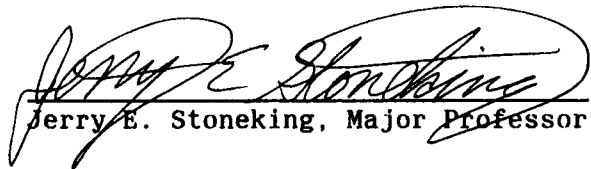
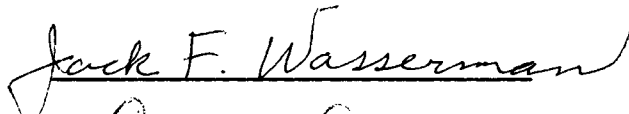
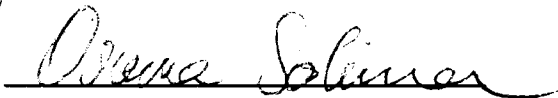


To the Graduate Council:

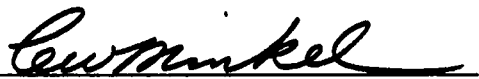
I am submitting herewith a thesis written by Christopher D. Pionke entitled "Comparison and Implementation of Two Different Out-Of-Core Solvers Both With and Without Efficiency Improvement Techniques into the CAEFRAME Finite Element Code." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a Major in Engineering Science.


Jerry E. Stoneking, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:


Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master of Science degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature



Date

July 25, 1988

COMPARISON AND IMPLEMENTATION OF TWO DIFFERENT OUT-OF-CORE
SOLVERS BOTH WITH AND WITHOUT EFFICIENCY IMPROVEMENT
TECHNIQUES INTO THE CAEFRAME FINITE ELEMENT CODE

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Christopher D. Plonke

August 1988

ACKNOWLEDGMENTS

I would like to express my appreciation to Dr. J. E. Stoneking for his help and guidance in defining the scope of this project, as well as his assistance in its development. In addition to Dr. Stoneking, I am also grateful to Dr. J. F. Wasserman and Dr. O. Soliman for their encouragement and belief that I would actually complete this project.

I appreciate the efforts of Dr. T. K. Parnell, formerly of SST Inc., in establishing the required agreements to pursue this project. His comments and reviews of the FORTRAN programs, as well as providing the timing algorithm to test all the solvers are also appreciated.

Finally, I would like to express my love and thankfulness to my wife Cynthia M. Pionke for her support, comments and encouragement throughout this entire project. She had to endure countless hours of my attention being fixed on the computer terminal instead of on her, as well as endure listening to my oral presentation six times. Her comments on grammar and structure of the paper, as well logic and flow were extremely helpful. Also, the magnificent job of proofreading this entire document (at least twice to catch all the tipographical and spelling errors) can not go unmentioned.

ABSTRACT

Currently, the solution algorithm used to solve the generated set of equilibrium equations in the CAEFRAME finite element code operates only in-core. Even though this in-core solution algorithm is very efficient, it is desirable to replace it with an out-of-core solution algorithm which uses external disk space so that larger problems can be solved.

This thesis describes the necessary modifications which were made to two out-of-core linear equation solvers that were originally written for other computer systems in order to integrate them into the CAEFRAME code. Efficiency improvement techniques specifically tailored to each of the two out-of-core solvers were also incorporated. These modifications resulted in four different out-of-core solver configurations.

Several structural models were analyzed using these solvers. Comparisons were made to ascertain which out-of-core solver configuration should replace the current CAEFRAME in-core solver.

It was determined that the blocked out-of-core solver using the modified Cholesky algorithm in conjunction with a bandwidth reducer offered the best trade-offs of solution runtime, disk storage use and maximum problem size versatility.

However, in order to retain the maximum efficiency in solving both small and large problems, it is recommended that both the current in-core solver and the blocked out-of-core solver each with bandwidth reduction capabilities be incorporated in the CAEFRAME code.

TABLE OF CONTENTS

SECTION	PAGE
I. INTRODUCTION	1
Purpose and Objective	1
The Out-Of-Core Solution Algorithms	2
Efficiency Improvement Techniques	7
Procedure for the Implementation of Solvers	9
Restrictions on Implementation	10
II. THE BLOCKED OUT-OF-CORE SOLVER	11
Development of the Modified Cholesky Algorithm	11
In-Core Computer Implementation	34
Out-of-Core Computer Implementation	45
Resolution Capabilities	59
Bandwidth Reduction	59
Modifications and Improvements to the Algorithms	69
III. THE FRONTAL SOLVER	72
Overview of the Frontal Algorithm	73
Development of the Frontal Algorithm	74
Computer Implementation	84
Resolution Capabilities	93
Frontwidth Reduction	95
Modifications and Improvements to the Algorithms	108
IV. RESULTS	111
Comparison Criteria and Procedure	111
Test Models Used to Establish a Quantifiable Comparison	113

SECTION	PAGE
Comparison of the Two In-Core Solver Configurations	117
Comparison of the Two Blocked Out-Of-Core Solver Configurations	122
Comparison of the Two Frontal Solver Configurations	128
Comparison of the Two Remaining Configurations of Out-Of-Core Solvers	133
V. CONCLUSIONS AND RECOMMENDATIONS	138
Effectiveness of the Bandwidth Reducer	138
Effectiveness of the Frontwidth Reducer	139
Chosen Out-Of-Core Solver	140
Final Recommendation	142
LIST OF REFERENCES	144
APPENDICES	147
APPENDIX A. SAMPLE PROBLEM SOLUTION STATISTICS.	148
APPENDIX B. SAMPLE PROBLEM DATA	213
APPENDIX C. PROGRAM LISTINGS FOR THE BLOCKED OUT-OF-CORE SOLVER	286
APPENDIX D. PROGRAM LISTINGS FOR THE FRONTAL SOLVER	343
VITA	373

LIST OF TABLES

TABLE	PAGE
2.1. Element Connectivity Array for Figure 2.9	66
2.2. Nodal Connectivity Array for Figure 2.9	66
3.1. Frontwidths for Current Element Ordering	100
3.2. Frontwidths for Revised Element Ordering	100
3.3. Original Element Bandwidths	102
3.4. Revised Node Numbers for Bandwidth Minimization . . .	102
3.5. Revised Element Bandwidths	104
3.6. Element's Lowest Revised Node Number	104
3.7. Frontwidths for Alternative Revised Element Ordering .	105
4.1. Runtimes for All The Solvers	114
4.2. Scratch File Requirements for Both The Blocked Out-Of-Core Solver and The Frontal Solver	115
4.3. Comparison of Runtimes between The In-core Solver With and Without The Bandwidth Reduction Technique . .	118
4.4. Comparison of the Effectiveness of Nodal Renumbering .	120
4.5. Comparison of Two Selected Stiffness Vectors Before and After Nodal Renumbering Is Performed	121
4.6. Comparison of Runtimes between The Blocked Out-Of-Core Solver With and Without The Bandwidth Reduction Technique	123
4.7. Comparison of Scratch File Requirements between The Blocked Out-Of-Core Solver With and Without The Bandwidth Reduction Technique	124
4.8. Comparison of Runtimes between The Frontal Solver With and Without The Frontwidth Reduction Technique .	129
4.9. Comparison of Scratch File Requirements between The Frontal Solver With and Without The Frontwidth Reduction Technique	130

TABLE	PAGE
4.10. Comparison of Total Scratch File Requirements on Two Selected Models	131
4.11. Comparison of Runtimes Between The Blocked Out-Of-Core Solver and The Frontal Solver With Their Respective Reduction Techniques	134
4.12. Comparison of Scratch File Requirements Between The Blocked Out-Of-Core Solver and The Frontal Solver With Their Respective Reduction Techniques	135
4.13. Comparison of Total Runtimes for Both Loadcase One and Two Between The Blocked Out-Of-Core Solver and The Frontal Solver With Their Respective Reduction Techniques	137

LIST OF FIGURES

FIGURE	PAGE
2.1. Example of an Assembled Stiffness Vector	34
2.2. Example of a Diagonal Address Vector	35
2.3. Example of a Large Stiffness Matrix	47
2.4. Example of a Blocked Stiffness Matrix	47
2.5. Example of Assembled Stiffness Vector Blocks	49
2.6. Example of Blocked Diagonal Address Vectors	49
2.7. Required Disk Operations to Reduce the Fourth Stiffness Vector Block	56
2.8. Required Disk Operations to Reduce the Load Vector and Back-substitute for the Unknowns	58
2.9. Example One of a Four Plate Model	59
2.10. Stiffness Matrix of Numbering Scheme One of the Four Plate Model	61
2.11. Example Two of a Four Plate Model	62
2.12. Stiffness Matrix of Numbering Scheme Two of the Four Plate Model	62
3.1. Mesh Used for Illustration of the Frontal Method . . .	77
3.2. Mesh Used for Illustration of Frontwidth Reduction . .	98
3.3. Mesh with Nodes Renumbered	103
3.4. Mesh with Nodes and Elements Renumbered	106
B.1. Model PLT2X10.STR	214
B.2. Model PLT10X2.STR	218
B.3. Model PLT4X10.STR	222
B.4. Model PLT10X4.STR	227
B.5. Model PL10X10.STR	232
B.6. Model PL10X10A.STR	239

FIGURE	PAGE
B.7. Model TMS1.STR	247
B.8. Model BRIDGE1.STR	253
B.9. Model TESTRUN.STR	259
B.10. Model TANK.STR	267
B.11. Model SPHERE1.STR	274

I. INTRODUCTION

In the use of computer based finite element programs, the problem of solving a large number of simultaneous algebraic equations arises. In general, the solution of these equations is the single most time-consuming, core-consuming, and therefore, most expensive part of the finite element process. For linear static analysis, the solution of the equilibrium equations can take from 20 to 50 percent of the total computation time, whereas in linear dynamic or non-linear static analysis this figure can be as high as 80 to 90 percent.^{1,2}

Because the efficiency of a finite element program is highly dependent on economically solving a set of simultaneous algebraic equations, considerable research has gone into the optimization of various solution techniques to reduce both the number of operations and the size of required in-core data storage area.^{1,3} However, even if optimal in-core solution techniques are employed, very large numbers of equations can be generated that will not fit into the allocated in-core data storage areas. Such problems can be handled, however, if out-of-core solution algorithms are employed.

Purpose and Objective

The CAEFRAME⁴ linear static analysis finite element code was developed specifically for use on an IBM PC-XT or IBM PC-AT personal computer. Currently, CAEFRAME only has an in-core solution algorithm. Even though the current algorithm is very efficient, the fact that it is an in-core solution algorithm limits the size

of solvable problems. It will be shown that it is desirable to replace the present solution algorithm with one that has out-of-core capabilities so that larger problems could be solved.

The author modified two different out-of-core solution algorithms that were first developed for codes that ran on large mainframe computers, and incorporated them into the CAEFRAME code. In addition, existing efficiency improvement techniques specifically tailored to each solution algorithm were modified and incorporated into each of the solution algorithms to ascertain if, and to what degree, their capabilities and efficiencies could be improved.

The capabilities and relative efficiencies of each out-of-core solution algorithm (with and without their respective efficiency improvement techniques) were compared with each other and with the current in-core solution algorithm to determine which solution algorithm should be implemented into the CAEFRAME code as a replacement for its current in-core solver.

The Out-Of-Core Solution Algorithms

In out-of-core solution techniques, only a portion of the stiffness coefficients and load terms that comprise the set of simultaneous algebraic equations are held in the core at any time. The remaining terms are stored on high speed back-up storage devices. At appropriate times during the solution process the terms held in-core are exchanged with the necessary terms retained on the high speed back-up storage device.²

Various out-of-core solution algorithms have been developed. Most are extensions of previously developed and optimized in-core solution algorithms. The differences between the algorithms lie in how they operate on the terms in-core, and in the order and number of terms that are held in-core at any particular step during the solution process.²

In the early use of finite elements, equations of the order of 100 were considered large. Now, because of efficient in-core and out-of-core solution algorithms, equations of the order of 100,000 are routinely solved.³

For either in-core or out-of-core implementation, there are two basic types of solution techniques that can be used to solve the set of simultaneous equations: direct solution techniques and iterative solution techniques.^{2,3,5} For use in the linear static finite element method, direct solution techniques have shown to be superior to iterative methods.^{2,6} Of the direct solution techniques, those that are based on the Gauss elimination method have been preferred.^{1,2,6,7} The primary reason for this is that it can be shown that no other solution algorithm has fewer arithmetic operations.^{1,2} Since execution time is proportional to the number of arithmetic steps, it is desirable to keep the number of steps to a minimum.

The most commonly used Gauss elimination algorithms are: direct Gauss elimination, standard Cholesky factorization (also called standard Crout factorization or LL^T decomposition), modified Cholesky factorization (also called modified Crout factorization or LDL^T

decomposition), and standard frontal elimination (also called wavefront elimination).^{1-3,5-8} Even though these solution algorithms are all based on Gauss elimination, there are significant differences between them. The differences lie in their approach to the assembly and elimination (or factorization) process of the stiffness matrix and the particular way that they take advantage of the symmetry, positive definiteness, sparseness, and bandedness of the stiffness matrix.^{7,8}

These four most common solution algorithms can be divided in two groups with the first three solution algorithms in the first group and the fourth solution algorithm in the second group. The reason that they are grouped this way, is that the first three algorithms assemble the complete stiffness matrix before elimination is performed, while the fourth algorithm performs the assembly and elimination process concurrently. In addition, the out-of-core versions of the first three methods are all extensions of previously developed in-core solution algorithms, while the fourth method was developed directly as an out-of-core solver.^{2,3}

The two out-of-core solution algorithms chosen by the author as possible replacements for the current in-core solution algorithm in the CAEFRAME code were: modified Cholesky factorization, and standard frontal elimination.

The first of two reasons for choosing the modified Cholesky factorization solution algorithm is that even though for a general set of simultaneous algebraic equations no solution technique can have fewer arithmetic steps than direct Gauss elimination, for

matrices that are sparsely populated and banded, both standard Cholesky factorization and modified Cholesky factorization have shown to have better data storage capabilities, better efficiency in reducing unnecessary operations that arise due to the large number of zeros in the stiffness matrix, and better resolution capabilities than direct Gauss elimination. The modified Cholesky factorization is preferred over the standard version because of the way it calculates the diagonal terms in the factored stiffness matrix. In the standard version, a square root operation is required to calculate the diagonal terms of the factored stiffness matrix. This operation is not desirable since it tends to lead to greater inaccuracies due to roundoff errors. The degree of inaccuracy will increase as the size of the problem increases. The modified version does not have this square root operation, and thus, roundoff errors are reduced. Therefore, the modified Cholesky solution algorithm offers the best blend of data storage efficiency, operation count efficiency, resolution capability, and higher accuracy than either direct Gauss elimination or standard Cholesky factorization.^{1,8,9}

The second reason for this choice is that the existing in-core solver is based on modified Cholesky factorization (probably for all the above reasons), and therefore, this out-of-core solver can be incorporated into the CAEFRAME code as a direct extension of its current in-core solver.

For the computer implementation of the modified Cholesky algorithm, several different methods of data storage and handling have been developed. All the methods surveyed take advantage of

the symmetry of the stiffness matrix, but differ in how they take advantage of the sparseness and bandedness of the stiffness matrix.

The approaches surveyed include: various banded storage schemes with blocked reduction;^{2,6,10} the skyline storage scheme with row-wise reduction;^{2,6} and the skyline storage scheme with column-wise reduction.^{6,8,9}

The skyline approach with either row-wise or column-wise reduction has been shown to be the most efficient in reducing the number of unnecessary operations on zero terms in the stiffness matrix even though this storage scheme requires slightly more indexing for stiffness term identification.^{2,7,8,9} Of the two, the skyline storage scheme with column-wise reduction was chosen by the author as the method to implement the modified Cholesky algorithm in the CAEFRAME program. This method was chosen over the row-wise reduction, again, for the simple reason that it uses the same approach as the the current in-core solver does, and thus, program continuity could be more easily preserved.

The particular FORTRAN program used as a model by the author for implementation of the out-of-core version of the modified Cholesky factorization solution algorithm is based on work by Mondkar and Powell,^{8,9} and is referred to as "The Blocked Out-Of-Core Solver."

For the second chosen solution algorithm (standard frontal elimination) no theoretical justification for its choice is presented. This algorithm was chosen because the basic approach to the solution process is completely different than that of the

other common methods and it offers an alternative basis for comparison.

The particular FORTRAN program used as a model by the author to implement the standard frontal elimination is based on work by Hinton and Owen.¹¹ Their work is an extension of earlier work by Irons,¹² and is considered one of the most detailed and straightforward explanations of the frontal algorithm.^{1,13}

Efficiency Improvement Techniques

Another difference between the modified Cholesky factorization method and the standard frontal elimination method, is that the efficiency of each algorithm is sensitive to different properties of the actual finite element model under study.

The efficiency of Cholesky factorization is highly dependent on the bandwidth of the stiffness matrix which in turn is dependent on the node numbering of the model. Therefore, if the nodes of the model are not numbered in such a way as to produce the minimum bandwidth of the stiffness matrix, it is desirable to renumber the nodes to produce a stiffness matrix with a minimum bandwidth.^{1,2}

An algorithm developed by Collins¹⁴ was used by the author as a means of renumbering the nodes of a model in order to reduce the bandwidth. The algorithm does not always produce the optimal renumbering, but in almost all cases it does *reduce* the bandwidth of the system.

The reason that this algorithm was chosen over ones that attempt to optimally renumber the system such as those developed by: Cuthill

and McKee,¹⁵ Rosen,¹⁶ Akyuz and Utku,¹⁷ and Grooms,¹⁸ which are all discussed and referenced in Collins'¹⁴ paper, is that the chosen algorithm is faster and requires fewer coding steps than the other methods.

Since the in-core storage area is severely limited already, the trade-offs of minimizing the use of in-core storage area, along with increased speed and ease of use were considered to outweigh any increased optimization that may be obtained by using a more sophisticated method.¹⁴

The efficiency of the standard frontal solver is dependent on the element numbering rather than the node numbering. The elements should be numbered in such a manner that the degrees of freedom of the model pass through the front as quickly as possible, and thus, minimize the maximum frontwidth of the system. (The details of this concept are presented in the general discussion of the standard frontal solver in Section III.) If the elements of the model are not originally numbered in such a manner to produce the smallest maximum frontwidth, then it is desirable to renumber the elements to produce the smallest maximum frontwidth.^{2,10}

An algorithm by Sloan and Randolph¹⁹ was used as a model by the author as a way of reducing the frontwidth of a system. This algorithm does not always produce the optimal element numbering, but for most models it does *reduce* the frontwidth

The reason this method was chosen over other methods such as those developed by: King,²⁰ Aiken and Pardue,²¹ and Razzaque,²² as discussed in Sloan and Randolph's¹⁹ paper or the method developed

by Pina,¹³ is that its coding and use of in-core storage area is relatively small. It is also fast and easy to use.

Procedure for the Implementation of Solvers

The CAEFRAME code is divided into three parts: a pre-processor that is used for data input and model construction; a main-processor that assembles the stiffness matrix and load vectors, calculates the displacements of the degrees of freedom, and calculates the element stresses; and a post-processor that is used to graphically examine the results of the analysis. The solution algorithm is an integral part of the main-processor.

To compare the various configurations of solution algorithms and efficiency improvement techniques, the existing in-core solver was replaced in turn by each of the out-of-core solvers previously discussed. In addition, the efficiency improvement technique used for the blocked out-of-core solver was also incorporated into the existing in-core solver to ascertain if its capabilities could also be improved. For each main-processor configuration, test problems of various sizes and constructions were run and comparisons of storage requirements and solution time were made.

The test problems that were small enough to be solved by the current in-core solution algorithm were used as a check (since the original code has been fully verified) to ensure that the out-of-core solution algorithms would calculate the correct results.

Restrictions on Implementation

In modifying and implementing each of the out-of-core solvers and efficiency improvement techniques, three stipulations were imposed.

First, all the capabilities of the original code had to remain intact. Second, the total in-core storage area required for any configuration of the modified main-processor could not exceed the in-core storage area required for the original main-processor. The amount of in-core storage available for use by the main-processor is 520 Kbytes of the 640 Kbytes that can be accessed on an IBM PC-XT and IBM PC-AT under the DOS operating system. The remaining Kbytes are needed by the operating system and the master controlling program that runs the pre-, main-, and post-processors. Third, none of the subroutines in the original program could be altered. However, subroutines could be deleted if they were not needed for the implementation of a particular configuration of the main-processor. The original program subroutines were treated as "black boxes" with the particular out-of-core solution algorithm subroutines written by the author extracting only necessary information from storage locations set up by the original code's subroutines and then putting the results back into the correct data storage locations to be accessed by other subroutines of the original program. However, the main program of the main-processor, (which only handles in-core storage allocations, changes in file status, and subroutine calls), could be altered as needed for the proper execution of any solver.

II. THE BLOCKED OUT-OF-CORE SOLVER

The first solution algorithm chosen by the author is a Blocked Out-Of-Core solver based on the modified Cholesky algorithm. This solver is an extension of the current CAEFRAME in-core solver. In this section, the general equations that comprise the modified Cholesky method are developed first. Then, an overview of the required modifications to these general equations, along with the required data storage and handling techniques that are necessary for the in-core computer implementation of the equations, are presented. This is followed by a detailed explanation of the modifications to the in-core equations and data storage and handling techniques that allow implementation of the modified Cholesky method for out-of-core use. Finally, the theoretical justification and computer implementation of the particular bandwidth reduction technique chosen by the author to increase the efficiency of the solution algorithm is presented.

Development of the Modified Cholesky Algorithm

Since most of the solution algorithms used for linear static finite element analysis are based on direct Gauss elimination, the development of the equations for the modified Cholesky algorithm starts with a brief review of direct Gauss elimination. The review of direct Gauss elimination is presented in general terms, and then in detail through a numerical example. The modified Cholesky method is developed through a comparison and extension of the direct Gauss

elimination method using both general equations and the same numerical example.

Consider the set of linear simultaneous equations given by:

$$KU = R \quad (2.1)$$

where:

$K = [k_{ij}] = N \times N$ non-singular matrix

$U = \{u_i\} = N \times 1$ vector of unknowns to be determined

$R = \{r_i\} = N \times 1$ vector of known components

In general, the procedure for direct Gauss elimination begins by combining multiples of the first equation of the set with all subsequent equations in such a manner as to reduce all the entries k_{i1} , ($i = 2$ to N), to zero. Next, multiples of the reduced second equation are combined with all of its subsequent equations of the set in such a manner as to reduce all the entries k_{i2} , ($i = 3$ to N), to zero. This process is continued for each subsequent reduced equation until the last equation of the original set is of the form $k'_{NN}u_N = r'_N$ (where the symbol (') indicates a term that has been reduced one or more times), and the reduced coefficient matrix K is in upper triangular form (i.e., all the entries below the diagonal are zero). Then, starting with the last equation, all of the unknowns can be found by back-substitution. The order in which the unknowns will be solved is $u_N, u_{N-1}, \dots, u_1$.^{3,8}

The following example taken from Bathe³ illustrates the details of the procedure. The example starts with the equation set 2.2.

$$\begin{array}{cccc|cc|cc|cc}
 5 & -4 & 1 & 0 & & u_1 & & & 0 & \\
 -4 & 6 & -4 & 1 & & u_2 & & = & 1 & \\
 1 & -4 & 6 & -4 & & u_3 & & & 0 & \\
 0 & 1 & -4 & 5 & & u_4 & & & 0 &
 \end{array} \quad (2.2)$$

Step 1: The entries in the first column of rows two through four of equation set 2.2 need to be reduced to zero. This is accomplished by adding the correct multiple of the equation in row one to each equation in the rows two through four. For the equation in row two, row one is multiplied by $\frac{4}{5}$ and then added to row two. For the equation in row three, row one is multiplied by $-\frac{1}{5}$ and then added to row three. Since the entry in the first column of the equation in row four is already zero, reduction of this row by the equation in row one is not necessary. After completion of the first step, equation set 2.2 is reduced to the following:

$$\begin{array}{cccc|cc|cc|cc}
 5 & -4 & 1 & 0 & & u_1 & & & 0 & \\
 0 & \frac{14}{5} & -\frac{16}{5} & 1 & & u_2 & & = & 1 & \\
 0 & -\frac{16}{5} & \frac{29}{5} & -4 & & u_3 & & & 0 & \\
 0 & 1 & -4 & 5 & & u_4 & & & 0 &
 \end{array} \quad (2.3)$$

Step 2: In this step it is necessary to add the correct multiples of the reduced equation in row two of equation set 2.3 to the equations in rows three and four so as to reduce the entries in column two of those rows to zero. To satisfy this requirement, first, the equation in row two is multiplied by $\frac{16}{14}$ (reduces to $\frac{8}{7}$) and added to the equation in row three. Then, the equation in row two is multiplied by $-\frac{5}{14}$ and added to the equation in row four.

After completion of this step, the resultant set of reduced equations is as follows:

$$\left| \begin{array}{cccc|c} 5 & -4 & 1 & 0 & u_1 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 & u_2 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} & u_3 \\ 0 & 0 & -\frac{20}{7} & \frac{65}{14} & u_4 \end{array} \right| = \left| \begin{array}{c} 0 \\ 1 \\ \frac{8}{7} \\ -\frac{5}{14} \end{array} \right| \quad (2.4)$$

Step 3: Finally, the entry in column three of row four in equation set 2.4 must be reduced to zero. This is accomplished by multiplying the equation in row three by $\frac{20}{15}$ (reduces to $\frac{4}{3}$) and adding it to row four. The final set of reduced equations is:

$$\left| \begin{array}{cccc|c} 5 & -4 & 1 & 0 & u_1 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 & u_2 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} & u_3 \\ 0 & 0 & 0 & \frac{5}{6} & u_4 \end{array} \right| = \left| \begin{array}{c} 0 \\ 1 \\ \frac{8}{7} \\ \frac{7}{6} \end{array} \right| \quad (2.5)$$

Step 4: Now, all of the unknowns can be found through back-substitution:

$$\begin{aligned} u_4 &= \frac{\frac{7}{6}}{-\frac{5}{6}} = \frac{7}{5} \\ u_3 &= \frac{\frac{8}{7} - \left(-\frac{20}{7}\right) * u_4}{\frac{15}{7}} = \frac{12}{5} \\ u_2 &= \frac{1 - \left(-\frac{16}{5}\right) * u_3 - (1) * u_4}{\frac{14}{5}} = \frac{13}{5} \end{aligned} \quad (2.6)$$

$$u_1 = \frac{0 - (-4) * u_2 - (1) * u_3 - (0) * u_4}{5} = \frac{8}{5}$$

The procedure given in the above example can be stated using general formulas. If equations 1 through $n-1$ have already been eliminated, the reduction of the n^{th} equation is given by the following formulas⁸:

Reduction of the coefficient matrix:

$$k_{ij}^{(n)} = k_{ij}^{(n-1)} - \frac{k_{in}^{(n-1)} k_{nj}^{(n-1)}}{k_{nn}^{(n-1)}} \quad ; \quad \begin{matrix} i = n+1, \dots, N \\ j = n, \dots, N \end{matrix} \quad (2.7)$$

Reduction of the vector of known components:

$$r_i^{(n)} = r_i^{(n-1)} - \frac{k_{in}^{(n-1)}}{k_{nn}^{(n-1)}} * r_n^{(n-1)} \quad ; \quad i = n+1, \dots, N \quad (2.8)$$

After formulas 2.7 and 2.8 have been applied to all rows in the set of equations, the unknowns can be solved through back-substitution using the following formulas:

$$u_N = \frac{r_N^{(N-1)}}{k_{NN}^{(N-1)}} \quad (2.9)$$

$$u_i = \frac{r_i^{(i-1)} - \sum_{j=i+1}^N k_{ij}^{(i-1)} * u_j}{k_{ii}^{(i-1)}} \quad ; \quad i = N-1, \dots, 1$$

By examining the steps used to solve the example problem listed above, it can be shown that reduction of equation set 2.2 to equation set 2.5 is equivalent to the following matrix multiplications:

$$L_3^{-1} L_2^{-1} L_1^{-1} \begin{vmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{vmatrix} = L_3^{-1} L_2^{-1} L_1^{-1} \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \quad (2.10)$$

where:

$$L_1^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ \frac{4}{5} & 1 & 0 & 0 \\ -\frac{1}{5} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$L_2^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{8}{7} & 1 & 0 \\ 0 & -\frac{5}{14} & 0 & 1 \end{vmatrix} \quad (2.11)$$

$$L_3^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{4}{3} & 1 \end{vmatrix}$$

By generalizing the above matrix multiplications and examining the steps used in developing equations 2.3 through 2.9 and equation set 2.11, the reduction of the coefficient matrix K to upper triangular form can be obtained through the following:

Equation 2.12 can be rewritten as:

$$K = L_1 L_2 \dots L_{N-1} S \quad (2.16)$$

The matrix L_i is obtained by simply reversing the signs of the elements in L_i^{-1} . That is:

$$L_i = \begin{vmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & l_{i+1,i} & & \\ & & & l_{i+2,i} & & \\ & & & \vdots & & \\ & & & & & \\ & & & & & l_{Ni} & & 1 \end{vmatrix} \quad \begin{array}{l} \text{All elements not} \\ \text{shown are zero.} \end{array} \quad (2.17)$$

Therefore, the coefficient matrix can be written as:

$$K = LS \quad (2.18)$$

where: $L = L_1 L_2 \dots L_{N-1}$, or

$$L = \begin{vmatrix} 1 & & & & & \\ & l_{21} & 1 & & & \\ & l_{31} & l_{32} & 1 & & \\ & l_{41} & l_{42} & l_{43} & 1 & \\ & \vdots & \vdots & \vdots & & \\ & \vdots & \vdots & \vdots & & \\ & l_{N1} & \vdots & \vdots & \vdots & l_{N,N-1} & 1 \end{vmatrix} \quad \begin{array}{l} \text{All elements not} \\ \text{shown are zero.} \end{array} \quad (2.19)$$

Since $\mathbf{S}$ is the reduced upper triangulated coefficient matrix calculated by equation 2.7, the matrix $\mathbf{S}$ can be rewritten as $\mathbf{S} = \mathbf{D}\tilde{\mathbf{S}}$, where $\mathbf{D}$ is a diagonal matrix that stores the diagonal elements of $\mathbf{S}$ (i.e., $d_{ii} = s_{ii}$ and $d_{ij} = 0$, for $i \neq j$). If $\mathbf{S} = \mathbf{D}\tilde{\mathbf{S}}$ is substituted into equation 2.18 and it is noted that $\mathbf{K}$ is symmetric and the decomposition unique, it can be shown that $\tilde{\mathbf{S}} = \mathbf{L}^T$. Therefore:

$$\mathbf{K} = \mathbf{LDL}^T \quad (2.20)$$

It is this $\mathbf{LDL}^T$ decomposition of the coefficient matrix that can be used effectively to solve the unknowns of the equation set given in 2.1. First, substitute equation 2.20 into equation 2.1.

$$\mathbf{LDL}^T \mathbf{U} = \mathbf{R} \quad (2.21)$$

Now, through the following matrix multiplications, the unknowns can be found.

$$\mathbf{DL}^T \mathbf{U} = \mathbf{L}^{-1} \mathbf{R} = \mathbf{V} \quad (2.22)$$

$$\mathbf{L}^T \mathbf{U} = \mathbf{D}^{-1} \mathbf{V} = \tilde{\mathbf{V}} \quad (2.23)$$

By examining the example used above and equations 2.10 through 2.19, it appears that the only way of calculating elements of $\mathbf{L}$ and $\mathbf{D}$ is by successive matrix multiplications on the coefficient matrix. However, the elements of $\mathbf{L}$, $\mathbf{L}^T$, and $\mathbf{D}$ can also be calculated directly by using the following formulas:^{3,8,9}

$$d_{11} = k_{11} \quad (2.24)$$

$$g_{1j} = k_{1j} \quad (2.25)$$

$$g_{ij} = k_{ij} - \sum_{h=1}^{i-1} l_{hi} g_{hj} \quad ; \quad i = 2, \dots, j-1 \quad (2.26)$$

$$l_{ij} = \frac{g_{ij}}{d_{ii}} \quad ; \quad i = 1, \dots, j-1 \quad (2.27)$$

$$d_{jj} = k_{jj} - \sum_{h=1}^{j-1} l_{hj} g_{hj} \quad (2.28)$$

A few comments should be made about the above formulas.

First, the elements l_{ij} are the elements of L^T rather than the elements of L . Since the elements of L can be found by simply transposing the subscripts of the elements L^T and vice versa, only the elements of one of the two matrices need to be calculated. (However, in the actual computer implementation of the solution algorithm, the elements of L are not directly calculated or used.) The reason the elements of L^T are calculated rather than the elements of L is that it allows a column-wise reduction of the coefficient matrix as opposed to the row-wise reduction performed in direct Gauss elimination. This column-wise reduction will be shown to be advantageous in both data storage and data handling in the computer implementation of the algorithm, especially for the out-of-core solver.

Second, the quantities g_{ij} are only intermediate calculations and the elements of L^T are not completely calculated until equation 2.27 is evaluated.

Third, because of the symmetry of the coefficient matrix in static finite element analysis, only the elements in the upper half of the original coefficient matrix are needed for the calculations. This means that only the upper half of the coefficient matrix must be stored. Because of the column-wise reduction, the elements of L^T and D can replace the elements of K as the decomposition proceeds down through the matrix K .

To demonstrate the decomposition algorithm given in equations 2.24 through 2.28, the same numerical example solved by direct Gauss elimination in equations 2.2 through 2.6 will be solved using the LDL^T decomposition.

First, rewrite the coefficient matrix using only the elements on and above the diagonal.

$$\begin{array}{cccc|c} 5 & -4 & 1 & 0 & \\ & 6 & -4 & 1 & \\ & & 6 & -4 & \\ & & & 5 & \end{array} \quad (2.29)$$

The equations given in 2.24 through 2.28 will be applied to the above matrix on a column by column basis with j being the column number of the column currently undergoing reduction.

For $j = 1$:

$$d_{11} = k_{11} = 5$$

The first column is unchanged by the reduction process.

Therefore, after the first step, the reduced matrix is unchanged

(the dotted line separates the reduced columns from the unreduced columns):

$$\left| \begin{array}{ccccc} 5 & : & -4 & 1 & 0 \\ & : & & & \\ & : & 6 & -4 & 1 \\ & : & & & \\ & : & & 6 & -4 \\ & : & & & \\ & : & & & 5 \end{array} \right| \quad (2.30)$$

For $j = 2$:

$$g_{12} = k_{12} = -4$$

$$l_{12} = \frac{g_{12}}{d_{11}} = \frac{-4}{5}$$

$$d_{22} = k_{22} - l_{12}g_{12} = 6 - (-4)\left(\frac{-4}{5}\right) = \frac{14}{5}$$

The resultant coefficient matrix after step 2 is:

$$\left| \begin{array}{ccccc} 5 & \frac{-4}{5} & : & 1 & 0 \\ & \frac{14}{5} & : & -4 & 1 \\ & & : & & \\ & & : & 6 & -4 \\ & & : & & \\ & & : & & 5 \end{array} \right| \quad (2.31)$$

For $j = 3$:

$$g_{13} = k_{13} = 1$$

$$g_{23} = k_{23} - l_{12}g_{13} = -4 - \left(\frac{-4}{5}\right)(1) = \frac{-16}{5}$$

$$l_{13} = \frac{g_{13}}{d_{11}} = \frac{1}{5}$$

$$l_{23} = \frac{g_{23}}{d_{22}} = \frac{-\frac{16}{5}}{\frac{14}{5}} = -\frac{8}{7}$$

$$\begin{aligned} d_{33} &= k_{33} - l_{13}g_{13} - l_{23}g_{23} \\ &= 6 - \left(\frac{1}{5}\right)(1) - \left(-\frac{8}{7}\right)\left(-\frac{16}{5}\right) = \frac{15}{7} \end{aligned}$$

The resultant coefficient matrix after step 3 is:

$$\left[\begin{array}{ccc|c} 5 & -\frac{4}{5} & \frac{1}{5} & 0 \\ & \frac{14}{5} & -\frac{8}{7} & 1 \\ & & \frac{15}{7} & -4 \\ & & & 5 \end{array} \right] \quad (2.32)$$

For $j = 4$:

$$g_{14} = k_{14} = 0$$

$$g_{24} = k_{24} - l_{12}g_{14} = 1 - \left(-\frac{4}{5}\right)(0) = 1$$

$$\begin{aligned} g_{34} &= k_{34} - l_{13}g_{14} - l_{23}g_{24} \\ &= -4 - \left(\frac{1}{5}\right)(0) - \left(-\frac{8}{7}\right)(1) = -\frac{20}{7} \end{aligned}$$

$$l_{14} = \frac{g_{14}}{d_{11}} = \frac{0}{5} = 0$$

$$l_{24} = \frac{g_{24}}{d_{22}} = \frac{1}{\frac{14}{5}} = \frac{5}{14}$$

$$l_{34} = \frac{g_{34}}{d_{33}} = \frac{-\frac{20}{7}}{\frac{15}{7}} = -\frac{4}{3}$$

$$\begin{aligned} d_{44} &= k_{44} - l_{14}g_{14} - l_{24}g_{24} - l_{34}g_{34} \\ &= 5 - (0)(0) - \left(\frac{5}{14}\right)(1) - \left(-\frac{4}{3}\right)\left(-\frac{20}{7}\right) = \frac{5}{6} \end{aligned}$$

Finally, the resultant coefficient matrix after step 4 is:

$$\begin{vmatrix} 5 & -\frac{4}{5} & \frac{1}{5} & 0 \\ & \frac{14}{5} & -\frac{8}{7} & \frac{5}{14} \\ & & \frac{15}{7} & -\frac{4}{3} \\ & & & \frac{5}{6} \end{vmatrix} \quad (2.33)$$

Note that the elements of D are stored on the diagonal and the elements l_{ij} have replaced the elements k_{ij} , $j > i$. The two matrices D and L^T can easily be separated from equation 2.33:

$$D = \begin{vmatrix} 5 & 0 & 0 & 0 \\ & \frac{14}{5} & 0 & 0 \\ & & \frac{15}{7} & 0 \\ & & & \frac{5}{6} \end{vmatrix} \quad (2.34)$$

$$L^T = \begin{vmatrix} 1 & -\frac{4}{5} & \frac{1}{5} & 0 \\ & 1 & -\frac{8}{7} & \frac{5}{14} \\ & & 1 & -\frac{4}{3} \\ & & & 1 \end{vmatrix} \quad (2.35)$$

It can be shown that $DL^T = S$ (where S is the coefficient matrix given in equation set 2.5). That is:

$$DL^T = \begin{vmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} \\ 0 & 0 & 0 & \frac{5}{6} \end{vmatrix} = S \quad (2.36)$$

The formulas given in equations 2.24 through 2.28 are only for the decomposition of the coefficient matrix to complete the algorithm, the equations that solve for the unknowns as shown in equations 2.22 and 2.23 need to be given.

The equations to do the partial reduction of the vector of known components, as in equation 2.22, are:

$$v_1 = r_1 \quad (2.37)$$

$$v_i = r_i - \sum_{h=1}^{i-1} l_{hi} v_h \quad ; \quad i = 2, \dots, N \quad (2.38)$$

To calculate the $\tilde{V}$ vector given in equation 2.23 the following equation is used:

$$\tilde{v}_i = \frac{v_i}{d_{ii}} \quad ; \quad i = 1, \dots, N \quad (2.39)$$

Finally, to find the unknowns, U , also given in equation 2.23, the following equations are used:

$$\tilde{V}^{(N)} = \tilde{V} \quad (2.40)$$

$$u_N = \tilde{v}_N^{(N)} \quad (2.41)$$

For $i = N-1, \dots, 1$:

$$\tilde{v}_h^{(i)} = \tilde{v}_h^{(i+1)} - l_{h, i+1} u_{i+1} \quad ; \quad h = 1, \dots, i \quad (2.42)$$

$$u_i = \tilde{v}_i^{(i)} \quad (2.43)$$

To complete the numerical example, the unknowns will be solved using equations 2.37 through 2.43. First rewrite the vector of known components:

$$\mathbf{R} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad (2.44)$$

Using equations 2.37 and 2.38, the vector $\mathbf{V}$ can be calculated:

$$v_1 = r_1 = 0$$

$$v_2 = r_2 - l_{12}v_1 = 1 - \left(-\frac{4}{5}\right)(0) = 1$$

$$\begin{aligned} v_3 &= r_3 - l_{13}v_1 - l_{23}v_2 = 0 - \left(\frac{1}{5}\right)(0) - \left(-\frac{8}{7}\right)(1) \\ &= \frac{8}{7} \end{aligned}$$

$$\begin{aligned} v_4 &= r_4 - l_{14}v_1 - l_{24}v_2 - l_{34}v_3 \\ &= 0 - (0)(0) - \left(\frac{5}{14}\right)(1) - \left(-\frac{4}{3}\right)\left(\frac{8}{7}\right) = \frac{7}{6} \end{aligned}$$

The resultant $\mathbf{V}$ vector is:

$$\mathbf{V} = \begin{bmatrix} 0 \\ 1 \\ \frac{8}{7} \\ \frac{7}{6} \end{bmatrix} \quad (2.45)$$

Next, equation 2.39 is used to calculate the $\tilde{\mathbf{V}}$ vector:

$$\tilde{v}_1 = \frac{v_1}{d_{11}} = \frac{0}{5} = 0$$

$$\tilde{v}_2 = -\frac{v_2}{d_{22}} = -\frac{1}{\frac{14}{5}} = \frac{5}{14}$$

$$\tilde{v}_3 = -\frac{v_3}{d_{33}} = -\frac{\frac{8}{7}}{\frac{15}{7}} = \frac{8}{15}$$

$$\tilde{v}_4 = -\frac{v_4}{d_{44}} = -\frac{\frac{7}{6}}{\frac{5}{6}} = \frac{7}{5}$$

Therefore, the resultant $\tilde{\mathbf{V}}$ vector is:

$$\tilde{\mathbf{V}} = \begin{bmatrix} 0 \\ \frac{5}{14} \\ \frac{8}{15} \\ \frac{7}{5} \end{bmatrix} \quad (2.46)$$

Finally, the unknowns, $\mathbf{U}$, can be found by using equations 2.40 through 2.43. First apply equations 2.40 and 2.41.

$$\tilde{v}_1^{(4)} = \tilde{v}_1 = 0$$

$$\tilde{v}_2^{(4)} = \tilde{v}_2 = \frac{5}{14}$$

$$\tilde{v}_3^{(4)} = \tilde{v}_3 = \frac{8}{15}$$

$$\tilde{v}_4^{(4)} = \tilde{v}_4 = \frac{7}{5}$$

$$u_4 = \tilde{v}_4^{(4)} = \frac{7}{5}$$

At this point the vector $\tilde{\mathbf{V}}$ is unchanged and, therefore, the above step is redundant. However, the step is shown for completeness

and to explain the logic underlying equations 2.40 through 2.43.

Using equations 2.42 and 2.43, the rest of the unknowns can be found.

For $i = 3$:

$$\tilde{v}_1^{(3)} = \tilde{v}_1^{(4)} - l_{14} u_4 = 0 - (0)\left(\frac{7}{5}\right) = 0$$

$$\tilde{v}_2^{(3)} = \tilde{v}_2^{(4)} - l_{24} u_4 = \frac{5}{14} - \left(\frac{5}{14}\right)\left(\frac{7}{5}\right) = -\frac{1}{7}$$

$$\tilde{v}_3^{(3)} = \tilde{v}_3^{(4)} - l_{34} u_4 = \frac{8}{15} - \left(-\frac{4}{3}\right)\left(\frac{7}{5}\right) = \frac{12}{5}$$

$$u_3 = \tilde{v}_3^{(3)} = \frac{12}{5}$$

For $i = 2$:

$$\tilde{v}_1^{(2)} = \tilde{v}_1^{(3)} - l_{13} u_3 = 0 - \left(\frac{1}{5}\right)\left(\frac{12}{5}\right) = -\frac{12}{25}$$

$$\tilde{v}_2^{(2)} = \tilde{v}_2^{(3)} - l_{23} u_3 = -\frac{1}{7} - \left(-\frac{8}{7}\right)\left(\frac{12}{5}\right) = \frac{13}{5}$$

$$u_2 = \tilde{v}_2^{(2)} = \frac{13}{5}$$

For $i = 1$:

$$\tilde{v}_1^{(1)} = \tilde{v}_1^{(2)} - l_{12} u_2 = -\frac{12}{25} - \left(-\frac{4}{5}\right)\left(\frac{13}{5}\right) = \frac{8}{5}$$

$$u_1 = \tilde{v}_1^{(1)} = \frac{8}{5}$$

The elements stored in the vector $\tilde{\mathbf{V}}$ after the application of equations 2.42 and 2.43 for $i = 4, 3, 2$, respectively are:

$$\begin{array}{|c|} \hline 0 \\ \hline -\frac{1}{7} \\ \hline \frac{12}{5} \\ \hline \frac{7}{5} \\ \hline \end{array} ; \begin{array}{|c|} \hline -\frac{12}{25} \\ \hline \frac{13}{5} \\ \hline \frac{12}{5} \\ \hline \frac{7}{5} \\ \hline \end{array} ; \begin{array}{|c|} \hline \frac{8}{5} \\ \hline \frac{13}{5} \\ \hline \frac{12}{5} \\ \hline \frac{7}{5} \\ \hline \end{array} \quad (2.47)$$

where the last vector is the solution to the vector of unknowns, U . As a check, this solution agrees with the solution found by direct Gauss elimination given in equation 2.6.

In linear static finite element analysis, the coefficient matrix corresponds to the structural stiffness matrix; the vector of known components corresponds to the vector of degree of freedom loads or the load vector; and the vector of unknowns corresponds to the vector of degree of freedom displacements or the displacement vector.

The stiffness matrix has several properties that all solution algorithms should address. These properties are: the stiffness matrix is positive definite (the diagonal terms are positive and non-zero), it is symmetric ($k_{ij} = k_{ji}$), and it is generally sparse (only a small number of the total elements of the matrix are non-zero, and in most cases, the non-zero terms tend to be concentrated near the diagonal).

Not only is the original stiffness matrix positive definite, but as the solution proceeds, the stiffness matrix must always remain positive definite or the structure is unstable. For a stable structure the modified Cholesky method preserves the positive definiteness of the matrix. As shown earlier, the equations that calculate the elements of L^T and D account for, and take advantage of the symmetry of the stiffness matrix. But, the decomposition given in equations 2.24 through 2.28 does not take advantage of sparsity of the matrix. However, the equations can be modified to do so.

If variable m_j is defined as the row number of the first non-zero entry in column j , then the set of variables $\{m_j\}$, $j = 1, \dots, N$ define what is known as the skyline of the stiffness matrix. If the numerical example shown above, and equations 2.24 through 2.28 are examined, it can be shown that all the zero entries in the stiffness matrix above the skyline remain zero after it is decomposed into the L^T and D matrices. In addition, the zero entries above the skyline do not affect the decomposition of any of the non-zero entries below the skyline of the stiffness matrix. Therefore, both the storage and reduction of the zero elements above the skyline of the stiffness matrix are redundant and the use of these terms can be eliminated from the solution technique. To incorporate this space saving and calculation saving aspect of the stiffness matrix into the solution algorithm, equations 2.24 through 2.28 can be rewritten as:^{3,8,9}

$$d_{11} = k_{11} \quad (2.48)$$

$$g_{m_j, j} = k_{m_j, j} \quad (2.49)$$

$$g_{ij} = k_{ij} - \sum_{h=m_j}^{i-1} l_{hi} g_{hj} \quad ; \quad i = m_j + 1, \dots, j-1 \quad (2.50)$$

$$; \quad m_m = \max\{m_i, m_j\}$$

$$l_{ij} = \frac{g_{ij}}{d_{ii}} \quad ; \quad i = m_j, \dots, j-1 \quad (2.51)$$

$$d_{jj} = k_{jj} - \sum_{h=m_j}^{j-1} l_{hj} g_{hj} \quad (2.52)$$

The advantage of eliminating the zero terms outside the skyline of the stiffness matrix can also be incorporated into equation 2.38 of the load reduction equations 2.37 through 2.39. The summation of the terms $l_{hi}v_i$ in equation 2.38 using elements of the stiffness matrix above the skyline of column i does not affect the value of v_i since all the l_{hi} terms above the skyline are zero by equations 2.49 through 2.51. Therefore, the summation in equation 2.38 only needs to be performed from $h = m_i$ to $i-1$. Even though this modification improves the efficiency of reducing the load vector by eliminating all unnecessary operations due to the zero terms outside the skyline of the stiffness matrix, still more improvements can be made if the load vector itself is examined.

If r_m is defined as the first non-zero load term, it is obvious that equation 2.37 is unnecessary if $m \neq 1$. Therefore, if the condition $m \neq 1$ is true, equation 2.37 can be ignored. Furthermore, the calculation of the terms v_i and $\tilde{v}_i$ by equations 2.38 and 2.39 for $i = 1, \dots, r_m - 1$ is not necessary since the terms v_i and $\tilde{v}_i$ $i = 1, \dots, r_m - 1$ are originally zero and will still remain zero if equations 2.38 and 2.39 are executed. In addition, if $r_m > m_i$, the summation of the terms $l_{hi}v_i$ in equation 2.38 from $h = m_i, \dots, r_m - 1$ does not affect the value of v_i since all the v_h terms for $h = m_i, \dots, r_m - 1$ are zero as stated in the previous sentence. Thus, for even greater efficiency, the summation in equation 2.38 should be performed from $h = \max\{m_i, r_m\}$ to $i-1$. The set of load reduction equations 2.37 through 2.38 can be modified to reduce the number of unnecessary operations as follows:

$$v_1 = r_1 \quad ; \quad \text{only if } m = 1 \quad (2.53)$$

$$v_i = r_i - \sum_{h=m}^{i-1} l_{hi} v_h \quad ; \quad i = \max\{2, r_m\}, \dots, N \quad (2.54)$$

$$; \quad m_m = \max\{m_i, r_m\}$$

$$\tilde{v}_i = \frac{v_i}{d_{ii}} \quad ; \quad i = r_m, \dots, N \quad (2.55)$$

For equations 2.40 through 2.43, which solve for the unknowns through back-substitution, only equation 2.42 can be modified to eliminate the unnecessary operations that arise due to the zero terms above the skyline in the stiffness matrix. The modified equations that perform the back-substitution are:

$$\tilde{\mathbf{v}}^{(N)} = \tilde{\mathbf{v}} \quad (2.56)$$

$$u_N = \tilde{v}_N^{(N)} \quad (2.57)$$

For $i = N-1, \dots, 1$:

$$\tilde{v}_h^{(i)} = \tilde{v}_h^{(i+1)} - l_{h, i+1} u_{i+1} \quad ; \quad h = m_{i+1}, \dots, i \quad (2.58)$$

$$u_i = \tilde{v}_i^{(i)} \quad (2.59)$$

Equations 2.48 through 2.59 comprise the complete set of equations that are actually used for the computer implementation of the modified Cholesky solution algorithm. As shown, these equations ensure that the stiffness matrix remains positive definite, and reduce the number of required operations and storage requirements by utilizing the fact that the stiffness matrix is symmetric. In

addition, these equations take advantage of the sparsity of the stiffness matrix by eliminating all unnecessary operations that arise due to the zero terms above the skyline of the stiffness matrix and all the zero load terms up to the first non-zero load term. However, these equations still do not reflect the optimal efficiency that could be attained.

If a l_{ij} term below the skyline of the coefficient matrix is zero, the multiplication that occurs in equations 2.50, 2.52, 2.54, and 2.58 using that term is unnecessary. Also, if a g_{ij} term below the skyline of the coefficient matrix is zero, the multiplication that occurs in equations 2.50 and 2.52, and the division in equation 2.51 using that term is unnecessary. Furthermore, the multiplication in equation 2.54 and the division in equation 2.55 are also redundant if a zero v_i term occurs after the first non-zero load term. However, the additional testing, logic, and coding that would be required to eliminate these unnecessary operations would be more time and space consuming than just carrying out the unnecessary operations for most cases.^{3,8,9} Therefore, this additional testing, logic, and coding is not incorporated into the solution algorithm.

The way to increase the efficiency of the algorithm is to assemble the set of equations in such a manner as to reduce both the number of zero and non-zero terms below the skyline of the stiffness matrix. An algorithm that will perform this function will be addressed in the last part of this section.

In-Core Computer Implementation

The actual computer implementation of the algorithm begins by first constructing the stiffness matrix. Even though the examples, and derivations, shown in equations 2.1 through 2.59 refer to the stiffness matrix in two dimensions, the stiffness matrix is in practice assembled as a one dimensional vector with only the elements from the top of the skyline through the diagonal of each column stored. The columns are stored with each column following the next in ascending column number order. For each individual column, the element that defines the top of the skyline occupies the first storage location allocated to that column, and the diagonal element occupies the last storage location. This compacted column storage system does not store any of the zero terms above the skyline of the stiffness matrix and, therefore, it is obviously more efficient than using a two dimensional array which would require storage of the zero terms above the skyline.^{3,8,9} For the numerical example used earlier, the assembled stiffness vector would be stored as shown in Figure 2.1.

Address:	1	2	3	4	5	6	7	8	9
Value:	5	-4	6	1	-4	6	1	-4	5

Figure 2.1. Example of an Assembled Stiffness Vector.

For proper execution of equations 2.48 through 2.52 in the computer implementation of the algorithm, each term in the one dimensional stiffness vector has to be correctly identified with its original location in the two dimensional stiffness matrix. In order to identify a particular term in the stiffness vector with its original column and row number, some form of indexing has to be established. To facilitate the required indexing, a one dimensional vector is also assembled that stores the addresses in the stiffness vector of the diagonal terms of each column. The address in this address vector corresponds to the column number, while the value in this address vector corresponds to the address in the stiffness vector of that column's diagonal term. For the same numerical example used above, the assembled diagonal address vector would be stored as shown in Figure 2.2.

Address:	1	2	3	4
Value:	1	3	6	9

Figure 2.2. Example of a Diagonal Address Vector.

Using the diagonal address vector, a simple procedure can be set up to find the original column and row number of any term in the stiffness vector. First, scan the address vector to find the address and column number of the first diagonal term that is stored in the stiffness vector after the term in question. The column

number of this diagonal term will be the same as the column number of the term in question. Since for any diagonal term, the row number is the same as the column number, it is easy to find the row number of the term in question by taking the difference between the addresses in the stiffness vector of the diagonal term stored immediately after the term in question and the term in question itself, and then subtracting that difference from the column number of the term in question. In the computer implementation of the algorithm, a slight variation of this method is actually used.

After the stiffness vector and the diagonal address vector are assembled, the stiffness vector is reduced using equations 2.48 through 2.52 to obtain the matrices L^T and D which will also be stored in vector form. To explain the computer implementation of equations 2.48 through 2.52, an outline of the step by step procedure for the reduction of the j^{th} column will be given. Since this version of the modified Cholesky method is a column-wise reduction process, columns 1 through $j-1$ have already been reduced and the elements of L^T and D have replaced the elements of K in the stiffness vector for these columns. Also, equation 2.48 has already been evaluated at this point in the procedure.

Step 1: Retrieve the value from the diagonal address vector at the position that corresponds to the current column number j , and store this value in a scratch index pointer called *current diagonal address*. (For purposes of this discussion, all scratch index pointers will be referred to by names written in italics.)

The value in *current diagonal address* will be the address in the stiffness vector of the diagonal term of the current column j .

Step 2: Retrieve the value from the diagonal address vector at the $j-1$ position and store the value in another scratch index pointer called *current skyline address*. At this point, the value stored in *current skyline address* will be the address in the stiffness vector of the diagonal term of the column previous to the column being reduced. Increment the value of *current skyline address* by one. This will now set its value to the address in the stiffness vector of the first non-zero term m_j of the current column j , (i.e., the address of the "skyline" of the current column j).

Step 3: The row number of the first non-zero term of the current column is found by taking the difference between the value stored in *current diagonal address* and the value stored in *current skyline address* and then subtracting the difference from the value of the current column number j . The resultant value is stored in a scratch index pointer called *current skyline row number*.

Step 4: Copy the values in *current skyline address* and *current skyline row number* to two other scratch index pointers respectively called *current term address* and *current term row number*. The values in these two scratch index pointers will give the address and the row number of the current term in the current column being reduced.

Step 5: Increment the values of both *current term address* and *current term row number* by one. This will evaluate equation

2.49 for this column and set the starting value of the index i in equations 2.50 and 2.51.

Step 6: Retrieve the value from the address vector at the position that matches the value stored in *current term row number* and store this value in a scratch index pointer called *reduced diagonal address*.

Step 7: Retrieve the address of the diagonal term of the column previous to the one just obtained. Increment that value by one to calculate the address of the first non-zero term of the already reduced column and store the new value in a scratch index pointer called *reduced skyline address*.

Step 8: By the same procedure used in step 3, calculate the row number of the first non-zero term in this already reduced column and store the value in a scratch index pointer called *reduced skyline row number*.

Step 9: Compare the values of *reduced skyline row number* and *current skyline row number* to see if the row number of the first non-zero term of the already reduced column is greater than the row number of the first non-zero term of the current column. If it is, go to step 13. If it is not, proceed with the next step.

Step 10: Set a scratch summation variable to zero. Take the difference between the values stored in *current skyline row number* and *reduced skyline row number* and then increment the value stored in *reduced skyline address* by the amount of the difference. This will set the value of *reduced skyline address* to the address in the stiffness vector that stores the term in the already reduced

column that has the same row number as the the first non-zero term of the current column being reduced.

Step 11: Multiply the value stored in the stiffness vector at the address given by *reduced skyline address* by the value stored at the address given by *current skyline address* and add the result to the summation variable.

Step 12: Subtract the value of the summation variable from the value in the stiffness vector stored in the location given by *current term address* and store the result back in that same location of the stiffness vector. This completes the evaluation of equation 2.50 for the first value of i , and results in the storage of the value of $g_{mj+1,j}$ in the same loaction as $k_{mj+1,j}$ in the stiffness vector.

Step 13: Increment the values of both *current term row number* and *current term address* by one so that the next term in the current column can be reduced by equation 2.50.

Step 14: Obtain the value from the address vector at the position given by the new value of *current term row number* and store it in *reduced diagonal address*. Using the same procedure as in steps 7 and 8, reset the new values of both *reduced skyline address* and *reduced skyline row number* for this new already reduced column.

Step 15: Compare the values of *reduced skyline row number* and *current term row number*. If the value of *reduced skyline row number* is greater than or equal to the value of *current term row number*, go to step 20. If it is not, proceed with the next step.

Step 16: Compare the values of *current skyline row number* and *reduced skyline row number*. Reset the value of *reduced skyline row number* to the higher of the two values. Set two new scratch index pointers *summation reduced* and *summation current* to the addresses in the stiffness vector of the terms in the already reduced column and the current column respectively, that have the row number given by the value now stored in *reduced skyline row number*. Also, reset the summation variable to zero.

Step 17: Retrieve the values from the stiffness vector at the address given by *reduced summation* and *current summation* and then multiply the values together and add to the summation variable.

Step 18: Increment the values of both *reduced summation* and *current summation* and then repeat step 17. This step is repeated for all values of *current summation* that are less than the value of *current term address*.

Step 19: Subtract the value of the summation variable from the value of the term of the stiffness vector stored at the location given by *current term address* and store the result in the same location of the stiffness vector. This completes the evaluation of equation 2.50 for the current term of the current column.

Step 20: Repeat steps 13 through 19 for all values of *current term address* that are less than the value of *current diagonal address*.

At this point, all the values of g_{ij} have been calculated for the current value of j and they have replaced the values of k_{ij}

in the stiffness vector. This completes the evaluation of equation 2.50 for the current column.

The evaluation of both equations 2.51 and 2.52 for the current column j are performed at the same time in the following manner.

Step 21: Reset the summation variable to zero.

Step 22: Copy the value of the term in the stiffness vector at the address given by the value of *current skyline address* to a scratch variable called *partial reduction*.

Step 23: Retrieve the value from the address vector at the position given by the value of *current skyline row number* and store the value in *reduced diagonal address*.

Step 24: Divide the term in the stiffness vector at the address given by *current skyline address* by the term in the stiffness vector at the address given by *reduced diagonal address* and store the results back in the location of the stiffness vector given by the value of *current skyline address*. This evaluates equation 2.51 for the current term of the current column.

Step 25: Multiply the just calculated new value of the term in the stiffness vector at the address given by *current skyline address* and the value in the scratch variable *partial reduction* and add the results to the summation variable.

Step 26: Increment the value of *current skyline address* by one and repeat steps 22 through 25. This step is executed for all the values of *current skyline address* that are less than the value of *current diagonal address*.

