages may be found:

UIM 6447:
ITERATIVE SOLVER DIAGNOSTIC OUTPUT

The following example shows diagnostics for MSCFLG=yes and MSGFLG=no (default).

$$\frac{\|R\|}{\|B\|} \quad \text{and} \quad \frac{\|X(I+1)-X(I)\|}{\|X(I)\|}$$

are as given in Eq. 5-6 and Eq. 5-13, EPSILON is given in Eq. 5-24, and EXTERNAL WORK is the denominator of Eq. 5-24.

MSGFLG=yes

```
*** USER INFORMATION MESSAGE 6447 (SITDR3)
    ITERATIVE SOLVER DIAGNOSTIC OUTPUT
    IPS :  0.9999999975E-06
    BIC PRECONDITIONING
    ITERATION NUMBER       ||R|| / ||B||        ||X(I+1)-X(I)|| / ||X(I)||
            1             .1209539266E+00            .10000000E+01
            2             .2251168868E-02            .14229420E-01
            3             .8846335248E-04            .61378225E-03
            4             .1581571489E-05            .13831341E-04
            5             .5083508633E-07            .47836956E-06
*** USER INFORMATION MESSAGE 5293 (SBUT5 )
    FOR DATA BLOCK KLL
    LOAD SEQ. NO.              EPSILON              EXTERNAL WORK
               1          -4.3350458E-16          1.2827542E+05
*** USER INFORMATION MESSAGE 6448 (SITDR3)
    SOLUTION CONVERGED WITH ITERATIVE METHOD.
```

MSGFLG=no (Default)

```
*** USER INFORMATION MESSAGE 6447 (SITDR3)
    ITERATIVE SOLVER DIAGNOSTIC OUTPUT
    EPS :  0.9999999975E-06
    BIC PRECONDITIONING
    ITERATION NUMBER       ||R|| / ||B||     ||X(I+1)-X(I)|| / ||X(I)||
            5             .5083508633E-07         .47836956E-06
*** USER INFORMATION MESSAGE 5293 (SBUT5 )
    FOR DATA BLOCK KLL
    LOAD SEQ. NO.             EPSILON            EXTERNAL WORK
              1         -4.3350458E-16        1.2827542E+05
*** USER INFORMATION MESSAGE 6448 (SITDR3)
    SOLUTION CONVERGED WITH ITERATIVE METHOD.
```

This last portion of the diagnostic table prints either the last iteration number only or all the iteration numbers based on the user selection of MSGFLG (see SOLVIT OPTIONS).

MSGFLG=YES will print the information in every iteration.

MSGFLG=NO must be set by the user to suppress the information (default).

UIM 6448:
SOLUTION CONVERGED WITH ITERATIVE METHOD.

UIM 6320:
SOLUTION WAS CONTINUED BECAUSE EXTERNAL CONVERGENCE CRITERION WAS PASSED.

This message is printed if convergence is not achieved within the maximum number of iterations, even though the solution is accepted due to the energy norm check (see "Iterative Solutions" on page 90).

UFM 6288:
UNABLE TO CONVERGE WITH ITERATIVE METHOD.

UIM 5293:
FOR DATA BLOCK XX
LOADSEQ NO EPSILON EXTERNAL WORK

Performance Diagnostics

Performance diagnostics as well as information about the system matrix and the memory requirements are given in the .f04 NX Nastran output file with SIM 4157, which is also used in the direct solution. An example is shown below with interpretation of the meaning of the different variables:

```
*** SYSTEM INFORMATION MESSAGE 4157 (SITDR3)
    PARAMETERS FOR THE ITERATIVE SOLUTION WITH DATA BLOCK KLL (TYPE = RDP ) FOLLOW
       MATRIX SIZE =  134333 ROWS                  DENSITY =  .00056
     STRING LENGTH =    5.19 AVG          NUMBER OF STRINGS =    1910 K
     NONZERO TERMS =   10107 K              FULL BAND WIDTH =    6243 AVG
  MEMORY AVAILABLE =   37359 K WORDS                   IPAD =       2
     NUMBER OF RHS =       1
        BLOCK SIZE =       5              EST OPTIMAL MEMORY=   27347 K WORDS
EST MINIMUM MEMORY =    4217 K WORDS
       MEMORY USED =   36639 K WORDS       PREFACE CPU TIME =   52.49 SECONDS
      SCRATCH FILE =       0 K WORDS          AVG. CPU/ITER =   .7693 SECONDS
```

MATRIX SIZE, DENSITY, STRING LENGTH, NUMBER OF STRINGS, NONZERO TERMS, and FULL BAND WIDTH are all obvious characteristics of the system matrix.

MEMORY AVAILABLE	=	K words of memory available to the iterative solver.
IPAD	=	padding level used (see "**User Interface**" on page 101).
NUMBER OF RHS	=	number of load vectors.
BLOCK SIZE	=	block size used to block-structure the matrix.
ESTIMATED OPTIMAL MEMORY	=	memory needed by iterative solver to run in core; ensures optimal performance.
ESTIMATED MINIMUM MEMORY	=	absolute minimum memory needed for iterative solver to run; will not give best performance.
MEMORY USED	=	memory that was actually used by iterative solver.
PREFACE CPU TIME	=	time required for memory estimation, re-ordering, and preconditioner calculation.
SCRATCH FILE	=0	$\Rightarrow$ in core run
	>0	$\Rightarrow$ amount of spill

The parameters IPAD, BLOCK SIZE, SCRATCH FILE, and MEMORY USED appear only for BIC preconditioning.

For p-version analysis, the following additional information is printed in SIM 4157.

GEOMETRIC DOFs = number of rows in stiffness matrix corresponding to the geometric degrees of freedom.

HIERARCHIC DOFs = number of rows in stiffness matrix corresponding to the p degrees of freedom.

5.8 Global Iterative Solver Estimates and Requirements

Time estimates for a complete iterative solution are not given, because the number of operations per iteration is different for each preconditioner. Moreover, it is never known in advance how many iterations will be required to achieve convergence.

However, the following paragraph gives the computation time for one iteration using Jacobi and BIC preconditioning. The calculation of the computation time *t* is based on the operations necessary in each iteration using a particular preconditioner.

- Jacobi:

  –1 matrix/vector multiplication

  –1 preconditioner application r_i / a_{ii}

  –3 dot products

  $\Rightarrow t = M \bullet N_{RHS} \bullet (N^2 + N + 6 \bullet N)$

- BIC:

  –1 matrix/vector multiplication

  –1 preconditioner application $z = P_r^{-1}$

  –8 dot products

  –2 saxpy

  $\Rightarrow t = M \bullet N_{RHS} \bullet (N^2 + 2 \bullet NZ_p + 10 \bullet N)$

where:

N_{RHS} = number of right-hand sides

N = number of rows in matrix

P = preconditioner

NZ_p = number of nonzero terms in P

M = average time of one multiply-add operation

The minimum and optimal memory estimates are equally difficult to determine, since they also depend on the preconditioner used. Since BIC preconditioning is the most frequently used option and since the estimates for Jacobi preconditioning are rather straightforward, some memory estimates for those two options are given below:

- Minimum memory in words (MINMEM)
  - Jacobi:

    $5 + (5N_{RHS} + 2) \bullet N \bullet NWPT$ symmetric

    $5 + (8N_{RHS} + 4) \bullet N \bullet NWPT$ unsymmetric

  - BIC:

    $0.5 \bullet NZA$
- Optimal memory in words (OPTMEM)
  - Jacobi:

    $OPTMEM = MINMEM + (NZA / 2 + N) \bullet NWPT$
  - BIC:

    $IPAD = 2, 3 \Rightarrow OPTMEM \approx 3 \bullet NZA$

    $IPAD = 4 \Rightarrow OPTMEM \approx 4 \bullet NZA$

where:

N_{RHS} = number of right-hand sides

NWPT = number of words per term
1 for long word machines
2 for short word machines

NZA = number of nonzero terms in system matrix

N = number of rows in system matrix

Remark

Memory requirements for BIC are only very rough estimates because of the:

- Automatic reduction of IPAD value.
- Automatic decision of post-truncation of the preconditioner for well-conditioned problems.

Recommendation

To obtain reliable estimates, perform a trial run and stop after the symbolic phase by setting:

PREFONLY = –1 on ITER Bulk Data entry.

5.9 Element Iterative Solver Memory Requirements

The minimum memory required in words:

$$50 * NGRIDs + PCG + 2{,}000{,}000$$

For models made up of mostly 10 node TETRA elements:

$$PCG = 90 * NE$$

Else;

$$PCG = 200 * NE$$

where

NGRID = number of grid points

NE = number of equations to be solved - sum of the L & M sets

5.10 References

Babikov, P. & Babikova, M. ***An Improved Version of Iterative Solvers for Positive Definite Symmetric Real and Non-Hermitian Symmetric Complex Problems***. ASTE, JA-A81, INTECO 1995.

Conca, J.M.G. ***Computational Assessment of Numerical Solutions***. ISNA '92, Prague, 1992.

Efrat, I.; Tismenetsky, M. ***Parallel Iterative Solvers for Oil Reservoir Models***. IBM J. Res. Dev. 30 (2), 1986.

Hageman & Young. ***Applied Iterative Methods***. Academic Press, 1981.

Manteuffel, T. A. ***An Incomplete Factorization Technique for Positve Definite Linear Systems***, Math. of Computation, Vol 34, #150, 1980.

McCormick, Caleb W., ***Review of NASTRAN Development Relative to Efficiency of Execution***. NASTRAN: Users' Exper., pp. 7-28, September, 1973, (NASA TM X-2893).

Poschmann, P.; Komzsik, L. ***Iterative Solution Technique for Finite Element Applications***. Journal of Finite Element Analysis and Design, 19, 1993.

Poschmann, P.; Komzsik, L., Sharapov, I. ***Preconditioning Techniques for Indefinite Linear Systems***. Journal of Finite Element Analysis and Design, 21, 1997.

CHAPTER

6

Real Symmetric Eigenvalue Analysis

- Real Eigenvalue Problems
- Theory of Real Eigenvalue Analysis
- Solution Method Characteristics
- DMAP User Interface
- Method Selection
- Option Selection
- Real Symmetric Eigenvalue Diagnostics
- Real Lanczos Estimates and Requirements
- References

6.1 Real Eigenvalue Problems

The solution of the real eigenvalue problem is very important for many analysis solution sequences. The problem is numerically difficult and time consuming; therefore, NX Nastran offers methods in two different categories: the reduction (tridiagonal) method, and the iterative (Lanczos) method.

The problem of normal modes analysis is of the form:

$$[K]x = \lambda[M]x \qquad \text{Eq. 6-1}$$

The problem of the buckling analysis is stated as

$$[K]x = \lambda[K_d]x \qquad \text{Eq. 6-2}$$

where:

$[K]$ = the stiffness

$[K_d]$ = differential stiffness

$[M]$ = mass matrices

λ = eigenvalue

x = eigenvector

These problems may be solved with a reduction type method by transforming to a canonical form and reducing the whole matrix to tridiagonal form. An iterative method usually does not modify the matrices K and M; it may use their linear combination of $[K] + \lambda_s[M]$ where λ_s is a shift value. The Lanczos method, as implemented in NX Nastran, is a method using this technique. The detailed theory of real eigenvalue analysis is discussed in "**Theory of Real Eigenvalue Analysis**" on page 125.

Although the methods are mathematically interchangeable, the Lanczos method is recommended for the solution of large buckling and normal modes problems, for example, those arising in the analysis of complete vehicles. The reduction methods are useful for small normal modes problems in analysis of structural components.

6.2 Theory of Real Eigenvalue Analysis

Two main methods of real eigenvalue extraction are provided in NX Nastran in order to solve the wide variety of problems arising in finite element analysis applications:

1. Reduction (Tridiagonal) Method
2. Iterative (Lanczos) Method

In a reduction method, the matrix of coefficients is first transformed, while preserving its eigenvalues, into a special form (diagonal, tridiagonal, or upper Hessenberg) from which the eigenvalues may easily be extracted. In an iterative method, a certain number of roots are extracted at a time by iterative procedures applied to the original dynamic matrix. One of the methods used in NX Nastran is a transformation method (tridiagonal method), and the other is an iterative method (shifted block Lanczos method).

The preliminary transformation procedure of the transformation methods requires that the major share of the total effort be expended prior to the extraction of the first eigenvalue. Thus, the total effort is not strongly dependent on the number of extracted eigenvalues. In marked contrast, the total effort in the iterative methods is linearly proportional to the number of extracted eigenvalues. Therefore, it follows that the iterative methods are more efficient when only a few eigenvalues are required and less efficient when a high proportion of eigenvalues are required.

The general characteristics of the real methods used in NX Nastran are compared in **Table 6-1**. The tridiagonal method is available only for the evaluation of the vibration modes of conservative systems and not for buckling analysis due to restrictions on the matrix form. The Lanczos method is available for all vibration modes and buckling problems currently solved by NX Nastran.

It may be noted from **Table 6-1** that a narrow bandwidth and a small proportion of extracted roots tend to favor the Lanczos method. An example of such a problem is the evaluation of the lowest few modes of a structure. When the bandwidth is relatively large, and/or when a high proportion of the eigenvalues are required, the tridiagonal method is probably more efficient, assuming the problem size is not too large.

The main advantage of including two methods is to provide a backup method if one method should fail (as sometimes happens with all methods of eigenvalue extraction).

Table 6-1 Comparison of Methods of Real Eigenvalue Extraction

Characteristic/Method	Tridiagonal Method	Lanczos Method
Matrix pencil	(A, I) $A = \frac{K}{M}$ or $A = \frac{M}{K + \lambda M}$	$(M(K - \sigma M)^{-1}M, M)$ or $(K(K - \sigma K_d)^{-1}K, K)$
Restrictions on matrix character	A real, symmetric, constant $M \neq$ Singular or $K + \lambda M \neq$ Singular	M positive semidefinite or K positive semidefinite
Obtains eigenvalues in order	All at once	Several—nearest to the shift point
Takes advantage of bandwidth or sparsity	No	Yes
Number of calculations	$0(n^3)$	$0(nb^2E)$
Recommended	All modes	Few modes

where:

n = number of equations

b = semi-bandwidth or similar decomposition parameter (such as average front size)

E = number of extracted eigenvalues

Reduction (Tridiagonal) Method

The reduction method of NX Nastran offers Givens or Householder tridiagonalization options.

Transformation to Canonical Form

In the tridiagonal method when the eigenvalue problem is solved in a canonical mathematical form, a Cholesky decomposition is performed as follows:

$$[M] = [C][C]^T \quad \text{Eq. 6-3}$$

where $[C]$ is a lower triangular matrix. The procedure used to obtain the factors is described in "**Decomposition Process**" on page 50. The symmetric matrix $[A]$ is then obtained by the following transformation:

$$[C]^{-1}[K]\{u\} - \lambda[C]^{-1}[C][C]^T\{u\} = 0 \quad \text{Eq. 6-4}$$

Let

$$\{u\} = [C]^{-1,\,T}\{x\} \quad \text{Eq. 6-5}$$

where $\{x\}$ is the transformed vector. Then

$$A = [C]^{-1}[K][C]^{-1,\,T} \quad \text{Eq. 6-6}$$

After the eigenvalue of **Eq. 6-4** is found, the eigenvectors can be calculated by using **Eq. 6-5**.

Tridiagonal Method

Tridiagonal methods are particularly effective for obtaining vibration modes when all or a substantial fraction of the eigenvalues and eigenvectors of a real symmetric matrix are desired. The general restrictions on the use of the method within NX Nastran are described in **Table 6-1**. The basic steps employed in the method are as follows. First, the canonical matrix is transformed into a tridiagonal matrix

$$A \rightarrow A_t \quad \text{Eq. 6-7}$$

Next A_t is transformed to diagonal form:

$$A_t \rightarrow diag(\lambda) \quad \text{Eq. 6-8}$$

Finally, the eigenvectors are computed over a given frequency range or for a given number of eigenvalues and are converted to physical form.

Givens Tridiagonalization Method

The most recognized and stable tridiagonalization methods are the Givens and Householder methods. In the tridiagonal method of NX Nastran, both the Givens and Householder solutions are used. This section describes the Givens solutions, and the next section describes the Householder solutions.

The Givens method depends on orthogonal transformations $[T][A][T]^T$ of a symmetric matrix $[A]$. An orthogonal transformation is one whose matrix $[T]$ satisfies the following:

$$[T][T]^T = [T]^T[T] = [I] \qquad \text{Eq. 6-9}$$

The eigenvalues of a matrix are preserved under an orthogonal transformation since

$$[T]([A] - \lambda[I])[T]^T = [T][A][T]^T - \lambda[I] \qquad \text{Eq. 6-10}$$

Consequently, if det $([A] - \lambda[I])$ vanishes, then det $([T][A][T]^T - \lambda[I])$ also vanishes.

The effect of a series of orthogonal transformations on the eigenvectors of a matrix is a succession of multiplications by orthogonal matrices. If

$$[A]\{x\} = \lambda\{x\} \qquad \text{Eq. 6-11}$$

and if $[T_1], [T_2], \ldots, [T_r]$ are orthogonal matrices, then

$$[T_r][T_{r-1}]\ldots[T_2][T_1][A]\{x\} = \lambda[T_r][T_{r-1}]\ldots[T_2][T_1]\{x\} \qquad \text{Eq. 6-12}$$

Through the substitution

$$\{x\} = [T_1]^T[T_2]^T\ldots[T_{r-1}]^T[T_r]^T\{y\} \qquad \text{Eq. 6-13}$$

we obtain

$$\begin{aligned} &[T_r][T_{r-1}]\ldots[T_2][T_1][A][T_1]^T[T_2]^T\ldots[T_{r-1}]^T[T_r]^T\{y\} \\ &= \lambda[T_r][T_{r-1}]\ldots[T_2][T_1][T_1]^T[T_2]^T\ldots[T_{r-1}]^T[T_r]^T\{y\} \\ &= \lambda\{y\} \end{aligned} \qquad \text{Eq. 6-14}$$

where **Eq. 6-9** is applied repeatedly to obtain the final form. Here $\{y\}$ is an eigenvector of the transformed matrix:

$$[T_r][T_{r-1}]\ldots[T_2][T_1][A][T_1]^T[T_2]^T\ldots[T_{r-1}]^T[T_r]^T$$

and $\{x\}$ is obtained from $\{y\}$ by **Eq. 6-13**.

The Givens method uses a series of specially constructed orthogonal rotation matrices $[T_k]$; each such matrix, for given indices i, j and a given angle $\theta_{i+1,j}$, matches the identity matrix $[I]$ except for the four elements:

$$\left.\begin{aligned} t_{i+1,i+1} &= t_{j,j} = \cos(\theta_{i+1,j}) \\ t_{i+1,j} &= -t_{j,i+1} = \sin(\theta_{i+1,j}) \end{aligned}\right\} \qquad \text{Eq. 6-15}$$

The orthogonal transformation $[T][A][T]^T$ leaves all the elements of $[A]$ unchanged except those in the $(i+1)$ st and j-th rows and columns, the so-called plane of rotation. The four pivotal elements of the transformed matrix are:

$$\left.\begin{aligned} a_{i+1,i+1} &\leftarrow a_{i+1,i+1}\cos^2(\theta_{i+1,j}) + a_{j,j}\sin^2(\theta_{i+1,j}) + a_{i+1,j}\sin(2\theta_{i+1,j}) \\ a_{j,j} &\leftarrow a_{i+1,i+1}\sin^2(\theta_{i+1,j}) + a_{j,j}\cos^2(\theta_{i+1,j}) - a_{i+1,j}\sin(2\theta_{i+1,j}) \\ a_{i+1,j6y} = a_{j,i+1} &\leftarrow a_{i+1,j}\cos(2\theta_{i+1,j}) - \frac{1}{2}(a_{i+1,i+1} - a_{j,j})\sin(2\theta_{i+1,j}) \end{aligned}\right\} \quad \text{Eq. 6-16}$$

where $a_{j,j}$, etc., are elements of the untransformed matrix. The other elements of the $(i+1)$-st and j-th rows and columns of the transformed matrix are:

$$\left.\begin{aligned} a_{i+1,s} = a_{s,i+1} &\leftarrow a_{i+1,s}\cos(\theta_{i+1,j}) + a_{j,s}\sin(\theta_{i+1,j}) \\ a_{j,s} = a_{s,j} &\leftarrow -a_{i+1,s}\sin(\theta_{i+1,j}) + a_{j,s}\cos(\theta_{i+1,j}) \end{aligned}\right\} \quad \text{Eq. 6-17}$$

In the Givens method, each rotation matrix is chosen so that $a_{i,j}$ vanishes, which happens when

$$\tan(\theta_{i+1,j}) = \frac{a_{i,j}}{a_{i,i+1}} \quad \text{Eq. 6-18}$$

The calculation of $\theta_{i+1,j}$ followed by the orthogonal transformation

$$[A^{[m]}] = [T_m][A^{(m-1)}][T_m]^T \quad \text{Eq. 6-19}$$

is carried out for a sequence of iterations with $[A^{(0)}] = [A]$. The values of i used in **Eq. 6-8**, **Eq. 6-9**, **Eq. 6-10**, and **Eq. 6-11** are $1, 2, 3, \ldots, (n-1)$. For each i, a set of $(n-i-1)$ transformations is performed with j assuming the values of $i+2, i+3, \ldots, n$ before the next value of i is used. As a result, the elements in the matrix positions $(1,3), (1,4), \ldots, (1,n), (2,4), (2,5), \ldots, (2,n), \ldots, (n-2,n)$ are successively reduced to zero together with their transposes, the $(3,1), (4,1), \ldots, (n,n-2)$ elements. Thus, the set of transformations reduces the matrix to tridiagonal form.

NX Nastran employs a procedure introduced by Wilkinson (1965) in which the Givens method is modified by grouping the $(n-i-1)$ transformations together, which produces zeroes in the i-th row and column. This procedure should not be confused with the Householder method which eliminates a row and column at a time. The Wilkinson process is particularly advantageous when the matrix $[A]$ is so large that all the elements cannot be held in memory at one time. The process requires only $(n-2)$ transfers of the matrix to and from auxiliary storage instead of the $(n-1)(n-2)/2$ transfers required by the unmodified Givens method. This modified Givens method requires $4n$ memory locations for working space that are divided into four groups of n storage locations each. The first $(i-1)$ rows and columns play no part in the i-th major step. This step has five substeps as follows:

1. The *i*-th rows of $[A]$ are transferred to the set of memory locations in group 1.
2. The values of $\cos\theta_{i+1,\,i+2}$, $\sin\theta_{i+1,\,i+2}$, …, $\cos\theta_{i+1,\,n}$, $\sin\theta_{i+1,\,n}$ are computed successively from the following:

$$\cos\theta_{i+1,\,j} = \frac{a_{i,\,j+1}^{(j-1)}}{\sqrt{(a_{i,\,i+1}^{(j-1)})^2 + a_{i,\,j-1}^2}}$$
$$\sin\theta_{i+1,\,j} = \frac{a_{i,\,i}}{\sqrt{(a_{i,\,i+1}^{(j-1)})^2 + a_{i,\,j}^2}} \qquad \text{Eq. 6-20}$$

   where the superscripted term is computed by the following:

$$a_{i,\,i+1}^{(j-1)} = \sqrt{(a_{i,\,i+1}^{(j-2)})^2 + a_{i,\,j-1}^2} \qquad \text{Eq. 6-21}$$

   and the starting value for $j = i+2$ is as follows:

$$a_{i,\,i+1}^{(i+1)} = a_{i,\,i+1} \qquad \text{Eq. 6-22}$$

   The $\cos\theta_{i,\,j}$ may be overwritten on those elements of the untransformed matrix $a_{i,\,j}$ which are no longer required, and the $\sin\theta_{i+1,\,j}$ is stored in the group 2 storage locations.
3. The $(i+1)$ st row of $[A]$ is transferred to the group 3 storage locations. Only those elements on and above the diagonal are used in this and succeeding rows. For $k = i+2, i+3, \ldots, n$, in turn, the operations in substeps 4 and 5 are carried out.
4. The *k*-th row $[A]$ is transferred to the group 4 storage locations. The elements $a_{i+1,\,i+1}$, $a_{i+1,\,k}$, and $a_{k,\,k}$ are subjected to the row and column operations involving $\cos\theta_{i,\,k}$ and $\sin\theta_{i+1,\,k}$. For $i = k+1, k+2, \ldots, n$ in turn, the part of the row transformation involving $\cos\theta_{i+1,\,k}$ and $\sin\theta_{i+1,\,k}$ is performed on a_{i+1} and a_k. At this point, all the transformations involving the *i*-th major step were performed on all the elements of row k and on the elements $i+2$, $i+3$, …, n of row $(i+1)$.
5. The completed *k*-th row is transferred to auxiliary storage.

When substeps 4 and 5 are complete for all appropriate values of k, the work on row $(i+1)$ is also complete. The values of $\cos\theta_{i+1,\,k}$ and $\sin\theta_{i+1,\,k}$ for $k = i+2$, $i+3$, …, n are transferred to the auxiliary storage, and row $(i+1)$ is transferred to the group 1 core storage locations. Since the $(i+1)$ row plays the same part in the next major step as in the *i*-th in the step just described, then the process is ready for substep 2 in the

next major step. In fact, substep 1 is only required in the first major step because the appropriate row is already in the storage locations of group 1 while performing subsequent steps.

Householder Tridiagonalization Method

The Householder tridiagonalization method uses the following transformation:

$$A_r = P_r A_{r-1} P_r \qquad \text{Eq. 6-23}$$

where:

$A_0 = A$

$P_r = I - 2\, w_r\, w_r^T$ symmetric, orthogonal

$w_r^T w_r = 1$

The elements of w_r are chosen so that A_r has all zeroes in the r-th row except for the three diagonal positions. The matrix A_{r-1} can be partitioned as:

$$A_{r-1} = \begin{matrix} r \\ n-r \end{matrix} \left\{ \begin{matrix} \left[\begin{matrix} x & x & & \\ x & x & x & \\ & x & x & x\ x\ x \end{matrix}\right] \\ \left[\begin{matrix} & x & x\ x\ x \\ & x & x\ x\ x \\ & x & x\ x\ x \end{matrix}\right] \end{matrix} \right. = \left[\begin{array}{c|c} C_{r-1} & \begin{matrix} 0 \\ b_{r-1}^T \end{matrix} \\ \hline \begin{matrix} 0 & b_{r-1} \end{matrix} & B_{r-1} \end{array}\right]$$

where:

C_{r-1} = a tridiagonal matrix of order r (partial result)

B_{r-1} = a square matrix of order $n - r$ (part of original matrix)

b_{r-1} = a vector having $n - r$ components

The P_r transformation matrix can be partitioned similarly as

$$P_r = \begin{matrix} r \\ n-r \end{matrix} \left\{ \begin{matrix} \left[\begin{matrix} I & 0 \end{matrix}\right] \\ \left[\begin{matrix} 0 & Q_r \end{matrix}\right] \end{matrix} \right. = \left[\begin{matrix} I & 0 \\ 0 & I - 2v_r v_r^T \end{matrix}\right]$$

where v_r = a vector of order $n-r$

By executing the multiplication given in the right-hand side of **Eq. 6-23**, the following is obtained:

$$A_{r-1} = \begin{matrix} r \\ n-r \end{matrix} \left\{ \begin{matrix} \left[\begin{matrix} & & 0 \\ & C_{r-1} & c_r^T \end{matrix}\right] \\ \left[\begin{matrix} 0 & c_r & Q_r B_{r-1} Q_r^T \end{matrix}\right] \end{matrix} \right.$$

where $c_r = Q_r b_{r-1}$.

If we have chosen v_r so that c_r is null except for its first component, then the first $(r+1)$ rows and columns of A_r are tridiagonal.

An algorithm formulation can be developed by writing P_r in the following form:

$$P_r = I - 2w_r w_r^T \qquad \text{Eq. 6-24}$$

where w_r is a vector of order n with zeroes as the first r elements. Furthermore, we can substitute $w_r = u_r/(2K_r)$:

$$P_r = I - \frac{u_r u_r^T}{2K_r^2} \qquad \text{Eq. 6-25}$$

where:

$$u_{ir} = 0, \quad i = 1, 2, \ldots, r$$

$$u_{r+1,\,r} = a_{r,\,r+1} \pm S_r$$

$$u_{ir} = a_{ri}, \quad i = r+2, \ldots, n$$

$$S_r^2 = \sum_{i\,=\,r+1}^{n} a_{ri}^2$$

$$2K_r^2 = S_r^2 \pm a_{r,\,r+1}\, S_r$$

and a_{ij} is the (i, j) element of A. Substituting **Eq. 6-25** into **Eq. 6-23**, the following equation results:

$$A_r = \left(I - \frac{u_r u_r^T}{2K_r^2}\right) A_{r-1} \left(I - \frac{u_r u_r^T}{2K_r^2}\right) \qquad \text{Eq. 6-26}$$

with a new notation as follows:

$$p_r = \frac{A_{r-1} u_r}{2K_r^2} \qquad \text{Eq. 6-27}$$

Now the form becomes the following:

$$A_r = A_{r-1} - u_r p_r^T - p_r u_r^T + \frac{u_r (u_r^T p_r) u_r^T}{2K_r^2} \qquad \text{Eq. 6-28}$$

By introducing one more intermediate element

$$q_r = p_r - \frac{1}{2} u_r \left(\frac{u_r^T p_r}{2K_r^2}\right) \qquad \text{Eq. 6-29}$$

the final algorithmic form is obtained

$$A_r = A_{r-1} - u_r q_r^T - q_r u_r^T \quad \text{Eq. 6-30}$$

which takes advantage of the symmetry.

Modified Tridiagonal Methods

The modified tridiagonal methods (i.e., modified Givens or Householder) are used if the mass matrix M is singular. These methods have a different transformation scheme to obtain the canonical form. First, perform a Cholesky decomposition on a shifted matrix as follows:

$$([K] + \lambda_s[M]) = [C][C]^T \quad \text{Eq. 6-31}$$

where λ_s is a positive shift value obtained, as shown below, from the diagonal terms of $[K]$ and $[M]$ matrices. The new form of **Eq. 6-1** is now

$$[A] - \bar{\lambda} I\{x\} = 0 \quad \text{Eq. 6-32}$$

where:

$$\bar{\lambda} = \frac{1}{\lambda + \lambda_s}$$

$$[A] = [C]^{-1}[M][C]^{-1,\,T}$$

The λ_s shift value is calculated as:

$$\lambda_s = \frac{1}{n^{1/2} \cdot \sum_{i=1}^{n} \frac{M_{ii}}{K_{ii}}}$$

where M_{ii} and K_{ii} are the diagonal elements of M and K respectively. If $M_{ii} = 0$ or $K_{ii} = 0$ or $M_{ii}/K_{ii} > 10^8$, the term is omitted from the summation. If all terms are omitted, λ_s is set to 0.001. If the λ_s value calculated does not result in a stable Cholesky decomposition, the value is slightly perturbed, and the decomposition is repeated up to 2 times. After three unsuccessful shifts, a fatal error is given.

The details of the above equation can be seen by rearranging **Eq. 6-7** as follows:

$$([K] + \lambda_s[M] - (\lambda + \lambda_s)[M])\{u\} = 0 \quad \text{Eq. 6-33}$$

Then by premultiplying $-1/(\lambda + \lambda_s)$ and substituting **Eq. 6-31** the following is obtained:

$$\left([M] - \frac{[C][C]^T}{\lambda + \lambda_s} \right) \{u\} = 0 \qquad \text{Eq. 6-34}$$

Finally, premultiplying by $[C]^{-1}$ and substituting the term

$$\{x\} = [C]^T \{u\} \qquad \text{Eq. 6-35}$$

gives the **Eq. 6-32**.