At this point, equation 2.51 has been evaluated for all values of i for the current column j and the values of l_{ij} have replaced the values of g_{ij} in the stiffness vector.

Step 27: Subtract the value of the summation variable from the term in the stiffness vector at the address given by *current diagonal address* and store the results back in the same location of the stiffness vector. This evaluates equation 2.52 for the current column and replaces the value of k_{jj} in the stiffness vector with the value of d_{jj} .

The entire procedure given by steps 1 through 27 is repeated for each succeeding column until the last column of the stiffness vector has been reduced. At this point, all the elements of both L^T and D will have been calculated and they will have replaced the original stiffness terms in the stiffness vector.

The application of equations 2.53 through 2.55 for the reduction of the load vector is not as complicated as the reduction of the stiffness vector.

Step 1: The load vector is searched starting with r_1 for the first non-zero load term. The index of the position of the first non-zero load term is stored in r_m .

Step 2: All the values of v_i are then set to the value of r_i (which are zero except for r_m) for $i = 1, \dots, r_m$.

The explanation given in the following steps will be for the reduction of an arbitrary r_j for any j greater than m .

Step 3: Retrieve the value from the diagonal address vector

at the position that corresponds to the current index number j , and store this value in *reduced diagonal address*.

Step 4: Retrieve the value from the diagonal address vector at the $j-1$ position and then increment the value by one. Store the new value in *reduced skyline address*.

Step 5: By the same procedure used in step 3 of the reduction of the stiffness vector, calculate the row number of the first non-zero term in the reduced column j of the stiffness vector and store the value in *reduced skyline row number*.

Step 6: Compare the values of *reduced skyline row number* and r_m . If the value of r_m is greater than the value of *reduced skyline row number*, reset the value of *reduced skyline row number* to the value of r_m and add the difference between the values of r_m and *reduced skyline row number* to the value of *reduced skyline address*.

Step 7: Reset the summation variable to zero.

Step 8: Retrieve the value from the reduced stiffness vector at the address given by *reduced skyline address* and the value from the load vector at the address given by *reduced skyline row number* and then multiply the two values together and add to the summation variable.

Step 9: Increment the values of both *reduced skyline address* and *reduced skyline row number* and then repeat step 8. This step is repeated for all values of *reduced skyline row number* that are less than the value of j .

Step 10: Subtract the value of the summation variable from the value of the term in the load vector stored at the location

given by j and store the result back in the same location of the load vector. This completes the evaluation of equation 2.54 for the current load term.

Step 11: Repeat steps 3 through 10 for all values of j . At this point, equation 2.54 has been evaluated for all of the terms in the load vector, and the load vector R has been reduced to, and replaced by, the vector V .

Step 12: Retrieve the value from the diagonal address vector at the position given by the value of r_m and store it the *reduced diagonal address*.

Step 13: Divide the value of the term of the reduced load vector at the address given by the value of r_m by the value of the term of the stiffness vector at the address given by the value of *reduced diagonal address* and store the results back in the same location of the load vector.

Step 14: Increment the value of r_m by one and repeat steps 12 and 13 for all the remaining load terms. This will complete the evaluation of equation 2.55 for all load terms. At this point, the vector V has been reduced to, and replaced by, the vector $\tilde{V}$.

Equations 2.56 through 2.59 can now be applied to solve for the unknown displacements.

Step 1: Set the value of the index j to the column number of the last column in the stiffness vector.

Step 2: Retrieve the value from the diagonal address vector at the position that corresponds to the current index number j , and store this value in *reduced diagonal address*.

Step 3: Retrieve the value from the diagonal address vector at the $j-1$ position and then increment the value by one. Store the new value in *reduced skyline address*.

Step 4: By the same procedure used in step 3 of the reduction of the stiffness vector, calculate the row number of the first non-zero term in the reduced column j of the stiffness vector and store the value in *reduced skyline row number*.

Step 5: Multiply the value of the load term at the address given by the value of j by the value of the stiffness term at the address given by the value of *reduced skyline address* and then subtract the product from the load term at the address given by *reduced skyline row number*.

Step 6: Increment the value of both *reduced skyline row number* and *reduced skyline address* by one and repeat steps 2 through 5 for all values of *reduced skyline row number* that are less than the value of j .

Step 7 Decrement the value of j by one and repeat steps 2 through 6 for all values of j that are not equal to one.

This will complete the evaluation of equations 2.56 through 2.59 and the values of the unknown displacements, U , will now have replaced the values of $\tilde{V}$ in the load vector.

Out-of-Core Computer Implementation

When the size of the stiffness vector is too large to fit in the allocated core space, modifications to equations 2.48 through 2.59 and the required data storage and handling techniques can be

made to allow for use of the available out-of-core storage area. The basic approach is to first divide the stiffness vector into blocks that will each individually fit into the allocated core space and then store these blocks on the available out-of-core disk area. Then, one by one, the individual blocks are brought into the core and reduced using equations 2.48 through 2.52 and restored on the disk. Equations 2.53 through 2.59 can then be used to reduce the load vector and solve for the unknown displacements. The actual equations of 2.48 through 2.59 will not change, but the order of their execution and the data storage and handling techniques will change.

As an example, assume that only 16 data storage areas are available in the core for the stiffness vector. The matrix shown in Figure 2.3 would be too large to fit in-core and it could not be solved by the current solver. To solve the problem, the matrix is divided into blocks and stored on the available disk storage area. There are two rules that must be followed in establishing these blocks. First, each block must contain at least one complete column of the stiffness matrix, and second, the size of a block cannot exceed one half of the core area allocated for the stiffness vector. The algorithm could be modified so that the first rule would not apply, but this would require considerable more coding and the benefits would be minimal. The second rule is necessary because at certain phases of the solution, two blocks will be required to be in the core at the same time. Applying these rules, the matrix in Figure 2.3 can be divided as shown in Figure 2.4.

Again, the stiffness matrix is not actually stored in matrix form, but rather in vector form. Therefore, the corresponding stiffness vector blocks of Figure 2.4 would be stored as shown in Figure 2.5.

As with the in-core solver, a method of determining the address in the stiffness vector blocks of each term has to be established. Again, the same method of using the addresses of the diagonal terms as indexes and storing these addresses in a separate address vector will be used. However, since various combinations of blocks can be in the core at any time, the diagonal addresses will not be stored as one vector, but rather they will be stored as separate blocks with one diagonal address vector block for each of the stiffness vector blocks. For the example given in Figure 2.5, the corresponding diagonal address blocks would be as shown in Figure 2.6.

The general procedure is similar to the in-core operations, namely the reduction is performed from the skyline term to the diagonal term of each column and the reduction proceeds from the upper left corner down to the right. The difference is that even though all of the columns in one particular block will be completely reduced before any reduction begins on the next block, the reduction performed in equations 2.49 through 2.52 may not be fully applied to all the terms in one column of the current block under reduction before they are applied to some of the terms in the next or subsequent columns of the same block.

BLOCK 1:	Address:	1	2	3	4	5	6	7	8
	Value:	1	2	3	4	5	6	Ø	Ø
BLOCK 2:	Address:	1	2	3	4	5	6	7	8
	Value:	7	8	9	10	11	12	13	Ø
BLOCK 3:	Address:	1	2	3	4	5	6	7	8
	Value:	14	15	16	17	18	Ø	Ø	Ø
BLOCK 4:	Address:	1	2	3	4	5	6	7	8
	Value:	19	20	21	22	23	24	25	26

Figure 2.5. Example of Assembled Stiffness Vector Blocks.

BLOCK 1:	Address:	1	2	3	4
	Value:	1	3	6	Ø
BLOCK 2:	Address:	1	2	3	4
	Value:	3	7	Ø	Ø
BLOCK 3:	Address:	1	2	3	4
	Value:	5	Ø	Ø	Ø
BLOCK 4:	Address:	1	2	3	4
	Value:	3	8	Ø	Ø

Figure 2.6. Example of Blocked Diagonal Address Vectors.

Similar to the in-core solver, the first step is to set up all the stiffness vector blocks and their associated diagonal address vector blocks and then write all these blocks to the disk. Since the original CAEFRAME code was set up to only assemble and reduce the stiffness vector in-core, those subroutines had to be deleted from the code and new ones written by the author were substituted. The subroutines written by the author control the assembly and writing to disk of the stiffness vector blocks and their associated diagonal address vectors are STFOUT, BLKSTIF, ASSTIFF, and STIFFS, and are listed in Appendix C. These subroutines are based on the original in-core subroutines of the CAEFRAME code.

Once both of these sets of blocks are established and written to disk, they are brought back into the core for reduction in ascending numerical order. The *primary block* will be the block currently undergoing reduction and will be stored in one half of the core area allocated for the stiffness vector designated as section A. A *subordinate block* is an already reduced stiffness vector block that is required to wholly or partially reduce some or all of the terms of the primary block. Subordinate blocks will be stored as needed in the second half of the core area allocated for the stiffness vector designated as section B.

The procedure is to bring the current primary block in the core and store it in section A. In addition, the primary block's diagonal address vector is brought into the core and stored in its allocated area. Then, the already reduced blocks are brought into the core sequentially in ascending order and stored in section B.

As each subordinate block is brought into core, its diagonal address vector is also brought into the core and stored in its allocated area.

While each subordinate block and its associated diagonal address vector are in the core, all of the off-diagonal terms of the primary block that lie in the *horizontal shadow* of the current subordinate block are partially reduced by equations 2.49 and 2.50. (The *horizontal shadow* of a subordinate block is all the rows of the primary block that lie between the diagonal term of the first column and the diagonal term of the last column of the subordinate block inclusively.) The procedure for the reduction of these off-diagonal terms is the same as outlined in steps 1 through 19 for the in-core implementation. The only difference is that the indexing used to find addresses of stiffness terms is a little more complicated since two different stiffness vectors and two different diagonal address vectors are in the core at the same time.

For the current subordinate block, the out-of-core procedure differs from the in-core procedure with step 20. Instead of reducing all the off-diagonal terms of the current column using steps 1 through 19, the partial reduction of equation 2.50 for the current column ends when the last term in that column that lies in the horizontal shadow of the current subordinate block is partially reduced. Also, when all the terms of the current column of the primary block that lie in the horizontal shadow of the current subordinate block are partially reduced by equation 2.50, the terms in the next column of the primary block that lie in the horizontal

shadow of the current subordinate block are reduced by equations 2.49 and 2.50 using steps 1 through 19 of the in-core procedure instead of using equation 2.51 to fully reduce the already partially reduced terms of the current column.

Thus, terms from several or all of the columns of the primary block may be partially reduced and the k_{ij} terms replaced the g_{ij} terms before the final reduction of any of the terms of the first column of the primary block is carried out.

This procedure to do the partial reduction of the off-diagonal terms of the primary block is carried out for each subordinate block. Therefore, the partial reduction of all the terms in the primary block that lie in the horizontal shadow of *all* of its subordinate block is carried out before the final reduction of equation 2.51 is carried out on any term of the primary block.

A simple indexing and testing method has been incorporated into the algorithm to determine if any of the terms of the primary block lie in the horizontal shadow of the current subordinate block. If no terms of the primary block lie in the shadow of the current subordinate block, a dummy read is performed on the file of reduced blocks to avoid unnecessary disk reads.

Once the partial reduction of all the off-diagonal terms in the primary block that lie in the horizontal shadow of all its subordinate blocks is completed, only the *diagonal* terms of each subordinate block are sequentially brought back in the core in ascending order and stored in section B. (Since the number of diagonal terms of a vector stiffness block will be significantly

less than the total number of terms in that block, an additional block will be established for each vector stiffness block that will contain a copy of only the reduced diagonal terms. This will reduce the number of required disk reads for this section of the algorithm.)

While each block of subordinate diagonal terms is in the core, all the terms that lie in its horizontal shadow for each column of the primary block are fully reduced by equations 2.51 using steps 21 through 27 of the in-core procedure. Again, this procedure proceeds down each column of the primary block and moves from column to column in ascending order when all the terms of the current column that lie in the horizontal shadow of the current subordinate block are fully reduced. In addition, as the terms of the current column of the primary block are fully reduced, the summation in equation 2.52 is simultaneously performed over those terms. When all the terms of the current column of the primary block that lie in the horizontal shadow of the current subordinate block are fully reduced, the value of the summation is subtracted from the diagonal of that column to *partially* evaluate equation 2.52 for that column. This partial summation followed by the subtraction can be done because of the associative nature of the subtraction and additions performed in that equation. When this section of the algorithm is completed, some or all of the diagonal terms of the primary block will have been partially reduced. (Again, if no terms of the primary block lie in the horizontal shadow of the diagonals of the current subordinate block, a dummy read is performed on that file to avoid unnecessary disk reads.)

At this point, all the terms in the primary block that lie in the horizontal shadow of all its subordinate blocks have been fully reduced. (This will include all the off-diagonal terms of the first column of the primary block.) In addition, the diagonal term of the first column of the primary block will be fully reduced and some or all of the remaining diagonal terms will have been partially reduced. However, none of the off-diagonal terms that lie below the horizontal shadow of all the primary block's subordinate blocks have undergone any reduction.

To complete the reduction process for the primary block, the non-reduced off-diagonal terms of the second column of the primary block are reduced using equations 2.49 and 2.50 as outlined in steps 1 through 20 of the in-core procedure. Then the same terms of the second column are fully reduced by equation 2.51. In addition, the diagonal term of the second column is finally fully reduced by simultaneously carrying out the summation in equation 2.52 over the same terms and then subtracting the value from the already partially reduced diagonal term. This same procedure is then carried out column by column in ascending numerical order until the reduction of the primary block is completed.

Once the primary block is completely reduced, it is written to disk for use in the reduction of the subsequent vector stiffness blocks and the load blocks. In addition, the required copy of the reduced diagonal terms are also written to a separate block.

The required disk operations to perform the reduction of the fourth stiffness vector block is schematically illustrated in Figure 2.7.

The subroutine that reduces the already assembled stiffness vector blocks is REDSTF listed in Appendix C. This subroutine is based on work by Mondkar and Powell.^{8,9}

For the reduction of the load vector and the back-substitution to find the degree of freedom displacements, the procedure is more straightforward and is very similar to the in-core solver.

First, the load vector is calculated by the CAEFRAME code and stored in section B of the core area allocated for the stiffness vector. Then the *reduced* vector stiffness blocks are sequentially brought into the core in ascending order and stored in section A of the allocated core area. While each reduced stiffness vector block is in the core, equation 2.54 is evaluated as outlined in steps 1 through 14 of the load reduction section of the in-core procedure. This will partially reduce all the load terms that lie in the horizontal shadow of the current reduced stiffness vector block in the core. (Note, if the horizontal shadow of the current block lies above the first non-zero load term, no reduction by the current block is necessary as outlined in steps 1 and 2 of the in-core procedure. Therefore, a dummy read is performed on that file to avoid unnecessary disk reads.) When this step is completed, all the load terms will be partially reduced and the terms v_i will have replaced the r_i terms in section B of the core.

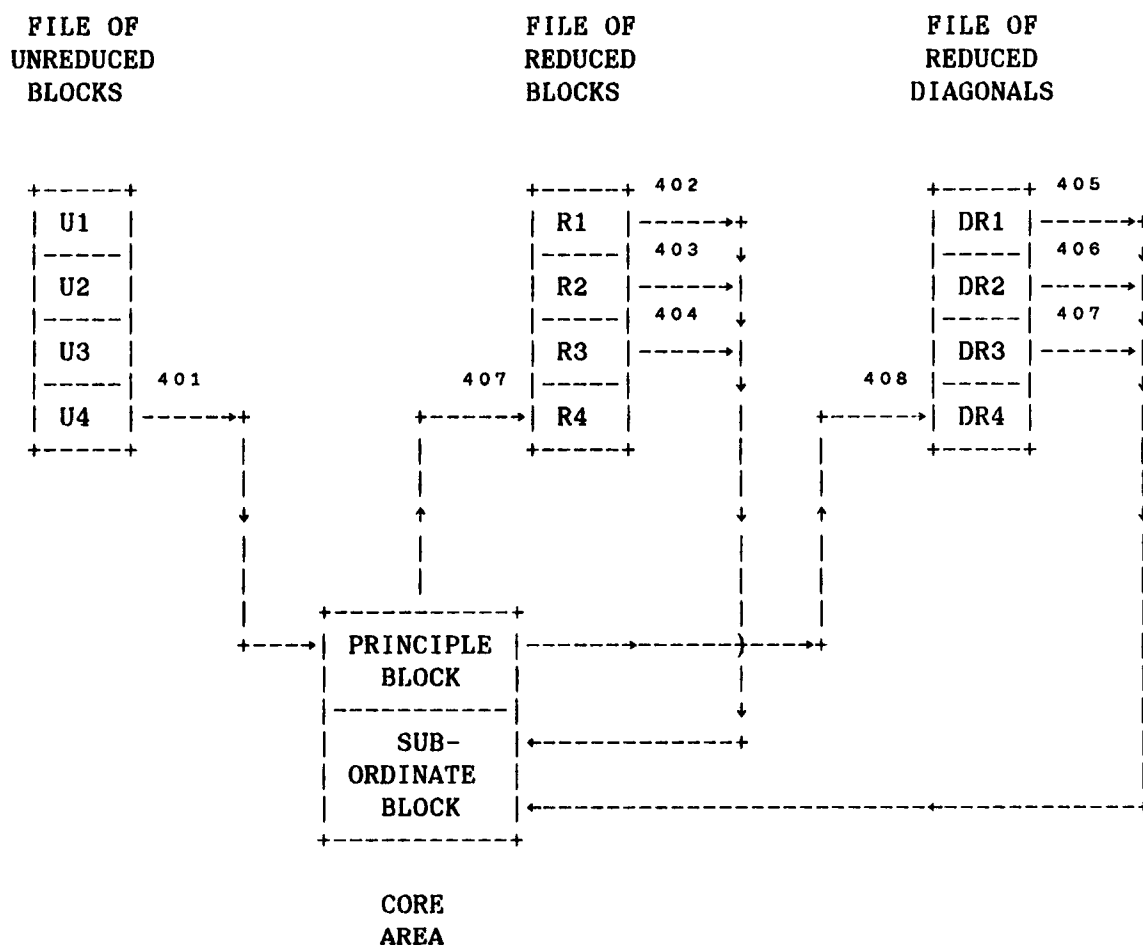


Figure 2.7. Required Disk Operations to Reduce the Fourth Stiffness Vector Block.

Then the blocks of the reduced diagonal terms are sequentially brought into the core in ascending order and stored in section A. While each of these blocks is in the core, equation 2.55 is evaluated for all the load terms in its horizontal shadow. (Again, if the horizontal shadow of the current block is above the first non-zero load term, the block is not needed and a dummy disk read is performed to avoid unnecessary operations.) Once this procedure is completed for all the reduced stiffness vector blocks, the load block will be completely reduced and the $\tilde{v}_1$ terms will now reside in section B of the core.

Lastly, the completely reduced stiffness vector blocks are again brought into the core and stored in section A. However, this time they are brought in via descending order. As each block is brought in the core, equations 2.58 and 2.59 are carried out for the appropriate reduced load vector terms that lie in its horizontal shadow. At the completion of this step, the solved unknown displacements u_j will have been calculated and will have replaced the load vector in section B of the core.

A schematic illustration of the required disk operations to perform the reduction of the load vector and the back-substitution is given in Figure 2.8 for a problem that has four stiffness vector blocks.

The subroutine that performs the reduction of the load vector and the back-substitution to find the degree of freedom displacements is SOLVE and is listed in Appendix C. Again, this subroutine is based on work by Mondkar and Powell.^{8,9}

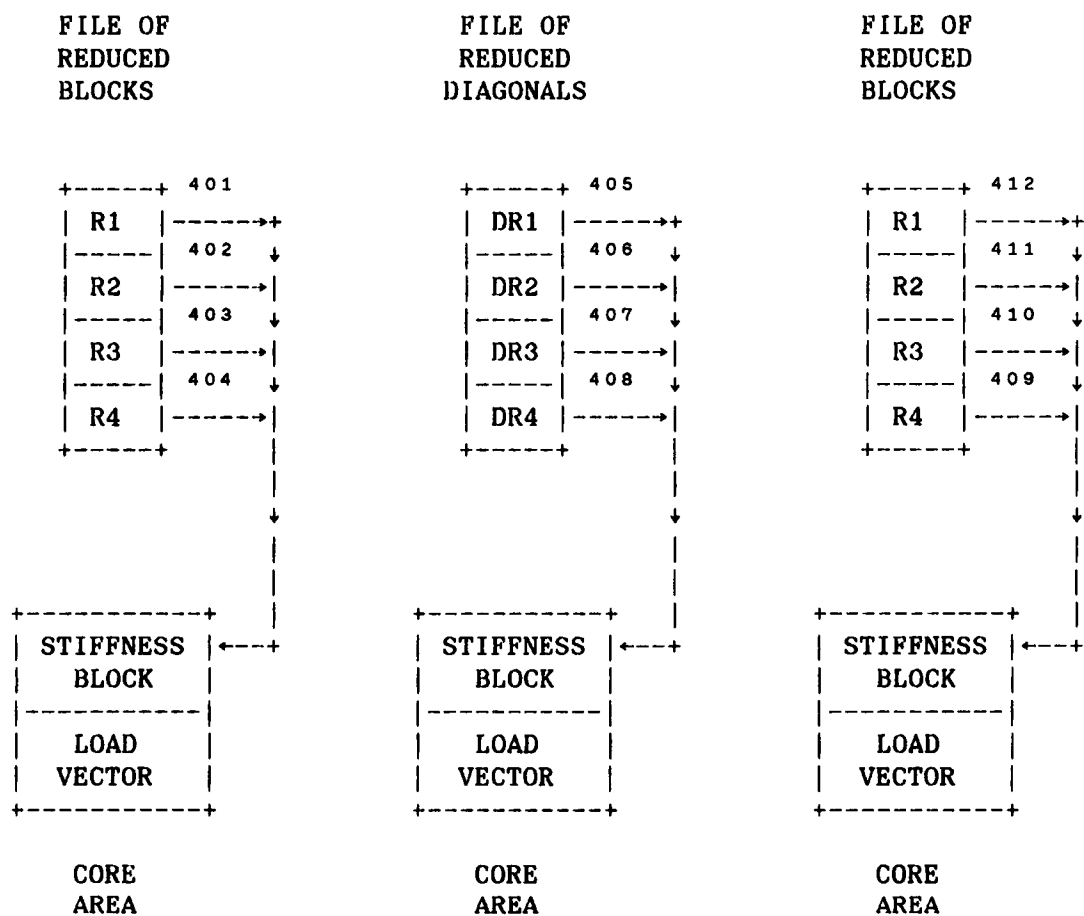


Figure 2.8. Required Disk Operations to Reduce the Load Vector and Back-substitute for the Unknowns.

Resolution Capabilities

By examining the procedure for the reduction of the load vector and the back-substitution of the unknowns, it is easy to see that the solution of a second or any other subsequent load case is relatively simple.

Since the reduced stiffness vector blocks along with their associated diagonal address and diagonal term blocks will still remain on disk, the procedure to solve any other loadcase is to simply put the new load vector into section B of the core and then follow the same procedure as outlined for the solution of the first load vector. This can be repeated for as many loadcases as desired.

Bandwidth Reduction

The way to increase the efficiency of both the in-core and out-of-core version of the modified Cholesky algorithm utilizing the compacted column storage scheme, is to decrease the number of terms below the skyline of the matrix. For example, assume a structure is composed of four plate elements attached end to end with each plate element having four nodes as shown in Figure 2.9.

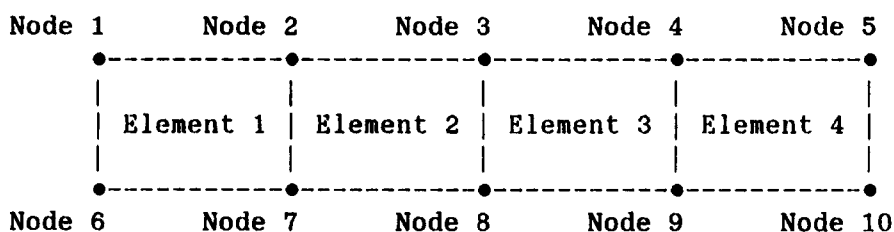


Figure 2.9. Example One of a Four Plate Model.

If only one degree of freedom per node is assumed, the resultant upper half of the stiffness matrix would be as shown in Figure 2.10. The stiffness matrix is again illustrated as a two dimensional matrix. But as shown previously in the algorithm, the stiffness matrix is actually stored as a one dimensional vector. The stiffness vector of Figure 2.10 would require 46 storage areas. Also, the same 46 terms stored in the stiffness vector would also have to be fully reduced by equations 2.48 through 2.52 in order to solve the problem for any loadcase.

However, if the nodes of the model are renumbered as shown in Figure 2.11, and the same assumption of one degree of freedom per node is made, the resultant upper half of the stiffness matrix would be as shown in Figure 2.12. When this matrix is stored in compacted vector form, only 31 terms require storage and reduction. This reduction in the number of stored stiffness terms reduces the number of required operations and the runtime of the solution. In addition, for the out-of-core version of the algorithm, this node numbering reduces the amount of required disk space for storage of both the unreduced and reduced stiffness vector blocks. This savings in execution time and disk storage area is significant for this small model. However, the savings will increase and become more important as the size of the model becomes larger and the number of degrees of freedom per node is increased.

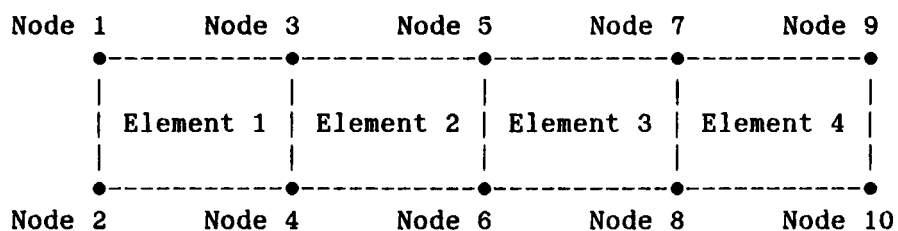


Figure 2.11 Example Two of a Four Plate Model.

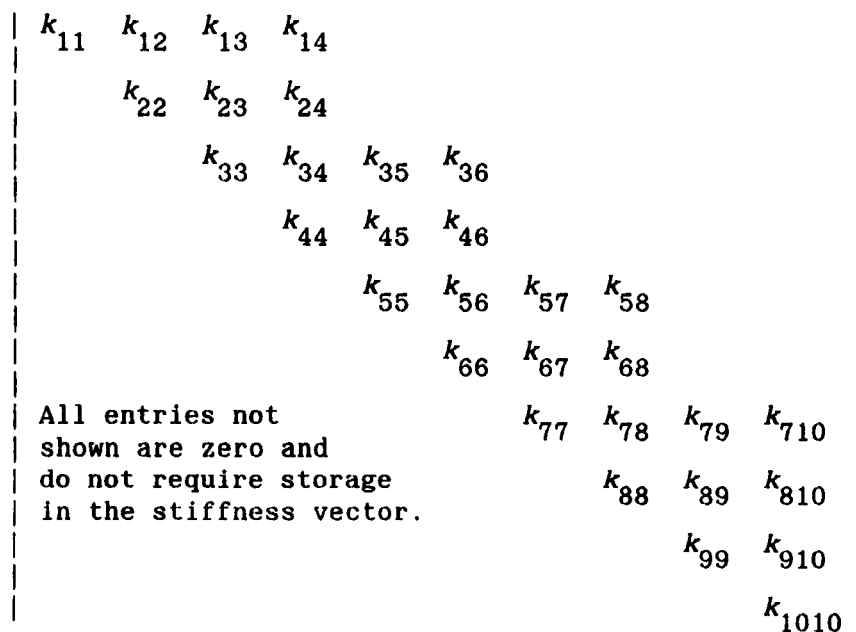


Figure 2.12 Stiffness Matrix of Numbering Scheme Two of the Four Plate Model.

The problem with the general use of the finite element code is that not all users are experienced enough to efficiently number a model to keep the number of stiffness terms below the skyline to a minimum. Also, even if the user is experienced, for some models it is not obvious which way to number the nodes for maximum efficiency and considerable time could be spent to determine that numbering. In addition, as changes are made to the model for re-analysis, additions and/or deletions of nodes and elements may result. These additions and/or deletions may alter the originally required numbering of the nodes that produced an optimized model, and a new number scheme may be required.

Therefore, a method that would automatically renumber the nodes of the model to produce the minimum number of stiffness terms to be stored and reduced is desired. Ideally, besides producing the optimum node numbering, the chosen method should be fast and should be transparent to the user. By being transparent, the renumbering scheme should revert back to the original nodal numbering system developed by the user when the answers are outputted. This is done in order to avoid confusion to the user.

In an attempt to achieve these goals, an algorithm developed by Collins¹⁴ was chosen by the author to check the nodal numbering of a model and renumber the nodes if a non-optimized model was incurred. As stated in the introduction, even though this particular algorithm does not necessarily optimize a non-optimized model, it was chosen because it was relatively easy to code and it is very fast.

The basic approach is to go through all the nodes of the model and try a numbering system with the current node as the first node number of the new numbering scheme. In this way there will be as many attempts to renumber the system as there are nodes in the system.

The parameter that the algorithm focuses on to determine if one numbering system is better than another numbering system is the *maximum half-bandwidth* of the structure. The half-bandwidth for any column of the stiffness matrix is the height of that column through the skyline term. Therefore, the maximum half-bandwidth of the system is the column with the maximum height. If Figure 2.10 is examined again, the size of the column with the maximum height is seven, and this maximum height occurs in columns six through ten. However, for Figure 2.12, the size of the column with the maximum height is four, and this maximum height occurs in columns four, six, eight, and ten. Thus, for the one degree of freedom models in Figures 2.9 and 2.11, the maximum half-bandwidth corresponds to the maximum difference between any two connected nodes plus one, and the system with the smallest maximum half-bandwidth will have fewer stiffness terms to store and reduce.

Even though in the CAEFRAME code a node can have up to six degrees of freedom, the nodal number difference between any two connected nodes can still be used as an indication of the half-bandwidth of a column since the column heights will still be proportional to the nodal numbering difference in a ratio equal to the number of degrees of freedom per node. Therefore, the actual

parameter that the reduction algorithm focuses on is the maximum difference between any two connected nodes.

The details of the procedure begin by first setting up two connectivity arrays.

The first or *elemental* connectivity array establishes which nodes are attached to each element. For the original four plate model given in Figure 2.9, this connectivity array would appear as in Table 2.1.

The second or *nodal* connectivity array is established by scanning the elemental connectivity array and determining which other nodes are attached to each node of the model under the original numbering scheme. Again, for the original four plate model of Figure 2.9, this array would appear as in Table 2.2.

Once both of these connectivity arrays are established, the nodal array is scanned to determine the maximum difference between any two connected nodes of the original numbering system and this value is stored in the index *minmax*. Then, the algorithm loops through all the nodes of the model in the order that they were originally numbered and tries a new numbering system based at each node. If a new numbering system is found that has a smaller maximum difference between any two connected nodes, the value of *maxmin* is set to the new difference, and a global array is established that maps the number of every node in the new system to its old number in the original system.

Table 2.1. Elemental Connectivity Array for Figure 2.9.

Element No.	Node 1	Node 2	Node 3	Node 4
1	1	2	7	6
2	2	3	8	7
3	3	4	9	8
4	4	5	10	9

Table 2.2. Nodal Connectivity Array for Figure 2.9.

Node No.	List of Connected Nodes
1	2, 7, 6
2	1, 7, 6, 3, 8
3	2, 8, 7, 4, 9
4	3, 9, 8, 5, 10
5	4, 9, 10
6	1, 2, 7
7	1, 2, 6, 3, 8
9	3, 4, 8, 5, 10
10	4, 5, 9

To illustrate the details of this process, assume that nodes 1 through $j-1$ have been examined and the algorithm is attempting a new numbering system based at node j .

First, a temporary map array is established that relates the new number of every node for this attempt at renumbering to its old node number under the original numbering system. In this temporary map array, the algorithm firsts renumbers node j as node number one. Then, all the nodes under the original numbering system that are attached to the new node number one are obtained from the nodal connectivity array. The algorithm then renumbers these connected nodes in ascending numerical order in the sequence that they are read from the nodal connectivity array. These new node numbers are then stored in the temporary map array.

When all the nodes that are attached to the new node number one are renumbered, the maximum difference between the new node number one and any of the nodes attached to the new node number one is checked to see if the difference is less than the current value of *maxmin*. If this difference is less than the current value of *maxmin*, the algorithm proceeds with this attempt at renumbering. If the difference is greater than the value of *maxmin*, this renumbering scheme is abandoned and the algorithm moves to the next node of the original system to attempt the next renumbering scheme.

If the algorithm is continuing with this attempt to renumber the nodes, the new node number two is related to its old number under the original numbering system using the temporary map. The nodal connectivity array for the original number is examined to

determine which nodes under the original numbering system are attached to the new node number two. As the nodes that are attached to the new node number two are examined in sequential order, a check is made using the temporary nodal map to see if it has been renumbered under the new scheme. If it has, the node is skipped. If it has not been renumbered under the new system, the next available node number under the new scheme is assigned to that node.

Again, once all the nodes attached to the new node two that have not been previously renumbered are renumbered, the algorithm checks for the maximum difference between the new node number two and all the new node numbers of all the nodes connected to it. The same conditions as the check with the new node number one apply again. Namely, if the node difference is greater than or equal to the current value of *maxmin*, the current attempt at renumbering is abandoned. If the difference is less than the value of *maxmin*, the algorithm moves to the new node number three and the process is repeated.

This particular attempt at renumbering is continued until either it is abandoned because the difference between any two connected nodes equals or exceeds the value of *minmax*, or all of the nodes have been renumbered. If all the nodes have been renumbered, then the current renumbering scheme is better than any previously attempted schemes. Therefore, the value of *maxmin* is reset to the value of the maximum difference of any two connected nodes for this renumbering scheme, and the map stored in the temporary array is copied to the global array.

The algorithm then moves to the next node of the original numbering system and repeats the entire process of attempting a renumbering of the system with this next node as the new node number one.

Once all attempts at renumbering have been made, the value of *maxmin* will have the value of the difference between any two connected nodes for the chosen numbering system, and the global map will contain the relationship of the original node numbers to the new node numbers under the chosen scheme.

The global map will be used to correlate the information provided by the original CAEFRAME code in establishing the stiffness and load vectors, as well as converting information on degree of freedom displacements for use by the output sections of the code. Therefore, transparency to the user of the renumbering process will be maintained.

The subroutine that performs the renumbering of the nodes in an attempt to reduce the maximum half-bandwidth of the model is RENUM and is listed in Appendix C.

Modifications and Improvements to the Algorithms

Even though the algorithms for both the blocked out-of-core solver and the bandwidth reduction technique are based on previously published work, some modifications to the original work were required in order for the algorithms to execute properly on the CAEFRAME code.

As stated earlier, the first modification that was required was for the subroutines that assemble the stiffness vectors. The original subroutines were removed from the code and new ones written by the author were substituted. However, the basics of these new subroutines that assemble the vector stiffness blocks were taken from the original in-core subroutines of the CAEFRAME code. But considerable modifications, mainly in the form of increased indexing, had to be done to subroutines BLKSTF, ASSTIF, and STIFFS in order for them to assemble the correct stiffness terms in the correct blocks. The subroutine STFOUT, which controls the entire assembly process, was written from scratch by the author.

For the subroutines REDSTF, which reduces the assembled stiffness vector blocks, and SOLVE, which reduces the load vectors and back-substitutes for the unknown displacements, very few functional modifications from the original work were made. However, both subroutines were modified to take on a more structured approach by substituting "do continue", "if then" and "if then else" statements for most of the "goto" statements. These changes make the programs easier to understand and they should make the subroutines use the available core space more efficiently. In addition, considerable more use of both mnemonic variable names and comment cards was employed in the author's version of these subroutines. These changes did not improve the efficiency of the subroutines, but they did make them easier to understand by other readers. Since these subroutines may be incorporated into the

CAEFRAME code, it is important that the engineers at SST Inc. understand the flow and logic of the subroutines.

For the bandwidth reduction technique incorporated into the subroutine RENUM, the only changes to the original work made by the author were those that were required to interface the algorithm with the rest of the program. These changes included such items as correct addresses of variable locations, proper array allocation, and proper subroutine calls.

III. THE FRONTAL SOLVER

As shown with the Blocked Out-Of-Core Solver, one approach to solve the set of simultaneous equations generated in linear static finite element analysis is to separately assemble then reduce the complete global stiffness matrix, which is followed by the complete assembly then reduction of the global load vector, and finally, back-substitution is performed to find the unknown displacements. An alternative approach is to assemble and reduce the set of simultaneous equations composed of the global stiffness matrix and the global load vector at the same time, and then perform the back-substitution. One type of solution algorithm that takes this approach is called the *frontal* solution algorithm.

The frontal algorithm was originated by Irons.^{11,12} Hinton and Owen¹¹ later modified Irons' algorithm to take a more modular approach. Since the CAEFRAME code is modularly structured, the later version by Hinton and Owen is the one that the author used as the basis for implementing the frontal solver into the CAEFRAME code.

In this section, both a brief overview and a detailed explanation of the frontal algorithm will be presented first. This will be followed by a discussion of the modifications to the general equations, along with the required data storage and handling techniques, that allow computer implementation of the algorithm. Finally, the theoretical justification and computer implementation of the particular frontwidth reduction technique chosen by the author to increase the efficiency of the algorithm will be presented.

Overview of the Frontal Algorithm

In the frontal algorithm, the basic approach is to sweep through the model on an elemental basis. As each element is considered, the contributions due to all of its attached degrees of freedom are assembled into the global set of equations currently stored in the core. This part of the algorithm differs from the assembly process used for the modified Cholesky method because the load vector is also being formed at this time.

Since the stiffness and load terms are being assembled at the same time for the frontal method, when an equation for a degree of freedom is completely assembled from the contributions of all relevant elements, the storage of that equation in-core becomes redundant. This is because no more contributions will be added to the stiffness or load terms of that equation. Therefore, this redundant equation could be eliminated from the set of equations residing in the core without affecting the assembly of the rest of the set of equations.

To accomplish the elimination of a redundant equation from the set, the equation is first written to disk storage. Then, through static condensation, the corresponding degree of freedom is removed from the global set of equations currently stored in-core. The in-core storage space used by the eliminated equation can now be used for an equation of a degree of freedom that has not yet started forming. In this way, at any given instant, the core only contains the degree of freedom equations which are still

being formed. In general, this is a small percentage of the total number of equations needed for the solution of the problem.

The equations still in the core at any given point in the algorithm, along with their corresponding nodes and degrees of freedom, are termed the *front*. The number of unknowns currently in the front is the *frontwidth*. (This number generally changes continuously during the assembly/reduction process.) The maximum number of equations that can reside in the core at any point is the *maximum frontwidth*. The equations, nodes and degrees of freedom that are currently in the front are termed *active*; those that have not yet been considered are termed *inactive*; those that have passed through the front and have been eliminated through static condensation are termed *deactivated*.

Development of the Frontal Algorithm

During the frontal process, elements are considered one at a time in a prescribed order. When a new element is considered, all the degrees of freedom of that element are examined to determine their status.

If this is the first appearance of a degree of freedom, the element stiffness contributions for the degree of freedom are stored in any vacant areas of the core allocated for the stiffness vector. Then, the total load term for the degree of freedom is entered into any vacant area of the core allocated for the load vector. The contributions of both the just entered stiffness terms and the just

entered load term form the basis for the global equation of this newly *activated* degree of freedom.

If a degree of freedom is already *active*, the element stiffness contributions for the degrees of freedom are summed into the correct stiffness terms of the already active equation for that degree of freedom. Since the total load term was entered into the load vector when this already active degree of freedom was first activated, nothing is done to the load vector at this point.

After all the degrees of freedom for the current element have been examined, and their stiffness and load contributions are assembled into the global set of equations by the rules stated above, they are each examined again to determine if this is the last appearance of any of them.

If a degree of freedom is appearing for the last time, then, the equation for the degree of freedom is written to disk storage and the degree of freedom is *deactivated* by statically condensing it out of the global equation set. (The actual process of static condensation will be addressed later in this section.) This reduces the amount of information stored in-core, thus providing free space for *inactive* degrees of freedom that will be considered as the rest of the elements pass through the front. This elimination process is repeated for each degree of freedom of the current element that is appearing for the last time in the assembly process.

This assembly/reduction process is continued until all the elements have passed through the front. After the last element has passed through the front, the core will be empty.

To solve for the displacements, the equations that were previously written to disk storage are brought back into the core for back-substitution. However, the order that these equations are brought back into the core is the reverse of the order in which they were written to disk storage. That is, the last equation written to disk storage is the first equation brought back into the core.

Since this last equation has only one unknown in it, (namely the last *deactivated* degree of freedom), it can be solved directly for that degree of freedom.

Then, the second to the last equation written to disk storage is brought back into the core. This equation only has two degrees of freedom in it, namely the last *deactivated* degree of freedom and the second to the last *deactivated* degree of freedom. But the last *deactivated* degree of freedom has already been found. Therefore, this equation can then be solved directly for the second to the last *deactivated* degree of freedom using back-substitution.

This process of bringing the eliminated equations back into the core and then performing back-substitution of the already solved for degrees of freedom is continued until the displacements for all the degrees of freedom have been found.

The element mesh given in Figure 3.1 will be used to illustrate the general procedure. This is an adaptation and expansion of an example given by Hinton and Owen.¹¹ To make the illustration as simple and useful as possible, the nodes will be restricted to only move translationally in the U direction.

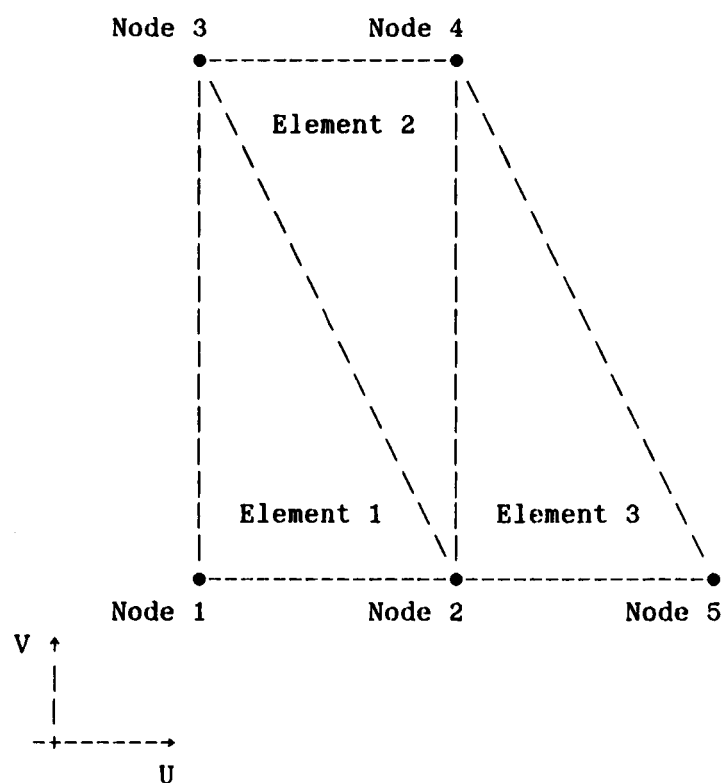


Figure 3.1. Mesh Used for Illustration of the Frontal Method.

Since element 1 is the first element to be considered, the global stiffness matrix and global load vector will be the same as the element stiffness matrix and element load vector. The static equilibrium equations stored in-core are:

$$\begin{vmatrix} k_{11}^1 & k_{12}^1 & k_{13}^1 \\ k_{21}^1 & k_{22}^1 & k_{23}^1 \\ k_{31}^1 & k_{32}^1 & k_{33}^1 \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} r_1 \\ r_2 \\ r_3 \end{vmatrix} \quad (3.1)$$

After re-examination of the mesh, it is determined that node 1 does not appear in any other element. Therefore, all the contributions to the global equation for the degree of freedom at node 1 have been made, and this degree of freedom can then be *deactivated*.

To *deactivate* the degree of freedom at node 1, the equation in row 1 of equation set 3.1 is first written to disk storage. (Later when u_2 and u_3 are found, the equation will be read back from disk storage and solved for u_1 .) Once the equation is written to disk storage, the degree of freedom at node 1 is statically condensed out of the global set of equations that will remain in the core.

The procedure used to statically condense a degree of freedom out of the global set of equilibrium equations is different for free degrees of freedom and stationary degrees of freedom.

For free degrees of freedom, the equations that will statically condense a degree of freedom out of a set of simultaneous equations are given by Irons.¹³ They are:

$$k'_{ij} = k_{ij} - \frac{k_{is}}{k_{ss}} * k_{sj} \quad (3.2a)$$

$$r'_i = r_i - \frac{k_{is}}{k_{ss}} * r_s \quad (3.2b)$$

where the symbol (') indicates terms in the modified set of equations remaining in the core, and the subscript *s* is the index matching the number of the degree of freedom being statically condensed out of the global set. This index number may or may not be the same as the row or column position number of the degree of freedom being deactivated. These equations are used to reduce all the terms in the global set of equations stored in the core up through the current *frontwidth* except for those terms with index numbers corresponding to the degree of freedom number that is being eliminated. After all the terms of both the stiffness matrix and the load vector remaining in-core are reduced by the equations in 3.2, the spaces in the core occupied by the terms of the equation for the eliminated degree of freedom are set to zero.

For stationary degrees of freedom, no reduction of either the stiffness or load terms remaining in-core is required. Only the spaces occupied by the terms of the equation for the eliminated degree of freedom need to be set to zero.

For the illustrative example, assume node 1 is free. Since the equation in row one of the equation set 3.1 has already been written to disk storage, the next step is to statically condense

the degree of freedom at node 1 out of the global set of equations.

The resulting global set of equations remaining in-core is:

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & k'_{22} & k'_{23} \\ 0 & k'_{32} & k'_{33} \end{vmatrix} \begin{vmatrix} u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} 0 \\ r'_2 \\ r'_3 \end{vmatrix} \quad (3.3)$$

where :

$$k'_{22} = k_{22}^1 - \frac{k_{21}^1}{k_{11}^1} * k_{12}^1 ; k'_{23} = k_{23}^1 - \frac{k_{21}^1}{k_{11}^1} * k_{13}^1 ; r'_2 = r_2 - \frac{k_{21}^1}{k_{11}^1} * r_1$$

$$k'_{32} = k_{32}^1 - \frac{k_{31}^1}{k_{11}^1} * k_{12}^1 ; k'_{33} = k_{33}^1 - \frac{k_{31}^1}{k_{11}^1} * k_{13}^1 ; r'_3 = r_3 - \frac{k_{31}^1}{k_{11}^1} * r_1$$

When element 2 is brought into the front, it has contributions for the already *active* nodes 2 and 3 and the newly *activated* node 4. The contributions to the stiffness terms for the equations of the degrees of freedom at nodes 2 and 3 are simply summed into the existing equations, while both the stiffness and load terms that form the equation for the degree of freedom at node 4 can go into the empty spaces provided when the equation for the degree of freedom at node 1 was *deactivated*. The current set of global equations that are stored in-core is:

$$\begin{vmatrix} k_{44}^2 & k_{42}^2 & k_{43}^2 \\ k_{24}^2 & k_{22}^2 & k_{23}^2 \\ k_{34}^2 & k_{32}^2 & k_{33}^2 \end{vmatrix} \begin{vmatrix} u_4 \\ u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} r_4 \\ r_2 \\ r_3 \end{vmatrix} \quad (3.4)$$

where:

$$\begin{aligned} \hat{k}_{22} &= k_{22}^1 + k_{22}^2 & ; & & \hat{k}_{23} &= k_{23}^1 + k_{23}^2 \\ \hat{k}_{32} &= k_{32}^1 + k_{32}^2 & ; & & \hat{k}_{33} &= k_{33}^1 + k_{33}^2 \end{aligned}$$

Since node 3 is not connected to any other element, its equation can be written to disk storage and the degree of freedom statically condensed out of the global set remaining in-core. This time assume that node 3 is a fixed node and, therefore, the only required in-core operation is to set the locations occupied by the equation to zero. After this procedure is completed, the global set of in-core equations is:

$$\begin{vmatrix} k_{44}^2 & k_{42}^2 & 0 \\ k_{24}^2 & \hat{k}_{22} & 0 \\ 0 & 0 & 0 \end{vmatrix} \begin{vmatrix} u_4 \\ u_2 \\ 0 \end{vmatrix} = \begin{vmatrix} r_4 \\ r_2 \\ 0 \end{vmatrix} \quad (3.5)$$

Then element 3 is brought into the front. This element will have contributions for the already *active* nodes 2 and 4 and will also have both stiffness and load terms to form the equation for the newly *activated* node 5. The equation for the degree of freedom at node 5 can go into the spaces left vacant by the elimination of the equation for the degree of freedom at node 3. The resulting new set of in-core global equations is:

$$\begin{vmatrix} k_{44}'' & k_{42}'' & k_{45} \\ k_{24}'' & k_{22}'' & k_{25} \\ k_{54} & k_{52} & k_{55} \end{vmatrix} \begin{vmatrix} u_4 \\ u_2 \\ u_5 \end{vmatrix} = \begin{vmatrix} r_4 \\ r_2 \\ r_5 \end{vmatrix} \quad (3.6)$$

where:

$$k_{22}'' = \hat{k}_{22} + k_{22}^3 ; \quad k_{24}'' = \hat{k}_{24} + k_{24}^3$$

$$k_{42}'' = \hat{k}_{42} + k_{42}^3 ; \quad k_{44}'' = \hat{k}_{44} + k_{44}^3$$

At this point all the elements have passed through the front and the assembly portion of the algorithm is complete. However, the process of writing an equation to disk storage and then eliminating its corresponding degree of freedom through static condensation is continued. For this example, assume all the rest of the degrees of freedom are free.

The order in which the remaining degrees of freedom are eliminated is arbitrary. For the example, first condense out the degree of freedom at node 2. Write the equation in row two of equation set 3.6 to disk storage and then reduce the other equations stored in-core. The resulting set stored in-core is:

$$\begin{vmatrix} k_{44}^{\$} & 0 & k_{45}^{\$} \\ 0 & 0 & 0 \\ k_{54}^{\$} & 0 & k_{55}^{\$} \end{vmatrix} \begin{vmatrix} u_4 \\ 0 \\ u_5 \end{vmatrix} = \begin{vmatrix} r_4^{\$} \\ 0 \\ r_5^{\$} \end{vmatrix} \quad (3.7)$$

where :

$$k_{44}^{\$} = k_{44}'' - \frac{k_{42}''}{k_{22}''} * k_{24}'' ; \quad k_{45}^{\$} = k_{45}'' - \frac{k_{42}''}{k_{22}''} * k_{25}'' ; \quad r_4^{\$} = r_4'' - \frac{k_{42}''}{k_{22}''} * r_2''$$

$$k_{54}^{\$} = k_{54}'' - \frac{k_{52}''}{k_{22}''} * k_{24}'' ; \quad k_{55}^{\$} = k_{55}'' - \frac{k_{52}''}{k_{22}''} * k_{25}'' ; \quad r_5^{\$} = r_5'' - \frac{k_{52}''}{k_{22}''} * r_2''$$

Next, the degree of freedom at node 4 will be eliminated. The equation in row one of equation set 3.7 is written to disk storage and the degree of freedom at node 4 is statically condensed out of the global set remaining in-core. The result is the following equation set left in-core:

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & k_{55}^* \\ \hline \end{array} \begin{array}{|c|} \hline u_5 \\ \hline \end{array} = \begin{array}{|c|} \hline 0 \\ \hline 0 \\ \hline r_5^* \\ \hline \end{array} \quad (3.8)$$

where :

$$k_{55}^* = k_{55}^{\$} - \frac{k_{54}^{\$}}{k_{44}^{\$}} * k_{45}^{\$} ; \quad r_5^* = r_5^{\$} - \frac{k_{54}^{\$}}{k_{44}^{\$}} * r_4^{\$}$$

The equation in row 3 of equation set 3.8 is then put on disk storage and all the terms in the core are set to zero.

From the equation in row 3 of equation set 3.8, the displacement of node 5 can be solved directly. Therefore, this equation is brought back into the core and the displacement at node 5 is found.

$$u_5 = \frac{r_5^*}{k_{55}^*} \quad (3.9)$$

After the displacement at node 5 is found, the equation in row 1 of the equation set 3.7 (which was written to disk storage earlier) is brought back into the core. This equation can then be used to find the displacement at node 4 in the same manner that equation 3.9 was used to find u_5 . Then the equation in row 2 of

equation set 3.6 is brought into the core and solved for the displacement at node 2.

This process of bringing the eliminated equations back into the core and then back-substituting, is continued until all the displacements at all the nodes have been found.

Computer Implementation

The frontal solution algorithm as implemented into the CAEFRAME code is presented in the subroutine FRONT in Appendix D. In addition to this solution subroutine, there are some supporting subroutines also listed in Appendix D that are required for the proper integration of the frontal algorithm into the CAEFRAME code. These supporting subroutines will be addressed as they arise in the discussion.

In the computer implementation of the frontal algorithm, (as with the blocked out-of-core solver), the stiffness matrix is again actually stored in vector form and not in matrix form as the example illustrated. Also again, since the stiffness matrix is symmetric, only the terms on and above the diagonal require storage. So, like the compacted column storage scheme used for the blocked out-of-core solver, the stiffness matrix is stored with the columns of the upper triangle strung together one after the next. But, in this storage scheme, the entire column is stored and not just the terms below the skyline. However, only the columns up through the current *frontwidth* are used at any point in the algorithm and all the

following columns are ignored except for future expansion of the *front*.

As seen in the example, any degree of freedom can be put in any vacant location in the front. Therefore, the indices of the stiffness and load terms actually correspond to the degree of freedom numbers of the degrees of freedom that these terms connect. These index numbers may or may not correspond to the row and column position numbers of the two dimensional matrix representation of the equations as they did with the blocked out-of-core solver. This lack of degree of freedom number correlation to matrix position implies that a considerable amount of both pre-solution and runtime index housekeeping will be required in order keep track of which degree of freedom stiffness and load terms are stored in which global stiffness and load vector positions.

Step 1: The first pre-solution housekeeping chore is to determine the order in which the elements are to be brought into the front. As will be shown later in this section, the order in which the elements pass through the front will constantly effect the instantaneous frontwidth. For each step in the assembly/reduction portion of the algorithm, the larger the instantaneous frontwidth, the slower the algorithm proceeds. Therefore, for maximum efficiency it is desirable to keep the instantaneous frontwidth as small as possible. However, at this point, no efficiency improvement is assumed, so the elements are to be brought into the front according to ascending numerical order.

(This will process the elements as per the dictates of the analyst using the program).

But, the CAEFRAME code does not always store the elements in ascending numerical order. Rather, it stores the elements by type. That is, all the truss elements are stored first, followed by all the beam elements, and finally all the plate elements. (Since the original CAEFRAME program used the in-core version of the modified Cholesky algorithm, and as seen in Section II, the nodal ordering and not the element ordering dictated the efficient use of that algorithm, the above element ordering scheme was used for ease of programming.) Therefore, if there is a mix of different element types, the element numbers will have to be sorted so that they are called in the correct ascending numerical order. This sorting is done in the subroutine ORDNRE listed in Appendix D.

Step 2: Subroutine PREFRT also transfers any spring stiffnesses in the problem from a node number reference used by the original CAEFRAME program to a direct degree of freedom number reference. This is necessary for the proper addition of the spring stiffnesses to the global stiffness vector in the frontal algorithm.

Step 3: The next pre-solution chore is to set up an index flag that will indicate when the last appearance of a degree of freedom occurs. The index that will be used is the elemental degree of freedom connectivity array already provided by another section of the CAEFRAME code. This connectivity array lists all the degrees of freedom that are attached to each element. The connectivity array of element degrees of freedom is scanned in the order in which

the elements will be brought into the front. When the element that has the last appearance of a degree of freedom is found, the sign of that degree of freedom number in the connectivity array for that last appearance element is changed to a negative. This negative sign will serve as the flag in the assembly/reduction portion of the algorithm to indicate that the degree of freedom can be statically condensed out of the in-core equation set. Since this connectivity array is used by other sections of the CAEFRAME code after the solution phase is finished, a working copy of the original connectivity is actually made and used by the subroutines of the frontal algorithm.

When all the preliminary housekeeping is finished, the assembly/reduction portion of the algorithm is executed. The program will loop over all the elements in the prescribed order determined by the subroutine ORDNRE, and individually bring each element into the front one at a time.

Step 4: When an element is brought into the front, the first task is to determine its type. When the type of the element is determined, the proper CAEFRAME subroutines are called to calculate the element's stiffness matrix and then transfer that matrix from local coordinates to global coordinates. All of the global element stiffness terms are then stored in a temporary buffer area.

Step 5: Using the elemental connectivity array, all the element's degrees of freedom numbers are compared, one at a time, against an existing vector list of *active* degrees of freedom to determine if any of the current element's degrees of freedom are

already in the front. (The signs of the element's degree of freedom numbers are ignored at this point, since the first concern is assembly, not reduction.)

Step 6: If one of the current element's degrees of freedom is already *active*, the current front is scanned to determine the position in the front where that degree of freedom is located. The location in the front for this already *active* degree of freedom is then stored in a buffer area called the destination vector and the program skips to step 9. If an element's degree of freedom is not already active, it has to be added to the front and the program continues at this point.

Step 7: To add a degree of freedom to the front, the current front is first scanned for any vacant locations. If a vacant location is found, the position in the front where the newly activated degree of freedom will go is stored in the destination vector, and the newly activated degree of freedom number is added to the vector list of active degrees of freedom. If no vacant position is found within the current front, the newly activated degree of freedom is put in the first location after the end of the current front and the frontwidth is expanded to include it. Even for this case, the degree's of freedom position in the front is stored in the destination vector and the degree of freedom number is added to the vector list of active degrees of freedom.

Step 8: For this newly activated degree of freedom, its total load calculated earlier by the CAEFAME code is put into the frontal load vector. Also, if there is any spring stiffness for this degree

of freedom, the value is added to the correct diagonal term in the element stiffness matrix stored in the temporary buffer.

Step 9: The program goes back to step 5 and incorporates the next degree of freedom of the current element into the front. When this procedure is finished for all of the degrees of freedom of the current element, the destination vector will have the frontal location for every degree of freedom of the current element.

Step 10: Next, all the stiffness terms in the upper triangular portion of the element stiffness matrix that are stored in the temporary buffer have to be added to the correct terms in the global stiffness vector. To accomplish this task, each element stiffness term is examined one at a time to determine which two degrees of freedom the stiffness term connects. (The two degrees of freedom will be the same number if this is a diagonal term, but the procedure is the same). Then the column locations in the front, where each of these two degrees of freedom are connected by the current elemental stiffness term, are obtained from the destination vector. Using these two location numbers as indices, the correct term in the global stiffness vector that connects these same two degrees of freedom is found. (The existing value at this location in the stiffness vector will be zero if the current element is the first element to connect the given two degrees of freedom.) Then the current element's stiffness contribution is taken from the temporary buffer and added to the just found global term. This is repeated for every stiffness term in the temporary buffer.

At this point, the assembly process for the current element is finished, and the set of in-core equations may be reduced if it is the last appearance of any of the degrees of freedom of the current element.

Step 11: To ascertain if any reduction is possible, all the positions in the destination vector are examined, one at a time, to see if any of the signs are negative. If a negative sign is encountered, the corresponding degree of freedom is immediately deactivated by the following procedure.

Step 12: To deactivate a degree of freedom, all of the coefficients of the equation relating the degree of freedom being *deactivated* to all the positions within the current front (whether or not this location is actually occupied by an *active* degree of freedom) are pulled out of the global stiffness vector and copied into a buffer. This is the same as copying the entire column of the stiffness matrix that has the about to be deactivated degree of freedom number as the double indices of the diagonal term. The same degree's of freedom load term is also pulled from the load vector and copied into another temporary storage buffer. (These stiffness and load terms may have undergone some reduction at this point due to the previous elimination of *deactivated* degrees of freedom.) Then the spaces in the stiffness vector where the equation coefficients were pulled from, and the space in the load vector where the load term was pulled from, are all set to zero to provide room for an equation of an *inactive* degree of freedom.

Step 13: Then the following is written to disk: the number of the degree of freedom being *deactivated*, its position in the current *front*, the current *frontwidth*, the degree's load term, and the buffer containing the coefficients connecting the *deactivated* degree of freedom to all the other positions in the current *front* (the number of terms in this buffer will be the same as the current *frontwidth*).

Step 14: The degree of freedom being eliminated is examined next to see if it is free or restrained.

Step 15: If the degree of freedom is restrained, no operations are performed on either the stiffness or load vector and the program skips to step 17.

Step 16: If the degree of freedom is free, both the stiffness vector terms and the load vector terms for all the degrees of freedom currently in the *front* are reduced according to equations 3.2a and 3.2b, respectively.

Step 17: The eliminated degree of freedom is then removed from the vector list of *active* degrees of freedom and the program goes back to step 11 until all degrees of freedom that can be *deactivated* are *deactivated*.

Step 18: After all the degrees of freedom of the current element that can be *deactivated* at the present time are *deactivated*, the last location of the current *front* is examined to see if it is now empty. If it is empty, the current *frontwidth* is decreased by one location. This step is repeated until the first non-empty

location in the *front* is found. This process will decrease the current *frontwidth* if possible.

This process of element assembly/reduction is continued until all the elements and all the degrees of freedom have been processed. At that point all the reduced equations and load terms for all the degrees of freedom will have been written to disk storage and the assembly/reduction portion of the program will be completed.

The next step is to read the equations back from the disk and solve for the degree of freedom displacements.

Step 19: As shown in the earlier discussion, the order of reading the equations is the reverse of the order in which they were written to the disk storage. Thus, the program loops over all the eliminated degrees of freedom in reverse order.

Step 20: The program reads one record at a time and it reads all the information that was written to disk for that record in the assembly/reduction portion of the algorithm above. From this information and the already solved degree of freedom displacements, the degree of freedom of the current record is solved by back-substitution.

Step 21: When each final degree of freedom displacement is calculated, it is stored in two different locations. The first area is a buffer that holds only the already determined displacements which are needed for back-substitution into the equations of the subsequent degrees of freedom. The second area is an in-core data storage location that the CAEFRAME program output routines use to write the results of the analysis.

Resolution Capabilities

As discussed in section II, in any finite element code it is desirable to have the ability to solve a problem for multiple loadcases without having to reduce the stiffness matrix for each loadcase.

There are two alternative methods to accomplish this goal with the frontal algorithm.

The first method is to have the global load vector for each loadcase assembled in the core all at same time. That is, when a new degree of freedom is brought into the front, its total load for all its load cases is entered into all the required separate load vectors. Then, all the load vectors for all the loadcases are reduced at the same time that the stiffness vector and the first loadcase is reduced. This is followed by solving for the displacements for all the loadcases simultaneously in the back-substitution phase.

The second method is to assemble and reduce the second load vector, (followed by the required back-substitution to find the displacements), after the first loadcase has been solved. This process is then repeated for the third or any other subsequent loadcase.

The second method for resolution is the method that was chosen by the author for the following reasons:

First, if all the loadcases are stored in the core at the same time, a much larger allocation of in-core storage area would be required for all the load and displacement vectors. But, if more

in-core storage area is required for the load and displacement vector, less in-core storage area is available for the stiffness vector. If the core area for the stiffness vector is reduced, then the maximum allowable frontwidth must also be reduced. This is not desirable since it decreases the flexibility of the algorithm.

Second, for the first resolution method, no matter how much core area is allocated for the load and displacement vectors, there is still a finite limit to the number of loadcases that can be solved. However, if the second resolution method is used, there is in theory no limit to the number of loadcases that can be solved.

Third, the original program is already structured to solve for loadcases using the second method, which means integration into the current CAEFRAME program would be easier.

Fourth, the required information to efficiently execute the second resolution method is already available in the form of the reduced equations that have been written to the disk in the assembly and reduction of the first loadcase.

In using the second method, the resolution of a second or subsequent loadcase proceeds through the subroutine FRONT in the same manner as for the solution of the first loadcase with the following exceptions:

Exception 1: The element stiffness matrices are not calculated and the global stiffness vector is not formed. Instead, only the new load vector is formed as the newly *activated* degrees of freedom of each element are brought into the *front*.

Exception 2: When a degree of freedom is ready to be deactivated, its new load term is written to a different file than was used in the solution of the first loadcase.

Exception 3: For the reduction of the load vector phase when a degree of freedom is being deactivated, the required stiffness terms are obtained by reading the corresponding equation from the already reduced set on disk storage. Since the degrees of freedom are deactivated in the same order for each loadcase, for this step, the records written in the reduction of the first loadcase are just read back in the same order.

Exception 4: In the back-substitution phase, after the original equation for the degree of freedom is read back from the disk, the new load term for that equation is read back from the file that was created in exception 2, and substituted for the load term that was generated in the solution of the first loadcase.

In this way, only the new load vector is assembled and reduced, and the new reduced load values are simply substituted into the originally reduced equations to obtain the new degree of freedom displacements.

Frontwidth Reduction

As with the blocked out-of-core solver, the efficiency of the frontal algorithm depends on the number of terms stored in-core at any point in the solution. However, because the equation for a newly activated degree of freedom can go into any vacant position of the front, a renumbering of the nodes will not necessarily reduce

the amount of terms stored in-core. But, the parameter that can influence the number of terms stored in-core at any point in the algorithm is the order in which the elements pass through the front.

For the illustrative example in Figure 3.1 on page 77, suppose element 3 was brought into the front before element 2. The global set of equations as shown in equation set 3.4 would now be as follows:

$$\begin{vmatrix} k_{55} & k_{54} & k_{52} & k_{53} \\ k_{45} & k_{44} & k_{42} & k_{43} \\ k_{25} & k_{24} & k_{22}^* & k_{23} \\ k_{35} & k_{34} & k_{32} & k_{33} \end{vmatrix} \begin{vmatrix} u_5 \\ u_4 \\ u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} r_5 \\ r_4 \\ r_2 \\ r_3 \end{vmatrix} \quad (3.10)$$

where: $k_{22}^* = k_{22}' + k_{22}^3$

Notice that equation set 3.10 has four equations in it instead of three as in equation set 3.5. Even if symmetry is taken into account, a total of four additional terms will require in-core storage if equation set 3.10 is used instead of equation set 3.4. Therefore, the frontwidth at this point for this element ordering is larger than the frontwidth at the same point in the algorithm under the original element ordering. In addition, when the degree of freedom at node 5 is *deactivated* at this point, these four extra terms will all have to be reduced by equation 3.2a, and one additional term will have to be written to disk when the equation for this degree of freedom is removed from the core. This is not desirable because of the additional time and out-of-core storage area required for these operations.

By examining the mesh again of Figure 3.1 on page 77, it can be shown that the reason processing element 2 after element 1 was more efficient than processing element 3 after element 1, is that element 2 is attached to more nodes that were already in the front at this point in the algorithm. By being attached to more nodes that are already in the front, there is a smaller chance of increasing the front by adding *inactive* nodes. Also, there is a greater chance that this element would provide the last appearance of one of the nodes already in the front and thereby decrease the current frontwidth.

Therefore, to make the frontal algorithm efficient, the elements should be brought into the front in such a manner that they help the degrees of freedom currently in the front pass through the front as quickly as possible. In an attempt to accomplish this goal, an algorithm that would renumber the elements in an order to reduce the instantaneous frontwidth was incorporated into the frontal algorithm. This frontwidth reduction algorithm is based on work by Sloan and Randolph.¹⁹

The mesh in Figure 3.2 will be used to illustrate the logic of this element reordering method. This mesh consists of eight nodes and seven elements and was arbitrarily numbered for both nodes and elements.

If the elements of Figure 3.2 are brought into the front in ascending element order (as in the previous discussion) the frontwidth for each step of the algorithm would be as shown

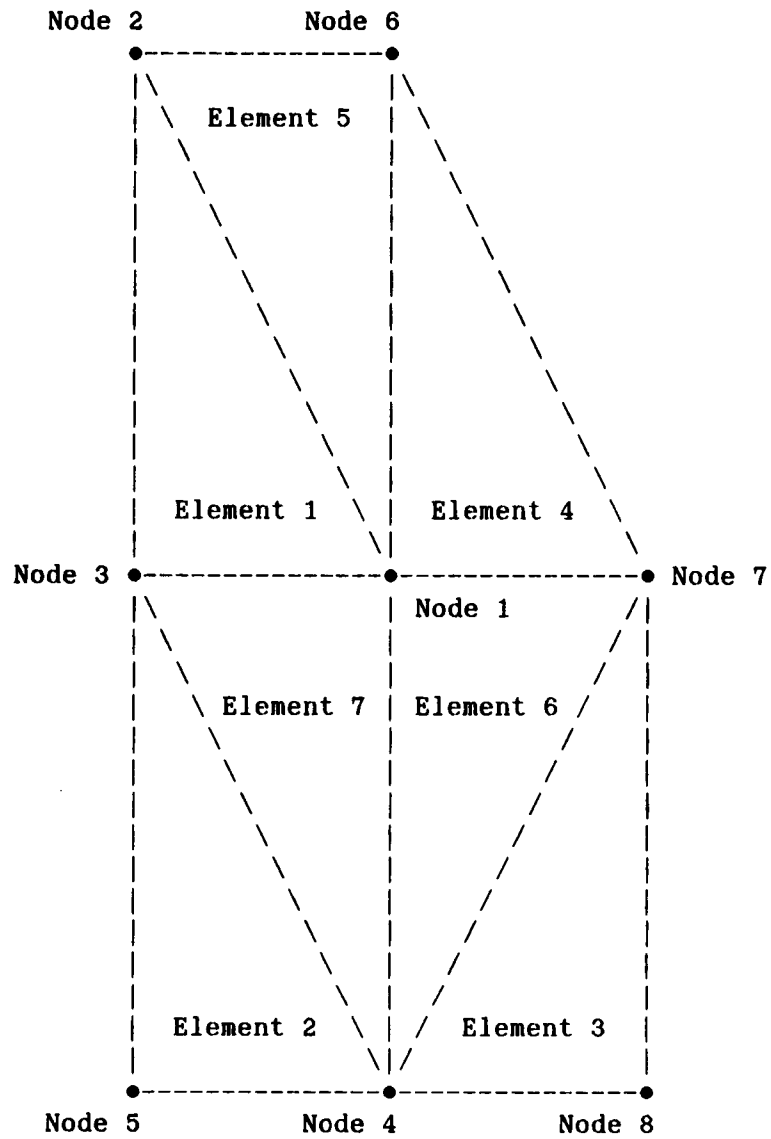


Figure 3.2. Mesh Used for Illustration of Frontwidth Reduction.

in Table 3.1. By examining this table, it can be seen that the maximum frontwidth is 6 and the average frontwidth is 4.7. (Note again that only one degree of freedom per node is assumed. But, the ratios of the frontwidths at the same steps in the algorithm would be the same no matter how many degrees of freedom per node is used. This is because the frontwidth at any step would simply be the number of nodes in the front at that step multiplied by the number of degrees of freedom per node. In fact, the more degrees of freedom per node, the more important it is to reduce the frontwidth due to the multiplication factor.)

However, if the elements are processed through the front in the following order: 5, 1, 4, 7, 6, 2, 3, the new frontwidths at the various steps in the algorithm would be as shown in Table 3.2. (The reason for this particular order will be explained later.) From this table it can be seen that the maximum frontwidth is now 4 and the average frontwidth now down to 3.7.

By changing the order of element calls, the maximum frontwidth was reduced by two nodes and the average frontwidth was reduced by one node. If six degrees of freedom per node was used instead of one, this would imply that for each step of the algorithm the core would contain an average of six equations less than if the original ordering was used. This is a significant savings.

It should be noted that there are other element orderings that will reduce the maximum frontwidth down to 4, but, there is no other possible element ordering that will reduce the frontwidth any smaller than 4.

Table 3.1 Frontwidths for Current Element Ordering.

Element in the Front	Nodes in the Front	Nodes to be Eliminated	Current Front- Width	List of Eliminated Nodes
1	1,2,3	-	3	-
2	1,2,3,4,5	5	5	5
3	1,2,3,4,7,8	8	6	5,8
4	1,2,3,4,7,6	-	6	5,8
5	1,2,3,4,7,6	2,6	6	5,8,2,6
6	1,3,4,7	7	4	5,8,2,6,7
7	1,3,4	1,3,4	3	5,8,2,6,7,1,3,4

Table 3.2. Frontwidths for Revised Element Ordering.

Element in the Front	Nodes in the Front	Nodes to be Eliminated	Current Front- Width	List of Eliminated Nodes
5	1,2,6	-	3	-
1	1,2,6,3	2	4	2
4	1,6,3,7	6	4	2,6
7	1,3,7,4	-	4	2,6
6	1,3,7,4	1	4	2,6,1
2	3,7,4,5	3,5	4	2,6,1,3,5
3	7,4,8	7,4,8	3	2,6,1,3,5,7,4,8

For this small example it is easy to determine an optimal element ordering. However, the problem is how to quickly, and with a minimum amount of coding, determine an element ordering that will have the minimum, or nearly the minimum, maximum frontwidth for any arbitrary structure.

If Figure 3.2 is examined again, it is obvious that the bandwidth of the mesh is also not optimized. Defining the bandwidth as the nodal number difference of any element plus one, (this would be the equivalent of the column height in the skyline storage scheme, or the half-bandwidth in most definitions), and if the element nodal differences are examined, the bandwidth of each element can be determined. Table 3.3 lists the bandwidth of each element of Figure 3.2 for its current node numbering. By examining Table 3.3, it can be seen that the current maximum bandwidth of the structure is 7 and the average bandwidth is 5.

However, if the nodes are renumbered to minimize the bandwidth as shown in Table 3.4 and Figure 3.3, and the new bandwidths are tabulated as in Table 3.5, the maximum bandwidth is reduced to 4 and the average bandwidth is reduced to 3.9. (It should be noted that the author could not find any other nodal renumbering that produced either the same or a smaller bandwidth, therefore, this renumbering should be the true minimum of the system.)

Now, if the elements are listed in ascending numerical order according to the lowest node number attached to each element, the list would be as shown in Table 3.6.

Table 3.3. Original Element Bandwidths.

Element Number	Nodes of Element	Band- Width
1	1,2,3	3
2	3,4,5	3
3	4,7,8	5
4	1,6,7	7
5	1,2,6	6
6	1,4,7	7
7	1,3,4	4

Table 3.4. Revised Node Numbers for Bandwidth Minimization.

Old Node Number	New Node Number
1	3
2	1
3	4
4	6
5	7
6	2
7	5

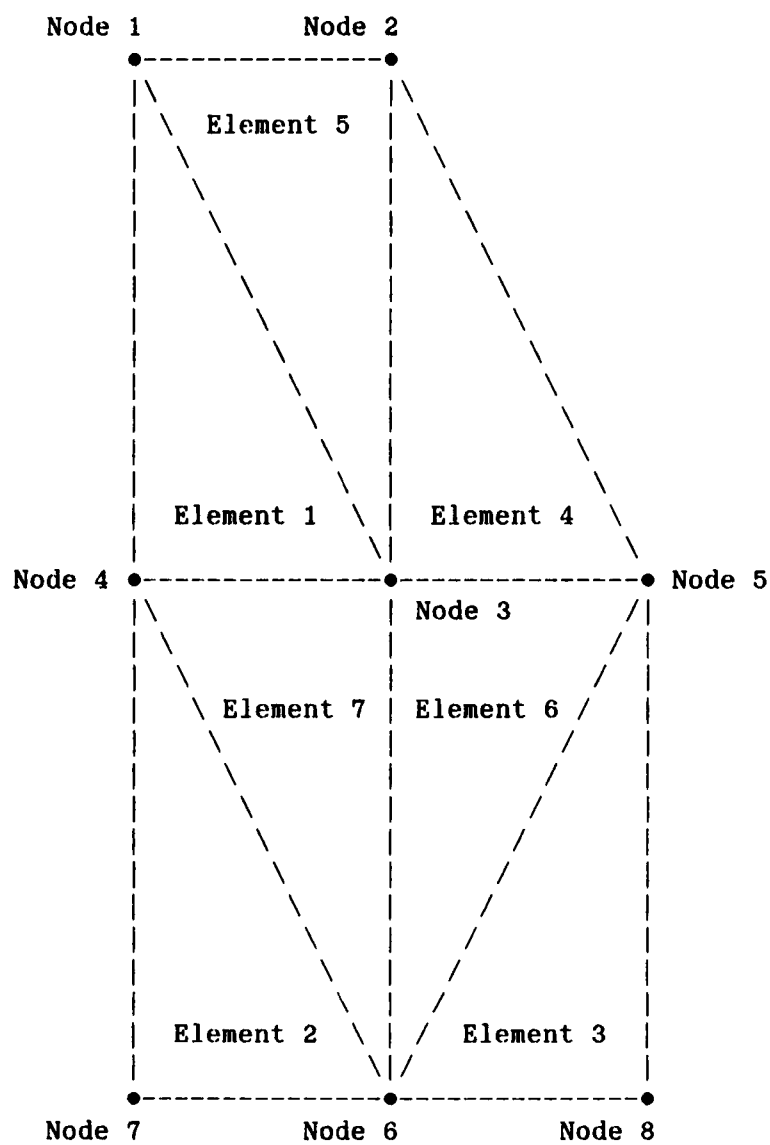


Figure 3.3. Mesh with Nodes Renumbered.

Table 3.5. Revised Element Bandwidths.

Element Number	Nodes of Element	Band- Width
1	1,3,4	4
2	4,6,7	4
3	5,6,8	4
4	2,3,5	4
5	1,2,3	3
6	3,5,6	4
7	3,4,6	4

Table 3.6. Element's Lowest Revised Node Number.

Lowest Node Number	Attached Element
1	5
1	2
2	4
3	7
3	6
4	2
5	3

Notice that the order of the elements listed in Table 3.6 is the order used earlier and is one of the possible orders of the elements that will produce that minimum frontwidth. Also, from Table 3.6, it is noted that both elements 5 and 2 have the new node one as their lowest node, and elements 7 and 6 have the new node number three as their lowest node number. However, if element 2 is processed before element 5, and element 6 is processed before element 7, as shown in Table 3.7, there is no effect on the minimum frontwidth.

Table 3.7. Frontwidths for Alternative Revised Element Ordering.

Element in the Front	Nodes in the Front	Nodes to be Eliminated	Current Front- Width	List of Eliminated Nodes
1	1,2,3	-	3	-
5	1,2,3,6	2	4	2
4	1,3,6,7	6	4	2,6
6	1,3,7,4	-	4	2,6
7	1,3,7,4	1	4	2,6,1
2	3,7,4,5	3,5	4	2,6,1,3,5
3	7,4,8	7,4,8	3	2,6,1,3,5,7,4,8

This relationship of renumbering the elements according to their lowest node number after bandwidth minimization has been performed on the structure is the basis of this reordering scheme.

If the structure is now renumbered for both minimum bandwidth and minimum frontwidth, it will appear as in Figure 3.4.

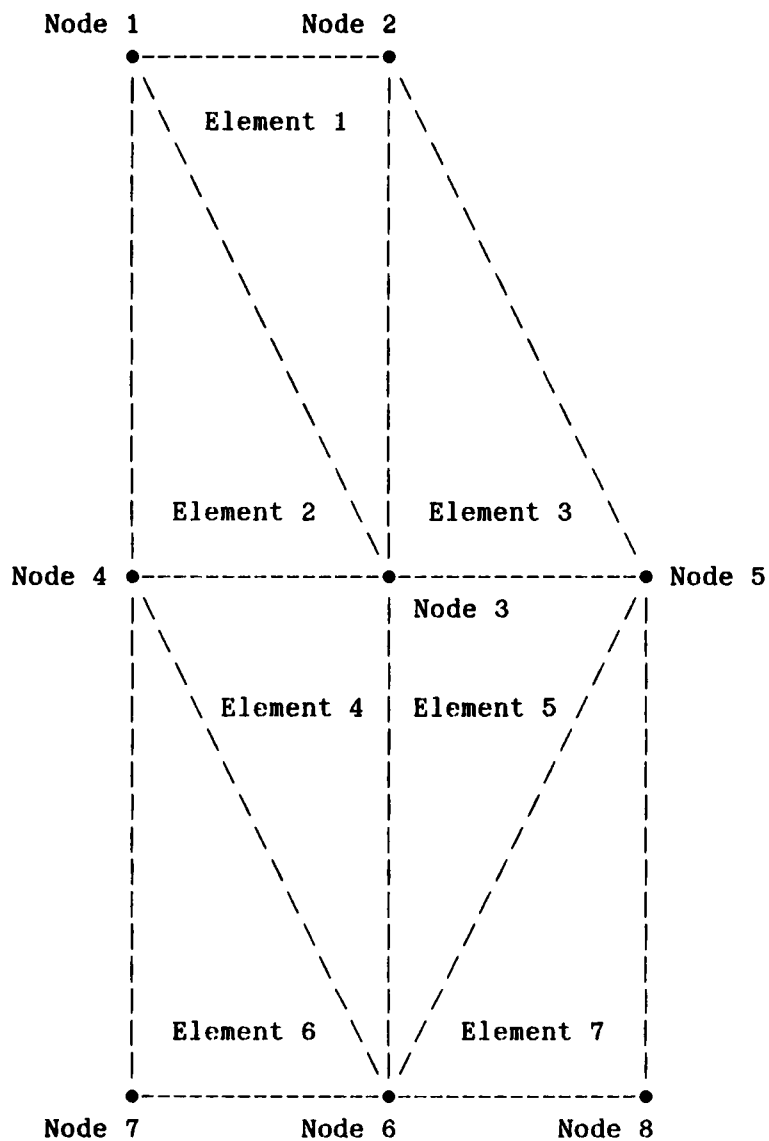


Figure 3.4. Mesh with Nodes and Elements Renumbered.

For any structure the following procedure is applied in order to try to minimize the frontwidth.

Step 1: The nodes of the structure are renumbered to produce the minimum bandwidth.

Step 2: Then the connectivity array that relates which nodes are attached to which element is scanned in order. The first element encountered that has the new node number 1 attached to it is renumbered to the new element 1. The scan then continues through the connectivity array and the next element that has the new node number 1 attached to it is renumbered to the new element number 2. The scan through the connectivity array is continued until the end of the array is encountered. At this point, all the elements that have the new node number 1 as their lowest node number have been renumbered.

Step 3: The connectivity array is scanned again from the beginning. This time, however, the first element encountered that is attached to the new node number 2 is assigned the next available element number provided that element has not already been renumbered. The scan through the connectivity array continues until the end of the array is encountered. During this scan, all the elements that have not already been renumbered and that have the new node number 2 as their lowest node number will be renumbered.

Step 4: Rescan the connectivity array for the next new node number and renumber all the elements that have not been renumbered and have the current new node number as their lowest node number. This is continued until all the elements have been renumbered.

The subroutine that renumbers the nodes for the minimum bandwidth is the same one used for the blocked out-of-core solver, namely, subroutine RENUM listed in Appendix C.

The subroutine that takes the renumbered node list generated in subroutine RENUM and uses it to renumber the elements is subroutine ORDRED listed in Appendix D.

Modifications and Improvements to the Algorithms

The author had to make considerable modifications to the original FORTRAN code by Hinton and Owen¹¹ in order for the algorithm to function properly in the CAEFRAME code. In addition to these functionally required changes, the author made one modification to the original code that was not necessary for its proper operation, but allowed for a considerable improvement in the efficiency of the algorithm.

The original program was designed for use on a code that had only one type of element with the same number of degrees of freedom per node. Since the CAEFRAME code has three types of elements and their nodes can have either three or six degrees of freedom as well as releases, the author's version of the algorithm had to be modified to handle this mix of elements and nodal degrees of freedom.

Also, in the original program, the element stiffness matrices were all calculated prior to the execution of the frontal algorithm and then written to disk storage, while the degree of freedom load terms were calculated from the contribution of each element for each degree of freedom. In the author's version, the element

stiffness matrices are calculated as the element is brought into the front. This eliminates a considerable amount of input/output operations which are expensive in both time and disk space. Also, instead of adding to the load term for a degree of freedom every time an element that contained that degree of freedom entered the front, the total load for each degree of freedom was just copied from the complete load vector already calculated by another part of the CAEFRAME code.

In the original program, the elimination flag was set to the last appearance of a node in the front. This was changed to the last appearance of the actual degrees of freedom. In this way, released degrees of freedom would be eliminated faster.

However, the most significant improvement the author made was with the buffer of stiffness coefficients used when a degree of freedom was being eliminated. In the original program, the buffer of stiffness coefficients needed to write the current equation to disk, contained all the terms relating the degree of freedom being eliminated to every position that was possible in the front. This included all the unoccupied positions that existed beyond the current frontwidth. Therefore, the number of coefficients written to disk for every eliminated degree of freedom equalled the maximum frontwidth. But, for most problems, the average frontwidth is only a small percentage of the maximum possible frontwidth, and often the maximum frontwidth of the problem is still much less than the maximum allowable. This means that considerable time and disk

storage space would be wasted writing all this unnecessary information that existed beyond the current frontwidth to the disk.

In the author's version, by using unformatted files and by writing the value of the current frontwidth to disk as part of the record, only the stiffness terms in the buffer up through the current frontwidth had to be written to disk. This modification saved considerable time and disk space and was not very difficult to incorporate into the logic of the algorithm.

For the frontwidth reduction algorithm, the only change the author made was to substitute the bandwidth reduction by Collins,¹⁴ used for the blocked out-of-core solver, for one of the routines in the original work by Sloan and Randolph.¹⁹

IV. RESULTS

In order to decide which configuration of which out-of-core solver should replace the current CAEFAME in-core solver, the results of actually using the various solvers to analyze several different test models should be compared. To ensure that the comparison of the different solvers is complete and straightforward, quantifiable criteria for making comparisons between any two solvers as well as the procedure, or logical order, of applying that given quantifiable comparison criteria has to be established. In addition, a brief description of the test models used to generate the data is also presented.

Comparison Criteria and Procedure

Three quantifiable criteria for making comparisons between any two solvers will be used. These three criteria are as follows:

1. the runtimes for the solution of any particular test model on any solver; (Note, this will include a discussion of the runtime to complete the solution of the first loadcase, as well as a separate discussion of the runtime to complete the resolution of a second or any other subsequent loadcase.)
2. where applicable, the total disk storage requirements needed for erasable scratch files for the solution of both the first and second loadcase of any particular model by any solver; (Note, for all solvers that use disk storage for erasable scratch files, no additional area

is needed for the resolution of any loadcase beyond the second. This is because any extra information that has to be written to disk for the resolution of any loadcase after the second one is simply written over the same information that had to be written to the disk for resolution of the second loadcase.)

3. the maximum size of any problem that can be solved by each solver.

As stated in the introduction, the procedure for applying the comparison criteria is to first compare each out-of-core solver to itself both with and without its respective efficiency improvement technique incorporated into it.

However, there is the possibility that no configuration of either out-of-core solver will be recommended as a replacement for the current in-core solver. Therefore, this initial comparison phase will also include an evaluation of the current in-core solver with and without the same bandwidth reduction technique used for the blocked out-of-core solver incorporated into it. This is done to ascertain if the current in-core solver's capabilities can be improved if it is decided not to replace it. This comparison of the two configurations of the in-core solver will actually be done first, followed by the comparison of the two configurations of the blocked out-of-core solver, and finally the comparison of the two configurations of the frontal solver will end the first phase of comparisons.

From this initial comparison phase, the best configuration of each type of solver will be chosen as a possible candidate for the final recommended solver. Then the remaining candidates will be compared with each other to determine the final recommended solver.

Test Models Used to Establish a Quantifiable Comparison

In order to obtain the data for a quantifiable basis of comparison, eleven test models were run on each configuration of each solver. For each model, two loadcases were run so that both the runtime for the solution of the first loadcase and the runtime for the resolution of one subsequent loadcase could be measured. All the test models and their respective runtimes for the solution of the first loadcase, as well as the runtimes for the resolution of a second loadcase, on each of the solvers are listed in Table 4.1. (Note, all times in all the tables of this section have been rounded to the nearest second, therefore, time differences may actually vary by nearly one second from those that are actually shown.)

Table 4.2 provides the disk storage requirements needed for erasable scratch files used to run each model on each solver where this requirement is applicable. The original data from which both Table 4.1 and Table 4.2 were compiled is taken from the raw output data for each solution given in Appendix A, while the information on the configuration and loading conditions of each model is given in Appendix B.

Table 4.1. Runtimes for All The Solvers.

Model Name	No. of Eqns.	Load ¹ Case No.	Original Solver		Blocked Out-Of-Core Solver		Frontal Solver	
			WITHOUT	WITH	WITHOUT	WITH	WITHOUT	WITH
			Band- width Reducer	Band- width Reducer	Band- width Reducer	Band- width Reducer	Front- width Reducer	Front- width Reducer
----- seconds -----								
PLT2X10.STR	164	1	86	86	113	115	127	128
		2	22	23	35	36	52	52
PLT10X2.STR	164	1	147	88	198	118	206	129
		2	25	23	55	36	89	54
PLT4X10.STR	288	1	200	201	290	291	280	281
		2	45	45	80	79	110	110
PLT10X4.STR	288	1	327	212	481	308	386	284
		2	48	45	113	83	149	110
PL10X10.STR	702	1	Too	Too	1473	1473	1090	1096
		2	Large	Large	331	329	392	390
PL10X10A.STR	486	1	615	619	875	882	995	1004
		2	123	123	226	227	381	382
TMS1.STR	240	1	190	200	265	283	Too	336
		2	45	45	77	79	Large	102
BRIDGE1.STR	180	1	197	114	277	157	285	209
		2	31	29	71	50	109	68
TESTRUN.STR	534	1	358	388	525	581	Too	562
		2	77	78	141	150	Large	183
TANK.STR	438	1	Too	Too	2416	1287	1644	1157
		2	Large	Large	331	235	507	387
SPHERE1.STR	432	1	Too	376	2193	548	1391	497
		2	Large	74	291	155	489	201

¹The time shown for the solution of the second loadcase for each model is the additional time required to complete that solution after the first loadcase has been solved.

Table 4.2. Scratch File Requirements for Both The Blocked Out-Of-Core Solver and The Frontal Solver.

Model Name	No. of Eqns.	Blocked Out-Of-Core Solver		Frontal Solver	
		WITHOUT Band- width Reducer	WITH Band- width Reducer	WITHOUT Front- width Reducer	WITH Front- width Reducer
		----- bytes -----			
PLT2X10.STR	164	57684	57684	50476	50476
PLT10X2.STR	164	122668	59308	110452	50476
PLT4X10.STR	288	157396	157396	115252	115252
PLT10X4.STR	288	265944	168364	179284	116800
PL10X10.STR	702	791854	791854	448660	448660
PL10X10A.STR	486	453830	453830	448660	448660
TMS1.STR	240	137218	145990	Too Large	97312
BRIDGE1.STR	180	159940	92394	127504	65368
TESTRUN.STR	534	368254	396210	Too Large	211708
TANK.STR	438	931534	586936	694708	515428
SPHERE1.STR	432	905784	358032	681948	240168

All of the first six models (PLT2X10.STR, PLT10X2.STR, PLT4X10.STR, PLT10X4.STR, PL10X10.STR, and PL10X10A.STR) use only plate elements and are two dimensional. These models were chosen because they were easy to develop and they could be used to test some important aspects of each of the solvers.

PLT2X10.STR and PL10X2.STR are models of the same structure, namely a two by ten rectangular mesh of plates. Similarly, models PLT4X10.STR and PLT10X4.STR are both four by ten rectangular meshes of the same structure. The differences are that models PLT2X10.STR and PLT4X10.STR have optimal numbering for both nodes and elements. Therefore, these two models will produce both the minimum bandwidth and minimum frontwidth for their respective structures. However, models PLT10X2.STR and PLT10X4.STR have both node and element numberings that will produce bandwidths and frontwidths that are considerably larger than minimum for their respective structures. Taken together, these four models will allow for a direct analysis of the effectiveness of using the two different efficiency improvement techniques.

PL10X10.STR and PL10X10A.STR are also models of the same structure, namely a ten by ten square mesh. The only difference in these two models is the boundary conditions. This difference in boundary conditions changes the number of free degrees of freedom and, therefore, changes the size of the problem. Model PL10X10.STR was chosen because even though this problem is both bandwidth and frontwidth optimized, it still will not run on the current in-core solver since its stiffness vector exceeds the size limit of the

current in-core solver by approximately fifty percent. As such, PL10X10.STR is considered a large size problem for this code. For model PL10X10A.STR, the boundary conditions were changed to reduce the size of the problem so that it could run on the current in-core solver. However, this problem's stiffness vector still exceeds ninety percent of the allowable size of the current in-core solver, and, therefore, model PL10X10A.STR is considered as the upper size limit for the current in-core solver.

The remaining five models, (TMS1.STR, BRIDGE1.STR, TESTRUN.STR, TANK.STR, and SPHERE1.STR), are general structures of two and three dimensions. These models are composed of all three types and mixes of elements, as well as of other features of the CAEFRAME code such as spring supports and released degrees of freedom. These models were provided by SST Inc. and are models that several of their customers have tried to analyze with the CAEFRAME code. Therefore, these models represent typical problems that an engineer using the software would analyze.

Comparison of the Two In-Core Solver Configurations

In Table 4.3, a comparison is made of the runtimes for all the models as run on the current in-core solver both with and without the bandwidth reduction technique. (There is no scratch file disk storage space requirements for either configuration of this solver.)

By examining the results for the non-optimized models PLT10X2.STR and PLT10X4.STR, it is evident that the addition of the bandwidth reducer *significantly improves* the runtime for the

Table 4.3. Comparison of Runtimes between The In-Core Solver
With and Without The Bandwidth Reduction Technique.

Model Name	No. of Eqns.	Load ¹ Case No.	In-Core Solver WITHOUT the Bandwidth Reducer	In-Core Solver WITH the Bandwidth Reducer	Difference	Percentage Difference
			----- seconds -----			
PLT2X10.STR	164	1	86	86	0	0.0
		2	22	23	+1	+4.5
PLT10X2.STR	164	1	147	88	-59	-40.1
		2	25	23	-2	-8.0
PLT4X10.STR	288	1	200	201	+1	+0.5
		2	45	45	0	0.0
PLT10X4.STR	288	1	327	212	-115	-35.2
		2	48	45	-3	-6.3
PL10X10.STR	702	1	Too	Too	-----	-----
		2	Large	Large	-----	-----
PL10X10A.STR	486	1	615	619	+4	+0.7
		2	123	123	0	0.0
TMS1.STR	240	1	190	200	+10	+5.3
		2	45	45	0	0.0
BRIDGE1.STR	180	1	197	114	-83	-42.1
		2	31	29	-2	-6.5
TESTRUN.STR	534	1	358	388	+30	+8.4
		2	77	78	+1	+1.3
TANK.STR	438	1	Too	Too	-----	-----
		2	Large	Large	-----	-----
SPHERE1.STR	432	1	Too	376	-----	-----
		2	Large	74	-----	-----

¹The time shown for the solution of the second loadcase for each model is the additional time required to complete that solution after the first loadcase has been solved.

solution of the first loadcase, and slightly improves the runtime for the resolution of a second or subsequent loadcase for simple problems that are obviously not optimally numbered. But, just as importantly, by examining the results for models PLT2X10.STR, PLT4X10.STR, and PL10X10A.STR (as stated earlier, PL10X10.STR would not run on either configuration of the in-core solver) the bandwidth reducer *does not* significantly increase the runtime for the solution of any loadcase on problems that are optimally numbered, even though an attempt to reduce the bandwidth is still made. The information in Table 4.3 indicates that the nodal reordering scheme takes: less than one second on a small problem like PLT2X10.STR; about one to two seconds on an intermediately sized problem like PLT4X10.STR; and about four to five seconds on a large problem like PL10X10A.STR.

Also, the runtimes for the non-optimized problems (PLT10X2.STR and PLT10X4.STR) approach the runtimes for the optimized models (PLT2X10.STR and PLT4X10.STR) when a nodal renumbering is performed. Although, if the actual lengths of the stiffness vectors are examined as shown in Table 4.4, the nodal renumbering for both cases is not optimum.

All of this data supports the statements made in the introduction that the selected reordering algorithm may not completely optimize a structure, but it is very fast and it will significantly *reduce* the time for solution of a non-optimized problem.

Thus, it appears that the bandwidth reducer should be used since there is virtually no penalty in runtime when used on optimized

Table 4.4. Comparison of the Effectiveness of Nodal Renumbering.

Model Name	Length of Stiffness Vector Before Reordering	Length of Stiffness Vector After Reordering	Length of Optimized Stiffness Vector	Difference Between Reordered & Optimized
PLT10X2.STR	7445	3456	3446	+10
PLT10X4.STR	16183	10179	9504	+675

problems and, even though complete optimization may not occur, significant gains in runtime can be achieved when used on problems that are not optimized.

However, in examining the results for the other five models some interesting and conflicting results are apparent.

For model BRIDGE1.STR, the runtimes for both the solution of the first loadcase and the resolution of the second loadcase were improved about the same percentage as in the non-optimal plate models when run with the bandwidth reducer. The results for this model tend to support the same conclusions that were made about the bandwidth reducer when the two dimensional plate models were examined.

But, if the results for the generalized three dimensional structures TMS1.STR and TESTRUN.STR are examined, a small but significant increase in runtimes for the solution of the first loadcase is noted for both structures when the bandwidth reducer is used. (There is little or no increase in the resolution for

the second loadcase for both structures.) An initial explanation for this increase might be that the additional time is required to execute the algorithm that renumbers the nodes of the model to reduce the bandwidth. However, as already shown for the optimized plate models, the execution of the renumbering algorithm is very fast even on large problems like PL10X10.STR and, therefore, it should have no significant effect on the total runtime for the solution. But, by investigating the output information for each solution listed in Appendix A, note that the actual size of the stiffness vector *increased* slightly. As shown in Table 4.5, this increase in vector stiffness length is about the same percentage as the increase in runtimes for both of these models. The reason for this increase is associated with the parameter on which the bandwidth reducer focuses.

Table 4.5. Comparison of Two Selected Stiffness Vectors Before and After Nodal Renumbering Is Performed.

Model Name	Length of Stiffness Vector Before Reordering	Length of Stiffness Vector After Reordering	Difference	Percentage Difference
TMS1.STR	8292	8832	+540	+6.5
TESTRUN.STR	22317	24045	+1728	+7.7

The parameter that the bandwidth reducer focuses on is the *maximum* bandwidth of the stiffness matrix. For some structures it is possible to renumber the nodes to reduce the maximum bandwidth of the stiffness matrix, while at the same time increasing the *average* bandwidth of the stiffness matrix. This increase in average bandwidth will in turn increase the total stiffness vector size, which will in turn increase the runtime for the solution of the first loadcase due to the increased time to perform the stiffness vector decomposition. This is what happened to models TMS1.STR and TESTRUN.STR.

Finally, models TANK.STR and SPHERE1.STR could not run on the in-core solver in its original configuration because their stiffness vectors were too large and, therefore, no runtime comparisons could be made using these two models. However, by using the bandwidth reducer, the in-core solver was able solve the SPHERE1.STR problem. This indicates that the bandwidth reducer can be useful in solving problems that normally could not run on the current in-core solver.

Comparison of the Two Blocked Out-Of-Core Solver Configurations

In Table 4.6 a comparison is made of the runtimes for all the models as run on the blocked out-of-core solver both with and without the bandwidth reduction technique. In Table 4.7, a comparison is made of the disk storage requirements used for erasable scratch files needed for all of the models as run on the same two solvers.

Since this solver is the out-of-core extension of the current in-core solver, similar results should be expected for the comparison

Table 4.6. Comparison of Runtimes between The Blocked Out-Of-Core Solver With and Without The Bandwidth Reduction Technique.

Model Name	No. of Eqns.	Load ¹ Case No.	Blocked Out-Of-Core Solver WITHOUT the Bandwidth Reducer	Blocked Out-Of-Core Solver WITH the Bandwidth Reducer	Difference	Percentage Difference
			seconds			
PLT2X10.STR	164	1	113	115	+2	+1.8
		2	35	36	+1	2.9
PLT10X2.STR	164	1	198	118	-80	-40.4
		2	55	36	-19	-34.5
PLT4X10.STR	288	1	290	291	+1	+0.3
		2	80	79	-1	+1.3
PLT10X4.STR	288	1	481	308	-173	-36.0
		2	113	83	-30	-26.5
PL10X10.STR	702	1	1473	1473	0	0.0
		2	331	329	-2	-0.6
PL10X10A.STR	486	1	875	882	+7	+0.8
		2	226	227	+1	+0.4
TMS1.STR	240	1	265	283	+22	+8.3
		2	77	79	+2	+2.6
BRIDGE1.STR	180	1	277	157	-120	-43.3
		2	71	50	-21	-29.6
TESTRUN.STR	534	1	525	581	+56	+10.7
		2	141	150	+9	+6.4
TANK.STR	438	1	2416	1287	-1129	-46.7
		2	331	235	-96	-29.0
SPHERE1.STR	432	1	2193	584	-1609	-73.4
		2	291	155	-136	-46.7

¹The time shown for the solution of the second loadcase for each model is the additional time required to complete that solution after the first loadcase has been solved.

Table 4.7. Comparison of Scratch File Requirements between The Blocked Out-Of-Core Solver With and Without The Bandwidth Reduction Technique.

Model Name	No. of Eqns.	WITHOUT Band-width Reducer	WITH Band-width Reducer	Difference	Percentage Difference
----- bytes -----					
PLT2X10.STR	164	57684	57684	0	0.0
PLT10X2.STR	164	122668	59308	-63360	-51.7
PLT4X10.STR	288	157396	157396	0	0.0
PLT10X4.STR	288	265944	168364	-97580	-36.7
PL10X10.STR	702	791854	791854	0	0.0
PL10X10A.STR	486	453830	453830	0	0.0
TMS1.STR	240	137218	145990	+8772	+6.4
BRIDGE1.STR	180	159940	92394	-67546	-42.2
TESTRUN.STR	534	368254	396210	+27956	+7.6
TANK.STR	438	931534	586936	-344598	-37.0
SPHERE1.STR	432	905784	358032	-547752	-60.5

of the two configurations of this solver to those that were found when the comparison was made between the in-core solver both with and without the bandwidth reduction technique. The results are similar, but both the increases and decreases in runtimes between using the two configurations of this solver are more pronounced.

For the non-optimized models PLT10X2.STR and PLT10X4.STR, improvements are again noted in the runtimes for the solution of both the first loadcase and the resolution of the second loadcase when the bandwidth reduction technique is utilized. (For this solver, these problems will be renumbered exactly the same as when using the bandwidth reducer on the in-core solver and, therefore, these models will still not be optimally renumbered.) However, the *percentage* improvement in runtimes is higher when the bandwidth reducer is used on the blocked out-of-core solver than when it is used on the in-core solver, especially for the the resolution of the second loadcase.

This larger percentage improvement can be accounted for by the fact that as the nodes are renumbered the size of the stiffness vector is decreased as shown previously in Table 4.4. However, for this solver, the disk storage requirements needed for erasable scratch files also decreases as shown in Table 4.7. This reduction in erasable scratch file size requirements results in fewer read and write operations to the disk. Since disk operations are comparatively slower than core operations, and the disk operations can consume a significant portion of the total solution time for an out-of-core solver, a small percentage decrease in the number

of disk operations can save a disproportionately higher percentage amount of the total runtime.

By examining the plate models which are optimally numbered, (PLT2X10.STR, PLT4X10.STR, PL10X10.STR, and PL10X10A.STR), it is evident that there is still no significant increase in the runtime to solve an optimized problem when the bandwidth reduction technique is utilized in the blocked out-of-core solver. Again, this shows that the algorithm that performed the bandwidth reduction is very fast. Since the nodes are not renumbered for these optimized problems, the stiffness vector remains the same size and, therefore, there is no difference in the disk storage requirements needed for erasable scratch files between the two configurations of the blocked out-of-core solver for these problems.

As with the runtimes for the in-core solver, the BRIDGE1.STR model's runtime for both the solution of the first loadcase and the resolution of the second loadcase improved about the same percentage as the non-optimal plate models PLT10X2.STR and PLT10X4.STR improved when the bandwidth reducer was used. However, for the blocked out-of-core solver, similar or better results were also found for the models TANK.STR and SPHERE1.STR. In fact, the results for SPHERE1.STR show over a seventy-five percent decrease in runtime for the solution of the first loadcase, and nearly a fifty percent decrease in the runtime for the resolution of the second loadcase when run using the bandwidth reduction technique. These results are the best of any model. Therefore, unlike the results for these two models when run on the two configurations

of the in-core solver, these two models tend to support the results of the earlier non-optimized plate models that the bandwidth reducer should be incorporated into the blocked out-of-core solver.

But, if the results for structures TMS1.STR and TESTRUN.STR are examined once more, a slight increase in runtimes for the solution of the first loadcase is again noted for both structures when run on the configuration with the bandwidth reduction technique. However, the *percentage* increase is slightly larger this time, and there is also a slight increase in the resolution of the second loadcase for both structures this time. The reason for the increase in runtimes is the same as that for the in-core solver, namely the average bandwidth has increased due to the nodal renumbering and, therefore, the length of the stiffness vector has increased. This time the increase in the length of the stiffness vector is also evident in the increase of the disk storage requirements needed for the erasable scratch files as shown in Table 4.7. The reason that the *percentage* increase in runtimes are higher than those of the comparison of the two configurations of the in-core solver, is the same as that given above for the larger percentage decrease in runtimes for the non-optimized plate models. Namely, since the stiffness vector has increased, the number of disk reads and writes has increased and, since these operations are slower than the core operations, the runtimes will increase disproportionately.

Even though most of the comparisons between the two configurations of the blocked out-of-core solver and the two configurations of the in-core solver have been similar, one

significant difference is that all the test models could run on both configurations of the blocked out-of-core solver. In fact, given sufficient disk storage space, the theoretical size limit for both configurations of the blocked out-of-core solver is limited only by the input parameters of the CAEFRAME code itself. Currently, the limit of the code is set at a maximum of 999 nodes, 5994 degrees of freedom, and 999 elements.

Comparison of the Two Frontal Solver Configurations

In Table 4.8 a comparison is made of the runtimes for all the models as run on the frontal solver both with and without the frontwidth reduction technique. Table 4.9 shows a comparison of the disk storage requirements used for erasable scratch files for all of the models as run on the same two solvers.

For the non-optimized models PLT10X2.STR and PLT10X4.STR, improvements are again noted in the runtimes for the solution of both the first loadcase and the resolution of the second loadcase when the frontwidth reduction technique is utilized. However, the percentage improvement in runtimes for the solution of the first loadcase are slightly less than those of the blocked out-of-core solver when the bandwidth reduction technique is used. But, the percentage improvements in the resolution of the second loadcase are much higher than those for the blocked out-of-core solver. Also, the runtimes for the solution of these two non-optimized problems more closely approachs the runtimes of the optimized versions of the same problems as run on the same two solvers.

Table 4.8. Comparison of Runtimes between The Frontal Solver
With and Without The Frontwidth Reduction Technique.

Model Name	No. Load ¹ of Case Eqns. No.		Frontal Solver WITHOUT the Frontwidth Reducer	Frontal Solver WITH the Frontwidth Reducer	Difference	Percentage Difference
	seconds					
PLT2X10.STR	164	1	127	128	+1	+0.8
		2	52	52	0	0.0
PLT10X2.STR	164	1	206	129	-77	-37.4
		2	89	54	-35	-39.3
PLT4X10.STR	288	1	280	281	+1	+0.4
		2	110	110	0	0.0
PLT10X4.STR	288	1	386	284	-102	-26.4
		2	149	110	-39	-26.2
PL10X10.STR	702	1	1090	1096	+6	+0.6
		2	392	390	-2	-0.5
PL10X10A.STR	486	1	995	1004	+9	+0.9
		2	381	382	+1	+0.3
TMS1.STR	240	1	Too	336	-----	-----
		2	Large	102	-----	-----
BRIDGE1.STR	180	1	285	209	-76	-26.7
		2	109	68	-41	-37.6
TESTRUN.STR	534	1	Too	562	-----	-----
		2	Large	183	-----	-----
TANK.STR	438	1	1644	1157	-487	-29.6
		2	507	387	-120	-23.7
SPHERE1.STR	432	1	1391	497	-894	-64.3
		2	489	201	-288	-58.9

¹The time shown for the solution of the second loadcase for each model is the additional time required to complete that solution after the first loadcase has been solved.

Table 4.9. Comparison of Scratch File Requirements between The Frontal Solver With and Without The Frontwidth Reduction Technique.

Model Name	No. of Eqns.	WITHOUT Front-width Reducer	WITH Front-width Reducer	Difference	Percentage Difference
----- bytes -----					
PLT2X10.STR	164	50476	50476	0	0.0
PLT10X2.STR	164	110452	50476	-59976	-54.3
PLT4X10.STR	288	115252	115252	0	0.0
PLT10X4.STR	288	179284	116800	-62484	-34.9
PL10X10.STR	702	448660	448660	0	0.0
PL10X10A.STR	486	448660	448660	0	0.0
TMS1.STR	240	Too Large	97312	-----	-----
BRIDGE1.STR	180	127504	65368	-62136	-48.7
TESTRUN.STR	534	Too Large	211708	-----	-----
TANK.STR	438	694708	515428	-179280	-25.8
SPHERE1.STR	432	681948	240168	-441780	-64.8

In fact, for this solver, the reduction algorithm actually optimizes model PLT10X2.STR, and more closely optimizes model PLT10X4.STR than the bandwidth reduction algorithm did on the blocked out-of-core solver. This greater degree of optimization can be seen in the comparison of disk storage requirements for both models PLT10X2.STR and PLT10X4.STR as shown in Table 4.10.

Table 4.10. Comparison of Total Scratch File Requirements on Two Selected Models.

Model Name	Scratch File Before Requirements Frontwidth Reduction	Scratch File After Requirements Frontwidth Reduction	Scratch File Requirements For Optimum Frontwidth	Difference
PLT10X2.STR	110452	50476	50476	0
PLT10X4.STR	179284	116800	115252	1548

Again, by examining the plate models which are optimally numbered (PLT2X10.STR, PLT4X10.STR, PL10X10.STR, and PL10X10A.STR), there is no significant increase in runtimes when the frontwidth reduction scheme is employed. In fact, the information in Table 4.8 indicates that the element reordering scheme takes: about one second on a small problem like PLT2X10.STR; about one to two seconds on an intermediately sized problem like PLT4X10.STR; and about six to seven seconds on a large problem like PL10X10A.STR. Also, there

is no difference in the disk storage requirements needed for erasable scratch files when the element reordering scheme is used on optimized problems.

As with the runtimes for the blocked out-of-core solver with bandwidth reduction, models BRIDGE1.STR, TANK.STR, and SPHERE1.STR runtime's for both the solution of the first loadcase and the resolution of the second loadcase were improved about the same percentage or better as the non-optimal plate models PLT10X2.STR and PLT10X4.STR when the frontwidth reduction scheme was employed.

All of these results support the statements made previously in Section I that the chosen frontwidth reduction scheme can be very effective in *reducing* the frontwidth, even though it may not optimize it; it is very fast and it will not significantly slow down the running of already optimized problems.

When the results for structures TMS1.STR and TESTRUN.STR are examined again, it is noted that this time the problems could not run on the frontal solver unless the frontwidth reduction technique is utilized. At first, it may appear that these two models will not provide any useful information on the effectiveness of the frontwidth reducer. However, these two models actually reveal some good information.

These two models show that what dictates whether a problem will run on the frontal solver is strictly the *maximum frontwidth*. Since the *maximum frontwidth* is the parameter that the frontwidth reducer reduces, these two problems were able to run because the *maximum frontwidth* of each problem was finally reduced below the

maximum allowable frontwidth, even though, the *maximum allowable frontwidth* for the frontal solver is 222 degrees of freedom without the frontwidth reduction and 190 degrees of freedom with the frontwidth reduction. (Note, the decrease in *maximum allowable frontwidth* occurs when the frontwidth reduction is employed due to the additional core space needed to contain the frontwidth reduction algorithm.)

Comparison of the Two Remaining Configurations of Out-Of-Core Solvers

The remaining candidates for possible replacement of the current in-core solver are the blocked out-of-core solver with bandwidth reduction and the frontal solver with frontwidth reduction. (Note, as shown previously, if neither of these two solvers is actually used as a replacement for the current in-core solver, the bandwidth reduction technique should be implemented into the current in-core solver.) In order to make a final recommendation, these two remaining solvers need to be compared.

Table 4.11 gives a comparison of the runtimes of all the models as run on the blocked out-of-core solver with bandwidth reduction and the frontal solver with frontwidth reduction. Table 4.12 gives a comparison of the disk storage requirements needed for erasable scratch files to solve the same problems on the same two solvers.

By examining Table 4.11, it is noted that the time for the solution of the first loadcase is longer for the frontal solver on some problems and shorter on others. In fact, it is almost an even split with six models running faster on the frontal solver

Table 4.11. Comparison of Runtimes Between The Blocked Out-Of-Core Solver and The Frontal Solver With Their Respective Reduction Techniques.

Model Name	No. of Eqns.	Load ¹ Case No.	Blocked Out-Of-Core Solver WITH the Bandwidth Reducer	Frontal Solver WITH the Frontwidth Reducer	Difference	Percentage Difference
			----- seconds -----			
PLT2X10.STR	164	1	115	128	+13	+11.3
		2	36	52	+16	+44.4
PLT10X2.STR	164	1	118	129	+11	+9.3
		2	36	54	+18	+50.0
PLT4X10.STR	288	1	291	281	-10	-3.4
		2	79	110	+31	+39.2
PLT10X4.STR	288	1	308	284	-24	-7.8
		2	83	110	+27	+32.5
PL10X10.STR	702	1	1473	1096	-377	-25.6
		2	329	390	+61	+18.5
PL10X10A.STR	486	1	882	1004	+122	+13.8
		2	227	382	+155	+68.3
TMS1.STR	240	1	283	336	+53	+18.7
		2	79	102	+23	+29.1
BRIDGE1.STR	180	1	157	209	+52	+33.1
		2	50	68	+18	+36.0
TESTRUN.STR	534	1	581	562	-19	-3.3
		2	150	183	+33	+22.0
TANK.STR	438	1	1287	1157	-130	-10.1
		2	235	387	+152	+64.7
SPHERE1.STR	432	1	548	497	-51	-9.3
		2	155	201	+46	+29.7

¹The time shown for the solution of the second loadcase for each model is the additional time required to complete that solution after the first loadcase has been solved.

Table 4.12. Comparison of Scratch File Requirements Between The Blocked Out-Of-Core Solver and The Frontal Solver With Their Respective Reduction Techniques.

Model Name	No. of Eqns.	Blocked Out-Of-Core Solver WITH Band-width Reducer	Frontal Solver WITH Front-width Reducer	Difference	Percentage Difference
----- bytes -----					
PLT2X10.STR	164	57684	50476	-7208	-12.5
PLT10X2.STR	164	59308	50476	-8832	-14.9
PLT4X10.STR	288	157396	115252	-42144	-26.8
PLT10X4.STR	288	168364	116800	-51564	-30.6
PL10X10.STR	702	791854	448660	-343194	-43.3
PL10X10A.STR	486	453830	448660	-5170	-1.1
TMS1.STR	240	145990	97312	-48678	-33.3
BRIDGE1.STR	180	92394	65368	-27026	-29.3
TESTRUN.STR	534	396210	211708	-184502	-46.6
TANK.STR	438	586936	515428	-71508	-12.2
SPHERE1.STR	432	358032	240168	-117864	-32.9

and five running slower. However, for the resolution of all the second loadcases, the frontal solver's runtimes are always longer. In fact, the runtime for the resolution of the second loadcase can take as much as sixty percent longer on the frontal solver than on the blocked out-of-core solver, and for several of the problems the runtimes are at least fifty percent higher.

In Table 4.13 the *total* time to solve both the first and second loadcases are shown. This table shows that for ten of the eleven models, the total time to complete the solution of both loadcases takes as long or longer on the frontal solver than on the blocked out-of-core solver. This trend of longer runtimes for the total solution, as run on the frontal solver, will increase as more loadcases are analyzed. From the author's experience, most analyses will consist of solving several loadcases and, therefore, from a solution runtime point of view, the blocked out-of-core solver is more versatile in most applications.

However, if the file requirements of Table 4.12 are examined, it is evident that the total disk storage requirements for erasable scratch files are less for the frontal solver than for the blocked out-of-core solver for all models. This aspect of the two solvers will not change no matter how many additional loadcases are analyzed since no additional disk storage space will be required for the solution of any subsequent loadcase beyond the second loadcase.

Table 4.13. Comparison of Total Runtimes for Both Loadcase One and Two Between The Blocked Out-Of-Core Solver and The Frontal Solver With Their Respective Reduction Techniques.

Model Name	No. Load of Case Eqns. No.		Blocked Out-Of-Core Solver WITH the Bandwidth Reducer	Frontal Solver WITH the Frontwidth Reducer	Difference	Percentage Difference
			seconds			
PLT2X10.STR	164	1&2	151	180	+29	+19.2
PLT10X2.STR	164	1&2	154	183	+29	+18.8
PLT4X10.STR	288	1&2	370	391	+21	+5.7
PLT10X4.STR	288	1&2	391	394	+3	+0.8
PL10X10.STR	702	1&2	1802	1486	-316	-17.5
PL10X10A.STR	486	1&2	1109	1386	+277	+25.0
TMS1.STR	240	1&2	362	438	+76	+21.0
BRIDGE1.STR	180	1&2	207	277	+70	+33.8
TESTRUN.STR	534	1&2	731	745	+14	+1.9
TANK.STR	438	1&2	1522	1544	+22	+1.4
SPHERE1.STR	432	1&2	703	698	-5	-0.7

V. CONCLUSIONS AND RECOMMENDATIONS

From the results of Section IV, conclusions can be made about the effectiveness of the bandwidth and frontwidth reduction schemes, as well as which out-of-core solver should replace the current in-core solver.

Effectiveness of the Bandwidth Reducer

From the comparisons of the current in-core solver and the blocked out-of-core solver both with and without bandwidth reduction, it is concluded that the bandwidth reducer should be utilized for either of these two solvers. The reasons for this conclusion are supported by the application of the comparison criteria.

For criteria one it was first shown that the bandwidth reducer does not significantly increase the runtimes of optimized problems. Then it was shown that even though the bandwidth reducer can slightly increase the runtime for some non-optimized problems, in most cases the bandwidth reducer should significantly decrease the runtimes of non-optimized problems.

For criteria two, the results are similar to that for criteria one, namely, it was first shown that the bandwidth reducer does not increase the disk storage requirements needed for erasable scratch files of optimized problems when run on the blocked out-of-core solver. Second, it was shown that even though the bandwidth reducer can slightly increase the disk storage requirements for some non-optimized problems when run on the blocked out-of-core solver, in most cases the bandwidth reducer can significantly

decrease the disk storage requirements of non-optimized problems. (Criteria two does not apply to the current in-core solver.)

For criteria three the results indicate that the bandwidth reducer will enable the in-core solver to run some problems that would not normally run on the original solver alone. Thus, the in-core solver with the bandwidth reducer has a higher maximum problem size limit. For both configurations of the blocked out-of-core solver, the maximum problem size limit is dictated by the other parameters of the code itself and, therefore, they both have the same maximum problem size limit.

Effectiveness of the Frontwidth Reducer

From the comparisons of the frontal solver both with and without frontwidth reduction, it is concluded that the frontwidth reducer should be utilized for this solver. The reasons for this conclusion are again supported by the application of the comparison criteria.

For criteria one, first it was shown that the frontwidth reduction scheme can significantly improve the runtimes on non-optimized problems. In fact, it was shown that the frontwidth reduction scheme more closely optimizes a model for its applicable solution algorithm than the bandwidth reduction scheme does for its applicable solution algorithm. Second, it was shown that the frontwidth reducer does not significantly increase the runtimes on already optimized models.

For criteria two, it was shown that the frontwidth reduction scheme can significantly decrease the disk storage requirements

for erasable scratch files on non-optimized problems and no increase occurs when it is used on optimized models.

For criteria three, it was shown that even though more core area is used to incorporate the frontwidth reduction technique, which in turn reduces the maximum allowable frontwidth, the application of the frontwidth reduction technique will actually solve more and larger problems.

Chosen Out-Of-Core Solver

For the final comparison of the blocked out-of-core solver with bandwidth reduction and the frontal solver with frontwidth reduction, the conclusion is that the blocked out-of-core solver with bandwidth reduction should be the one to replace the current in-core solver. The reason for this conclusion rests on the balance of the comparison criteria.

For criteria one, the two solvers were shown to be about equal for the total runtime for the solution of the first loadcase. However, when the second or subsequent loadcases were examined, the blocked out-of-core solver was shown to be superior in all cases. In addition, when the total runtimes for the solution of both loadcases were examined, the blocked out-of core solver was virtually equal or faster in ten of the eleven test models. Also, if more loadcases are solved, the differences in runtimes would increase proportionately. Thus, for a wide variety of problems, especially when solving multiple loadcases, the blocked out-of-core

solver with bandwidth reduction would be faster than the frontal solver with frontwidth reduction.

For criteria two, the frontal solver was shown to be superior in all cases. But, since disk space is usually readily available and cheap, this criteria is considered to be less important than the total runtimes.