QR Method of Eigenvalue Extraction

In the tridiagonal methods of Givens or Householder, NX Nastran employs the QR iteration of Francis (1962), which produces a series of orthogonal transformations $[A_{r+1}] = [Q_r]^T [A_r][Q_r]$ where $[A_r]$ is factored into the product $[Q_r][R_r]$, with $[R_r]$ an upper triangular matrix. Thus,

$$[A_r] = [Q_r][R_r] \qquad \text{Eq. 6-36}$$

and

$$\begin{aligned} [A_{r+1}] &= [Q_r]^T [A_r][Q_r] \\ &= [Q_r]^T [Q_r][R_r][Q_r] \end{aligned}$$

Now $[Q_r]^T [Q_r] = [I]$ by virtue of the orthogonality property; therefore,

$$[A_{r+1}] = [R_r][Q_r] \qquad \text{Eq. 6-37}$$

It follows that $[A_{r+1}]$ is determined from $[A_r]$ by performing in succession the decomposition given by **Eq. 6-36** and multiplication. Francis has shown that if a matrix $[A] = [A_1]$ is nonsingular, then $[A_r]$ approaches an upper triangular matrix as $r \to \infty$. Since eigenvalues are preserved under orthogonal transformation, it follows that the diagonal elements of the limiting matrix are the eigenvalues of the original matrix $[A]$. Although the method can be applied to any matrix, it is particularly suitable for tridiagonal matrices because the bandwidth of the matrix can be preserved as will be shown. In the case where $[A]$ is symmetric, the matrix $[A_r]$ converges to diagonal form as $r \to \infty$.

Even though the upper triangular matrix $[R_r]$ and the orthogonal matrix $[Q_r]$ are unique up to sign, several methods are available for performing the decomposition. In the method of calculation devised by Francis, $[Q_r]$ is expressed as a product of $(n-1)$ elementary rotation matrices as follows, where n is the order of $[A_r]$:

$$[Q_r] = [T^{(1)}][T^{(2)}] \ldots [T^{(n-1)}] \qquad \text{Eq. 6-38}$$

The nonzero elements of the j-th elementary rotation matrix are the following:

$$\left.\begin{aligned} t^{(j)}_{j,\,j} &= t^{(j)}_{j+1,\,j+1} = c_j \\ t^{(j)}_{j+1,\,j} &= -t^{(j)}_{j,\,j+1} = s_j \\ t^{(j)}_{k,\,k} &= 1 \qquad\qquad k \neq j,\, j+1 \end{aligned}\right\} \qquad \text{Eq. 6-39}$$

The manner in which the c_j and s_j coefficients are obtained from the elements of $[A_r]$ are shown later. From the orthogonality property

$$\begin{aligned} [R_r] &= [Q_r]^{-1}[A_r] = [Q_r]^T[A_r] \\ &= [T^{(n-1)}]^T[T^{(n-2)}]^T \ldots [T^{(2)}]^T[T^{(1)}]^T[A_r] \end{aligned} \qquad \text{Eq. 6-40}$$

we can define the nonzero elements of $[A_r]$, $[A_{r+1}]$ and $[R_r]$ as follows:

$$[A_r] = \begin{bmatrix} a_1 & b_1 & & \searrow & 0 \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & & \\ & \searrow & b_{n-1} & a_{n-1} & b_n \\ 0 & & & b_n & a_n \end{bmatrix} \qquad \text{Eq. 6-41}$$

$$[A_{r+1}] = \begin{bmatrix} \bar{a}_1 & \bar{b}_1 & & \searrow & 0 \\ \bar{b}_2 & \bar{a}_2 & \bar{b}_3 & & \\ & \ddots & \ddots & & \\ & \searrow & \bar{b}_{n-1} & \bar{a}_{n-1} & \bar{b}_n \\ 0 & & & \bar{b}_n & \bar{a}_n \end{bmatrix} \qquad \text{Eq. 6-42}$$

and

$$[R_r] = \begin{bmatrix} r_1 & q_1 & t_1 & & \\ & r_2 & q_2 & t_2 & \\ & & \ddots & \ddots & \ddots \\ & & & r_{n-1} & q_{n-1} \\ 0 & & & & r_n \end{bmatrix} \qquad \text{Eq. 6-43}$$

The coefficients of the elementary rotation matrices are selected to reduce the subdiagonal terms of $[R_r]$ to zero. Specifically,

$$\left.\begin{aligned} s_j &= \frac{b_{j+1}}{(p_j^2 + b_{j+1}^2)^{1/2}} \\ c_j &= \frac{p_j}{(p_j^2 + b_{j+1}^2)^{1/2}} \end{aligned}\right\} \qquad j = 1, 2, \ldots, n-1 \qquad \text{Eq. 6-44}$$

where

$$\begin{aligned} p_1 &= a_1 \\ p_2 &= c_1 a_2 - s_1 b_2 \\ p_j &= c_{j-1} a_j - s_{j-1} c_{j-2} b_j \qquad j = 3, 4, \ldots, n-1 \end{aligned} \qquad \text{Eq. 6-45}$$

Substitution yields the elements of $[R_r]$ as follows:

$$\left.\begin{aligned} r_j &= c_j p_j + s_j b_{j+1} \qquad j = 1, 2, \ldots, n-1 \\ r_n &= p_n \\ q_1 &= c_1 b_2 + s_1 a_2 \\ q_j &= s_j a_{j+1} + c_j c_{j-1} b_{j+1} \qquad j = 2, 3, \ldots, n-1 \\ t_j &= s_j b_{j+2} \qquad j = 1, 2, \ldots, n-2 \end{aligned}\right\} \qquad \text{Eq. 6-46}$$

From **Eq. 6-37**, the elements of A_{r+1} are the following:

$$\left.\begin{aligned} \bar{a}_1 &= c_1 r_1 + s_1 q_1 \\ \bar{a}_j &= c_{j-1} c_j r_j + s_j q_j \qquad j = 2, 3, \ldots, n-1 \\ \bar{a}_n &= c_{n-1} r_n \\ \bar{b}_{j+1} &= s_j r_{j-1} \qquad j = 1, 2, \ldots, n-2 \end{aligned}\right\} \quad \text{Eq. 6-47}$$

NX Nastran uses a variation of Francis's original method that avoids the calculation of square roots. This variation uses the following equations in place of **Eq. 6-47** as follows:

$$a_j = (1 + s_j^2) g_j + s_j^2 a_{j+1} \qquad j = 1, 2, \ldots, n-1 \quad \text{Eq. 6-48}$$

$$\left.\begin{aligned} \bar{a}_n &= g_n \\ \bar{b}_{j+1}^2 &= s_j^2 (p_{j+1}^2 + b_{j+2}^2) \qquad j = 1, 2, \ldots, n-2 \\ \bar{b}_n^2 &= s_{n-1}^2 p_n^2 \end{aligned}\right\} \quad \text{Eq. 6-49}$$

where

$$\left.\begin{aligned} g_1 &= a_1 \\ g_j &= c_{j-1} p_j = a_j - s_{j-1}^2 (a_j + g_{j-1}) \qquad j = 2, 3, \ldots, n \\ g_1^2 &= a_1^2 \\ g_1^2 &= \begin{cases} \dfrac{g_j^2}{c_{j-1}^2} & \text{if } c_{j-1} \neq 0 \qquad j = 2, 3, \ldots, n \\ c_{j-2}^2 b_j^2 & \text{if } c_{j-1} = 0 \qquad j = 2, 3, \ldots, n \end{cases} \end{aligned}\right\} \quad \text{Eq. 6-50}$$

The reason that the use of **Eq. 6-48**, **Eq. 6-49**, and **Eq. 6-50** in place of **Eq. 6-47** avoids the calculation of square roots can best be seen by considering the terms input to and produced by these equations. For **Eq. 6-47**, the input terms consist of the elements of $[A_r]$ (which are $\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_n$) and $\bar{b}_2, \bar{b}_3, \ldots, \bar{b}_n$ (which are the elements of $[A_{r+1}]$). This completes one iteration step but involves the calculation of square roots. However, for **Eq. 6-48**, **Eq. 6-49**, and **Eq. 6-50**, the input terms consist of $a_1, a_2, \ldots, a_n$ and $b_1^2, b_2^2, \ldots, b_n^2$. The data produced is $\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_n$ and $\bar{b}_2, \bar{b}_3, \ldots, \bar{b}_n$, which serve as the input to the next iteration step. No square roots need to be computed here.

Convergence of the tridiagonal matrix to a diagonal form is speeded up by origin shifting. NX Nastran employs a procedure suggested by Wilkinson's quadratic shift which has shown that when the eigenvalues are real, the best shift strategy is to subtract a_n from each diagonal element of $[A]$ and thereby reduce each eigenvalue by a_n.

Another device, useful in speeding up the determination of eigenvalues of tridiagonal matrices, takes advantage of zeroes that may occur in the off-diagonal rows. Let $[A_r]$ matrix be represented in partitioned form as follows:

$$\left[\begin{array}{ccc|ccc|cc}
a_1 & b_1 & & & & & & \\
b_1 & a_2 & b_2 & & & & & \\
 & \ddots & \ddots & & & & & \\
 & & a_{j-2} & b_{j-2} & & & & \\
 & & b_{j-2} & a_{j-1} & & & & \\
\hline
 & & & a_j & b_j & & & \\
 & & & b_j & a_{j+1} & b_{j+1} & & \\
 & & & & \ddots & \ddots & & \\
 & & & & b_{m-2} & a_{m-1} & b_{m-1} & \\
 & & & & & b_{m-1} & a_m & \\
\hline
 & & & & & & a_{m+1} & \\
 & & & & & & & \ddots \\
 & & & & & & & a_n
\end{array}\right] \qquad \text{Eq. 6-51}$$

In this matrix, the diagonal terms in the lower right-hand partition are eigenvalues that were determined in previous iterations. The *j*-th is the next lowest row in which the off-diagonal term b_{j-1} is zero thereby uncoupling the equations in the first $j-1$ rows from the rest. As a result, the eigenvalues of the matrix in the central block may be obtained separately. Other uncoupled blocks may be found in the upper left partition.

The iteration described by **Eq. 6-48** and **Eq. 6-49** is continued until $(\bar{b}_{m-1})^2$ decreases to a satisfactory level of accuracy so that $\bar{a}_m$ may be accepted as an eigenvalue of the shifted matrix. $(\bar{b}_{m-1})^2$ must be negligible compared to $(\bar{a}_m)^2$, that is, $(\bar{b}_{m-1})^2 + (\bar{a}_m)^2$ must approximately equal $(\bar{a}_m)^2$. Then $(\bar{a}_m)$ is transferred to the lower partition, and the process is continued until all the eigenvalues of the partitioned matrix are extracted.

Computation of Eigenvectors

The eigenvector corresponding to any eigenvalue λ_i of the tridiagonal matrix may be determined by solving $n-1$ of the following equations:

$$\begin{aligned}(a_1 - \lambda_i)x_1 + b_2\, x_2 &= 0\\ b_2\, x_1 + (a_2 - \lambda_i)x_2 + b_3\, x_3 &= 0\\ &\vdots\\ b_{n-1}\, x_{n-1} + (a_{n-1} - \lambda_i)x_{n-1} + b_n\, x_n &= 0\\ b_n\, x_{n-1} + (a_n - \lambda_i)x_n &= 0\end{aligned} \qquad \text{Eq. 6-52}$$

If the first $(n-1)$ equations are used, the solution is obtained by taking x_1 to be unity and substituting in the equations to obtain values of $x_2, x_3, \ldots, x_n$. Wilkinson (1958) shows that this method is unstable and suggests the following approach, which is used in NX Nastran.

The tridiagonal matrix $[A-\lambda_i I]$ is factored into the product of a lower unit triangle $[L_i]$ and an upper triangle $[U_i]$. Partial pivoting is used in this decomposition; i.e., the pivotal row at each stage is selected to be the equation with the largest coefficient of the variable being eliminated. An approximation to the eigenvector $[\phi_i]$ is then obtained from the solution of the following equation:

$$[U_i][\phi_i] = [C] \qquad \text{Eq. 6-53}$$

where $[C]$ is randomly selected. The solution is easily obtained by backward substitution because $[U_i]$ has the form below:

$$[U_i] = \begin{bmatrix} p_1 & q_1 & r_1 & & & & \\ & p_2 & q_2 & r_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & p_{n-2} & q_{n-2} & r_{n-2} & \\ & & & & p_{n-1} & q_{n-1} & \\ & & & & & p_n & \end{bmatrix} \qquad \text{Eq. 6-54}$$

An improved solution is obtained by repeated application of **Eq. 6-53** using the current estimate of $[\phi_i]$ on the right-hand side. Thus,

$$[U_i]\left\{\phi_i^{(n)}\right\} = \left\{\phi_i^{(n-1)}\right\} \qquad \text{Eq. 6-55}$$

where

$$\left\{ \phi_1^0 \right\} = [C]$$

Wilkinson showed that, if the computed eigenvalue λ_i is a close approximation to the true eigenvalue, convergence is so rapid that not more than two iterations are required. The applied test is for the maximum component of the eigenvector to remain unchanged (to single precision accuracy) in one iteration. The initial vector $[C]$ is chosen so that each element is unity.

In the case of a multiple eigenvalue, the above method gives one eigenvector $\{\phi_1\}$. If you start with any initial vector $\{b\}$ orthogonal to $\{\phi_1\}$ and apply the previous algorithm, convergence to the other eigenvector $\{\phi_2\}$ results. The following procedure is used in NX Nastran to compute eigenvectors associated with multiple eigenvalues. If eigenvectors $\{\phi_1\}, \{\phi_2\}, \ldots, \{\phi_m\}$ with elements $\{\phi_d\} = \{a_{1s}, a_{2s}, \ldots, a_{ns}\}^T$ are obtained, an initial vector $\{b\}$ orthogonal to each of these eigenvectors is obtained by taking the components $b_{m+1}, b_{m+2}, \ldots, b_n$ as unity and solving the simultaneous equations as follows:

$$\left.\begin{aligned} b_1 a_{11} + b_2 a_{21} + \ldots + b_m a_{m1} &= -\sum_{i=m+1}^{n} (a_{i1}) \\ b_1 a_{12} + b_2 a_{22} + \ldots + b_m a_{m2} &= -\sum_{i=m+1}^{n} (a_{i2}) \\ &\vdots \\ b_1 a_{1m} + b_{2m} a_{2m} + \ldots + b_m a_{mm} &= -\sum_{i=m+1}^{n} (a_{im}) \end{aligned}\right\} \quad \text{Eq. 6-56}$$

Accumulated round-off errors result in the computed multiple eigenvectors since they are not exactly orthogonal to one another. The following Gram-Schmidt algorithm is used to produce an orthogonal set of k eigenvectors $\{y_s\}$ from the almost orthogonal set $\{x_s\}$. For $s = 1$, select as follows:

$$\{y_1\} = \frac{\{x_1\}}{\|\{x_1\}\|} \quad \text{Eq. 6-57}$$

Then for $1 < s \le k$, calculate as follows:

$$\{z_s\} = \{x_s\} - \sum_{t=1}^{s-1} (\{x_s\}^T \{y_t\})\{y_t\}$$
$$\{y_t\} = \frac{|z_t|}{\|\{z_t\}\|} \qquad \text{Eq. 6-58}$$

where $\|\{z_t\}\|$ denotes the Euclidean norm

$$\sqrt{z_{t1}^2 + z_{t2}^2 + \ldots + z_{tn}^2}$$

of vector $\{z_{t1}, z_{t2}, \ldots, z_{tn}\}^T$ and $\{x_s\}^T\{y_t\}$ is a scalar product of the vectors $\{x_s\}$ and $\{y_t\}$.

When all the eigenvectors of the tridiagonal matrix are extracted, the back transformations are carried out to obtain the eigenvectors of the original matrix $[A]$. Finally, for all of the tridiagonal methods, the eigenvectors are given one last stage of orthogonalization as follows. First compute:

$$M_{modal} = \phi^T M \phi$$

Then, find L, such that

$$L^T L = M_{modal}$$

by Cholesky factorization. Finally, find the reorthogonalized eigenvectors $\bar{\phi}$ by a forward solution pass of $L \bullet \bar{\phi}^T = \phi^T$. It can be shown that this operation is equivalent to orthogonalizing the second vector with respect to the first, orthogonalizing the third with respect to the purified second and first vector, … , and orthogonalizing the last vector with respect to all of the other purified vectors. If M_{modal} is poorly conditioned with respect to the Cholesky decomposition, a fatal error message is produced.

Shared Memory Parallel Householder Method

The most time-consuming part of the reduction type methods is the reduction to tridiagonal form. Besides being more easily vectorizable, the Householder method also lends itself easily to parallelization. The computer science aspects of the parallel Householder method (i.e., creating subtasks, waiting, etc.) are similar to those of the parallel methods mentioned previously and are not detailed here. The necessary mathematical details follow.

A logical suggestion is to "aggregate" the Householder transformations, i.e., work in column blocks instead of columns. For example, by observing the first k steps, the following can be written:

$$[A]_k = ([P]_k[P]_{k-1}, \ldots, [P]_1)A([P]_k[P]_{k-1}, \ldots, [P]_1)^T$$
$$= [Q]_k[A][Q]_k^T$$

where:

$$[Q]_k = I + [W]_k[Y]_k^T$$

$[W]$ and $[Y] = n \times k$ matrices (WY representation)

Completing the process in k-size blocks produces the following:

$$[A]_j = [Q]_j[A][Q]_j^T \qquad j = 1, 2, \ldots, n/k$$

where $[Q]_j = [P]_{jk}[P]_{jk-1}\ldots,[P_{jk-k+1}]$

Before the j-th block step, the structure of A is as follows:

$[A^j] =$

$[C_{j-1}]$	$[B_{j-1}]$	$[A_1^{(j-1)}]$
$(j-1) \bullet k$	k	$n-(jk)$

where:

C_{j-1} = tridiagonal matrix of order $n \cdot (j-1)k$

B_{j-1} = general matrix of order $n \cdot k$

$A^{(j-1)}$ = general matrix of order $n \cdot n - jk$

The process is then to generate $I + W_j Y_j^T$ so that B_{j-1} assumes tridiagonal form and afterwards update the remainder of the matrix as shown:

$$A^{(j)} = (I + W_j Y_j^T) A^{(j-1)} (I + W_j Y_j^T)^T$$

This block update can be parallelized and forms the parallel portion of the NX Nastran Householder code. It is intended for shared memory parallel computers. It does not run on distributed memory computers. A similar approach can also be used in the reduction to Hessenberg form, which is one of NX Nastran's complex eigenvalue methods. However, this parallelized Hessenberg approach is not implemented as yet.

Real Symmetric Lanczos Method

The basic Lanczos recurrence (Lanczos, 1950) is a transformation process to tridiagonal form. However, the Lanczos algorithm truncates the tridiagonalization process and provides approximations to the eigenpairs (eigenvalues and eigenvectors) of the original matrix. The block representation increases performance in general and reliability on problems with multiple roots. The matrices used in the Lanczos method are specifically selected to allow the best possible formulation of the Lanczos iteration and are described in **Table 6-1** of "**Theory of Real Eigenvalue Analysis**" on page 125.

Basic Lanczos Recurrence

The basic Lanczos algorithm solves the following problem:

$$Ax = \lambda x \qquad \text{Eq. 6-59}$$

and can be formulated as follows:

1. Initialization
   a. A starting vector q_1 is chosen with $\|q_1\| = 1$.
   b. Initialize the scalar $\beta_1 = 0$ and vector $q_0 = 0$.
2. Iteration

For $j = 1, 2, 3, \ldots$, iterate as below until convergence:

$$\alpha_j = q_j^T A q_j$$

$$r_{j+1} = A q_j - \alpha_j q_j - \beta_j q_{j-1}$$

$$\beta_{j+1} = \|r_{j+1}\|$$

$$q_{j+1} = r_{j+1} / \beta_{j+1}$$

If this algorithm is carried out without round-off errors, the vectors $q_1, q_2, q_3, \ldots$, are orthonormal. These vectors can be considered to be the columns of an orthogonal matrix $Q_j = (q_1, q_2, \ldots, q_j)$. The scalars α_j and β_j can be combined in a tridiagonal matrix T_j as follows:

$$T_j = \begin{bmatrix} \alpha_1 & \beta_2 & & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \beta_i & \alpha_i & \beta_{i+1} & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & \beta_j & \\ & & & & & \beta_j & \alpha_j \end{bmatrix}$$

With this notation, the first j steps of the Lanczos algorithm can be combined in matrix form as follows:

$$A Q_j - Q_j T_j = \beta_{j+1} q_{j+1} e_j^T \qquad \text{Eq. 6-60}$$

where the vector e_j is the unit vector with zero entries except for the *j*-th row which holds the value of one.

This mathematical presentation is not sufficient for a practical implementation of the Lanczos algorithm that is capable of coping with finite precision arithmetic. The computation of the Lanczos loop can be carried out in many mathematically equivalent ways, one of which is the best for finite arithmetic, while several others do not work in finite arithmetic (Grimes, R.G., et al.). Even with the recurrence written in the proper form, round-off errors can cause the algorithm to behave quite differently from the ideal description given above. More details on the effects of round-off errors are given in the section on preserving orthogonality.

The Lanczos algorithm is valuable for solving sparse eigenvalue problems because it needs only a partial tridiagonalization and does not actually modify the matrix A. The matrix A enters the algorithm only in the formation of the matrix vector product Aq. In practice, this means that the sparsity of A can be exploited in a natural way since all that is needed is an efficient subroutine to compute Av from v.

The eigenvalues and corresponding eigenvectors of A are computed from those of T. Let θ and s be an eigenpair of the tridiagonal matrix T_j; that is, let

$$T_j s = \theta s \qquad \text{Eq. 6-61}$$

Then θ and y with $y = Q_j s$ can be considered an approximate eigenpair for the original problem. Note that in a typical application, the order of matrix A may be in the tens of thousands, whereas the order of T_j is about 20 to 30. Hence, it is much easier to solve the eigenvalue problem of **Eq. 6-61** than of **Eq. 6-60**. But the question is: How good is the approximate eigenpair θ, y, or more directly, when should the Lanczos recurrence be stopped so that the eigenpairs are satisfactorily accurate?

The correct answer requires the norm of the residual for the eigenpair $\|Ay - \theta y\|$, which seems to require the full computation of a candidate eigenpair. In fact, this residual norm can be obtained from **Eq. 6-61** by observing that

$$Ay - \theta y = \beta_{j+1} q_{j+1} s_j \qquad \text{Eq. 6-62}$$

where s_j is the j-th or bottom element of the eigenvector s. Hence, the norm of the residual is given by $\beta_{j+1} s_j$ and the quantity can be computed without computing the eigenvector y explicitly. A small extra effort allows the convergence of the eigenpairs to be monitored and terminates the Lanczos loop whenever the required number of sufficiently accurate eigenpairs is found. The eigenvector y is computed only once at the end.

Why should it be expected that some of the eigenvalues of the tridiagonal matrix T_j converge quickly to the eigenvalues of A? This question is answered by some intricate mathematical theorems (see Parlett for an overview) which guarantee that, under reasonable assumptions on the properties of the starting vector, approximations to some of the extreme eigenvalues appear in the tridiagonal matrix very quickly. Two factors can impede this rapid convergence. If the starting vector has no components in the direction of one of the eigenvectors, the convergence of this eigenvector is delayed. A cluster of poorly separated eigenvalues also causes slow convergence.

Shifted Algorithm

The basic Lanczos algorithm as discussed in the previous section must be modified in two respects to solve the practical vibration problem. The vibration problem is a symmetric generalized eigenvalue problem using the following form:

$$Kx = \lambda Mx \qquad \text{Eq. 6-63}$$

where K is a symmetric matrix, M is a symmetric positive semidefinite matrix, and λ is the eigenvalue. If M is positive definite, the Cholesky factorization of M can be computed and **Eq. 6-63** can be reduced to an ordinary eigenvalue equation. However, there are considerably more difficulties when M is only semidefinite (nonnegative definite). The implementation of the Lanczos algorithm in NX Nastran avoids the Cholesky factorization of M to use the more general form of the problem.

The typical eigenvalue problem is to compute the smallest (closest to zero) eigenvalues and eigenvectors of **Eq. 6-63**. These eigenvalues are usually clustered closely together when compared to the largest eigenvalues. The Lanczos algorithm applied directly to **Eq. 6-63** yields precise approximations to these large, well-separated, but uninteresting eigenvalues, and poor approximations to the small, clustered, and interesting eigenvalues. In order to overcome this convergence difficulty, the Lanczos algorithm is applied to the following shifted and inverted eigenvalue problem:

$$M(K - \sigma M)^{-1} Mx = \frac{1}{\lambda - \sigma} Mx \qquad \text{Eq. 6-64}$$

In **Eq. 6-64** the shift σ is chosen close to the eigenvalues of interest. This formulation of the shifted and inverted eigenvalue problem is only one of several possible ones. **Eq. 6-64** is the preferred choice for vibration problems because of its improved rate of convergence to the desired eigenvalues and the fact that the eigenvectors of **Eq. 6-64** are also eigenvectors of **Eq. 6-63**.

The relationship between the application of the Lanczos algorithm to **Eq. 6-63** and **Eq. 6-64** is comparable to the relationship between the power method and the inverse power method (inverse iteration). The choice of shift in the inverse power method directs the convergence toward the eigenpair closest to the shift. For well-chosen shifts, the convergence can be very rapid. The same is true for the shifted and inverted Lanczos algorithm. A properly chosen shift guarantees rapid convergence not only to the closest eigenpair but also to a whole group of eigenpairs in the vicinity of the shift. The cost of this accelerated convergence towards the desirable eigenvalues is the factorization of $(K - \sigma M)$ which, for large matrices, can be the dominating cost in either algorithm. The computational superiority of the Lanczos algorithm derives from its efficient use of this expensive factorization. Even a relatively short run of the shifted and inverted Lanczos algorithm can extract many eigenvalues using only one factorization.

The Lanczos algorithm applied to the vibration problem of **Eq. 6-64** can be formulated as follows:

1. Initialization
   a. A starting vector r_0 is chosen.

b. $r_1 = (K - \sigma M)^{-1} M r_0$ is computed.

c. The mass-normalized vector $q_1 = r_1 / (r_1^T M r_1)^{1/2}$ is computed.

d. $\beta_1 = 0$ and $q_0 = 0$ are initialized.

2. Iteration

For $j = 1, 2, 3, \ldots$ the following iteration is performed until convergence occurs:

$$\alpha_j = q_j^T M (K - \sigma M)^{-1} M q_j$$

$$r_{j+1} = (K - \sigma M)^{-1} M q_j - \alpha_j q_j - \beta_j q_{j-1}$$

$$\beta_{j+1} = (r_{j+1}^T M r_{j+1})^{1/2}$$

$$q_{j+1} = r_{j+1} / \beta_{j+1}$$

There are two remarkable facts about this formulation of the Lanczos algorithm for the eigenvalue problem. First, no factorization of the possibly singular mass matrix M is required. Second, the Lanczos vectors $q_1, q_2, \ldots$ are orthogonal with respect to the inner product defined by M.

When M is semidefinite, it does not define a true inner product, but the algorithm still works as described. The null space of M is spanned by the eigenvectors corresponding to infinite eigenvalues of **Eq. 6-63**. In that case, the initialization step (1b) purges the infinite eigenvector components from the first Lanczos vector (and therefore all successive Lanczos vectors, in exact arithmetic). Thus, the Lanczos algorithm still works by implicitly restricting M to a positive definite matrix operating on the row space of M, this restriction defining a true inner product. An extra benefit is that the Lanczos algorithm in the version above generates no approximations to the infinite eigenvalues of **Eq. 6-63** that arise when M is semidefinite. It is worth noting that in implementation, the Lanczos vectors may collect some infinite eigenvector components due to round-off; these components do not affect the Lanczos recurrence, and can be purged from the computed eigenvectors by an Ericsson-Ruhe correction.

The tridiagonal matrix T_j defined in **Eq. 6-61** contains approximations to the finite eigenvalues of **Eq. 6-64** via the so-called spectral transformation. Specifically, if θ is an eigenvalue of T_j, then

$$\lambda = \sigma + \frac{1}{\theta} \qquad \text{Eq. 6-65}$$

is an approximate eigenvalue of Eq. 6-64. The approximate eigenvector y can be computed in the same way as in the previous section. Since the Lanczos vectors are M-orthogonal, the eigenvectors corresponding to different eigenvalues are also M-orthogonal.

To solve the buckling problem of the form

$$Kx = \lambda K_d x \qquad \text{Eq. 6-66}$$

a different shifting and inverting strategy is required. In Eq. 6-66, the differential stiffness matrix K_d is merely symmetric and has no definite properties, whereas the stiffness matrix K is positive semidefinite. The semidefinite property of M in the vibration analysis is critical to the Lanczos algorithm, so the shifting strategy applied in the vibration case cannot be applied to the buckling case simply by substituting K_d for M.

Since the desired eigenvalues are usually the ones closest to zero, a simple approach is to interchange the roles of K and K_d and then compute the largest eigenvalues of the problem. Therefore, by applying the Lanczos algorithm without shift, we can write

$$K_d x = \mu K x \qquad \text{Eq. 6-67}$$

where $\mu = 1/\lambda$. This formulation is not quite sufficient, because it does not allow any shifting for other eigenvalues.

A general shifting and inverting strategy is possible for the buckling problem. As shown previously, the operator $K - \sigma K_d$ is factored for an arbitrary shift, but the Lanczos recurrence is carried out using K-orthogonality among the Lanczos vectors. Each multiplication by the mass matrix M in the vibration case is replaced by a multiplication by the stiffness matrix in the buckling case. The rest of the Lanczos recurrence remains the same. Hence, in the buckling case the Lanczos algorithm works with the operator $(K - \sigma K_d)^{-1} K$ and K-orthogonality.

This shifted and inverted operator allows for arbitrary shifts with the exception of a shift at zero that reduces the problem to an identity matrix. For all other shifts, an eigenvalue θ of T_j can be transformed as

$$\lambda = \sigma \frac{\theta}{\theta - 1} \qquad \text{Eq. 6-68}$$

to yield an approximate eigenvalue of Eq. 6-66. Eigenvectors are computed as before without any additional back transformation resulting in a K-orthogonal set. Since the stiffness matrix K is used in the initialization step in the same way as M is used in the vibration problem, the sequence of Lanczos vectors and hence the

eigenvectors are orthogonal to the null space of K. Therefore, T_j does not yield any approximation to the exact zero eigenvalues of the buckling problem. The rigid body modes of the structure are not computed during the buckling analysis.

Block Method

In exact arithmetic, the single-vector Lanczos algorithm can only compute one eigenvector in a degenerate set. Because of round-off errors introduced into the Lanczos recurrence, additional eigenvectors of multiple eigenvalues may eventually appear in the tridiagonal matrix T_j. A second eigenvector of a multiple eigenvalue only converges a number of steps after the first eigenvector converged. (Effectively, this is the case where the starting vector is orthogonal to the desired eigenvector.) Thus, the single-vector Lanczos algorithm has difficulties with eigenvalues of high multiplicity.

Each step of the shifted Lanczos recurrence requires the solution of a sparse linear system of equations of the form $(K - \sigma M)x = b$ and one multiplication by the matrix M. In NX Nastran these operations require accessing matrices stored on disk files and thus entail significant I/O costs.

Block Lanczos algorithms have been developed in which the basic Lanczos recurrence is carried out for p vectors simultaneously. If the idea of a block code is combined with the shifted and inverted Lanczos algorithm, the following recurrence is obtained for the vibration problem:

1. Initialization
   a. A starting block of p column vectors R_0 is chosen.
   b. $R_1 = (K - \sigma M)^{-1} M R_0$ is computed.
   c. An upper triangular $p \times p$ matrix B_0 and an M-orthonormal $n \times p$ matrix Q_1 are computed so that $R_1 = Q_1 B_0$.
   d. The upper triangular $p \times p$ matrix B_1 is set to 0 as well as $Q_0 = 0$.
2. Iteration
   a. For $j = 1, 2, 3, \ldots$, the process iterates as follows, until convergence:

$$R_{j+1} = (K - \sigma M)^{-1} M Q_j - Q_j A_j - Q_{j-1} B_j^T$$

   where $A_j = Q_j^T M (K - \sigma M)^{-1} M Q_j$

   b. Compute the following factorization:

$$R_{j+1} = Q_{j+1} B_{j+1}$$

   where Q_{j+1} is an $n \times p$ matrix with M-orthonormal columns and B_{j+1} is a $p \times p$ upper triangular matrix.

Using the block version of the Lanczos algorithm provides the benefit that multiple eigenvalues of the original problem can be computed more easily. All degenerate eigenvectors can be found together when the multiplicity of an eigenvalue is less than or equal to the block size p. Further, the amount of I/O per column is reduced by a factor of $1/p$ since it is possible to solve and perform the matrix multiplication for p vectors simultaneously.