For criteria three, the blocked out-of-core solver has shown to be superior since the maximum size of solvable problems is limited by other parameters of the CAEFRAME code and the configuration of the solver. As shown, the blocked out-of-core solver can in theory solve a problem that has as many as all the 5994 degrees of freedom allowed by the CAEFRAME code. In fact all of these 5994 degrees of freedom can be connected. However, even though the frontal solver can in theory solve a problem with the allowable 5994 degrees of freedom, if more than 190 of the degrees of freedom are required to be in the front at any one time during the solution, the problem will not run because the maximum allowable frontwidth will be exceeded.

It should be noted that the results and stated conclusions concerning the blocked out-of-core solver with bandwidth reduction versus the frontal solver with frontwidth reduction may not be the same if these two solvers are run on other systems. The constraint of the current system that dictated the stated conclusions is the available core area. If more core area were available, the maximum allowable front could be expanded on the frontal solver and multiple loadcases could be reduced and solved simultaneously. Therefore,

the frontal solver might be shown to be superior in total runtimes, as well as in disk storage requirements. Also, the maximum solvable problem size limit would be increased for the frontal solver if more core was available. In that case, the conclusion would be to use the frontal solver with frontwidth reduction.

Final Recommendation

As shown through the process of comparison and elimination of the various configurations of the two out-of-core solvers, if the current in-core solver is to be replaced, it should be replaced by the blocked out-of-core solver with bandwidth reduction. This solver proved to have the best blend of solution runtime and disk storage requirements over a wide variety of problem types, especially for multiple loadcases. In addition, this solver has the the highest maximum solvable problem size limit of any of the solvers examined (in-core or out-of-core).

However, the current in-core solver with the bandwidth reduction scheme added to it, was shown to be almost as fast as the original in-core solver and it did extend the maximum size limit of solvable problems, which was the objective of the project. This extended range of problem size, along with the quickness of in-core operation, makes the in-core solver with bandwidth reduction a very attractive alternative to the original in-core solver alone or any out-of-core solver. But, this solver still has a maximum problem size limit that is less than desirable, and thus, should not alone replace the current in-core solver.

Therefore, the final recommendation is to replace the current in-core solver with both the blocked out-of-core solver with bandwidth reduction, *and* the current in-core solver with bandwidth reduction.

Instead of installing both solvers directly, they both should be installed as overlays. Thus, after the nodal renumbering is performed, a quick calculation of the maximum size limit of the stiffness vector can be made by simply multiplying the reordered maximum bandwidth of the structure, (converted from a nodal basis to a degree of freedom basis), by the total number of *free* degrees of freedom. If a stiffness vector of this size can still fit in the core, then the in-core solver is utilized to solve the problem. If not, the blocked out-of core solver is utilized.

The combination of these two solvers will offer the efficiency of the current in-core solver on small problems, and the versatility of an out-of-core solver to solve very large problems. However, to incorporate both solvers simultaneously as described, the controlling program of the CAEFRAME code would have to be modified. The author did not have access to this code and, therefore, this combination of both solvers could not be examined. But, the engineers at SST Inc. should be able to implement this combination and evaluate its merits.

LIST OF REFERENCES

LIST OF REFERENCES

1. Akin, J. E., Application and Implementation of Finite Element Methods, Academic Press, London, 1982.
2. Meyers, J.V., Matrix Analysis of Structures, Harper and Row, New York, 1983.
3. Bathe, K. J., Finite Element Procedures in Engineering Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
4. "CAEFRAME: a Module of the CADVANTAGE Library of Engineering Design Software." SST Systems Inc., Cupertino, California, 1984.
5. James, M. L., Smith, G. M., and Wolkoff, J. C., Applied Numerical Methods for Digital Computers with Fortran and CSMP, 2nd. Ed., Harper and Row, New York, 1977.
6. Wilson, E. L., Bathe, K. L., and Doherty, W. P., "Direct Solution of Large Systems of Linear Equations." *Computers and Structures*, Vol. 4, pp. 363-372, 1974.
7. Felippa, C. A., "Solutions of Linear Equations with Skyline-Stored Symmetric Matrix." *Computers and Structures*, Vol. 5, pp. 13-29, 1975.
8. Mondkar, D. P., and Powell, G. H., "Towards Optimal In-Core Equation Solving." *Computers and Structures*, Vol. 4, pp. 531-548, 1974.
9. Mondkar, D. P., and Powell, G. H., "Large Capacity Equation Solver for Structural Analysis." *Computers and Structures*, Vol. 4, pp. 699-728, 1974.
10. Dessi, M., and Priolo, P., "An Active Areas Bandsolver for the Resolution of Moderately Large Systems of Equations." *Computers and Structures*, Vol. 11, pp. 489-506, 1980.
11. Hinton, E., and Owen, D. R. J., Finite Element Programming, Academic Press, London, 1977.
12. Irons, B. M., "A Frontal Solution Algorithm Program for Finite Element Analysis." *International Journal for Numerical Methods in Engineering*, Vol. 2, pp. 5-32, 1970.
13. Pina, H. L. G., "An Algorithm for Frontwidth Reduction." *International Journal for Numerical Methods in Engineering*, Vol. 17, pp. 1539-1546, 1981.

14. Collins, R. J., "Bandwidth Reduction by Automatic Renumbering." *International Journal for Numerical Methods in Engineering*, Vol. 6, pp. 345-356, 1973.
15. Cuthill, E., and McKee, J., "Reducing the Bandwidth of Sparse Symmetric Matrices." *ACM National Conference*, San Francisco, pp. 157-172, 1969.
16. Rosen, R., "Matrix Bandwidth Minimization." *ACM National Conference*, Princeton, New Jersey, pp. 585-595, 1968.
17. Akyuz, F. A., and Utku, S., "An Automatic Relabeling Scheme for Bandwidth Minimization of Stiffness Matrices." *Journal of American Aeronautics and Astronautics*, Vol. 6, pp. 728-730, 1968.
18. Grooms, H. R., "Algorithm for Matrix Bandwidth Reduction." *ASCE Journal, Structural Division*, Vol. 98, pp. 203-214, 1972.
19. Sloan, S. W., and Randolph, M. F., "Automatic Reordering for Finite Element Analysis with Frontal Solution Schemes." *International Journal for Numerical Methods in Engineering*, Vol. 19, pp. 1153-1181, 1983.
20. King, I. P., "An Automatic Reordering Scheme for Simultaneous Equations Derived from Network Systems." *International Journal for Numerical Methods in Engineering*, Vol. 2, pp. 523-533, 1970.
21. Aiken, J. E., and Pardue, R. M., "Element Resequencing for Frontal Solutions." in The Mathematics of Finite Elements and Applications, Whiteman, J. R., Ed., Academic Press, London, pp. 535-541, 1975.
22. Razzaque, A., "Automatic Reduction of Frontwidth for Finite Element Analysis." *International Journal for Numerical Methods in Engineering*, Vol. 15, pp. 1315-1324, 1980.

APPENDICES

APPENDIX A

SAMPLE PROBLEM SOLUTION STATISTICS

Original solver WITHOUT bandwidth reduction

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	3.41	3.41
Read element data	3.57	6.98
Read restraint data	2.03	9.01
Calculate stiffness matrix	34.55	43.56
Forward reduction of stiffness matrix	17.30	60.86

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	5.82	66.68
Back-substitution	4.34	71.02
Calculate displacements	2.09	73.11
Calculate stresses	10.93	84.04
Calculate support reactions	1.70	85.74

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.46	89.20
Back-substitution	4.34	93.54
Calculate displacements	2.09	95.63
Calculate stresses	10.93	106.56
Calculate support reactions	1.65	108.21

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Length of element arrays . . .	2152
Length of nodal arrays	2409
Length of stiffness vector . .	3446

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	3.41	3.41
Read element data	3.57	6.98
Read restraint data	2.14	9.12
Reorder nodes to reduce bandwidth	0.55	9.67
Calculate stiffness matrix	34.55	44.22
Forward reduction of stiffness matrix	17.30	61.52

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	5.87	67.39
Back-substitution	4.34	71.73
Calculate displacements	2.09	73.82
Calculate stresses	10.93	84.75
Calculate support reactions	1.65	86.40

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.46	89.86
Back-substitution	4.34	94.20
Calculate displacements	2.08	96.28
Calculate stresses	10.93	107.21
Calculate support reactions	1.71	108.92

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Length of element arrays . . .	2152
Length of nodal arrays	2442
Length of stiffness vector . .	3446
Original Connectivity Range (Max)	4
Reduced Connectivity Range (Max)	4

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	3.41	3.41
Read element data	3.62	7.03
Read section and material data	0.44	7.47
Read restraint data	2.09	9.56
Calculate stiffness vector blocks	39.05	48.61
Reduce stiffness vector blocks	25.76	74.37

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	6.59	80.96
Back-substitution	17.58	98.54
Calculate displacements	2.14	100.68
Calculate stresses	10.93	111.61
Calculate support reactions	1.70	113.31

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.68	116.99
Back-substitution	16.97	133.96
Calculate displacements	2.15	136.11
Calculate stresses	11.04	147.15
Calculate support reactions	1.64	148.79

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Number of stiffness terms . . .	3446
Number of stiffness vector blocks	1

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	28004	5-27-88	3:10p
REDUCA	DMP	28004	5-27-88	3:10p
DIAGAD	DMP	338	5-27-88	3:10p
PIVOTT	DMP	1338	5-27-88	3:10p

Blocked out-of-core solver WITH bandwidth reduction

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	3.30	3.30
Read element data	3.62	6.92
Read section and material data	0.44	7.36
Read restraint data	1.92	9.28
Reorder nodes to reduce bandwidth	0.50	9.78
Calculate stiffness vector blocks	40.26	50.04
Reduce stiffness vector blocks	25.81	75.85

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	6.59	82.44
Back-substitution	17.69	100.13
Calculate displacements	2.14	102.27
Calculate stresses	10.99	113.26
Calculate support reactions	1.98	115.24

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.62	118.86
Back-substitution	17.08	135.94
Calculate displacements	2.14	138.08
Calculate stresses	11.04	149.12
Calculate support reactions	1.71	150.83

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Number of stiffness terms	3446
Number of stiffness vector blocks	1

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	28004	5-27-88	3:12p
REDUCA	DMP	28004	5-27-88	3:13p
DIAGAD	DMP	338	5-27-88	3:12p
PIVOTT	DMP	1338	5-27-88	3:13p

Frontal solver WITHOUT frontwidth reduction

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	3.52	3.52
Read element data	3.51	7.03
Read section and material data	0.44	7.47
Read restraint data	2.03	9.50
Determine element order	0.28	9.78
Do pre-front housekeeping	10.21	19.99

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	7.86	27.85
Begin wavefront solver	0.16	28.01
Initialize variables	3.68	31.69
Wavefront reduction	64.10	95.79
Back substitution	16.64	112.43
Calculate displacements	2.20	114.63
Calculate stresses	10.93	125.56
Calculate support reactions	1.65	127.21

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	4.39	131.60
Begin wavefront solver	0.17	131.77
Initialize variables	0.33	132.10
Wavefront reduction	13.51	145.61
Back substitution	18.67	164.28
Calculate displacements	2.14	166.42
Calculate stresses	10.94	177.36
Calculate support reactions	1.64	179.00

Number of elements	20
Number of nodes	33
Number of degrees of freedom	198
Number of restraints	34
Largest frontwidth	30

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	48494	5-27-88	9:02p
SCRA2	DMP	1982	5-27-88	9:03p

Frontal solver WITH frontwidth reduction

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	3.68	3.68
Read element data	3.62	7.30
Read section and material data	0.44	7.74
Read restraint data	2.14	9.88
Determine element order	0.94	10.82
Do pre-front housekeeping	10.21	21.03

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	8.24	29.27
Begin wavefront solver	0.17	29.44
Initialize variables	2.80	32.24
Wavefront reduction	64.21	96.45
Back substitution	16.69	113.14
Calculate displacements	2.09	115.23
Calculate stresses	10.99	126.22
Calculate support reactions	1.64	127.86

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	4.40	132.26
Begin wavefront solver	0.22	132.48
Initialize variables	0.22	132.70
Wavefront reduction	13.56	146.26
Back substitution	18.73	164.99
Calculate displacements	2.15	167.14
Calculate stresses	10.93	178.07
Calculate support reactions	1.64	179.71

Number of elements	20
Number of nodes	33
Number of degrees of freedom	198
Number of restraints	34
Original bandwidth	4
Reduced bandwidth	4
Largest frontwidth	30

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	48494	5-27-88	9:06p
SCRA2	DMP	1982	5-27-88	9:06p

Original solver WITHOUT bandwidth reduction

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	3.35	3.35
Read element data	3.57	6.92
Read restraint data	2.03	8.95
Calculate stiffness matrix	35.27	44.22
Forward reduction of stiffness matrix	76.29	120.51

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	5.77	126.28
Back-substitution	6.42	132.70
Calculate displacements	2.09	134.79
Calculate stresses	10.98	145.77
Calculate support reactions	1.65	147.42

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.41	150.83
Back-substitution	6.48	157.31
Calculate displacements	2.09	159.40
Calculate stresses	10.93	170.33
Calculate support reactions	1.64	171.97

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Length of element arrays	2152
Length of nodal arrays	2409
Length of stiffness vector . . .	7445

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	3.18	3.18
Read element data	3.63	6.81
Read restraint data	2.20	9.01
Reorder nodes to reduce bandwidth	0.60	9.61
Calculate stiffness matrix	34.60	44.21
Forward reduction of stiffness matrix	18.18	62.39

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	5.88	68.27
Back-substitution	4.40	72.67
Calculate displacements	2.08	74.75
Calculate stresses	11.15	85.90
Calculate support reactions	1.65	87.55

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.52	91.07
Back-substitution	4.33	95.40
Calculate displacements	2.09	97.49
Calculate stresses	10.93	108.42
Calculate support reactions	1.70	110.12

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Length of element arrays	2152
Length of nodal arrays	2442
Length of stiffness vector . . .	3546
Original Connectivity Range (Max)	12
Reduced Connectivity Range (Max)	5

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	3.41	3.41
Read element data	3.51	6.92
Read section and material data	0.50	7.42
Read restraint data	2.09	9.51
Calculate stiffness vector blocks	45.53	55.04
Reduce stiffness vector blocks	85.57	140.61

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	6.70	147.31
Back-substitution	36.09	183.40
Calculate displacements	2.20	185.60
Calculate stresses	11.04	196.64
Calculate support reactions	1.70	198.34

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.68	202.02
Back-substitution	36.03	238.05
Calculate displacements	2.20	240.25
Calculate stresses	10.98	251.23
Calculate support reactions	1.71	252.94

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Number of stiffness terms	7445
Number of stiffness vector blocks	1

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	60496	5-27-88	3:27p
REDUCA	DMP	60496	5-27-88	3:29p
DIAGAD	DMP	338	5-27-88	3:27p
PIVOTT	DMP	1338	5-27-88	3:29p

Blocked out-of-core solver WITH bandwidth reduction

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	3.35	3.35
Read element data	3.62	6.97
Read section and material data	0.44	7.41
Read restraint data	1.92	9.33
Reorder nodes to reduce bandwidth	0.66	9.99
Calculate stiffness vector blocks	40.48	50.47
Reduce stiffness vector blocks	26.92	77.39

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	6.75	84.14
Back-substitution	18.46	102.60
Calculate displacements	2.19	104.79
Calculate stresses	11.04	115.83
Calculate support reactions	1.76	117.59

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	3.68	121.27
Back-substitution	17.69	138.96
Calculate displacements	2.20	141.16
Calculate stresses	11.04	152.20
Calculate support reactions	1.70	153.90

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Number of equations	164
Number of stiffness terms	3546
Number of stiffness vector blocks	1

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	28816	5-27-88	3:31p
REDUCA	DMP	28816	5-27-88	3:32p
DIAGAD	DMP	338	5-27-88	3:31p
PIVOTT	DMP	1338	5-27-88	3:32p

Frontal solver WITHOUT frontwidth reduction

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	3.57	3.57
Read element data	3.63	7.20
Read section and material data	0.44	7.64
Read restraint data	2.03	9.67
Determine element order	0.22	9.89
Do pre-front housekeeping	10.27	20.16

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	8.07	28.23
Begin wavefront solver	0.22	28.45
Initialize variables	3.63	32.08
Wavefront reduction	121.44	153.52
Back substitution	37.46	190.98
Calculate displacements	2.19	193.17
Calculate stresses	10.93	204.10
Calculate support reactions	1.71	205.81

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	4.39	210.20
Begin wavefront solver	0.22	210.42
Initialize variables	0.28	210.70
Wavefront reduction	29.33	240.03
Back substitution	39.82	279.85
Calculate displacements	2.08	281.93
Calculate stresses	10.99	292.92
Calculate support reactions	1.59	294.51

Number of elements	20
Number of nodes	33
Number of degrees of freedom . .	198
Number of restraints	34
Largest frontwidth	78

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	108470	5-27-88	9:34p
SCRA2	DMP	1982	5-27-88	9:36p

Frontal solver WITH frontwidth reduction

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	3.79	3.79
Read element data	3.63	7.42
Read section and material data	0.49	7.91
Read restraint data	2.09	10.00
Determine element order	0.99	10.99
Do pre-front housekeeping	10.27	21.26

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	8.40	29.66
Begin wavefront solver	0.17	29.83
Initialize variables	2.80	32.63
Wavefront reduction	64.31	96.94
Back substitution	17.58	114.52
Calculate displacements	2.25	116.77
Calculate stresses	10.93	127.70
Calculate support reactions	1.71	129.41

Loadcase # 2 Title: Thermal Load - dT=100 deg.F.

Read load data	4.39	133.80
Begin wavefront solver	0.22	134.02
Initialize variables	0.27	134.29
Wavefront reduction	14.12	148.41
Back substitution	19.99	168.40
Calculate displacements	2.25	170.65
Calculate stresses	11.15	181.80
Calculate support reactions	1.60	183.40

Number of elements	20
Number of nodes	33
Number of degrees of freedom	198
Number of restraints	34
Original bandwidth	12
Reduced bandwidth	5
Largest frontwidth	30

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	48494	5-27-88	9:38p
SCRA2	DMP	1982	5-27-88	9:39p

Original solver WITHOUT bandwidth reduction

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	4.39	4.39
Read element data	6.59	10.98
Read restraint data	2.36	13.34
Calculate stiffness matrix	68.88	82.22
Forward reduction of stiffness matrix	68.27	150.49

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	10.60	161.09
Back-substitution	12.25	173.34
Calculate displacements	3.35	176.69
Calculate stresses	21.53	198.22
Calculate support reactions	1.93	200.15

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	5.93	206.08
Back-substitution	12.25	218.33
Calculate displacements	3.24	221.57
Calculate stresses	21.53	243.10
Calculate support reactions	1.87	244.97

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Length of element arrays	4292
Length of nodal arrays	4029
Length of stiffness vector . . .	9504

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	4.23	4.23
Read element data	6.64	10.87
Read restraint data	2.42	13.29
Reorder nodes to reduce bandwidth	0.99	14.28
Calculate stiffness matrix	68.87	83.15
Forward reduction of stiffness matrix	68.28	151.43

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	10.65	162.08
Back-substitution	12.25	174.33
Calculate displacements	3.29	177.62
Calculate stresses	21.59	199.21
Calculate support reactions	1.87	201.08

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	5.98	207.06
Back-substitution	12.25	219.31
Calculate displacements	3.24	222.55
Calculate stresses	21.53	244.08
Calculate support reactions	1.98	246.06

Number of elements	40
Number of nodes	55
Number of degrees of freedom	330
Number of restraints	42
Number of equations	288
Length of element arrays	4292
Length of nodal arrays	4084
Length of stiffness vector	9504
Original Connectivity Range (Max)	6
Reduced Connectivity Range (Max)	6

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	4.56	4.56
Read element data	6.65	11.21
Read section and material data	0.38	11.59
Read restraint data	2.58	14.17
Calculate stiffness vector blocks	89.64	103.81
Reduce stiffness vector blocks	98.59	202.40

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	11.59	213.99
Back-substitution	49.16	263.15
Calculate displacements	3.46	266.61
Calculate stresses	21.86	288.47
Calculate support reactions	1.92	290.39

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.32	296.71
Back-substitution	46.91	343.62
Calculate displacements	3.40	347.02
Calculate stresses	21.81	368.83
Calculate support reactions	1.92	370.75

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Number of stiffness terms . . .	9504
Number of stiffness vector blocks	2

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	77228	5-27-88	3:39p
REDUCA	DMP	77228	5-27-88	3:41p
DIAGAD	DMP	592	5-27-88	3:39p
PIVOTT	DMP	2348	5-27-88	3:41p

Blocked out-of-core solver WITH bandwidth reduction

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	4.45	4.45
Read element data	6.70	11.15
Read section and material data	0.38	11.53
Read restraint data	2.26	13.79
Reorder nodes to reduce bandwidth	0.98	14.77
Calculate stiffness vector blocks	91.57	106.34
Reduce stiffness vector blocks	97.76	204.10

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	11.59	215.69
Back-substitution	47.73	263.42
Calculate displacements	3.41	266.83
Calculate stresses	21.75	288.58
Calculate support reactions	2.09	290.67

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.15	296.82
Back-substitution	46.13	342.95
Calculate displacements	3.41	346.36
Calculate stresses	21.86	368.22
Calculate support reactions	1.92	370.14

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Number of stiffness terms . . .	9504
Number of stiffness vector blocks	2

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	77228	5-27-88	3:45p
REDUCA	DMP	77228	5-27-88	3:47p
DIAGAD	DMP	592	5-27-88	3:45p
PIVOTT	DMP	2348	5-27-88	3:47p

Frontal solver WITHOUT frontwidth reduction

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	4.39	4.39
Read element data	6.70	11.09
Read section and material data	0.44	11.53
Read restraint data	2.53	14.06
Determine element order	0.44	14.50
Do pre-front housekeeping	33.50	48.00

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	11.92	59.92
Begin wavefront solver	0.17	60.09
Initialize variables	3.68	63.77
Wavefront reduction	149.94	213.71
Back substitution	39.60	253.31
Calculate displacements	3.35	256.66
Calculate stresses	21.76	278.42
Calculate support reactions	1.92	280.34

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.37	286.71
Begin wavefront solver	0.22	286.93
Initialize variables	0.27	287.20
Wavefront reduction	32.79	319.99
Back substitution	43.56	363.55
Calculate displacements	3.35	366.90
Calculate stresses	21.53	388.43
Calculate support reactions	1.81	390.24

Number of elements	40
Number of nodes	55
Number of degrees of freedom	330
Number of restraints	42
Number of equations	288
Largest frontwidth	42

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	111950	5-27-88	9:44p
SCRA2	DMP	3302	5-27-88	9:45p

Frontal solver WITH frontwidth reduction

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

	Time	Cum. Time
Read nodal data	4.55	4.55
Read element data	6.71	11.26
Read section and material data	0.43	11.69
Read restraint data	2.64	14.33
Determine element order	2.03	16.36
Do pre-front housekeeping	33.56	49.92

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	12.20	62.12
Begin wavefront solver	0.22	62.34
Initialize variables	2.74	65.08
Wavefront reduction	149.89	214.97
Back substitution	39.39	254.36
Calculate displacements	3.35	257.71
Calculate stresses	21.53	279.24
Calculate support reactions	1.92	281.16

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.54	287.70
Begin wavefront solver	0.21	287.91
Initialize variables	0.28	288.19
Wavefront reduction	32.63	320.82
Back substitution	43.28	364.10
Calculate displacements	3.29	367.39
Calculate stresses	21.53	388.92
Calculate support reactions	1.87	390.79

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Original bandwidth	6
Reduced bandwidth	6
Largest frontwidth	42

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	111950	5-27-88	9:50p
SCRA2	DMP	3302	5-27-88	9:52p

Original solver WITHOUT bandwidth reduction

Model: PLT10X4.STR Title: 10 x 4 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	4.40	4.40
Read element data	6.59	10.99
Read restraint data	2.41	13.40
Calculate stiffness matrix	70.03	83.43
Forward reduction of stiffness matrix	190.82	274.25

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	10.65	284.90
Back-substitution	15.76	300.66
Calculate displacements	3.30	303.96
Calculate stresses	21.53	325.49
Calculate support reactions	1.98	327.47

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	5.93	333.40
Back-substitution	15.76	349.16
Calculate displacements	3.30	352.46
Calculate stresses	21.53	373.99
Calculate support reactions	1.87	375.86

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Length of element arrays . . .	4292
Length of nodal arrays	4029
Length of stiffness vector . .	16183

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: PLT10X4.STR Title: 10 x 4 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	4.28	4.28
Read element data	6.59	10.87
Read restraint data	2.42	13.29
Reorder nodes to reduce bandwidth	1.32	14.61
Calculate stiffness matrix	68.99	83.60
Forward reduction of stiffness matrix	78.48	162.08

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	10.71	172.79
Back-substitution	12.58	185.37
Calculate displacements	3.30	188.67
Calculate stresses	21.58	210.25
Calculate support reactions	1.93	212.18

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	5.93	218.11
Back-substitution	12.63	230.74
Calculate displacements	3.24	233.98
Calculate stresses	21.59	255.57
Calculate support reactions	1.92	257.49

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Length of element arrays	4292
Length of nodal arrays	4084
Length of stiffness vector . . .	10179
Original Connectivity Range (Max)	12
Reduced Connectivity Range (Max)	9

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: PLT10X4.STR Title: 10 x 4 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	4.51	4.51
Read element data	6.64	11.15
Read section and material data	0.39	11.54
Read restraint data	2.58	14.12
Calculate stiffness vector blocks	112.93	127.05
Reduce stiffness vector blocks	236.07	363.12

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	11.42	374.54
Back-substitution	79.15	453.69
Calculate displacements	3.46	457.15
Calculate stresses	21.75	478.90
Calculate support reactions	1.98	480.88

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.31	487.19
Back-substitution	79.09	566.28
Calculate displacements	3.41	569.69
Calculate stresses	21.86	591.55
Calculate support reactions	1.92	593.47

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Number of stiffness terms . . .	16183
Number of stiffness vector blocks	3

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	131498	5-27-88	3:57p
REDUCA	DMP	131498	5-27-88	4:01p
DIAGAD	DMP	596	5-27-88	3:57p
PIVOTT	DMP	2352	5-27-88	4:02p

Blocked out-of-core solver WITH bandwidth reduction

Model: PLT10X4.STR Title: 10 x 4 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	4.51	4.51
Read element data	6.64	11.15
Read section and material data	0.44	11.59
Read restraint data	2.26	13.85
Reorder nodes to reduce bandwidth	1.26	15.11
Calculate stiffness vector blocks	92.33	107.44
Reduce stiffness vector blocks	108.64	216.08

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	11.81	227.89
Back-substitution	52.51	280.40
Calculate displacements	3.35	283.75
Calculate stresses	21.86	305.61
Calculate support reactions	2.03	307.64

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.21	313.85
Back-substitution	49.76	363.61
Calculate displacements	3.41	367.02
Calculate stresses	21.86	388.88
Calculate support reactions	1.92	390.80

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Number of stiffness terms . . .	10179
Number of stiffness vector blocks	2

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	82712	5-27-88	4:07p
REDUCA	DMP	82712	5-27-88	4:09p
DIAGAD	DMP	592	5-27-88	4:07p
PIVOTT	DMP	2348	5-27-88	4:09p

Frontal solver WITHOUT frontwidth reduction

Model: PLT10X4.STR Title: 10 x 4 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	4.45	4.45
Read element data	6.59	11.04
Read section and material data	0.39	11.43
Read restraint data	2.58	14.01
Determine element order	0.44	14.45
Do pre-front housekeeping	33.50	47.95

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	11.76	59.71
Begin wavefront solver	0.22	59.93
Initialize variables	3.62	63.55
Wavefront reduction	233.76	297.31
Back substitution	61.90	359.21
Calculate displacements	3.41	362.62
Calculate stresses	21.59	384.21
Calculate support reactions	1.92	386.13

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.42	392.55
Begin wavefront solver	0.17	392.72
Initialize variables	0.33	393.05
Wavefront reduction	49.65	442.70
Back substitution	65.58	508.28
Calculate displacements	3.30	511.58
Calculate stresses	21.53	533.11
Calculate support reactions	1.81	534.92

Number of elements	40
Number of nodes	55
Number of degrees of freedom	330
Number of restraints	42
Number of equations	288
Largest frontwidth	78

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	175982	5-27-88	9:59p
SCRA2	DMP	3302	5-27-88	10:01p

Frontal solver WITH frontwidth reduction

Model: PLT10X4.STR Title: 10 x 4 Mesh Plates - Inefficient Ordering

	Time	Cum. Time
Read nodal data	4.61	4.61
Read element data	6.59	11.20
Read section and material data	0.44	11.64
Read restraint data	2.58	14.22
Determine element order	2.37	16.59
Do pre-front housekeeping	33.55	50.14

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	12.03	62.17
Begin wavefront solver	0.22	62.39
Initialize variables	2.75	65.14
Wavefront reduction	152.53	217.67
Back substitution	39.93	257.60
Calculate displacements	3.40	261.00
Calculate stresses	21.53	282.53
Calculate support reactions	1.93	284.46

Loadcase # 2 Title: Thermal Load - dT = 100 deg.F.

Read load data	6.59	291.05
Begin wavefront solver	0.16	291.21
Initialize variables	0.33	291.54
Wavefront reduction	32.68	324.22
Back substitution	43.50	367.72
Calculate displacements	3.30	371.02
Calculate stresses	21.53	392.55
Calculate support reactions	1.87	394.42

Number of elements	40
Number of nodes	55
Number of degrees of freedom . .	330
Number of restraints	42
Number of equations	288
Original bandwidth	12
Reduced bandwidth	9
Largest frontwidth	54

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	113498	5-27-88	10:06p
SCRA2	DMP	3302	5-27-88	10:08p

Original solver WITHOUT bandwidth reduction

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.30	7.30
Read element data	16.04	23.34
Read restraint data	1.21	24.55

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	24
Number of equations	702
Length of element arrays	10712
Length of nodal arrays	8931
Length of stiffness vector . . .	48285

Problem size too large. Max. Stiffness Vector Length = 30000

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.36	7.36
Read element data	15.93	23.29
Read restraint data	1.10	24.39
Reorder nodes to reduce bandwidth	3.73	28.12

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	24
Number of equations	702
Length of element arrays	10712
Length of nodal arrays	9052
Length of stiffness vector . . .	48285
Original Connectivity Range (Max)	12
Reduced Connectivity Range (Max)	12

Problem size too large. Max. Stiffness Vector Length = 30000

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.69	7.69
Read element data	15.82	23.51
Read section and material data	0.44	23.95
Read restraint data	1.86	25.81
Calculate stiffness vector blocks	341.86	367.67
Reduce stiffness vector blocks	819.60	1187.27

Loadcase # 1 Title: 1000 Lb load at center

Read load data	3.13	1190.40
Back-substitution	220.58	1410.98
Calculate displacements	7.25	1418.23
Calculate stresses	54.10	1472.33
Calculate support reactions	0.72	1473.05

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	25.87	1498.92
Back-substitution	242.88	1741.80
Calculate displacements	7.41	1749.21
Calculate stresses	54.60	1803.81
Calculate support reactions	0.71	1804.52

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	24
Number of equations	702
Number of stiffness terms	48285
Number of stiffness vector blocks	7

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	392340	5-27-88	4:20p
REDUCA	DMP	392340	5-27-88	4:33p
DIAGAD	DMP	1448	5-27-88	4:20p
PIVOTT	DMP	5726	5-27-88	4:35p

Blocked out-of-core solver WITH bandwidth reduction

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.53	7.53
Read element data	15.93	23.46
Read section and material data	0.49	23.95
Read restraint data	0.88	24.83
Reorder nodes to reduce bandwidth	3.73	28.56
Calculate stiffness vector blocks	343.56	372.12
Reduce stiffness vector blocks	817.46	1189.58

Loadcase # 1 Title: 1000 Lb load at center

Read load data	2.91	1192.49
Back-substitution	218.60	1411.09
Calculate displacements	7.25	1418.34
Calculate stresses	54.16	1472.50
Calculate support reactions	0.77	1473.27

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	25.81	1499.08
Back-substitution	240.85	1739.93
Calculate displacements	7.42	1747.35
Calculate stresses	54.59	1801.94
Calculate support reactions	0.72	1802.66

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	24
Number of equations	702
Number of stiffness terms . . .	48285
Number of stiffness vector blocks	7

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	392340	5-27-88	4:50p
REDUCA	DMP	392340	5-27-88	5:04p
DIAGAD	DMP	1448	5-27-88	4:50p
PIVOTT	DMP	5726	5-27-88	5:05p

Frontal solver WITHOUT frontwidth reduction

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.52	7.52
Read element data	15.98	23.50
Read section and material data	0.44	23.94
Read restraint data	1.93	25.87
Determine element order	1.20	27.07
Do pre-front housekeeping	182.96	210.03

Loadcase # 1 Title: 1000 Lb load at center

Read load data	3.13	213.16
Begin wavefront solver	0.22	213.38
Initialize variables	3.68	217.06
Wavefront reduction	645.98	863.04
Back substitution	165.93	1028.97
Calculate displacements	7.19	1036.16
Calculate stresses	53.61	1089.77
Calculate support reactions	0.66	1090.43

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	26.04	1116.47
Begin wavefront solver	0.22	1116.69
Initialize variables	0.33	1117.02
Wavefront reduction	127.70	1244.72
Back substitution	176.91	1421.63
Calculate displacements	7.03	1428.66
Calculate stresses	53.39	1482.05
Calculate support reactions	0.66	1482.71

Number of elements	100
Number of nodes	121
Number of degrees of freedom	726
Number of restraints	24
Largest frontwidth	78

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	441398	5-27-88	10:24p
SCRA2	DMP	7262	5-27-88	10:30p

Frontal solver WITH frontwidth reduction

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.63	7.63
Read element data	15.93	23.56
Read section and material data	0.44	24.00
Read restraint data	1.92	25.92
Determine element order	9.06	34.98
Do pre-front housekeeping	182.96	217.94

Loadcase # 1 Title: 1000 Lb load at center

Read load data	4.12	222.06
Begin wavefront solver	0.16	222.22
Initialize variables	2.86	225.08
Wavefront reduction	645.32	870.40
Back substitution	164.39	1034.79
Calculate displacements	7.09	1041.88
Calculate stresses	53.38	1095.26
Calculate support reactions	0.66	1095.92

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	25.93	1121.85
Begin wavefront solver	0.22	1122.07
Initialize variables	0.33	1122.40
Wavefront reduction	127.15	1249.55
Back substitution	175.05	1424.60
Calculate displacements	7.08	1431.68
Calculate stresses	53.61	1485.29
Calculate support reactions	0.66	1485.95

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	24
Original bandwidth	12
Reduced bandwidth	12
Largest frontwidth	78

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	441398	5-27-88	10:49p
SCRA2	DMP	7262	5-27-88	10:55p

Original solver WITHOUT bandwidth reduction

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.58	7.58
Read element data	15.82	23.40
Read restraint data	3.46	26.86
Calculate stiffness matrix	169.67	196.53
Forward reduction of stiffness matrix	319.11	515.64

Loadcase # 1 Title: 1000 Lb load at center

Read load data	1.98	517.62
Back-substitution	34.66	552.28
Calculate displacements	6.81	559.09
Calculate stresses	53.39	612.48
Calculate support reactions	2.74	615.22

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	25.27	640.49
Back-substitution	34.66	675.15
Calculate displacements	6.92	682.07
Calculate stresses	53.33	735.40
Calculate support reactions	2.69	738.09

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	240
Number of equations	486
Length of element arrays . . .	10712
Length of nodal arrays	8715
Length of stiffness vector . .	27621

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.47	7.47
Read element data	15.93	23.40
Read restraint data	3.46	26.86
Reorder nodes to reduce bandwidth	3.73	30.59
Calculate stiffness matrix	169.67	200.26
Forward reduction of stiffness matrix	319.11	519.37

Loadcase # 1 Title: 1000 Lb load at center

Read load data	2.04	521.41
Back-substitution	34.65	556.06
Calculate displacements	6.87	562.93
Calculate stresses	53.44	616.37
Calculate support reactions	2.80	619.17

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	25.05	644.22
Back-substitution	34.66	678.88
Calculate displacements	6.86	685.74
Calculate stresses	53.39	739.13
Calculate support reactions	2.69	741.82

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	240
Number of equations	486
Length of element arrays	10712
Length of nodal arrays	8836
Length of stiffness vector . . .	27621
Original Connectivity Range (Max)	12
Reduced Connectivity Range (Max)	12

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.80	7.80
Read element data	15.81	23.61
Read section and material data	0.44	24.05
Read restraint data	4.07	28.12
Calculate stiffness vector blocks	263.75	291.87
Reduce stiffness vector blocks	392.28	684.15

Loadcase # 1 Title: 1000 Lb load at center

Read load data	2.96	687.11
Back-substitution	123.64	810.75
Calculate displacements	7.09	817.84
Calculate stresses	54.15	871.99
Calculate support reactions	2.75	874.74

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	25.92	900.66
Back-substitution	135.01	1035.67
Calculate displacements	7.20	1042.87
Calculate stresses	55.20	1098.07
Calculate support reactions	2.91	1100.98

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	240
Number of equations	486
Number of stiffness terms	27621
Number of stiffness vector blocks	4

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	224436	5-27-88	5:20p
REDUCA	DMP	224436	5-27-88	5:27p
DIAGAD	DMP	998	5-27-88	5:20p
PIVOTT	DMP	3960	5-27-88	5:27p

Blocked out-of-core solver WITH bandwidth reduction

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	8.02	8.02
Read element data	15.88	23.90
Read section and material data	0.49	24.39
Read restraint data	3.08	27.47
Reorder nodes to reduce bandwidth	3.73	31.20
Calculate stiffness vector blocks	266.23	297.43
Reduce stiffness vector blocks	392.27	689.70

Loadcase # 1 Title: 1000 Lb load at center

Read load data	3.02	692.72
Back-substitution	124.96	817.68
Calculate displacements	7.41	825.09
Calculate stresses	54.27	879.36
Calculate support reactions	3.13	882.49

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	26.15	908.64
Back-substitution	136.16	1044.80
Calculate displacements	7.25	1052.05
Calculate stresses	54.76	1106.81
Calculate support reactions	2.91	1109.72

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	240
Number of equations	486
Number of stiffness terms . . .	27621
Number of stiffness vector blocks	4

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	224436	5-27-88	5:39p
REDUCA	DMP	224436	5-27-88	5:45p
DIAGAD	DMP	998	5-27-88	5:39p
PIVOTT	DMP	3960	5-27-88	5:46p

Frontal solver WITHOUT frontwidth reduction

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.53	7.53
Read element data	15.81	23.34
Read section and material data	0.50	23.84
Read restraint data	4.06	27.90
Determine element order	1.21	29.11
Do pre-front housekeeping	182.96	212.07

Loadcase # 1 Title: 1000 Lb load at center

Read load data	3.40	215.47
Begin wavefront solver	0.17	215.64
Initialize variables	3.68	219.32
Wavefront reduction	552.33	771.65
Back substitution	160.00	931.65
Calculate displacements	6.92	938.57
Calculate stresses	53.39	991.96
Calculate support reactions	2.69	994.65

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	26.03	1020.68
Begin wavefront solver	0.17	1020.85
Initialize variables	0.33	1021.18
Wavefront reduction	121.66	1142.84
Back substitution	169.44	1312.28
Calculate displacements	6.87	1319.15
Calculate stresses	53.33	1372.48
Calculate support reactions	2.69	1375.17

Number of elements	100
Number of nodes	121
Number of degrees of freedom	726
Number of restraints	240
Largest frontwidth	78

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	441398	5-27-88	11:12p
SCRA2	DMP	7262	5-27-88	11:19p

Frontal solver WITH frontwidth reduction

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

	Time	Cum. Time
Read nodal data	7.69	7.69
Read element data	15.88	23.57
Read section and material data	0.49	24.06
Read restraint data	4.12	28.18
Determine element order	9.06	37.24
Do pre-front housekeeping	182.96	220.20

Loadcase # 1 Title: 1000 Lb load at center

Read load data	3.95	224.15
Begin wavefront solver	0.17	224.32
Initialize variables	2.85	227.17
Wavefront reduction	552.72	779.89
Back substitution	161.37	941.26
Calculate displacements	6.98	948.24
Calculate stresses	53.60	1001.84
Calculate support reactions	2.64	1004.48

Loadcase # 2 Title: Uniform Transvers Pressure P = 1 psi

Read load data	26.09	1030.57
Begin wavefront solver	0.22	1030.79
Initialize variables	0.33	1031.12
Wavefront reduction	121.66	1152.78
Back substitution	170.93	1323.71
Calculate displacements	6.86	1330.57
Calculate stresses	53.28	1383.85
Calculate support reactions	2.64	1386.49

Number of elements	100
Number of nodes	121
Number of degrees of freedom . .	726
Number of restraints	240
Original bandwidth	12
Reduced bandwidth	12
Largest frontwidth	78

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	441398	5-27-88	11:36p
SCRA2	DMP	7262	5-27-88	11:42p

Original solver WITHOUT bandwidth reduction

Model: TMS1.STR Title: Roof Structure for TMS Inc.

	Time	Cum. Time
Read nodal data	3.68	3.68
Read element data	12.57	16.25
Read restraint data	1.32	17.57
Calculate stiffness matrix	61.02	78.59
Forward reduction of stiffness matrix	63.11	141.70

Loadcase # 1 Title: Downward Pressure on Roof Elements 1-70=45 lb/SqF

Read load data	7.91	149.61
Back-substitution	9.50	159.11
Calculate displacements	2.86	161.97
Calculate stresses	27.08	189.05
Calculate support reactions	0.66	189.71

Loadcase # 2 Title: Deadweight Load

Read load data	4.89	194.60
Back-substitution	9.50	204.10
Calculate displacements	2.91	207.01
Calculate stresses	27.08	234.09
Calculate support reactions	0.60	234.69

Number of elements	78
Number of nodes	44
Number of degrees of freedom . .	264
Number of restraints	24
Number of equations	240
Length of element arrays	8419
Length of nodal arrays	3233
Length of stiffness vector . . .	8292

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: TMS1.STR Title: Roof Structure for TMS Inc.

	Time	Cum. Time
Read nodal data	3.62	3.62
Read element data	12.58	16.20
Read restraint data	1.48	17.68
Reorder nodes to reduce bandwidth	0.99	18.67
Calculate stiffness matrix	61.13	79.80
Forward reduction of stiffness matrix	71.41	151.21

Loadcase # 1 Title: Downward Pressure on Roof Elements 1-70=45 lb/SqF

Read load data	7.96	159.17
Back-substitution	9.78	168.95
Calculate displacements	2.85	171.80
Calculate stresses	27.08	198.88
Calculate support reactions	0.66	199.54

Loadcase # 2 Title: Deadweight Load

Read load data	4.95	204.49
Back-substitution	9.77	214.26
Calculate displacements	2.91	217.17
Calculate stresses	27.08	244.25
Calculate support reactions	0.61	244.86

Number of elements	78
Number of nodes	44
Number of degrees of freedom . .	264
Number of restraints	24
Number of equations	240
Length of element arrays . . .	8419
Length of nodal arrays	3277
Length of stiffness vector . .	8832
Original Connectivity Range (Max)	40
Reduced Connectivity Range (Max)	8

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: TMS1.STR Title: Roof Structure for TMS Inc.

	Time	Cum. Time
Read nodal data	3.95	3.95
Read element data	12.63	16.58
Read section and material data	0.88	17.46
Read restraint data	1.21	18.67
Calculate stiffness vector blocks	75.80	94.47
Reduce stiffness vector blocks	90.46	184.93

Loadcase # 1 Title: Downward Pressure on Roof Elements 1-70=45 lb/SqF

Read load data	8.68	193.61
Back-substitution	40.75	234.36
Calculate displacements	2.91	237.27
Calculate stresses	27.36	264.63
Calculate support reactions	0.60	265.23

Loadcase # 2 Title: Deadweight Load

Read load data	5.16	270.39
Back-substitution	40.21	310.60
Calculate displacements	2.91	313.51
Calculate stresses	27.74	341.25
Calculate support reactions	0.60	341.85

Number of elements	78
Number of nodes	44
Number of degrees of freedom . .	264
Number of restraints	24
Number of equations	240
Number of stiffness terms . . .	8292
Number of stiffness vector blocks	2

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	67382	5-27-88	6:00p
REDUCA	DMP	67382	5-27-88	6:01p
DIAGAD	DMP	496	5-27-88	6:00p
PIVOTT	DMP	1958	5-27-88	6:02p

Blocked out-of-core solver WITH bandwidth reduction

Model: TMS1.STR Title: Roof Structure for TMS Inc.

	Time	Cum. Time
Read nodal data	3.90	3.90
Read element data	12.58	16.48
Read section and material data	0.99	17.47
Read restraint data	0.71	18.18
Reorder nodes to reduce bandwidth	0.99	19.17
Calculate stiffness vector blocks	81.18	100.35
Reduce stiffness vector blocks	98.81	199.16

Loadcase # 1 Title: Downward Pressure on Roof Elements 1-70=45 lb/SqF

Read load data	8.63	207.79
Back-substitution	43.61	251.40
Calculate displacements	2.91	254.31
Calculate stresses	27.57	281.88
Calculate support reactions	0.71	282.59

Loadcase # 2 Title: Deadweight Load

Read load data	5.17	287.76
Back-substitution	43.11	330.87
Calculate displacements	2.97	333.84
Calculate stresses	27.52	361.36
Calculate support reactions	0.60	361.96

Number of elements	78
Number of nodes	44
Number of degrees of freedom	264
Number of restraints	24
Number of equations	240
Number of stiffness terms	8832
Number of stiffness vector blocks	2

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	71768	5-27-88	6:06p
REDUCA	DMP	71768	5-27-88	6:07p
DIAGAD	DMP	496	5-27-88	6:06p
PIVOTT	DMP	1958	5-27-88	6:08p

Frontal solver WITHOUT frontwidth reduction

Model: TMS1.STR Title: Roof Structure for TMS Inc.

	Time	Cum. Time
Read nodal data	3.90	3.90
Read element data	12.52	16.42
Read section and material data	0.94	17.36
Read restraint data	1.15	18.51
Determine element order	0.82	19.33
Do pre-front housekeeping	52.24	71.57

Loadcase # 1 Title: Downward Pressure on Roof Elements 1-70=45 lb/SqF

Read load data	9.45	81.02
Begin wavefront solver	0.22	81.24
Initialize variables	3.68	84.92
Wavefront reduction		

Number of elements	78
Number of nodes	44
Number of degrees of freedom . .	264
Number of restraints	24

Frontwidth too large, analysis stopped

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	2	5-27-88	11:47p
SCRA2	DMP	2	5-27-88	11:47p

Frontal solver WITH frontwidth reduction

Model: TMS1.STR Title: Roof Structure for TMS Inc.

	Time	Cum. Time
Read nodal data	3.73	3.73
Read element data	12.53	16.26
Read section and material data	0.98	17.24
Read restraint data	1.21	18.45
Determine element order	2.25	20.70
Do pre-front housekeeping	52.24	72.94

Loadcase # 1 Title: Downward Pressure on Roof Elements 1-70=45 lb/SqF

Read load data	9.77	82.71
Begin wavefront solver	0.17	82.88
Initialize variables	2.80	85.68
Wavefront reduction	186.75	272.43
Back substitution	33.23	305.66
Calculate displacements	2.91	308.57
Calculate stresses	27.13	335.70
Calculate support reactions	0.61	336.31

Loadcase # 2 Title: Deadweight Load

Read load data	5.60	341.91
Begin wavefront solver	0.16	342.07
Initialize variables	0.28	342.35
Wavefront reduction	29.71	372.06
Back substitution	35.59	407.65
Calculate displacements	2.92	410.57
Calculate stresses	27.07	437.64
Calculate support reactions	0.61	438.25

Number of elements	78
Number of nodes	44
Number of degrees of freedom . .	264
Number of restraints	24
Original bandwidth	40
Reduced bandwidth	8
Largest frontwidth	54

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	94670	5-27-88	11:53p
SCRA2	DMP	2642	5-27-88	11:55p

Original solver WITHOUT bandwidth reduction

Model: BRIDGE1.STR Title: Truss Bridge

	Time	Cum. Time
Read nodal data	3.24	3.24
Read element data	11.48	14.72
Read restraint data	1.04	15.76
Calculate stiffness matrix	28.62	44.38
Forward reduction of stiffness matrix	120.18	164.56

Loadcase # 1 Title: Gravity Load

Read load data	2.91	167.47
Back-substitution	8.13	175.60
Calculate displacements	2.19	177.79
Calculate stresses	18.46	196.25
Calculate support reactions	0.66	196.91

Loadcase # 2 Title: Vehicle Load

Read load data	1.81	198.72
Back-substitution	8.13	206.85
Calculate displacements	2.25	209.10
Calculate stresses	18.24	227.34
Calculate support reactions	0.60	227.94

Number of elements	69
Number of nodes	32
Number of degrees of freedom . .	192
Number of restraints	12
Number of equations	180
Length of element arrays	7419
Length of nodal arrays	2357
Length of stiffness vector . . .	9728

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: BRIDGE1.STR Title: Truss Bridge

	Time	Cum. Time
Read nodal data	3.13	3.13
Read element data	11.59	14.72
Read restraint data	1.09	15.81
Reorder nodes to reduce bandwidth	0.72	16.53
Calculate stiffness matrix	27.96	44.49
Forward reduction of stiffness matrix	39.71	84.20

Loadcase # 1 Title: Gravity Load

Read load data	3.13	87.33
Back-substitution	5.93	93.26
Calculate displacements	2.25	95.51
Calculate stresses	18.18	113.69
Calculate support reactions	0.66	114.35

Loadcase # 2 Title: Vehicle Load

Read load data	1.87	116.22
Back-substitution	5.93	122.15
Calculate displacements	2.20	124.35
Calculate stresses	18.23	142.58
Calculate support reactions	0.61	143.19

Number of elements	69
Number of nodes	32
Number of degrees of freedom	192
Number of restraints	12
Number of equations	180
Length of element arrays	7419
Length of nodal arrays	2389
Length of stiffness vector	5572
Original Connectivity Range (Max)	17
Reduced Connectivity Range (Max)	7

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: BRIDGE1.STR Title: Truss Bridge

	Time	Cum. Time
Read nodal data	3.24	3.24
Read element data	11.59	14.83
Read section and material data	0.66	15.49
Read restraint data	0.94	16.43
Calculate stiffness vector blocks	44.49	60.92
Reduce stiffness vector blocks	142.58	203.50

Loadcase # 1 Title: Gravity Load

Read load data	3.63	207.13
Back-substitution	48.22	255.35
Calculate displacements	2.31	257.66
Calculate stresses	18.62	276.28
Calculate support reactions	0.55	276.83

Loadcase # 2 Title: Vehicle Load

Read load data	2.25	279.08
Back-substitution	47.67	326.75
Calculate displacements	2.26	329.01
Calculate stresses	18.62	347.63
Calculate support reactions	0.60	348.23

Number of elements	69
Number of nodes	32
Number of degrees of freedom	192
Number of restraints	12
Number of equations	180
Number of stiffness terms	9728
Number of stiffness vector blocks	2

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	79048	5-27-88	6:12p
REDUCA	DMP	79048	5-27-88	6:14p
DIAGAD	DMP	374	5-27-88	6:12p
PIVOTT	DMP	1470	5-27-88	6:14p

Blocked out-of-core solver WITH bandwidth reduction

Model: BRIDGE1.STR Title: Truss Bridge

	Time	Cum. Time
Read nodal data	3.46	3.46
Read element data	11.59	15.05
Read section and material data	0.77	15.82
Read restraint data	0.55	16.37
Reorder nodes to reduce bandwidth	0.71	17.08
Calculate stiffness vector blocks	36.91	53.99
Reduce stiffness vector blocks	50.04	104.03

Loadcase # 1 Title: Gravity Load

Read load data	3.57	107.60
Back-substitution	27.74	135.34
Calculate displacements	2.25	137.59
Calculate stresses	18.40	155.99
Calculate support reactions	0.66	156.65

Loadcase # 2 Title: Vehicle Load

Read load data	2.09	158.74
Back-substitution	26.25	184.99
Calculate displacements	2.20	187.19
Calculate stresses	18.56	205.75
Calculate support reactions	0.61	206.36

Number of elements	69
Number of nodes	32
Number of degrees of freedom . .	192
Number of restraints	12
Number of equations	180
Number of stiffness terms . . .	5572
Number of stiffness vector blocks	1

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	45278	5-27-88	6:17p
REDUCA	DMP	45278	5-27-88	6:18p
DIAGAD	DMP	370	5-27-88	6:17p
PIVOTT	DMP	1468	5-27-88	6:18p

Frontal solver WITHOUT frontwidth reduction

Model: BRIDGE1.STR Title: Truss Bridge

	Time	Cum. Time
Read nodal data	3.40	3.40
Read element data	11.76	15.16
Read section and material data	0.71	15.87
Read restraint data	0.93	16.80
Determine element order	0.72	17.52
Do pre-front housekeeping	33.78	51.30

Loadcase # 1 Title: Gravity Load

Read load data	4.23	55.53
Begin wavefront solver	0.22	55.75
Initialize variables	3.62	59.37
Wavefront reduction	160.38	219.75
Back substitution	44.38	264.13
Calculate displacements	2.31	266.44
Calculate stresses	18.24	284.68
Calculate support reactions	0.60	285.28

Loadcase # 2 Title: Vehicle Load

Read load data	2.53	287.81
Begin wavefront solver	0.22	288.03
Initialize variables	0.27	288.30
Wavefront reduction	37.63	325.93
Back substitution	46.68	372.61
Calculate displacements	2.20	374.81
Calculate stresses	18.45	393.26
Calculate support reactions	0.72	393.98

Number of elements	69
Number of nodes	32
Number of degrees of freedom . .	192
Number of restraints	12
Largest frontwidth	102

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	125582	5-28-88	12:00a
SCRA2	DMP	1922	5-28-88	12:02a

Frontal solver WITH frontwidth reduction

Model: BRIDGE1.STR Title: Truss Bridge

	Time	Cum. Time
Read nodal data	3.46	3.46
Read element data	11.48	14.94
Read section and material data	0.71	15.65
Read restraint data	0.93	16.58
Determine element order	1.49	18.07
Do pre-front housekeeping	33.72	51.79

Loadcase # 1 Title: Gravity Load

Read load data	5.05	56.84
Begin wavefront solver	0.17	57.01
Initialize variables	2.80	59.81
Wavefront reduction	105.90	165.71
Back substitution	22.35	188.06
Calculate displacements	2.20	190.26
Calculate stresses	18.23	208.49
Calculate support reactions	0.61	209.10

Loadcase # 2 Title: Vehicle Load

Read load data	2.69	211.79
Begin wavefront solver	0.16	211.95
Initialize variables	0.28	212.23
Wavefront reduction	19.55	231.78
Back substitution	23.95	255.73
Calculate displacements	2.14	257.87
Calculate stresses	18.24	276.11
Calculate support reactions	0.55	276.66

Number of elements	69
Number of nodes	32
Number of degrees of freedom . .	192
Number of restraints	12
Original bandwidth	17
Reduced bandwidth	7
Largest frontwidth	48

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	63446	5-28-88	12:06a
SCRA2	DMP	1922	5-28-88	12:07a

Original solver WITHOUT bandwidth reduction

Model: TESTRUN.STR Title: Gail Deck Tiedown

	Time	Cum. Time
Read nodal data	5.93	5.93
Read element data	21.64	27.57
Read restraint data	2.64	30.21
Calculate stiffness matrix	38.72	68.93
Forward reduction of stiffness matrix	212.34	281.27

Loadcase # 1 Title: Gail Deck Tiedown Port Side Combined Max. Loads

Read load data	1.71	282.98
Back-substitution	35.92	318.90
Calculate displacements	5.55	324.45
Calculate stresses	32.68	357.13
Calculate support reactions	1.31	358.44

Loadcase # 2 Title: Gail Deck Tiedown port side Fx= 100 kips

Read load data	1.71	360.15
Back-substitution	35.92	396.07
Calculate displacements	5.66	401.73
Calculate stresses	32.62	434.35
Calculate support reactions	1.27	435.62

Number of elements	141
Number of nodes	89
Number of degrees of freedom	534
Number of restraints	0
Number of equations	534
Length of element arrays	9819
Length of nodal arrays	6769
Length of stiffness vector	22317

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: TESTRUN.STR Title: Gail Deck Tiedown

	Time	Cum. Time
Read nodal data	5.93	5.93
Read element data	21.69	27.62
Read restraint data	2.75	30.37
Reorder nodes to reduce bandwidth	2.36	32.73
Calculate stiffness matrix	39.05	71.78
Forward reduction of stiffness matrix	238.60	310.38

Loadcase # 1 Title: Gail Deck Tiedown Port Side Combined Max. Loads

Read load data	1.76	312.14
Back-substitution	36.85	348.99
Calculate displacements	5.55	354.54
Calculate stresses	32.63	387.17
Calculate support reactions	1.26	388.43

Loadcase # 2 Title: Gail Deck Tiedown port side Fx= 100 kips

Read load data	1.76	390.19
Back-substitution	36.85	427.04
Calculate displacements	5.49	432.53
Calculate stresses	32.63	465.16
Calculate support reactions	1.32	466.48

Number of elements	141
Number of nodes	89
Number of degrees of freedom . . .	534
Number of restraints	0
Number of equations	534
Length of element arrays	9819
Length of nodal arrays	6858
Length of stiffness vector . . .	24045
Original Connectivity Range (Max)	43
Reduced Connectivity Range (Max)	10

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: TESTRUN.STR Title: Gail Deck Tiedown

	Time	Cum. Time
Read nodal data	6.15	6.15
Read element data	21.64	27.79
Read section and material data	1.15	28.94
Read restraint data	2.48	31.42
Calculate stiffness vector blocks	76.95	108.37
Reduce stiffness vector blocks	263.47	371.84

Loadcase # 1 Title: Gail Deck Tiedown Port Side Combined Max. Loads

Read load data	2.53	374.37
Back-substitution	110.62	484.99
Calculate displacements	5.60	490.59
Calculate stresses	33.51	524.10
Calculate support reactions	1.37	525.47

Loadcase # 2 Title: Gail Deck Tiedown port side Fx= 100 kips

Read load data	1.98	527.45
Back-substitution	98.86	626.31
Calculate displacements	5.83	632.14
Calculate stresses	33.45	665.59
Calculate support reactions	1.31	666.90

Number of elements	141
Number of nodes	89
Number of degrees of freedom . . .	534
Number of restraints	0
Number of equations	534
Number of stiffness terms	22325
Number of stiffness vector blocks	3

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	181404	5-27-88	6:23p
REDUCA	DMP	181404	5-27-88	6:27p
DIAGAD	DMP	1096	5-27-88	6:23p
PIVOTT	DMP	4350	5-27-88	6:28p

Blocked out-of-core solver WITH bandwidth reduction

Model: TESTRUN.STR Title: Gail Deck Tiedown

	Time	Cum. Time
Read nodal data	6.37	6.37
Read element data	21.75	28.12
Read section and material data	1.21	29.33
Read restraint data	1.87	31.20
Reorder nodes to reduce bandwidth	2.42	33.62
Calculate stiffness vector blocks	80.41	114.03
Reduce stiffness vector blocks	316.20	430.23

Loadcase # 1 Title: Gail Deck Tiedown Port Side Combined Max. Loads

Read load data	2.58	432.81
Back-substitution	107.93	540.74
Calculate displacements	5.60	546.34
Calculate stresses	33.29	579.63
Calculate support reactions	1.43	581.06

Loadcase # 2 Title: Gail Deck Tiedown port side Fx= 100 kips

Read load data	1.92	582.98
Back-substitution	107.43	690.41
Calculate displacements	5.83	696.24
Calculate stresses	33.45	729.69
Calculate support reactions	1.31	731.00

Number of elements	141
Number of nodes	89
Number of degrees of freedom . .	534
Number of restraints	0
Number of equations	534
Number of stiffness terms . . .	24045
Number of stiffness vector blocks	4

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	195380	5-27-88	6:34p
REDUCA	DMP	195380	5-27-88	6:40p
DIAGAD	DMP	1098	5-27-88	6:34p
PIVOTT	DMP	4352	5-27-88	6:40p

Frontal solver WITHOUT frontwidth reduction

Model: TESTRUN.STR Title: Gail Deck Tiedown

	Time	Cum. Time
Read nodal data	6.26	6.26
Read element data	21.75	28.01
Read section and material data	1.10	29.11
Read restraint data	2.53	31.64
Determine element order	2.14	33.78
Do pre-front housekeeping	102.49	136.27

Loadcase # 1 Title: Gail Deck Tiedown Port Side Combined Max. Loads

Read load data	3.41	139.68
Begin wavefront solver	0.16	139.84
Initialize variables	3.68	143.52
Wavefront reduction		

Number of elements	141
Number of nodes	89
Number of degrees of freedom . .	534
Number of restraints	0

Frontwidth too large, analysis stopped

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	191701	5-28-88	12:12a
SCRA2	DMP	2	5-28-88	12:10a

Frontal solver WITH frontwidth reduction

Model: TESTRUN.STR Title: Gail Deck Tiedown

	Time	Cum. Time
Read nodal data	6.70	6.70
Read element data	21.75	28.45
Read section and material data	1.21	29.66
Read restraint data	2.58	32.24
Determine element order	5.99	38.23
Do pre-front housekeeping	102.43	140.66

Loadcase # 1 Title: Gail Deck Tiedown Port Side Combined Max. Loads

Read load data	3.63	144.29
Begin wavefront solver	0.16	144.45
Initialize variables	2.80	147.25
Wavefront reduction	300.28	447.53
Back substitution	75.41	522.94
Calculate displacements	5.61	528.55
Calculate stresses	32.51	561.06
Calculate support reactions	1.38	562.44

Loadcase # 2 Title: Gail Deck Tiedown port side Fx= 100 kips

Read load data	2.80	565.24
Begin wavefront solver	0.22	565.46
Initialize variables	0.27	565.73
Wavefront reduction	60.58	626.31
Back substitution	80.14	706.45
Calculate displacements	5.60	712.05
Calculate stresses	32.52	744.57
Calculate support reactions	1.32	745.89

Number of elements	141
Number of nodes	89
Number of degrees of freedom . . .	534
Number of restraints	0
Original bandwidth	43
Reduced bandwidth	10
Largest frontwidth	60

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	206366	5-28-88	12:20a
SCRA2	DMP	5342	5-28-88	12:23a

Original solver WITHOUT bandwidth reduction

Model: TANK.STR Title: STEEL TANK ON PILES

	Time	Cum. Time
Read nodal data	6.26	6.26
Read element data	13.34	19.60
Read restraint data	2.31	21.91

Number of elements	84
Number of nodes	97
Number of degrees of freedom . . .	582
Number of restraints	144
Number of equations	438
Length of element arrays	9000
Length of nodal arrays	7035
Length of stiffness vector . . .	57045

Problem size too large. Max. Stiffness Vector Length = 30000

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: TANK.STR Title: STEEL TANK ON PILES

	Time	Cum. Time
Read nodal data	6.21	6.21
Read element data	13.40	19.61
Read restraint data	2.41	22.02
Reorder nodes to reduce bandwidth	4.51	26.53

Number of elements	84
Number of nodes	97
Number of degrees of freedom . . .	582
Number of restraints	144
Number of equations	438
Length of element arrays	9000
Length of nodal arrays	7132
Length of stiffness vector . . .	35841
Original Connectivity Range (Max)	47
Reduced Connectivity Range (Max)	23

Problem size too large. Max. Stiffness Vector Length = 30000

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: TANK.STR Title: STEEL TANK ON PILES

	Time	Cum. Time
Read nodal data	6.64	6.64
Read element data	13.35	19.99
Read section and material data	0.49	20.48
Read restraint data	2.86	23.34
Calculate stiffness vector blocks	352.68	376.02
Reduce stiffness vector blocks	1675.11	2051.13

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	22.36	2073.49
Back-substitution	288.41	2361.90
Calculate displacements	5.77	2367.67
Calculate stresses	46.03	2413.70
Calculate support reactions	1.81	2415.51

Loadcase # 2 Title: 10 Lb. load in Y direction for nodes 26-49

Read load data	4.01	2419.52
Back-substitution	273.20	2692.72
Calculate displacements	5.98	2698.70
Calculate stresses	45.92	2744.62
Calculate support reactions	1.87	2746.49

Number of elements	84
Number of nodes	97
Number of degrees of freedom . .	582
Number of restraints	144
Number of equations	438
Number of stiffness terms . . .	57045
Number of stiffness vector blocks	8

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	463518	5-27-88	7:02p
REDUCA	DMP	463518	5-27-88	7:30p
DIAGAD	DMP	912	5-27-88	7:02p
PIVOTT	DMP	3586	5-27-88	7:32p

Blocked out-of-core solver WITH bandwidth reduction

Model: TANK.STR Title: STEEL TANK ON PILES

	Time	Cum. Time
Read nodal data	6.70	6.70
Read element data	13.40	20.10
Read section and material data	0.55	20.65
Read restraint data	2.14	22.79
Reorder nodes to reduce bandwidth	4.50	27.29
Calculate stiffness vector blocks	296.33	323.62
Reduce stiffness vector blocks	709.42	1033.04

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	22.13	1055.17
Back-substitution	178.56	1233.73
Calculate displacements	5.72	1239.45
Calculate stresses	45.64	1285.09
Calculate support reactions	1.92	1287.01

Loadcase # 2 Title: 10 Lb. load in Y direction for nodes 26-49

Read load data	4.07	1291.08
Back-substitution	177.40	1468.48
Calculate displacements	5.94	1474.42
Calculate stresses	45.86	1520.28
Calculate support reactions	1.87	1522.15

Number of elements	84
Number of nodes	97
Number of degrees of freedom . .	582
Number of restraints	144
Number of equations	438
Number of stiffness terms . . .	35841
Number of stiffness vector blocks	5

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	291226	5-27-88	7:47p
REDUCA	DMP	291226	5-27-88	7:59p
DIAGAD	DMP	908	5-27-88	7:47p
PIVOTT	DMP	3576	5-27-88	8:00p

Frontal solver WITHOUT frontwidth reduction

Model: TANK.STR

Title: STEEL TANK ON PILES

	Time	Cum. Time
Read nodal data	6.97	6.97
Read element data	13.40	20.37
Read section and material data	0.55	20.92
Read restraint data	2.86	23.78
Determine element order	0.93	24.71
Do pre-front housekeeping	123.37	148.08

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	22.90	170.98
Begin wavefront solver	0.16	171.14
Initialize variables	3.68	174.82
Wavefront reduction	1156.35	1331.17
Back substitution	260.29	1591.46
Calculate displacements	5.66	1597.12
Calculate stresses	45.26	1642.38
Calculate support reactions	1.70	1644.08

Loadcase # 2 Title: 10 Lb. load in Y direction for nodes 26-49

Read load data	4.78	1648.86
Begin wavefront solver	0.22	1649.08
Initialize variables	0.33	1649.41
Wavefront reduction	182.08	1831.49
Back substitution	267.26	2098.75
Calculate displacements	5.61	2104.36
Calculate stresses	44.82	2149.18
Calculate support reactions	1.75	2150.93

Number of elements	84
Number of nodes	97
Number of degrees of freedom . .	582
Number of restraints	144
Largest frontwidth	162

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	688886	5-28-88	12:47a
SCRA2	DMP	5822	5-28-88	12:56a

Frontal solver WITH frontwidth reduction

Model: TANK.STR Title: STEEL TANK ON PILES

	Time	Cum. Time
Read nodal data	7.09	7.09
Read element data	13.46	20.55
Read section and material data	0.55	21.10
Read restraint data	2.96	24.06
Determine element order	7.86	31.92
Do pre-front housekeeping	123.36	155.28

Loadcase # 1 Title: Uniform Transverse Pressure - P = 10psi

Read load data	24.28	179.56
Begin wavefront solver	0.21	179.77
Initialize variables	2.81	182.58
Wavefront reduction	734.35	916.93
Back substitution	187.41	1104.34
Calculate displacements	5.60	1109.94
Calculate stresses	44.93	1154.87
Calculate support reactions	1.70	1156.57

Loadcase # 2 Title: 10 Lb. load in Y direction for nodes 26-49

Read load data	4.72	1161.29
Begin wavefront solver	0.22	1161.51
Initialize variables	0.33	1161.84
Wavefront reduction	135.89	1297.73
Back substitution	193.23	1490.96
Calculate displacements	5.60	1496.56
Calculate stresses	44.87	1541.43
Calculate support reactions	1.76	1543.19

Number of elements	84
Number of nodes	97
Number of degrees of freedom . .	582
Number of restraints	144
Original bandwidth	47
Reduced bandwidth	23
Largest frontwidth	144

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	509606	5-28-88	1:16a
SCRA2	DMP	5822	5-28-88	1:23a

Original solver WITHOUT bandwidth reduction

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

	Time	Cum. Time
Read nodal data	5.00	5.00
Read element data	28.89	33.89
Read restraint data	3.57	37.46

Number of elements	120
Number of nodes	72
Number of degrees of freedom . .	552
Number of restraints	120
Number of equations	432
Length of element arrays . . .	8379
Length of nodal arrays	6049
Length of stiffness vector . .	55464

Problem size too large. Max. Stiffness Vector Length = 30000

Stop - Program terminated.

Original solver WITH bandwidth reduction

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

	Time	Cum. Time
Read nodal data	5.11	5.11
Read element data	28.94	34.05
Read restraint data	3.52	37.57
Reorder nodes to reduce bandwidth	1.97	39.54
Calculate stiffness matrix	32.63	72.17
Forward reduction of stiffness matrix	229.75	301.92

Loadcase # 1 Title: DEAD LOAD

Read load data	2.42	304.34
Back-substitution	27.46	331.80
Calculate displacements	13.68	345.48
Calculate stresses	27.57	373.05
Calculate support reactions	2.53	375.58

Loadcase # 2 Title: LATERAL LOADS

Read load data	2.42	378.00
Back-substitution	27.46	405.46
Calculate displacements	13.62	419.08
Calculate stresses	27.63	446.71
Calculate support reactions	2.41	449.12

Number of elements	120
Number of nodes	72
Number of degrees of freedom . . .	552
Number of restraints	120
Number of equations	432
Length of element arrays	8379
Length of nodal arrays	6121
Length of stiffness vector	21760
Original Connectivity Range (Max)	60
Reduced Connectivity Range (Max)	14

Stop - Program terminated.

Blocked out-of-core solver WITHOUT bandwidth reduction

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

	Time	Cum. Time
Read nodal data	5.16	5.16
Read element data	28.95	34.11
Read section and material data	1.04	35.15
Read restraint data	3.46	38.61
Calculate stiffness vector blocks	135.23	173.84
Reduce stiffness vector blocks	1727.46	1901.30

Loadcase # 1 Title: DEAD LOAD

Read load data	3.02	1904.32
Back-substitution	243.81	2148.13
Calculate displacements	14.06	2162.19
Calculate stresses	28.13	2190.32
Calculate support reactions	2.52	2192.84

Loadcase # 2 Title: LATERAL LOADS

Read load data	2.80	2195.64
Back-substitution	243.10	2438.74
Calculate displacements	14.39	2453.13
Calculate stresses	28.40	2481.53
Calculate support reactions	2.53	2484.06

Number of elements	120
Number of nodes	72
Number of degrees of freedom	552
Number of restraints	120
Number of equations	432
Number of stiffness terms	55464
Number of stiffness vector blocks	8

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	450672	5-28-88	2:14a
REDUCA	DMP	450672	5-28-88	2:42a
DIAGAD	DMP	904	5-28-88	2:14a
PIVOTT	DMP	3536	5-28-88	2:44a

Blocked out-of-core solver WITH bandwidth reduction

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

	Time	Cum. Time
Read nodal data	5.22	5.22
Read element data	28.94	34.16
Read section and material data	1.05	35.21
Read restraint data	2.74	37.95
Reorder nodes to reduce bandwidth	1.93	39.88
Calculate stiffness vector blocks	72.55	112.43
Reduce stiffness vector blocks	280.56	392.99

Loadcase # 1 Title: DEAD LOAD

Read load data	3.30	396.29
Back-substitution	107.10	503.39
Calculate displacements	14.06	517.45
Calculate stresses	28.18	545.63
Calculate support reactions	2.58	548.21

Loadcase # 2 Title: LATERAL LOADS

Read load data	2.69	550.90
Back-substitution	106.45	657.35
Calculate displacements	14.39	671.74
Calculate stresses	28.72	700.46
Calculate support reactions	2.53	702.99

Number of elements	120
Number of nodes	72
Number of degrees of freedom . .	552
Number of restraints	120
Number of equations	432
Number of stiffness terms . . .	21760
Number of stiffness vector blocks	3

Stop - Program terminated.

Directory of C:\OUT

UNREDA	DMP	176810	5-28-88	2:54a
REDUCA	DMP	176810	5-28-88	2:59a
DIAGAD	DMP	890	5-28-88	2:54a
PIVOTT	DMP	3522	5-28-88	3:00a

Frontal solver WITHOUT frontwidth reduction

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

	Time	Cum. Time
Read nodal data	5.28	5.28
Read element data	28.78	34.06
Read section and material data	1.10	35.16
Read restraint data	3.46	38.62
Determine element order	1.59	40.21
Do pre-front housekeeping	90.13	130.34

Loadcase # 1 Title: DEAD LOAD

Read load data	4.40	134.74
Begin wavefront solver	0.22	134.96
Initialize variables	3.68	138.64
Wavefront reduction	947.63	1086.27
Back substitution	261.28	1347.55
Calculate displacements	13.84	1361.39
Calculate stresses	27.41	1388.80
Calculate support reactions	2.41	1391.21

Loadcase # 2 Title: LATERAL LOADS

Read load data	3.79	1395.00
Begin wavefront solver	0.22	1395.22
Initialize variables	0.33	1395.55
Wavefront reduction	175.21	1570.76
Back substitution	265.24	1836.00
Calculate displacements	13.84	1849.84
Calculate stresses	27.79	1877.63
Calculate support reactions	2.42	1880.05

Number of elements	120
Number of nodes	72
Number of degrees of freedom	552
Number of restraints	120
Largest frontwidth	190

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	676426	5-28-88	1:45a
SCRA2	DMP	5522	5-28-88	1:53a

Frontal solver WITH frontwidth reduction

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

	Time	Cum. Time
Read nodal data	5.49	5.49
Read element data	29.11	34.60
Read section and material data	1.27	35.87
Read restraint data	3.57	39.44
Determine element order	4.34	43.78
Do pre-front housekeeping	90.13	133.91

Loadcase # 1 Title: DEAD LOAD

Read load data	5.05	138.96
Begin wavefront solver	0.22	139.18
Initialize variables	2.97	142.15
Wavefront reduction	226.95	369.10
Back substitution	84.15	453.25
Calculate displacements	13.73	466.98
Calculate stresses	27.35	494.33
Calculate support reactions	2.42	496.75

Loadcase # 2 Title: LATERAL LOADS

Read load data	3.68	500.43
Begin wavefront solver	0.22	500.65
Initialize variables	0.27	500.92
Wavefront reduction	64.59	565.51
Back substitution	88.54	654.05
Calculate displacements	13.79	667.84
Calculate stresses	27.35	695.19
Calculate support reactions	2.42	697.61

Number of elements	120
Number of nodes	72
Number of degrees of freedom	552
Number of restraints	120
Original bandwidth	60
Reduced bandwidth	14
Largest frontwidth	60

Stop - Program terminated.

Directory of C:\FRONT

SCRA1	DMP	234646	5-28-88	2:05a
SCRA2	DMP	5522	5-28-88	2:08a

APPENDIX B

SAMPLE PROBLEM DATA

2 x 10 Mesh Plates - Optimal Ordering

PLT2X10

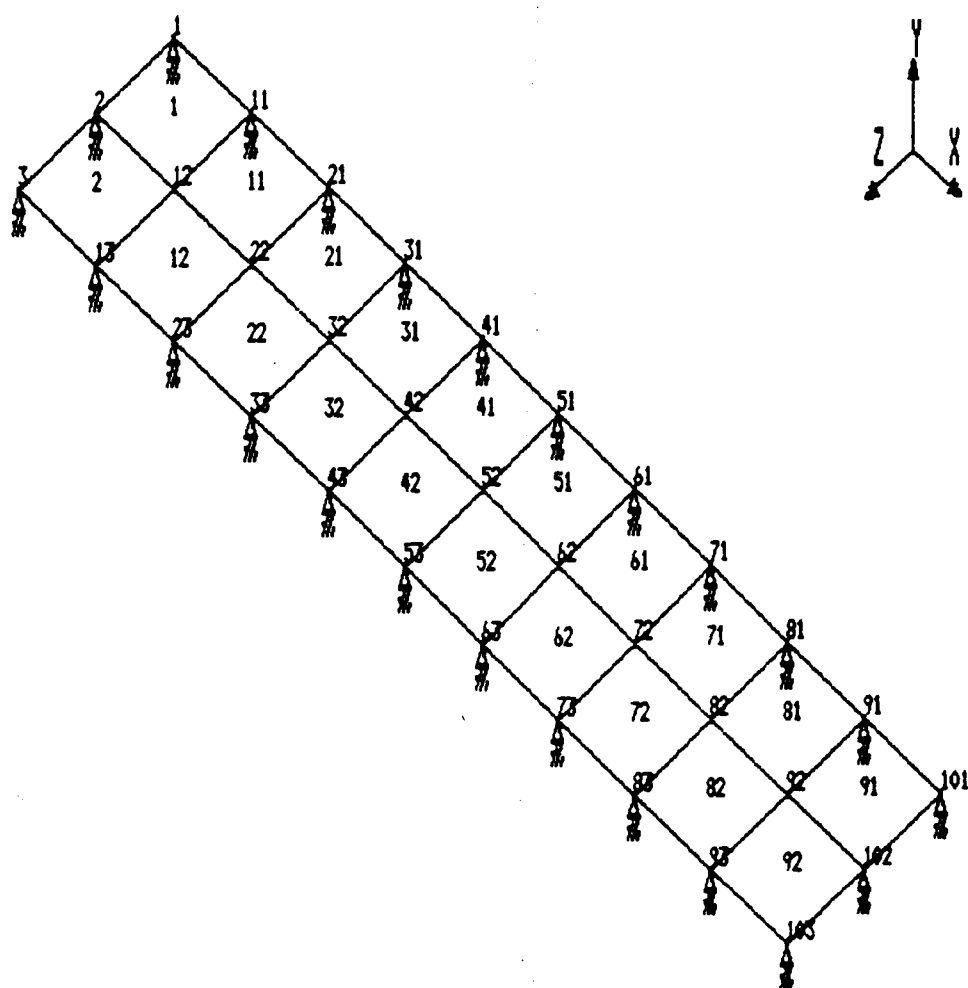


Figure B.1. Model PLT2X10.STR.

Model: PLT2X10.STR Title: 2 x 10 Mesh Plates - Optimal Ordering

MODEL PARAMETERS

Number of nodes 33
 Number of restrained nodes . . 24
 Number of elements 20
 Trusses 0
 Beams 0
 Plates 20
 Number of sections 0
 Number of materials 1

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
2	.0000E+00	.0000E+00	5.0000E+01
3	.0000E+00	.0000E+00	1.0000E+02
11	5.0000E+01	.0000E+00	.0000E+00
12	5.0000E+01	.0000E+00	5.0000E+01
13	5.0000E+01	.0000E+00	1.0000E+02
21	1.0000E+02	.0000E+00	.0000E+00
22	1.0000E+02	.0000E+00	5.0000E+01
23	1.0000E+02	.0000E+00	1.0000E+02
31	1.5000E+02	.0000E+00	.0000E+00
32	1.5000E+02	.0000E+00	5.0000E+01
33	1.5000E+02	.0000E+00	1.0000E+02
41	2.0000E+02	.0000E+00	.0000E+00
42	2.0000E+02	.0000E+00	5.0000E+01
43	2.0000E+02	.0000E+00	1.0000E+02
51	2.5000E+02	.0000E+00	.0000E+00
52	2.5000E+02	.0000E+00	5.0000E+01
53	2.5000E+02	.0000E+00	1.0000E+02
61	3.0000E+02	.0000E+00	.0000E+00
62	3.0000E+02	.0000E+00	5.0000E+01
63	3.0000E+02	.0000E+00	1.0000E+02
71	3.5000E+02	.0000E+00	.0000E+00
72	3.5000E+02	.0000E+00	5.0000E+01
73	3.5000E+02	.0000E+00	1.0000E+02
81	4.0000E+02	.0000E+00	.0000E+00
82	4.0000E+02	.0000E+00	5.0000E+01
83	4.0000E+02	.0000E+00	1.0000E+02
91	4.5000E+02	.0000E+00	.0000E+00
92	4.5000E+02	.0000E+00	5.0000E+01
93	4.5000E+02	.0000E+00	1.0000E+02
101	5.0000E+02	.0000E+00	.0000E+00
102	5.0000E+02	.0000E+00	5.0000E+01
103	5.0000E+02	.0000E+00	1.0000E+02

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	11	12	2	1.0000E+00	1	0	0
2	2	12	13	3	1.0000E+00	1	0	0
11	11	21	22	12	1.0000E+00	1	0	0
12	12	22	23	13	1.0000E+00	1	0	0
21	21	31	32	22	1.0000E+00	1	0	0
22	22	32	33	23	1.0000E+00	1	0	0
31	31	41	42	32	1.0000E+00	1	0	0
32	32	42	43	33	1.0000E+00	1	0	0
41	41	51	52	42	1.0000E+00	1	0	0
42	42	52	53	43	1.0000E+00	1	0	0
51	51	61	62	52	1.0000E+00	1	0	0
52	52	62	63	53	1.0000E+00	1	0	0
61	61	71	72	62	1.0000E+00	1	0	0
62	62	72	73	63	1.0000E+00	1	0	0
71	71	81	82	72	1.0000E+00	1	0	0
72	72	82	83	73	1.0000E+00	1	0	0
81	81	91	92	82	1.0000E+00	1	0	0
82	82	92	93	83	1.0000E+00	1	0	0
91	91	101	102	92	1.0000E+00	1	0	0
92	92	102	103	93	1.0000E+00	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	3.0000E+07	3.0000E-01	3.6000E+04	2.8400E-01	6.5000E-06

N O D A L R E S T R A I N T S

Node Number	---Translations---			---Rotations---		
	X	Y	Z	X	Y	Z
1	1	1	0	0	0	0
2	1	1	1	0	1	0
3	1	1	0	0	0	0
11	0	1	0	0	0	0
13	0	1	0	0	0	0
21	0	1	0	0	0	0
23	0	1	0	0	0	0
31	0	1	0	0	0	0
33	0	1	0	0	0	0
41	0	1	0	0	0	0
43	0	1	0	0	0	0
51	0	1	0	0	0	0

53	0	1	0	0	0	0
61	0	1	0	0	0	0
63	0	1	0	0	0	0
71	0	1	0	0	0	0
73	0	1	0	0	0	0
81	0	1	0	0	0	0
83	0	1	0	0	0	0
91	0	1	0	0	0	0
93	0	1	0	0	0	0
101	1	1	0	0	0	0
102	1	1	1	0	1	0
103	1	1	0	0	0	0

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom 198
 Number of restraints 34
 Number of equations 164

L O A D C A S E N U M B E R 1

Uniform Transverse Pressure - $P = 10\text{psi}$

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
 Number of nodes with applied displacements 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads 1
 Number of elements with concentrated loads 0
 Number of elements with thermal loads 0

L O A D C A S E N U M B E R 2

Thermal Load - $dT=100 \text{ deg.F.}$

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
 Number of nodes with applied displacements 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads 0
 Number of elements with concentrated loads 0
 Number of elements with thermal loads 1

10 x 2 Mesh Plates - Inefficient Ordering

PLT10X2

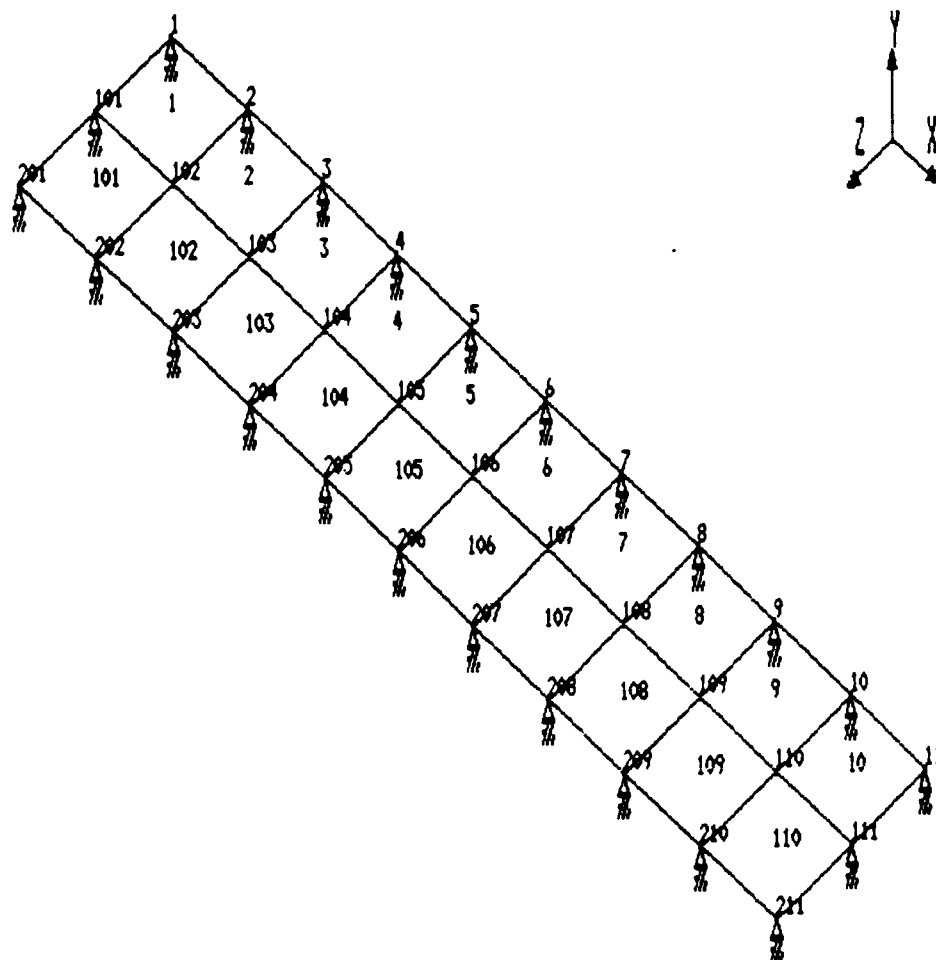


Figure B.2. Model PLT10X2.STR.

Model: PLT10X2.STR Title: 10 x 2 Mesh Plates - Inefficient Ordering

MODEL PARAMETERS

Number of nodes 33
 Number of restrained nodes . . 24
 Number of elements 20
 Trusses 0
 Beams 0
 Plates 20
 Number of sections 0
 Number of materials 1

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
2	5.0000E+01	.0000E+00	.0000E+00
3	1.0000E+02	.0000E+00	.0000E+00
4	1.5000E+02	.0000E+00	.0000E+00
5	2.0000E+02	.0000E+00	.0000E+00
6	2.5000E+02	.0000E+00	.0000E+00
7	3.0000E+02	.0000E+00	.0000E+00
8	3.5000E+02	.0000E+00	.0000E+00
9	4.0000E+02	.0000E+00	.0000E+00
10	4.5000E+02	.0000E+00	.0000E+00
11	5.0000E+02	.0000E+00	.0000E+00
101	.0000E+00	.0000E+00	5.0000E+01
102	5.0000E+01	.0000E+00	5.0000E+01
103	1.0000E+02	.0000E+00	5.0000E+01
104	1.5000E+02	.0000E+00	5.0000E+01
105	2.0000E+02	.0000E+00	5.0000E+01
106	2.5000E+02	.0000E+00	5.0000E+01
107	3.0000E+02	.0000E+00	5.0000E+01
108	3.5000E+02	.0000E+00	5.0000E+01
109	4.0000E+02	.0000E+00	5.0000E+01
110	4.5000E+02	.0000E+00	5.0000E+01
111	5.0000E+02	.0000E+00	5.0000E+01
201	.0000E+00	.0000E+00	1.0000E+02
202	5.0000E+01	.0000E+00	1.0000E+02
203	1.0000E+02	.0000E+00	1.0000E+02
204	1.5000E+02	.0000E+00	1.0000E+02
205	2.0000E+02	.0000E+00	1.0000E+02
206	2.5000E+02	.0000E+00	1.0000E+02
207	3.0000E+02	.0000E+00	1.0000E+02
208	3.5000E+02	.0000E+00	1.0000E+02
209	4.0000E+02	.0000E+00	1.0000E+02
210	4.5000E+02	.0000E+00	1.0000E+02
211	5.0000E+02	.0000E+00	1.0000E+02

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	2	102	101	1.0000E+00	1	0	0
2	2	3	103	102	1.0000E+00	1	0	0
3	3	4	104	103	1.0000E+00	1	0	0
4	4	5	105	104	1.0000E+00	1	0	0
5	5	6	106	105	1.0000E+00	1	0	0
6	6	7	107	106	1.0000E+00	1	0	0
7	7	8	108	107	1.0000E+00	1	0	0
8	8	9	109	108	1.0000E+00	1	0	0
9	9	10	110	109	1.0000E+00	1	0	0
10	10	11	111	110	1.0000E+00	1	0	0
101	101	102	202	201	1.0000E+00	1	0	0
102	102	103	203	202	1.0000E+00	1	0	0
103	103	104	204	203	1.0000E+00	1	0	0
104	104	105	205	204	1.0000E+00	1	0	0
105	105	106	206	205	1.0000E+00	1	0	0
106	106	107	207	206	1.0000E+00	1	0	0
107	107	108	208	207	1.0000E+00	1	0	0
108	108	109	209	208	1.0000E+00	1	0	0
109	109	110	210	209	1.0000E+00	1	0	0
110	110	111	211	210	1.0000E+00	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	3.0000E+07	3.0000E-01	3.6000E+04	2.8400E-01	6.5000E-06

N O D A L R E S T R A I N T S

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
1	1	1	0	0	0	0
2	0	1	0	0	0	0
3	0	1	0	0	0	0
4	0	1	0	0	0	0
5	0	1	0	0	0	0
6	0	1	0	0	0	0
7	0	1	0	0	0	0
8	0	1	0	0	0	0
9	0	1	0	0	0	0
10	0	1	0	0	0	0

11	1	1	0	0	0	0
101	1	1	1	0	1	0
111	1	1	1	0	1	0
201	1	1	0	0	0	0
202	0	1	0	0	0	0
203	0	1	0	0	0	0
204	0	1	0	0	0	0
205	0	1	0	0	0	0
206	0	1	0	0	0	0
207	0	1	0	0	0	0
208	0	1	0	0	0	0
209	0	1	0	0	0	0
210	0	1	0	0	0	0
211	1	1	0	0	0	0

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom . . . 198
Number of restraints 34
Number of equations 164

L O A D C A S E N U M B E R 1

Uniform Transverse Pressure - $P = 10\text{psi}$

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
Number of nodes with applied displacements . . . 0
Indicator for gravitational loads 0
Number of elements with distributed loads . . . 1
Number of elements with concentrated loads . . . 0
Number of elements with thermal loads 0

L O A D C A S E N U M B E R 2

Thermal Load - $dT=100 \text{ deg.F.}$

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
Number of nodes with applied displacements . . . 0
Indicator for gravitational loads 0
Number of elements with distributed loads . . . 0
Number of elements with concentrated loads . . . 0
Number of elements with thermal loads 1

4 x 10 Mesh Plates - Optimal Ordering

PLT4X10

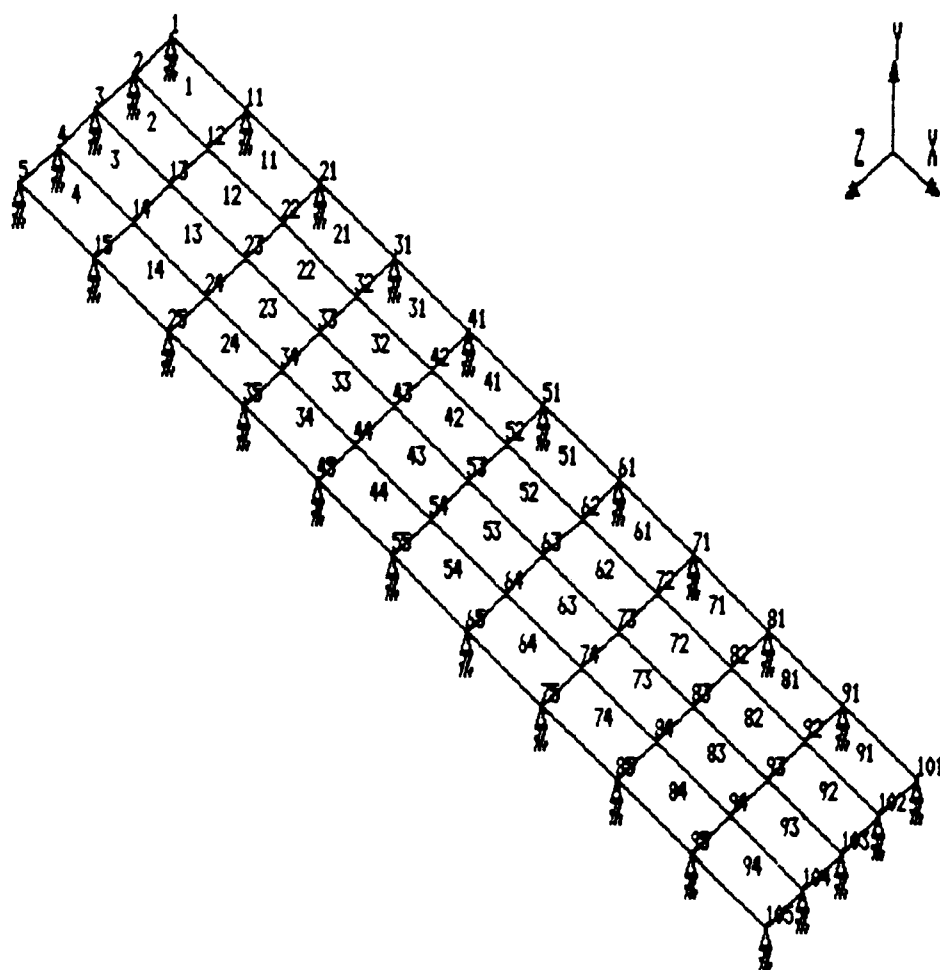


Figure B.3. Model PLT4X10.STR.

Model: PLT4X10.STR Title: 4 x 10 Mesh Plates - Optimal Ordering

M O D E L P A R A M E T E R S

Number of nodes 55
 Number of restrained nodes . . 28
 Number of elements 40
 Trusses 0
 Beams 0
 Plates 40
 Number of sections 0
 Number of materials 1

N O D A L C O O R D I N A T E S

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
2	.0000E+00	.0000E+00	2.5000E+01
3	.0000E+00	.0000E+00	5.0000E+01
4	.0000E+00	.0000E+00	7.5000E+01
5	.0000E+00	.0000E+00	1.0000E+02
11	5.0000E+01	.0000E+00	.0000E+00
12	5.0000E+01	.0000E+00	2.5000E+01
13	5.0000E+01	.0000E+00	5.0000E+01
14	5.0000E+01	.0000E+00	7.5000E+01
15	5.0000E+01	.0000E+00	1.0000E+02
21	1.0000E+02	.0000E+00	.0000E+00
22	1.0000E+02	.0000E+00	2.5000E+01
23	1.0000E+02	.0000E+00	5.0000E+01
24	1.0000E+02	.0000E+00	7.5000E+01
25	1.0000E+02	.0000E+00	1.0000E+02
31	1.5000E+02	.0000E+00	.0000E+00
32	1.5000E+02	.0000E+00	2.5000E+01
33	1.5000E+02	.0000E+00	5.0000E+01
34	1.5000E+02	.0000E+00	7.5000E+01
35	1.5000E+02	.0000E+00	1.0000E+02
41	2.0000E+02	.0000E+00	.0000E+00
42	2.0000E+02	.0000E+00	2.5000E+01
43	2.0000E+02	.0000E+00	5.0000E+01
44	2.0000E+02	.0000E+00	7.5000E+01
45	2.0000E+02	.0000E+00	1.0000E+02
51	2.5000E+02	.0000E+00	.0000E+00
52	2.5000E+02	.0000E+00	2.5000E+01
53	2.5000E+02	.0000E+00	5.0000E+01
54	2.5000E+02	.0000E+00	7.5000E+01
55	2.5000E+02	.0000E+00	1.0000E+02
61	3.0000E+02	.0000E+00	.0000E+00
62	3.0000E+02	.0000E+00	2.5000E+01
63	3.0000E+02	.0000E+00	5.0000E+01

64	3.0000E+02	.0000E+00	7.5000E+01
65	3.0000E+02	.0000E+00	1.0000E+02
71	3.5000E+02	.0000E+00	.0000E+00
72	3.5000E+02	.0000E+00	2.5000E+01
73	3.5000E+02	.0000E+00	5.0000E+01
74	3.5000E+02	.0000E+00	7.5000E+01
75	3.5000E+02	.0000E+00	1.0000E+02
81	4.0000E+02	.0000E+00	.0000E+00
82	4.0000E+02	.0000E+00	2.5000E+01
83	4.0000E+02	.0000E+00	5.0000E+01
84	4.0000E+02	.0000E+00	7.5000E+01
85	4.0000E+02	.0000E+00	1.0000E+02
91	4.5000E+02	.0000E+00	.0000E+00
92	4.5000E+02	.0000E+00	2.5000E+01
93	4.5000E+02	.0000E+00	5.0000E+01
94	4.5000E+02	.0000E+00	7.5000E+01
95	4.5000E+02	.0000E+00	1.0000E+02
101	5.0000E+02	.0000E+00	.0000E+00
102	5.0000E+02	.0000E+00	2.5000E+01
103	5.0000E+02	.0000E+00	5.0000E+01
104	5.0000E+02	.0000E+00	7.5000E+01
105	5.0000E+02	.0000E+00	1.0000E+02

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	11	12	2	1.0000E+00	1	0	0
2	2	12	13	3	1.0000E+00	1	0	0
3	3	13	14	4	1.0000E+00	1	0	0
4	4	14	15	5	1.0000E+00	1	0	0
11	11	21	22	12	1.0000E+00	1	0	0
12	12	22	23	13	1.0000E+00	1	0	0
13	13	23	24	14	1.0000E+00	1	0	0
14	14	24	25	15	1.0000E+00	1	0	0
21	21	31	32	22	1.0000E+00	1	0	0
22	22	32	33	23	1.0000E+00	1	0	0
23	23	33	34	24	1.0000E+00	1	0	0
24	24	34	35	25	1.0000E+00	1	0	0
31	31	41	42	32	1.0000E+00	1	0	0
32	32	42	43	33	1.0000E+00	1	0	0
33	33	43	44	34	1.0000E+00	1	0	0
34	34	44	45	35	1.0000E+00	1	0	0
41	41	51	52	42	1.0000E+00	1	0	0
42	42	52	53	43	1.0000E+00	1	0	0
43	43	53	54	44	1.0000E+00	1	0	0
44	44	54	55	45	1.0000E+00	1	0	0
51	51	61	62	52	1.0000E+00	1	0	0
52	52	62	63	53	1.0000E+00	1	0	0
53	53	63	64	54	1.0000E+00	1	0	0

54	54	64	65	55	1.0000E+00	1	0	0
61	61	71	72	62	1.0000E+00	1	0	0
62	62	72	73	63	1.0000E+00	1	0	0
63	63	73	74	64	1.0000E+00	1	0	0
64	64	74	75	65	1.0000E+00	1	0	0
71	71	81	82	72	1.0000E+00	1	0	0
72	72	82	83	73	1.0000E+00	1	0	0
73	73	83	84	74	1.0000E+00	1	0	0
74	74	84	85	75	1.0000E+00	1	0	0
81	81	91	92	82	1.0000E+00	1	0	0
82	82	92	93	83	1.0000E+00	1	0	0
83	83	93	94	84	1.0000E+00	1	0	0
84	84	94	95	85	1.0000E+00	1	0	0
91	91	101	102	92	1.0000E+00	1	0	0
92	92	102	103	93	1.0000E+00	1	0	0
93	93	103	104	94	1.0000E+00	1	0	0
94	94	104	105	95	1.0000E+00	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	3.0000E+07	3.0000E-01	3.6000E+04	2.8400E-01	6.5000E-06

N O D A L R E S T R A I N T S

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
1	1	1	0	0	0	0
2	1	1	0	0	0	0
3	1	1	1	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	0	0
11	0	1	0	0	0	0
15	0	1	0	0	0	0
21	0	1	0	0	0	0
25	0	1	0	0	0	0
31	0	1	0	0	0	0
35	0	1	0	0	0	0
41	0	1	0	0	0	0
45	0	1	0	0	0	0
51	0	1	0	0	0	0
55	0	1	0	0	0	0
61	0	1	0	0	0	0
65	0	1	0	0	0	0
71	0	1	0	0	0	0
75	0	1	0	0	0	0
81	0	1	0	0	0	0

85	0	1	0	0	0	0
91	0	1	0	0	0	0
95	0	1	0	0	0	0
101	1	1	0	0	0	0
102	1	1	0	0	0	0
103	1	1	1	0	1	0
104	1	1	0	0	0	0
105	1	1	0	0	0	0

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom . . . 330
 Number of restraints 42
 Number of equations 288

L O A D C A S E N U M B E R 1

Uniform Transverse Pressure - $P = 10\text{psi}$

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . 1
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . . 0

L O A D C A S E N U M B E R 2

Thermal Load - $dT = 100 \text{ deg.F.}$

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . 0
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . . 1

10 x 4 Mesh Plates - Inefficient Ordering

PLT10X4

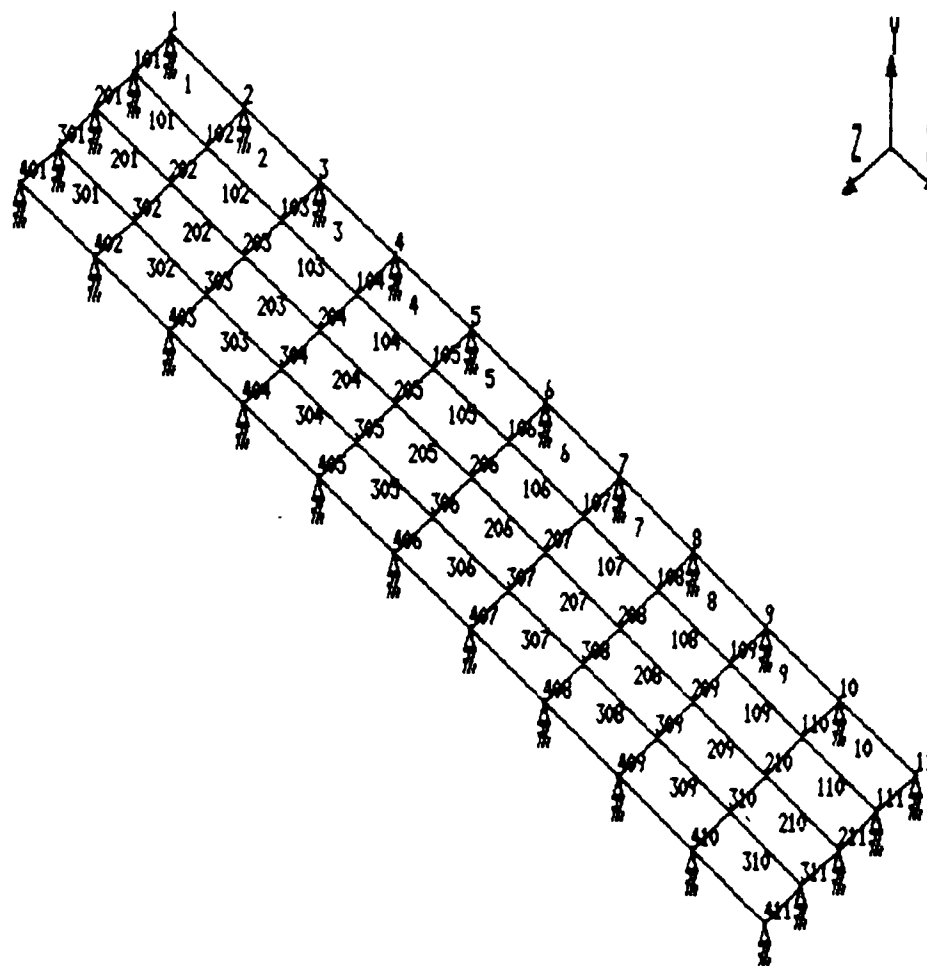


Figure B.4. Model PLT10X4.STR.

Model: PLT10X4.STR Title : 10 x 4 Mesh Plates - Inefficient Ordering

M O D E L P A R A M E T E R S

Number of nodes 55
 Number of restrained nodes . . 28
 Number of elements 40
 Trusses 0
 Beams 0
 Plates 40
 Number of sections 0
 Number of materials 1

N O D A L C O O R D I N A T E S

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
2	5.0000E+01	.0000E+00	.0000E+00
3	1.0000E+02	.0000E+00	.0000E+00
4	1.5000E+02	.0000E+00	.0000E+00
5	2.0000E+02	.0000E+00	.0000E+00
6	2.5000E+02	.0000E+00	.0000E+00
7	3.0000E+02	.0000E+00	.0000E+00
8	3.5000E+02	.0000E+00	.0000E+00
9	4.0000E+02	.0000E+00	.0000E+00
10	4.5000E+02	.0000E+00	.0000E+00
11	5.0000E+02	.0000E+00	.0000E+00
101	.0000E+00	.0000E+00	2.5000E+01
102	5.0000E+01	.0000E+00	2.5000E+01
103	1.0000E+02	.0000E+00	2.5000E+01
104	1.5000E+02	.0000E+00	2.5000E+01
105	2.0000E+02	.0000E+00	2.5000E+01
106	2.5000E+02	.0000E+00	2.5000E+01
107	3.0000E+02	.0000E+00	2.5000E+01
108	3.5000E+02	.0000E+00	2.5000E+01
109	4.0000E+02	.0000E+00	2.5000E+01
110	4.5000E+02	.0000E+00	2.5000E+01
111	5.0000E+02	.0000E+00	2.5000E+01
201	.0000E+00	.0000E+00	5.0000E+01
202	5.0000E+01	.0000E+00	5.0000E+01
203	1.0000E+02	.0000E+00	5.0000E+01
204	1.5000E+02	.0000E+00	5.0000E+01
205	2.0000E+02	.0000E+00	5.0000E+01
206	2.5000E+02	.0000E+00	5.0000E+01
207	3.0000E+02	.0000E+00	5.0000E+01
208	3.5000E+02	.0000E+00	5.0000E+01
209	4.0000E+02	.0000E+00	5.0000E+01
210	4.5000E+02	.0000E+00	5.0000E+01
211	5.0000E+02	.0000E+00	5.0000E+01

301	.0000E+00	.0000E+00	7.5000E+01
302	5.0000E+01	.0000E+00	7.5000E+01
303	1.0000E+02	.0000E+00	7.5000E+01
304	1.5000E+02	.0000E+00	7.5000E+01
305	2.0000E+02	.0000E+00	7.5000E+01
306	2.5000E+02	.0000E+00	7.5000E+01
307	3.0000E+02	.0000E+00	7.5000E+01
308	3.5000E+02	.0000E+00	7.5000E+01
309	4.0000E+02	.0000E+00	7.5000E+01
310	4.5000E+02	.0000E+00	7.5000E+01
311	5.0000E+02	.0000E+00	7.5000E+01
401	.0000E+00	.0000E+00	1.0000E+02
402	5.0000E+01	.0000E+00	1.0000E+02
403	1.0000E+02	.0000E+00	1.0000E+02
404	1.5000E+02	.0000E+00	1.0000E+02
405	2.0000E+02	.0000E+00	1.0000E+02
406	2.5000E+02	.0000E+00	1.0000E+02
407	3.0000E+02	.0000E+00	1.0000E+02
408	3.5000E+02	.0000E+00	1.0000E+02
409	4.0000E+02	.0000E+00	1.0000E+02
410	4.5000E+02	.0000E+00	1.0000E+02
411	5.0000E+02	.0000E+00	1.0000E+02

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	2	102	101	1.0000E+00	1	0	0
2	2	3	103	102	1.0000E+00	1	0	0
3	3	4	104	103	1.0000E+00	1	0	0
4	4	5	105	104	1.0000E+00	1	0	0
5	5	6	106	105	1.0000E+00	1	0	0
6	6	7	107	106	1.0000E+00	1	0	0
7	7	8	108	107	1.0000E+00	1	0	0
8	8	9	109	108	1.0000E+00	1	0	0
9	9	10	110	109	1.0000E+00	1	0	0
10	10	11	111	110	1.0000E+00	1	0	0
101	101	102	202	201	1.0000E+00	1	0	0
102	102	103	203	202	1.0000E+00	1	0	0
103	103	104	204	203	1.0000E+00	1	0	0
104	104	105	205	204	1.0000E+00	1	0	0
105	105	106	206	205	1.0000E+00	1	0	0
106	106	107	207	206	1.0000E+00	1	0	0
107	107	108	208	207	1.0000E+00	1	0	0
108	108	109	209	208	1.0000E+00	1	0	0
109	109	110	210	209	1.0000E+00	1	0	0
110	110	111	211	210	1.0000E+00	1	0	0
201	201	202	302	301	1.0000E+00	1	0	0
202	202	203	303	302	1.0000E+00	1	0	0
203	203	204	304	303	1.0000E+00	1	0	0

204	204	205	305	304	1.0000E+00	1	0	0
205	205	206	306	305	1.0000E+00	1	0	0
206	206	207	307	306	1.0000E+00	1	0	0
207	207	208	308	307	1.0000E+00	1	0	0
208	208	209	309	308	1.0000E+00	1	0	0
209	209	210	310	309	1.0000E+00	1	0	0
210	210	211	311	310	1.0000E+00	1	0	0
301	301	302	402	401	1.0000E+00	1	0	0
302	302	303	403	402	1.0000E+00	1	0	0
303	303	304	404	403	1.0000E+00	1	0	0
304	304	305	405	404	1.0000E+00	1	0	0
305	305	306	406	405	1.0000E+00	1	0	0
306	306	307	407	406	1.0000E+00	1	0	0
307	307	308	408	407	1.0000E+00	1	0	0
308	308	309	409	408	1.0000E+00	1	0	0
309	309	310	410	409	1.0000E+00	1	0	0
310	310	311	411	410	1.0000E+00	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	3.0000E+07	3.0000E-01	3.6000E+04	2.8400E-01	6.5000E-06

N O D A L R E S T R A I N T S

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
1	1	1	0	0	0	0
2	0	1	0	0	0	0
3	0	1	0	0	0	0
4	0	1	0	0	0	0
5	0	1	0	0	0	0
6	0	1	0	0	0	0
7	0	1	0	0	0	0
8	0	1	0	0	0	0
9	0	1	0	0	0	0
10	0	1	0	0	0	0
11	1	1	0	0	0	0
101	1	1	0	0	0	0
111	1	1	0	0	0	0
201	1	1	1	0	1	0
211	1	1	1	0	1	0
301	1	1	0	0	0	0
311	1	1	0	0	0	0
401	1	1	0	0	0	0
402	0	1	0	0	0	0
403	0	1	0	0	0	0

404	0	1	0	0	0	0
405	0	1	0	0	0	0
406	0	1	0	0	0	0
407	0	1	0	0	0	0
408	0	1	0	0	0	0
409	0	1	0	0	0	0
410	0	1	0	0	0	0
411	1	1	0	0	0	0

SOLUTION PARAMETERS

Number of degrees of freedom . . . 330
 Number of restraints 42
 Number of equations 288

LOAD CASE NUMBER 1

Uniform Transverse Pressure - $P = 10\text{psi}$

LOAD CASE PARAMETERS

Number of loaded nodes 0
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . 1
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . . 0

LOAD CASE NUMBER 2

Thermal Load - $dT = 100 \text{ deg.F.}$

LOAD CASE PARAMETERS

Number of loaded nodes 0
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . 0
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . . 1

Plate Model 10 x 10 Mesh

PL10X10

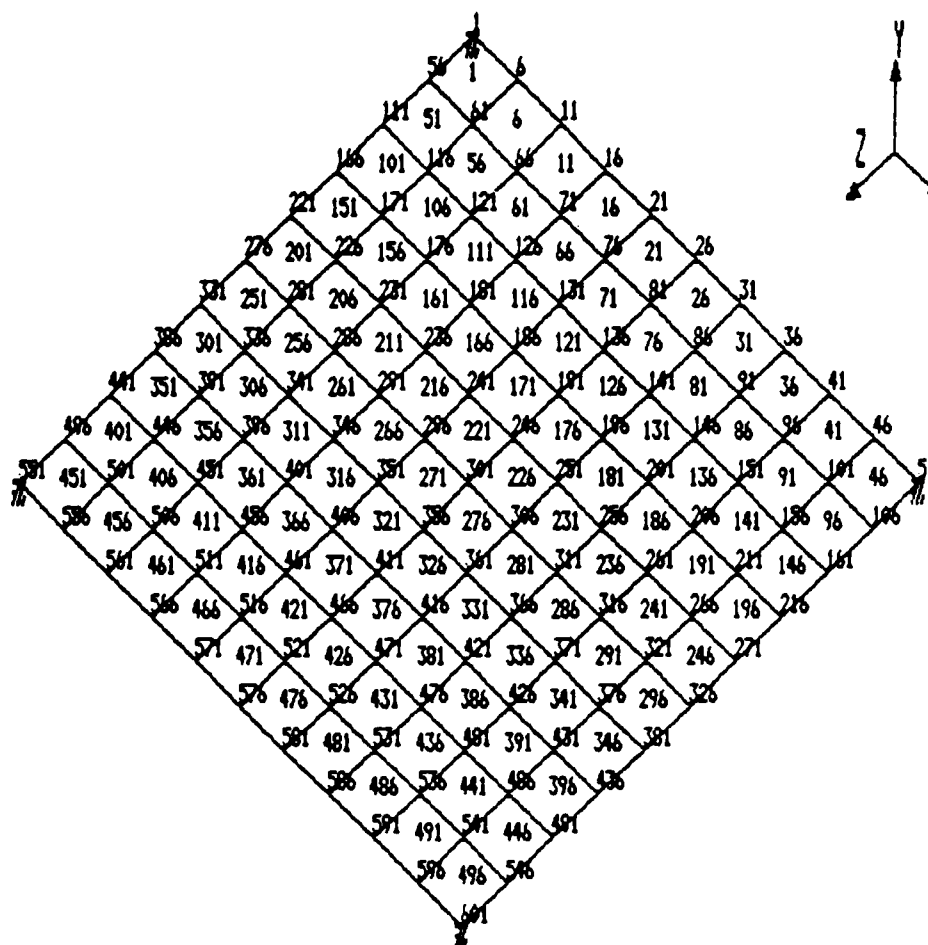


Figure B.5. Model PL10X10.STR.