The description of the block Lanczos code is incomplete without a description of the factorization $R = QB$ and consideration of the fact that T_j is now block tridiagonal. This factorization is required to obtain M-orthonormal vectors in the recurrence. Here, R and Q are $n \times p$ matrices and B is an upper triangular $p \times p$ matrix, which is chosen so that the columns of Q are M-orthonormal. This procedure is implemented using a generalization of the modified Gram-Schmidt procedure that avoids repeated multiplications by matrix M. Both vectors in R and a second set, initially set to MR, are updated during the Gram-Schmidt orthogonalization. At the completion of the procedure, the first set of vectors are transformed to Q and the second set to MQ. A multiplication by M is required at the beginning of the procedure. A second multiplication is made at the end to ensure that MQ is accurate, but no additional multiplications by M are required during the procedure.

Another consequence of using a block Lanczos algorithm is that matrix T_j, from which the approximations to the eigenvalues are computed, is now a block tridiagonal matrix represented as follows:

$$T_j = \begin{bmatrix} A_1 & B_2^T & & & & & \\ B_2 & A_2 & B_3^T & & & & \\ & B_3 & A_3 & B_4^T & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & B_i & A_i & B_{i+1}^T & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & B_j^T & \\ & & & & & B_j & A_j \end{bmatrix}$$

Since the $\{B_i\}$ blocks are upper triangular, T_j is actually a banded matrix with bandwidth $p + 1$.

The eigenvalues and eigenvectors of T_j are computed by the following procedure. First, reduce T_j to a tridiagonal form T. An orthogonal matrix Q_H is found so that

$$Q_H^T \, T_j \, Q_H = T \qquad \text{Eq. 6-69}$$

Then perform an eigenextraction for T. An orthogonal matrix Q_T is found so that

$$Q_T^T \, T \, Q_T = \Lambda \qquad \text{Eq. 6-70}$$

where Λ is the diagonal matrix of the eigenvalues of T. Then, by combining **Eq. 6-69** and **Eq. 6-70**, the following is obtained:

$$(Q_T^T \, Q_H^T) T_j \, (Q_H \, Q_T) = \Lambda \qquad \text{Eq. 6-71}$$

where the orthogonal matrix $(Q_H \, Q_T)$ is the eigenvector matrix for H.

The orthogonal matrices Q_H and Q_T are products of simple orthogonal matrices (Givens rotations)

$$Q_{H_1} \cdot Q_{H_2} \cdot \ldots Q_{H_s}$$

and

$$Q_{T_1} \cdot Q_{T_2} \cdot \ldots Q_{T_r}$$

respectively. These product matrices are accumulated by beginning with the identity matrix *I* and successively multiplying on the right by

$$Q_{H_i}$$

where $i = 1, \ldots, r$.

Note: The eigenvector matrix is formed by products that take linear combinations of the columns. In the intermediate steps when only the last p rows of the eigenvector matrix are desired, the leading rows are ignored and the rotations are applied only to the last p rows of the identity matrix.

The algorithms used in **Eq. 6-69** and **Eq. 6-70** are standard and numerically stable.

The actual implementation of the band reduction algorithm of **Eq. 6-69** uses modified (square root free) Givens rotations to reduce the overall cost by eliminating the square root computations associated with each rotation.

The extraction of the eigenvalues and eigenvectors of T (see **Eq. 6-70**) is essentially the same procedure used by the QR algorithm (see "**QR Method of Eigenvalue Extraction**" on page 135). The procedure used here is an explicitly shifted QL algorithm using ordinary Givens rotations. The square root

$$\sqrt{a^2 + b^2}$$

required for the calculation of the Givens rotations is computed by a special recursion based on the Pythagorean Theorem. This recursion avoids overflow by not forming a^2 or b^2 and avoids destructive underflow occurring with the implicitly shifted QL algorithm for the matrices produced by the Lanczos shifted block. The spectral transformation is applied as before to find approximate eigenvalues for the original problem of **Eq. 6-64**. If s is an eigenvector of T_j, then $y = Q_j s$ with $Q_j = (Q_1, Q_2, \ldots, Q_j)$ is an approximate eigenvector of **Eq. 6-64**. The residual norms of the eigenpairs are now given by $\|B_{j+1} s_j\|$ where the vector s_j consists of the last p (block size) components of the eigenvector s.

The expense of I/O operations suggests that p should be as large as possible. The available memory precludes choosing very large block sizes. However, large block sizes also entail other costs because the M-orthogonal factorization requires additional computation and the overall convergence rate depends largely on the number of Lanczos steps and less on the dimension of T_j. Therefore, the actual block size is taken as a compromise among the reduction of I/O costs, the possible increase in arithmetic operations, and the largest multiplicity of the expected eigenvalues.

Orthogonalization

The Lanczos algorithm produces an orthonormal (or M-orthonormal) set of Lanczos vectors in exact arithmetic. (For simplicity, the orthogonality referred to always implies orthogonality with respect to the inner product defined by M). The numerical algorithm is affected by round-off errors that cause the Lanczos vectors to lose their orthogonality. Maintenance of orthogonality is essential for preserving the convergence properties of the Lanczos algorithm. Early implementations of the Lanczos algorithm employed a "full reorthogonalization" scheme in which each new block of Lanczos vectors was explicitly reorthogonalized against all previous Lanczos vectors. This process required pj^2 vector operations at step j as well as access to all previous vectors (usually stored out-of-core). Instead of this expensive scheme, the Lanczos algorithm in NX Nastran employs a combination of several efficient reorthogonalization mechanisms that together accomplish the computation of a sufficiently orthonormal set of Lanczos vectors.

Loss of orthogonality can occur in four different areas:

1. Within a given block of Lanczos vectors (internal).

2. With respect to the previous two blocks of Lanczos vectors (local).
3. With respect to the set of all previously computed Lanczos vectors (global).
4. With respect to eigenvectors from different shifts (external).

Problems with orthogonality within a block of Lanczos vectors can arise if the vectors of R_{j+1} are almost linearly dependent. For example, this problem occurs when the shift σ is extremely close to an eigenvalue. In this case, one step of the generalized modified Gram-Schmidt procedure is not sufficient to produce vectors that are orthogonal to working precision. Gram-Schmidt is considered an iterative procedure that possibly can be repeated several times until the Lanczos vectors are orthogonal. The Gram-Schmidt procedure is referred to as "internal reorthogonalization," which also requires updating the elements of B_{j+1}.

The local loss of orthogonality involves the previous two blocks of Lanczos vectors. The recurrence can be considered an orthogonalization of the block R_{j+1} against the blocks Q_j and Q_{j-1}. The block R_{j+1} computed from the Lanczos recurrence may not be orthogonal to Q_j and Q_{j-1} to full working precision. Investigations indicate that the orthogonality between R_{j+1} and Q_j is crucial for the correct continuation of the Lanczos process. Therefore, one step of local reorthogonalization is carried out; i.e., the block R_{j+1} is reorthogonalized against the block Q_j. This procedure may be repeated until the two blocks are orthogonal to working precision. This local reorthogonalization also requires updating the elements of A_j.

The global loss of orthogonality between the block Q_{j+1} and previous Lanczos blocks is of a different nature. The Lanczos recurrence involves only three blocks of Lanczos vectors. The beauty and efficiency of this recurrence lies in the fact that the three-term recurrence in exact arithmetic is enough to assure global orthogonality among all of the Lanczos vectors. Unfortunately, this is no longer true under the influence of round-off errors. Once some tiny error is introduced into the recurrence, it becomes amplified in the course of the next Lanczos steps and soon the global orthogonality property is lost. The mechanisms of this loss of orthogonality have been investigated in the last decade by several researchers and are now well understood. There are two important insights from the theoretical works that provide for an efficient implementation of the global reorthogonalization. First, it is possible to monitor the loss of orthogonalization inexpensively by updating certain estimates for loss of orthogonality at every Lanczos step. Second, it is sufficient to maintain orthogonality at the level of the square root of the machine's precision (semi-orthogonality) and still obtain fully accurate eigenvalues. These two observations give rise to the scheme of partial reorthogonalization that is generalized to the block Lanczos algorithm in the NX Nastran implementation.

The fourth type of loss of orthogonality can only occur in the context of the shifted and inverted algorithm, and has nothing to do with the Lanczos recurrence directly. The NX Nastran Lanczos algorithm begins by computing some eigenvalues and eigenvectors with an initial shift σ_1. If not all of the desired eigenvalues are found, then a second Lanczos run with a new shift σ_2 is made. For reasons of efficiency and simplicity in bookkeeping, the previously computed eigenvalues are prevented from being recomputed. This benefit is achieved by the external selective orthogonalization implemented in NX Nastran. For each new shift, a set of critical converged eigenvalues is determined. The Lanczos vectors are then kept orthogonal against the corresponding eigenvectors of this selected group of computed eigenvalues using a modification of the technique of selective orthogonalization. Estimates for the loss of orthogonality with respect to the computed eigenvectors are updated at each Lanczos step, and reorthogonalizations are performed only when semi-orthogonality is about to be lost.

Shift Strategy

The eigenanalysis problem in NX Nastran is to compute either the lowest m eigenvalues in magnitude or all eigenvalues in an interval $[a, b]$ where a and b can be any real numbers, $a < b$. The shift strategy incorporated in the block shifted Lanczos code allows the shifts selected in the interval $[a, b]$ to rapidly find the desired eigenvalues without wasted factorizations. The main objective for the shift strategy is to encompass the region containing the eigenvalues of interest with a trust region in which all the eigenvalues have been computed and the number of eigenvalues has been verified with a Sturm sequence check.

The first shift is chosen at the primary point of interest (usually 0 in vibration or –1 in buckling) or the left endpoint if it is finite. Additional shifts are chosen to form or extend a trust region until the requested eigenvalues are computed. These shifts are selected based on the information computed during previous Lanczos runs. A trust region can be extended in either direction.

The shift strategy is designed to place the final shift at the correct place to simultaneously compute the last required eigenvalues and to make a Sturm sequence check for the entire interval. However, a final shift may be required for a separate final Sturm sequence check. This check is designed conservatively so that shifts are not taken so far as to skip past part of the spectrum. However, if this does happen, the Sturm sequence count indicates that not all the eigenvalues are computed between two shift points. The shift strategy immediately attempts to find the missing eigenvalues rather than to extend the trust region further.

Two values, called the left and right sentinels, are associated with all shifts. A right sentinel is defined as the rightmost accepted eigenvalue that has no unaccepted eigenvalue approximations between the shift and itself. The left sentinel is defined similarly for the accepted and unaccepted approximations to the left of the shift.

When eigenvalues are missing between two shifts, it is usually because the last shift was too far from the previous shift. In this case, the missing eigenvalues should be between the right sentinel of the leftmost of the two shifts and the left sentinel of the rightmost of the two shifts. A new shift is selected to bisect the interval between the above two sentinels. In the rare case when the sentinels overlap each other, the new shift bisects the full interval between the two previous shifts. (The usual reason that such an overlap occurs is that the multiplicity of eigenvalues is greater than the Lanczos block size.)

There are several ways in which the shifting may deviate from the description above. Four major special cases are listed in this section. The first special case is the setting of the shift scale δ. Initially, δ is an approximation to the first nonrigid body mode of the structure. In the case of an initial shift at 0 for a structure with rigid body modes, only the rigid body modes in the initial Lanczos run are computed. In such cases the shift strategy does not update the value of δ, but instead uses the initial δ to move away from 0 towards the first nonzero eigenvalue. There are other special cases related to the rigid body modes where the Lanczos algorithm may terminate with no new information available for the next shift selection. In these cases δ is not updated, and the previous value is maintained.

The second special case is when no new information is computed at two consecutive shifts. This case may occur if there is a very large gap between modes of a structure or all remaining modes are infinite. The shift strategy expands its step size to cross the suspected hole in the spectrum. If the user-specified interval $[a, b]$ has a finite endpoint that was not used previously, the shift strategy selects the new shift at that endpoint. If the interval has only infinite endpoints remaining, then the new shift is chosen at $10\ \delta$ past the last shift. If no new information is computed at this shift, the next shift is chosen at $100\ \delta$ past the last shift. If there is still no new information, then the Lanczos procedure terminates with the assumption that the remaining eigenvalues are infinite. An appropriate warning is returned with those eigenvalues that were computed.

The third special case is the buckling problem. The operator used in the buckling problem for the Lanczos iteration is ill-posed for shifts at or near 0. The shift strategy process of the buckling problem is similar to the vibration problem with the exception that the shifts near 0 are perturbed away from 0.

The fourth major special case is the processing of factorization errors. In rare cases when the selected shift σ is exactly an eigenvalue, the operator $K - \sigma M$ is singular, and the factorization can fail. If this occurs, the shift is perturbed, and an additional factorization is performed. If three factorization errors occur consecutively, then the Lanczos procedure terminates with an appropriate error message and returns any computed eigenvalues and eigenvectors. Presumably, the cause of such a failure is an improperly posed eigenvalue problem for which the mass and stiffness matrices have a common null space (massless mechanisms).

Summary of Procedure

The general Lanczos procedure implemented in NX Nastran can be summarized in the following figures. There are two major levels in the procedure: the outer level in the shift strategy and administration, and the inner level in the actual Lanczos iteration. Figure 6-1 describes the outer level and Figure 6-2 describes the inner level.

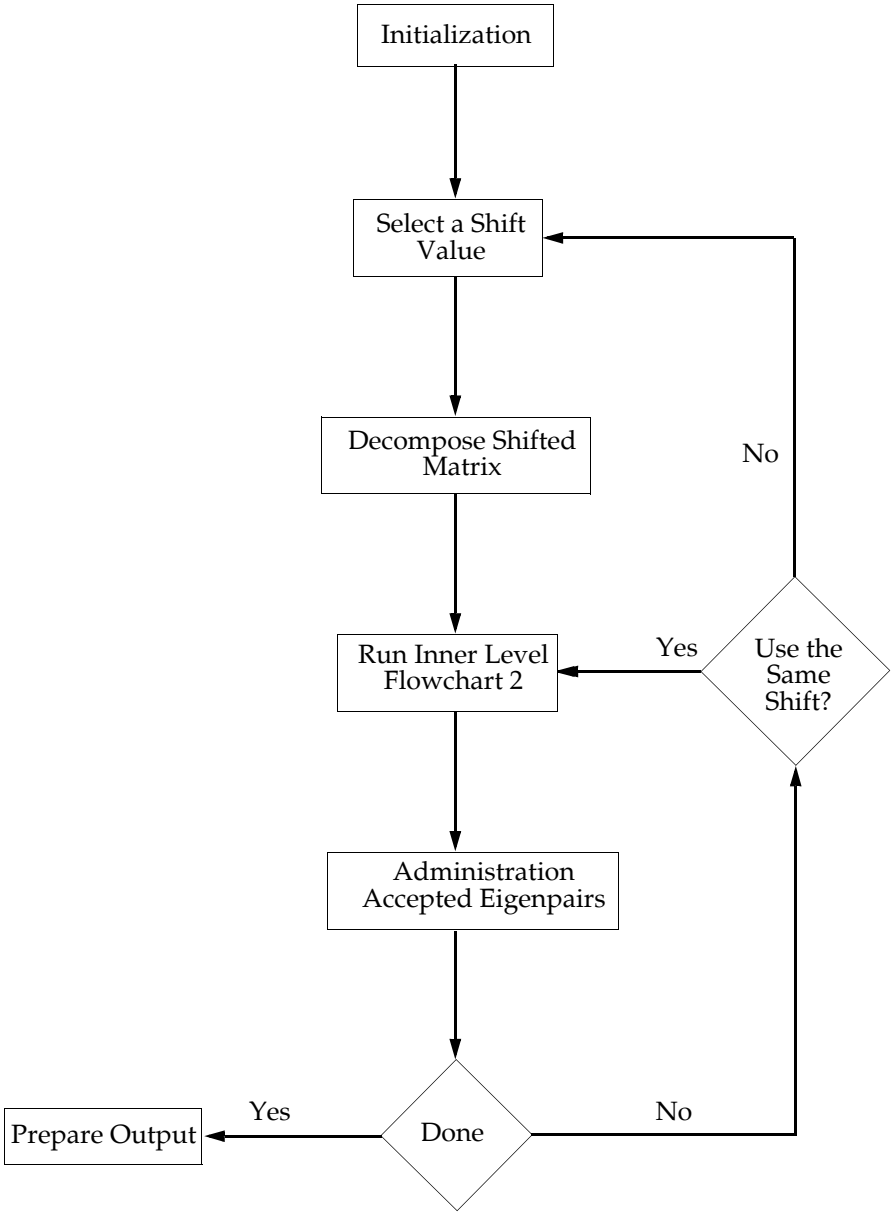


Figure 6-1 Outer Level of the Lanczos Procedure

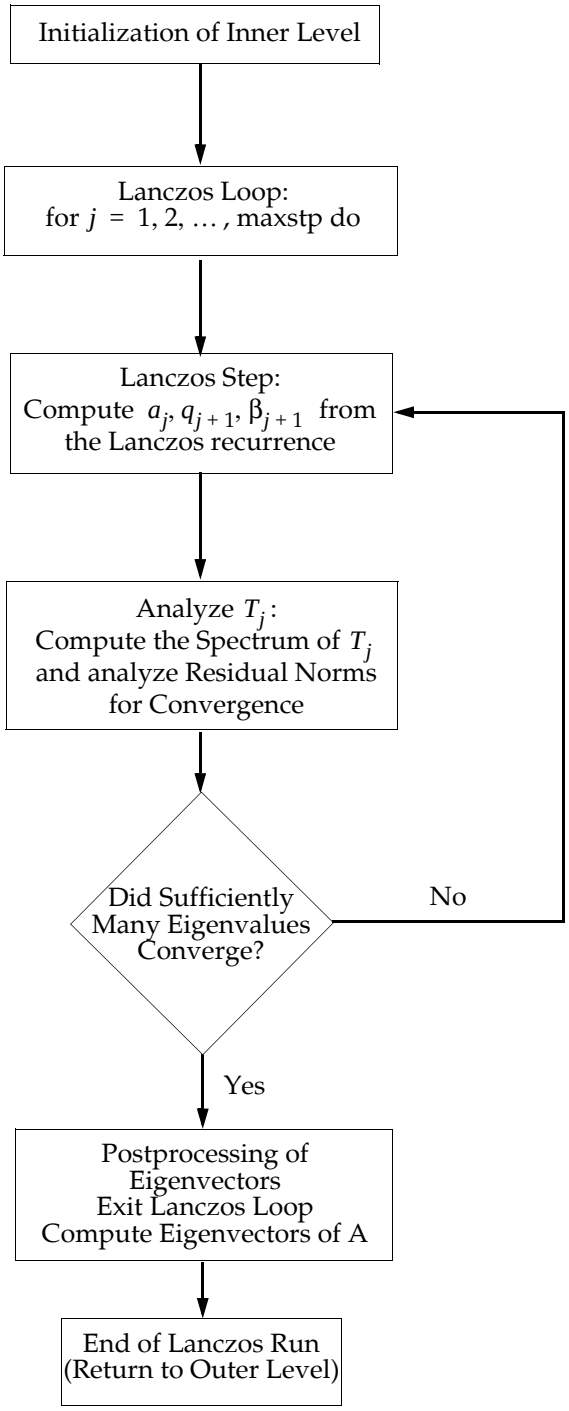


Figure 6-2 Inner Level of the Lanczos Procedure

Segmented Lanczos Method

The segmented version of Lanczos is intended to alleviate the orthogonality problems encountered on very wide frequency range runs. The orthogonality may be lost when very long Lanczos runs are executed while trying to span a wide frequency range of interest. The segment version forces intermittent shifts at semi-

regular intervals resulting in more frequent restarts of the Lanczos process. While it has significantly improved the solution quality, i.e., we avoid aborts due to loss of orthogonality, the number of shifts is usually more than the non-segmented run.

Frequency Domain Decomposition-Based Distributed Parallel Lanczos Method

The frequency domain decomposition normal modes analysis is built on the frequency segment approach of the Lanczos method. The frequency range of interest given by the user is subdivided automatically. The user also has the choice of introducing intermediate frequencies directly on the EIGRL continuation card. This may be especially advantageous in case of repeated runs (the most practical situation), where the user may be able to enforce better load balancing than the automatic intermediate frequency generation method. This process is executed in a master/slave paradigm. One of the processors (the master) will execute the coordination and collection of the separately calculated eigenvectors into an appropriately ordered set. This guarantees the continuation and proper exit of the master process. The slave processes will contain only the eigenvalues and eigenvectors they found upon exiting the READ module of NX Nastran.

Geometric Domain Decomposition-Based Distributed Parallel Lanczos Method

The principle of geometric domain decomposition is not confined completely to the READ module, as is the frequency domain decomposition. The geometric domain decomposition principle encompasses many DMAP modules prior to the READ module and the eigenproblem presented to READ is only a subset of the original problem.

It is important to emphasize that this version still solves the global (see **Eq. 6-1**) eigenvalue problem as "exactly" as the single processor run. The significant difference is that the geometry and therefore the global matrices are partitioned and the local processors see only the local portions. In the current implementation, we restrict ourselves to as many subdomains as we have processors. One might call this a "distributed global solution technique."

In general, there are two alternatives to execute this process. One is to keep only short, local n_o^j, n_a^j size vectors (where o indicates the interior set and a the boundary set), and another is to also add a global n_a size vector to the local Lanczos processes. The first method, while minimal in storage requirements, requires interprocess communication even when executing vector operations such as the Lanczos step. The latter technique requires communication only when matrix operations are executed (while it has n_a word redundancy on each processor), so this more attractive option is used in our implementation.

In the geometry domain parallel Lanczos method, the stiffness and mass matrices are partitioned into submatrices based on geometry subdomains as follows:

$$K = \begin{bmatrix} K_{oo}^{1} & & & & K_{oa}^{1} \\ & K_{oo}^{2} & & & K_{oa}^{2} \\ & & . & & . \\ & & & K_{oo}^{j} & K_{oa}^{j} \\ & & & & . \quad . \\ K_{ao}^{1} & K_{ao}^{2} & . & K_{ao}^{j} & . \quad K_{aa} \end{bmatrix} \qquad \text{Eq. 6-72}$$

$$M = \begin{bmatrix} M_{oo}^{1} & & & & M_{oa}^{1} \\ & M_{oo}^{2} & & & M_{oa}^{2} \\ & & . & & . \\ & & & M_{oo}^{j} & M_{oa}^{j} \\ & & & & . \quad . \\ M_{ao}^{1} & M_{ao}^{2} & . & M_{ao}^{j} & . \quad M_{aa} \end{bmatrix} \qquad \text{Eq. 6-73}$$

where the superscript j refers to the j-th subdomain, subscript a refers to the common boundary of the subdomains, and s is the number of the subdomains, so $j = 1, 2, \ldots, s$. The size of these global matrices as well as the eigenvectors is N. For the presentation of the algorithm, let us partition the Lanczos vectors accordingly:

$$x = \begin{bmatrix} x_{o}^{1} \\ x_{o}^{2} \\ . \\ x_{o}^{j} \\ . \\ x_{a} \end{bmatrix} \qquad \text{Eq. 6-74}$$

Furthermore, the boundary portion may be partitioned as:

$$x_a = \begin{bmatrix} x_a^1 \\ x_a^2 \\ . \\ x_a^j \\ . \\ x_a^s \end{bmatrix} \qquad \text{Eq. 6-75}$$

i.e., the *a* set is the global boundary set. Finally, the *j*-th processor will receive the *j*-th subdomain information:

$$\begin{bmatrix} K_{oo}^j & K_{oa}^j \\ K_{ao}^j & K_{aa}^j \end{bmatrix}, \begin{bmatrix} M_{oo}^j & M_{oa}^j \\ M_{ao}^j & M_{aa}^j \end{bmatrix}, \begin{bmatrix} x_o^j \\ x_a^j \end{bmatrix} \qquad \text{Eq. 6-76}$$

where the submatrices are partitioned into local *o* and *a*-sets and the appropriate sizes are n_o^j and n_a^j.

The main computational elements of the Lanczos method are executed in the distributed form as follows.

Simultaneous Shifted Matrix Generation. Since the inputs to the READ module are the K^j and M^j matrices, the task of creating the local system matrix will be simultaneously executed on each of the processors serially:

$$A^j = \begin{bmatrix} A_{oo}^j & A_{oa}^j \\ A_{ao}^j & A_{aa}^j \end{bmatrix} = \left(\begin{bmatrix} K_{oo}^j & K_{oa}^j \\ K_{ao}^j & K_{aa}^j \end{bmatrix} - \lambda_0 \begin{bmatrix} M_{oo}^j & M_{oa}^j \\ M_{ao}^j & M_{aa}^j \end{bmatrix} \right) \qquad \text{Eq. 6-77}$$

The A^j matrix is the distributed submatrix on the local memory of the *j*-th processor (node). Naturally, the local mass matrix component will also be saved locally and, since it is needed in each Lanczos step, it will be stored in local memory if possible. The shifted stiffness matrix will be stored on the local scratch disk of the node.

Important:	The λ_0 shift is calculated from local information such as matrix norms and runtime measures; therefore, it is processor dependent. Hence, some communication between the nodes is required to assure that the shift is uniform across the processors.

Distributed Factorization. The parallel, distributed implementation will execute the following steps:

1. The partial decomposition will formally decompose this *j*-th subdomain as follows:

$$A^j = \begin{bmatrix} A^j_{oo} & A^j_{oa} \\ A^j_{ao} & A^j_{aa} \end{bmatrix} = \begin{bmatrix} L^j_{oo} & 0 \\ L^j_{ao} & I \end{bmatrix} \begin{bmatrix} D^j_{oo} & 0 \\ 0 & \bar{A}^j_{aa} \end{bmatrix} \begin{bmatrix} L^j_{oo} & 0 \\ L^j_{ao} & I \end{bmatrix}^T \quad \text{Eq. 6-78}$$

where the identity matrices are not formed explicitly (they are only presented to make the matrix equation algebraically correct), and the $\bar{A}^j_{aa}$ submatrix is the boundary submatrix of the *j*-th partition updated by the factorization contributions from the interior as:

$$\bar{A}^j_{aa} = A^j_{aa} - L^j_{ao}\, D^j_{oo} (\, L^j_{ao})^T \quad \text{Eq. 6-79}$$

2. The boundary submatrices are summed up as:

$$\bar{A}_{aa} = \sum_{j=1}^{s} \bar{A}^j_{aa} \quad \text{Eq. 6-80}$$

3. The boundary is decomposed as:

$$\bar{A}_{aa} = L_{aa}\, D_{aa}\, L^T_{aa} \quad \text{Eq. 6-81}$$

The distributed decomposition step will be executed outside the Lanczos recurrence logic and the resulting partial factor:

$$\begin{bmatrix} L^j_{oo} \\ L^j_{ao} \end{bmatrix}$$

and the global boundary L_{aa} factor will be utilized inside the recurrence.

Since one important function of the decomposition operation in the Lanczos framework is to establish the Sturm count (S), another interprocessor communication is needed:

$$S_g = S_a + \sum_{j=1}^{s} S^j_o$$

where the Sturm counts in the equation above are the global, boundary, and local interior Sturm counts in that order.

Distributed Matrix Multiply. This is the first operation where the fact of having $n_o^j + n_a^j + n_a$ long Lanczos vectors in all local processors, but executing a local $n_o^j + n_a^j$ size matrix operation requires extra care. The basic local matrix-vector multiply operation is:

$$y^j = \begin{bmatrix} y_o^j \\ y_a^j \end{bmatrix} = \begin{bmatrix} M_{oo}^j & M_{oa}^j \\ M_{ao}^j & M_{aa}^j \end{bmatrix} \begin{bmatrix} x_o^j \\ x_a^j \end{bmatrix} = M^j x^j \qquad \text{Eq. 6-82}$$

where the local vector partitions are defined as:

$$x = \begin{bmatrix} x^j \\ x_a \end{bmatrix} = \begin{bmatrix} x_o^j \\ x_a^j \\ x_a \end{bmatrix} \qquad \text{Eq. 6-83}$$

and

$$y = \begin{bmatrix} y^j \\ y_a \end{bmatrix} = \begin{bmatrix} y_o^j \\ y_a^j \\ y_a \end{bmatrix} \qquad \text{Eq. 6-84}$$

Note that the x_a^j, y_a^j boundary partitions are the local subsets of the x_a, y_a. That is, each processor contains an identical copy of the global boundary in x_a, y_a, and an additional copy with local boundary entries only in x_a^j, y_a^j. The operation may be executed in the following steps:

1. Execute $y^j = M^j x^j$
2. Scatter y_a^j to y_a
3. Send y_a to master processor
4. Master sums up y_a contributions from all processors
5. Receive y_a from master processor
6. Gather y_a^j from y_a

Operation 1. will be executed by the existing architecture, which consists of two phases. In phase 1 we save the M matrix in a sparse form, storing the indices and numerical terms separately. This in turn is read in and interpreted by the 2nd phase that executes the actual numerical operation by calling the XPI2R* indexed, double SAXPY kernel. This phase does not need to understand the local versus boundary structure of the M matrix.

In addition, the option to keep a portion or all of the mass matrix in memory is available in the distributed environment. The operations **3.** through **5.** execute the communication and summing up of the boundary segments between the processors sharing the boundary. Finally, the scattering and gathering of the local boundary segments (operations **2.**, **6.**) is straightforward.

Upon completing this step, all processors have a shared and complete y_a vector, identical in content. This is necessary to proceed with the local Lanczos process.

Distributed F-B Substitution. This phase contains the following elements. Here, y is the right-hand side, $\dot{z}$ is an intermediate result, and z is the solution vector.

1. The partial factorization of the interior of the *j*-th subdomain is independent of the boundary, allowing a forward substitution on the interior as:

$$(L_{oo}^{j})^{T}\, z_{o}^{j} + (L_{ao}^{j})^{T}\, z_{a}^{j} \;=\; [L_{oo}^{j}\, D_{oo}^{j}]^{-1}\;\; y_{o}^{j} \;=\; \dot{z}_{o}^{j} \qquad \text{Eq. 6-85}$$

   It is important to note that in this step the overlapping boundary regions are zeroed out on all subprocesses, except for one, which will do the update.

2. The global boundary solution is a complete forward-backward substitution of:

$$L_{aa}\, D_{aa}\, L_{aa}^{T}\, z_{a} \;=\; \sum_{j\,=\,1}^{s} (y_{a}^{j} - L_{ao}^{j}\, D_{oo}^{j}\, \dot{z}_{o}^{j}) \qquad \text{Eq. 6-86}$$

3. Finally, the interior solution is a backward only substitution:

$$z_{o}^{j} \;=\; (L_{oo}^{j})^{-T} \left[\dot{z}_{o}^{j} - (L_{ao}^{j})^{T}\, z_{a}^{j}\right] \qquad \text{Eq. 6-87}$$

   The local z vector is again partitioned as:

$$z \;=\; \begin{bmatrix} z^{j} \\ z_{a} \end{bmatrix} \;=\; \begin{bmatrix} z_{o}^{j} \\ z_{a}^{j} \\ z_{a} \end{bmatrix} \qquad \text{Eq. 6-88}$$

   therefore, a scattering and gathering operation is performed as described in steps **2.** and **6.** of "**Distributed Matrix Multiply**" on page 163.

Simultaneous Lanczos Step. Since at this stage all processors have their local $y_k^j = M^j x_k^j$ and $z_k^j = A^j x_k^j$ vectors (k refers to the Lanczos step number) as well as the last two Lanczos vectors x_k^j, x_{k-1}^j, the Lanczos step will be executed simultaneously as follows:

1. Execute local inner product:

$$\alpha_k^j = (y_k^j)^T z_k^j \qquad \text{Eq. 6-89}$$

2. Create global inner product via communication:

$$\alpha_k = \sum_{j=1}^{s} \alpha_k^j \qquad \text{Eq. 6-90}$$

3. Execute local saxpy:

$$x_{k+1}^j = z_k^j - \alpha_k x_k^j - \beta_{k-1} x_{k-1}^j \qquad \text{Eq. 6-91}$$

In order to normalize the next Lanczos vector, another matrix vector multiplication is needed inside of the Lanczos step (using the same method described in "**Distributed Matrix Multiply**" on page 163):

$$y_{k+1}^j = M^j x_{k+1}^j \qquad \text{Eq. 6-92}$$

The calculation of the normalization parameter follows steps **1.** through **3.** above:

$$\beta_k^j = \left((x_{k+1}^j)^T y_{k+1}^j \right)^{1/2}$$

and

$$\beta_k = \sum_{j=1}^{s} (\beta_k^j)^{1/2} \qquad \text{Eq. 6-93}$$

This leads to the next normalized Lanczos vector

$$x_{k+1}^j = x_{k+1}^j / \beta_k \qquad \text{Eq. 6-94}$$

and the step is completed.