Model: PL10X10.STR Title: Plate Model 10 x 10 Mesh

MODEL PARAMETERS

Number of nodes 121
 Number of restrained nodes . . . 4
 Number of elements 100
 Trusses 0
 Beams 0
 Plates 100
 Number of sections 0
 Number of materials 1

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
6	1.0000E+00	.0000E+00	.0000E+00
11	2.0000E+00	.0000E+00	.0000E+00
16	3.0000E+00	.0000E+00	.0000E+00
21	4.0000E+00	.0000E+00	.0000E+00
26	5.0000E+00	.0000E+00	.0000E+00
31	6.0000E+00	.0000E+00	.0000E+00
36	7.0000E+00	.0000E+00	.0000E+00
41	8.0000E+00	.0000E+00	.0000E+00
46	9.0000E+00	.0000E+00	.0000E+00
51	1.0000E+01	.0000E+00	.0000E+00
56	.0000E+00	.0000E+00	1.0000E+00
61	1.0000E+00	.0000E+00	1.0000E+00
66	2.0000E+00	.0000E+00	1.0000E+00
71	3.0000E+00	.0000E+00	1.0000E+00
76	4.0000E+00	.0000E+00	1.0000E+00
81	5.0000E+00	.0000E+00	1.0000E+00
86	6.0000E+00	.0000E+00	1.0000E+00
91	7.0000E+00	.0000E+00	1.0000E+00
96	8.0000E+00	.0000E+00	1.0000E+00
101	9.0000E+00	.0000E+00	1.0000E+00
106	1.0000E+01	.0000E+00	1.0000E+00
111	.0000E+00	.0000E+00	2.0000E+00
116	1.0000E+00	.0000E+00	2.0000E+00
121	2.0000E+00	.0000E+00	2.0000E+00
126	3.0000E+00	.0000E+00	2.0000E+00
131	4.0000E+00	.0000E+00	2.0000E+00
136	5.0000E+00	.0000E+00	2.0000E+00
141	6.0000E+00	.0000E+00	2.0000E+00
146	7.0000E+00	.0000E+00	2.0000E+00
151	8.0000E+00	.0000E+00	2.0000E+00
156	9.0000E+00	.0000E+00	2.0000E+00
161	1.0000E+01	.0000E+00	2.0000E+00

166	.0000E+00	.0000E+00	3.0000E+00
171	1.0000E+00	.0000E+00	3.0000E+00
176	2.0000E+00	.0000E+00	3.0000E+00
181	3.0000E+00	.0000E+00	3.0000E+00
186	4.0000E+00	.0000E+00	3.0000E+00
191	5.0000E+00	.0000E+00	3.0000E+00
196	6.0000E+00	.0000E+00	3.0000E+00
201	7.0000E+00	.0000E+00	3.0000E+00
206	8.0000E+00	.0000E+00	3.0000E+00
211	9.0000E+00	.0000E+00	3.0000E+00
216	1.0000E+01	.0000E+00	3.0000E+00
221	.0000E+00	.0000E+00	4.0000E+00
226	1.0000E+00	.0000E+00	4.0000E+00
231	2.0000E+00	.0000E+00	4.0000E+00
236	3.0000E+00	.0000E+00	4.0000E+00
241	4.0000E+00	.0000E+00	4.0000E+00
246	5.0000E+00	.0000E+00	4.0000E+00
251	6.0000E+00	.0000E+00	4.0000E+00
256	7.0000E+00	.0000E+00	4.0000E+00
261	8.0000E+00	.0000E+00	4.0000E+00
266	9.0000E+00	.0000E+00	4.0000E+00
271	1.0000E+01	.0000E+00	4.0000E+00
276	.0000E+00	.0000E+00	5.0000E+00
281	1.0000E+00	.0000E+00	5.0000E+00
286	2.0000E+00	.0000E+00	5.0000E+00
291	3.0000E+00	.0000E+00	5.0000E+00
296	4.0000E+00	.0000E+00	5.0000E+00
301	5.0000E+00	.0000E+00	5.0000E+00
306	6.0000E+00	.0000E+00	5.0000E+00
311	7.0000E+00	.0000E+00	5.0000E+00
316	8.0000E+00	.0000E+00	5.0000E+00
321	9.0000E+00	.0000E+00	5.0000E+00
326	1.0000E+01	.0000E+00	5.0000E+00
331	.0000E+00	.0000E+00	6.0000E+00
336	1.0000E+00	.0000E+00	6.0000E+00
341	2.0000E+00	.0000E+00	6.0000E+00
346	3.0000E+00	.0000E+00	6.0000E+00
351	4.0000E+00	.0000E+00	6.0000E+00
356	5.0000E+00	.0000E+00	6.0000E+00
361	6.0000E+00	.0000E+00	6.0000E+00
366	7.0000E+00	.0000E+00	6.0000E+00
371	8.0000E+00	.0000E+00	6.0000E+00
376	9.0000E+00	.0000E+00	6.0000E+00
381	1.0000E+01	.0000E+00	6.0000E+00
386	.0000E+00	.0000E+00	7.0000E+00
391	1.0000E+00	.0000E+00	7.0000E+00
396	2.0000E+00	.0000E+00	7.0000E+00
401	3.0000E+00	.0000E+00	7.0000E+00
406	4.0000E+00	.0000E+00	7.0000E+00
411	5.0000E+00	.0000E+00	7.0000E+00
416	6.0000E+00	.0000E+00	7.0000E+00
421	7.0000E+00	.0000E+00	7.0000E+00

426	8.0000E+00	.0000E+00	7.0000E+00
431	9.0000E+00	.0000E+00	7.0000E+00
436	1.0000E+01	.0000E+00	7.0000E+00
441	.0000E+00	.0000E+00	8.0000E+00
446	1.0000E+00	.0000E+00	8.0000E+00
451	2.0000E+00	.0000E+00	8.0000E+00
456	3.0000E+00	.0000E+00	8.0000E+00
461	4.0000E+00	.0000E+00	8.0000E+00
466	5.0000E+00	.0000E+00	8.0000E+00
471	6.0000E+00	.0000E+00	8.0000E+00
476	7.0000E+00	.0000E+00	8.0000E+00
481	8.0000E+00	.0000E+00	8.0000E+00
486	9.0000E+00	.0000E+00	8.0000E+00
491	1.0000E+01	.0000E+00	8.0000E+00
496	.0000E+00	.0000E+00	9.0000E+00
501	1.0000E+00	.0000E+00	9.0000E+00
506	2.0000E+00	.0000E+00	9.0000E+00
511	3.0000E+00	.0000E+00	9.0000E+00
516	4.0000E+00	.0000E+00	9.0000E+00
521	5.0000E+00	.0000E+00	9.0000E+00
526	6.0000E+00	.0000E+00	9.0000E+00
531	7.0000E+00	.0000E+00	9.0000E+00
536	8.0000E+00	.0000E+00	9.0000E+00
541	9.0000E+00	.0000E+00	9.0000E+00
546	1.0000E+01	.0000E+00	9.0000E+00
551	.0000E+00	.0000E+00	1.0000E+01
556	1.0000E+00	.0000E+00	1.0000E+01
561	2.0000E+00	.0000E+00	1.0000E+01
566	3.0000E+00	.0000E+00	1.0000E+01
571	4.0000E+00	.0000E+00	1.0000E+01
576	5.0000E+00	.0000E+00	1.0000E+01
581	6.0000E+00	.0000E+00	1.0000E+01
586	7.0000E+00	.0000E+00	1.0000E+01
591	8.0000E+00	.0000E+00	1.0000E+01
596	9.0000E+00	.0000E+00	1.0000E+01
601	1.0000E+01	.0000E+00	1.0000E+01

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	6	61	56	1.0000E-01	1	0	0
6	6	11	66	61	1.0000E-01	1	0	0
11	11	16	71	66	1.0000E-01	1	0	0
16	16	21	76	71	1.0000E-01	1	0	0
21	21	26	81	76	1.0000E-01	1	0	0
26	26	31	86	81	1.0000E-01	1	0	0
31	31	36	91	86	1.0000E-01	1	0	0
36	36	41	96	91	1.0000E-01	1	0	0
41	41	46	101	96	1.0000E-01	1	0	0

46	46	51	106	101	1.0000E-01	1	0	0
51	56	61	116	111	1.0000E-01	1	0	0
56	61	66	121	116	1.0000E-01	1	0	0
61	66	71	126	121	1.0000E-01	1	0	0
66	71	76	131	126	1.0000E-01	1	0	0
71	76	81	136	131	1.0000E-01	1	0	0
76	81	86	141	136	1.0000E-01	1	0	0
81	86	91	146	141	1.0000E-01	1	0	0
86	91	96	151	146	1.0000E-01	1	0	0
91	96	101	156	151	1.0000E-01	1	0	0
96	101	106	161	156	1.0000E-01	1	0	0
101	111	116	171	166	1.0000E-01	1	0	0
106	116	121	176	171	1.0000E-01	1	0	0
111	121	126	181	176	1.0000E-01	1	0	0
116	126	131	186	181	1.0000E-01	1	0	0
121	131	136	191	186	1.0000E-01	1	0	0
126	136	141	196	191	1.0000E-01	1	0	0
131	141	146	201	196	1.0000E-01	1	0	0
136	146	151	206	201	1.0000E-01	1	0	0
141	151	156	211	206	1.0000E-01	1	0	0
146	156	161	216	211	1.0000E-01	1	0	0
151	166	171	226	221	1.0000E-01	1	0	0
156	171	176	231	226	1.0000E-01	1	0	0
161	176	181	236	231	1.0000E-01	1	0	0
166	181	186	241	236	1.0000E-01	1	0	0
171	186	191	246	241	1.0000E-01	1	0	0
176	191	196	251	246	1.0000E-01	1	0	0
181	196	201	256	251	1.0000E-01	1	0	0
186	201	206	261	256	1.0000E-01	1	0	0
191	206	211	266	261	1.0000E-01	1	0	0
196	211	216	271	266	1.0000E-01	1	0	0
201	221	226	281	276	1.0000E-01	1	0	0
206	226	231	286	281	1.0000E-01	1	0	0
211	231	236	291	286	1.0000E-01	1	0	0
216	236	241	296	291	1.0000E-01	1	0	0
221	241	246	301	296	1.0000E-01	1	0	0
226	246	251	306	301	1.0000E-01	1	0	0
231	251	256	311	306	1.0000E-01	1	0	0
236	256	261	316	311	1.0000E-01	1	0	0
241	261	266	321	316	1.0000E-01	1	0	0
246	266	271	326	321	1.0000E-01	1	0	0
251	276	281	336	331	1.0000E-01	1	0	0
256	281	286	341	336	1.0000E-01	1	0	0
261	286	291	346	341	1.0000E-01	1	0	0
266	291	296	351	346	1.0000E-01	1	0	0
271	296	301	356	351	1.0000E-01	1	0	0
276	301	306	361	356	1.0000E-01	1	0	0
281	306	311	366	361	1.0000E-01	1	0	0
286	311	316	371	366	1.0000E-01	1	0	0
291	316	321	376	371	1.0000E-01	1	0	0
296	321	326	381	376	1.0000E-01	1	0	0
301	331	336	391	386	1.0000E-01	1	0	0

306	336	341	396	391	1.0000E-01	1	0	0
311	341	346	401	396	1.0000E-01	1	0	0
316	346	351	406	401	1.0000E-01	1	0	0
321	351	356	411	406	1.0000E-01	1	0	0
326	356	361	416	411	1.0000E-01	1	0	0
331	361	366	421	416	1.0000E-01	1	0	0
336	366	371	426	421	1.0000E-01	1	0	0
341	371	376	431	426	1.0000E-01	1	0	0
346	376	381	436	431	1.0000E-01	1	0	0
351	386	391	446	441	1.0000E-01	1	0	0
356	391	396	451	446	1.0000E-01	1	0	0
361	396	401	456	451	1.0000E-01	1	0	0
366	401	406	461	456	1.0000E-01	1	0	0
371	406	411	466	461	1.0000E-01	1	0	0
376	411	416	471	466	1.0000E-01	1	0	0
381	416	421	476	471	1.0000E-01	1	0	0
386	421	426	481	476	1.0000E-01	1	0	0
391	426	431	486	481	1.0000E-01	1	0	0
396	431	436	491	486	1.0000E-01	1	0	0
401	441	446	501	496	1.0000E-01	1	0	0
406	446	451	506	501	1.0000E-01	1	0	0
411	451	456	511	506	1.0000E-01	1	0	0
416	456	461	516	511	1.0000E-01	1	0	0
421	461	466	521	516	1.0000E-01	1	0	0
426	466	471	526	521	1.0000E-01	1	0	0
431	471	476	531	526	1.0000E-01	1	0	0
436	476	481	536	531	1.0000E-01	1	0	0
441	481	486	541	536	1.0000E-01	1	0	0
446	486	491	546	541	1.0000E-01	1	0	0
451	496	501	556	551	1.0000E-01	1	0	0
456	501	506	561	556	1.0000E-01	1	0	0
461	506	511	566	561	1.0000E-01	1	0	0
466	511	516	571	566	1.0000E-01	1	0	0
471	516	521	576	571	1.0000E-01	1	0	0
476	521	526	581	576	1.0000E-01	1	0	0
481	526	531	586	581	1.0000E-01	1	0	0
486	531	536	591	586	1.0000E-01	1	0	0
491	536	541	596	591	1.0000E-01	1	0	0
496	541	546	601	596	1.0000E-01	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	1.0000E+06	3.0000E-01	1.0000E+05	1.0000E+00	1.0000E-05

N O D A L R E S T R A I N T S

Node --Translations- ---Rotations---

Number	X	Y	Z	X	Y	Z
1	1	1	1	1	1	1
51	1	1	1	1	1	1
551	1	1	1	1	1	1
601	1	1	1	1	1	1

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom . . . 726
 Number of restraints 24
 Number of equations 702

L O A D C A S E N U M B E R 1

1000 Lb load at center :

Node Number	-----Forces-----			-----Moments-----		
	X	Y	Z	X	Y	Z
301	.0000E+00	1.0000E+03	.0000E+00	.0000E+00	.0000E+00	.0000

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 1
 Number of nodes with applied displacements . . . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . . . 0
 Number of elements with concentrated loads . . . 0
 Number of elements with thermal loads 0

L O A D C A S E N U M B E R 2

Uniform Transvers Pressure P = 1 psi

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
 Number of nodes with applied displacements . . . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . . . 1
 Number of elements with concentrated loads . . . 0
 Number of elements with thermal loads 0

Plate Model 10 x 10 Mesh

PL10X10A

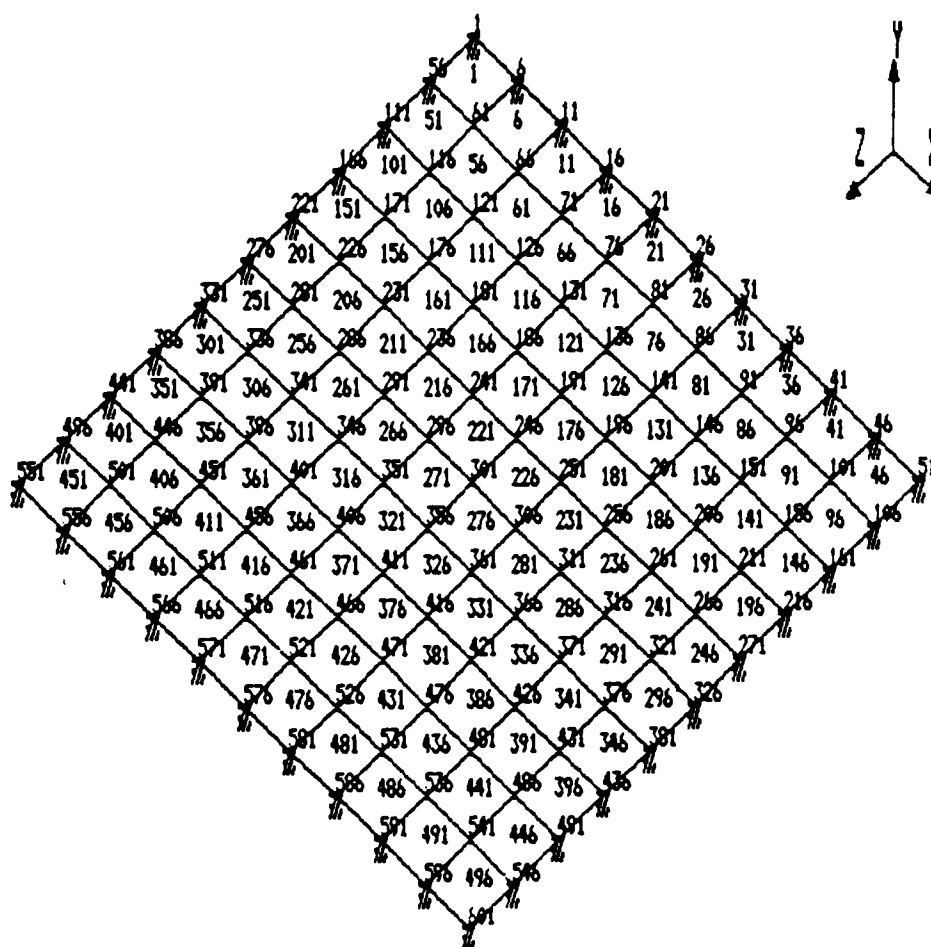


Figure B.6. Model PL10X10A.STR.

Model: PL10X10A.STR Title: Plate Model 10 x 10 Mesh

MODEL PARAMETERS

Number of nodes 121
 Number of restrained nodes . . 40
 Number of elements 100
 Trusses 0
 Beams 0
 Plates 100
 Number of sections 0
 Number of materials 1

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
6	1.0000E+00	.0000E+00	.0000E+00
11	2.0000E+00	.0000E+00	.0000E+00
16	3.0000E+00	.0000E+00	.0000E+00
21	4.0000E+00	.0000E+00	.0000E+00
26	5.0000E+00	.0000E+00	.0000E+00
31	6.0000E+00	.0000E+00	.0000E+00
36	7.0000E+00	.0000E+00	.0000E+00
41	8.0000E+00	.0000E+00	.0000E+00
46	9.0000E+00	.0000E+00	.0000E+00
51	1.0000E+01	.0000E+00	.0000E+00
56	.0000E+00	.0000E+00	1.0000E+00
61	1.0000E+00	.0000E+00	1.0000E+00
66	2.0000E+00	.0000E+00	1.0000E+00
71	3.0000E+00	.0000E+00	1.0000E+00
76	4.0000E+00	.0000E+00	1.0000E+00
81	5.0000E+00	.0000E+00	1.0000E+00
86	6.0000E+00	.0000E+00	1.0000E+00
91	7.0000E+00	.0000E+00	1.0000E+00
96	8.0000E+00	.0000E+00	1.0000E+00
101	9.0000E+00	.0000E+00	1.0000E+00
106	1.0000E+01	.0000E+00	1.0000E+00
111	.0000E+00	.0000E+00	2.0000E+00
116	1.0000E+00	.0000E+00	2.0000E+00
121	2.0000E+00	.0000E+00	2.0000E+00
126	3.0000E+00	.0000E+00	2.0000E+00
131	4.0000E+00	.0000E+00	2.0000E+00
136	5.0000E+00	.0000E+00	2.0000E+00
141	6.0000E+00	.0000E+00	2.0000E+00
146	7.0000E+00	.0000E+00	2.0000E+00
151	8.0000E+00	.0000E+00	2.0000E+00
156	9.0000E+00	.0000E+00	2.0000E+00
161	1.0000E+01	.0000E+00	2.0000E+00

166	.0000E+00	.0000E+00	3.0000E+00
171	1.0000E+00	.0000E+00	3.0000E+00
176	2.0000E+00	.0000E+00	3.0000E+00
181	3.0000E+00	.0000E+00	3.0000E+00
186	4.0000E+00	.0000E+00	3.0000E+00
191	5.0000E+00	.0000E+00	3.0000E+00
196	6.0000E+00	.0000E+00	3.0000E+00
201	7.0000E+00	.0000E+00	3.0000E+00
206	8.0000E+00	.0000E+00	3.0000E+00
211	9.0000E+00	.0000E+00	3.0000E+00
216	1.0000E+01	.0000E+00	3.0000E+00
221	.0000E+00	.0000E+00	4.0000E+00
226	1.0000E+00	.0000E+00	4.0000E+00
231	2.0000E+00	.0000E+00	4.0000E+00
236	3.0000E+00	.0000E+00	4.0000E+00
241	4.0000E+00	.0000E+00	4.0000E+00
246	5.0000E+00	.0000E+00	4.0000E+00
251	6.0000E+00	.0000E+00	4.0000E+00
256	7.0000E+00	.0000E+00	4.0000E+00
261	8.0000E+00	.0000E+00	4.0000E+00
266	9.0000E+00	.0000E+00	4.0000E+00
271	1.0000E+01	.0000E+00	4.0000E+00
276	.0000E+00	.0000E+00	5.0000E+00
281	1.0000E+00	.0000E+00	5.0000E+00
286	2.0000E+00	.0000E+00	5.0000E+00
291	3.0000E+00	.0000E+00	5.0000E+00
296	4.0000E+00	.0000E+00	5.0000E+00
301	5.0000E+00	.0000E+00	5.0000E+00
306	6.0000E+00	.0000E+00	5.0000E+00
311	7.0000E+00	.0000E+00	5.0000E+00
316	8.0000E+00	.0000E+00	5.0000E+00
321	9.0000E+00	.0000E+00	5.0000E+00
326	1.0000E+01	.0000E+00	5.0000E+00
331	.0000E+00	.0000E+00	6.0000E+00
336	1.0000E+00	.0000E+00	6.0000E+00
341	2.0000E+00	.0000E+00	6.0000E+00
346	3.0000E+00	.0000E+00	6.0000E+00
351	4.0000E+00	.0000E+00	6.0000E+00
356	5.0000E+00	.0000E+00	6.0000E+00
361	6.0000E+00	.0000E+00	6.0000E+00
366	7.0000E+00	.0000E+00	6.0000E+00
371	8.0000E+00	.0000E+00	6.0000E+00
376	9.0000E+00	.0000E+00	6.0000E+00
381	1.0000E+01	.0000E+00	6.0000E+00
386	.0000E+00	.0000E+00	7.0000E+00
391	1.0000E+00	.0000E+00	7.0000E+00
396	2.0000E+00	.0000E+00	7.0000E+00
401	3.0000E+00	.0000E+00	7.0000E+00
406	4.0000E+00	.0000E+00	7.0000E+00
411	5.0000E+00	.0000E+00	7.0000E+00
416	6.0000E+00	.0000E+00	7.0000E+00
421	7.0000E+00	.0000E+00	7.0000E+00

426	8.0000E+00	.0000E+00	7.0000E+00
431	9.0000E+00	.0000E+00	7.0000E+00
436	1.0000E+01	.0000E+00	7.0000E+00
441	.0000E+00	.0000E+00	8.0000E+00
446	1.0000E+00	.0000E+00	8.0000E+00
451	2.0000E+00	.0000E+00	8.0000E+00
456	3.0000E+00	.0000E+00	8.0000E+00
461	4.0000E+00	.0000E+00	8.0000E+00
466	5.0000E+00	.0000E+00	8.0000E+00
471	6.0000E+00	.0000E+00	8.0000E+00
476	7.0000E+00	.0000E+00	8.0000E+00
481	8.0000E+00	.0000E+00	8.0000E+00
486	9.0000E+00	.0000E+00	8.0000E+00
491	1.0000E+01	.0000E+00	8.0000E+00
496	.0000E+00	.0000E+00	9.0000E+00
501	1.0000E+00	.0000E+00	9.0000E+00
506	2.0000E+00	.0000E+00	9.0000E+00
511	3.0000E+00	.0000E+00	9.0000E+00
516	4.0000E+00	.0000E+00	9.0000E+00
521	5.0000E+00	.0000E+00	9.0000E+00
526	6.0000E+00	.0000E+00	9.0000E+00
531	7.0000E+00	.0000E+00	9.0000E+00
536	8.0000E+00	.0000E+00	9.0000E+00
541	9.0000E+00	.0000E+00	9.0000E+00
546	1.0000E+01	.0000E+00	9.0000E+00
551	.0000E+00	.0000E+00	1.0000E+01
556	1.0000E+00	.0000E+00	1.0000E+01
561	2.0000E+00	.0000E+00	1.0000E+01
566	3.0000E+00	.0000E+00	1.0000E+01
571	4.0000E+00	.0000E+00	1.0000E+01
576	5.0000E+00	.0000E+00	1.0000E+01
581	6.0000E+00	.0000E+00	1.0000E+01
586	7.0000E+00	.0000E+00	1.0000E+01
591	8.0000E+00	.0000E+00	1.0000E+01
596	9.0000E+00	.0000E+00	1.0000E+01
601	1.0000E+01	.0000E+00	1.0000E+01

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	6	61	56	1.0000E-01	1	0	0
6	6	11	66	61	1.0000E-01	1	0	0
11	11	16	71	66	1.0000E-01	1	0	0
16	16	21	76	71	1.0000E-01	1	0	0
21	21	26	81	76	1.0000E-01	1	0	0
26	26	31	86	81	1.0000E-01	1	0	0
31	31	36	91	86	1.0000E-01	1	0	0
36	36	41	96	91	1.0000E-01	1	0	0
41	41	46	101	96	1.0000E-01	1	0	0

46	46	51	106	101	1.0000E-01	1	0	0
51	56	61	116	111	1.0000E-01	1	0	0
56	61	66	121	116	1.0000E-01	1	0	0
61	66	71	126	121	1.0000E-01	1	0	0
66	71	76	131	126	1.0000E-01	1	0	0
71	76	81	136	131	1.0000E-01	1	0	0
76	81	86	141	136	1.0000E-01	1	0	0
81	86	91	146	141	1.0000E-01	1	0	0
86	91	96	151	146	1.0000E-01	1	0	0
91	96	101	156	151	1.0000E-01	1	0	0
96	101	106	161	156	1.0000E-01	1	0	0
101	111	116	171	166	1.0000E-01	1	0	0
106	116	121	176	171	1.0000E-01	1	0	0
111	121	126	181	176	1.0000E-01	1	0	0
116	126	131	186	181	1.0000E-01	1	0	0
121	131	136	191	186	1.0000E-01	1	0	0
126	136	141	196	191	1.0000E-01	1	0	0
131	141	146	201	196	1.0000E-01	1	0	0
136	146	151	206	201	1.0000E-01	1	0	0
141	151	156	211	206	1.0000E-01	1	0	0
146	156	161	216	211	1.0000E-01	1	0	0
151	166	171	226	221	1.0000E-01	1	0	0
156	171	176	231	226	1.0000E-01	1	0	0
161	176	181	236	231	1.0000E-01	1	0	0
166	181	186	241	236	1.0000E-01	1	0	0
171	186	191	246	241	1.0000E-01	1	0	0
176	191	196	251	246	1.0000E-01	1	0	0
181	196	201	256	251	1.0000E-01	1	0	0
186	201	206	261	256	1.0000E-01	1	0	0
191	206	211	266	261	1.0000E-01	1	0	0
196	211	216	271	266	1.0000E-01	1	0	0
201	221	226	281	276	1.0000E-01	1	0	0
206	226	231	286	281	1.0000E-01	1	0	0
211	231	236	291	286	1.0000E-01	1	0	0
216	236	241	296	291	1.0000E-01	1	0	0
221	241	246	301	296	1.0000E-01	1	0	0
226	246	251	306	301	1.0000E-01	1	0	0
231	251	256	311	306	1.0000E-01	1	0	0
236	256	261	316	311	1.0000E-01	1	0	0
241	261	266	321	316	1.0000E-01	1	0	0
246	266	271	326	321	1.0000E-01	1	0	0
251	276	281	336	331	1.0000E-01	1	0	0
256	281	286	341	336	1.0000E-01	1	0	0
261	286	291	346	341	1.0000E-01	1	0	0
266	291	296	351	346	1.0000E-01	1	0	0
271	296	301	356	351	1.0000E-01	1	0	0
276	301	306	361	356	1.0000E-01	1	0	0
281	306	311	366	361	1.0000E-01	1	0	0
286	311	316	371	366	1.0000E-01	1	0	0
291	316	321	376	371	1.0000E-01	1	0	0
296	321	326	381	376	1.0000E-01	1	0	0
301	331	336	391	386	1.0000E-01	1	0	0

306	336	341	396	391	1.0000E-01	1	0	0
311	341	346	401	396	1.0000E-01	1	0	0
316	346	351	406	401	1.0000E-01	1	0	0
321	351	356	411	406	1.0000E-01	1	0	0
326	356	361	416	411	1.0000E-01	1	0	0
331	361	366	421	416	1.0000E-01	1	0	0
336	366	371	426	421	1.0000E-01	1	0	0
341	371	376	431	426	1.0000E-01	1	0	0
346	376	381	436	431	1.0000E-01	1	0	0
351	386	391	446	441	1.0000E-01	1	0	0
356	391	396	451	446	1.0000E-01	1	0	0
361	396	401	456	451	1.0000E-01	1	0	0
366	401	406	461	456	1.0000E-01	1	0	0
371	406	411	466	461	1.0000E-01	1	0	0
376	411	416	471	466	1.0000E-01	1	0	0
381	416	421	476	471	1.0000E-01	1	0	0
386	421	426	481	476	1.0000E-01	1	0	0
391	426	431	486	481	1.0000E-01	1	0	0
396	431	436	491	486	1.0000E-01	1	0	0
401	441	446	501	496	1.0000E-01	1	0	0
406	446	451	506	501	1.0000E-01	1	0	0
411	451	456	511	506	1.0000E-01	1	0	0
416	456	461	516	511	1.0000E-01	1	0	0
421	461	466	521	516	1.0000E-01	1	0	0
426	466	471	526	521	1.0000E-01	1	0	0
431	471	476	531	526	1.0000E-01	1	0	0
436	476	481	536	531	1.0000E-01	1	0	0
441	481	486	541	536	1.0000E-01	1	0	0
446	486	491	546	541	1.0000E-01	1	0	0
451	496	501	556	551	1.0000E-01	1	0	0
456	501	506	561	556	1.0000E-01	1	0	0
461	506	511	566	561	1.0000E-01	1	0	0
466	511	516	571	566	1.0000E-01	1	0	0
471	516	521	576	571	1.0000E-01	1	0	0
476	521	526	581	576	1.0000E-01	1	0	0
481	526	531	586	581	1.0000E-01	1	0	0
486	531	536	591	586	1.0000E-01	1	0	0
491	536	541	596	591	1.0000E-01	1	0	0
496	541	546	601	596	1.0000E-01	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	1.0000E+06	3.0000E-01	1.0000E+05	1.0000E+00	1.0000E-05

N O D A L R E S T R A I N T S

Node --Translations- ---Rotations---

Number	X	Y	Z	X	Y	Z
1	1	1	1	1	1	1
6	1	1	1	1	1	1
11	1	1	1	1	1	1
16	1	1	1	1	1	1
21	1	1	1	1	1	1
26	1	1	1	1	1	1
31	1	1	1	1	1	1
36	1	1	1	1	1	1
41	1	1	1	1	1	1
46	1	1	1	1	1	1
51	1	1	1	1	1	1
56	1	1	1	1	1	1
106	1	1	1	1	1	1
111	1	1	1	1	1	1
161	1	1	1	1	1	1
166	1	1	1	1	1	1
216	1	1	1	1	1	1
221	1	1	1	1	1	1
271	1	1	1	1	1	1
276	1	1	1	1	1	1
326	1	1	1	1	1	1
331	1	1	1	1	1	1
381	1	1	1	1	1	1
386	1	1	1	1	1	1
436	1	1	1	1	1	1
441	1	1	1	1	1	1
491	1	1	1	1	1	1
496	1	1	1	1	1	1
546	1	1	1	1	1	1
551	1	1	1	1	1	1
556	1	1	1	1	1	1
561	1	1	1	1	1	1
566	1	1	1	1	1	1
571	1	1	1	1	1	1
576	1	1	1	1	1	1
581	1	1	1	1	1	1
586	1	1	1	1	1	1
591	1	1	1	1	1	1
596	1	1	1	1	1	1
601	1	1	1	1	1	1

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom . .	726
Number of restraints	240
Number of equations	486

L O A D C A S E N U M B E R 1

1000 Lb load at center:

Node Number	-----Forces-----			-----Moments-----		
	X	Y	Z	X	Y	Z
301	.0000E+00	1.0000E+03	.0000E+00	.0000E+00	.0000E+00	.0000

LOAD CASE PARAMETERS

Number of loaded nodes	1
Number of nodes with applied displacements .	0
Indicator for gravitational loads	0
Number of elements with distributed loads .	0
Number of elements with concentrated loads .	0
Number of elements with thermal loads . . .	0

LOAD CASE NUMBER 2

Uniform Transvers Pressure P = 1 psi

LOAD CASE PARAMETERS

Number of loaded nodes	0
Number of nodes with applied displacements .	0
Indicator for gravitational loads	0
Number of elements with distributed loads .	1
Number of elements with concentrated loads .	0
Number of elements with thermal loads . . .	0

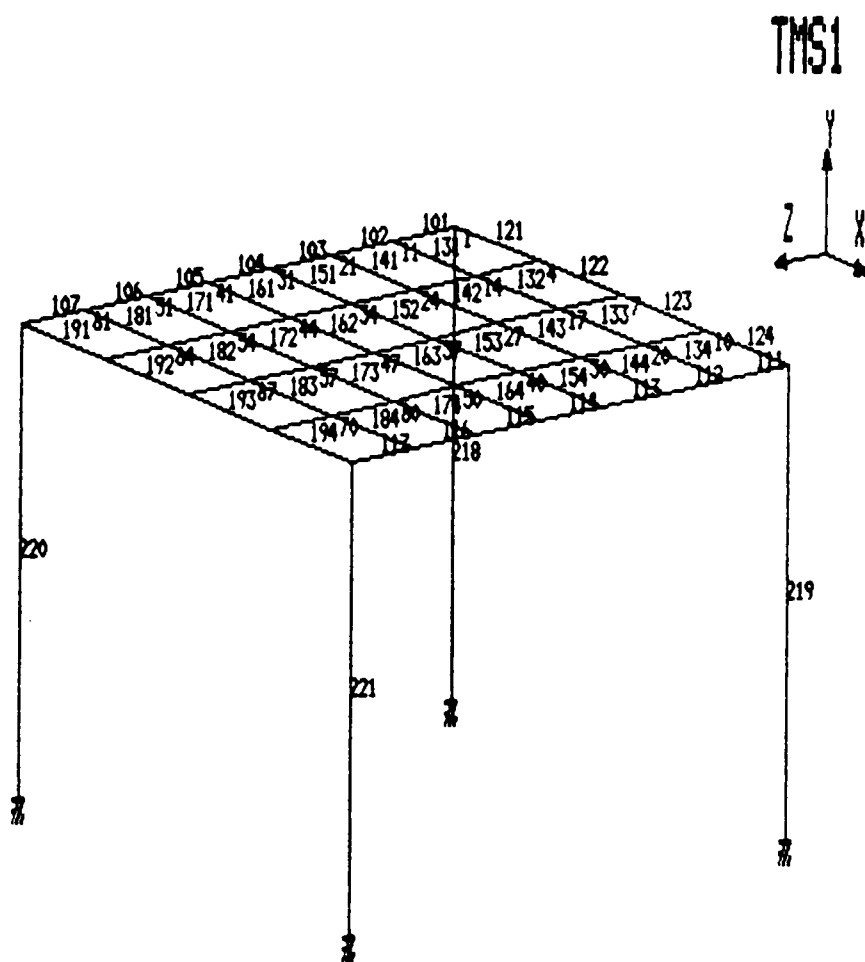


Figure B.7. Model TMS1.STR.

Model: TMS1.STR Title: Roof Structure for TMS Inc.

MODEL PARAMETERS

Number of nodes 44
 Number of restrained nodes . . . 4
 Number of elements 78
 Trusses 0
 Beams 50
 Plates 28
 Number of sections 3
 Number of materials 3

NODAL COORDINATES

Node Number	-----Coordinates----- X Y Z		
1	.0000E+00	.0000E+00	.0000E+00
3	1.3200E+02	.0000E+00	.0000E+00
5	2.6400E+02	.0000E+00	.0000E+00
7	3.9600E+02	.0000E+00	.0000E+00
9	5.2800E+02	.0000E+00	.0000E+00
11	.0000E+00	.0000E+00	7.2000E+01
13	1.3200E+02	.0000E+00	7.2000E+01
15	2.6400E+02	.0000E+00	7.2000E+01
17	3.9600E+02	.0000E+00	7.2000E+01
19	5.2800E+02	.0000E+00	7.2000E+01
21	.0000E+00	.0000E+00	1.4400E+02
23	1.3200E+02	.0000E+00	1.4400E+02
25	2.6400E+02	.0000E+00	1.4400E+02
27	3.9600E+02	.0000E+00	1.4400E+02
29	5.2800E+02	.0000E+00	1.4400E+02
31	.0000E+00	.0000E+00	2.1600E+02
33	1.3200E+02	.0000E+00	2.1600E+02
35	2.6400E+02	.0000E+00	2.1600E+02
37	3.9600E+02	.0000E+00	2.1600E+02
39	5.2800E+02	.0000E+00	2.1600E+02
41	.0000E+00	.0000E+00	2.8800E+02
43	1.3200E+02	.0000E+00	2.8800E+02
45	2.6400E+02	.0000E+00	2.8800E+02
47	3.9600E+02	.0000E+00	2.8800E+02
49	5.2800E+02	.0000E+00	2.8800E+02
51	.0000E+00	.0000E+00	3.6000E+02
53	1.3200E+02	.0000E+00	3.6000E+02
55	2.6400E+02	.0000E+00	3.6000E+02
57	3.9600E+02	.0000E+00	3.6000E+02
59	5.2800E+02	.0000E+00	3.6000E+02
61	.0000E+00	.0000E+00	4.3200E+02
63	1.3200E+02	.0000E+00	4.3200E+02
65	2.6400E+02	.0000E+00	4.3200E+02

67	3.9600E+02	.0000E+00	4.3200E+02
69	5.2800E+02	.0000E+00	4.3200E+02
71	.0000E+00	.0000E+00	5.0400E+02
73	1.3200E+02	.0000E+00	5.0400E+02
75	2.6400E+02	.0000E+00	5.0400E+02
77	3.9600E+02	.0000E+00	5.0400E+02
79	5.2800E+02	.0000E+00	5.0400E+02
101	.0000E+00	-2.6400E+02	.0000E+00
102	5.2800E+02	-2.6400E+02	.0000E+00
171	.0000E+00	-2.6400E+02	5.0400E+02
172	5.2800E+02	-2.6400E+02	5.0400E+02

B E A M E L E M E N T I N F O R M A T I O N

Elem. Number	--Nodes-- 1 2	Section Number	Material Number	Gamma angle (degrees)	Release Flag	Offset Flag
101	1 11	1	3	.0000E+00	0	0
102	11 21	1	3	.0000E+00	0	0
103	21 31	1	3	.0000E+00	0	0
104	31 41	1	3	.0000E+00	0	0
105	41 51	1	3	.0000E+00	0	0
106	51 61	1	3	.0000E+00	0	0
107	61 71	1	3	.0000E+00	0	0
111	9 19	1	3	.0000E+00	0	0
112	19 29	1	3	.0000E+00	0	0
113	29 39	1	3	.0000E+00	0	0
114	39 49	1	3	.0000E+00	0	0
115	49 59	1	3	.0000E+00	0	0
116	59 69	1	3	.0000E+00	0	0
117	69 79	1	3	.0000E+00	0	0
121	1 3	2	2	.0000E+00	0	0
122	3 5	2	2	.0000E+00	0	0
123	5 7	2	2	.0000E+00	0	0
124	7 9	2	2	.0000E+00	0	0
131	11 13	2	2	.0000E+00	0	0
132	13 15	2	2	.0000E+00	0	0
133	15 17	2	2	.0000E+00	0	0
134	17 19	2	2	.0000E+00	0	0
141	21 23	2	2	.0000E+00	0	0
142	23 25	2	2	.0000E+00	0	0
143	25 27	2	2	.0000E+00	0	0
144	27 29	2	2	.0000E+00	0	0
151	31 33	2	2	.0000E+00	0	0
152	33 35	2	2	.0000E+00	0	0
153	35 37	2	2	.0000E+00	0	0
154	37 39	2	2	.0000E+00	0	0
161	41 43	2	2	.0000E+00	0	0
162	43 45	2	2	.0000E+00	0	0
163	45 47	2	2	.0000E+00	0	0
164	47 49	2	2	.0000E+00	0	0

171	51	53	2	2	.0000E+00	0	0
172	53	55	2	2	.0000E+00	0	0
173	55	57	2	2	.0000E+00	0	0
174	57	59	2	2	.0000E+00	0	0
181	61	63	2	2	.0000E+00	0	0
182	63	65	2	2	.0000E+00	0	0
183	65	67	2	2	.0000E+00	0	0
184	67	69	2	2	.0000E+00	0	0
191	71	73	2	2	.0000E+00	0	0
192	73	75	2	2	.0000E+00	0	0
193	75	77	2	2	.0000E+00	0	0
194	77	79	2	2	.0000E+00	0	0
218	1	101	3	3	.0000E+00	0	0
219	9	102	3	3	.0000E+00	0	0
220	71	171	3	3	.0000E+00	0	0
221	79	172	3	3	.0000E+00	0	0

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	11	13	3	6.2500E-02	1	0	0
4	3	13	15	5	6.2500E-02	1	0	0
7	5	15	17	7	6.2500E-02	1	0	0
10	7	17	19	9	6.2500E-02	1	0	0
11	11	21	23	13	6.2500E-02	1	0	0
14	13	23	25	15	6.2500E-02	1	0	0
17	15	25	27	17	6.2500E-02	1	0	0
20	17	27	29	19	6.2500E-02	1	0	0
21	21	31	33	23	6.2500E-02	1	0	0
24	23	33	35	25	6.2500E-02	1	0	0
27	25	35	37	27	6.2500E-02	1	0	0
30	27	37	39	29	6.2500E-02	1	0	0
31	31	41	43	33	6.2500E-02	1	0	0
34	33	43	45	35	6.2500E-02	1	0	0
37	35	45	47	37	6.2500E-02	1	0	0
40	37	47	49	39	6.2500E-02	1	0	0
41	41	51	53	43	6.2500E-02	1	0	0
44	43	53	55	45	6.2500E-02	1	0	0
47	45	55	57	47	6.2500E-02	1	0	0
50	47	57	59	49	6.2500E-02	1	0	0
51	51	61	63	53	6.2500E-02	1	0	0
54	53	63	65	55	6.2500E-02	1	0	0
57	55	65	67	57	6.2500E-02	1	0	0
60	57	67	69	59	6.2500E-02	1	0	0
61	61	71	73	63	6.2500E-02	1	0	0
64	63	73	75	65	6.2500E-02	1	0	0
67	65	75	77	67	6.2500E-02	1	0	0
70	67	77	79	69	6.2500E-02	1	0	0

SECTION PROPERTIES

Number	Area	Jxx	Iyy	Izz	fy	fz
1	2.0100E+01	1.8700E+00	7.0400E+01	1.8300E+03	.0000E+00	.0000
2	2.4700E+01	3.7000E+00	9.4400E+01	2.3700E+03	.0000E+00	.0000
3	9.1300E+00	5.4000E-01	3.7100E+01	1.1000E+02	.0000E+00	.0000

MATERIAL PROPERTIES

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	2.9600E+07	2.8000E-01	3.5000E+04	2.8400E-01	6.5000E-06
2	2.9600E+07	2.8000E-01	3.6000E+04	2.8400E-01	6.5000E-06
3	2.9600E+07	2.8000E-01	3.6000E+04	2.8400E-01	6.5000E-06

NODAL RESTRAINTS

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
101	1	1	1	1	1	1
102	1	1	1	1	1	1
171	1	1	1	1	1	1
172	1	1	1	1	1	1

SOLUTION PARAMETERS

Number of degrees of freedom 264
 Number of restraints 24
 Number of equations 240

LOAD CASE NUMBER 1

Pressure on Roof Elements 1-70=45 lb/SqFt

LOAD CASE PARAMETERS

Number of loaded nodes 0
 Number of nodes with applied displacements 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads 1
 Number of elements with concentrated loads 0
 Number of elements with thermal loads 0

LOAD CASE NUMBER 2

Deadweight Load:

X g-factor Y g-factor Z g-factor

.0000E+00 -1.0000E+00 .0000E+00

LOAD CASE PARAMETERS

Number of loaded nodes	0
Number of nodes with applied displacements .	0
Indicator for gravitational loads	1
Number of elements with distributed loads .	0
Number of elements with concentrated loads .	0
Number of elements with thermal loads . . .	0

Model: BRIDGE1.STR Title: Truss Bridge

MODEL PARAMETERS

Number of nodes 32
 Number of restrained nodes . . . 4
 Number of elements 69
 Trusses 2
 Beams 60
 Plates 7
 Number of sections 1
 Number of materials 2

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	.0000E+00
11	3.0000E+02	.0000E+00	.0000E+00
21	6.0000E+02	.0000E+00	.0000E+00
31	9.0000E+02	.0000E+00	.0000E+00
41	1.2000E+03	.0000E+00	.0000E+00
51	1.5000E+03	.0000E+00	.0000E+00
61	1.8000E+03	.0000E+00	.0000E+00
71	2.1000E+03	.0000E+00	.0000E+00
101	.0000E+00	3.4500E+02	.0000E+00
111	3.0000E+02	4.7000E+02	.0000E+00
121	6.0000E+02	5.9500E+02	.0000E+00
131	9.0000E+02	7.2000E+02	.0000E+00
141	1.2000E+03	7.2000E+02	.0000E+00
151	1.5000E+03	5.9500E+02	.0000E+00
161	1.8000E+03	4.7000E+02	.0000E+00
171	2.1000E+03	3.4500E+02	.0000E+00
201	.0000E+00	.0000E+00	-4.8000E+02
211	3.0000E+02	.0000E+00	-4.8000E+02
221	6.0000E+02	.0000E+00	-4.8000E+02
231	9.0000E+02	.0000E+00	-4.8000E+02
241	1.2000E+03	.0000E+00	-4.8000E+02
251	1.5000E+03	.0000E+00	-4.8000E+02
261	1.8000E+03	.0000E+00	-4.8000E+02
271	2.1000E+03	.0000E+00	-4.8000E+02
301	.0000E+00	3.4500E+02	-4.8000E+02
311	3.0000E+02	4.7000E+02	-4.8000E+02
321	6.0000E+02	5.9500E+02	-4.8000E+02
331	9.0000E+02	7.2000E+02	-4.8000E+02
341	1.2000E+03	7.2000E+02	-4.8000E+02
351	1.5000E+03	5.9500E+02	-4.8000E+02
361	1.8000E+03	4.7000E+02	-4.8000E+02
371	2.1000E+03	3.4500E+02	-4.8000E+02

T R U S S E L E M E N T I N F O R M A T I O N

Elem. Number	--Nodes-- 1 2	Section Number	Material Number	Gamma angle (degrees)
3	121 321	1	1	.0000E+00
23	151 351	1	1	.0000E+00

B E A M E L E M E N T I N F O R M A T I O N

Elem. Number	--Nodes-- 1 2	Section Number	Material Number	Gamma angle (degrees)	Release Flag	Offset Flag
1	1 11	1	1	.0000E+00	0	0
11	11 21	1	1	.0000E+00	0	0
21	21 31	1	1	.0000E+00	0	0
31	31 41	1	1	.0000E+00	0	0
41	41 51	1	1	.0000E+00	0	0
51	51 61	1	1	.0000E+00	0	0
61	61 71	1	1	.0000E+00	0	0
101	101 111	1	1	.0000E+00	0	0
111	111 121	1	1	.0000E+00	0	0
121	121 131	1	1	.0000E+00	0	0
131	131 141	1	1	.0000E+00	0	0
141	141 151	1	1	.0000E+00	0	0
151	151 161	1	1	.0000E+00	0	0
161	161 171	1	1	.0000E+00	0	0
201	101 1	1	1	.0000E+00	0	0
211	111 11	1	1	.0000E+00	0	0
221	121 21	1	1	.0000E+00	0	0
231	131 31	1	1	.0000E+00	0	0
241	141 41	1	1	.0000E+00	0	0
251	151 51	1	1	.0000E+00	0	0
261	161 61	1	1	.0000E+00	0	0
271	171 71	1	1	.0000E+00	0	0
301	101 11	1	1	.0000E+00	0	0
311	111 21	1	1	.0000E+00	0	0
321	121 31	1	1	.0000E+00	0	0
331	131 41	1	1	.0000E+00	0	0
341	141 31	1	1	.0000E+00	0	0
351	151 41	1	1	.0000E+00	0	0
361	161 51	1	1	.0000E+00	0	0
371	171 61	1	1	.0000E+00	0	0
501	201 301	1	1	.0000E+00	0	0
511	211 311	1	1	.0000E+00	0	0
521	221 321	1	1	.0000E+00	0	0
531	231 331	1	1	.0000E+00	0	0
541	241 341	1	1	.0000E+00	0	0
551	251 351	1	1	.0000E+00	0	0
561	261 361	1	1	.0000E+00	0	0

571	271	371	1	1	.0000E+00	0	0
601	301	311	1	1	.0000E+00	0	0
611	311	321	1	1	.0000E+00	0	0
621	321	331	1	1	.0000E+00	0	0
631	331	341	1	1	.0000E+00	0	0
641	341	351	1	1	.0000E+00	0	0
651	351	361	1	1	.0000E+00	0	0
661	361	371	1	1	.0000E+00	0	0
701	301	211	1	1	.0000E+00	0	0
711	311	221	1	1	.0000E+00	0	0
721	321	231	1	1	.0000E+00	0	0
731	331	241	1	1	.0000E+00	0	0
741	341	231	1	1	.0000E+00	0	0
751	351	241	1	1	.0000E+00	0	0
761	361	251	1	1	.0000E+00	0	0
771	371	261	1	1	.0000E+00	0	0
801	201	211	1	1	.0000E+00	0	0
811	211	221	1	1	.0000E+00	0	0
821	221	231	1	1	.0000E+00	0	0
831	231	241	1	1	.0000E+00	0	0
841	241	251	1	1	.0000E+00	0	0
851	251	261	1	1	.0000E+00	0	0
861	261	271	1	1	.0000E+00	0	0

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
401	1	11	211	201	1.0000E+00	2	0	0
411	11	21	221	211	1.0000E+00	2	0	0
421	21	31	231	221	1.0000E+00	2	0	0
431	31	41	241	231	1.0000E+00	2	0	0
441	41	51	251	241	1.0000E+00	2	0	0
451	51	61	261	251	1.0000E+00	2	0	0
461	61	71	271	261	1.0000E+00	2	0	0

S E C T I O N P R O P E R T I E S

Number	Area	Jxx	Iyy	Izz	fy	f
1	1.7600E+01	2.4800E+00	1.1600E+02	3.4100E+02	.0000E+00	.0000

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	2.9600E+07	2.8000E-01	3.6000E+04	2.8400E-01	6.5000E-06

2 2.9600E+07 2.8000E-01 3.5000E+04 2.8400E-01 6.5000E-06

N O D A L R E S T R A I N T S

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
1	1	1	1	0	1	0
71	0	1	1	0	0	0
201	1	1	1	0	1	0
271	0	1	1	0	0	0

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom	192
Number of restraints	12
Number of equations	180

L O A D C A S E N U M B E R 1

Gravity Load:

X g-factor	Y g-factor	Z g-factor
.0000E+00	-1.0000E+00	.0000E+00

L O A D C A S E P A R A M E T E R S

Number of loaded nodes	0
Number of nodes with applied displacements	0
Indicator for gravitational loads	1
Number of elements with distributed loads	0
Number of elements with concentrated loads	0
Number of elements with thermal loads	0

L O A D C A S E N U M B E R 2

Vehicle Load:

Elem. Number	--Location--		-----Forces-----			-----Moments-----		
	x	y	X	Y	Z	X	Y	Z
411	.00	.25	.000E+00	.000E+00	-1.000E+03	.000E+00	.000E+00	.000
421	.00	.25	.000E+00	.000E+00	-1.000E+03	.000E+00	.000E+00	.000
411	.00	.75	.000E+00	.000E+00	-1.000E+03	.000E+00	.000E+00	.000

421 .00 .75 .000E+00 .000E+00-1.000E+03 .000E+00 .000E+00 .000

L O A D C A S E P A R A M E T E R S

Number of loaded nodes 0
Number of nodes with applied displacements . 0
Indicator for gravitational loads 0
Number of elements with distributed loads . 0
Number of elements with concentrated loads . 2
Number of elements with thermal loads . . . 0

Gail Deck Tiedown

TESTRUN

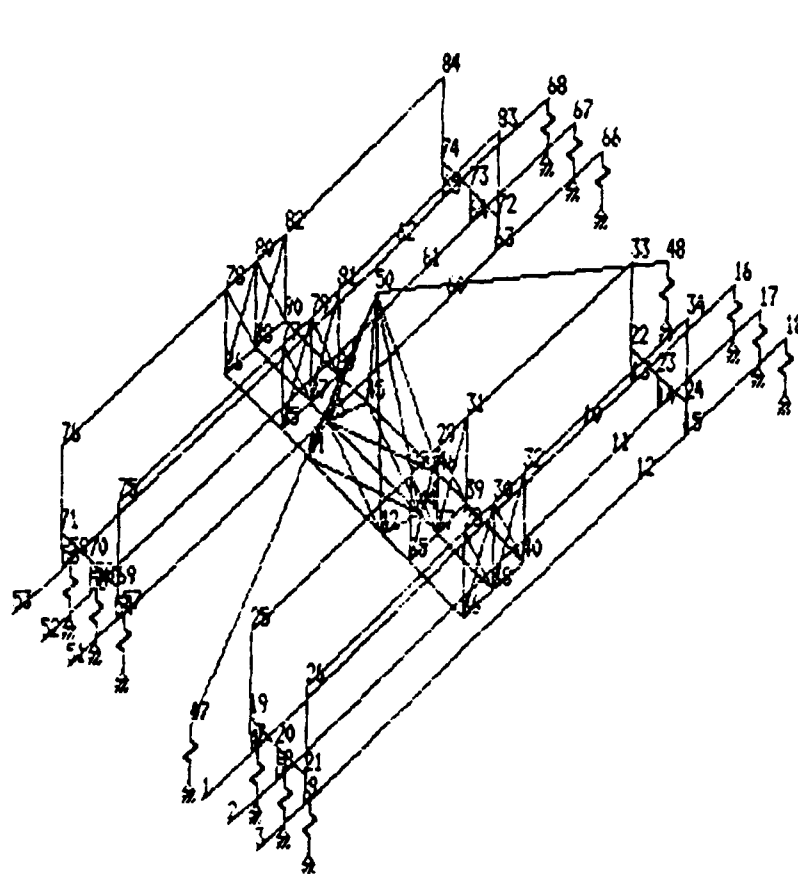


Figure B.9. Model TESTRUN.STR.

Model: TESTRUN.STR Title: Gail Deck Tiedown

MODEL PARAMETERS

Number of nodes 89
 Number of restrained nodes . . 14
 Number of elements 141
 Trusses 0
 Beams 141
 Plates 0
 Number of sections 6
 Number of materials 1

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	4.2996E+01	2.1000E+01	1.5000E+02
2	6.0000E+01	2.1000E+01	1.5000E+02
3	7.7004E+01	2.1000E+01	1.5000E+02
4	4.2996E+01	2.1000E+01	1.2000E+02
5	6.0000E+01	2.1000E+01	1.2000E+02
6	7.7004E+01	2.1000E+01	1.2000E+02
7	4.2996E+01	2.1000E+01	1.1520E+02
8	6.0000E+01	2.1000E+01	1.1520E+02
9	7.7004E+01	2.1000E+01	1.1520E+02
10	4.2996E+01	2.1000E+01	-9.0000E+01
11	6.0000E+01	2.1000E+01	-9.0000E+01
12	7.7004E+01	2.1000E+01	-9.0000E+01
13	4.2996E+01	2.1000E+01	-1.2000E+02
14	6.0000E+01	2.1000E+01	-1.2000E+02
15	7.7004E+01	2.1000E+01	-1.2000E+02
16	4.2996E+01	2.1000E+01	-1.8480E+02
17	6.0000E+01	2.1000E+01	-1.8480E+02
18	7.7004E+01	2.1000E+01	-1.8480E+02
19	4.2000E+01	3.2040E+01	1.2000E+02
20	6.0000E+01	3.2040E+01	1.2000E+02
21	7.8000E+01	3.2040E+01	1.2000E+02
22	4.2000E+01	3.2040E+01	-1.2000E+02
23	6.0000E+01	3.2040E+01	-1.2000E+02
24	7.8000E+01	3.2040E+01	-1.2000E+02
25	4.2000E+01	6.0000E+01	1.2000E+02
26	7.8000E+01	6.0000E+01	1.2000E+02
27	4.2000E+01	6.0000E+01	1.8372E+01
28	7.8000E+01	6.0000E+01	1.8372E+01
29	4.2000E+01	6.0000E+01	.0000E+00
30	7.8000E+01	6.0000E+01	.0000E+00
31	4.2000E+01	6.0000E+01	-1.8372E+01
32	7.8000E+01	6.0000E+01	-1.8372E+01
33	4.2000E+01	6.0000E+01	-1.2000E+02

34	7.8000E+01	6.0000E+01	-1.2000E+02
35	4.2000E+01	3.2040E+01	1.8372E+01
36	7.8000E+01	3.2040E+01	1.8372E+01
37	4.2000E+01	3.2040E+01	.0000E+00
38	7.8000E+01	3.2040E+01	.0000E+00
39	4.2000E+01	3.2040E+01	-1.8372E+01
40	7.8000E+01	3.2040E+01	-1.8372E+01
41	-2.3760E+01	3.2040E+01	1.8372E+01
42	2.3760E+01	3.2040E+01	1.8372E+01
43	-3.0000E+01	3.2040E+01	.0000E+00
44	3.0000E+01	3.2040E+01	.0000E+00
45	-2.3760E+01	3.2040E+01	-1.8372E+01
46	2.3760E+01	3.2040E+01	-1.8372E+01
47	.0000E+00	6.0000E+00	1.1520E+02
48	.0000E+00	6.0000E+00	-1.8480E+02
50	.0000E+00	8.9040E+01	.0000E+00
51	-4.2996E+01	2.1000E+01	1.5000E+02
52	-6.0000E+01	2.1000E+01	1.5000E+02
53	-7.7004E+01	2.1000E+01	1.5000E+02
54	-4.2996E+01	2.1000E+01	1.2000E+02
55	-6.0000E+01	2.1000E+01	1.2000E+02
56	-7.7004E+01	2.1000E+01	1.2000E+02
57	-4.2996E+01	2.1000E+01	1.1520E+02
58	-6.0000E+01	2.1000E+01	1.1520E+02
59	-7.7004E+01	2.1000E+01	1.1520E+02
60	-4.2996E+01	2.1000E+01	-9.0000E+01
61	-6.0000E+01	2.1000E+01	-9.0000E+01
62	-7.7004E+01	2.1000E+01	-9.0000E+01
63	-4.2996E+01	2.1000E+01	-1.2000E+02
64	-6.0000E+01	2.1000E+01	-1.2000E+02
65	-7.7004E+01	2.1000E+01	-1.2000E+02
66	-4.2996E+01	2.1000E+01	-1.8480E+02
67	-6.0000E+01	2.1000E+01	-1.8480E+02
68	-7.7004E+01	2.1000E+01	-1.8480E+02
69	-4.2000E+01	3.2040E+01	1.2000E+02
70	-6.0000E+01	3.2040E+01	1.2000E+02
71	-7.8000E+01	3.2040E+01	1.2000E+02
72	-4.2000E+01	3.2040E+01	-1.2000E+02
73	-6.0000E+01	3.2040E+01	-1.2000E+02
74	-7.8000E+01	3.2040E+01	-1.2000E+02
75	-4.2000E+01	6.0000E+01	1.2000E+02
76	-7.8000E+01	6.0000E+01	1.2000E+02
77	-4.2000E+01	6.0000E+01	1.8372E+01
78	-7.8000E+01	6.0000E+01	1.8372E+01
79	-4.2000E+01	6.0000E+01	.0000E+00
80	-7.8000E+01	6.0000E+01	.0000E+00
81	-4.2000E+01	6.0000E+01	-1.8372E+01
82	-7.8000E+01	6.0000E+01	-1.8372E+01
83	-4.2000E+01	6.0000E+01	-1.2000E+02
84	-7.8000E+01	6.0000E+01	-1.2000E+02
85	-4.2000E+01	3.2040E+01	1.8372E+01
86	-7.8000E+01	3.2040E+01	1.8372E+01

87	-4.2000E+01	3.2040E+01	.0000E+00
88	-7.8000E+01	3.2040E+01	.0000E+00
89	-4.2000E+01	3.2040E+01	-1.8372E+01
90	-7.8000E+01	3.2040E+01	-1.8372E+01

B E A M E L E M E N T I N F O R M A T I O N

Elem. Number	--Nodes-- 1 2	Section Number	Material Number	Gamma angle (degrees)	Release Flag	Offset Flag
1	36 35	3	1	.0000E+00	0	0
2	38 37	3	1	.0000E+00	0	0
3	40 39	3	1	.0000E+00	0	0
4	35 42	3	1	.0000E+00	0	0
5	37 44	3	1	.0000E+00	0	0
6	39 46	3	1	.0000E+00	0	0
7	42 41	3	1	.0000E+00	0	0
8	44 43	3	1	.0000E+00	0	0
9	46 45	3	1	.0000E+00	0	0
10	41 85	3	1	.0000E+00	0	0
11	43 87	3	1	.0000E+00	0	0
12	45 89	3	1	.0000E+00	0	0
13	85 86	3	1	.0000E+00	0	0
14	87 88	3	1	.0000E+00	0	0
15	89 90	3	1	.0000E+00	0	0
16	1 4	1	1	.0000E+00	0	0
17	2 5	1	1	.0000E+00	0	0
18	3 6	1	1	.0000E+00	0	0
19	4 7	1	1	.0000E+00	0	0
20	5 8	1	1	.0000E+00	0	0
21	6 9	1	1	.0000E+00	0	0
22	7 10	1	1	.0000E+00	0	0
23	8 11	1	1	.0000E+00	0	0
24	9 12	1	1	.0000E+00	0	0
25	10 13	1	1	.0000E+00	0	0
26	11 14	1	1	.0000E+00	0	0
27	12 15	1	1	.0000E+00	0	0
28	13 16	1	1	.0000E+00	0	0
29	14 17	1	1	.0000E+00	0	0
30	15 18	1	1	.0000E+00	0	0
31	51 54	1	1	.0000E+00	0	0
32	52 55	1	1	.0000E+00	0	0
33	53 56	1	1	.0000E+00	0	0
34	54 57	1	1	.0000E+00	0	0
35	55 58	1	1	.0000E+00	0	0
36	56 59	1	1	.0000E+00	0	0
37	57 60	1	1	.0000E+00	0	0
38	58 61	1	1	.0000E+00	0	0
39	59 62	1	1	.0000E+00	0	0
40	60 63	1	1	.0000E+00	0	0
41	61 64	1	1	.0000E+00	0	0

42	62	65	1	1	.0000E+00	0	0
43	63	66	1	1	.0000E+00	0	0
44	64	67	1	1	.0000E+00	0	0
45	65	68	1	1	.0000E+00	0	0
46	25	27	2	1	.0000E+00	0	0
47	26	28	2	1	.0000E+00	0	0
48	27	29	2	1	.0000E+00	0	0
49	28	30	2	1	.0000E+00	0	0
50	29	31	2	1	.0000E+00	0	0
51	30	32	2	1	.0000E+00	0	0
52	31	33	2	1	.0000E+00	0	0
53	32	34	2	1	.0000E+00	0	0
54	75	77	2	1	.0000E+00	0	0
55	76	78	2	1	.0000E+00	0	0
56	77	79	2	1	.0000E+00	0	0
57	78	80	2	1	.0000E+00	0	0
58	79	81	2	1	.0000E+00	0	0
59	80	82	2	1	.0000E+00	0	0
60	81	83	2	1	.0000E+00	0	0
61	82	84	2	1	.0000E+00	0	0
62	19	20	3	1	.0000E+00	0	0
63	20	21	3	1	.0000E+00	0	0
64	22	23	3	1	.0000E+00	0	0
65	23	24	3	1	.0000E+00	0	0
66	69	70	3	1	.0000E+00	0	0
67	70	71	3	1	.0000E+00	0	0
68	72	73	3	1	.0000E+00	0	0
69	73	74	3	1	.0000E+00	0	0
70	4	19	4	1	.0000E+00	0	0
71	5	20	4	1	.0000E+00	0	0
72	6	21	4	1	.0000E+00	0	0
73	13	22	4	1	.0000E+00	0	0
74	14	23	4	1	.0000E+00	0	0
75	15	24	4	1	.0000E+00	0	0
76	54	69	4	1	.0000E+00	0	0
77	55	70	4	1	.0000E+00	0	0
78	56	71	4	1	.0000E+00	0	0
79	63	72	4	1	.0000E+00	0	0
80	64	73	4	1	.0000E+00	0	0
81	65	74	4	1	.0000E+00	0	0
82	36	28	4	1	.0000E+00	0	0
83	38	30	4	1	.0000E+00	0	0
84	40	32	4	1	.0000E+00	0	0
85	35	27	4	1	.0000E+00	0	0
86	37	29	4	1	.0000E+00	0	0
87	39	31	4	1	.0000E+00	0	0
88	85	77	4	1	.0000E+00	0	0
89	87	79	4	1	.0000E+00	0	0
90	89	81	4	1	.0000E+00	0	0
91	86	78	4	1	.0000E+00	0	0
92	88	80	4	1	.0000E+00	0	0
93	90	82	4	1	.0000E+00	0	0

94	19	25	4	1	.0000E+00	0	0
95	21	26	4	1	.0000E+00	0	0
96	22	33	4	1	.0000E+00	0	0
97	24	34	4	1	.0000E+00	0	0
98	69	75	4	1	.0000E+00	0	0
99	71	76	4	1	.0000E+00	0	0
100	72	83	4	1	.0000E+00	0	0
101	74	84	4	1	.0000E+00	0	0
102	41	50	4	1	.0000E+00	0	0
103	42	50	4	1	.0000E+00	0	0
104	43	50	4	1	.0000E+00	0	0
105	44	50	4	1	.0000E+00	0	0
106	45	50	4	1	.0000E+00	0	0
107	46	50	4	1	.0000E+00	0	0
108	41	44	4	1	.0000E+00	0	0
109	42	43	4	1	.0000E+00	0	0
110	43	46	4	1	.0000E+00	0	0
111	44	45	4	1	.0000E+00	0	0
112	47	50	6	1	.0000E+00	0	0
113	48	50	6	1	.0000E+00	0	0
126	41	43	4	1	.0000E+00	0	0
127	43	45	4	1	.0000E+00	0	0
128	42	44	4	1	.0000E+00	0	0
129	44	46	4	1	.0000E+00	0	0
130	35	29	4	1	.0000E+00	0	0
131	37	27	4	1	.0000E+00	0	0
132	37	31	4	1	.0000E+00	0	0
133	39	29	4	1	1.6616E+07	0	0
134	36	30	4	1	1.6616E+07	0	0
135	38	28	4	1	1.6616E+07	0	0
136	38	32	4	1	.0000E+00	0	0
137	40	30	4	1	.0000E+00	0	0
138	85	79	4	1	.0000E+00	0	0
139	87	77	4	1	.0000E+00	0	0
140	87	81	4	1	.0000E+00	0	0
141	89	79	4	1	.0000E+00	0	0
142	86	80	4	1	.0000E+00	0	0
143	88	78	4	1	.0000E+00	0	0
144	88	82	4	1	.0000E+00	0	0
145	90	80	4	1	.0000E+00	0	0
146	35	37	4	1	.0000E+00	0	0
147	37	39	4	1	.0000E+00	0	0
148	36	38	4	1	.0000E+00	0	0
149	38	40	4	1	.0000E+00	0	0
150	85	87	4	1	.0000E+00	0	0
151	87	89	4	1	.0000E+00	0	0
152	86	88	4	1	.0000E+00	0	0
153	88	90	4	1	.0000E+00	0	0

SECTION PROPERTIES

Number	Area	Jxx	Iyy	Izz	fy	f
1	3.9700E+01	6.9900E+00	2.2500E+02	7.8000E+03	.0000E+00	.0000E
2	9.5600E+01	2.8600E+01	1.3970E+03	2.4530E+04	3.0000E-01	3.0000E
3	1.2900E+02	1.7800E+02	2.2670E+03	7.3060E+03	2.3300E-01	2.3300E
4	4.0000E+01	1.0000E+01	2.0000E+01	2.0000E+01	1.0000E+00	1.0000E
5	6.0000E+01	1.0000E+03	2.0000E+05	2.0000E+05	1.0000E+00	1.0000E
6	5.0280E+01	3.6700E+03	1.8350E+03	1.8350E+03	.0000E+00	.0000E

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	2.9600E+07	2.8000E-01	3.6000E+04	2.8400E-01	6.5000E-06

N O D A L R E S T R A I N T S

Node	--Translations--			---Rotations---		
Number	X	Y	Z	X	Y	Z

E L A S T I C S U P P O R T S

Node Number	-----Translational Stiffness-----			-----Rotational Stiffness-----		
	X	Y	Z	X	Y	Z
7	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
8	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
9	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
16	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
17	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
18	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
47	1.0000E+06	1.0000E+06	5.0000E+05	.0000E+00	.0000E+00	.0000
48	1.0000E+06	1.0000E+06	5.0000E+05	.0000E+00	.0000E+00	.0000
57	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
58	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
59	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
66	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
67	1.5000E+06	1.5000E+06	1.5000E+06	.0000E+00	.0000E+00	.0000
68	1.5000E+06	1.5000E+06	1.5000E+06	1.0000E+03	1.0000E+03	1.0000

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom . . .	534
Number of restraints	0
Number of equations	534

LOAD CASE NUMBER 1

Port Side Combined Max. Loads:

Node Number	Forces			Moments		
	X	Y	Z	X	Y	Z
50	3.9721E+04	-2.5710E+06	4.8330E+05	.0000E+00	.0000E+00	.0000

LOAD CASE PARAMETERS

Number of loaded nodes 1
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . 0
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . . 0

LOAD CASE NUMBER 2

Port Side Fx= 100 kips:

Node Number	Forces			Moments		
	X	Y	Z	X	Y	Z
50	1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000E+00	.0000

LOAD CASE PARAMETERS

Number of loaded nodes 1
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads 0
 Number of elements with distributed loads . 0
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . . 0

STEEL TANK ON PILES

TANK

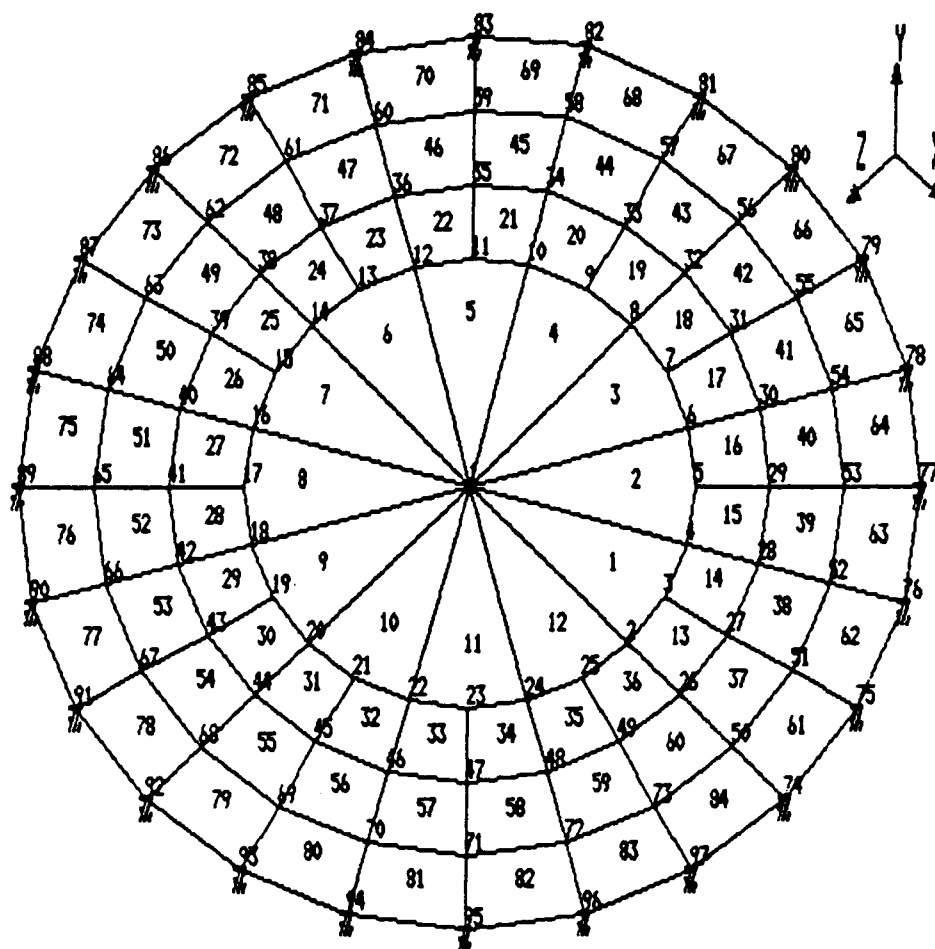


Figure B.10. Model TANK.STR.