Distributed Orthogonalization Scheme. While this operation does not explicitly involve matrix operations, its performance impact is such that we have a distributed implementation. The main kinds of orthogonalization are: against the last two Lanczos blocks; against earlier found eigenvectors at the initial phase and during the iteration, respectively; and, finally, within the current block.

The efficiency of the distributed orthogonalization scheme is coming from the fact that we can distribute $O(k \bullet n)$ (vector) operations but need to communicate only $O(k)$ (scalar) data.

The distributed orthogonalization is executed in the following steps:

1. Calculate local inner products.
2. Exchange and sum up local inner products.
3. Execute local saxpy step.

In step **1.**, each processor is to calculate:

$$\omega_i^j = (x_{k+1}^j)^T M^j x_i^j \qquad \text{Eq. 6-95}$$

where i is the set of selected Lanczos vectors. In step **2.**, the following global sum is needed on each processor:

$$\omega_i = \sum_{j=1}^{s} \omega_i^j \qquad \text{Eq. 6-96}$$

Finally, step **3.** is executed in two segments, x_a, x_o^j, as

$$x_{o,\,k+1}^j = x_{o,\,k+1}^j - \omega_i x_{o,\,i}^j \qquad \text{Eq. 6-97}$$

and

$$x_{a,\,k+1} = x_{a,\,k+1} - \omega_i x_{a,\,i} \qquad \text{Eq. 6-98}$$

followed by a gathering into the $x_{a,\,k+1}^j$ array from $x_{a,\,k+1}$

After this step, the Lanczos process may be continued again from the "**Distributed Matrix Multiply**" on page 163. The process continues on all the nodes until a certain number of steps are executed or convergence achieved, which decision again needs communication..

Simultaneous Tridiagonal Solution Generation. Since all processors have the Lanczos blocks, and hence the same block tridiagonal matrix, the solution of the tridiagonal problem of **Eq. 6-69** will be executed on all nodes simultaneously. Once the tridiagonal solution has been completed, selected λ will be accepted as

eigenvalues of the original problem and the corresponding u will become the basis to compute eigenvectors. Note that the u vectors are the same size on all the nodes. Since all the nodes have $n_o^j + n_a^j + n_a$ length Lanczos vectors x_i, $i = 1, 2, \ldots, k$, the eigenvector computation of the local segments will not require additional communication.

The decision to stop the Lanczos run is based on analyzing the T_k matrix and cost analysis. Since the terms of this matrix are identical on all nodes, the stop must be simultaneous, but is synchronized for safety. If more eigenvectors are needed, another Lanczos run may be executed or ultimately the complete process starting from "**Simultaneous Shifted Matrix Generation**" on page 161 may be repeated with a different λ_0 shift value. If all required eigenvectors are found, the final orthogonality test will be executed on the local matrices and vectors and the READ module will complete by setting ND to the number of accepted eigenvectors. Note that upon exiting the READ module, all the processors have a complete LAMA table of eigenvalues; however, the PHIA matrix contains only the local rows of the eigenvectors.

The data recovery for each domain is done independently on each processor on the local segments of the eigenvectors provided by READ. As this does not require any communication, it helps in the overall speedup and disk-space savings during that part of the run. Following data recovery, the output may or may not be merged based on the mergeofp keyword.

Hierarchic Domain Decomposition-Based Distributed Parallel Lanczos

The hierarchic domain decomposition approach combines the frequency domain decomposition and geometric domain decomposition techniques. It is well-suited for very large problems with a wide frequency range, using a large number of processors. At the beginning of the job, the processors are divided into *nclust* equal-size subsets of processors, called clusters. When the eigenanalysis is begun, the user-specified frequency range is subdivided into *nclust* frequency segments, just as in frequency domain decomposition. Then, each cluster performs a complete eigenpair computation on its own frequency segment, independent of the other clusters, using geometric domain decomposition.

As in geometric domain decomposition, after exiting the READ module, the PHIA matrix on each processor contains only partial results (eigenvector entries for local rows). Unlike geometric domain decomposition, the LAMA table and columns of the PHIA matrix correspond to only the eigenvalues computed in that processor's cluster, rather than the global eigensolution. The LAMA tables and PHIA matrices are exchanged after the READ module as necessary, in order to produce complete eigenvalue results and data recovery.

To use hierarchic domain decomposition, the number of clusters is set by the "nclust" keyword.

The HDMP enables the usage of a cluster of workstation nodes by using both geometric domain and frequency domain partitioning of the normal modes analysis problem. The number of geometry partitions multiplied by the number of frequency segments equals the number of processors available in the work environment.

The number of frequency segments equals the number of clusters available which is defined by the new submittal keyword "nclust". In order to improve the balance of the frequency domain decomposition, the existing ALPHA tuning value of the EIGRL continuation card may be used.

The number of processors is specified with the existing dmp submittal keyword. The number of geometry partitions is the value of the dmp keyword divided by the value of the nclust keyword. The division result must be an integer greater than 1; for best results, a power of 2 is recommended. It is possible that the problem is not large enough to be partitioned, in which case a message is printed and a serial execution is done.

The HDMP is available for SOL 103. The user must use the EIGRL card, not the EIGR card, and both Fmin and Fmax must be specified. Fluid grids, disjoint structures and superelements are not permitted in the original implementation. HDMP is available with fewer restrictions and in a broader range of solution sequences if it is used together with the gpart option, introduced in NX Nastran 4; see the Parallel Processing User's Guide for details.

Recursive Domain Decomposition-Based Distributed Parallel Lanczos

The Recursive Domain Lanczos method (RDMODES) extends the DMP parallel capability via substructruing technology for very large scal normal node analysis.

In general, the RDMODES approach computes fewer modes with lower accuracy compared to the standard Lanczos approaches in order to gain performance.

RDMODES begins with partitioning the model into *nrec* external partitions. Each interior eigensolution corresponding to its external partition is the performed in serial, independent of the others. If the keyword nclust is specified, the processors are divided into nclusters as in HDMODES. In this case, each interior eigensolution is performed in GDMODES fashion in its own cluster.

As in hierarchic domain decomposition, the eigenvector matrix on each processor is required to exchange across the processors after READ module in order to complete data recovery. In the case of nclust presented, data recovery steps occur in two phases: first, on all partitions local to that cluster, and secondly, across the cluster.

To use RDMODES, the number of external partitions is set by the "nrec" keyword.

6.3 Solution Method Characteristics

The real eigenvalue solution methods are categorized as shown in the following table:

Table 6-2 Real Eigenvalue Methods

Method	Type	Identifier	Application	Restriction
Givens	Reduction	GIV	All Roots	M Positive Definite
Householder	Reduction	HOU	All Roots	M Positive Definite
Modified Reduction	Reduction	M_{HOU}^{GIV}, A_{HOU}^{GIV}	All Roots	$[K] + \lambda_s[M]$ Not Singular
Lanczos	Iteration	LANC	Small Number of Roots	$[K] + \lambda_s[M]$ Not Singular M Positive Semidefinite K Symmetric

6.4 DMAP User Interface

```
READ      KAA,MAA,MR,DAR,DYNAMIC,USET,CASECC,
          PARTVEC,SIL,VACOMP,INVEC,LLL,EQEXIN,GAPAR/
          LAMA,PHIA,MI,OEIGS,LAMMAT,OUTVEC/
          FORMAT/S,N,NEIGV/NSKIP/FLUID/SETNAME/SID/METH/
          F1/F2/NE/ND/MSGLVL/MAXSET/SHFSCL/NORM/PRTSUM/
          MAXRATIO $
```

Input Data Blocks:

KAA	Stiffness matrix.
MAA	Mass matrix.
MR	Rigid body mass matrix
DAR	Rigid body transformation matrix.
DYNAMIC	Eigenvalue Extraction Data (output by IFP module).
USET	Degree-of-freedom set membership table.
CASECC	Case Control Data Table (selects EIGR, EIGRL, or EIGB entries, output by IFP module).
PARTVEC	Partitioning vector with values of 1.0 at the rows corresponding to degrees of freedom which were eliminated in the partition to obtain KAA and MAA. Required for maximum efficiency. See SETNAME parameter description below.
SIL	Scalar index list. Required for maximum efficiency.
VACOMP	Partitioning vector of size of a-set with a value of 1.0 at the rows corresponding to r-set degrees-of-freedom. The USET table may be specified here as well. If VACOMP is purged and DAR does not have the same number of rows as KAA, then the partitioning vector will be determined from the size of MR.
INVEC	Starting vector(s) for Lanczos method only or EQMAP data blocks for geometry domain parallel.
LLL	Lower triangular factor from decomposition of KAA. Use to enhance shift logic for buckling eigenvalue extraction or VF01: interior boundary partitioning vector for geometry domain parallel Lanczos method.

EQEXIN Equivalence between external and internal grid identification numbers. Required for maximum efficiency.

GAPAR Local-global boundary partitioning vector for geometry domain parallel Lanczos method.

Output Data Blocks:

LAMA Normal modes eigenvalue summary table.

PHIA Normal modes eigenvector matrix in the a-set.

OEIGS Real eigenvalue extraction summary.

MI Modal mass matrix.

LAMMAT Diagonal matrix containing eigenvalues on the diagonal (Lanczos and QLHOU only).

OUTVEC Last vector block (Lanczos only).

Parameters:

FORMAT Input-Character-no default. If FORMAT≠ 'MODES', READ will solve a buckling problem of $([K] + \lambda[K^d])$.

NEIGV Output-integer-no default. NEIGV is the number of eigenvectors found. If none were found, NEIGV = 0. If m modes were found (but error encountered), NEIGV = –m. If m modes were found, NEIGV = m.

NSKIP Input-integer-default=1. The method used by READ is taken from the NSKIP record of CASECC.

FLUID Input-logical-default=FALSE. If FLUID = TRUE, then the EIGRL or EIGR entry is selected from METHOD(FLUID) Case Control command.

SETNAME Input-character-default='A'. For maximum efficiency, the rows and columns KAA and MAA must correspond to or be a partition of the displacement set specified by SETNAME. If KAA and MAA are a partition then PARTVEC must also be specified.

SID Input-integer-default=0. Alternate set identification number.

If SID=0, the set identification number is obtained from the METHOD command in CASECC and used to select the EIGR or EIGRL entries in DYNAMIC.

If SID>0, then METHOD command is ignored and the EIGR or EIGRL is selected by this parameter value. All subsequent parameter values (METH, F1, etc.) are ignored.

If SID<0, then both the METHOD command and all EIGR or EIGRL entries are ignored and the subsequent parameter values (METH, F1, etc.) will be used to control the eigenvalue extraction.

METH Input-character-default='LAN'. If SID<0, then METH specifies the method of eigenvalue extraction.

LAN	Lanczos
GIV	Givens
MGIV	Modified Givens
HOU	Householder
MHOU	Modified Householder
AGIV	Automatic selection of GIV or MGIV
AHOU	Automatic selection of HOU or MHOU

F1 Input-real-default=0.0. The lower frequency bound.

F2 Input-real-default=0.0. The upper frequency bound. The default value of 0.0 implies machine infinity.

NE Input-integer-default=0. The number of estimated eigenvalues for non-Lanczos methods only. For the Lanczos method, NE is the problem size (default=20) below which the QL Householder option is used if it is enabled.

ND Input-integer-default=0. The number of desired eigenvalues.

MSGLVL Input-integer-default=0. The level of diagnostic output for the Lanczos method only.

0	no output
1	warning and fatal messages
2	summary output
3	detailed output on cost and convergence
4	detailed output on orthogonalization

MAXSET	Input-integer-default=0. Vector block size for Lanczos method only.
SHFSCL	Input-real-default=0.0. Estimate of the first flexible natural frequency. SHFSCL must be greater than 0.0. For Lanczos method only.
NORM	Input-character-default='MASS'. Method for normalizing eigenvectors. See "**Option Selection**" on page 176 for details.
PRTSUM	Input-logical-default=TRUE. Lanczos eigenvalue summary print flag. See "**Performance Diagnostics**" on page 116 for details.
MAXRATIO	Input-real-default=10^5. May be overwritten in the DMAP by: param, maxratio, value.

6.5 Method Selection

EIGR Entry. The method selection of any method may be performed with the EIGR Bulk Data entry using the following format:

EIGR	SID	METHOD	F1	F2	NE	ND			
	NORM	G	C						

The SID is the set ID number corresponding to a METHOD command in the Case Control Section. METHOD should be equal to any of the identifiers given in "**Solution Method Characteristics**" on page 169. F1, F2 are frequency minimum and maximum values specifying the boundaries of the user's frequency range of interest. NE and ND are the number of roots estimated and desired to be found, respectively. On the continuation entry, the user can choose some normalization options, which are detailed in "**Option Selection**" on page 176.

EIGRL Entry. To select the Lanczos method in detail, the user should use the EIGRL Bulk Data entry with the following format:

EIGRL	SID	F1	F2	ND	MSGLVL	MAXSET	SHFSCL	NORM	
	ALPH	NUMS	f1	f2	f3	f4	f5	f6	
	f7	f8	f9	f10	f11	f12	f13	f14	
	f15								

The MSGLVL entry (0 through 3, default = 0) controls the amount of diagnostics output. MAXSET specifies the maximum number of vectors in a block of the Lanczos iteration. It is also equivalent to or may be overridden by the value of SYSTEM cell 263. The value of SHFSCL is an estimate for the location of the first nonzero eigenvalue of the problem.

The following parameters are only used if the F1 and F2 frequency range is to be broken up into segments. ALPH is the constant defining the modal distributions function ("**Frequency Segment Option**" on page 178). Its default value is 1.0, which results in a uniform distribution of segments. NUMS is the number of segments in the frequency range (default = 1). f1 to f15 are segment boundaries such that F1< f1 < f2 ... < f15 < F2. f1 to f15 if not specified will be computed based on a distribution given by ALPH.

Different combinations of F1, F2, and ND specify different options in the Lanczos module (see "**Frequency and Mode Options**" on page 176).

The main purpose of the SHFSCL is to aid the automatic shift logic in finding the eigenvalues especially in crossing the (possibly big) gap between the computationally zero (rigid body) modes and the finite (flexible) modes. Another use of SHFSCL is to create a cutoff frequency for the so-called supported modes.

The NORMalization parameter for Lanczos is described in "**Normalization Options**" on page 176.

6.6 Option Selection

Real symmetric eigenvalue analysis offers several normalization, frequency and mode, and performance options.

Normalization Options

The methods accessed by the EIGR entry have several normalization options. They are:

NORM = MASS	Mass normalization of eigenvectors (normalize to unit value of generalized mass).
NORM = MAX	Maximum normalization of eigenvectors (maximum component of vectors is unity).
NORM = POINT	A selected point normalized to unity. The point is specified by G (grid point number) and C (components 1 through 6).

The Lanczos method (using EIGRL) has MASS or MAX normalization capability only.

The following options are valid for the Lanczos method only unless otherwise stated.

Frequency and Mode Options

At present, based on the F1, F2, and ND combinations, the following options are supported in the Lanczos algorithm:

F1	F2	ND	Option
Given	Given	Given	Lowest ND or all in range
Given	Given	Blank	All in range
Given	Blank	Given	Lowest ND in range (F1, + ∞)
Given	Blank	Blank	Lowest one in range (F1, + ∞)
Blank	Blank	Given	Lowest ND in (– ∞, + ∞)
Blank	Blank	Blank	Lowest one in (– ∞, + ∞)
Blank	Given	Given	Lowest ND below F2
Blank	Given	Blank	All below F2

Note that if ND is not given in buckling, then both F1 and F2 need to be specified.

Performance Options

Space Saver. Another option available in Lanczos method is called the space saver option. This option is selected by setting SYSTEM (229) = 1 (default = 0) and results in significant reduction in scratch space usage by not preserving factor matrices for later use in the case both F1, F2 are given. However, CPU-time may increase.

Sparse Solver in Lanczos. For faster CPU execution, the Lanczos method by default executes sparse matrix operations. The memory available for the sparse solver inside the Lanczos module can be controlled by setting SYSTEM (146) to greater than 1. The larger the number, the larger the area reserved for the factor. Recommended values are 2, 3, or 4. SYSTEM(146) is equivalent to the FBSMEM keyword. This increased memory space is taken from the space available for the eigenvectors; consequently, the user must find a satisfactory compromise.

Model-Specific Lanczos Options. The REDORTH and REDMULT options may improve performance for models with certain properties. For very sparse (e.g. shell-dominated) models, setting SYSTEM(417) = 1 (the REDORTH keyword) improves performance by reducing the reorthogonalization cost component of the Lanczos run. For models containing virtual mass, setting SYSTEM(426) = 1 (the REDMULT keyword) reduces the matrix-vector multiply cost component. Both options incur some overhead, and therefore will not be beneficial for all models. These options may be used independently, in either serial or frequency domain Lanczos runs.

I/O Reduction Options. Other performance-enhancing options which control the I/O to CPU time ratio are described in the following table:

Table 6-3 I/O Reduction Options

System		Performance Option
(199) =	l	Set memory for incore mass matrix multiply to $2 \times l \times$BUFFSIZE (default: $l = 1$)
	0	Automatically fits the mass matrix in core if sufficient memory is available
(193) =	0	Save result of mass matrix multiply (default = 0)
	1	Do not save result of mass matrix multiply (default = 0)

Accuracy Options. The user may reset the Lanczos iteration tolerance by:

$$\text{SYSTEM(123)} = k$$

where k is the exponent of the Lanczos tolerance criterion if it is negative, or it is the exponent of the maximum matrix factor diagonal ratio tolerated by Lanczos if it is positive.

It is also possible to reset the pivot criterion for the decomposition inside the READ module by SYSTEM(89) = –k, resulting in 10^{-k} used.

In RDMODES, the user may modify the selected frequency range in the EIGRL specification for eigensolutions of each substructure to increase the accuracy of the solution by:

$$\text{rdscal} = d$$
$$\text{e}$$

where d is a positive real number. In most practical circumstances values in the range of 1.0-2.0 are acceptable. The trade-off is that the computational time increases with higher values of rdscale. This keyword is unique to RDMODES.

RDMODES Sparse Eigenvector Recovery Option. In many instances, a user is only interested in the solutions at a few key locations instead of all degrees of freedom, especially for large problems with millions of degrees of freedom. In such cases, the sparse eigenvector recovery method can significantly reduce the overall computation time and storage resource.

In RDMODES, the sparse eigenvector recovery option will be determined automatically based on the user's output request. If full eigenvectors are desired with only few output requests, a user can deactivate sparse data recovery with PARAM, RDSPARSE, NO in BULK data.

Note that currently residual vectors (PARAM, RESVEC, YES) and user input matrices (K2GG, M2GG, B2GG) are not supported for this sparse eigenvector recovery. If these features are needed, a user must specify PARAM, RDSPARSE, NO in BULK data for correct results.

Frequency Segment Option. The frequency segment option is controlled as follows.

The number of frequency segments may be specified on the EIGRL entry (NUMS) or on the NASTRAN entry (NUMSEG). In the case both are given, NUMS is set to NUMSEG. The intermediate frequencies may be directly given (f1 ... f15) by the user on the EIGRL entry. It is also possible to specify ALPH on the EIGRL entry or by setting FRQSEQ = SYSTEM(195). If both are given, then ALPH on the EIGRL card takes priority; if only SYSTEM(195) is set, then ALPH = SYSTEM(195)/100 is used.

If ALPH is specified by either means, the intermediate frequencies are automatically calculated by the formula shown in **Table 6-4**. Otherwise, the frequency segment is divided uniformly into equal-size subsegments.

Table 6-4 Frequency Segment Specification

PARAMETERS	Definition
NUMS	Number of frequency spectrum subsegments
ALPH	Subsegment boundary, $ALPH \neq 1$: $f_i = f_{min} + (f_{max} - f_{min})\frac{1 - ALPH^i}{1 - ALPH^{NUMS}}$

Miscellaneous Options

In the READ module, a new shift logic was introduced for the Lanczos method. If you wish to use the old method (delivered via the REIGL module), SYSTEM(273) must be set to a non-zero value. The default is zero.

Incompatible DAR and KAA sizes: If the DAR matrix has fewer rows than KAA, an appropriate size identity matrix is generated inside READ and merged into DAR to produce a DAA matrix. This assures that the rigid body modes included will have the same size as the flexible modes computed. This operation requires the presence of the VACOMP data block. If VACOMP is purged, a fatal error exit is produced.

Parallel Options

1) Frequency domain (fdmodes): dmp = n numseg = n on submittal

The value of dmp is equivalent to SYSTEM(231). The value of numseg in fdmodes is equivalent to SYSTEM(197), and should equal the value of dmp.

2) Geometry domain (gdmodes): dmp = n on submittal

The value of dmp is equivalent to SYSTEM(231). The numdom value in gdmodes is equivalent to SYSTEM(349), and equals the value of dmp by default.

3) Hierarchic domain (hdmodes): dmp=n nclust=m on submittal

The number of frequency segments equals the number of clusters available which is equivalent to SYSTEM(408) and defined with the nclust keyword.

The number of geometry partitions equals the value of the dmp keyword divided by the value of the nclust keyword, i.e. g = n/m. The division must be an integer.

Another choice is: gdoms=g fsegs=m on submittal

The number of processors equals the product of the value of the gdoms keyword and the value of the fsegs keyword.

4) Recursive domain (rdmodes): dmp=n nrec=p on submittal

The number of external partitions of the model equals the value of the nrec keyword, which is equivalent to SYSTEM(445). If nclust, an existing keyword introduced in HDMODES, is specified, the interior eigensolution is performed in GDMODES fashion in each cluster.

Mass Matrix Analysis Options

Indefinite Test

The mass matrix MAA is examined for all real vibration (non-buckling) eigenvalue problems if SYSTEM(303) $\leq$ 0. The NASTRAN keyword MINDEF is equivalent to SYSTEM(303). First and foremost, MAA is checked to determine if it is indefinite. If MAA is indefinite, UFM 4646 is printed, which includes a User Action suggesting the use of SYSTEM(304) to perturb the mass matrix, which may render it positive semi-definite. If SYSTEM(304) (aka MPERTURB keyword) is activated, a small value is added to the diagonal of MAA prior to the indefinite test. Then, if the indefinite test passes, the perturbed MAA is used in the subsequent eigenvalue analysis regardless of extraction method. Under no circumstances will an eigenvalue analysis proceed if MAA is determined to be indefinite.

Rank Test

A rank test is performed only when the indefinite test is requested. The rank of MAA (NRANK) is determined in order to detect the presence of infinite roots. The number of eigenvectors requested is restricted to min (N,NRANK) for the tridiagonal methods (HOU,GIV). The rank test is controlled by SYSTEM(313), which defines a maximum ratio of MAA diagonals to MAA-factor diagonals.

Density Control

Neither of the above tests is performed if it is estimated that the tests themselves would require too much time. A test is made to determine if the density of MAA exceeds some threshold (default 10%). If MAA exceeds the threshold, it is deemed "dense"; therefore, its factorization might be nontrivial, and these tests are bypassed. The threshold is controlled by SYSTEM(314). If N (the problem size) is less than or equal to 100, the density check is bypassed.

These analyses are summarized in **Table 6-5**:

Table 6-5 Mass Matrix Analyses

System Cell	Comment	Description	Default Value
303	Indefinite Test	=0 cutoff=1.0e-6 <0 cutoff=10**sys303 > bypass test	0 (Do the Test)
304	M Perturbation	=0 perturb=1.0e-6 <0 perturb=10**sys304 >0 bypass perturbation	+1 (Do not Perturb)
313	Rank Test	=0 rank ratio=1.0e+7 >0 rank ratio=10**sys313 <0 bypass test	0 (Do the Test)
314	Density Threshold	=0 thresh=0.1 >0 thresh=sys314/10000. <0 do not check density	0 (10% Threshold)

QL Householder Option

If sufficient memory is available, a modern (QL iteration based) version of Householder methods (AHOU, MHOU, or HOU) is automatically used. This is also used in place of Lanczos when the problem size is smaller than the value of the NE

parameter (default = 20). Setting SYSTEM(359) = 0 will turn off the QL Householder option (default SYSTEM(359)=1) and deactivate Lanczos fallback to Householder for small problems.

6.7 Real Symmetric Eigenvalue Diagnostics

The diagnostics of the eigenvalue methods can be categorized as execution diagnostics, numerical diagnostics, and error messages.

Execution Diagnostics

A certain level of execution diagnostics of the READ module is requested by DIAG 16. For the Lanczos method, MSGLVL = 1, 2, or 3 in the EIGRL entry gives different levels of execution diagnostics. These diagnostics pages are elaborate, and therefore are described separately in "**Lanczos Diagnostics**" on page 188.

The following two tables are printed only when PRTSUM = TRUE (default)

Table of Shifts

The table of shifts shows the sequence of shifts taken, the Sturm counts, and the number of modes computed at each shift for the Lanczos method. It appears as follows:

TABLE OF SHIFTS (REIGL)

SHIFT #	SHIFT VALUE	FREQUENCY, CYCLES	# EIGENVALUES BELOW	# NEW EIGENVALUES FOUND
X1	X2	X3	X4	X5

X1 The shift number

X2 The shift value in eigenvalue units

X3 The shift value in frequency units (typically Hertz)

X4 The number of modes below this shift (the Sturm count)
If X4 is "FACTOR ERROR" then this shift is rejected because the MAXRATIO is too large. To override this, the user may set SYSTEM(166) = 4 and the factor will be used despite the high MAXRATIO.

X5 The number of modes found by the algorithm at this shift

Execution Summary. The execution summary table for the Lanczos method is as follows:

EIGENVALUE ANALYSIS SUMMARY (REAL LANCZOS METHOD)

BLOCK SIZE USED	X
NUMBER OF DECOMPOSITIONS	X
NUMBER OF ROOTS FOUND	X

NUMBER OF SOLVES REQUIRED X

TERMINATION MESSAGE: TEXT

The TEXT of the termination message can be any of the following:

- REQUIRED NUMBER OF EIGENVALUES FOUND
- ALL EIGENVALUES FOUND IN RANGE
- NOT ALL EIGENVALUES FOUND IN RANGE
- INSUFFICIENT TIME TO FIND MORE EIGENVALUES

The run may also finish with the following message:

- USER FATAL MESSAGE 5405, ERROR X OCCURRED DURING ITERATION

where X can be equal to
– 31: INSUFFICIENT WORKSPACE
– 32: QL ALGORITHM DID NOT CONVERGE
– 33: MORE EIGENVALUES FOUND THAN EXIST
– 34: FILE I/O ERROR
– 35: SVD ALGORITHM DID NOT CONVERGE

Numerical Diagnostics

UWM 3034:
ORTHOGONALITY TEST FAILED, LARGEST TERM = X NUMBER FAILED = PAIR = X, EPSILON = X.

This message is printed when the eigenvector accuracy is in doubt (up to a certain numerical limit). This message is given for all methods.

SFM 3034.2:
INTERNAL FAILURE IN THE LANCZOS PROCEDURE: M-ORTHOGONAL QR PROCEDURE FAILED TO CONVERGE. PROBABLE CAUSE: MASS MATRIX IS INDEFINITE (MODES) OR STIFFNESS MATRIX IS INDEFINITE (BUCKLING).

Indicates that the mass/stiffness matrix is indefinite or badly scaled.

UIM 5010:
STURM SEQUENCE DATA FOR EIGENVALUE EXTRACTION TRIAL EIGENVALUE = X, CYCLES = X, NUMBER OF EIGENVALUES BELOW THIS VALUE = X.

This information is very important in establishing the number of roots existing in certain subregions of the frequency region.

UFM 4646:
THE MASS MATRIX IS NOT POSITIVE DEFINITE USING HOUSEHOLDER OR GIVENS METHOD.

UFM 4645:
THE SHIFTED STIFFNESS MATRIX IS NOT POSITIVE DEFINITE IN MGIVENS OR MHOUSEHOLDER METHOD.

UFM 4648:
THE MODAL MASS MATRIX IS NOT POSITIVE DEFINITE.

These messages report problems from the reduction methods.

UWM 5411:
NEGATIVE TERM ON DIAGONAL OF MASS (NORMAL MODES) OR STIFFNESS (BUCKLING) MATRIX.

This is a Lanczos diagnostic message for information only.

Error Diagnostics

UFM 5429:
INSUFFICIENT TIME TO START LANCZOS ITERATION.

UFM 5400:
INCORRECT RELATIONSHIP BETWEEN FREQUENCY LIMITS.

This means the upper frequency limit has a lower value than the lower frequency limit.

SFM 5401:
LANCZOS METHOD IS UNABLE TO FIND ALL EIGENVALUES IN RANGE. ACCEPTED EIGENVALUES AND ADDITIONAL ERROR MESSAGES MAY BE LISTED ABOVE. A POSSIBLE CAUSE IS THE OCCURENCE OF HIGH MAXRATIOS. CHECK MODEL FOR MECHANISMS IF HIGH MAXRATIOS EXIST. USER ACTION: RERUN WITH ANOTHER METHOD OR ANOTHER SETTING ON EIGRL ENTRY.

UFM 5402:
THE PROBLEM HAS NO STIFFNESS MATRIX.

UFM 4683:
MASS (OR STIFFNESS) MATRIX NEEDED FOR EIGENVALUE ANALYSIS.

UWM 6243 (REIG):
THE DOF REQUESTED FOR POINT NORMALIZATION HAS NOT BEEN SPECIFIED ON THE EIGR OR EIGB ENTRY.

SFM 5299:
FINITE INTERVAL ANALYSIS ERROR or

STARTING BLOCK COMPUTATION ERROR or

INSUFFICIENT STORAGE FOR LANCZOS or

FACTORIZATION ERROR ON THREE CONSECUTIVE SHIFTS or

RESTORATION OF VECTORIZATION ERROR or

IMPROPER PARAMETER SPECIFICATION FOR LANCZOS or

TRUST REGION OVERFLOW IN LANCZOS or

UNRECOVERABLE TERMINATION FROM LANCZOS ITERATION

These messages issued under 5299 are from the Lanczos method. The first one, the finite interval analysis error, is the one most frequently encountered. This error indicates a high matrix-factor diagonal ratio at the F1 or F2 shifts which can be caused by a modeling error or matrix singularity.

SFM 5407:
INERTIA (STURM SEQUENCE) COUNT DISAGREES WITH THE NUMBER OF MODES ACTUALLY COMPUTED IN AN INTERVAL.

This message flags a serious problem, i.e., spurious modes were found in the Lanczos method.

UWM 5406:
NO CONVERGENCE IN SOLVING THE TRIDIAGONAL PROBLEM.

This message signals the abortion of the reduction methods.

UWM 9282 (SUBDMAP):
VIRTUAL MASS WILL BE IGNORED IN GDMODES EIGEN SOLUTION WHEN GPART = 0.

UWM 9288 (XREADR):
UNABLE TO PARTITION INTO NPART COMPONENTS.

RUNNING CONVENTIONAL LANCZOS SOLUTION INSTEAD.

UFM 9289 (XREADR):
EXIT DUE TO NO DESIRED INTERIOR MODES FOUND.

USER ACTION: INCREASE RDSCALE VALUE.

Performance Diagnostics

UIM 5403:
BREAKDOWN OF CPU USAGE DURING LANCZOS ITERATIONS

Eigenvalue analysis can be computationally expensive and may dominate overall CPU time. To help assess numerical performance, this message shows how much time the primary operations (forward-backward substitution, matrix-vector multiplication, and matrix summation and decomposition) consume during Lanczos iterations. The last entry, "LANCZOS RUN", refers to the duration of a complete set of Lanczos cycles at one shift and contains within it all FBS and matrix multiplication times, but not the shift and factor times. The sum of total times for "SHIFT AND FACTOR" and "LANCZOS RUN" should then approximate the total time taken by the REIGL or LANCZOS modules.

```
*** USER INFORMATION MESSAGE 5403 (LNNRIGL)
 BREAKDOWN OF CPU USAGE DURING LANCZOS ITERATIONS:
 OPERATION               REPETITIONS                                  AVERAGE      TOTAL
 FBS (BLOCK SIZE=...)         ....                                   ........   ........
 MATRIX-VECTOR MULTIPLY       ....                                   ........   ........
 SHIFT AND FACTOR             ....                                  ........   ........
 LANCZOS RUN                  ....                                  ........   ........
```

UIM 2141:
GIVENS (OR HOUSEHOLDER) TIME ESTIMATE IS X SECONDS. SPILL WILL OCCUR FOR THIS CORE AT A PROBLEM SIZE OF X.

The reduction type methods are fairly predictable (not a characteristic of other eigenvalue methods). The CPU time and storage estimate are given in this message.

Lanczos Diagnostics

Since the Lanczos method is the most robust and modern of the eigenvalue extraction methods, its execution diagnostics are described here in greater detail.

The printing of diagnostic information is controlled by the MSGLVL parameter. When left at its default value of zero, only the Table of Shifts and the Execution Summary block are printed in addition to the eigensolution. MSGLVL values of 1, 2, 3, or 4 yield increasingly more detailed diagnostic information about the Lanczos operations.

The MSGLVL=1 diagnostics are organized into four major sections. Section I reports on the original problem specification and the setting of module parameters. In this section an interval analysis check is also shown to establish the number of eigenvalues in the range set by the user.

Most of the detailed diagnostics are self explanatory. Some of the parameter values are detailed below:

MODE FLAG = 1 Vibration problem

2 Buckling problem

PROBLEM TYPE	=	1	Lowest ND roots in interval
		2	Highest ND roots
		3	All roots in interval
		4	ND roots nearest to center frequency

The LEFT and RIGHT END POINTS are the F1, F2 values set on the EIGRL entry. The center frequency is the center (not necessarily the arithmetic center) of the interval.

ACCURACY REQUIRED is a value automatically set by the program.

The CP TIME ALLOWED is the remainder of the time left to complete the run using the limit set on the TIME entry.

The SHIFTING SCALE is an estimate of the smallest (in magnitude) nonzero eigenvalue in the spectrum. This estimate can be specified by the user or automatically calculated by the program. The INERTIA values at the specific locations are Sturm numbers.

Section II provides diagnostics on the memory usage, the setting of the working array sizes based on the memory available, and the maximum memory allocated. The term RITZ VECTORS means the approximate eigenvectors. A TRUST REGION is a segment of the frequency spectrum, bounded by two shifts where all the eigenvalues are found. By the nature of the automatic shift algorithm, there can be several of these segments.

Section III is the Lanczos run section. The text shown in this section can be repeated for each new shift. This section also prints occasional user information messages (for example, 5010 and 4158) that report the results from the decomposition module. This section is further expanded when the user requests additional diagnostics from the Lanczos run by setting MSGLVL = 2 or 3.

Section IV reports the conclusion of the Lanczos run. The most frequently occurring warning flag settings are listed below:

COMPLETION FLAG	=	– 99	ND roots nearest to center frequency
		– 26	Starting block computation error
		– 25	Restoration of factorization error
		– 24	Illegal parameters error
		– 23	Core structure error
		– 22	Internal factorization error
		– 21	Insufficient storage
		– 20	Factorization error at a boundary shift

– 5	Incorrect frequency range
– 2	No eigenvalues found in range
0	Required number of roots found
1	All roots in interval found
2	Not all roots in interval found
3	Insufficient time to finish
4 – 7	Same as 1– 3; however, the inertia count error (see SFM 5407) occurred during the iteration

MSGLVL = 2 provides detailed information about the shift logic, the spectrum distribution, and the cause of the termination of a Lanczos run. This TERMINATION CRITERION may have the following values:

0	Preset maximum number of Lanczos steps reached
1	Cost of eigenvalue calculation increasing
2	Number of needed eigenvalues was found
3	Shift is too close to an eigenvalue
4	Lanczos block is singular
5	Running out of time
– 1	Insufficient workspace
– 2	QL algorithm does not converge
– 3	Too many eigenvalues found
– 4	File I/O error
– 5	Singular value decomposition error

Finally, MSGLVL = 3 describes the Lanczos run including norms, condition numbers, convergence criterion, shifted eigenvalues, and their residuals.

6.8 Real Lanczos Estimates and Requirements

The time estimates for the Lanczos method are the following:

Shifting time (sec):

$$T_d \cdot N_{shifts} \quad \text{Eq. 6-99}$$

Recursion time (sec):

$$N_{steps} \cdot N_{shifts}(2T_M + T_s) \quad \text{Eq. 6-100}$$

Orthonormalization time (sec):

$$2\ N_{shift} \cdot N_{steps}^2 \cdot M \quad \text{Eq. 6-101}$$

Packing time (sec):

$$2\ (N_{des} + N_{steps}) \cdot N \cdot P \quad \text{Eq. 6-102}$$

where:

N_{shifts} = number of shifts

N_{steps} = number of Lanczos steps

B = block size used

N_{des} = number of modes desired

T_d = decomposition time (see "**Decomposition Estimates and Requirements**" on page 71 for details)

T_s = solution time (see "**FBS Estimates and Requirements**" on page 87 for details)

T_M = matrix multiply time (see "**MPYAD Estimates and Requirements**" on page 47 for details)

The minimum storage requirements are as follows:

Disk: $N_{des} \cdot N \cdot \text{IPREC} + D_{factor}$

Memory: $2(N_{steps} \cdot B)^2 \cdot \text{IPREC} + 4\ B \cdot N \cdot \text{IPREC}$

where D_{factor} is the factor disk space requirement.

The number of shifts in most Lanczos runs is 2, occasionally 1, sometimes 3 or more. The number of Lanczos steps is 10 on the average.

6.9 References

Cullum, J. K.; Willoughby, R. A. ***Lanczos Algorithms for Large Symmetric Eigenvalue Computations.*** Birkhäuser, 1985.

Givens, W. ***Numerical Computation of the Characteristic Values of a Real Symmetric Matrix***. Oak Ridge National Lab., ORNL-1574, 1954.

Grimes, R. G., et al. ***A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems***. SIAM, J. Mat. Analysis Appl., 13, 1992.

Householder, A.S.; Bauer, F.L. ***On Certain Methods for Expanding the Characteristic Polynomial.*** Numerische Mathematik, Volume 1, 1959, pp. 29-37.

Lanczos, C. ***An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators.*** Journal of the Research of the National Bureau of Standards., Volume 45, 1950, pp. 255-282.

Lewis, J. G.; Grimes, R. G. ***Practical Lanczos Algorithms for Solving Structural Engineering Eigenvalue Problems.*** Sparse Matrices, and Their Uses, edited by I. E. Duff, Academic Press, London, 1981.

MacNeal, R. H.; Komzsik, L. ***Speeding Up the Lanczos Process.*** RILEM, Kosice, Slovakia, 1995.

Ortega, J. M.; Kaiser, H. F. ***The LR and QR Methods for Symmetric Tridiagonal Matrices.*** The Computer Journal, Volume 6, No. 1, Jan. 1963, pp. 99-101.

Parlett, B. N. ***The Symmetric Eigenvalue Problem***. Prentice Hall, Englewood Cliffs, 1980.

Smith, B. T. et al. ***Matrix Eigensystem Routines - EISPACK Guide.*** Springer Verlag, 1974.

Wilkinson, J. H. ***The Algebraic Eigenvalue Problem***. Oxford University Press, 1965.

Wilkinson, J. H. ***The Calculation of the Eigenvectors of Codiagonal Matrices.*** The Computer Journal, Volume 1, 1958, p. 90.

CHAPTER 7

Complex Eigenvalue Analysis

- Damped Models
- Theory of Complex Eigenvalue Analysis
- Solution Method Characteristics
- User Interface
- Method Selection
- Option Selection
- Complex Eigenvalue Diagnostics
- Complex Lanczos Estimates and Requirements

7.1 Damped Models

The solution of complex eigenvalue problems is important for damped models. The solution method is applied when either viscous or structural (or both) damping is used.

The basic equation of the damped free vibration problem is

$$M\ddot{u} + B\dot{u} + Ku = 0 \qquad \text{Eq. 7-1}$$

where:

$[M]$ = mass matrix

$[B]$ = damping matrix

$[K]$ = stiffness matrix

The B matrix may be null, and the M and K matrices may be real or complex, symmetric or unsymmetric.

The eigenvalue λ is given by

$$\lambda = a + i\omega \qquad \text{Eq. 7-2}$$

The solution u in terms of the complex eigenvalue and eigenvector is of the form:

$$u = e^{\lambda t}\Phi \qquad \text{Eq. 7-3}$$

7.2 Theory of Complex Eigenvalue Analysis

Canonical Transformation to Mathematical Form

The complex eigenvalue analysis problem is derived from:

$$M\ddot{u} + B\dot{u} + Ku = 0 \qquad \text{Eq. 7-4}$$

where u is the displacement vector. $\ddot{u}$ is the acceleration of the grid points, i.e., the second time derivative of u. $\dot{u}$ refers to the velocity or first time derivative. The solution of this homogeneous system (the free, but damped vibrations) is of the form

$$u = e^{\lambda t}\,\Phi \qquad \text{Eq. 7-5}$$

where Φ is a vector of complex numbers and the λ eigenvalue is also complex in general. By substituting **Eq. 7-5** into **Eq. 7-4** we get:

$$(M\lambda^2 + B\lambda + K)\Phi = 0 \qquad \text{Eq. 7-6}$$

In order to solve this quadratic eigenvalue problem, first a linearization transformation is executed. This transformation converts the original quadratic problem to a linear problem of twice the size.

It is obtained by simply rewriting **Eq. 7-6** as a 2×2 block matrix equation:

$$\lambda \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} + \begin{bmatrix} B & K \\ -I & 0 \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} = 0 \qquad \text{Eq. 7-7}$$

where:

$$\dot{\Phi} = \lambda\Phi \qquad \text{Eq. 7-8}$$

as in **Eq. 7-7**. This equation is now linear; however, there are shortcomings. Independently of the eigenvalue solution method, one would need to invert both the mass and damping matrices and an unsymmetric, indefinite matrix built from the damping and stiffness matrices, in order to reach a solution. Although the explicit inverses are not needed, the numerical decompositions on either of these matrices may not be well defined.

An advancement is possible by executing a spectral transformation, i.e., introducing an appropriate shift as:

$$\lambda = \lambda_0 + \mu \qquad \text{Eq. 7-9}$$

With the shift (whose appropriate selection in the complex case is rather heuristic) the linear equation may be rewritten as:

$$\begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} = \mu \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} \quad \text{Eq. 7-10}$$

Another recommended improvement is to invert the problem by introducing:

$$\Lambda = \frac{1}{\mu} \quad \text{Eq. 7-11}$$

By substituting and reordering we get:

$$\Lambda \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} = \begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^{-1} \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} \quad \text{Eq. 7-12}$$

The latter equation is a canonical form of

$$\Lambda x = Ax \quad \text{Eq. 7-13}$$

where:

$$A = \begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^{-1} \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \quad \text{Eq. 7-14}$$

and

$$x = \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix}$$

The form allows the singularity of the mass matrix. However, the zero subspaces of K, B, and M may not coincide. This is a much lighter and more practical restriction than requiring both the mass and the damping matrices to be nonsingular, as in **Eq. 7-7**. **Eq. 7-12** is the formulation used in the Lanczos methods.

If M is nonsingular, then a simplified formulation of

$$\begin{bmatrix} -(M^{-1}B + \lambda_0 I) & -M^{-1}K \\ I & -\lambda_0 I \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} = \mu \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} \quad \text{Eq. 7-15}$$

also results in a canonical form of

$$Ax = \mu x \qquad \text{Eq. 7-16}$$

Eq. 7-15 is the formulation used in the Hessenberg methods.

When B is null, then the double-size representation can be avoided.

The problem then becomes

$$[M\lambda^2 + K]\Phi = 0 \qquad \text{Eq. 7-17}$$

Using the following:

$$\lambda^2 = \lambda_0 + \mu \qquad \text{Eq. 7-18}$$

We can write

$$-[\lambda_0 M + K]\Phi = \mu M\Phi \qquad \text{Eq. 7-19}$$

Premultiplying **Eq. 7-19** by M^{-1} gives

$$-[M^{-1}K + \lambda_0 I]\ \Phi = \mu\Phi \qquad \text{Eq. 7-20}$$

Therefore, from **Eq. 7-15** and **Eq. 7-20**, the dynamic matrix A for the case of nonsingular M matrix is as follows:

$$A = \begin{cases} \begin{bmatrix} -(M^{-1}B + \lambda_0 I) & -M^{-1}K \\ I & -\lambda_0 I \end{bmatrix} & \text{when } B \neq 0 \\ -[M^{-1}K + \lambda_0 I] & \text{when } B = 0 \end{cases} \qquad \text{Eq. 7-21}$$

Because of the unsymmetric nature of the problem, the following left-handed solution also exists:

$$\psi^H(\lambda^2 M + \lambda B + K) = 0 \qquad \text{Eq. 7-22}$$

For the special case when B is null, the left-handed solution of the form:

$$\psi^H(\lambda^2 M + K) = 0 \qquad \text{Eq. 7-23}$$

also exists.

In **Eq. 7-22** and **Eq. 7-23**, the superscript H stands for complex conjugate transpose.

The left-handed eigenvectors of the problems will only be found in the Lanczos (**page 214**) and QZ Hessenberg (**page 212**) methods. The left-handed eigenvectors are useful in establishing convergence quality.

The physical eigenvalues may easily be recovered from the shift and invert operations in **Eq. 7-9** and **Eq. 7-11**:

$$\lambda = \frac{1}{\Lambda} + \lambda_0 \qquad \text{Eq. 7-24}$$

In order to find the relationship between the mathematical and physical eigenvectors, let us rewrite **Eq. 7-7** in the following block notation:

$$(\lambda \bar{M} + \bar{K})\ x = 0 \qquad \text{Eq. 7-25}$$

where again

$$x = \begin{bmatrix} \dot{\Phi} \\ \Phi \end{bmatrix} \qquad \text{Eq. 7-26}$$

The block matrices are simply:

$$\bar{K} = \begin{bmatrix} B & K \\ -I & 0 \end{bmatrix} \qquad \text{Eq. 7-27}$$

and

$$\bar{M} = \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \qquad \text{Eq. 7-28}$$

Substituting **Eq. 7-24** into **Eq. 7-25** and reordering yields:

$$[(\bar{K} + \lambda_0\ \bar{M})^{-1}\bar{M} + \Lambda I]\ x = 0 \qquad \text{Eq. 7-29}$$

which is the same as **Eq. 7-12** in block notation. This proves that the right eigenvectors are invariant under the shifted, inverted transformations, i.e., the right physical eigenvectors are the same as their mathematical counterparts, apart from the relevant partitioning.

For the left-handed problem of **Eq. 7-22**, we rewrite **Eq. 7-22** using block notation:

$$\underline{y}^H(\lambda \bar{M} + \bar{K}) = 0 \qquad \text{Eq. 7-30}$$

with a left-handed physical eigenvector of:

$$\underline{y}^H = [\dot{\psi}^H, \psi^H] \qquad \text{Eq. 7-31}$$

Substituting **Eq. 7-24** again gives:

$$\underline{y}^H[\bar{M} + \Lambda(\bar{K} + \lambda_0\,\bar{M})] = 0 \qquad \text{Eq. 7-32}$$

Factoring,

$$\underline{y}^H(\bar{K} + \lambda_0\,\bar{M})[(\bar{K} + \lambda_0\,\bar{M})^{-1}\bar{M} + \Lambda I] = 0 \qquad \text{Eq. 7-33}$$

which is equivalent to:

$$[(\bar{K} + \lambda_0\,\bar{M})^H\underline{y}]^H[(\bar{K} + \lambda_0\,\bar{M})^{-1}\bar{M} + \Lambda I] = 0 \qquad \text{Eq. 7-34}$$

Thus, the mathematical problem we solve is:

$$y^H[(\bar{K} + \lambda_0\,\bar{M})^{-1}\bar{M} + \Lambda I] = 0 \qquad \text{Eq. 7-35}$$

where

$$y^H = [(\bar{K} + \lambda_0\,\bar{M})^H\underline{y}]^H \qquad \text{Eq. 7-36}$$

This means that the left-handed physical eigenvectors are *not* invariant under the transformation. Expanding **Eq. 7-36** into the solution terms

$$y = -\begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^H \underline{y} \qquad \text{Eq. 7-37}$$

and finally

$$\underline{y} = -\begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^{-H} y \qquad \text{Eq. 7-38}$$

The cost of this back-transformation is not very large since the factors of the dynamic matrix are available and we need a forward-backward substitution only. The importance of this back-transformation is rather academic since there is no physical meaning associated with the left physical eigenvectors. On the other hand, these are the eigenvectors output in the DMAP data block PSI, and they are not orthogonal to PHI unless properly converted.

Dynamic Matrix Multiplication

In any eigenvalue solution method, the dynamic matrix times vector (block) multiplication is the most time consuming. From **Eq. 7-12** and **Eq. 7-13**, the dynamic matrix A is

$$A = \begin{bmatrix} -B & -\lambda_0 M & -K \\ I & & -\lambda_0 I \end{bmatrix}^{-1} \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \quad \text{Eq. 7-39}$$

From its structure it is clear that the matrix does not need to be built explicitly. In the following, the implicit execution of the dynamic matrix multiplication for both the transpose and non-transpose case is detailed.

For the non-transpose case, any $z = Ax$ operation in the recurrence will be equivalent to solving the following system of equations:

$$\begin{bmatrix} -B & -\lambda_0 M & -K \\ I & & -\lambda_0 I \end{bmatrix} z = \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} x \quad \text{Eq. 7-40}$$

Partitioning z and x accordingly:

$$\begin{bmatrix} -B & -\lambda_0 M & -K \\ I & & -\lambda_0 I \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{Eq. 7-41}$$

Developing the first row:

$$(-B - \lambda_0 M)\, z_1 - K\, z_2 = M\, x_1 \quad \text{Eq. 7-42}$$

Developing the second row and rearranging, we obtain:

$$z_1 = \lambda_0\, z_2 + x_2 \quad \text{Eq. 7-43}$$

Substituting the latter into the first row gives:

$$(-B - M\, \lambda_0)\, (\lambda_0\, z_2 + x_2) - K\, z_2 = M\, x_1 \quad \text{Eq. 7-44}$$

Which, after reordering, becomes:

$$-(K + \lambda_0\, B + \lambda_0^2\, M)\, z_2 = M\, x_1 + (B + \lambda_0\, M)\, x_2 \quad \text{Eq. 7-45}$$

This formulation has significant advantages. Besides avoiding the explicit formulation of A, the decomposition of the 2N size problem is also avoided.

It is important that the transpose operation be executed without any matrix transpose. Any $y^T = x^T A$ operation in the recurrence will be equivalent to solving the following equation for y^T:

$$y^T = x^T \begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^{-1} \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \quad \text{Eq. 7-46}$$

Let us introduce an intermediate vector z:

$$z^T = x^T \begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^{-1} \quad \text{Eq. 7-47}$$

Now partitioning these vectors accordingly and transforming we obtain:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -B - \lambda_0 M & -K \\ I & -\lambda_0 I \end{bmatrix}^T \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \text{Eq. 7-48}$$

Developing the first row:

$$x_1 = (-B - \lambda_0 M)^T z_1 + z_2 \quad \text{Eq. 7-49}$$

Developing the second row, we have:

$$x_2 = -K^T z_1 - \lambda_0 z_2 \quad \text{Eq. 7-50}$$

Solving for z_2 from the first equation, substituting into the second and reordering yields:

$$x_2 + \lambda_0 x_1 = -(K^T + \lambda_0 B^T + \lambda_0^2 M^T) z_1 \quad \text{Eq. 7-51}$$

or solving for z_1:

$$z_1 = -(K^T + \lambda_0 B^T + \lambda_0^2 M^T)^{-1} (x_2 + \lambda_0 x_1) \quad \text{Eq. 7-52}$$

Now, the lower part of the z vector is recovered from **Eq. 7-49** by:

$$z_2 = x_1 + (B + \lambda_0 M)^T z_1 \quad \text{Eq. 7-53}$$

Finally,

$$y^T = z^T \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \quad \text{Eq. 7-54}$$

Physical Solution Diagnosis

From the eigenvalue solution, it will be guaranteed that the left and right mathematical eigenvectors are bi-orthonormal:

$$Y^H X = I = XY^H \quad \text{Eq. 7-55}$$

where I is an identity matrix in essence with computational zeroes as off diagonal terms.

Based on the physical eigenvalues recovered by the shift formulae and the physical eigenvectors, another orthogonality criterion can be formed. Using the left and right solutions, the following equations hold for the problem:

$$(\lambda_i^2 M + \lambda_i B + K)\phi_i = 0 \quad \text{Eq. 7-56}$$

$$\psi_j^H(\lambda_j^2 M + \lambda_j B + K) = 0 \quad \text{Eq. 7-57}$$

By appropriate pre- and post-multiplications, we get:

$$\psi_j^H(\lambda_i^2 M + \lambda_i B + K)\phi_i = 0 \quad \text{Eq. 7-58}$$

$$\psi_j^H(\lambda_j^2 M + \lambda_j B + K)\phi_i = 0 \quad \text{Eq. 7-59}$$

A subtraction yields:

$$(\lambda_i^2 - \lambda_j^2)\psi_j^H M\phi_i + (\lambda_i - \lambda_j)\psi_j^H B\phi_i = 0 \quad \text{Eq. 7-60}$$

Assuming $\lambda_j \neq \lambda_i$, we can divide by $\lambda_i - \lambda_j$ and obtain a mass orthogonality criterion:

$$[O_1]_{ji} = (\lambda_i + \lambda_j)\psi_j^H M\phi_i + \psi_j^H B\phi_i \quad \text{Eq. 7-61}$$

Thus, the O_1 matrix given by **Eq. 7-61** has (computational) zeroes as off diagonal terms (when $i \neq j$) and nonzero (proportional to $2\lambda_i$) diagonal terms (corresponding to $i = j$).

To obtain another orthogonality condition, we premultiply **Eq. 7-56** by $\lambda_j \psi_j^H$, postmultiply **Eq. 7-57** by $\lambda_i \phi_i$ and subtract to obtain:

$$\lambda_j \psi_j^H (\lambda_i^2 M + \lambda_i B + K)\phi_i - \lambda_i \psi_j^H (\lambda_j^2 M + \lambda_j B + K)\ \phi_i = 0 \qquad \text{Eq. 7-62}$$

By expanding and simplifying we get:

$$(\lambda_i - \lambda_j)\lambda_i\ \lambda_j \psi_j^H M \phi_i + (\lambda_j - \lambda_i)\psi_j\ K\phi_i = 0 \qquad \text{Eq. 7-63}$$

Assuming again that $\lambda_j \neq \lambda_i$ we can divide by $\lambda_i - \lambda_j$ and obtain another orthogonality condition recommended mainly for the structural damping option as:

$$[O_2]_{ji} = \lambda_i \lambda_j\ \psi_j^H M \phi_i - \psi_j^H K\ \phi_i \qquad \text{Eq. 7-64}$$

The related O_2 orthogonality matrix will also have zero off-diagonal terms, and nonzero (proportional to λ_i^2) diagonal terms.

Hessenberg Method

The method utilized in NX Nastran uses the Householder reduction to upper Hessenberg form followed by Francis's QR steps and a direct eigenvector generation scheme. To distinguish from the QZ method, this is called the QR Hessenberg method.

Householder Transformations. An n by n matrix with the following form:

$$P = I - \frac{2vv^T}{v^T v} \qquad \text{Eq. 7-65}$$

was introduced in "**Theory of Real Eigenvalue Analysis**" on page 125 as a Householder transformation or reflection. These matrices are symmetric and orthogonal, and are capable of zeroing out specified entries or any block of vector components. We generate a Householder matrix for a given nonzero vector x:

$$x^T = [x_1, \ldots, x_n] \qquad \text{Eq. 7-66}$$

so that for $1 \leq k \leq j \leq n$ the elements from $k + 1$ to j of the transformed vector are zero; i.e.,

$$Px = [x_1, \ldots, \overline{x_k}, 0, \ldots, 0, x_{j+1}, \ldots, x_n]^T \qquad \text{Eq. 7-67}$$

The procedure is as follows. We calculate

$$a^2 = x_k^2 + \ldots + x_j^2 \qquad \text{Eq. 7-68}$$

and build

$$v^T = [0, \ldots, 0, \{x_k + a \text{ sign } (x_k)\}, x_{k+1}, \ldots, x_j, 0, \ldots, 0] \quad \text{Eq. 7-69}$$

It can be shown that if P is defined as in **Eq. 7-65**, then

$$Px = [x_1, \ldots, x_{k-1} \{-\text{sign}(x_k)a\}, 0, \ldots, 0, x_{j+1}, \ldots, x_n]^T \quad \text{Eq. 7-70}$$

which is the form described in **Eq. 7-67**. The matrix update is similar:

$$PA = (I - \beta vv^T)A = A - \beta\, v(A^T v)^T \quad \text{Eq. 7-71}$$

It is clear that P need not be formed explicitly if v and $\beta = 2/v^T v$ are available.

Implicit Matrix Update. For given values of A, v, and β, the following algorithm in NX Nastran overwrites A with PA, where

$$P = (I - \beta vv^T)$$

For $p = 1, \ldots, n$

$$s \leftarrow v(k)A(k, p) + \ldots + v(j)A(j, p)$$
$$s \leftarrow \beta s$$

For $i = k, \ldots, j$

$$A(i, p) \leftarrow A(i, p) - s\, v(i)$$

End loop i.

End loop p.

Note: $v^T = [0, \ldots, v_k, \ldots, v_j, 0, \ldots, 0]$ above. An analogous algorithm is used for the AP update.

Householder Reduction to the Hessenberg Form. Let A be the general matrix on which the transformation must be executed. Consider the following transformation:

$$A_r = P_r\, A_{r-1}\, P_r \quad \text{Eq. 7-72}$$

where:

$$A_0 = A$$

$$P_r = I - 2w_r\, w_r^T \text{ (symmetric)}$$

$$w_r^T w_r = 1$$

The elements of w_r are chosen so that A_r has zeroes in the positions 1 through $r-2$ in the r-th row. The configuration and partitioning of A_{r-1} can be shown as:

$$A_{r-1} = \begin{matrix} r \\ n-r \end{matrix} \left\{ \begin{matrix} \left[\begin{matrix} x & x & x & x & x & x \\ x & x & x & x & x & x \end{matrix}\right. \\ \left[\begin{matrix} & x & x & x & x & x \\ & & x & x & x & x \\ & & & x & x & x \\ & & & x & x & x \end{matrix}\right. \end{matrix} \right. = \left[\begin{array}{c|c|c} H_{r-1} & & C_{r-1} \\ \hline 0 & b_{r-1} & B_{r-1} \end{array}\right]$$

where:

C_{r-1} = a submatrix of order r

H_{r-1} = of upper Hessenberg form

B_{r-1} = a square matrix order $n-r$ (part of original matrix)

b_{r-1} = a vector having $n-r$ components

The transformation matrix can be partitioned as follows:

$$P_r = \begin{matrix} r \\ n-r \end{matrix} \left\{ \left[\begin{array}{c|c} I & 0 \\ \hline 0 & Q_r \end{array}\right] \right. = \left[\begin{array}{c|c} I & 0 \\ \hline 0 & I - 2v_r v_r^T \end{array}\right]$$

where v_r is a unit vector of order $n-r$.

By executing the multiplication given in the right-hand side of **Eq. 7-72**, the following A_r is obtained:

$$A_r = \begin{matrix} r \\ n-r \end{matrix} \left\{ \left[\begin{array}{c|c} H_{r-1} & C_{r-1}Q_r \\ \hline 0 \;\; c_r & Q_r B_{r-1} Q_r^T \end{array}\right] \right.$$

where $c_r = Q_r\, b_{r-1}$.

If v_r is chosen so that c_r is null except for its first component, then H_r of the order $r+1$ takes the Hessenberg form.

In-Memory Algorithm. This formulation can be developed by writing P_r as follows:

$$P_r = I - 2w_r w_r^T \quad \text{Eq. 7-73}$$

where w_r is a unit vector of order n with zeroes as the first r elements. By further changes:

$$P_r = I - \frac{u_r u_r^T}{2K_r^2} \quad \text{Eq. 7-74}$$

where:

$$u_{ir} = 0,\ i = 1, 2, \ldots, r$$

$$u_{r+1,\,r} = a_{r,\,r+1} \pm S_r$$

$$u_{ir} = a_{ir},\ i = r+2, \ldots, n$$

$$S_r^2 = \sum_{i=r+1}^{n} a_{ir}^2$$

$$2K_r^2 = S_r^2 \pm a_{r+1,\,r}\, S_r$$

Because of the lack of symmetry, the pre- and post-multiplications in **Eq. 7-72** must be considered separately. The premultiplication takes the following form:

$$P_r A_{r-1} = \left(I - \frac{u_r u_r^T}{2K_r^2}\right) A_{r-1} = A_{r-1} - \frac{u_r (u_r^T A_{r-1})}{2K_r^2} = F_r \quad \text{Eq. 7-75}$$

The postmultiplication is as follows:

$$A_r = F_r\, P_r = F_r \left(I - \frac{u_r\, u_r^T}{2K_r^2}\right) = F_r - \frac{(F_r\, u_r) u_r^T}{2K_r^2} \quad \text{Eq. 7-76}$$

Because u_r has zero elements in the 1 through r positions, it is seen that the premultiplication leaves the first r rows of A_{r-1} unchanged, while the postmultiplication leaves the first r columns unchanged. By introducing new intermediate vectors, the following may be written:

$$u_r^T A_{r-1} = p_r^T \quad \text{Eq. 7-77}$$

where p_r^T has its first $(r-1)$ elements as zero. This results in

$$F_r = A_{r-1} - \left(\frac{u_r}{2K_r^2}\right) p_r^T \qquad \text{Eq. 7-78}$$

For the postmultiplication, the q_r vector is introduced

$$F_r\, u_r = q_r \qquad \text{Eq. 7-79}$$

which has no zero components. Finally,

$$A_r = F_r - q_r \left(\frac{u_r}{2K_r^2}\right)^T \qquad \text{Eq. 7-80}$$

Two-Level Storage (Spill) Algorithm. If the execution of memory transfers is a significant requirement, the following formulation is more convenient. The vector p_r is again defined as in **Eq. 7-77**:

$$p_r^T = u_r^T\, A_{r-1} \qquad \text{Eq. 7-81}$$

However, q_r is now defined as follows:

$$q_r = A_{r-1}\, u_r \qquad \text{Eq. 7-82}$$

By scaling elements of u_r, the following is obtained:

$$v_r = \frac{u_r}{2K_r^2} \qquad \text{Eq. 7-83}$$

Finally, introducing the scalar a_r as follows:

$$a_r = p_r^T v_r \qquad \text{Eq. 7-84}$$

Eq. 7-72 now can be written in the following form:

$$\begin{aligned} A_r &= A_{r-1} - v_r\, p_r^T - q_r\, v_r^T + (\, p_r^T v_r) u_r\, v_r^T \\ &= A_{r-1} - v_r\, p_r^T - (q_r - a_r\, u_r)\, v_r^T \end{aligned} \qquad \text{Eq. 7-85}$$

QR Iteration Using the Householder Matrices

It is proven (Wilkinson, 1965) that the general A matrix can be factored into the product of a unitary matrix Q and an upper triangular matrix R. The algorithm at the r-th stage is as follows:

$$A_r = Q_r R_r$$
$$A_{r+1} = Q_r^H A_r Q_r = Q_r^H Q_r R_r Q_r = R_r Q_r \quad \text{Eq. 7-86}$$

By including the successive iterates, we obtain

$$A_{r+1} = Q_r^H A_r Q_r = (Q_r^H Q_{r-1}^H \dots Q_1^H) A_1 (Q_1 Q_2 \dots Q_r) \quad \text{Eq. 7-87}$$

From the following form of **Eq. 7-86**:

$$(Q_1 Q_2 \dots Q_r) A_{r+1} = A_1 (Q_1 Q_2 \dots Q_r) \quad \text{Eq. 7-88}$$

it is seen that A_r is unitarily similar to A_1. In general, A_r tends towards an upper triangular form. The matrix Q_r is the product of $(n-1)$ elementary unitary transformations necessary to reduce A_r to the upper triangular form R_r. The transformation matrices can be Givens rotations or Householder reflections.