Model: TANK.STR Title: STEEL TANK ON PILES

MODEL PARAMETERS

Number of nodes 97
 Number of restrained nodes . . 24
 Number of elements 84
 Trusses 0
 Beams 0
 Plates 84
 Number of sections 0
 Number of materials 1

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	5.9055E+01	.0000E+00
2	1.1811E+02	5.9055E+01	.0000E+00
3	1.1409E+02	5.9055E+01	-3.0551E+01
4	1.0228E+02	5.9055E+01	-5.9055E+01
5	8.3504E+01	5.9055E+01	-8.3504E+01
6	5.9055E+01	5.9055E+01	-1.0228E+02
7	3.0551E+01	5.9055E+01	-1.1409E+02
8	.0000E+00	5.9055E+01	-1.1811E+02
9	-3.0551E+01	5.9055E+01	-1.1409E+02
10	-6.2047E+01	5.9055E+01	-1.0228E+02
11	-8.3504E+01	5.9055E+01	-8.3504E+01
12	-1.0228E+02	5.9055E+01	-5.9055E+01
13	-1.1409E+02	5.9055E+01	-3.0551E+01
14	-1.1811E+02	5.9055E+01	.0000E+00
15	-1.1409E+02	5.9055E+01	3.0551E+01
16	-1.0228E+02	5.9055E+01	5.9055E+01
17	-8.3504E+01	5.9055E+01	8.3504E+01
18	-5.9055E+01	5.9055E+01	1.0228E+02
19	-3.0551E+01	5.9055E+01	1.1409E+02
20	.0000E+00	5.9055E+01	1.1811E+02
21	3.0551E+01	5.9055E+01	1.1409E+02
22	5.9055E+01	5.9055E+01	1.0228E+02
23	8.3504E+01	5.9055E+01	8.3504E+01
24	1.0228E+02	5.9055E+01	5.9055E+01
25	1.1409E+02	5.9055E+01	3.0551E+01
26	1.5748E+02	5.9055E+01	.0000E+00
27	1.5213E+02	5.9055E+01	-4.0748E+01
28	1.3640E+02	5.9055E+01	-7.8740E+01
29	1.1134E+02	5.9055E+01	-1.1134E+02
30	7.8740E+01	5.9055E+01	-1.3638E+02
31	4.0748E+01	5.9055E+01	-1.5213E+02
32	.0000E+00	5.9055E+01	-1.5748E+02
33	-4.0748E+01	5.9055E+01	-1.5213E+02

34	-8.2752E+01	5.9055E+01	--1.3638E+02
35	-1.1134E+02	5.9055E+01	-1.1134E+02
36	--1.3638E+02	5.9055E+01	-7.8740E+01
37	-1.5213E+02	5.9055E+01	-4.0748E+01
38	--1.5748E+02	5.9055E+01	.0000E+00
39	-1.5213E+02	5.9055E+01	4.0748E+01
40	--1.3638E+02	5.9055E+01	7.8740E+01
41	-1.1134E+02	5.9055E+01	1.1134E+02
42	-7.8740E+01	5.9055E+01	1.3638E+02
43	-4.0748E+01	5.9055E+01	1.5213E+02
44	.0000E+00	5.9055E+01	1.5748E+02
45	4.0748E+01	5.9055E+01	1.5213E+02
46	7.8740E+01	5.9055E+01	1.3638E+02
47	1.1134E+02	5.9055E+01	1.1134E+02
48	1.3638E+02	5.9055E+01	7.8740E+01
49	1.5213E+02	5.9055E+01	4.0748E+01
50	1.9685E+02	5.9055E+01	.0000E+00
51	1.9016E+02	5.9055E+01	-5.0945E+01
52	1.7052E+02	5.9055E+01	-9.8425E+01
53	1.3917E+02	5.9055E+01	-1.3917E+02
54	9.8425E+01	5.9055E+01	-1.7047E+02
55	5.0945E+01	5.9055E+01	-1.9016E+02
56	.0000E+00	5.9055E+01	-1.9685E+02
57	-5.0945E+01	5.9055E+01	-1.9016E+02
58	-1.0346E+02	5.9055E+01	--1.7047E+02
59	-1.3917E+02	5.9055E+01	-1.3917E+02
60	-1.7047E+02	5.9055E+01	-9.8425E+01
61	-1.9016E+02	5.9055E+01	-5.0945E+01
62	-1.9685E+02	5.9055E+01	.0000E+00
63	-1.9016E+02	5.9055E+01	5.0945E+01
64	--1.7047E+02	5.9055E+01	9.8425E+01
65	-1.3917E+02	5.9055E+01	1.3917E+02
66	-9.8425E+01	5.9055E+01	1.7047E+02
67	-5.0945E+01	5.9055E+01	1.9016E+02
68	.0000E+00	5.9055E+01	1.9685E+02
69	5.0945E+01	5.9055E+01	1.9016E+02
70	9.8425E+01	5.9055E+01	1.7047E+02
71	1.3917E+02	5.9055E+01	1.3917E+02
72	1.7047E+02	5.9055E+01	9.8425E+01
73	1.9016E+02	5.9055E+01	5.0945E+01
74	2.3622E+02	5.9055E+01	.0000E+00
75	2.2819E+02	5.9055E+01	-6.1142E+01
76	2.0464E+02	5.9055E+01	--1.1811E+02
77	1.6701E+02	5.9055E+01	-1.6701E+02
78	1.1811E+02	5.9055E+01	-2.0457E+02
79	6.1142E+01	5.9055E+01	-2.2819E+02
80	.0000E+00	5.9055E+01	-2.3622E+02
81	-6.1142E+01	5.9055E+01	-2.2819E+02
82	--1.2416E+02	5.9055E+01	-2.0457E+02
83	-1.6701E+02	5.9055E+01	-1.6701E+02
84	-2.0457E+02	5.9055E+01	--1.1811E+02
85	-2.2819E+02	5.9055E+01	-6.1142E+01

86	-2.3622E+02	5.9055E+01	.0000E+00
87	-2.2819E+02	5.9055E+01	6.1142E+01
88	-2.0457E+02	5.9055E+01	1.1811E+02
89	-1.6701E+02	5.9055E+01	1.6701E+02
90	-1.1811E+02	5.9055E+01	2.0457E+02
91	-6.1142E+01	5.9055E+01	2.2819E+02
92	.0000E+00	5.9055E+01	2.3622E+02
93	6.1142E+01	5.9055E+01	2.2819E+02
94	1.1811E+02	5.9055E+01	2.0457E+02
95	1.6701E+02	5.9055E+01	1.6701E+02
96	2.0457E+02	5.9055E+01	1.1811E+02
97	2.2819E+02	5.9055E+01	6.1142E+01

P L A T E E L E M E N T I N F O R M A T I O N

Elem. Number	-----Nodes-----				Plate Thickness	Material Number	Release Flag	Offset Flag
	1	2	3	4				
1	1	2	3	4	2.3622E-01	1	0	0
2	1	4	5	6	2.3622E-01	1	0	0
3	1	6	7	8	2.3622E-01	1	0	0
4	1	8	9	10	2.3622E-01	1	0	0
5	1	10	11	12	2.3622E-01	1	0	0
6	1	12	13	14	2.3622E-01	1	0	0
7	1	14	15	16	2.3622E-01	1	0	0
8	1	16	17	18	2.3622E-01	1	0	0
9	1	18	19	20	2.3622E-01	1	0	0
10	1	20	21	22	2.3622E-01	1	0	0
11	1	22	23	24	2.3622E-01	1	0	0
12	1	24	25	2	2.3622E-01	1	0	0
13	2	26	27	3	2.3622E-01	1	0	0
14	3	27	28	4	2.3622E-01	1	0	0
15	4	28	29	5	2.3622E-01	1	0	0
16	5	29	30	6	2.3622E-01	1	0	0
17	6	30	31	7	2.3622E-01	1	0	0
18	7	31	32	8	2.3622E-01	1	0	0
19	8	32	33	9	2.3622E-01	1	0	0
20	9	33	34	10	2.3622E-01	1	0	0
21	10	34	35	11	2.3622E-01	1	0	0
22	11	35	36	12	2.3622E-01	1	0	0
23	12	36	37	13	2.3622E-01	1	0	0
24	13	37	38	14	2.3622E-01	1	0	0
25	14	38	39	15	2.3622E-01	1	0	0
26	15	39	40	16	2.3622E-01	1	0	0
27	16	40	41	17	2.3622E-01	1	0	0
28	17	41	42	18	2.3622E-01	1	0	0
29	18	42	43	19	2.3622E-01	1	0	0
30	19	43	44	20	2.3622E-01	1	0	0
31	20	44	45	21	2.3622E-01	1	0	0
32	21	45	46	22	2.3622E-01	1	0	0
33	22	46	47	23	2.3622E-01	1	0	0

34	23	47	48	24	2.3622E-01	1	0	0
35	24	48	49	25	2.3622E-01	1	0	0
36	25	49	26	2	2.3622E-01	1	0	0
37	26	50	51	27	2.3622E-01	1	0	0
38	27	51	52	28	2.3622E-01	1	0	0
39	28	52	53	29	2.3622E-01	1	0	0
40	29	53	54	30	2.3622E-01	1	0	0
41	30	54	55	31	2.3622E-01	1	0	0
42	31	55	56	32	2.3622E-01	1	0	0
43	32	56	57	33	2.3622E-01	1	0	0
44	33	57	58	34	2.3622E-01	1	0	0
45	34	58	59	35	2.3622E-01	1	0	0
46	35	59	60	36	2.3622E-01	1	0	0
47	36	60	61	37	2.3622E-01	1	0	0
48	37	61	62	38	2.3622E-01	1	0	0
49	38	62	63	39	2.3622E-01	1	0	0
50	39	63	64	40	2.3622E-01	1	0	0
51	40	64	65	41	2.3622E-01	1	0	0
52	41	65	66	42	2.3622E-01	1	0	0
53	42	66	67	43	2.3622E-01	1	0	0
54	43	67	68	44	2.3622E-01	1	0	0
55	44	68	69	45	2.3622E-01	1	0	0
56	45	69	70	46	2.3622E-01	1	0	0
57	46	70	71	47	2.3622E-01	1	0	0
58	47	71	72	48	2.3622E-01	1	0	0
59	48	72	73	49	2.3622E-01	1	0	0
60	49	73	50	26	2.3622E-01	1	0	0
61	50	74	75	51	2.3622E-01	1	0	0
62	51	75	76	52	2.3622E-01	1	0	0
63	52	76	77	53	2.3622E-01	1	0	0
64	53	77	78	54	2.3622E-01	1	0	0
65	54	78	79	55	2.3622E-01	1	0	0
66	55	79	80	56	2.3622E-01	1	0	0
67	56	80	81	57	2.3622E-01	1	0	0
68	57	81	82	58	2.3622E-01	1	0	0
69	58	82	83	59	2.3622E-01	1	0	0
70	59	83	84	60	2.3622E-01	1	0	0
71	60	84	85	61	2.3622E-01	1	0	0
72	61	85	86	62	2.3622E-01	1	0	0
73	62	86	87	63	2.3622E-01	1	0	0
74	63	87	88	64	2.3622E-01	1	0	0
75	64	88	89	65	2.3622E-01	1	0	0
76	65	89	90	66	2.3622E-01	1	0	0
77	66	90	91	67	2.3622E-01	1	0	0
78	67	91	92	68	2.3622E-01	1	0	0
79	68	92	93	69	2.3622E-01	1	0	0
80	69	93	94	70	2.3622E-01	1	0	0
81	70	94	95	71	2.3622E-01	1	0	0
82	71	95	96	72	2.3622E-01	1	0	0
83	72	96	97	73	2.3622E-01	1	0	0
84	73	97	74	50	2.3622E-01	1	0	0

M A T E R I A L P R O P E R T I E S

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	2.9603E+07	2.8000E-01	4.0002E+04	2.8400E-01	6.5000E-06

N O D A L R E S T R A I N T S

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
74	1	1	1	1	1	1
75	1	1	1	1	1	1
76	1	1	1	1	1	1
77	1	1	1	1	1	1
78	1	1	1	1	1	1
79	1	1	1	1	1	1
80	1	1	1	1	1	1
81	1	1	1	1	1	1
82	1	1	1	1	1	1
83	1	1	1	1	1	1
84	1	1	1	1	1	1
85	1	1	1	1	1	1
86	1	1	1	1	1	1
87	1	1	1	1	1	1
88	1	1	1	1	1	1
89	1	1	1	1	1	1
90	1	1	1	1	1	1
91	1	1	1	1	1	1
92	1	1	1	1	1	1
93	1	1	1	1	1	1
94	1	1	1	1	1	1
95	1	1	1	1	1	1
96	1	1	1	1	1	1
97	1	1	1	1	1	1

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom	582
Number of restraints	144
Number of equations	438

L O A D C A S E N U M B E R 1

Uniform Transverse Pressure - P = 10psi

L O A D C A S E P A R A M E T E R S

Number of loaded nodes	0
Number of nodes with applied displacements .	0
Indicator for gravitational loads	0
Number of elements with distributed loads .	1
Number of elements with concentrated loads .	0
Number of elements with thermal loads . . .	0

L O A D C A S E N U M B E R 2

10 Lb. load in Y direction for nodes 26-49

L O A D C A S E P A R A M E T E R S

Number of loaded nodes	24
Number of nodes with applied displacements .	0
Indicator for gravitational loads	0
Number of elements with distributed loads .	0
Number of elements with concentrated loads .	0
Number of elements with thermal loads . . .	0

AMMONIA SPHERE STUDY

SPHERE1

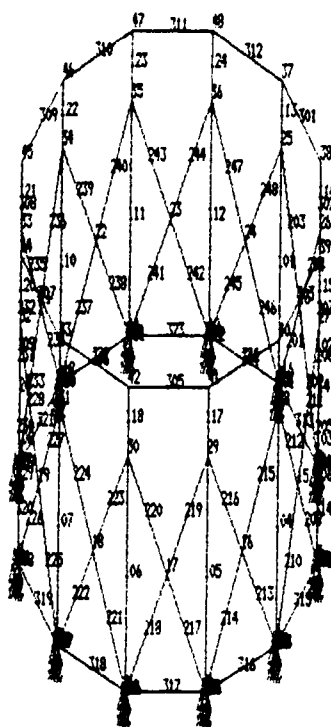


Figure B.11. Model SPHERE1.STR.

Model: SPHERE1.STR Title: AMMONIA SPHERE STUDY

MODEL PARAMETERS

Number of nodes 72
 Number of restrained nodes . . . 36
 Number of elements 120
 Trusses 0
 Beams 120
 Plates 0
 Number of sections 5
 Number of materials 3

NODAL COORDINATES

Node Number	-----Coordinates-----		
	X	Y	Z
1	.0000E+00	.0000E+00	-3.8400E+02
2	1.9200E+02	.0000E+00	-3.3252E+02
3	3.3252E+02	.0000E+00	-1.9200E+02
4	3.8400E+02	.0000E+00	.0000E+00
5	3.3252E+02	.0000E+00	1.9200E+02
6	1.9200E+02	.0000E+00	3.3252E+02
7	.0000E+00	.0000E+00	3.8400E+02
8	-1.9200E+02	.0000E+00	3.3252E+02
9	-3.3252E+02	.0000E+00	1.9200E+02
10	-3.8400E+02	.0000E+00	.0000E+00
11	-3.3252E+02	.0000E+00	-1.9200E+02
12	-1.9200E+02	.0000E+00	-3.3252E+02
13	9.6000E+01	1.7400E+02	-3.5826E+02
14	2.6226E+02	1.7400E+02	-2.6226E+02
15	3.5826E+02	1.7400E+02	-9.6000E+01
16	3.5826E+02	1.7400E+02	9.6000E+01
17	2.6226E+02	1.7400E+02	2.6226E+02
18	9.6000E+01	1.7400E+02	3.5826E+02
19	-9.6000E+01	1.7400E+02	3.5826E+02
20	-2.6226E+02	1.7400E+02	2.6226E+02
21	-3.5826E+02	1.7400E+02	9.6000E+01
22	-3.5826E+02	1.7400E+02	-9.6000E+01
23	-2.6226E+02	1.7400E+02	-2.6226E+02
24	-9.6000E+01	1.7400E+02	-3.5826E+02
25	.0000E+00	3.4800E+02	-3.8400E+02
26	1.9200E+02	3.4800E+02	-3.3252E+02
27	3.3252E+02	3.4800E+02	-1.9200E+02
28	3.8400E+02	3.4800E+02	.0000E+00
29	3.3252E+02	3.4800E+02	1.9200E+02
30	1.9200E+02	3.4800E+02	3.3252E+02
31	.0000E+00	3.4800E+02	3.8400E+02
32	-1.9200E+02	3.4800E+02	3.3252E+02
33	-3.3252E+02	3.4800E+02	1.9200E+02

34	-3.8400E+02	3.4800E+02	.0000E+00
35	-3.3252E+02	3.4800E+02	-1.9200E+02
36	-1.9200E+02	3.4800E+02	-3.3252E+02
37	.0000E+00	4.5000E+02	-3.8400E+02
38	1.9200E+02	4.5000E+02	-3.3252E+02
39	3.3252E+02	4.5000E+02	-1.9200E+02
40	3.8400E+02	4.5000E+02	.0000E+00
41	3.3252E+02	4.5000E+02	1.9200E+02
42	1.9200E+02	4.5000E+02	3.3252E+02
43	.0000E+00	4.5000E+02	3.8400E+02
44	-1.9200E+02	4.5000E+02	3.3252E+02
45	-3.3252E+02	4.5000E+02	1.9200E+02
46	-3.8400E+02	4.5000E+02	.0000E+00
47	-3.3252E+02	4.5000E+02	-1.9200E+02
48	-1.9200E+02	4.5000E+02	-3.3252E+02
49	1.1592E+01	.0000E+00	-3.8089E+02
50	2.0359E+02	.0000E+00	-3.2941E+02
51	3.4411E+02	.0000E+00	-1.8889E+02
52	3.9559E+02	.0000E+00	3.1080E+00
53	3.4411E+02	.0000E+00	1.9511E+02
54	2.0359E+02	.0000E+00	3.3563E+02
55	1.1592E+01	.0000E+00	3.8711E+02
56	-1.8041E+02	.0000E+00	3.3563E+02
57	-3.2093E+02	.0000E+00	1.9511E+02
58	-3.7241E+02	.0000E+00	3.1080E+00
59	-3.2093E+02	.0000E+00	-1.8889E+02
60	-1.8041E+02	.0000E+00	-3.2941E+02
61	-3.1080E+00	.0000E+00	-3.7241E+02
62	1.8889E+02	.0000E+00	-3.2093E+02
63	3.2941E+02	.0000E+00	-1.8041E+02
64	3.8089E+02	.0000E+00	1.1592E+01
65	3.2941E+02	.0000E+00	2.0359E+02
66	1.8889E+02	.0000E+00	3.4411E+02
67	-3.1080E+00	.0000E+00	3.9559E+02
68	-1.9511E+02	.0000E+00	3.4411E+02
69	-3.3563E+02	.0000E+00	2.0359E+02
70	-3.8711E+02	.0000E+00	1.1592E+01
71	-3.3563E+02	.0000E+00	-1.8041E+02
72	-1.9511E+02	.0000E+00	-3.2093E+02

B E A M E L E M E N T I N F O R M A T I O N

Elem. Number	--Nodes-- 1 2	Section Number	Material Number	Gamma angle (degrees)	Release Flag	Offset Flag
101	1 25	1	1	.0000E+00	1	0
102	2 26	1	1	.0000E+00	1	0
103	3 27	1	1	.0000E+00	1	0
104	4 28	1	1	.0000E+00	1	0
105	5 29	1	1	.0000E+00	1	0
106	6 30	1	1	.0000E+00	1	0

107	7	31	1	1	.0000E+00	1	0
108	8	32	1	1	.0000E+00	1	0
109	9	33	1	1	.0000E+00	1	0
110	10	34	1	1	.0000E+00	1	0
111	11	35	1	1	.0000E+00	1	0
112	12	36	1	1	.0000E+00	1	0
113	25	37	2	1	.0000E+00	0	0
114	26	38	2	1	.0000E+00	0	0
115	27	39	2	1	.0000E+00	0	0
116	28	40	2	1	.0000E+00	0	0
117	29	41	2	1	.0000E+00	0	0
118	30	42	2	1	.0000E+00	0	0
119	31	43	2	1	.0000E+00	0	0
120	32	44	2	1	.0000E+00	0	0
121	33	45	2	1	.0000E+00	0	0
122	34	46	2	1	.0000E+00	0	0
123	35	47	2	1	.0000E+00	0	0
124	36	48	2	1	.0000E+00	0	0
201	1	13	3	1	.0000E+00	1	0
202	2	13	3	1	.0000E+00	1	0
203	25	13	3	1	.0000E+00	1	0
204	26	13	3	1	.0000E+00	1	0
205	2	14	3	1	.0000E+00	1	0
206	3	14	3	1	.0000E+00	1	0
207	26	14	3	1	.0000E+00	1	0
208	27	14	3	1	.0000E+00	1	0
209	3	15	3	1	.0000E+00	1	0
210	4	15	3	1	.0000E+00	1	0
211	27	15	3	1	.0000E+00	1	0
212	28	15	3	1	.0000E+00	1	0
213	4	16	3	1	.0000E+00	1	0
214	5	16	3	1	.0000E+00	1	0
215	28	16	3	1	.0000E+00	1	0
216	29	16	3	1	.0000E+00	1	0
217	5	17	3	1	.0000E+00	1	0
218	6	17	3	1	.0000E+00	1	0
219	29	17	3	1	.0000E+00	1	0
220	30	17	3	1	.0000E+00	1	0
221	6	18	3	1	.0000E+00	1	0
222	7	18	3	1	.0000E+00	1	0
223	30	18	3	1	.0000E+00	1	0
224	31	18	3	1	.0000E+00	1	0
225	7	19	3	1	.0000E+00	1	0
226	8	19	3	1	.0000E+00	1	0
227	31	19	3	1	.0000E+00	1	0
228	32	19	3	1	.0000E+00	1	0
229	8	20	3	1	.0000E+00	1	0
230	9	20	3	1	.0000E+00	1	0
231	32	20	3	1	.0000E+00	1	0
232	33	20	3	1	.0000E+00	1	0
233	9	21	3	1	.0000E+00	1	0
234	10	21	3	1	.0000E+00	1	0

235	33	21	3	1	.0000E+00	1	0
236	34	21	3	1	.0000E+00	1	0
237	10	22	3	1	.0000E+00	1	0
238	11	22	3	1	.0000E+00	1	0
239	34	22	3	1	.0000E+00	1	0
240	35	22	3	1	.0000E+00	1	0
241	11	23	3	1	.0000E+00	1	0
242	12	23	3	1	.0000E+00	1	0
243	35	23	3	1	.0000E+00	1	0
244	36	23	3	1	.0000E+00	1	0
245	12	24	3	1	.0000E+00	1	0
246	1	24	3	1	.0000E+00	1	0
247	36	24	3	1	.0000E+00	1	0
248	25	24	3	1	.0000E+00	1	0
301	37	38	2	1	.0000E+00	0	0
302	38	39	2	1	.0000E+00	0	0
303	39	40	2	1	.0000E+00	0	0
304	40	41	2	1	.0000E+00	0	0
305	41	42	2	1	.0000E+00	0	0
306	42	43	2	1	.0000E+00	0	0
307	43	44	2	1	.0000E+00	0	0
308	44	45	2	1	.0000E+00	0	0
309	45	46	2	1	.0000E+00	0	0
310	46	47	2	1	.0000E+00	0	0
311	47	48	2	1	.0000E+00	0	0
312	48	37	2	1	.0000E+00	0	0
313	1	2	4	2	.0000E+00	0	0
314	2	3	4	2	.0000E+00	0	0
315	3	4	4	2	.0000E+00	0	0
316	4	5	4	2	.0000E+00	0	0
317	5	6	4	2	.0000E+00	0	0
318	6	7	4	2	.0000E+00	0	0
319	7	8	4	2	.0000E+00	0	0
320	8	9	4	2	.0000E+00	0	0
321	9	10	4	2	.0000E+00	0	0
322	10	11	4	2	.0000E+00	0	0
323	11	12	4	2	.0000E+00	0	0
324	12	1	4	2	.0000E+00	0	0
401	1	49	5	3	.0000E+00	1	0
402	2	50	5	3	.0000E+00	1	0
403	3	51	5	3	.0000E+00	1	0
404	4	52	5	3	.0000E+00	1	0
405	5	53	5	3	.0000E+00	1	0
406	6	54	5	3	.0000E+00	1	0
407	7	55	5	3	.0000E+00	1	0
408	8	56	5	3	.0000E+00	1	0
409	9	57	5	3	.0000E+00	1	0
410	10	58	5	3	.0000E+00	1	0
411	11	59	5	3	.0000E+00	1	0
412	12	60	5	3	.0000E+00	1	0
413	1	61	5	3	.0000E+00	1	0
414	2	62	5	3	.0000E+00	1	0

415	3	63	5	3	.0000E+00	1	0
416	4	64	5	3	.0000E+00	1	0
417	5	65	5	3	.0000E+00	1	0
418	6	66	5	3	.0000E+00	1	0
419	7	67	5	3	.0000E+00	1	0
420	8	68	5	3	.0000E+00	1	0
421	9	69	5	3	.0000E+00	1	0
422	10	70	5	3	.0000E+00	1	0
423	11	71	5	3	.0000E+00	1	0
424	12	72	5	3	.0000E+00	1	0

B E A M R E L E A S E S

Elem. Number	Node Number	Translational			Rotational		
		X	Y	Z	X	Y	Z
101	1	0	0	0	0	1	1
	25	0	0	0	0	0	0
102	2	0	0	0	0	1	1
	26	0	0	0	0	0	0
103	3	0	0	0	0	1	1
	27	0	0	0	0	0	0
104	4	0	0	0	0	1	1
	28	0	0	0	0	0	0
105	5	0	0	0	0	1	1
	29	0	0	0	0	0	0
106	6	0	0	0	0	1	1
	30	0	0	0	0	0	0
107	7	0	0	0	0	1	1
	31	0	0	0	0	0	0
108	8	0	0	0	0	1	1
	32	0	0	0	0	0	0
109	9	0	0	0	0	1	1
	33	0	0	0	0	0	0
110	10	0	0	0	0	1	1
	34	0	0	0	0	0	0
111	11	0	0	0	0	1	1
	35	0	0	0	0	0	0
112	12	0	0	0	0	1	1
	36	0	0	0	0	0	0
201	1	0	0	0	0	1	0
	13	0	0	0	0	0	0
202	2	0	0	0	0	1	0
	13	0	0	0	0	0	0
203	25	0	0	0	0	1	0
	13	0	0	0	0	0	0
204	26	0	0	0	0	1	0
	13	0	0	0	0	0	0
205	2	0	0	0	0	1	0
	14	0	0	0	0	0	0
206	3	0	0	0	0	1	0

	14	0	0	0	0	0	0
207	26	0	0	0	0	1	0
	14	0	0	0	0	0	0
208	27	0	0	0	0	1	0
	14	0	0	0	0	0	0
209	3	0	0	0	0	1	0
	15	0	0	0	0	0	0
210	4	0	0	0	0	1	0
	15	0	0	0	0	0	0
211	27	0	0	0	0	1	0
	15	0	0	0	0	0	0
212	28	0	0	0	0	1	0
	15	0	0	0	0	0	0
213	4	0	0	0	0	1	0
	16	0	0	0	0	0	0
214	5	0	0	0	0	1	0
	16	0	0	0	0	0	0
215	28	0	0	0	0	1	0
	16	0	0	0	0	0	0
216	29	0	0	0	0	1	0
	16	0	0	0	0	0	0
217	5	0	0	0	0	1	0
	17	0	0	0	0	0	0
218	6	0	0	0	0	1	0
	17	0	0	0	0	0	0
219	29	0	0	0	0	1	0
	17	0	0	0	0	0	0
220	30	0	0	0	0	1	0
	17	0	0	0	0	0	0
221	6	0	0	0	0	1	0
	18	0	0	0	0	0	0
222	7	0	0	0	0	1	0
	18	0	0	0	0	0	0
223	30	0	0	0	0	1	0
	18	0	0	0	0	0	0
224	31	0	0	0	0	1	0
	18	0	0	0	0	0	0
225	7	0	0	0	0	1	0
	19	0	0	0	0	0	0
226	8	0	0	0	0	1	0
	19	0	0	0	0	0	0
227	31	0	0	0	0	1	0
	19	0	0	0	0	0	0
228	32	0	0	0	0	1	0
	19	0	0	0	0	0	0
229	8	0	0	0	0	1	0
	20	0	0	0	0	0	0
230	9	0	0	0	0	1	0
	20	0	0	0	0	0	0
231	32	0	0	0	0	1	0
	20	0	0	0	0	0	0
232	33	0	0	0	0	1	0

	20	0	0	0	0	0	0
233	9	0	0	0	0	1	0
	21	0	0	0	0	0	0
234	10	0	0	0	0	1	0
	21	0	0	0	0	0	0
235	33	0	0	0	0	1	0
	21	0	0	0	0	0	0
236	34	0	0	0	0	1	0
	21	0	0	0	0	0	0
237	10	0	0	0	0	1	0
	22	0	0	0	0	0	0
238	11	0	0	0	0	1	0
	22	0	0	0	0	0	0
239	34	0	0	0	0	1	0
	22	0	0	0	0	0	0
240	35	0	0	0	0	1	0
	22	0	0	0	0	0	0
241	11	0	0	0	0	1	0
	23	0	0	0	0	0	0
242	12	0	0	0	0	1	0
	23	0	0	0	0	0	0
243	35	0	0	0	0	1	0
	23	0	0	0	0	0	0
244	36	0	0	0	0	1	0
	23	0	0	0	0	0	0
245	12	0	0	0	0	1	0
	24	0	0	0	0	0	0
246	1	0	0	0	0	1	0
	24	0	0	0	0	0	0
247	36	0	0	0	0	1	0
	24	0	0	0	0	0	0
248	25	0	0	0	0	1	0
	24	0	0	0	0	0	0
401	1	0	0	0	0	1	1
	49	0	0	0	0	0	0
402	2	0	0	0	0	1	1
	50	0	0	0	0	0	0
403	3	0	0	0	0	1	1
	51	0	0	0	0	0	0
404	4	0	0	0	0	1	1
	52	0	0	0	0	0	0
405	5	0	0	0	0	1	1
	53	0	0	0	0	0	0
406	6	0	0	0	0	1	1
	54	0	0	0	0	0	0
407	7	0	0	0	0	1	1
	55	0	0	0	0	0	0
408	8	0	0	0	0	1	1
	56	0	0	0	0	0	0
409	9	0	0	0	0	1	1
	57	0	0	0	0	0	0
410	10	0	0	0	0	1	1

	58	0	0	0	0	0	0
411	11	0	0	0	0	1	1
	59	0	0	0	0	0	0
412	12	0	0	0	0	1	1
	60	0	0	0	0	0	0
413	1	0	0	0	0	1	1
	61	0	0	0	0	0	0
414	2	0	0	0	0	1	1
	62	0	0	0	0	0	0
415	3	0	0	0	0	1	1
	63	0	0	0	0	0	0
416	4	0	0	0	0	1	1
	64	0	0	0	0	0	0
417	5	0	0	0	0	1	1
	65	0	0	0	0	0	0
418	6	0	0	0	0	1	1
	66	0	0	0	0	0	0
419	7	0	0	0	0	1	1
	67	0	0	0	0	0	0
420	8	0	0	0	0	1	1
	68	0	0	0	0	0	0
421	9	0	0	0	0	1	1
	69	0	0	0	0	0	0
422	10	0	0	0	0	1	1
	70	0	0	0	0	0	0
423	11	0	0	0	0	1	1
	71	0	0	0	0	0	0
424	12	0	0	0	0	1	1
	72	0	0	0	0	0	0

SECTION PROPERTIES

Number	Area	Jxx	Iyy	Izz	fy	fz
1	5.5760E+01	1.7570E+04	8.7860E+03	8.7860E+03	.0000E+00	.0000E
2	9.9000E+04	9.9000E+04	9.9000E+04	9.9000E+04	.0000E+00	.0000E
3	1.9500E+01	1.9220E+02	1.2820E+02	6.3800E+01	.0000E+00	.0000E
4	5.7600E+02	1.0000E-01	1.0000E-01	1.0000E-01	.0000E+00	.0000E
5	3.1800E-01	1.0000E+02	1.0000E+02	1.0000E+02	.0000E+00	.0000E

MATERIAL PROPERTIES

Material Number	Young's Modulus	Poisson's Ratio	Yield Stress	Weight Density	C.T.E.
1	2.9600E+07	2.8000E-01	3.6000E+04	2.8400E-01	6.5000E-06
2	3.5000E+06	3.0000E-01	6.0000E+03	.0000E+00	.0000E+00
3	1.5000E+06	3.0000E-01	6.0000E+03	.0000E+00	.0000E+00

N O D A L R E S T R A I N T S

Node Number	--Translations--			---Rotations---		
	X	Y	Z	X	Y	Z
1	0	1	0	1	1	1
2	0	1	0	1	1	1
3	0	1	0	1	1	1
4	0	1	0	1	1	1
5	0	1	0	1	1	1
6	0	1	0	1	1	1
7	0	1	0	1	1	1
8	0	1	0	1	1	1
9	0	1	0	1	1	1
10	0	1	0	1	1	1
11	0	1	0	1	1	1
12	0	1	0	1	1	1
49	1	1	1	0	0	0
50	1	1	1	0	0	0
51	1	1	1	0	0	0
52	1	1	1	0	0	0
53	1	1	1	0	0	0
54	1	1	1	0	0	0
55	1	1	1	0	0	0
56	1	1	1	0	0	0
57	1	1	1	0	0	0
58	1	1	1	0	0	0
59	1	1	1	0	0	0
60	1	1	1	0	0	0
61	1	1	1	0	0	0
62	1	1	1	0	0	0
63	1	1	1	0	0	0
64	1	1	1	0	0	0
65	1	1	1	0	0	0
66	1	1	1	0	0	0
67	1	1	1	0	0	0
68	1	1	1	0	0	0
69	1	1	1	0	0	0
70	1	1	1	0	0	0
71	1	1	1	0	0	0
72	1	1	1	0	0	0

S O L U T I O N P A R A M E T E R S

Number of degrees of freedom 552
 Number of restraints 120
 Number of equations 432

L O A D C A S E N U M B E R 1

DEAD LOAD

LOAD CASE PARAMETERS

Number of loaded nodes 12
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads . . . 0
 Number of elements with distributed loads . 0
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . 0

NODAL LOADS

Node Number	Forces			Moments		
	X	Y	Z	X	Y	Z
37	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
38	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
39	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
40	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
41	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
42	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
43	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
44	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
45	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
46	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
47	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000
48	.0000E+00	-1.0000E+05	.0000E+00	.0000E+00	.0000E+00	.0000

LOAD CASE NUMBER 2

LATERAL LOADS

LOAD CASE PARAMETERS

Number of loaded nodes 12
 Number of nodes with applied displacements . 0
 Indicator for gravitational loads . . . 0
 Number of elements with distributed loads . 0
 Number of elements with concentrated loads . 0
 Number of elements with thermal loads . . 0

NODAL LOADS

Node Number	Forces			Moments		
	X	Y	Z	X	Y	Z

37	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
38	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
39	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
40	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
41	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
42	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
43	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
44	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
45	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
46	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
47	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000
48	9.6600E+04	.0000E+00	2.5900E+04	.0000E+00	.0000E+00	.0000

APPENDIX C

PROGRAM LISTINGS FOR THE BLOCKED OUT-OF-CORE SOLVER

```

      subroutine stfout(s,x,y,z,idn,ndofn,ispr,spr,na,numet,lc,ise,
&imat,ioff,el,r,ide,offset,geom,prop,ncol,nblok,ksum)

```

```

c=====
c
c Subroutine to control the assembly and writing to disk of all the
c stiffness vector blocks and their associated diagonal address
c vector blocks, as well as to adjust all necessary parameters that
c are needed for other parts of the program.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c This subroutine is called by the main program and is used to
c integrate the main program to the out-of-core solver coded in
c subroutines "redstf" and "solve".
c
c=====
c
c Variable names:
c
c Variables brought in from main program:
c
c      mdofe .... maximum number of d.o.f.s per element
c      mdofn .... maximum number of d.o.f.s per node
c      mgeom .... maximum number of properties per section
c      mnodee ... maximum number of nodes per element
c      moffe .... maximum number of offset distances per element
c      mprop .... maximum number of properties per material
c      nblok .... number of stiffness vector blocks
c      nblksz ... maximum size of a stiffness vector block
c      nelem .... number of elements
c      neqn ..... number of free d.o.f.s
c      nmat ..... number of materials
c      nnode .... number of nodes
c      noff ..... number of elements with offsets
c      nsec ..... number of sections
c      nspr ..... number of nodes with spring supports
c
c Variables used only in this and subsequent subroutines:
c
c      ieqn ..... index counter over the total number of free d.o.f.s
c      idofe .... index counter over d.o.f.s per element
c      ielem .... index counter over element numbers
c      isprg .... index counter over the number of d.o.f.s with spring
c                  stiffnesses
c      jdofe .... index counter over d.o.f.s per element
c      kdofe .... index counter over d.o.f.s per element

```

```

c      ksum ..... total number of stiffness terms in all blocks
c      ktot ..... projected total number of stiffness terms in the
c                  active block
c      ktotpr ... total number of stiffness terms in the active block
c      lna ..... index for the diagonal address vector "na(lna)"
c      lrow ..... row number for the first non-zero term in a column
c      maxcol ... column number of first column in the active block
c      mincol ... column number of last column in the active block
c      nnodee ... number of nodes per element
c      ndofe .... number of d.o.f.s per element
c      ndofn .... number of d.o.f.s per node
c      nnspr .... location in global stiffness vector of a current
c                  d.o.f. spring stiffness
c      numcol ... number of columns in the active block
c      numdof ... total number of d.o.f.s of the current element

```

```

c=====

```

```

c      Array names:

```

```

c      Arrays brought in from main program:

```

```

c      el(nelem) ..... element length or thickness
c      geom(mgeom,nsec) ..... section properties
c      ide(mdofe,nelem) ..... element d.o.f. vectors and releases
c      idn(mdofn,nnode) ..... nodal d.o.f. vectors
c      imat(nelem) ..... material number
c      ioff(nelem) ..... offset number
c      isec(nelem) ..... section number
c      ispr(nspr) ..... spring node numbers
c      lc(mnodee,nelem) ..... list of connectivity
c      na(neqn) ..... diagonal address in stiffness vector
c      ncol(nblok) ..... number of columns in each stiffness
c                  vector block
c      ndofn(nnode) ..... number of d.o.f.s per node
c      numet(nelem) ..... vector list of element types
c      offset(mnodee*3,noff) .. offsets
c      prop(mprop,nmat) ..... material properties
c      r(9,nelem) ..... element rotation matrix
c      s(nblksh) ..... global stiffness vector
c      spr(mdofn,nspr) ..... spring stiffnesses
c      x(nnode) ..... x-coordinates of nodes
c      y(nnode) ..... y-coordinates of nodes
c      z(nnode) ..... z-coordinates of nodes

```

```

c      Arrays used only in this or subsequent subroutines:

```

```

c      ndofe(3) ..... number of d.o.f.s for each type of
c                  element

```

```

c=====

```

```

c

```

```

c Subroutines called:
c
c   blkstf .... assembles a global stiffness vector and writes it
c                 and its associated diagonal address vector to disk.
c
c=====
c
c Element types:
c
c   1  Truss:   2 nodes
c                3 d.o.f. per node
c
c   2  Beam:   2 nodes
c                6 d.o.f. per node
c
c   3  Plate:  4 nodes
c                6 d.o.f. per node
c
c=====
c
c Files and file names:
c
c   * .. screen:      displays program progress
c
c=====
c
c a. Set block size.
c
c   parameter ( nblksz = 8000 )
c
c   common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
c   &nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
c   &ndof,nres,neqn,mdofn,mnodee,mdofe,mprop,mgeom,moffe,iai,iao
c
c   real*8 s(nblksz)
c
c   dimension x(1),y(1),z(1),idn(mdofn,1),ndofn(1),ispr(1),
c   &spr(mdofn,1),na(1),numet(1),lc(mnodee,1),isec(1),imat(1),
c   &ioff(1),el(1),r(9,1),ide(mdofe,1),offset(moffe,1),
c   &geom(mgeom,1),prop(mprop,1),ncol(1)
c
c   dimension ndofe(3)
c
c   integer*4 ksum
c
c b. Set the number of d.o.f.s for each type of element.
c
c   ndofe(1)=6
c   ndofe(2)=12
c   ndofe(3)=24
c
c
c c. Initialize the diagonal address vector "na( )".

```



```

c
      do 1 ieqn=1,neqn
        na(ieqn)=ieqn
      1 continue

c*****
c
c Section to set "na(icol)" equal to the row number of the first
c non-zero entry in column "icol".
c
c*****
c
c d. Loop over all the elements.
c
      do 4 ielem=1,nelem
c
c e. Set the number of d.o.f.s for the current element and set the
c row number of the first non-zero entry for all columns
c associated with this element to the number of equations.
c
        numdof=ndofe(numet(ielem))
        lrow=neqn
c
c f. Loop over all the d.o.f.s of the current element. Check to see
c if the current d.o.f. is less than the value stored in "lrow". If
c it is, reset the value of "lrow" to this new d.o.f. number. The
c result is that after all the d.o.f.s of the current element have
c been examined, "lrow" will have the value of the lowest d.o.f.
c number of the current element. This means that the row number
c of the first non-zero entry in all the columns corresponding to
c the d.o.f. numbers of the current element will be at least the
c number stored in "lrow".
c
        do 2 jdofe=1,numdof
          if (ide(jdofe,ielem).lt.lrow) lrow=ide(jdofe,ielem)
        2 continue
c
c g. Loop over all the d.o.f.s of the current element again. First
c check to see if the d.o.f. is less than the number of equations
c (this means it is a free d.o.f.). If this is true, then check
c the row number of the first non-zero entry for the column with
c the current d.o.f. number to see if "lrow" is less than that
c value. If that is true, reset the row number of the first
c non-zero entry stored in "na( )" for the current d.o.f. number
c to the value stored in "lrow".
c
        do 3 jdofe=1,numdof
          kdofe=ide(jdofe,ielem)
          if (kdofe.le.neqn) then
            if (lrow.lt.na(kdofe)) na(kdofe)=lrow
          endif
        3 continue

```

```

c
c h. Loop back and examine the next element.
c
c   4 continue
c
c i. Check to make sure that every column has at least one off-
c   diagonal term in the column skyline. (This is necessary due to
c   an error in the solution algorithm that will occur otherwise).
c
c   do 5 ieqn=2,neqn
c       if (na(ieqn).eq.0) na(ieqn)=na(ieqn)-1
c   5 continue

c*****
c
c Section to assemble and write to disk the global stiffness vector
c blocks and their associated diagonal address vector blocks.
c
c*****
c
c j. Initialize indexes and counters.
c
c   ksum=0
c   ktot=1
c   lna=1
c   mincol=1
c   maxcol=0
c   nblok=0
c
c k. Loop over all the columns except the first column.
c
c   do 6 ieqn=2,neqn
c
c l. Set "ktotpr" to the total number of stiffness terms needed in
c   the active global stiffness vector block for all the columns
c   from the first column of the active block through the previously
c   current column of the active block and set "ktot" to the total
c   number of stiffness terms needed in the active global stiffness
c   vector block for all the columns from the first column of the
c   active block through the current column of the active block.
c
c   ktotpr=ktot
c   ktot=ktot+ieqn-na(ieqn)+1
c
c m. Check to see if the total number of terms needed in the
c   active block through the current column exceeds the maximum
c   allowable for a block size.
c
c   if (ktot.gt.nblks) then
c
c n. If the above condition is true, set parameters associated with
c   this block, go assemble the global stiffness vector for this block

```

```

c   and write it and its associated diagonal address vector to disk,
c   and then reset the necessary indexes needed for the next block.
c

```

```

      ksum=ksum+ktotpr
      mincol=maxcol+1
      maxcol=ieqn-1
      numcol=maxcol-mincol+1
      nblok=nblok+1
      ncol(nblok)=numcol

```

```

      call blkstf(s,x,y,z,idn,ndofn,ispr,spr,na,numet,lc,isec,
&               imat,ioff,el,r,ide,offset,geom,prop,ndofe,
&               ktotpr,mincol,maxcol,numcol,nblksz)

```

```

      ktot=ktot-ktotpr
      lna=0
    endif

```

```

c
c o. Increment the diagonal address vector index, set the address of
c   the diagonal term for the current d.o.f., then loop back up and
c   examine next d.o.f..
c

```

```

      lna=lna+1
      na(lna)=ktot
6 continue

```

```

c
c p. Assemble the last block.
c

```

```

      ksum=ksum+ktot
      mincol=maxcol+1
      maxcol=ieqn-1
      numcol=maxcol-mincol+1
      nblok=nblok+1
      ncol(nblok)=numcol

```

```

      call blkstf(s,x,y,z,idn,ndofn,ispr,spr,na,numet,lc,isec,imat,
&               ioff,el,r,ide,offset,geom,prop,ndofe,ktot,mincol,
&               maxcol,numcol,nblksz)

```

```

c
c q. Return to the main program.
c

```

```

      return
    end

```

```

      subroutine blkstf(s,x,y,z,idn,ndofn,ispr,spr,na,numet,lc,isec,
&imat,ioff,el,r,ide,offset,geom,prop,ndofe,ktot,mincol,maxcol,
&numcol,nblksz)

```

```

c=====
c
c Subroutine that assembles a global stiffness vector and writes it
c and its associated diagonal address vector to disk.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c This subroutine is called by the subroutine "stfout" and is part
c of the integration of the main program to the out-of-core solver
c which is controlled by subroutine "stfout".
c
c=====
c
c Variable names:
c
c   Variables brought in from main program or calling subroutine:
c
c   ktot ..... total number of stiffness terms in the active block
c   maxcol ... column number of first column in the active block
c   mdofe .... maximum number of d.o.f.s per element
c   mdofn .... maximum number of d.o.f.s per node
c   mgeom .... maximum number of properties per section
c   mincol ... column number of last column in the active block
c   mnodee ... maximum number of nodes per element
c   moffe .... maximum number of offset distances per element
c   mprop .... maximum number of properties per material
c   nblok .... number of stiffness vector blocks
c   nblksize ... maximum size of a stiffness vector block
c   nelem .... number of elements
c   nmat ..... number of materials
c   nnode .... number of nodes
c   noff ..... number of elements with offsets
c   nsec ..... number of sections
c   nspr ..... number of nodes with spring supports
c   numcol ... number of columns in the active block
c
c   Variables used only in this and subsequent subroutines:
c
c   ielem .... index counter over element numbers
c   in ..... flag to indicate if an element has stiffness terms
c           that belong in the active block
c   istiff.... index counter over the global stiffness vector
c   kdofe .... index counter over d.o.f.s per element
c   kmat ..... material number of active element
c   koff ..... offset number of active element
c   ksec ..... section number of active element
c   mdofeo ... number of d.o.f.s per element

```

```

c
c=====
c
c  Array names:
c
c    Arrays brought in from main program:
c
c      el(nelem) ..... element length or thickness
c      geom(mgeom,nsec) ..... section properties
c      ide(mdofe,nelem) ..... element d.o.f. vectors and releases
c      idn(mdofn,nnode) ..... nodal d.o.f. vectors
c      imat(nelem) ..... material number
c      ioff(nelem) ..... offset number
c      isec(nelem) ..... section number
c      ispr(nspr) ..... spring node numbers
c      lc(mnodee,nelem) ..... list of connectivity
c      na(numcol) ..... diagonal address in stiffness vector
c      ncol(nblok) ..... number of columns in each stiffness
c                          vector block
c      ndofe(3) ..... number of d.o.f.s for each type of
c                          element
c      ndofn(nnode) ..... number of d.o.f.s per node
c      numet(nelem) ..... vector list of element types
c      offset(mnodee*3,noff) .. offsets
c      prop(mprop,nmat) ..... material properties
c      r(9,nelem) ..... element rotation matrix
c      s(nblksize) ..... global stiffness vector
c      spr(mdofn,nspr) ..... spring stiffnesses
c      x(nnode) ..... x-coordinates of nodes
c      y(nnode) ..... y-coordinates of nodes
c      z(nnode) ..... z-coordinates of nodes
c
c    Arrays used only in this or subsequent subroutines:
c
c      sb(12,12) ..... beam element stiffness matrix
c      sp(24,24) ..... plate element stiffness matrix
c      st(6,6) ..... truss element stiffness matrix
c
c=====
c
c  Subroutines called:
c
c      asstif .... adds element stiffnesses to a global stiffness vector
c      stiffs .... adds spring stiffnesses to a global stiffness vector
c      ostif2 .... offsets beam stiffness matrix, if offsets exist
c      ostif3 .... offsets plate stiffness matrix, if offsets exist
c      rstif2 .... rotates beam stiffness matrix to global coordinates
c      rstif1 .... rotates truss stiffness matrix to global coordinates
c      rstif3 .... rotates plate stiffness matrix to global coordinates
c      stiff1 .... calculates stiffness matrix for truss element
c      stiff2 .... calculates stiffness matrix for beam element
c      stiff3 .... calculates stiffness matrix for plate element

```

```

c
c=====
c
c Element types:
c
c   1  Truss:  2 nodes
c             3 d.o.f. per node
c
c   2  Beam:   2 nodes
c             6 d.o.f. per node
c
c   3  Plate:  4 nodes
c             6 d.o.f. per node
c
c=====
c
c Files and file names:
c
c   * .. screen:      displays program progress
c
c   1 .. unreda.dmp:  scratch file for the unreduced stiffness
c                     vector blocks
c
c   3 .. diagad.dmp:  scratch file for the diagonal address
c                     vector blocks
c
c=====

      common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
&nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
&ndof,nres,neqn,mdofn,mnodee,mdofe,mprop,mgeom,moffe,iai,iao

      real*8 s(nblksz),st(6,6),sb(12,12),sp(24,24)

      dimension x(1),y(1),z(1),idn(mdofn,1),ndofn(1),ispr(1),
&spr(mdofn,1),na(1),numet(1),lc(mnodee,1),isec(1),imat(1),
&ioff(1),el(1),r(9,1),ide(mdofe,1),offset(moffe,1),
&geom(mgeom,1),prop(mprop,1),ndofe(3)
c
c a. Initialize the global stiffness vector.
c
      do 1 istiff=1,ktot
        s(istiff)=0.0
      1 continue
c
c b. Loop over all the elements.
c
      do 8 ielem=1,nelem
c
c c. Set a flag that will indicate if any of the d.o.f.s of the
c    current element lie in the range of the column numbers of the
c    active block to the negative condition. Then loop over all the

```

```

c   d.o.f.s of the current element and check each one to see if any
c   lie in the range of the d.o.f.s of the active block. When one
c   d.o.f. of the current element is found to lie in the range of the
c   d.o.f.s of the active block, set the indicator flag to positive
c   and then exit the loop.
c
      in=0
      do 2 kdofe=1,ndofe(numet(ielem))
        if (ide(kdofe,ielem).ge.mincol) then
          if (ide(kdofe,ielem).le.maxcol) then
            in=1
            goto 3
          endif
        endif
      2 continue
c
c d. If the indicator flag is positive, then the element stiffness
c   contributions have to be added to the current global stiffness
c   vector block.
c
      3 if (in.eq.1) then
c
c e. Set the section number, material number, and offset number of
c   the current element.
c
      ksec=isec(ielem)
      kmat=imat(ielem)
      koff=ioff(ielem)
c
c f. Determine the type of the current element and go to the
c   appropriate section.
c
      goto (4,5,6) numet(ielem)
c
c * * * truss * * * * *
c
c gt. If the current element is a truss, get its stiffness matrix and
c   rotate it to global coordinates. Then add the necessary
c   element stiffness terms to the global stiffness vector and skip
c   the beam and plate sections.
c
      4      mdofeo=ndofe(numet(ielem))
         call stiff1(st,el(ielem),geom(1,ksec),prop(1,kmat))
         call rstif1(st,r(1,ielem))
         call asstif(s,st,na,ide(1,ielem),mincol,maxcol,mdofeo,
           &          nblksz)
         go to 7
c
c * * * beam * * * * *
c
c gb. If the current element is a beam, get its stiffness matrix and
c   rotate it to global coordinates. If offsets exist for this

```

```

c      element, offset the element stiffness matrix, then add the
c      necessary element stiffness terms to the global stiffness
c      vector and skip the plate section.
c
c      5      mdofeo=ndofe(numet(ielem))
c             call stiff2(sb,el(ielem),geom(1,ksec),prop(1,kmat))
c             call rstif2(sb,r(1,ielem))
c             if (koff.gt.0) call ostif2(sb,offset(1,koff))
c             call asstif(s,sb,na,ide(1,ielem),mincol,maxcol,mdofeo,
c             &          nblksz)
c             go to 7
c
c      * * * plate * * *
c
c      gp. If the current element is a plate, get its stiffness matrix and
c      rotate it to global coordinates. If offsets exist for this
c      element, offset the element stiffness matrix, then add the
c      necessary element stiffness terms to the global stiffness
c      vector.
c
c      6      mdofeo=ndofe(numet(ielem))
c             call stiff3(sp,x,y,z,lc(1,ielem),r(1,ielem),
c             &          el(ielem),prop(1,kmat),offset(1,koff),koff)
c             call rstif3(sp,r(1,ielem))
c             if (koff.gt.0) call ostif3(sp,offset(1,koff))
c             call asstif(s,sp,na,ide(1,ielem),mincol,maxcol,mdofeo,
c             &          nblksz)
c
c      7      continue
c
c      h. End of section that adds element contributions to the global
c      stiffness vector block if the current element has d.o.f.s in the
c      range of the columns of the active global stiffness vector block.
c
c      endif
c
c      i. Loop back and look at the next element.
c
c      8 continue
c
c      j. If there are springs, add the contributions of spring
c      stiffnesses to the global stiffness vector.
c
c      if (nspr.ne.0) then
c         call stiffs(s,idn,ndofn,ispr,spr,na,mincol,maxcol,nblksz)
c      endif
c
c      k. Write the current stiffness vector block and its associated
c      diagonal address vector to disk.
c
c         write(1) ktot, (s(istiff),istiff=1,ktot)
c         write(3) numcol, (na(icol),icol=1,numcol)

```



```

c
c 1. Return to subroutine "stfout".
c

```

```

    return
end

```

```

subroutine asstif(s,se,na,ide,mincol,maxcol,ndofe,nblksz)

```

```

c=====
c
c Subroutine to add contributions from an element stiffness matrix
c to a global stiffness vector block.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c This subroutine is called by the subroutine "blkstf" and is part
c of the integration of the main program to the out-of-core solver.
c The integration is controlled by subroutine "stfout".
c
c=====
c
c Variable names:
c
c   Variables brought in from main program or calling subroutine:
c
c   maxcol ... column number of first column in the active block
c   mincol ... column number of last column in the active block
c   mdofe ... maximum number of d.o.f.s per element
c   nblksz ... maximum size of a stiffness vector block
c   ndofe ... number of d.o.f.s per element
c   nelem ... number of elements
c   numcol ... number of columns in the active block
c
c   Variables used only in this and subsequent subroutines:
c
c   icol ..... column number for current stiffness term
c   index .... index for diagonal address vector "na(index)"
c   irow ..... row number for current stiffness term
c   item ..... temporary storage area
c   jdofe .... index counter over element d.o.f.s
c   kdofe .... index counter over element d.o.f.s
c   locat .... location in global stiffness vector of a current
c               d.o.f. spring stiffness
c

```

```

C=====
C
C  Array names:
C
C    Arrays brought in from main program:
C
C      ide(ndofe,nelem) ..... element d.o.f. vectors and releases
C      na(numcol) ..... diagonal address in stiffness vector
C      s(nblkosz) ..... global stiffness vector
C      se(ndofe,ndofe)..... element stiffness matrix
C
C=====

      real*8 s(nblkosz),se(ndofe,ndofe)
      dimension na(1),ide(ndofe)

C
C a. Loop over all the d.o.f.s of the active element.
C
      do 3 jdofe=1,ndofe
C
C b. Loop over all the d.o.f.s of the active elements from the
C    outer loop d.o.f. to the last d.o.f..
C
      do 4 kdofe=jdofe,ndofe
C
C c. Set the row number to the d.o.f. number of the outer loop.
C
      irow=ide(jdofe)
C
C d. Set the column number to the d.o.f. number of the inner loop.
C
      icol=ide(kdofe)
C
C e. If the column d.o.f. number is less than the row number d.o.f.,
C    switch the row and column numbers.
C
      if (icol.lt.irow) then
        item=irow
        irow=icol
        icol=item
      endif
C
C f. Check to see if the row and column indexes of the current
C    stiffness term lie in the range of the columns in the active
C    global stiffness vector block.
C
      if (irow.le.maxcol) then
        if (icol.le.maxcol) then
          if (icol.ge.mincol) then
C
C g. If the above is true, get the address in the global stiffness
C    vector where the current element stiffness term should go,

```

```

c      and then add the current element contribution to that global
c      stiffness term.
c
c          index=icol-minicol+1
c          locat=na(index)-icol+irow
c          s(locat)=s(locat)+se(jdofe,kdofe)
c
c h. End of "if" sections to determine whether the current element
c      stiffness term belongs in the active global stiffness block.
c
c          endif
c          endif
c          endif
c
c i. Continue the inner loop over the element d.o.f.s.
c
c      4  continue
c
c j. Continue the outer loop over the element d.o.f.s.
c
c      3  continue
c
c k. Return to subroutine "blkstf".
c
c      return
c      end

```

```

      subroutine stiff(s,idn,ndofn,ispr,spr,na,mincol,maxcol,nblksz)

```

```

c=====
c
c Subroutine to add spring stiffnesses to a global stiffness vector
c block.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c This subroutine is called by the subroutine "blkstf" and is part
c of the integration of the main program to the out-of-core solver.
c The integration is controlled by subroutine "stfout".
c
c=====
c
c Variable names:
c

```

```

c   Variables brought in from main program and calling subroutine:
c
c   nspr ..... number of nodes with spring supports
c   mdofn .... maximum number of d.o.f.s per node
c   maxcol ... column number of first column in the active block
c   mincol ... column number of last column in the active block
c
c   Variables used only in this and subsequent subroutines:
c
c   icol ..... column number for current d.o.f. diagonal term
c   index .... index for diagonal address vector "na(index)"
c   jdofn .... index counter over nodal d.o.f.s
c   jsprg .... index counter over the number of nodes with spring
c               supports
c   locat .... location in global stiffness vector of a current
c               d.o.f. spring stiffness
c   nodspr ... node number of active node
c   numdof ... total number of d.o.f.s of the active node
c
c=====
c
c   Array names:
c
c   Arrays brought in from calling subroutine:
c
c   idn(mdofn,nnode) ..... nodal d.o.f. vectors
c   ispr(nspr) ..... spring node numbers
c   na(neqn) ..... diagonal address in stiffness vector
c   ndofn(nnode) ..... number of d.o.f.s per node
c   s(nblksz) ..... global stiffness vector
c   spr(mdofn,nspr) ..... spring stiffnesses
c
c=====
c
c   common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
c   &nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
c   &ndof,nres,neqn,mdofn,mnodee,mdofe,mprop,mgeom,moffe,iai,iao
c
c   real*8 s(nblksz)
c
c   dimension idn(mdofn,1),ndofn(1),ispr(1),spr(mdofn,1),na(1)
c
c a. Loop over the total number of nodes with springs.
c
c   do 2 jsprg=1,nspr
c
c b. Set node number of current spring number.
c
c   nodspr=ispr(jsprg)
c
c c. Set number of d.o.f.s of active node.
c

```

```

      numdof=ndofn(nodspr)
c
c d. Loop over all the active node d.o.f.s.
c
      do 1 jdofn=1,numdof
c
c e. Get the column number for the current d.o.f. of the active node.
c
      icol=idn(jdofn,nodspr)
c
c f. If the column number of the current d.o.f. of the active node is
c    in the range of column numbers for the current block, get the
c    address in the active global stiffness vector block of the
c    diagonal term for the current d.o.f. of the active node. Then
c    add the spring stiffness to the stiffness vector term at the
c    found diagonal address.
c
      if (icol.ge.mincol) then
        if (icol.le.maxcol) then
          index=icol-mincol+1
          locat=na(index)
          s(locat)=s(locat)+dble(spr(jdofn,jsprg))
        endif
      endif
c
c g. Continue the loop over the active node d.o.f.s.
c
      1  continue
c
c h. Continue the loop over all the nodes with springs.
c
      2  continue
c
c i. Return to subroutine "blkstf".
c
      return
      end

```

```

      subroutine redstf(a,b,pivot,na,nb,ncol,nblok,neqn)

```

```

c=====
c
c Subroutine to do out-of-core reduction on the stiffness vector
c blocks.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.