In NX Nastran the latter Householder reflections are used. A shift k_r may be incorporated into the single step QR logic as follows:

$$A_r - k_r I = Q_r R_r$$
$$A_{r+1} = R_r Q_r + k_r I \quad \text{Eq. 7-89}$$

If the matrix has complex conjugate eigenvalues, the most economical way is to execute two steps (double step). For example, the first double shift with shifts k_1 and k_2 can be written as follows:

$$A_1 - k_1 I = Q_1 R_1 \qquad A_2 = R_1 Q_1 + k_1 I$$
$$A_2 - k_2 I = Q_2 R_2 \qquad A_3 = R_2 Q_2 + k_2 I \quad \text{Eq. 7-90}$$

Note that **Eq. 7-87** still holds in the shifted case, so we have

$$A_3 = (Q_1 Q_2)^H A_1 Q_1 Q_2 \quad \text{Eq. 7-91}$$

By introducing

$$Q = Q_1 Q_2$$
$$R = R_2 R_1 \quad \text{Eq. 7-92}$$

the following can be shown:

$$QR = (A_1 - k_1 I)(A_1 - k_2 I) \quad \text{Eq. 7-93}$$

or

$$Q^T(A_1 - k_1 I)(A_1 - k_2 I) = R \qquad \text{Eq. 7-94}$$

which means that Q^T is the orthogonal matrix that reduces $(A_1 - k_1 I)(A_1 - k_2 I)$ to the upper triangular form. Note that if A_1 is real and the complex shifts k_1, k_2 are chosen as a conjugate pair, then the factorization in **Eq. 7-93** can be performed using only real arithmetic.

Implicit QR Step. Using the Householder matrices again,

$$P_r = I - 2w_r w_r^T \qquad \text{Eq. 7-95}$$

where $w_r = (0, 0, \ldots, 0, x, \ldots, x)$

The derivation of A_3 can be produced as follows. First we create a Householder matrix which provides the following:

$$P_1 x = ke_1 \qquad \text{Eq. 7-96}$$

where:

$x^T = (x_1, y_1, z_1, 0, \ldots, 0)$

$x_1 = (a_{11} - k_1)(a_{11} - k_2) + a_{12}\, a_{21}$

$y_1 = a_{21}(a_{11} - k_2) + (a_{22} - k_1)a_{21}$

$z_1 = a_{32}\, a_{21}$

$e_1 =$ first unit vector

Then we compute the following:

$$C_1 = P_1 A_1 P_1 \qquad \text{Eq. 7-97}$$

(See the update with Householder matrices in "**Theory of Real Eigenvalue Analysis**" on page 125.)

The C_1 matrix no longer takes Hessenberg form. Therefore, this matrix must be reduced by new Householder matrices:

$$P_{n-2}\ldots P_2 P_1 A_1 P_1 P_2 \ldots P_{n-2} = A_3 \qquad \text{Eq. 7-98}$$

Now the question is how to formulate the P_r matrices. The nonzero elements of the P_r Householder matrices are determined by x_1, y_1, z_1 for P_1 and by x_r, y_r, z_r for P_r, $r = 2, \ldots, n-2$. A convenient representation of P_r (see also "**Theory of Real Eigenvalue Analysis**" on page 125 on Householder method) for the current case is as follows:

$$P_r = I - \frac{2v_r v_r^T}{v_r^T v_r}$$

$$v_r^T = (0, \ldots, 0, 1, u_r, v_r, 0, \ldots, 0)$$

$$u_r = \frac{y_r}{(x_r \pm a_r)} \qquad \text{Eq. 7-99}$$

$$v_r = \frac{z_r}{(x_r \pm a_r)}$$

$$\frac{2}{v_r^T v_r} = \frac{2}{(1 + u_r^2 + v_r^2)} = \beta_r$$

Then

$$P_r = I - v_r(\beta_r\, v_r^T) \qquad \text{Eq. 7-100}$$

For the update, the algorithm described in "**Theory of Real Eigenvalue Analysis**" on page 125 is used.

Eigenvector Computation

The Hessenberg form is convenient for generating eigenvectors when the eigenvalues are known. An eigenvector of the Hessenberg matrix H can be found by solving the following:

$$(H - \lambda\, I)y = 0 \qquad \text{Eq. 7-101}$$

where λ is the corresponding eigenvalue.

This equation in detailed form is as follows:

$$\begin{bmatrix} h_{11} - \lambda & h_{12} & h_{13} & h_{1n} \\ h_{21} & h_{22} - \lambda & h_{23} & h_{2n} \\ & h_{i,\,i-1} & h_{i,\,i} - \lambda & h_{in} \\ & & \ddots & \\ & & h_{n,\,n-1} & h_{nn} - \lambda \end{bmatrix} \begin{bmatrix} y_1 \\ \\ y_i \\ y_{n-1} \\ y_n \end{bmatrix} = 0 \qquad \text{Eq. 7-102}$$

The matrix of the system is singular, but the minor corresponding to the upper right corner element h_{1n} is not. Therefore, we normalize the y vector so that the last element in it is equal to 1.

$$y_n = 1 \qquad \text{Eq. 7-103}$$

The next to the last element in this case is given by solving the following:

$$h_{n,\,n-1}\,y_{n-1} + (h_{nn} - \lambda)y_n = 0 \qquad \text{Eq. 7-104}$$

which yields the following:

$$y_{n-1} = \frac{1}{h_{n,\,n-1}}(\lambda - h_{nn}) \qquad \text{Eq. 7-105}$$

The other elements can be evaluated in recursive form from:

$$h_{i,\,i-1}\,y_{i-1} + (h_{i,\,i} - \lambda)\,y_i + \sum_{l\,=\,i+1}^{n} h_{i,\,l}\,y_l = 0 \qquad \text{Eq. 7-106}$$

Then the $(i-1)$-th element, $i = n-1, n-2, \ldots, 2$ is as follows:

$$y_{i-1} = \frac{1}{h_{i,\,i-1}}\left[(\lambda - h_{i,\,i})\,y_i - \sum_{l\,=\,i+1}^{n} h_{i,\,l}\,y_l\right] \qquad \text{Eq. 7-107}$$

The practical implementation of the above procedure also takes care of the cases where decoupling occurs, i.e., when $h_{i,\,i-1} = 0$. This process is still very unstable and an alternative iterative solution of **Eq. 7-101** exists in NX Nastran, which is similar to the procedure detailed in "**Theory of Real Eigenvalue Analysis**" on page 125.

Transformation of the Vectors. If **Eq. 7-72** is extended, the following is obtained:

$$H = P_{n-2}P_{n-1}\ldots P_1\,A_1\,P_1\ldots P_{n-2} \qquad \text{Eq. 7-108}$$

By introducing

$$Z = P_1\,\ldots\,P_{n-2} \qquad \text{Eq. 7-109}$$

the following is obtained:

$$H = Z^T A Z \qquad \text{Eq. 7-110}$$

or

$$AZ = ZH \qquad \text{Eq. 7-111}$$

If **Eq. 7-111** is post-multiplied by an eigenvector of the H matrix, the following is obtained:

$$AZy = ZHy \quad \text{Eq. 7-112}$$

Note from Eq. 7-101 that

$$Hy = \lambda y \quad \text{Eq. 7-113}$$

The result is the following equation:

$$AZy = \lambda Zy \quad \text{Eq. 7-114}$$

which indicates that an eigenvector x of A can be obtained by the following:

$$x = Zy \quad \text{Eq. 7-115}$$

This calculation is also straightforward if the Z matrix is accumulated during the transformation from A to H. However, it is not practical to explicitly form the Z matrix when the P_r matrices are not formed. An effective implicit formulation is given below.

Implicit Vector Transformation. Write Eq. 7-115 as follows:

$$x = P_1 P_2 \ldots P_{n-2} y \quad \text{Eq. 7-116}$$

where P_i represents the Householder transformation matrices. Considering Eq. 7-116 as a series of orthogonal transformations with $y = y^{(n-2)}$ and $x = y^{(0)}$, write the following:

$$y^{(r-1)} = P_r y^{(r)} \quad \text{Eq. 7-117}$$

and substitute to get

$$y^{(r-1)} = y^{(r)} - \frac{u_r^T y^{(r)}}{2K_r^2} u_r \quad \text{Eq. 7-118}$$

With this formulation, only the u_r vectors need to be accumulated during the A to H transformation and saved in secondary storage for the vector transformations.

QZ Hessenberg Method

The QZ method is a robust algorithm for computing eigenvalues and eigenvectors of the eigenvalue problem

$$Ax = \lambda Cx$$

and, as such, is applicable to the $B = 0$ case. If $B \neq 0$, the appropriate canonical transformation to linear form is executed. There are no restrictions on A or C. For details, see Golub and Van Loan, p. 375.

Hessenberg-Triangular Form

The first stage of the QZ algorithm is to determine unitary matrices $\bar{Q}$ and $\bar{Z}$ so that the matrix $T = \bar{Q}^H C \bar{Z}$ is upper triangular and $S = \bar{Q}^H A \bar{Z}$ is upper Hessenberg.

This is accomplished by first computing a unitary matrix U such that $U^H C$ is upper triangular. Next, the matrix $U^H A$ is reduced to an upper Hessenberg matrix S by premultiplying by a series of unitary matrices Q_j and postmultiplying by a series of unitary matrices Z_j, $j = 1, \ldots, k$.

The matrices Q_j and Z_j are carefully chosen so that

$$S_j = Q_j^H Q_{j-1}^H \ldots Q_1^H (U^H A) Z_1 Z_2 \ldots Z_j = \bar{Q}^H A \bar{Z}$$

is upper Hessenberg, while

$$T_j = Q_j^H Q_{j-1}^H \ldots Q_1^H (U^H C) Z_1 Z_2 \ldots Z_j = \bar{Q}^H C \bar{Z}$$

remains upper triangular.

The QZ Step

The derivation of the QZ step is motivated by the case where T is nonsingular (although it generalizes the singular case). If T is nonsingular, we could form ST^{-1} and apply the Francis QR algorithm. Rather than forming ST^{-1} explicitly, the QZ algorithm updates S and T using unitary matrices Q and Z:

$$S_j = Q^H S Z$$
$$T_j = Q^H T Z$$

The matrices Q and Z are chosen so that the matrix $S_j T_j^{-1}$ is essentially the same as if a QR step had been explicitly applied to ST^{-1}. However, since we operate with S and T rather than ST^{-1}, it is not necessary for T to be invertible.

The QZ iteration continues until the matrix S converges to upper triangular form.

Eigenvalue Computation

Once S has converged to upper triangular form, the QZ algorithm will have determined unitary matrices $Q = Q_1, Q_2, \ldots, Q_j$ and $Z = Z_1, Z_2, \ldots, Z_j$ so that both $S = Q^H A Z$ and $T = Q^H C Z$ are upper triangular. Denote the diagonal entries of S by $\alpha_1, \alpha_2, \ldots, \alpha_n$, and the diagonal entries of T by $\beta_1, \beta_2, \ldots, \beta_n$.

Then, for each $j = 1, 2, \ldots, n$, the matrix $\beta_j S - \alpha_j T$ is singular. It follows that there exists a vector u_j so that $\beta_j S u_j = \alpha_j T u_j$.

Substituting for S and T and pre-multiplying by Q, we have $\beta_j A Z u_j = \alpha_j C Z u_j$.

Hence, if $\beta_j \neq 0$, we set $\lambda_j = \alpha_j/\beta_j$ and $x_j = Zu_j$ to get $Ax_j = \lambda_j Cx_j$, as desired. However, if $\beta_j = 0$, then we have two cases to consider:

1. $\beta_j = 0, \alpha_j \neq 0$ In this case, λ_j is said to be an infinite eigenvalue.
2. $\beta_j = 0, \alpha_j = 0$ Here λ_j is indeterminate.

The Complex Lanczos Method

The main steps of the complex Lanczos Method are: reduction to tridiagonal form, solution of the tridiagonal problem, and eigenvector computation.

NX Nastran currently supports two complex Lanczos methods: the single vector method and the adaptive, block method.

The Single Vector Method

Reduction to Tridiagonal Form. The recurrence procedure introduced in "**Theory of Real Eigenvalue Analysis**" on page 125 can also be used to reduce a general matrix to tridiagonal form. This is the so-called biorthogonal Lanczos method. Find a nonsingular V matrix such that:

$$AV = VT \qquad \text{Eq. 7-119}$$

or

$$V^{-1}AV = T = \begin{bmatrix} \alpha_1 & \gamma_2 & & & \\ \beta_2 & \alpha_2 & \gamma_3 & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & \cdot & \gamma_n \\ & & & B_n & \alpha_n \end{bmatrix} \qquad \text{Eq. 7-120}$$

Then with $U = V^{-T}$, the following relation holds:

$$A^T U = UT^T \qquad \text{Eq. 7-121}$$

Eq. 7-119 and **Eq. 7-121** can be written as:

$$\begin{aligned} Av_j &= \gamma_j v_{j-1} + \alpha_j v_j + \beta_{j+1} v_{j+1} \\ A^T u_j &= \beta_j u_{j-1} + \alpha_j u_j + \gamma_{j+1} u_{j+1} \end{aligned} \qquad \text{Eq. 7-122}$$

where $\gamma_1 v_0 = \beta_1 u_0 = 0$ and for $j = 1, 2, \ldots, n-1$, u_j and v_j are complex columns of the U and V matrices.

Reordering Eq. 7-122 results in the following recurrence relations:

$$\begin{aligned} \beta_{j+1} v_{j+1} &= A v_j - \alpha_j v_j - \gamma_j v_{j-1} \\ \gamma_{j+1} u_{j+1} &= A^T u_j - \alpha_j u_j - \beta_j u_{j-1} \end{aligned} \qquad \text{Eq. 7-123}$$

There is some flexibility in choosing the scale factors of β_j, γ_j. One possibility is to select $\beta_j = 1$, which results in an unsymmetric T matrix with ones on the subdiagonal. This form is advantageous for a direct eigenvector generation scheme.

Another possibility is to select $\gamma_j = \beta_j$, which results in a complex but symmetric T matrix. This matrix has significant advantages in the eigenvalue extraction from the tridiagonal form, and NX Nastran uses this method in the implementation. In this case, Eq. 7-123 becomes

$$\begin{aligned} \beta_{j+1} v_{j+1} &= A v_j - \alpha_j v_j - \beta_j v_{j-1} \\ \beta_{j+1} u_{j+1} &= A^T u_j - \alpha_j u_j - \beta_j u_{j-1} \end{aligned} \qquad \text{Eq. 7-124}$$

which corresponds to the matrix form (see Eq. 7-120):

$$U^T A V = T = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & & & \\ & \cdot & \cdot & & \\ & & \cdot & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{bmatrix} \qquad \text{Eq. 7-125}$$

The explicit form of the coefficients of T can be derived from the biorthonormality of vectors u and v. Biorthonormality means that

$$u_i^T v_j = \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases} \qquad \text{Eq. 7-126}$$

The premultiplication of Eq. 7-124 by u_j^T, u_{j+1}^T, and v_j^T, v_{j+1}^T, respectively, results in the following (using Eq. 7-126):

$$\alpha_j^u = u_i^T A v_j \text{ or } \alpha_j^v = v_j^T A u_j$$

and

$$\beta_{j+1}^u = u_{j+1}^T A v_j \text{ or } \beta_{j+1}^v = v_{j+1}^T A^T u_j \qquad \text{Eq. 7-127}$$

The two versions of α_j and β_{j+1} should be the same in exact arithmetic. In the actual implementation, the average of the two versions is used.

The algorithm can be developed as follows:

$$\begin{aligned}
\alpha_j^u = u_j^T A v_j \; ; \; \alpha_j^v = v_j^T A u_j \; ; \; \alpha_j = (\alpha_j^u + \alpha_j^v)/2 \\
\bar{v}_{j+1} = A v_j - \alpha_j \, v_j - \beta_j \, v_{j-1} \\
\bar{u}_{j+1} = A^T u_j - \alpha_j \, u_j - \beta_j \, u_{j-1} \\
(\beta_{j+1}^u)^2 = \bar{u}_{j+1}^T A \, v_j \; ; \; (\beta_{j+1}^v)^2 = \bar{v}_{j+1}^T A^T u_j \; ; \; \beta_{j+1} = (\beta_{j+1}^u + \beta_{j+1}^v)/2 \\
v_{j+1} = \frac{\bar{v}_{j+1}}{\beta_{j+1}} \\
u_{j+1} = \frac{\bar{u}_{j+1}}{\beta_{j+1}}
\end{aligned} \qquad \text{Eq. 7-128}$$

where $j = 1, \ldots, m < n$. The algorithm starts with $u_0 = v_0 = 0, \beta_1 = 0$ as well as with biorthonormal random starting vectors for u_1 and v_1.

The procedure will break down if β_{j+1} becomes equal to zero. In that case, the process is restarted with new u_{j+1}, v_{j+1} vectors. In our implementation this is done when

$$\beta_{j+1} < \varepsilon \alpha_j \qquad \text{Eq. 7-129}$$

where ε is a small number (related to the machine precision).

Solution of Tridiagonal Problem. The method for extracting the eigenvalues of the complex tridiagonal form is a variant of the basic QL procedure, mentioned previously in "**Theory of Real Eigenvalue Analysis**" on page 125. At each iteration the basic QL procedure factors a shifted version of the current iterate as follows:

$$T - \omega I = QL \qquad \text{Eq. 7-130}$$

where L is a lower triangular matrix, Q is orthogonal and obeys the following:

$$Q^T Q = I \qquad \text{Eq. 7-131}$$

The next iterate T_{i+1} is obtained as follows:

$$T_1 = LQ + \omega I \qquad \text{Eq. 7-132}$$

Premultiplying by Q^T and postmultiplying by Q in **Eq. 7-130** gives

$$T_1 = Q^T T Q \qquad \text{Eq. 7-133}$$

By repeatedly applying **Eq. 7-130** and **Eq. 7-133**, T_i is finally converted into a diagonal matrix whose elements are the eigenvalues of the original tridiagonal matrix; that is,

$$T_n = \Lambda \qquad \text{Eq. 7-134}$$

Note that if the T matrix is complex, then the Q matrix is also complex.

The computation of Q is performed by a sequence of complex Givens transformation matrices P_k. Each P_k is the identity matrix except for the entries $P_k(i,j)$ where $k \le i, j \le k+1$. These terms are defined as follows:

$$\begin{aligned} -P_k(k,\,k) &= P_k(k+1,\,k+1) = C_k \\ P_k(k+1,\,k) &= P_k(k,\,k+1) = S_k \\ C_k^2 + S_k^2 &= 1 \end{aligned} \qquad \text{Eq. 7-135}$$

C_k and S_k are complex scalars (to be defined later). Each P_k matrix satisfies

$$P_k^T = P_k = P_k^{-1} \qquad \text{Eq. 7-136}$$

The Q matrix is built as follows:

$$Q = P_{n-1}P_{n-2}\cdots P_1 \qquad \text{Eq. 7-137}$$

First, P_{n-1} is determined from the n-th column of $T - \omega I$. Applying this transformation matrix to T, a new nonzero term is introduced in the $(n-2,\,n)$ position. Subsequent P_j are defined so that the nonzero introduced by P_{n-1} is forced up and out of the matrix, thereby preserving the complex symmetric tridiagonal structure.

The algorithm given below does not perform the explicit shifting, factorization, and recombination of **Eq. 7-130** and **Eq. 7-132**. Explicit shifting can result in significant loss of accuracy. Instead, one iteration step (generating T_{i+1} from T_i), is replaced by the following sequence:

$$T_i^{(k)} = P_k\, T_i^{(k-1)} P_k,\; k = n-1,\, \ldots,\, 1 \qquad \text{Eq. 7-138}$$

where $T_0^{(k)} = T$, the original matrix.

Finally, the following is obtained:

$$T_{i+1}^{(n)} = T_i^{(1)} \qquad \text{Eq. 7-139}$$

The process is repeated for $i = 1, 2, \ldots, m$ or until the matrix T_i becomes diagonal. Note that this diagonalization occurs by successive decoupling of 1×1 and 2×2 submatrices of T_i. Decoupling occurs when an off-diagonal term is less than the product of an ε and the sum of the absolute values of the two corresponding diagonal entries.

The selection of the C_k, S_k parameters can now be described. For $k = n - 1$, we must first determine the shift ω, which is chosen to be the eigenvalue of the upper 2×2 submatrix of $T_i^{(n)}$ closest to $T_i^{(n)}(1, 1)$. Then the parameters are as follows:

$$C_{n-1} = \frac{T_i^{(n)}(n, n) - \omega}{a_{n-1}} \qquad \text{Eq. 7-140}$$

$$S_{n-1} = \frac{T_i^{(n)}(n-1, n)}{a_{n-1}} \qquad \text{Eq. 7-141}$$

where

$$a_{n-1} = (T_i^{(n)}(n, n) - \omega)^2 + T_i^{(n)}(n-1, n)^2 \qquad \text{Eq. 7-142}$$

Note again that by executing this transformation, the term $T_i^{(n)}(n-2, n)$ becomes nonzero. For the subsequent rotations $k < n - 1$, the parameters are selected as follows:

$$C_k = \frac{T_i^{(k+1)}(k+1, k+2)}{a_k} \qquad S_k = \frac{T_i^{(k+1)}(k, k+2)}{a_k} \qquad \text{Eq. 7-143}$$

where $a_k = T_i^{(k+1)}(k+1, k+2)^2 + T_i^{(k+1)}(k, k+2)^2$

This step is reviewed in the following figure by using $k = n - 2$ (the sub- and superscripts of the T are ignored).

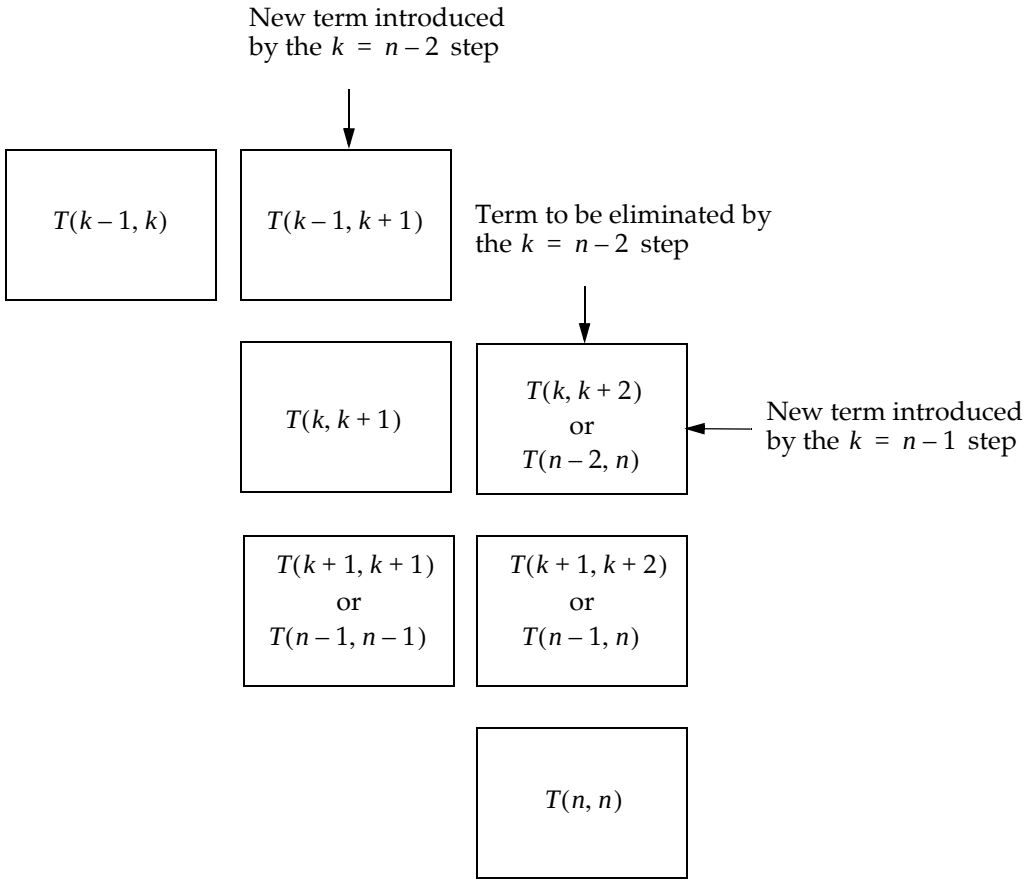


Figure 7-1 Chasing of Nonzero Offdiagonals.

As shown in **Figure 7-1**, the k-th transformation zeroes out $T(k,\ k+2)$, which was introduced by the $(k+1)$ th transformation, but generates a new nonzero term in $T(k-1,\ k+1)$.

Using **Eq. 7-143** the new terms of the matrix $T_i^{(k)}$ are calculated as follows. Let

$$b_k = 2C_k T_i^{(k+1)}(k, k+1) + S_k(T_i^{(k+1)}(k, k) - T_i^{(k+1)}(k+1, k+1)) \qquad \text{Eq. 7-144}$$

Then

$$\begin{aligned} T_i^{(k)}(k,\ k) &= T_i^{(k+1)}(k,\ k) - S_k\ b_k \\ T_i^{(k)}(k+1,\ k+1) &= T_i^{(k+1)}(k+1,\ k+1) - S_k\ b_k \\ T_i^{(k)}(k,\ k+1) &= T_i^{(k+1)}(k,\ k+1) - C_k\ b_k \\ T_i^{(k)}(k-1,\ k) &= C_k T_i^{(k)}(k-1,\ k) \\ T_i^{(k)}(k-1,\ k+1) &= S_k T_i^{(k)}(k-1,\ k) \end{aligned} \qquad \text{Eq. 7-145}$$

Finishing the calculation of Eq. 7-145, set $k \leftarrow k - 1$ and repeat from Eq. 7-144. When Eq. 7-145 is finished for $k = 1$, then use Eq. 7-142 and start the process from Eq. 7-144 again.

Breakdown of the Procedure. The calculation of c_k and s_k is performed using a denominator with the following form:

$$a^2 + b^2 \qquad \text{Eq. 7-146}$$

which can be very small without either a or b being small. For example, for $a = 1$ and $b = \sqrt{-1}$, $a^2 + b^2 = 0$. When this situation occurs, the process breaks down. The condition of this problem can be formulated as follows. If

$$\left|a^2 + b^2\right| \leq \varepsilon(|a|^2 + |b|^2) \qquad \text{Eq. 7-147}$$

then the procedure will terminate. ε in Eq. 7-147 is a small number related to the machine precision.

The storage and computational efficiency of this algorithm is excellent since it needs $O(n)$ storage and $O(n^2)$ operations.

Error Bounds. Theoretically, the solutions of the following reduced eigenproblem

$$Ts_i = \bar{\lambda}_i s_i \qquad \text{Eq. 7-148}$$

are approximations to the solutions of the original problem.

To estimate the accuracy of the solution, we use

$$\left|\lambda_i - \bar{\lambda}_i\right| \leq \left|\beta_{m+1} s_i(m)\right| \qquad \text{Eq. 7-149}$$

Eq. 7-149 is a generalization of a similar error bound in the current symmetric Lanczos code.

The error bound for the original eigenvalues of Eq. 7-6 can be found as follows:

$$\left|\frac{\lambda_i - \bar{\lambda}_i}{\bar{\lambda}_i}\right| \leq \frac{\left|\beta_{m+1} s_i(m)\right|}{\left|\bar{\lambda}_i\right|}$$

$$\text{for } B \neq 0: \quad \left|\frac{|p_i - \lambda_0|}{|\bar{p}_i - \lambda_0|} - 1\right| \leq \frac{\left|\beta_{m+1} s_i(m)\right|}{\left|\bar{\lambda}_i\right|} \qquad \text{Eq. 7-150}$$

$$\text{for } B = 0: \quad \left|\frac{\left|p_i^2 - \lambda_0\right|}{\left|\bar{p}_i^2 - \lambda_0\right|} - 1\right| \leq \frac{\left|\beta_{m+1} s_i(m)\right|}{\left|\bar{\lambda}_i\right|}$$

Eigenvector Computation. To calculate the eigenvectors of the tridiagonal form, an inverse power iteration procedure is used. First, a random right-hand side is generated for the inverse iteration. Then an LU decomposition of the tridiagonal matrix is performed by Gaussian elimination with partial pivoting. After a back substitution pass, the convergence of the approximate eigenvector is checked by its norm. If the norm is greater than one, then the eigenvector is accepted. This norm shows sufficient growth, assuming that the procedure began with a random right-hand side vector with elements less than one. Otherwise, the eigenvector is normalized, and the process is repeated with this vector as the right-hand side. The iteration is repeated up to three times.

Practice indicates that most of the time one iteration pass is sufficient. The computation is very stable, especially with partial pivoting. The process can be summarized as follows.

1. Decomposition

$$P(T - \lambda_i I) = LU \qquad \text{Eq. 7-151}$$

where:

T = the tridiagonal matrix

λ_i = an eigenvalue

P = permutation matrix (row interchanges)

L, U = factor matrices

2. Iteration

$$LUu_2 = u_1 \qquad \text{Eq. 7-152}$$

where:

u_1 = random starting vector

u_2 = approximate eigenvector

If $\|u_2\| < 1$ and the iteration count is less than 3, then

$$u_1 \leftarrow \frac{u_2}{\|u_2\|} \qquad \text{Eq. 7-153}$$

and the iteration continues; otherwise, u_2 is the eigenvector.

3. Converting the Solution

The conversion of eigenvalues is performed by the following equations:

$$\begin{aligned} p_i &= \bar{\lambda}_i + \lambda_0,\ B \neq 0 \\ p_i^2 &= \bar{\lambda}_i + \lambda_0,\ B = 0 \end{aligned} \qquad \text{Eq. 7-154}$$

The eigenvector conversion requires the computation of the following:

$$x_i = Vs_i, \text{ for the right vectors}$$

or

$$y_i = Us_i, \text{ for the left vectors}$$

where:

s_i = eigenvector of **Eq. 7-148** corresponding to $\bar{\lambda}_i$

$U = [u_1, \ldots, u_m]$

$V = [v_1, \ldots, v_m]$

m = number of roots found by the procedure at the particular shift

4. Initialization

   The Lanczos recurrence is started with orthonormalized random vectors. Any numerical procedure can be used that generates sufficiently long pseudorandom number sequences. Suppose the random vectors are as follows:

   $$\bar{u}_1,\ \bar{v}_1$$

   Orthonormalization can be performed in the following form:

   $$\begin{aligned} u_1 &= \frac{\bar{u}_1}{(\bar{u}_1^T \bar{v}_1)^{1/2}} \\ v_1 &= \frac{\bar{v}_1}{(\bar{u}_1^T \bar{v}_1)^{1/2}} \end{aligned} \qquad \text{Eq. 7-155}$$

   These vectors are now suitable to start the Lanczos iteration.

5. Outer Orthogonalization

   It is well known that repeated occurrence of eigenvectors can be prevented by assuring orthogonality between the already accepted eigenvectors and the Lanczos vectors. This outer orthogonality can be maintained using the following iteration process (i.e. modified Gram-Schmidt):

$$v_i^j \leftarrow v_i^j - \sum_{k=1}^{c} (x_k^T v_i) x_k$$
$$\text{for} \quad j = 1, 2, \ldots, l \quad \text{until}$$
$$\left| x_k^T v_i^l \right| < \varepsilon \text{ for max } k = 1, \ldots, c \qquad \text{Eq. 7-156}$$

   This process should be executed similarly with the u_i vectors.

   The respective orthogonalization formula is as follows:

$$u_i^j \leftarrow u_i^j - \sum_{k=1}^{c} (y_k^T u_i) y_k$$

   In the above formulae, x_k, y_k are already accepted eigenvectors and $k = 1, 2, \ldots, c$.

6. Inner Orthogonalization

   When the Lanczos process is carried out in practice using finite precision arithmetic, strict biorthogonality of the u, v sequences is lost. The biorthogonality can be maintained if u_{j+1} is reorthogonalized with respect to $v_1, \ldots, v_{j+1}$, and v_{j+1} is reothogonalized with respect to $u_1, \ldots, u_{j+1}$ using the following formulae:

$$v_{j+1} \leftarrow v_{j+1} - \sum_{i=1}^{j} u_i^T v_{j+1} v_i$$
$$u_{j+1} \leftarrow u_{j+1} - \sum_{i=1}^{j} v_i^T u_{j+1} u_i \qquad \text{Eq. 7-157}$$

   For numerical stability, the subtraction is performed immediately after each term is calculated (modified Gram-Schmidt), and the sum is not accumulated.

   Inner orthogonality monitoring is performed by the following:

$$\omega_j^u = \left| u_{j+1}^T v_j \right|$$
$$\omega_j^v = \left| v_{j+1}^T u_j \right| \qquad \text{Eq. 7-158}$$

No orthogonalization is necessary when

$$\max\left(\omega_j^u, \omega_j^v\right) < \varepsilon$$

If necessary, the orthogonalization is performed against all previous vectors. If the orthogonalizations are kept in the core, this fact implies a limit on m.

7. Shift Strategy

   The default logic begins with a shift at (0.1, 1.0). The Lanczos recurrence process is performed until breakdown. Then the eigensolutions of the reduced problem are evaluated as approximations. If the user desires, the shifts are executed at user-specified locations as shown in **Figure 7-2**.

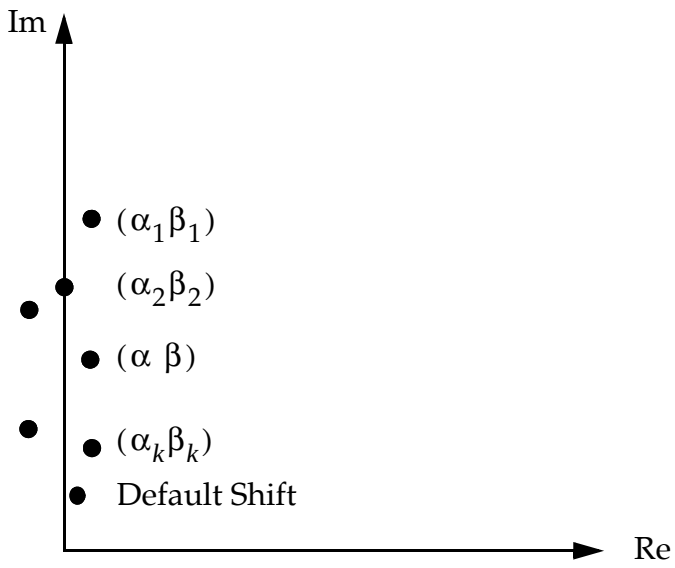


Figure 7-2 Recommended Shift Points for Complex Lanczos

The shifts are used until the required number of eigenvalues are found. Unfortunately, there is no analogue to the Sturm sequence theory used for the real eigensolution. As a result, there is no assurance that no gaps exist between the eigenvalues found.

The Adaptive Block Lanczos Method

The solution of the mathematical eigenproblem in its canonical form shown in **Eq. 7-13** will be more efficiently accomplished with the block Lanczos method.

The block Lanczos method (see Bai, et al., 1996) generates two sets of biorthonormal blocks of vectors P_j and Q_j such that:

$$P_i^H Q_j = I \qquad \text{Eq. 7-159}$$

when $i = j$ and zero otherwise. Note that we are using superscript H to denote the complex conjugate transpose. These vector sets reduce the A system matrix to T_j block tridiagonal matrix form:

$$T_j = \bar{P}_j^H A \bar{Q}_j \qquad \text{Eq. 7-160}$$

where the matrices

$$\bar{P}_j = P_1, P_2, \ldots, P_j \qquad \text{Eq. 7-161}$$

and

$$\bar{Q}_j = Q_1, Q_2, \ldots, Q_j \qquad \text{Eq. 7-162}$$

are the collections of the Lanczos blocks. The structure of the tridiagonal matrix is:

$$T_j = \begin{bmatrix} A_1 & B_2 & & \\ C_2 & A_2 & \ddots & \\ & \ddots & \ddots & B_j \\ & & C_j & A_j \end{bmatrix} \qquad \text{Eq. 7-163}$$

The block Lanczos process is executed by the following three term recurrence matrix equations:

$$B_{j+1} P_{j+1}^H = P_j^H A - A_j P_j^H - C_j P_{j-1}^H \qquad \text{Eq. 7-164}$$

and

$$Q_{j+1} C_{j+1} = A Q_j - Q_j A_j - Q_{j-1} B_j \qquad \text{Eq. 7-165}$$

Note that in both of these equations the transpose of the system matrix A is avoided.

In order to find the mathematical eigenvalues and eigenvectors, we solve the block tridiagonal eigenvalue problems posed as:

$$w^H T_j = \theta w^H \quad \text{Eq. 7-166}$$

and

$$T_j z = \theta z \quad \text{Eq. 7-167}$$

where the order of the reduced tridiagonal matrix is j times p, assuming a fixed block size p for now. The eigenvalues θ of the tridiagonal problem (the so-called Ritz values) are approximations to the eigenvalues Λ of the mathematical problem stated in **Eq. 7-13**. The approximations to the eigenvectors of the original problem are calculated from the left and right eigenvectors w, z of the tridiagonal problem (Ritz vectors) by:

$$y = \bar{P}_j w \quad \text{Eq. 7-168}$$

and

$$x = \bar{Q}_j z \quad \text{Eq. 7-169}$$

where $\bar{P}_j, \bar{Q}_j$ are the matrices containing the first j Lanczos blocks of vectors. Finally, x, y are the right and left approximated eigenvectors of the mathematical problem.

A beautiful aspect of the Lanczos method (exploited also in the READ module) is that the error norm of the original problem may be calculated from the tridiagonal solution, without calculating the eigenvectors. Let us introduce a rectangular matrix E_j having an identity matrix as the bottom square block. Using this, a residual vector for the left-handed solution is:

$$s^H = y^H A - \theta y^H = (w^H E_j) B_{j+1} P_{j+1}^H \quad \text{Eq. 7-170}$$

which means that only the bottom p (if the current block size is p) terms of the new Ritz vector w are required due to the structure of E_j. Similarly for the right-handed vectors:

$$r = Ax - \theta x = Q_{j+1} C_{j+1} (E_j^H z) \quad \text{Eq. 7-171}$$

An easy acceptance criterion (an extension of the one used in the real case) may be based on the norm of the above residual vectors as:

$$\min \left(\left\| s^H \right\|_2 \bullet \| r \|_2 \right) \leq \varepsilon_{con} \quad \text{Eq. 7-172}$$

where the ε_{con} value to accept convergence is either user given or related to an automatically calculated machine epsilon. The $\|.\|_2$ denotes the Euclidean norm.

Based on a detailed error analysis of these quantities, we modify this criterion by considering the spectral gap:

$$gap(\theta, T_j) = \min |\theta - \theta_i| \quad \text{Eq. 7-173}$$

where $\theta_i \neq \theta$. With this, the recommended criterion is

$$\min (\|s^H\|_2 \bullet \|r\|_2) \frac{\|s^H\|_2 \bullet \|r\|_2}{gap(\theta, T_j)} \leq \varepsilon_{con} \quad \text{Eq. 7-174}$$

In the above, we assumed that the Lanczos blocks have a uniform size of p. It is possible to generalize this to allow for the j-th iteration to have p_j variable block size. Such flexibility may be advantageous in the case of clustered eigenvalues or to avoid the breakdown of the Lanczos process.

Let us assume at the $(j+1)$-st iteration, the block size is increased by some k and the $(j+1)$-st Lanczos vectors are augmented as:

$$[P_{j+1} \; \underline{P}_{j+1}] \quad \text{Eq. 7-175}$$

and

$$[Q_{j+1} \; \underline{Q}_{j+1}] \quad \text{Eq. 7-176}$$

where the $\underline{*}$ vectors are the still undefined augmentations. It is easy to see that appropriate augmentations will maintain the validity of the three member recurrence of **Eq. 7-164** and **Eq. 7-165** as follows:

$$[B_{j+1} \; 0] \begin{bmatrix} P_{j+1}^H \\ \underline{P}_{j+1}^H \end{bmatrix} = P_j^H A - A_j \; P_j^H - C_j \; P_{j-1}^H \quad \text{Eq. 7-177}$$

and

$$[Q_{j+1} \; \underline{Q}_{j+1}] \begin{bmatrix} C_{j+1} \\ 0 \end{bmatrix} = AQ_j - Q_j \; A_j - Q_{j-1} \; B_j \quad \text{Eq. 7-178}$$

The Lanczos process can therefore formally continue with the following substitutions:

$$B_{j+1} \leftarrow [B_{j+1} 0] \quad \text{Eq. 7-179}$$

$$C_{j+1} \leftarrow \begin{bmatrix} C_{j+1} \\ 0 \end{bmatrix} \quad \text{Eq. 7-180}$$

and

$$P_{j+1} \leftarrow [P_{j+1}\ \underline{P}_{j+1}] \quad \text{Eq. 7-181}$$

$$Q_{j+1} \leftarrow [Q_{j+1}\ \underline{Q}_{j+1}] \quad \text{Eq. 7-182}$$

The conditions of successful continuation with augmented blocks are the orthogonality requirements of:

$$\bar{P}^H{}_{j+1}\ \bar{Q}_j = 0 \quad \text{Eq. 7-183}$$

and

$$\bar{P}_j^H\ \underline{Q}_{j+1} = 0 \quad \text{Eq. 7-184}$$

It is of course necessary that the inner product of the newly created, augmented pair of Lanczos vector blocks

$$\underline{P}_j^H\ \underline{Q}_{j+1}$$

is not singular, since its decomposition will be needed by the algorithm. Specifically, we need the smallest singular values of the inner product matrix to be larger than a certain small number. A possible choice for the augmentations is to choose k pairs of random vectors and orthogonalize them against the earlier vectors by using a modified Gram-Schmidt procedure. The orthogonalization may be repeated several times to assure that the smallest value is above the threshold.

The most prevalent usage of the block size adaptation is to cover existing clusters of eigenvalues. In the real block implementation, we were not able to change the block size on the fly (adaptively). Therefore, an estimate of the largest possible cluster was needed a priori. For typical structural applications, the default block size of 7 (anticipating 6 as the highest multiplicity) was used.

The multiplicity of a cluster may be found on the fly with the help of the Ritz values. The number of expected multiplicities in a cluster is the number of elements of the set satisfying:

$$\left|\theta_i - \theta_k\right| \le \varepsilon_{clu} \max\left(\left|\theta_i\right|, \left|\theta_k\right|\right) \quad \text{Eq. 7-185}$$

where ε_{clu} is the user specified cluster threshold. The order of the largest cluster of the Ritz values is calculated every time a convergence test is made and the block size is appropriately adjusted. This procedure is not very expensive, since it is done on the tridiagonal problem.

Preventing Breakdown. It is easy to see that the Lanczos process breaks down in some circumstances. These are:

- Either R_j or S_j or both are rank deficient.
- Neither are rank deficient, but $R_j^H S_j$ is rank deficient.

The breakdown of the first kind prevents the execution of the QR decomposition of the *j*-th blocks. This is fairly easy to overcome by an orthogonalization procedure also applied in the current complex Lanczos implementation.

Specifically, if S_j is rank deficient, then we restart the Lanczos process with a random Q_{j+1} made orthogonal to all previous left Lanczos vectors $\bar{P}_j$ as:

$$\bar{P}_j^H Q_{j+1} = 0 \qquad \text{Eq. 7-186}$$

If S_j is just nearly rank deficient (detected by the QR decomposition of $S_j = Q_{j+1} C_{j+1}$), then we reorthogonalize this Q_{j+1} to the previous left Lanczos vectors, as shown in the above equation.

Rank deficiency (full or near) of R_j is treated similarly with respect to the right Lanczos vectors.

The breakdown of the second type (a serious breakdown) manifests itself in the singular value decomposition of $P_{j+1}^H Q_{j+1}$. In this type of breakdown, some or all of the singular values are zero, as follows:

$$P_{j+1}^H Q_{j+1} = U_j \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V_j^H \qquad \text{Eq. 7-187}$$

where Σ is nonsingular if it exists. This problem may also be overcome using the augmentation techniques shown earlier. First, calculate and partition as follows:

$$P_{j+1} U_j = \begin{bmatrix} P_{(1)} & P_{(2)} \end{bmatrix} \qquad \text{Eq. 7-188}$$

and

$$Q_{j+1} V_j = \begin{bmatrix} Q_{(1)} & Q_{(2)} \end{bmatrix} \qquad \text{Eq. 7-189}$$

where the number of columns in the second partition is equal to the number of zero singular values. Create the following projector matrix:

$$\Pi_j = \bar{Q}_j \bar{P}_j^H \quad \text{Eq. 7-190}$$

Bai et al., 1996 proves that the choice of vectors:

$$\underline{P}_{(2)} = (I - \Pi_j)^H Q_{(2)} \quad \text{Eq. 7-191}$$

and

$$\underline{Q}_{(2)} = (I - \Pi_j)^H P_{(2)} \quad \text{Eq. 7-192}$$

in the following augmentation:

$$P_{j+1} = \left[P_{(1)} \quad P_{(2)} \quad \underline{P}_{(2)} \right] \quad \text{Eq. 7-193}$$

and

$$Q_{j+1} = \left[Q_{(1)} \quad Q_{(2)} \quad \underline{Q}_{(2)} \right] \quad \text{Eq. 7-194}$$

will always result in a nonsingular $P_{j+1}^H Q_{j+1}$ product.

It is possible to extend this procedure to the case of near-breakdown when the singular values may not be exact zeroes, but smaller than desired. In this case, it is recommended to increase the block size for those singular values that are below a specified threshold. Finally, one may only use random vectors instead of the projection matrix.

Maintaining Biorthonormality. The maintenance of the biorthonormality of the P_j and Q_j vectors is the cornerstone of the Lanczos algorithm. Local biorthonormality, i.e. maintaining the condition between the consecutive Lanczos vectors, is fairly simple by executing the following steps:

$$R_j \leftarrow R_j - P_j(Q_j^H R_j) \quad \text{Eq. 7-195}$$

$$S_j \leftarrow S_j - Q_j(S_j^H R_j) \quad \text{Eq. 7-196}$$

These steps use data only available in memory, therefore they are cost effective even when executed repeatedly. Unfortunately, this method does not ensure that converged eigenvectors will not reappear. This may be prevented by a full reorthonormalization scheme using a modified Gram-Schmidt process. This is

implemented in the single vector complex Lanczos method of NX Nastran. A measure of the orthogonality of the current Lanczos vectors, with respect to the already accepted eigenvector, is:

$$d_{j+1} = \max\left(\frac{\left\|\bar{P}_j^H Q_{j+1}\right\|_1}{\left\|\bar{P}_j\right\|_1 \left\|Q_{j+1}\right\|_1}, \frac{\left\|\bar{Q}_j^H P_{j+1}\right\|_1}{\left\|\bar{Q}_j\right\|_1 \left\|P_{j+1}\right\|_1}\right) \quad \text{Eq. 7-197}$$

where $\|.\|_1$ is the matrix column norm. This quantity is usually compared to a machine computational accuracy indicator as:

$$d_{j+1} \leq \sqrt{\varepsilon_{mac}} \quad \text{Eq. 7-198}$$

where ε_{mac} is the automatically calculated machine epsilon. The appropriateness of the choice of the square root was proven in the real Lanczos method (called partial orthogonality there) and was already successfully employed in the READ module.

That measure, however, is very expensive, both in CPU and I/O regards. Specifically, the numerator requires the retrieval of the $\bar{P}_j$ and $\bar{Q}_j$ vector blocks from secondary storage and a multiplication by them. A method of maintaining partial orthogonality with a limited access of the $\bar{P}_j$ and $\bar{Q}_j$ matrices, uses

$$d_{j+1} = \max\left(\frac{\left\|X_{j+1}\right\|_{inf}}{\left\|\bar{P}_j\right\|_1 \left\|Q_{j+1}\right\|_1}, \frac{\left\|Y_{j+1}\right\|_{inf}}{\left\|\bar{Q}_j\right\|_1 \left\|P_{j+1}\right\|_1}\right) \quad \text{Eq. 7-199}$$

where $\|.\|_{inf}$ is now the matrix row norm. The norms of the denominator may be updated in every iteration step without retrieving the eigenvectors. This method calculates the above numerator terms

$$X_{j+1} = \bar{P}_j^H Q_{j+1} \quad \text{Eq. 7-200}$$

and

$$Y_{j+1} = \bar{P}_{j+1}^H Q_j \quad \text{Eq. 7-201}$$

utilizing the fact that these satisfy the three term recurrence equations perturbed by rounding errors as follows:

$$X_{j+1} = T_j \begin{bmatrix} X_j \\ 0 \end{bmatrix} - \begin{bmatrix} X_j \\ 0 \end{bmatrix} A_j - \begin{bmatrix} X_{j-1} \\ W_1^l \end{bmatrix} B_j + \begin{bmatrix} 0 \\ B_{j+1} W_2^l \end{bmatrix} \quad \text{Eq. 7-202}$$

where

$$W_1^l = P_j^H Q_{j+1} \quad \text{Eq. 7-203}$$

and

$$W_2^l = P^H{}_{j+1} Q_j \quad \text{Eq. 7-204}$$

The superscript l refers to the left side. Similarly, for the right side:

$$Y_{j+1} = \begin{bmatrix} X_j & 0 \end{bmatrix} T_j - A_j \begin{bmatrix} Y_j & 0 \end{bmatrix} - C_j \begin{bmatrix} Y_{j-1} & W_1^r \end{bmatrix} + \begin{bmatrix} 0 & \\ W_2^r & C_{j+1} \end{bmatrix} \quad \text{Eq. 7-205}$$

where

$$W_1^r = P_j^H Q_{j+1} \quad \text{Eq. 7-206}$$

and

$$W_2^r = P_{j+1}^H Q_j \quad \text{Eq. 7-207}$$

with superscript r referring to the right side. After these steps, the matrices are perturbed to simulate the round-off effect as:

$$X_{j+1} = (X_{j+1} + F_j) C_{j+1}^{-1} \quad \text{Eq. 7-208}$$

and

$$Y_{j+1} = B_{j+1}^{-1} (Y_{j+1} + F_j^H) \quad \text{Eq. 7-209}$$

where F_j is a random matrix having a norm of ε_{mac}. If the test

$$\bar{d}_{j+1} \leq \sqrt{\varepsilon_{mac}} \quad \text{Eq. 7-210}$$

fails, then a retroactive modified Gram-Schmidt procedure (similar to the local one above) is needed. The cost of this procedure is $O(n)$, the dominant cost being the inner products of the Lanczos blocks producing the W vector blocks. This assumes that the block structure of T_j is taken into consideration.

Mathematical Algorithm. A simplified summary of the mathematical algorithm follows:

1. Initialization
   a. Choose starting vector blocks P_1, Q_1 such that $P_1^H Q_1 = I$
   b. Calculate $R_1 = (P_1^H A)^H$ and $S_1 = AQ_1$
2. Iteration for $j = 1, 2, \ldots,$
   a. Compute:
      $A_j = P_j^H S_j$
      $R_j \leftarrow R_j - P_j A_j^H$
      $S_j \leftarrow S_j - Q_j A_j$
   b. QR decomposition:
      $R_{j+1} = P_{j+1} B_{j+1}^H$
      $S_j = Q_{j+1} C_{j+1}$
   c. SV decomposition:
      $P_{j+1}^H Q_{j+1} = U_j \Sigma_j V_j^H$
   d. Compute recurrence:
      $B_{j+1} = B_{j+1} U_j \Sigma_j^{1/2}$
      $C_{j+1} = \Sigma_j^{1/2} V_j^H C_{j+1}$
      $P_{j+1} = P_{j+1} U_j \Sigma_j^{-1/2}$
      $Q_{j+1} = Q_{j+1} V_j \Sigma_j^{-1/2}$
   e. Start recurrence:
      $R_{j+1} = (P_{j+1}^H A - C_{j+1} P_j^H)^H$
      $S_{j+1} = AQ_{j+1} - Q_j B_{j+1}$

Figure 7-3 Block Lanczos Logic

Singular Value Decomposition (SVD)

Since a crucial element of the algorithm in Figure 7-3 is the singular value decomposition, step (c) in Figure 7-3, this section gives more detail.

Given any $n \times m$ matrix A, $(n \geq m)$ there exist unitary matrices U and V such that

$$A = U\Sigma V^H$$

where Σ has the same dimensions as A, but is in the form

$$\Sigma = \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix}$$

where $\hat{\Sigma}$ is a diagonal matrix with real diagonal entries $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$. The diagonal entries of $\hat{\Sigma}$ are the singular values of A, and are mathematically defined to be the positive square roots of the eigenvalues of $A^H A$.

The singular value decomposition of A is computed without forming $A^H A$. The first step is to compute unitary matrices U_1 and V_1 such that

$$B = U_1^H A V_1$$

is bidiagonal.

The main step of the SVD algorithm implicitly performs the Francis QR iteration on the matrix $B^H B$.

It is also possible to define an "economy" size SVD. Let $\hat{U}$ be the matrix consisting of the first m columns of U. Then

$$A = \hat{U} \hat{\Sigma} V^H$$

is an alternate form of the SVD.

The method is also available directly to the user from the CEAD module, as shown in "User Interface" on page 237.

The Iterative Schur-Rayleigh-Ritz Method (ISRR)

The Iterative Schur-Rayleigh-Ritz Method (ISRR) is a procedure which extracts a specified number of roots which lie within a circle in the complex plane centered at the origin. The ISSR method computes a Schur factorization of the canonical matrix A such that:

$$AQ = QT$$

where A has dimension n x n, Q is n x m, and T is m x m. The reduced eigenproblem:

$$Ty = \lambda y$$

is then solved. Transformation of y with Q recovers the eigenvectors corresponding to the eigenvalues λ.

The advantages of this approach are that a much smaller m x m problem is solved spanning the subspace of the first m eigenvalues of A, and a Schur decomposition is constructed which, within the limits of a numerical method, provides a greater degree of confidence than the complex Lanczos method that all modes have been found.

7.3 Solution Method Characteristics

The available methods in NX Nastran are the Hessenberg methods, the complex Lanczos methods, SVD, and ISRR. The Hessenberg method is a reduction method, as is the SVD, while the Lanczos method and ISRR are iterative methods. The characteristics of these methods are:

Method	Type	Identifier	Application	Restriction
Hessenberg	Reduction	HESS	All roots, few vectors	M nonsingular
QZ Hessenberg	Reduction	QZHESS	All roots, few vectors	None
Complex Lanczos	Iterative	CLAN	Few roots	$[K] + \lambda_s[B] + \lambda_s^2[M] \neq 0$
SVD	Reduction	SVD	Singular value and/or vectors of K	B, M must be purged
ISRR	Iterative	ISRR	Roots closest to origin	M nonsingular

7.4 User Interface

```
CEAD      KXX,BXX,MXX,DYNAMIC,CASECC,VDXC,VDXR/
          CPHX,CLAMA,OCEIG,LCPHX,CLAMMAT/
          S,N,NEIGV/UNUSED2/SID/METH/EPS/ND1/ALPHAJ/OMEGAJ/
          MAXBLK/IBLK/KSTEP/NDJ $
```

Input Data Blocks:

KXX	Stiffness matrix.
BXX	Viscous damping matrix.
MXX	Mass matrix.
DYNAMIC	Table of Bulk Data entry images related to dynamics.
CASECC	Table of Case Control command images.
VDXC	Partitioning vector with 1.0 at rows corresponding to null columns in K, B, and M.
VDXR	Partitioning vector with 1.0 at rows corresponding to null rows in K, B, and M.

Output Data Blocks:

CPHX	Complex eigenvector matrix, or right singular vectors V (SVD method with ND1 > 0).
CLAMA	Complex eigenvalue summary table.
OCEIG	Complex eigenvalue extraction report.
LCPHX	Left-handed complex eigenvector matrix (Lanczos only), or left singular vectors U (SVD method with ND1 > 0).
CLAMMAT	Diagonal matrix with complex eigenvalues on the diagonal, or diagonal matrix Σ of singular values (SVD method). See Remark 8.

Parameters:

NEIGV	Output-integer-no default. NEIGV indicates the number of eigenvalues found. If none were found, NEIGV is set to -1.
UNUSED2	Input-integer-default=1. Unused.

SID — Input-integer-default=0. Alternate set identification number.

If SID=0, the set identification number is obtained from the CMETHOD command in CASECC and used to select the EIGC entry in DYNAMIC.

If SID>0, then the CMETHOD command is ignored and the EIGC entry is selected by this parameter value. Applicable for all methods.

If SID<0, then both the CMETHOD command and all EIGC entries are ignored and the subsequent parameter values (E, ND1, etc.) will be used to control the eigenvalue extraction. Applicable for single vector Lanczos, block Lanczos, QZ Hessenberg, QR Hessenberg, and SVD (Singular Value Decomposition).

METH — Input-character-default='CLAN'. If SID<0, then METH specifies the method of eigenvalue extraction:

- CLAN — Complex Lanczos (block or single vector),
- HESS — QZ Hessenberg or QR Hessenberg,
- SVD — Singular Value Decomposition,
- ISRR — Iterative Schur-Rayleigh-Ritz Method.

EPS — Input-real-default=1.E-5. Used only when SID<0.

ND1 — Input-integer-default=0. The number of desired eigenvectors. Used only when SID<0.

ALPHAJ — Input-real-default=0.0. Real part of shift point. Used only when SID<0.

OMEGAJ — Input-real-default=0.0. Imaginary part of shift point. Used only when SID<0.

MAXBLK — Input-integer-default=7. Maximum block size. Used only when SID<0.

IBLK — Input-integer-default= see Remark **10.** Initial block size. Used only when SID<0.

KSTEP — Input-integer-default=ND1 / (10*IBLK) + 2. Frequency of solve. Used only when SID<0.

NDJ — Input-integer-default=0. The number of desired eigenvectors at desired shift point for pre-Version 70.5 Lanczos method. Used only when SID<0.

7.5 Method Selection

EIGC Complex Eigenvalue Extraction Data

Defines the data needed to perform complex eigenvalue analysis.

Format:

1	2	3	4	5	6	7	8	9	10
EIGC	SID	METHOD	NORM	G	C	E	ND0		

The following continuation is repeated for each desired search region. (J = 1 to n, where n is the number of search regions.)

	ALPHAAJ	OMEGAAJ	ALPHABJ	OMEGABJ	LJ	NEJ	NDJ		

Alternate Format for Continuation Entry for Block Complex Lanczos:

	ALPHAAJ	OMEGAAJ	MBLKSZ	IBLKSZ	KSTEPS		NJi		

Alternate Format for Continuation Entry for ISRR:

	ALPHACJ	OMEGACJ				ISRRFLG	NJi		

Examples:

EIGC	14	CLAN							
		+5.6					4		
		-5.5					3		

EIGC	15	ISRR							
	-1.0	0.0				3	4		

EIGC	16	HESS					6		

Field	Contents
SID	Set identification number. (Unique Integer > 0)
METHOD	Method of complex eigenvalue extraction. (Character: “INV,” “HESS,” “CLAN” or “ISRR”)

Field	Contents
NORM	Method for normalizing eigenvectors. (Character: "MAX" or "POINT"; Default = "MAX"). See "Normalization Options" on page 245.
G	Grid or scalar point identification number. Required if and only if NORM = "POINT". (Integer > 0).
C	Component number. Required if and only if NORM="POINT" and G is a geometric grid point. ($0 \leq$ Integer ≤ 6)
E	Convergence criterion. (Real ≥ 0.0. Default values are: 10^{-4} for METHOD = "INV", 10^{-15} for METHOD = "HESS", E is machine dependent for METHOD = "CLAN".)
MBLKSZ	Maximum block size. (Default = 7, Integer ≥ 0) Block Lanczos only.
IBLKSZ	Initial block size. (Default = See Remark 10., Integer ≥ 0) Block Lanczos only.
KSTEPS	Frequency of solve. (Default = 5, Integer > 0) Block Lanczos only.
ISRRFLG	Used only for ISRR (see "ISRR Option" on page 248).
ALPHACJ	Used only for ISRR (see "ISRR Option" on page 248).
OMEGACJ	Used only for ISRR (see "ISRR Option" on page 248).

Field	METHOD Field			
	HESS	INV	CLAN	ISRR
NDj (Integer > 0)	Desired number of eigenvectors. (No default)	Desired number of roots and eigenvectors in j-th search region. (Default = 3* NEj)	Desired number of roots and eigenvectors to be extracted at j-th shift point. (No default)	Desired number of eigenvectors. (No default)
ALPHAAj OMEGAAj Real and imaginary parts of Aj in radians/ time (Real).	Not used	End point Aj of j-th search region in complex plane. (Default = 0.0)	j-th shift point. (Default = 0.0)	Fields have alternate meaning (See "ISRR Option" on page 248.)

Field	METHOD Field			
	HESS	INV	CLAN	ISRR
ALPHABj OMEGABj Real and imaginary parts of Bj in radians/ time (Real).	Not used	End point Bj of j-th search region in complex plane. (Default = 0.0)	See alternate definition below.	Not used
Lj (Real > 0.0)	Not used	Width of j-th search region. (Default = 1.0)	See alternate definition below.	Not used
NEj (Integer > 0)	Not used	Estimated number of roots in j-th search region. (Default = 0)	Not used	Not used
MBLKSZ For block CLAN only	Not used	Not used	Maximum Block Size Default = 7	Not used
IBLKSZ For block CLAN only	Not used	Not used	Initial Block Size Default = 2	Not used

Remarks:

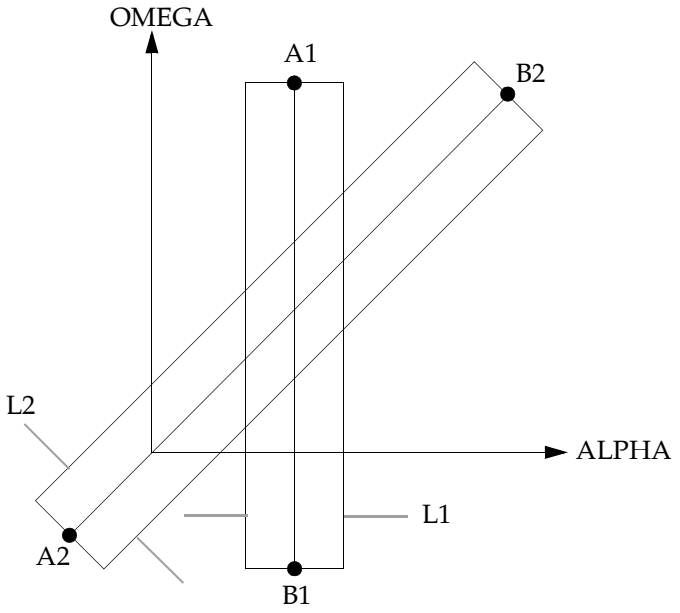


Figure 7-4 Sample Search Regions

1. The EIGC entry must be selected in the Case Control Section with the command CMETHOD = SID. Methods of solution are also controlled by SYSTEM(108); see "**Hessenberg and Lanczos Options**" on page 245.
2. The "HESS" method is generally more reliable and economical for small and moderate-size problems. It computes all eigenvalues and ND eigenvectors.
3. The "ISRR" method works well on sparse matrices, confines the search region to a circle centered on the origin of the complex plane, and provides some reliability that all modes within the circle have been found.
4. The EIGC entry may or may not require continuations as noted below.
   - For the "HESS" method, continuations are not required; and their contents are ignored when present, except for ND1. However, it is recommended that continuations are not used.
   - For the "CLAN" method, when the continuation entry is not used a shift is calculated automatically. When a shift is input on the first continuation entry it is used as the initial shift. Only one shift is used. Data on other continuation entries is ignored.
   - For METHOD = "INV", each continuation defines a rectangular search region. Any number of regions may be used and they may overlap. Roots in overlapping regions will not be extracted more than once.
   - For METHOD = "ISRR" continuation, see "**ISRR Option**" on page 248.
   - For all methods, if no continuation is present, then ND0 must be specified on the first entry. If a continuation is present, then NDj must be specified on the continuation and not on the first entry.
5. The units of ALPHAAJ, OMEGAAJ, ALPHABJ, and OMEGABJ are radians per unit time.
6. See *The NASTRAN Theoretical Manual*, Sections 10.4.4.5 and 10.4.4.6, for a discussion of convergence criteria and the search procedure with the INV method.
7. DIAG 12 prints diagnostics for the inverse power method, the complex Lanczos method, the QZ HESS method and the ISRR method.
8. If METHOD = "HESS" and the LR or QR methods (non-default methods) are selected by system cell 108 the mass matrix must be nonsingular.
9. When using METHOD = CLAN, the following should be noted. The modern CLAN method (default for METHOD entry of CLAN) has been enhanced to include a block complex Lanczos approach. This method is more reliable and will not accept inaccurate roots which the old method had a tendency to do. Thus, given the same input, the new method may often accept fewer roots. For continuity the old method has been maintained and may be selected by setting SYSTEM(108).
10. The initial block size (IBLKSZ) default is as follows:
    If N < 1000, IBLKSZ = 1.
    If N < 50,000, IBLKSZ = 2.
    If N < 100,000, IBLKSZ = 4.
    If N >100,000, IBLKSZ = 5.

Alternate EIGC Bulk Data Entry

The following alternate format is valid for all methods except for the inverse power method:

1	2	3	4	5	6	7	8	9	10
EIGC	SID	METHOD	NORM	G	C	E	ND0		
	KEYWORD1=<value> KEYWORD2=<value> KEYWORD3=<value>								

where KEYWORD may be any of the parameters from the original entry except SID, as well as:

NDj	Number of desired roots at shift j. (Integer > 0).
SHIFTRj	The real part of shift j. (Real)
SHIFTIj	The imaginary part of shift j. (Real)
KSTEPSj	Block tridiagonal solution frequency at shift j; (only block Lanczos).(Integer > 0)
MBLKSZj	Maximum block size at shift j (only block Lanczos). (Integer > 0)
IBLKSZj	Initial block size at shift j (only block Lanczos). (Integer > 0)

Note: In the parameters above, the value of j ranges from 1 to 10.

Examples:

EIGC	1	CLAN							
	eps=1.E-12, nd1=12, shiftr1=0, shifti1=2.4E2								

EIGC	2	HESS							
	ND1=10								

EIGC	3	CLAN							
	shiftr1=0.0, shifti1=20., nd1=5, iblksz1=2, mblksz1=5								
	shiftr2=0.0, shifti2=50., nd2=5, iblksz2=2, mblksz2=5								
	shiftr3=0.0, shifti3=100., nd3=5, iblksz3=1, mblksz3=5								

Remarks about alternate entry options:

1. The first of the keyword-driven continuation entry must be blank.

2. If any of the parameters METHOD, NORM, G, C, EPS, or ND1 are specified on the continuation entry, the corresponding field on the original entry must be blank.
3. A maximum of 10 shifts may be specified.

7.6 Option Selection

Complex eigenvalue analysis in NX Nastran supports general damping and normalization options as well as specific Hessenberg and Lanczos options.

Damping Options

The presence of the $[B]$ matrix indicates the viscous damping option when the following equation is solved:

$$[M\lambda^2 + B\lambda + K]u = 0 \qquad \text{Eq. 7-211}$$

This problem is transformed into a linear problem that is twice the size of the original matrices and provides eigenvalue solutions in complex conjugate pairs.

When the $[B]$ matrix is not present and damping is introduced via imaginary stiffness terms (structural damping), then the following problem is solved:

$$[\lambda^2 M + K]u = 0 \qquad \text{Eq. 7-212}$$

In this case the roots are not complex conjugate pairs. The mixed case of having both viscous and structural damping is also possible.

Normalization Options

The default normalization (and the only method available for the Hessenberg and Lanczos methods) is MAX. This option normalizes the component of the eigenvector with the largest magnitude to one for the real part and zero for the imaginary part.

The POINT normalization option uses G for a grid and C for a component to set the component to a value of (1.0, 0.0). This option is not currently available for the complex eigenvalue methods.

Hessenberg and Lanczos Options

Hessenberg Spill Option. The spill option of the Hessenberg method (with QR) is less robust than the default (with QZ), but the latter has no spill. Since the spill option requires a much smaller amount of memory to run a problem, larger problems can be solved on a given computer.

Single Vector Lanczos Option. The adaptive block option of the Lanczos method is very robust and efficient even for large, direct complex eigenvalue analysis jobs, and is the default. Thus, the single vector option must be specified, to override the default.

These options are selected via NEWHESS = SYSTEM(108).

SYSTEM(108)			
Bit	Decimal	EIGC Entry	Selection
0	0	HESS	QZ Hessenberg without spill (default)
1	1	HESS	QR Hessenberg with spill
2	2	CLAN	Single vector complex Lanczos
3	4	CLAN	Adaptive, block complex Lanczos (default)
4	8	CLAN	Debug output for both complex Lanczos
9	256	HESS	Force LR Hess (aka old Hessenberg without spill)
10	512	HESS	Force QZ Hess

Since the cell is binary, appropriate combinations are also valid. For example, SYSTEM(108) = 12 is a proper setting for the block method with debug output.

Internal Block Lanczos Options. There are several internal detailed options available for the block Lanczos method (also in SYSTEM(108)) as shown in the following table:

Option	Action
16	Turn off block size reduction in block Lanczos
32	Turn off block size augmentation in block Lanczos
64	Enforce full orthogonality in block Lanczos
128	Turn off initial vector preprocessing in block Lanczos
1024	Override defaults for small problems in block Lanczos
2048	Output XORTH matrix in place of ROOTS matrix
4096	Turn off block FBS for complex matrices in block Lanczos
8192	Turn off symmetric decomposition in block Lanczos
16384	Turn off real reduction phase (always use complex arithmetic)
32768	Force spill of Lanczos vectors (testing purposes only)
65536	Old semi-algebraic sorting criterion
131072	Force spill during eigenvector computation (testing purposes only)

Option	Action
262144	Turn off autoshift logic
2097152	Turn off warning message 5411 if mass matrix has negative diagonal entries

Alternative Methods

SVD Option. The singular value decomposition of the K matrix is produced if B and M are purged. If used in SOLs 107 or 110, and mass or damping terms are present, a user fatal exit is taken. The SVD operation decomposes the input stiffness matrix K into the factors U, Σ, and V as described in "**Solution Method Characteristics**" on page 236. The ND1 value is interpreted differently for the SVD than for an eigensolution.

ND1	Output
>0	All vectors of U and V are output.
=0	U and V are returned in a purged state, that is, only the singular value matrix Σ is computed..
<0	Σ is returned as a square matrix whose number of columns is equal to the minimum number of rows or columns of the input matrix. U and V are truncated to be commensurate with Σ. This is a method to reduce the costs of solving very rectangular input matrices by providing a partial solution for the most interesting vectors.

Linear Solution Option. The new BLOCK method and the QZHESS method enable the solution of the $[B\lambda + K]u = 0$ problem also. This option is automatically selected when the M matrix is purged.

ISRR Option. Using the METHOD=ISRR alternate continuation card, field 7 (ISRRFLG) can be used to define the following instructions;

ISRRFLG	Instruction
1	Reserves fields 2 (ALPHACJ) and 3 (OMEGACJ) for a user supplied shift. Shift does not redefine search region, but is only used during decomposition to avoid a singularity. The use of the shift is recommended for better performance.
2	Forces the out-of-core path in the code.
4	Overrides system cell 405.
8	Forces balanced iteration for real unsymmetrical problems only.
16	Forces generation of starting vectors for quadratic problems from the values found for the linear case when damping is ignored. In all other cases, the starting vectors are randomly generated.
M * 32	Sets the maximum size of the subspace to M vectors.

The above ISRRFLG values may be summed to obtain a combination of settings. For example ISRRFLG = 323 would indicate options "1", "2" and a maximum subspace of 10 vectors (1 + 2 + (10*32) = 323).

7.7 Complex Eigenvalue Diagnostics

Hessenberg Diagnostics

The Hessenberg method has no internal diagnostics when the no spill option is used. The spill option has the following diagnostics:

- NEW HESSENBERG ROUTINE
- TIME OF TRANSFORMATION TO HESS FORM: X (REDUCTION TIME)
- STARTING QR ITERATION WITH NINC = X

  The NINC value is the number of columns that are in memory.
- INFINITY NORM = X

  This is the maximum term (in magnitude) in the Hessenberg matrix.
- FINAL EPSILON = ε

  The E convergence criterion (from the EIGC entry) is adjusted for the problem.
- TIME OF FINDING EIGENVALUES=X (QR ITERATION TIME)
- VECTOR GENERATION WITH NINC=X

  The number of columns of the Hessenberg matrix held in core.
- TIME OF VECTOR GENERATION=X
- VECTOR REVERSE TRANSFORMATION WITH NINC=X

  The number of eigenvectors held in core is NINC.
- TIME OF VECTOR TRANSFORMATION = X (MULTIPLICATION BY HOUSEHOLDER VECTORS).

Hessenberg diagnostics contain both numerical and performance information. The performance is low if NINC « N at any phase where N is the problem size.

The adjusted convergence criterion is

$$\varepsilon = \sqrt{N} \bullet \|H\|_{\infty} \bullet E \qquad \text{Eq. 7-213}$$

where $\|H\|_{\infty}$ is the infinity norm of the Hessenberg matrix and E is the user-given (or default) convergence criterion.

Complex Lanczos Internal Diagnostics

Here, the newer block method diagnostics is shown.

Complex Lanczos Diagnostics DIAG 12. The two levels of internal diagnostics of complex Lanczos are requested via DIAG 12 and SYSTEM(108) = 8. The structure of the DIAG 12 diagnostics is as follows:

*** USER INFORMATION MESSAGE 6361 - COMPLEX LANCZOS DIAGNOSTICS

THIS DIAGNOSTICS IS REQUESTED BY DIAG 12.

INITIAL PROBLEM SPECIFICATION

DEGREES OF FREEDOM	=	XXX	ACCURACY REQUIRED	=	XXX
REQUESTED MODES	=	XXX	NUMBER OF SHIFTS	=	XXX
DAMPING MODE FLAG	=	XXX	SIZE OF WORKSPACE	=	XXX
BLOCK SIZE	=	XXX	STEP SIZE	=	XXX

The accuracy required is an echo of E on the EIGC entry or the default if E is not used. Default = 10^{-6}. The number of shifts equals to the number of the continuation entries. The damping mode flag is 0 when damping matrix is present; otherwise, it is 1.

CURRENT SHIFT IS AT X,Y
CURRENT BLOCK SIZE IS X
NUMBER OF MODES REQUIRED AT THIS SHIFT IS XX

The most important parts of the debugging diagnostics (requested by SYSTEM(108) = 8) are as follows:

	MATHEMATICAL EIGENVALUE		ESTIMATED ACCURACY	
EIGENVALUE #	REAL	IMAGINARY	LEFT	RIGHT
. . . .	. . .	. . . .	. . . .	. . . .
. . . .	. . .	. . . .	. . . .	. . . .

This message may appear any number of times. It indicates the end of an internal Lanczos process.

At the end of the Lanczos run, the following table is printed:

MATHEMATICAL EIGENVALUES

$$\bar{\lambda}_i(\text{real})\ \bar{\lambda}_i(\text{imag}) \qquad i = 1, \ldots, NR_j \qquad \text{Eq. 7-214}$$

Mathematical Solution. This table contains the shifted solutions where NR_j is the reduced size at shift j.

Besides the diagnostics detailed above, additional information is printed for debugging purposes, such as the dynamic matrix calculation and the eigenvectors of the tridiagonal form. These outputs are not explained here. This part of the diagnostics may be extensive, therefore the user should not use SYSTEM(108) = 8 on large problems.

The acceptance of these approximate roots is documented in the following table, again requested by DIAG12:

MATHEMATICAL SOLUTION		DIRECT RESIDUALS	
REAL	IMAGINARY	LEFT	RIGHT
$\lambda_i(\text{real})$	$\lambda_i(\text{imag})$	$y_i^T(A - \lambda_i I)$	$(A - \lambda_i I)x_i$

When the error value is less than the E convergence criterion set by the user on the EIGC entry, then the i-th approximate eigenvalue is accepted as a physical eigenvalue. These accepted eigenvalues are printed in the following table:

PHYSICAL EIGENVALUES

$$\lambda_k(\text{real})\ \lambda_k(\text{imag}) \qquad k = 1, \ldots, NF_j \qquad \text{Eq. 7-215}$$

where NF_j is the number of acceptable roots found at shift j.

The final acceptance is documented by the state equation summary table as follows:

PHYSICAL SOLUTION		STATE EQUATION RESIDUALS	
REAL	IMAGINARY	LEFT	RIGHT
$\lambda_i(\text{real})$	$\lambda_i(\text{imag})$	$\psi_i^T(M\lambda_i^2 + B\lambda_i + K)$	$(M\lambda_i^2 + B\lambda_i^2 + K)\Phi_i$

Any rejected roots are printed in a similar table preceded by the following message:

THE FOLLOWING ROOTS HAD AN UNACCEPTABLY LARGE
STATE EQUATION RESIDUAL

The accepted physical eigenvalues are also printed in the regular complex eigenvalue summary output of the CEAD module: CLAMA. Finally, an eigenvalue summary table is always printed as follows:

EIGENVALUE ANALYSIS SUMMARY (COMPLEX LANCZOS METHOD)

NUMBER OF MODES FOUND	X
NUMBER OF SHIFTS USED	X
NUMBER OF DECOMPOSITIONS	X
NUMBER OF VECTORS IN CORE	X

Complex Lanczos Messages and Errors

To help the user monitor the process (especially in the case of multiple shifts), the complex Lanczos method may issue the following messages:

UWM 5451:
NO ROOTS FOUND AT THIS SHIFT.

UIM 5453:
FEWER ROOTS THAN REQUIRED HAVE BEEN FOUND.

UWM 5452:
NO ROOTS ACCEPTED AT THIS SHIFT.

UIM 5445:
NO ROOTS FOUND AT ALL.

UIM 5444:
ALL ROOTS HAVE BEEN FOUND.

The following message may be repeated up to three times:

UIM 5443:
DYNAMIC MATRIX IS SINGULAR AT THE SHIFT OF X, Y.

The program attempts to perturb the shift point (up to three times) to obtain an acceptable decomposition.

SWM 6938,*:
BREAKDOWN IN BLOCK LANCZOS METHOD.

This message is given on the various (*) breakdown conditions of the Lanczos process along with a recommendation.

UFM 5446:
COMPLEX LANCZOS NEEDS X MORE WORDS.

This message could come from various places.

The user must provide more memory.

SIM 6941:
INVARIANT SUBSPACE DETECTED IN BLOCK LANCZOS.

This message indicates the need for augmenting the current block.

SFM 6939.*:
UNABLE TO READ EIGENVECTORS FROM SCRATCH FILE.
USER ACTION: CLEAN UP DEVICE.

These I/O related messages should not occur normally. If they do occur, the user should clean up the disk, verify the database allocations, etc. (* = 0, 1 or 2)

Performance Diagnostics

Block CLAN Performance Analysis

*** USER INFORMATION MESSAGE 5403 (CLASD*)
BREAKDOWN OF CPU USAGE DURING COMPLEX LANCZOS ITERATIONS:

OPERATION	REPETITIONS	TIMES (SEC): AVERAGE	TOTAL
SHIFT AND FACTOR	5	.8	3.9
MATRIX-VECTOR MULTIPLY & FBS	908	.1	93.2
REORTHOGONALIZATION	574	.1	38.3
SOLVE BLOCK TRIDIAGONAL PROBLEM	74	.5	34.5
EIGENVECTORS AND RESIDUALS	5	13.8	69.2

*** SYSTEM INFORMATION MESSAGE 6940 (CLRRDD)
SPILL OCCURRED WHEN CALCULATING LANCZOS VECTORS.
X OUT OF A TOTAL OF Y LANCZOS VECTORS HAVE BEEN STORED OUT OF CORE.
USER ACTION: TO PREVENT SPILL, INCREASE OPEN CORE SIZE BY AT LEAST Z WORDS

*** SYSTEM INFORMATION MESSAGE 6940 (CLRVRD)
SPILL OCCURRED WHEN CALCULATING PHYSICAL EIGENVECTORS.
USER ACTION: TO PREVENT SPILL, INCREASE MAXIMUM BLOCK SIZE BY AT LEAST Z

Orthogonality Analysis

Two additional orthogonality criteria (see theory in "**Theory of Complex Eigenvalue Analysis**" on page 195) that can be used for solution testing purposes are the O_1, O_2 matrices.

The matrices created by these criteria should be diagonal. The off-diagonal terms are supposed to be computational zeroes. The absolute magnitude of the off-diagonal terms is a good indicator of the correctness of the solution.

The user can use the following DMAP to calculate these matrices in connection with the CEAD module:

```
SMPYAD    LCPHX,K,CPHX,,,/PTKP/3////1 $
MPYAD     CPHX,EIGENV,/PL $
MPYAD     LCPHX,EIGENV,/PTL $
SMPYAD    PTL,M,PL,,,PTKP/O2/3//-1//1 $
SMPYAD    LCPHX,B,CPHX,,,/PTBP/3////1 $
SMPYAD    LCPHX,M,PL,,,PTBP/SUM2/3////1 $
SMPYAD    PTL,M,CPHX,,,SUM2/O1/3//+1//1 $
```

The EIGENV matrix is a matrix containing the eigenvalues on the diagonal. This matrix can be created as a conversion of the CLAMA table by

```
LAMX      ,,CLAMA/EIGENV/-2 $
```

The user can filter out the small terms by using:

```
MATMOD     O1,O2,,,,/ORTHO1F,ORTHO2F/2////1.-6 $
```

The filtered matrices will only contain those off-diagonal terms greater than 10^{-6} in magnitude. Terms greater than 10^{-6} point to those eigenpairs that do not satisfy the orthogonality criterion.

The DMAP statement

```
MATPRN     ORTHO1F,ORTHO2F// $
```

can be used to print out the matrices.

7.8 Complex Lanczos Estimates and Requirements

The time estimates for single vector complex Lanczos are detailed below.

Shifting time (sec) is

$$T_d \quad \text{Eq. 7-216}$$

Recursion time (sec) is

$$4 \cdot N_{steps}(N \cdot C \cdot M + T_s) \quad \text{Eq. 7-217}$$

Normalization time (sec) is

$$(2 \cdot \text{IPREC}) N_{des} \cdot N^2 \cdot M \quad \text{Eq. 7-218}$$

Packing time (sec) is

$$(2 \cdot \text{IPREC}) N_{des} \cdot N \cdot P \quad \text{Eq. 7-219}$$

where:

N_{des} = number of modes desired

N_{steps} = number of Lanczos steps

T_d = decomposition time (see "**Decomposition Estimates and Requirements**" on page 71 for details)

T_s = solution time (see "**Decomposition Estimates and Requirements**" on page 71 for details)

C = average front size

The minimum storage requirements are as follows:

Memory: $(2 \cdot \text{IPREC})(6 \cdot MBLKSZ^2 + 8 \cdot N \cdot MBLKSZ + 8 \cdot MBLKSZ \cdot MSZT)$

where:

$MBLKSZ$ = maximum block size

$MSZT$ = maximum size of T_j matrix

N = problem size

The rest of the available memory is used for temporary storage of accepted eigenvectors to reduce the I/O cost of outer orthogonalization.

7.9 References

Bai, Z., et al. **ABLE: An Adaptive Block Lanczos Method for Non-Hermitian Eigenvalue Problems**. Department of Mathematics, University of Kentucky, 1996.

Cullum, J. K.; Willoughby, R. A. ***Lanczos Algorithms for Large Symmetric Eigenvalue Computations.*** Birkhäuser, 1985.

Cullum, J.; Willoughby, R. A. ***Large Scale Eigenvalue Problems.*** North-Holland, 1986.

Golub, G. H.; Van Loan, C. F. ***Matrix Computations***. John Hopkins University Press, 1983.

Householder, A.S.; Bauer, F.L. ***On Certain Methods for Expanding the Characteristic Polynomial.*** Numerische Mathematik, Volume 1, 1959, pp. 29-37.

Komzsik, L. ***Implicit Computational Solution of Generalized Quadratic Eigenvalue Problems.*** Journal of Finite Element Analysis and Design, 2000.

Kowalski, T. ***Extracting a Few Eigenpairs of Symmetric Indefinite Matrix Pairs***. Ph.D. Thesis, University of Kentucky, 2000.

Smith, B. T. et al. ***Matrix Eigensystem Routines - EISPACK Guide.*** Springer Verlag, 1974.

Wilkinson, J. H. ***The Algebraic Eigenvalue Problem***. Oxford University Press, 1965.

Glossary of Terms

AVG	Average.
BUFFER	An area of memory reserved for data transfer between secondary storage and memory.
BUFFPOOL	A pool of buffers used by the executive system.
BUFFSIZE	Buffersize in words (in machine precision).
C	Average front size.
CEAD	Complex eigenvalue analysis module.
cell	An element of the SYSTEM common block of NX Nastran.
CLAN	Complex Lanczos method identifier.
DECOMP	Matrix decomposition functional module.
Dense	A matrix or vector is dense when it has few or no zero elements.
DIAGs	Diagnostic flags of NX Nastran.
DOF(s)	Degree(s)-of-freedom.
EXECUTIVE	The portion of NX Nastran controlling the execution of the program.
FACTOR	Triangular matrix, computed in the DECOMP module.
FBS	Forward-backward substitution functional module.
GINO	General input-output system.
GIV	Givens method identifier.
HESS	Hessenberg method identifier.
HOU	Householder method identifier.
ID	Identification.
IPREC	Machine precision: 1 for short-word machines. 2 for long-word machine.
Kernel	Internal numerical and I/O routines used heavily by functional modules.
Keyword	A word specific to a particular function or operation in NX Nastran.
LHS	Left-hand side.
M	M value, unit numerical kernel time in msec.
MAXRATIO	A diagnostics parameter to show ill-conditioning.
MEM	Memory area reserved for memory files.

MPC	Multipoint constraint.
MPYAD	Matrix multiply and add functional module.
N	Problem size.
NZ	Nonzero words in matrix.
P	P value, unit data packing time using columns.
PARALLEL	Keyword to specify multiple CPU execution.
P_i	Pi value, unit data packing time using terms.
P_s	Ps value, unit data packing time using strings.
RAM	Random access memory area used by the executive system.
READ	Real eigenvalue analysis module.
RHS	Right-hand side.
RMS	Root mean squared.
SEQP	Sequencing module.
SOLVIT	Iterative equation solution module.
Sparse	A matrix or vector is sparse when it has many zero elements.
SPARSE	Keyword to specify indexed kernel usage.
String	A sequence of consecutive nonzero terms in a matrix column.
STRL	Average string length.
STURM Number	Number of negative terms on the factor diagonal.
Trailer	An information record following (trailing) a matrix, which contains the main characteristics.
ρ	Matrix density.

Bibliography

Babikov, P. & Babikova, M. ***An Improved Version of Iterative Solvers for Positive Definite Symmetric Real and Non-Hermitian Symmetric Complex Problems***. ASTE, JA-A81, INTECO 1995.

Bai, Z., et al. **ABLE: An Adaptive Block Lanczos Method for Non-Hermitian Eigenvalue Problems**. Department of Mathematics, University of Kentucky, 1996.

Brown, J. ***Price/Performance Analysis of MSC/NASTRAN***. Proc. of the Sixteenth MSC Eur. Users' Conf., Paper No. 17, September, 1989.

Bunch, J. R.; Parlett, B. N. ***Direct Methods for Solving Symmetric Indefinite Systems of Linear Equations.*** Society for Industrial and Applied Mathematics Journal of Numerical Analysis, Volume 8, 1971.

Caldwell, Steve P.; Wang, B.P. ***An Improved Approximate Method for Computing Eigenvector Derivatives in MSC/NASTRAN***. 1992 MSC World Users' Conf. Proc., Vol. I, Paper No. 22, May, 1992.

Chatelin, F. ***Eigenvalues of Matrices***. Wiley, 1993.

Chan, T.; Wan, W. L. ***Analysis of Projection Methods for Solving Linear Systems with Multiple Right Hand Sides***. CAM Report #26, UCLA, 1994.

Chiang, K. N.; Komzsik, L. ***The Effect of a Lagrange Multiplier Approach in MSC/NASTRAN on Large Scale Parallel Applications***. Comp. Systems in Engineering, Vol 4, #4-6, 1993.

Conca, J.M.G. ***Computational Assessment of Numerical Solutions***. ISNA '92, Praque, 1992

Cullum, J. K.; Willoughby, R. A. ***Lanczos Algorithms for Large Symmetric Eigenvalue Computations.*** Birkhäuser, 1985.

Cullum, J.; Willoughby, R. A. ***Large Scale Eigenvalue Problems.*** North-Holland, 1986.

Duff, I. S.; Reid, J. K. ***The Multifrontal Solution of Indefinite Sparse Symmetric Linear Systems.*** Harwell Report CSS122, England, 1982.

Efrat, I.; Tismenetsky, M. ***Parallel iterative solvers for oil reservoir models***. IBM J. Res. Dev. 30 (2), 1986.

Francis, J. G. F. ***The QR Transformation, A Unitary Analogue to the LR Transformation.*** The Computer Journal, Volume 4, No. 3, Oct. 1961, and No. 4, Jan. 1962.

George, A.; Liu, J. W. ***Computer Solutions of Large Sparse Positive Definite Systems.*** Prentice Hall, 1981.

Givens, W. ***Numerical Computation of the Characteristic Values of a Real Symmetric Matrix***. Oak Ridge National Lab., ORNL-1574, 1954.

Goehlich, D.; Komzsik, L. ***Decomposition of Finite Element Matrices on Parallel Computers***. Proc. of the ASME Int. Computers in Engineering Conf., 1987.

Goehlich, D.; Fulton, R.; Komzsik, L. ***Application of a Parallel Equation Solver to Static FEM Problems***. Int. J. for Computers and Structures, 1989.

Gockel, M. A. ***Handbook for Dynamic Analysis***. The MacNeal-Schwendler Corp., Los Angeles, 1983.

Golub, G. H.; Van Loan, C. F. ***Matrix Computations***. John Hopkins University Press, 1983.

Grimes, R. G.; Lewis, J. G.; Simon, H. D.; Komzsik, L.; Scott, D. S. ***Shifted Block Lanczos Algorithm in MSC/NASTRAN***. MSC/NASTRAN Users' Conf. Proc. Paper No. 12, March, 1985.

Grimes, R. G., et al. ***A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems***. SIAM, J. Mat. Analysis Appl., 13, 1992.

Hageman, L., Young, D. ***Applied Iterative Methods***. Academic Press, 1981.

Hendrickson, B., Rothberg, E. I**mproving the Runtime and Quality of Nested Dissection Ordering**. Silicon Graphics, Inc., Mountain View, CA, April 11, 1996.

Householder, A.S.; Bauer, F.L. ***On Certain Methods for Expanding the Characteristic Polynomial.*** Numerische Mathematik, Volume 1, 1959, pp. 29-37.

Karypis, G., Kumar, V. ***METIS©, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 3.0.3***, University of Minnesota, Department of Computer Sciences/Army HPC Research Center, Minneapolis, MN, November 5, 1997. (http://www.cs.umn.edu/~karypis)

Komzsik, L. ***Implicit Computational Solution of Generalized Quadratic Eigenvalue Problems.*** Journal of Finite Element Analysis and Design, 2000.

Komzsik, L. ***New Features of the Lanczos Module in Version 67 of MSC/NASTRAN***. Proc. of the 18th MSC Eur. Users' Conf., Paper No. 13, June, 1991, Rome.

Komzsik, L. ***Optimization of Finite Element Software Systems on Supercomputers***. Supercomputing in Engineering Analysis, ed. Hojjat Adeli, Marcel-Dekker, 1991.

Komzsik, L. ***Parallel Processing in Finite Element Analysis***. Finite Element News, England, June, 1986.

Komzsik, L. ***Parallel Static Solution in Finite Element Analysis***. The MSC 1987 World Users Conf. Proc., Vol. I, Paper No. 17, March, 1987.

Komzsik, L; Rose, T. ***Substructuring in MSC/NASTRAN for Large Scale Parallel Applications***. Computing Systems in Engineering, Vol 2, #1, 1991.

Kowalski, T. ***Extracting a Few Eigenpairs of Symmetric Indefinite Matrix Pairs***. Ph.D. Thesis, University of Kentucky, 2000.

Lanczos, C. ***An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators.*** Journal of the Research of the National Bureau of Standards., Volume 45, 1950, pp. 255-282.

Lewis, J. G.; Grimes, R. G. ***Practical Lanczos Algorithms for Solving Structural Engineering Eigenvalue Problems.*** Sparse Matrices, and Their Uses, edited by I. E. Duff, Academic Press, London, 1981.

Levy, R.; Wall, S. ***Savings in NASTRAN Decomposition Time by Sequencing to Reduce Active Columns***. NASTRAN: Users' Exper., pp. 627-632, September, 1971, (NASA TM X-2378).

MacNeal, R. H.; Komzsik, L. ***Speeding Up the Lanczos Process.*** RILEM, Kosice, Slovakia, 1995.

MacNeal, R. H. ***The NASTRAN Theoretical Manual***. The MacNeal-Schwendler Corp., Los Angeles, 1972.

Manteuffel, T. A. ***An Incomplete Factorization Technique for Positive Definite Linear Systems***, Math. of Computation, Vol 34, #150, 1980.

McCormick, C.W. ***Review of NASTRAN Development Relative to Efficiency of Execution***. NASTRAN: Users' Experience, pp. 7-28. September, 1973. (NASA TM X-2893)

Mera, A. ***MSC/NASTRAN's Numerical Efficiency for Large Problems on CYBER Versus Cray Computer***. Proc. of the MSC/NASTRAN Eur. Users' Conf., June, 1983.

Newmark, N. M. ***A Method of Computation for Structural Dynamics.*** Proceedings of the Americal Society of Civil Engineers, 1959.

Ortega, J. M.; Kaiser, H. F. ***The LR and QR Methods for Symmetric Tridiagonal Matrices.*** The Computer Journal, Volume 6, No. 1, Jan. 1963, pp. 99-101.

Parlett, B. N. ***The Symmetric Eigenvalue Problem***. Prentice Hall, Englewood Cliffs, 1980.

Petesch, D.; Deuermeyer, D.; Clifford, G. ***Effects of Large Memory on MSC/NASTRAN Performance***. Proc. of the 18th MSC Eur. Users' Conf., Paper No. 12, June, 1991.

Pissanetzsky, S. ***Sparse Matrix Technology.*** Academic Press, 1984.

Poschmann, P.; Komzsik, L. ***Iterative Solution Technique for Finite Element Applications***. Journal of Finite Element Analysis and Design, 19, 1993.

Poschmann, P.; Komzsik, L., Sharapov, I. ***Preconditioning Techniques for Indefinite Linear Systems***. Journal of Finite Element Analysis and Design, 21, 1997.

Rothberg, E. **Ordering Sparse Matrices Using Approximate Minimum Local Fill**, Silicon Graphics, Inc., Mountain View, CA, April 11, 1996.

Saad, Y. ***Numerical Methods for Large Eigenvalue Problems***. Halsted Press, 1992.

Shamsian, S.; Komzsik, L. ***Sparse Matrix Methods in MSC/NASTRAN***. The MSC 1990 World Users Conf. Proc., Vol. II, Paper No. 64, March, 1990.

Sinkiewicz, J. E. ***Numerical Stability of Fine Mesh Torus Models***. Proc. of the MSC/NASTRAN Users' Conf., March, 1979.

Smith, B. T. et al. ***Matrix Eigensystem Routines - EISPACK Guide.*** Springer Verlag, 1974.

Wilkinson, J. H. ***The Algebraic Eigenvalue Problem***. Oxford University Press, 1965.

Wilkinson, J. H. ***The Calculation of the Eigenvectors of Codiagonal Matrices.*** The Computer Journal, Volume 1, 1958, p. 90.

I N D E X

T

U

V