```

```

c
c January, 1987
c
c Based on:
c
c   Mondkar, D. P., and G. H. Powell, "Large Capacity Equation
c   Solver for Structural Analysis", Computers & Structures,
c   Vol 4, 1974, pp. 699-728.
c
c This subroutine is called by the main program to reduce the global
c stiffness vector blocks stored on the back-up storage device. It
c is called once.
c
c=====
c  Generally:
c
c   i***** .. is an index over row numbers
c   j***** .. is an index over column numbers
c
c       where: ***** means any legal variable name characters
c
c=====
c  Variable names:
c
c  Variables brought in from main program:
c
c   nblok .... number of vector stiffness blocks
c   neqn ..... number of equations
c
c  Variables used only in this subroutine:
c
c   div ..... temporary storage of division of an off-diagonal
c               term by a diagonal term
c   ifirst ... global row number of the first non-zero term (i.e.
c               the top of the skyline) of the current column
c   ii ..... dummy index counter
c   ilast .... global row number of last row of a loop over global
c               row numbers
c   iloop .... index counter over global row numbers
c   ip ..... number of terms in the vector space "pivot( )"
c   iread .... index counter of the terms going into the vector
c               spaces "a( )", "b( )", or "pivot( )"
c   ireada ... number of terms in the vector space "a( )"
c   ireadb ... number of terms in the vector space "b( )"
c   ireadm ... number of terms in the vector space "nb( )"
c   ireadn ... number of terms in the vector space "na( )"
c   istart ... global row number of first row of a loop over global
c               row numbers
c   jcolct ... index counter over the columns of the active vector
c               stiffness block
c   jcoled ... address in the vector space "a( )" of the first

```

```

c          non-zero term (i.e. the top of the skyline) of the
c          current column
c      jcp1 ..... global column number of the first column of the
c          current subordinate vector stiffness block
c      jdum ..... dummy variable
c      jfirst ... global column number of the first column of a loop
c          over column numbers
c      jfirst1 ... global column number of the first column of a loop
c          over column numbers for loops within the active
c          vector stiffness block
c      jia ..... address in the vector space "a( )" of the term
c          being reduced
c      jj ..... dummy index counter
c      jlast .... global column number of the last column of a loop
c          over column numbers
c      jloop .... index counter over global column numbers
c      jm ..... address offset index
c      kfirst ... global row number where a summation will start
c      klast .... global row number where a summation will end
c      kloop .... index counter over the global row numbers of a
c          summation
c      lcolsc ... global column number of the last column of the
c          current subordinate vector stiffness block
c      lcolsp ... global column number of the last column of the
c          previous subordinate vector stiffness block
c      lsblok ... index counter over the subordinate vector stiffness
c          blocks
c      min ..... lowest global row number of any term in the active
c          vector stiffness block
c      nai ..... address in the vector space "a( )" of the diagonal
c          term of an already reduced column
c      naii ..... dummy index that contains the address in the vector
c          space "a( )" of the diagonal term of the column
c          previous to the current already reduced column being
c          used to reduce the current column if that column
c          were a complete column.
c      naip ..... address in the vector space "a( )" of the first
c          non-zero term (i.e. the top of the skyline) of the
c          current already reduced column
c      naj ..... address in the vector space "a( )" of the diagonal
c          term of the current column
c      najj ..... dummy index that contains the address in the vector
c          space "a( )" of the diagonal term of the column
c          previous to the current column if the current column
c          were a complete column.
c      nbi ..... address in the vector space "b( )" of the diagonal
c          term of an already reduced column
c      nbii ..... dummy index that contains the address in the vector
c          space "b( )" of the diagonal term of the column
c          previous to the current already reduced column being
c          used to reduce the current column if that column
c          were a complete column.

```

```

c      nbip ..... address in the vector space "b( )" of the first
c                  non-zero term (i.e. the top of the skyline) of the
c                  already reduced column that is needed for reducing
c                  a term in the current column
c      nblok .... number of vector stiffness blocks
c      ncola .... number of columns in the vector stiffness block
c                  stored in the vector space "a( )"
c      ncolb .... number of columns in the vector stiffness block
c                  stored in the vector space "b( )"
c      npblok ... index counter over the vector stiffness blocks
c      nsblok ... number of subordinate blocks to the current
c                  vector stiffness block
c      suma ..... summation variable
c
c=====
c
c      Array names:
c
c      Arrays brought in from main program:
c
c      a(mtots) ..... vector space for the active block
c      b(mtott) ..... vector space for the current subordinate
c                  block
c      na(ncol(nblok)) ... vector space for the addresses of the
c                  diagonal terms of the active block
c      nb(ncol(nblok)) ... vector space for the addresses of the
c                  diagonal terms of the current subordinate
c                  block
c      ncol(nblok) ..... number of columns in each vector stiffness
c                  block
c      pivot(mtotu) ..... vector space for the diagonal terms of the
c                  current subordinate block
c
c=====
c
c      Files and file names:
c
c      1 .. unreda.dmp:  file of unreduced vector stiffness blocks
c
c      2 .. reduca.dmp:  file of reduced vector stiffness blocks
c
c      3 .. diagad.dmp:  file of addresses of the diagonal terms in both
c                  "unreda.dmp" and "reduca.dmp"
c
c      5 .. pivott.dmp:  file of diagonal terms of the reduced vector
c                  stiffness blocks
c
c=====
c
c      a.  Set array sizes and set array and variable types.
c
c          real*8 a(8000),b(8000),pivot(5000),div,suma

```



```

dimension na(1),nb(1),ncol(1)
c
c b. Initialize first global column number and last global column
c   number indexes of the active block, and the index of the global
c   column number of the last column of the previous subordinate
c   block.
c
      jfirst=1
      jlast=0
      lcolsp=0
c
c c. Set all needed files to their first record.
c
      rewind 1
      rewind 2
      rewind 3
      rewind 5
c
c d. Loop over all the unreduced vector stiffness blocks.
c
      do 15 npblok=1,nblok
c
c e. Read the active unreduced stiffness vector block and its
c   associated diagonal address vector block.
c
      read(1) ireada,(a(iread),iread=1,ireada)
      read(3) ireadn,(na(iread),iread=1,ireadn)
c
c f. Set the number of columns in the unreduced active stiffness
c   vector block.
c
      ncola=ncol(npblok)
c
c g. Set the last column number index to the last global column
c   number of the active block.
c
      jlast=jlast+ncola
c
c h. Check to see if this is the first block to be reduced. If it
c   is, skip to the appropriate section, if it is not, continue
c   at this point.
c
      if (npblok.ne.1) then
c
c i. Find the lowest global row number of any term in any of the
c   columns of the active block. This will define the horizontal
c   shadow of the active block. The value will be stored in "min".
c
      jcoled=1
      min=jfirst
      jdum=0

```

```

do 1 jloop=jfirst,jlast
  jdum=jdum+1
  naj=na(jdum)
  ifirst=jloop-naj+jcoled
  if (min.gt.ifirst) min=ifirst
  jcoled=naj+1
1  continue
c
c j. Set the number of subordinate blocks for the active block.
c
      nsblok=npblok-1
c
c k. Reset the needed files associated with the already reduced
c   stiffness vector blocks to their first record.
c
      rewind 2
      rewind 3
c
c l. Initialize the index of the last column number of the previous
c   subordinate block to zero since no subordinate blocks have been
c   brought in yet.
c
      lcolsp=0
c
c m. Loop over all the subordinate blocks of the active block.
c
      do 5 lsblok=1,nsblok
c
c n. Set the number of columns in the current subordinate block.
c
      ncolb=ncol(lsblok)
c
c o. Set the global column number of the last column of the
c   current subordinate block.
c
      lcolsc=lcolsp+ncolb
c
c p. Check to see if the current subordinate block is in the
c   horizontal shadow of the active block. If it is not, skip to
c   the appropriate section. If it is, continue at this point.
c
      if (min.lt.lcolsc) then
c
c q. If the current subordinate block is needed to reduce terms in
c   the active block, first read the current subordinate block and
c   its diagonal address vector.
c
      read(2) ireadb,(b(iread),iread=1,ireadb)
      read(3) ireadm,(nb(iread),iread=1,ireadm)
c
c r. Initialize an index counter that will count over the columns of
c   the active block. Initialize an index that will give the

```

```

c   address in the vector "a( )" of the first non-zero term (i.e.
c   the top of the skyline) of the current column of the active
c   block. Set the global column number of the first column of
c   the current subordinate block.
c
c       jj=0
c       jcoled=1
c       jcp1=lcolsp+1
c
c s. Loop over all the columns of the active block. The loop will
c   be performed over the global column numbers of the current
c   block and not the counter index.
c
c       do 4 jloop=jfirst,jlast
c
c t. Increment the column index counter by one, and set the index
c   that contains the address in the vector "a( )" of the diagonal
c   term for the current column being reduced.
c
c       jj=jj+1
c       naj=na(jj)
c
c u. Set a dummy index that contains the address in the vector "a( )"
c   of the diagonal term of the previous column if the current
c   column was a complete column.
c
c       najj=naj-jloop
c
c v. Find the global row number of the first non-zero term (i.e. the
c   top of the skyline) of the current column by taking the
c   difference between the address in the vector "a( )" of the
c   true top of the skyline and the dummy diagonal term of the
c   previous column if the current column was a complete column.
c
c       ifirst=jcoled-najj
c
c w. Check to see if the row number of the first non-zero term of the
c   current column of the active block is less than the last column
c   number of the current subordinate block. If it is, some of the
c   terms of the current column in the active block will have to be
c   reduced by some of the terms in the current subordinate block.
c
c       if (ifirst.lt.lcolsc) then
c
c x. If the above condition is true, set the starting global row
c   number in the current column where the reduction of the
c   off-diagonal terms will start. This will be the greater of:
c   1) the global row number of the second non-zero term of the
c   current column (since the first non-zero term of any column
c   does not undergo the partial reduction step), or 2) the first
c   global column number of the current subordinate block.
c

```

```

c          1start=max0(1first+1,jcp1)
c y. Find the column counter index of the already reduced column in
c the current subordinate block that will be needed to reduce the
c off-diagonal term of the current column in the active block with
c the global row number "1start". Then get the address in the
c vector "b( )" of the first non-zero term (i.e. the top of the
c skyline) of the already reduced column.
c
c          ii=1start-1colsp
c          if (ii.eq.1) then
c            nbip=1
c          else
c            nbip=nb(ii-1)+1
c          endif
c
c z. Loop over all the rows of the current column of the active block
c with global row numbers greater than or equal to "1start" for
c reduction by terms of the subordinate block.
c
c          do 3 iloop=1start,1colsc
c
c aa. Set the address in the vector "b( )" of the diagonal term of
c the column of already reduced terms that are needed to partially
c reduce the current term of the current column.
c
c          nbi=nb(ii)
c
c ab. Set a dummy index that contains the address in the vector "b( )"
c of the diagonal term of the previous column to the needed
c already reduced column if the already reduced column was a
c complete column.
c
c          nbii=nbi-iloop
c
c ac. Find the global row number of the first non-zero term (i.e. the
c top of the skyline) of the already reduced column by taking the
c difference between the address in the vector "b( )" of the
c true top of the skyline and the dummy diagonal term of the column
c previous to the already reduced column if the already reduced
c column was a complete column.
c
c          kfirst=nbip-nbii
c
c ad. Check to see if the global row number of the first non-zero
c term of the column of already reduced terms is less than the
c global row number of the term being reduced. If it is, there
c will be some partial reduction of the current term. If it is
c not, there are no terms in the already reduced column that
c affect the partial reduction of the current off-diagonal term
c of the current column, and this section can be skipped.
c

```

```

        if (kfirst.lt.iloop) then
c
c ae. Set the address in the vector "a( )" of the current
c   off-diagonal term that is about to be partially reduced.
c
c       jia=najj+iloop
c
c af. Set the addresses in the vector "a( )" of the first and last
c   terms in the current column that will be needed to partially
c   reduce the current term of that column. The first term that
c   will be needed is the one with the global row number that is
c   the greater of: 1) the first non-zero term of the current
c   column (i.e. the top of the skyline), 2) the first non-zero
c   term of the already reduced column (i.e. the top of the
c   skyline) that is needed to partially reduce the current term of
c   the current column. The last term that will be needed
c   is the term in the current column just above the current term
c   that is being partially reduced.
c
c       kfirst=najj+max0(ifirst,kfirst)
c       klast=jia-1
c
c ag. Set an address offset index that will give the address in the
c   vector "b( )" for terms in the already reduced column with
c   the same global row numbers as the terms in the current column
c   of the active block stored in vector space "a( )".
c
c       jm=nbii-najj
c
c ah. Initialize a summation variable.
c
c       suma=0.0
c
c ai. Loop over the correct terms in the column that are being reduced
c   and the terms in the already reduced column and perform the
c   summation.
c
c       do 2 kloop=kfirst,klast
c           suma=suma+b(jm+kloop)*a(kloop)
c       2   continue
c
c aj. Subtract the summation calculated above to partially reduce the
c   current term of the current column of the active block.
c
c       a(jia)=a(jia)-suma
c
c ak. End of the section to partially reduce the current term of the
c   current column of the active block.
c
c       endif
c
c al. Increment the index counter of the already reduced column by

```

```

c      one so that the correct reduced column will be used for the
c      partial reduction of the next off-diagonal term of the current
c      column. Then set the address in "b( )" of the first non-zero
c      term (i.e. the top of the skyline) of that already reduced
c      column.
c
c              ii=ii+1
c              nbip=nbi+1
c
c am. Loop back up and partially reduce the next off-diagonal term
c      of the current column.
c
c      3          continue
c
c an. End of the section to do the partial reduction of the
c      off-diagonal terms of the current column of the active block.
c
c              endif
c
c ao. Set the address in the vector "a( )" of the first non-zero term
c      (i.e. the top of the skyline) of the next column in the active
c      block.
c
c              jcoled=naj+1
c
c ap. Loop back up and partially reduce the terms of the next column
c      of the active block that are in the horizontal shadow of the
c      current subordinate block.
c
c      4          continue
c
c aq. If no reduction is needed for this subordinate block, do a
c      dummy read on the current records to skip to the next records.
c
c              else
c
c                  read(2)
c                  read(3)
c
c ar. End of section that does the partial reduction of terms in
c      the active block by terms in the current subordinate block.
c
c              endif
c
c as. Set the index of the last global column number of the previous
c      subordinate block equal to the last global column number of
c      the current subordinate block.
c
c              lcolsp=lcolsc
c
c at. Loop back and do the the partial reduction of terms in the
c      active block by the terms in the next subordinate block.

```

```

c
c      5      continue

c=====
c
c Section to do the partial reduction of terms in the active block
c that require reduction by terms in the active block that are in the
c horizontal shadow of any subordinate block and also do the final
c reduction of terms in the active block and by the diagonal terms
c of all its subordinate blocks and do the partial reduction of the
c diagonal terms of the active block.
c
c=====
c
c au. Set the file of diagonal terms to its first record.
c
c      rewind 5
c
c av. Reinitialize the index of the last column number of the
c      previous subordinate block.
c
c      lcolsp=0
c
c aw. Loop over all the subordinate blocks of the current active
c      block.
c
c      do 10 lsblok=1,nsblok
c
c ax. Set the number of columns of the current subordinate block.
c
c      ncolb=ncol(lsblok)
c
c ay. Set the global column number of the last column in the current
c      subordinate block.
c
c      lcolsc=lcolsp+ncolb
c
c az. Check to see if the active block lies in the horizontal shadow
c      of the current subordinate block. If it does not, skip to the
c      appropriate section.
c
c      if (min.le.lcolsc) then
c
c ba. Read the diagonal terms of the current subordinate block.
c
c      read(5) ip,(pivot(iread),iread=1,ip)
c
c bb. Initialize an index counter that will count over the columns of
c      the active block. Initialize an index that will give the
c      address in the vector "a( )" of the first non-zero term (i.e.
c      the top of the skyline) of the current column of the active
c      block. Set the global column number of the first column of

```

```

c   the current subordinate block.
c
c       jj=0
c       jcoled=1
c       jcp1=lcolsp+1
c
c bc. Loop over all the columns of the active block. The loop will
c     be performed over the global column numbers of the active block
c     and not the counter index.
c
c       do 9 jloop=jfirst,jlast
c
c bd. Increment the column index counter by one, and set the index
c     that contains the address in the vector "a( )" of the diagonal
c     term of the current column.
c
c       jj=jj+1
c       naj=na(jj)
c
c be. Set a dummy index that contains the address in the vector "a( )"
c     of the diagonal term of the previous column if the current
c     column was a complete column.
c
c       najj=naj-jloop
c
c bf. Find the global row number of the first non-zero term (i.e. the
c     top of the skyline) of the current column by taking the
c     difference between the address in the vector "a( )" of the
c     true top of the skyline and the dummy diagonal term of the
c     previous column if the current column was a complete column.
c
c       ifirst=jcoled-najj
c
c bg. Check to see if the row number of the first non-zero term (i.e.
c     the top of the skyline) of the current column of the active
c     block is less than the last column number of the current
c     subordinate block. If it is, some of the terms of the current
c     column in the active block will have to be partially reduced by
c     some of the terms of the active block and then by some of the
c     diagonal terms in the current subordinate block.
c
c       if (ifirst.le.lcolsc) then
c
c bh. Set the global row number of the last off-diagonal term of the
c     current column that has to be reduced in this section.
c
c       ilast=jloop-1
c
c bi. Set the address in "a( )" of the last off-diagonal of the
c     current column that will be used to reduce other terms of the
c     current column in this section.
c

```



```

                                klast=najj+lcolsc
c
c bj. Check to see if the current column is at least the second column
c   of the active block. If it is, there may be some partial
c   reduction of terms in the current column by terms of the active
c   block. If not, skip to the section that reduces terms in the
c   current column by the diagonal terms of the current subordinate
c   block.
c
                                if (jfirst.le.ilast) then
c
c bk. Initialize a column counter index, and set the address in the
c   vector "a( )" of the first non-zero term (i.e. the top of the
c   skyline) of the first column of the active block.
c
                                ii=0
                                naip=1
c
c bl. Loop over all the terms in the current column of the active
c   block that lie below the horizontal shadow of the closest
c   subordinate block to the active block.
c
                                do 7 iloop=jfirst,ilast
c
c bm. Increment the column counter index, and get the address in the
c   vector "a( )" of the diagonal term corresponding to the column
c   number given by "ii".
c
                                ii=ii+1
                                nai=na(ii)
c
c bn. Set a dummy index that contains the address in the vector "a( )"
c   of the diagonal term of the column previous to the one given by
c   the column counter index "ii" if the column given by the counter
c   "ii" was a complete column.
c
                                naii=nai-iloop
c
c bo. Find the global row number of the first non-zero term (i.e. the
c   top of the skyline) of the column given by the column counter
c   index "ii" by taking the difference between the address in the
c   vector "a( )" of the true top of the skyline and the dummy
c   diagonal term of the previous column if the column given by the
c   column counter index "ii" was a complete column.
c
                                kfirst=naip-naii
c
c bp. Check to see if the global row number of the first non-zero term
c   (i.e. the top of the skyline) of the column given by the index
c   counter "ii" is less than the global column number of the last
c   column of the current subordinate block. If it is, there will
c   be some partial reduction of terms in the current column of the

```

```

c   active block by terms in the column given by the index counter
c   "ii" of the active block before there is reduction by the
c   diagonal terms of the current subordinate block.
c
c           if (kfirst.le.lcolsc) then
c
c   bq. Set the address in the vector "a( )" of the term in the current
c   column that is being reduced where the reduction will start.
c   This is found by adding to the dummy top of the column if the
c   current column was a complete column the global row number of
c   term where the reduction will begin. The global row number of
c   term where the reduction will begin is given by the greater of:
c   1) the global row number of the first non-zero term of current
c   column, 2) the global row number of the first non-zero term of
c   column given by the column counter index "ii", 3) the global
c   column number of the first column of the current subordinate
c   block.
c
c           kfirst=najj+max0(ifirst,kfirst,jcp1)
c
c   br. Set an offset index in the vector "a( )" between the terms in
c   the current column and the terms in the column given by the
c   column counter index "ii", and initialize a summation variable.
c
c           jm=naii-najj
c           suma=0.0
c
c   bs. Loop over the correct terms in the column that is being reduced
c   and the terms in the column given by the column counter "ii" and
c   perform the summation.
c
c           do 6 kloop=kfirst,klast
c               suma=suma+a(jm+kloop)*a(kloop)
c   6           continue
c
c   bt. Set the address in the vector "a( )" of the term of the current
c   column that is being reduced, and then reduce that term.
c
c           jia=najj+iloop
c           a(jia)=a(jia)-suma
c
c   bu. End of the section to reduce terms in the current column by the
c   terms in the column given by the column counter index "ii".
c
c           endif
c
c   bv. Reset the address of the first non-zero term (i.e. the top of
c   the skyline) of the column given by the column counter index
c   "ii" to the next column of the active block and then loop back
c   up to reduce terms in the current column of the active block by
c   that next column of the active block.
c

```

```

                                naip=nai+1
7                                continue
c
c bw. End of the section that does the partial reduction of the terms
c   of the current column of the active block by terms in the active
c   block before the reduction by the diagonal terms of the current
c   subordinate block.
c
                                endif
c
c * * * * *
c
c Section to do the final reduction of the terms of the current column
c of the active block that lie in the horizontal shadow of the current
c subordinate block by the diagonal terms of the current subordinate
c block.
c
c * * * * *
c
c bx. Set the address in the vector "a( )" of the term in the current
c   column that is being reduced where the reduction will start.
c   This is found by adding to the dummy top of the column if the
c   current column was a complete column the global row number of
c   term where the reduction will begin. The global row number of
c   term where the reduction will begin is given by the greater of:
c   1) the global row number of the first non-zero term of the
c   current column, 2) the global column number of the first column
c   of the current subordinate block.
c
                                kfirst=najj+max0(ifirst,jcp1)
c
c by. Set an offset index in the vector "pivot" between the terms in
c   current column of the active block and the diagonal terms in the
c   current subordinate block, and initialize a summation variable.
c
                                jm=-(najj+lcolsp)
                                suma=0.0
c
c bz. Perform the summation needed to partially reduce the diagonal
c   term of the current column and perform the final reduction of
c   the off-diagonal terms of the current column by the diagonal
c   terms of the current subordinate block.
c
                                do 8 kloop=kfirst,klast
                                    div=a(kloop)/pivot(jm+kloop)
                                    suma=suma+div*a(kloop)
                                    a(kloop)=div
8                                continue
c
c ca. Reduce the diagonal term of the current column.
c
                                a(naj)=a(naj)-suma

```

```

c
c cb. End of the section to reduce the current column of the active
c   block.
c
c           endif
c
c cc. Reset the index that gives the address in the vector "a( )" of
c   the first non-zero term (i.e. the top of the skyline) of the
c   current column to the address of the top of the skyline of the
c   next column of the active block and then loop back up and reduce
c   the next column of the active block.
c
c           jcoled=naj+1
c   9         continue
c
c cd. If no reduction is needed for this subordinate block, do a
c   dummy read on the current record to skip to the next record.
c
c           else
c             read(5)
c
c ce. End of section that does the partial reduction of terms in the
c   active block by terms in the active block and by the diagonal
c   terms of the current subordinate block.
c
c           endif
c
c cf. Reset the global column number of the last column of the
c   previous subordinate block to the global column of the current
c   subordinate block and loop back up and do the reduction of terms
c   in the active block by the terms in the active block that lie in
c   the horizontal shadow of the next subordinate block and by the
c   diagonal terms of the next subordinate block.
c
c           lcolsp=lcolsc
c   10        continue
c
c cg. After completion of the reduction of the active block by all of
c   its subordinate blocks, do a dummy read on the the file of
c   diagonal addresses to set it at the correct record for reduction
c   of the next active block.
c
c           read(3)
c
c ch. End of the section to do all the reduction of the active block
c   by all of its subordinate blocks. This is also where the
c   program skips to if this is the first vector block to be
c   reduced.
c
c           endif
c
c=====

```

```

c
c Section to reduce the terms of the active stiffness vector block
c that are beneath the horizontal shadow of all its subordinate blocks
c by terms in the active stiffness vector block that also lie beneath
c the horizontal shadow of all the subordinate blocks of the active
c block. Also, this section does the final reduction of the diagonal
c terms of the active block.
c
c=====
c
c ci. Since the reduction of the terms in any particular column of
c the current stiffness vector block depends on terms in previous
c columns of the current block that have already been reduced,
c the first column of the current block is not reduced by any
c terms in the active block. Therefore, there needs to be more
c than one column in the current block for there to be reduction
c of terms in the active block by terms in the active block. So
c this condition is checked first.
c
    if (ncola.gt.1) then
c
c cj. If the above condition is true, set the global column number of
c the first column of the active block to be reduced by terms in
c the active block.
c
        jfirst1=jfirst+1
c
c ck. Since the first column of the active block will be skipped,
c initialize the counter over the number of columns of the current
c block to one, and initialize the index that contains the address
c in the vector "a( )" of the first non-zero term (i.e. the top of
c the skyline) of the column that is about to be reduced.
c
        jcolct=1
        jcoled=na(jcolct)+1
c
c cl. Loop over all the columns of the active block for reduction
c starting from the second column of the block. The index for
c the loop will be the global column number of the current
c column and not the counter index of the current column.
c
        do 14 jloop=jfirst1,jlast
c
c cm. Increment the column index counter by one, and set the index
c that contains the address in the vector "a( )" of the diagonal
c term for the current column being reduced.
c
            jcolct=jcolct+1
            naj=na(jcolct)
c
c cn. Set a dummy index that contains the address in the vector "a( )"
c of the diagonal term of the previous column if the current

```

```

c      column was a complete column.
c
c          najj=naj-jloop
c
c co. Find the global row number of the first non-zero term (i.e. the
c      top of the skyline) of the current column by taking the
c      difference between the address in the vector "a( ) " of the
c      true top of the skyline and the dummy diagonal term of the
c      previous column if the current column were a complete column.
c
c          ifirst=jcoled-najj
c
c cp. Check to see if the the global row number of the first non-zero
c      term in the current column is less than the global column
c      number of the current column (i.e. is there at least one
c      off-diagonal term in the current column?).
c
c          if (ifirst.lt.jloop) then
c
c      * * * * *
c
c      If the above condition is true, then all of the off-diagonal terms
c      of the current column, (except the term at the top of the skyline),
c      need to be partially reduced and the following section is executed.
c      If the above condition is not true, the section that reduces the
c      diagonal terms and does the final reduction of the off-diagonal
c      terms is executed.
c
c      * * * * *
c
c cq. Set the starting global row number in the current column where
c      the reduction of the off-diagonal terms will start. This will
c      be the greater of: 1) the second non-zero term of the current
c      column (since the first non-zero term of any column does not
c      undergo the partial reduction step), or 2) the first global
c      column number of the active block (since the partial reduction
c      of terms in the current column of the active block by terms in
c      columns previous to the active block has already been performed
c      in the above two sections of the program).
c
c          istart=max0(ifirst+1,jfrst1)
c
c cr. Set the global row number of the last off-diagonal term of the
c      current column that has to be reduced (this is the off-diagonal
c      term next to the diagonal term).
c
c          ilast=jloop-1
c
c cs. Check to make sure that there is at least one off-diagonal term
c      to be reduced in the current column.
c
c          if (istart.le.ilast) then

```

```

c
c ct. Find the column counter index of the already reduced column
c   that will be needed to reduce the first off-diagonal term of the
c   current column that will undergo partial reduction by terms in
c   the active block. Then get the address in the vector "a( )" of
c   the diagonal term of the column previous to the needed already
c   reduced column.
c
      ii=istart-lcolsp
      naip=na(ii-1)
c
c cu. Loop over all of the off-diagonal terms of the current column
c   that require partial reduction by terms in the active block.
c
      do 12 iloop=istart,ilast
c
c cv. Set the address in the vector "a( )" of the diagonal term of
c   the column of already reduced terms that are needed to partially
c   reduce the current column.
c
      nai=na(ii)
c
c cw. Set a dummy index that contains the address in the vector "a( )"
c   of the diagonal term of the previous column to the needed
c   already reduced column if the already reduced column was a
c   complete column.
c
      naii=nai-iloop
c
c cx. Find the global row number of the first non-zero term (i.e. the
c   top of the skyline) of the already reduced column by taking the
c   difference between the address in the vector "a( )" of the
c   true top of the skyline and the dummy diagonal term of the column
c   previous to the already reduced column if the already reduced
c   column was a complete column.
c
      kfirst=naip-naii
c
c cy. Check to see if the global row number of the first non-zero
c   term of the column of already reduced terms is less than the
c   global row number of the term being reduced. If it is, there
c   will be some partial reduction of the current term. If it is
c   not, there are no terms in the already reduced column that
c   affect the partial reduction of the current off-diagonal term
c   of the current column, and this section can be skipped.
c
      if (kfirst.lt.iloop) then
c
c cz. Set the address in the vector "a( )" of the current
c   off-diagonal term that is about to be partially reduced.
c
      jia=najj+iloop

```

```

c
c da. Set the addresses in the vector "a( )" of the first and last
c terms in the current column that will be needed to partially
c reduce the current term of that column. The first term that
c will be needed is the one with the global row number that is
c the greater of: 1) the first non-zero term of the current
c column (i.e. the top of the skyline), 2) the first non-zero
c term of the already reduced column (i.e. the top of the
c skyline) that is needed to partially reduce the current term of
c the current column, 3) the global column number of the first
c column of the active block. The last term that will be needed
c is the term in the current column just above the current term
c that is being partially reduced.
c
      kfirst=najj+max0(ifirst,kfirst,jfirst)
      klast=jia-1
c
c db. Set an address offset index that will give the offset in the
c vector "a( )" between the terms in the current column with the
c same global row numbers of terms in the already reduced column.
c
      jm=naii-najj
c
c dc. Initialize a summation variable.
c
      suma=0.0
c
c dd. Loop over the correct terms in the column that is being reduced
c and the terms in the already reduced column and perform the
c summation.
c
      do 11 kloop=kfirst,klast
         suma=suma+a(jm+kloop)*a(kloop)
11      continue
c
c de. Do the partial reduction of the current off-diagonal term of
c the current column.
c
      a(jia)=a(jia)-suma
c
c df. End of the section to partially reduce the current term of the
c current column of the active block.
c
      endif
c
c dg. Increment the index counter of the already reduced column by
c one so that the correct reduced column will be used for the
c partial reduction of the next off-diagonal term of the current
c column. Then set the address in "a( )" of the first non-zero
c term (i.e. the top of the skyline) of that already reduced
c column.
c

```



```

        ii=ii+1
        naip=nai+1
c
c dh. Loop back up and partially reduce the next off-diagonal term
c   of the current column.
c
c   12          continue
c
c di. End of the section to do the partial reduction of the
c   off-diagonal terms of the current column.
c
c       endif
c
c * * * * *
c
c This section does the final reduction of the off-diagonal terms and
c then reduces the diagonal term of the current column of the
c active block.
c
c * * * * *
c
c dj. Set global row number where the final reduction of the current
c   column will start. This will be the greater of: 1) the
c   global row number of the first non-zero term (i.e. the top of
c   the skyline) of the current column, or 2) the global column
c   number of the first column of the active block.
c
c       kfirst=max0(ifirst,jfirst)
c
c dk. Set the index counter to the column corresponding to the global
c   row number set above.
c
c       ii=kfirst-lcolsp
c
c dl. Change "kfirst" from the global row number of the first term
c   of the current column that requires final reduction to the
c   terms index in the vector "a( )". Then set the index of the
c   last off-diagonal term of the current column that requires
c   final reduction to the address of the off-diagonal term next
c   to the diagonal term of the current column.
c
c       kfirst=kfirst+najj
c       klast=najj-1
c
c dm. Initialize a summation variable.
c
c       suma=0.0
c
c dn. Loop over the rows of the current column from "kfirst" to
c   "klast". Divide the current partially reduced off-diagonal
c   term in row "kloop" by the fully reduced diagonal term for
c   that row. The term "div" will now contain the fully reduced

```

```

c      off-diagonal term in row "kloop" of the current column.
c      Multiply the above results by the partially reduced value in
c      row "kloop" of the current column and add the results to the
c      previous sum. Then replace the partially reduced term in row
c      "kloop" of the current column by its fully reduced value stored
c      in the term "div". Increment the index for obtaining the
c      address of the diagonal term so the next row can be considered.
c      Continue the loop over all the required rows of the current
c      column to reduce all the off-diagonal terms and obtain the
c      the summation term needed to reduce the diagonal term of the
c      current column.

```

```

c
c          do 13 kloop=kfirst,klast
c              nai=na(ii)
c              div=a(kloop)/a(nai)
c              suma=suma+div*a(kloop)
c              a(kloop)=div
c              ii=ii+1

```

```

13          continue

```

```

c
c do. Do the final reduction of the diagonal term.
c

```

```

c          a(naj)=a(naj)-suma

```

```

c
c dp. End of the section that does the final reduction of the current
c      column of the active block.
c

```

```

c          endif

```

```

c
c dq. Set the address in "a( )" of the first non-zero term (i.e. the
c      top of the skyline) of next column of the active block to be
c      reduced. Then loop back up and reduce the next column of the
c      active block.
c

```

```

c          jcoled=naj+1

```

```

14          continue

```

```

c
c dr. End of the section to reduce the terms in the active stiffness
c      vector block by terms in the active stiffness vector block that
c      lie below the horizontal shadow of all the subordinate blocks.
c

```

```

c          endif

```

```

c
c ds. Write to disk the reduced active stiffness vector block and
c      its associated reduced pivot block.
c

```

```

c          write(2) ireada,(a(iread),iread=1,ireada)
c          write(5) ireadn,(a(na(iread)),iread=1,ireadn)

```

```

c
c dt. Set the global column number of the first column of the next
c      stiffness vector block to be reduced and then loop back up and

```

```

c      reduce the next stiffness vector block.
c
c      jfirst=jlast+1
c
c      15 continue
c
c du. After all the stiffness vector blocks have been completely
c      reduced, return to main program.
c
c      return
c      end

```

```

      subroutine solve(a,pivot,b,na,ncol,nblok,neqn,dn,ac)

```

```

c=====
c
c Subroutine to reduce the load vectors and back substitute to find
c the d.o.f. displacements.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c January, 1987
c
c Based on:
c
c      Mondkar, D. P., and G. H. Powell, "Large Capacity Equation
c      Solver for Structural Analysis", Computers & Structures,
c      Vol 4, 1974, pp. 699-728.
c
c This subroutine is called by the main program to reduce the global
c load vector block stored in core and then do back substitution to
c find the d.o.f. displacements. It is called once for each load
c case.
c
c=====
c
c Variable names:
c
c      Variables brought in from main program:
c
c      nblok .... number of vector stiffness blocks
c      neqn ..... number of equations
c
c      Variables used only in this subroutine:
c
c      ieqn ..... index counter over the number of equations

```

```

c      iloop .... index counter over global row numbers
c      ip ..... number of terms in the vector space "pivot( )"
c      iread .... index counter of the terms going into the vector
c                spaces "a( )", b( )", or "pivot( )"
c      ireada ... number of terms in the vector space "a( )"
c      ireadn ... number of terms in the vector space "na( )"
c      istart ... global row number of first row of a loop over global
c                row numbers
c      jcolcb ... global column number of the first column of the
c                current subordinate vector stiffness block
c      jcolct ... index counter over the columns of the active vector
c                stiffness block
c      kfirst ... global row number where a summation will start
c      klast .... global row number where a summation will end
c      kloop .... index counter over the global row numbers of a
c                summation
c      lcolcb ... global column number of the last column of the
c                current subordinate vector stiffness block
c      lcolpb ... global column number of the last column of the
c                previous subordinate vector stiffness block
c      min ..... global d.o.f. number of the first non-zero load term
c      nai ..... address in the vector space "a( )" of the diagonal
c                term of the current column
c      naii ..... dummy index that contains the address in the vector
c                space "a( )" of the diagonal term of the column
c                previous to the current column if the current
c                column was a complete column
c      naip ..... address in the vector space "a( )" of the first
c                non-zero term (i.e. the top of the skyline) of the
c                current already reduced column
c      nans ..... counter of currently solved displacement
c      nblkbk ... dummy counter of vector stiffness blocks
c      nblok .... number of vector stiffness blocks
c      ncola .... number of columns in the vector stiffness block
c                stored in the vector space "a( )"
c      npblok ... index counter over the vector stiffness blocks
c      sumb ..... summation variable

```

```

c=====
c

```

```

c      Array names:
c

```

```

c      Arrays brought in from main program:
c

```

```

c      a(mtots) ..... vector space for the active block
c      ac(neqn) ..... vector space of d.o.f. load terms used by
c                the rest of the program
c      b(mtott) ..... vector space for the load terms and the
c                displacements
c      dn(neqn) ..... vector space for the displacements used by
c                the rest of the program
c      na(ncol(nblok)) ... vector space for the addresses of the

```

```

c                                diagonal terms of the active block
c      ncol(nblok) ..... number of columns in each vector stiffness
c                                block
c      pivot(mtotu) ..... vector space for the diagonal terms of the
c                                current subordinate block
c
c=====
c
c  Files and file names:
c
c    2 .. reduca.dmp:  file of reduced vector stiffness blocks
c
c    3 .. diagad.dmp:  file of addresses of the diagonal terms
c                      "reduca.dmp"
c
c    5 .. pivott.dmp:  file of diagonal terms of the reduced vector
c                      stiffness blocks
c
c=====
c
c a.  Set array sizes and set array and variable types.
c
c      real*8 a(8000),pivot(8000),b(5000),sumb
c
c      dimension na(1),ncol(1),dn(1),ac(1)
c
c b.  Put the d.o.f. load terms in the vector space "b( )" and change
c      all the load terms from single precision to double precision.
c
c      do 1 ieqn=1,neqn
c        b(ieqn)=dble(ac(ieqn))
c      1 continue
c
c c.  Find the global row number of the first non-zero load term.  The
c      result will be stored in the variable "min".
c
c      do 2 ieqn=1,neqn
c        if (b(ieqn).ne.0.0) then
c          min=ieqn
c          goto 3
c        endif
c      2 continue
c      3 continue
c
c=====
c
c Section to partially reduce the load vector by the off-diagonal
c terms of the reduced vector stiffness blocks.
c
c=====
c
c d.  Set all needed files to their first record.

```

```

c
c      rewind 2
c      rewind 3

c
c e. Initialize the index of the last column number of the previous
c      reduced vector stiffness block to zero since no reduced vector
c      stiffness blocks have been brought in yet.
c
c      lcolpb=0
c
c f. Loop over all the reduced stiffness vector blocks. The reduced
c      vector stiffness block that is in the core at any time will be
c      referred to as the "active block".
c
c      do 6 npblok=1,nblok
c
c g. Set the number of columns in the active block.
c
c      ncola=ncol(npblok)
c
c h. Set the global column number of the last column of the active
c      block.
c
c      lcolcb=lcolpb+ncola
c
c i. Check to see if the global row number of the first non-zero load
c      term is less than the global column number of the last column of
c      the active block. If it is, some of the terms of the load vector
c      will have to be reduced by some of the terms of the active block.
c
c      if(min.lt.lcolcb) then
c
c j. If the active block is needed to reduce some of the terms in the
c      load vector, first read the active block and its diagonal address
c      vector.
c
c      read(2) ireada,(a(iread),iread=1,ireada)
c      read(3) ireadn,(na(iread),iread=1,ireadn)
c
c k. Set the global column number of the first column of the active
c      block.
c
c      jcolcb=lcolpb+1
c
c l. Set the starting global row number of the active block where the
c      partial reduction of the load vector will start. This will be
c      the greater of: 1) the global row number of the second non-zero
c      load term (since the first non-zero load term does not undergo
c      the partial reduction step), or 2) the first global column number
c      of the active block.
c

```

```

        1start=max0(min+1,jcolcb)
c
c m. Set the column counter index to the first column of the active
c   block that will be needed to partially reduce some of the terms
c   in the load vector.
c
        jcolct=1start-1colpb
c
c n. Set the address in the vector "a( )" of the first non-zero term
c   (i.e the top of the skyline) of the column given by the column
c   counter index "jcolct".
c
        if(jcolct.eq.1) then
            naip=1
        else
            naip=na(jcolct-1)+1
        endif
c
c o. Loop over all of the columns of the active block that will be
c   used to partially reduce some of the terms of the load vector.
c
        do 5 iloop=1start,1colcb
c
c p. Set the index that contains the address in the vector "a( )" of
c   the diagonal term for the current column being used to partially
c   reduce some of the load terms.
c
            nai=na(jcolct)
c
c q. Set a dummy index that contains the address in the vector "a( )"
c   of the diagonal term of the previous column if the current
c   column was a complete column.
c
            naii=nai-iloop
c
c r. Find the global row number of the first non-zero term (i.e. the
c   top of the skyline) of the current column of the active block by
c   taking the difference between the address in the vector "a( )"
c   of the true top of the skyline and the dummy diagonal term of
c   the previous column if the current column was a complete column.
c
            kfirst=naip-naii
c
c s. Check to see if the global row number of the first non-zero term
c   of the current column is less than the global column number of
c   the current column (i.e. is there at least one off-diagonal term
c   in the current column). If there is, there will be some partial
c   reduction of some of the load terms by the terms in the current
c   column.
c
            if (kfirst.lt.iloop) then

```

```

c t. Set the global row number of the first term where the reduction
c   summation will start. This will be the greater of: 1) the global
c   row number of the first non-zero load term, 2) the global row
c   number of the first non-zero term of the current column.
c
c       kfirst=max0(min,kfirst)
c
c u. Set the global row number of the last term of the current column
c   that will be used to partially reduce the current load term.
c   This will be the first off-diagonal term of the current column.
c
c       klast=iloop-1
c
c v. Initialize a summation variable.
c
c       sumb=0.0
c
c w. Loop over the correct terms in the current column of the active
c   block and the terms in the load vector and perform the summation.
c   The term "nail" is used as an offset address in the vector
c   "a( )".
c
c       do 4 kloop=kfirst,klast
c           sumb=sumb+a(nail+kloop)*b(kloop)
4       continue
c
c x. Subtract the summation calculated above to partially reduce the
c   current load term.
c
c       b(iloop)=b(iloop)-sumb
c
c y. End of the section to partially reduce the current load term.
c
c       endif
c
c z. Increment the index counter by one so that the correct reduced
c   column will be used for the partial reduction of the next load
c   term. Then set the address in "a( )" of the first non-zero term
c   (i.e. the top of the skyline) of that already reduced column.
c
c       jcolct=jcolct+1
c       nail=nail+1
c
c aa. Loop back up and partially reduce the next load term that
c   depends on partial reduction by terms in the active block.
c
c       5   continue
c
c ab. If there will be no partial reduction of any load term by the
c   active block, do a dummy read on the appropriate files to
c   advance them to the next record.
c

```



```

        else
            read(2)
            read(3)
c
c ac. End of the section to partially reduce some of the load terms by
c     the active block.
c
        endif
c
c ad. Set the index of the last global column number of the previous
c     active block equal to the last global column number of the
c     active block.
c
        lcolpb=lcolcb
c
c ae. Loop back up and do the partial reduction of some of the load
c     terms by the terms of the next vector stiffness block.
c
        6 continue
c
c=====
c
c Section to do the final reduction of the load terms by the diagonal
c terms of the reduced stiffness vector blocks.
c
c=====
c
c af. Set the needed file to its first record.
c
        rewind 5
c
c ag. Reinitialize the index of the last column number of the previous
c     reduced vector stiffness block to zero since no reduced vector
c     stiffness blocks have been brought in yet for this section.
c
        lcolpb=0
c
c ah. Loop over all the reduced stiffness vector blocks.
c
        do 8 npblok=1,nblok
c
c ai. Set the number of columns in the active block.
c
            ncola=ncol(npblok)
c
c aj. Set the global column number of the last column of the active
c     block.
c
            lcolcb=lcolpb+ncola
c
c ak. Check to see if the global row number of the first non-zero load
c     term is less than the global column number of the last column of

```

```

c   the active block.  If it is, some of the terms of the load
c   vector will have to be reduced by the diagonal terms of the
c   active block.
c
c       if (min.le.lcolcb) then
c
c   al. If the active block is needed to reduce some of the terms in the
c       load vector, first read the diagonal terms of the active block.
c
c       read(5) ip,(pivot(iread),iread=1,ip)
c
c   am. Set the global column number of the first column of the active
c       block.
c
c       jcolcb=lcolpb+1
c
c   an. Set the starting global row number of the active block where the
c       reduction of the load vector by the diagonal terms of the active
c       block will start.  This will be the greater of: 1) the global
c       row number of the first non-zero load term, or 2) the first
c       global column number of the active block.
c
c       istart=max0(min,jcolcb)
c
c   ao. Change the starting location of the final reduction from the
c       global column number of the active block to the local position
c       of that column in the active block.
c
c       istart=istart-lcolpb
c
c   ap. Loop over all the load terms that require the final reduction by
c       the diagonal terms of the active block.
c
c       do 7 iloop=istart,ncola
c           b(lcolpb+iloop)=b(lcolpb+iloop)/pivot(iloop)
c       7   continue
c
c   aq. If there will be no final reduction of any load term by the
c       diagonal terms of the active block, do a dummy read on the
c       appropriate file to advance it to the next record.
c
c       else
c           read(5)
c
c   ar. End of the section to do the final reduction of some of the load
c       terms by the diagonal terms of the active block.
c
c       endif
c
c   as. Set the index of the last global column number of the previous
c       active block equal to the last global column number of the
c       active block.

```

```

c
c      lcolpb=lcolcb
c
c at. Loop back up and do the final reduction of some of the load
c      terms by the diagonal terms of the next vector stiffness block.
c
c      8 continue
c
c=====
c
c This section does the back substitution to find the d.o.f.
c displacements. The displacements will replace the load terms in the
c vector space "b( )".
c
c=====
c
c au. Set scratch variables equal to the number of d.o.f.s and the
c      number of reduced vector stiffness blocks.
c
c      nans=neqn
c      nblkbk=nblok
c
c av. Loop over the total number of blocks, this loop will actually
c      be performed in reverse order.
c
c      do 12 npblok=1,nblok
c
c aw. Read the active reduced stiffness vector block and its
c      associated diagonal address vector block. Since the files are
c      at their last record after completion of the above two sections,
c      the reads will be performed from the end of the file backward.
c
c      backspace 2
c      read(2) ireada,(a(iread),iread=1,ireada)
c      backspace 2
c
c      backspace 3
c      read(3) ireadn,(na(iread),iread=1,ireadn)
c      backspace 3
c
c ax. Set the number of columns in the active block, and set a dummy
c      column counter index equal to the number of columns of the
c      active block.
c
c      ncola=ncol(nblkbk)
c      jcolct=ncola
c
c ay. Set the address of the last diagonal term in the active block.
c
c      nai=na(jcolct)
c
c az. Check to see if there is more than one column in the active

```

```

c      block.
c
c          if (ncola.ne.1) then
c
c      * * * * *
c
c      Section to do the back substitution by all the columns of the active
c      block except the first column.
c
c      * * * * *
c
c      ba. Loop over all of the columns of the active block except the
c          first one.
c
c          do 10 iloop=2,ncola
c
c      bb. Set the address in the vector "a( )" of the diagonal term of the
c          column previous to the column given by column counter index
c          "jcolct".
c
c          naip=na(jcolct-1)
c
c      bc. Set a dummy index that contains the address in the vector "a( )"
c          of the diagonal term of the previous column if the current
c          column was a complete column.
c
c          naii=nai-nans
c
c      bd. Find the global row number of the first non-zero term (i.e. the
c          top of the skyline) of the current column of the active block by
c          taking the difference between the address in the vector "a( )"
c          of the true diagonal term of the previous column and the dummy
c          diagonal term of the previous column if the current column was
c          a complete column and then adding one.
c
c          kfirst=naip-naii+1
c
c      be. Check to see if the global row number of the first non-zero term
c          of the current column is less than the global column number of
c          the current column (i.e. is there at least one off-diagonal term
c          in the current column). If there is, these off-diagonal terms
c          will be used to partially back substitute the current
c          displacement term into the unsolved displacement terms.
c
c          if (kfirst.lt.nans) then
c
c      bf. Set the global row number of the last term of the current column
c          that will be used to partially back substitute the unsolved
c          displacement terms. This will be the first off-diagonal term of
c          the current column.
c
c          klast=nans-1

```

```

c
c bg. Loop over the correct terms in the current column of the active
c   block and perform the partial back-substitution of the unsolved
c   displacements with the current solved displacement. The term
c   "nail" is used as an offset address in the vector "a( )".
c
c       do 9 kloop=kfirst,klast
c           b(kloop)=b(kloop)-a(nail+kloop)*b(nans)
c   9       continue
c
c bh. End of the section to do the back substitution by the current
c   solved displacement. At this point the displacement with a
c   global d.o.f. number one less than the one used to do the
c   back substitution has now been found.
c
c       endif
c
c bi. Decrement the column counter index for the back substitution
c   by the displacement just found. Decrement the index of the
c   global d.o.f. number so that the newly solved displacement
c   can be back substituted into the unsolved displacements. Set
c   the address of the current column of the active block to the
c   address of the previous column of the active block.
c
c       jcolct=jcolct-1
c       nans=nans-1
c       nai=naip
c
c bj. Loop back up and back substitute the newly solved displacement
c   into the unsolved displacements.
c
c   10    continue
c
c bk. End of the section to the back substitution with all of the
c   columns of the active block except the first column.
c
c       endif
c
c *****
c
c Section to do the back substitution by the first column of the
c active block.
c
c *****
c
c bl. Check to see if this the first reduced global vector stiffness
c   block. If it is, this section is skipped.
c
c       if (nblkbk.ne.1) then
c
c   bm. Set a dummy index that contains the address in the vector "a( )"
c       of the diagonal term of the previous column if the first column

```

```

c      was a complete column.
c
c      nani=nai-nans
c
c bn. Find the global row number of the first non-zero term (i.e. the
c      top of the skyline) of the first column of the active block by
c      adding one to the negative of the dummy address in the vector
c      "a( )" of the column previous to the first column.
c
c      kfirst=-nani+1
c
c bo. Check to see if the global row number of the first non-zero term
c      of the first column is less than the global column number of
c      the first column (i.e. is there at least one off-diagonal term
c      in the first column). If there is, these off-diagonal terms
c      will be used to partially back-substitute the current
c      displacement term in to the unsolved displacement terms.
c
c      if (kfirst.lt.nans) then
c
c bp. Set the global row number of the last term of the first column
c      that will be used to partially back-substitute the unsolved
c      displacement terms. This will be the first off-diagonal term of
c      the first column.
c
c      klast=nans-1
c
c bq. Loop over the correct terms in the first column of the active
c      block and perform the partial back-substitution of the unsolved
c      displacements with the current solved displacement. The term
c      "nani" is used as an offset address in the vector "a( )".
c
c      do 11 kloop=kfirst,klast
c          b(kloop)=b(kloop)-a(nani+kloop)*b(nans)
11      continue
c
c br. End of the section to do the back substitution by the current
c      solved displacement. At this point the displacement with global
c      d.o.f. number one less than the one used to do the
c      back-substitution has now been found.
c
c      endif
c
c bs. Decrement the global d.o.f. number and the global block number.
c
c      nans=nans-1
c      nblkbk=nblkbk-1
c
c bt. End of the section to do the back substitution by the terms in
c      the first column of the active block.
c
c      endif

```

```

c
c bu. Loop back up and do the back substitution by the next reduced
c   vector stiffness block.
c
c   12 continue
c
c bv. Change displacements to single precision and store in
c   locations used by the rest of the program.
c
c   do 13 ieqn=1,neqn
c     dn(ieqn)=sngl(b(ieqn))
c   13 continue
c
c bw. Return to main program.
c
c   return
c   end

```

```

      subroutine renum(lc,nodcon,nodlis,nodnew,nodold,nodren,numet,
&ioreord,nelem,nnode,mnodee,maxall,ibanwd,minmax)

```

```

c=====
c
c Subroutine to renumber the nodes to reduce the bandwidth.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c Based on:
c
c   Collins, R. J. "Bandwidth Reduction by Automatic Renumbering."
c   International Journal for Numerical Methods in Engineering,
c   Vol 6, 1973, pp. 345-356.
c
c This subroutine is used by two different programs for use in
c optimization. It is used by the frontal solver to reorder the node
c numbers before the order of element calls is determined. In that
c case it is called by subroutine "ordred". It is also used by the
c blocked out-of-core solver to reorder the node numbers to minimize
c the bandwidth. In that case it is called by subroutine
c "banred". In both cases this subroutine is called only once.
c
c=====
c
c Variable names:
c

```

```

c   Variables brought in from calling subroutine:
c
c   ibanwd ... bandwidth of current numbering scheme
c   ibanw2 ... bandwidth of attempted numbering scheme
c   ireord ... flag to indicate to stop reordering process
c   maxall ... maximum allowable number of nodes that can be
c             connected to a particular node
c   minmax ... smallest bandwidth of any renumbering scheme
c   mnodee ... maximum number of nodes per element
c   nelem ... number of elements
c   nnode ... number of nodes
c
c   Variables used only in this subroutine:
c
c   inode .... index counter over all the nodes
c   itest .... temporary test of current bandwidth
c   jcurrt ... node number of node under consideration
c   jelem .... index counter over element numbers
c   jnode .... index counter over all the nodes
c   jpoint ... offset pointer for use in the vector of related
c             node numbers
c   jrel ..... index counter over the nodes related to the current
c             node
c   jrelno ... node number of current related node
c   knode .... increment index for new node numbers
c   list ..... index counter over the node numbers of the nodes
c             that are related to the current node
c   lnode .... index over node numbers
c   max ..... bandwidth based on the two nodes being examined
c             at the current time
c   nodrel ... node number of current related node
c   numlis ... number of nodes related to current node
c   numnod ... number of nodes per element
c   numrel ... number of nodes related to current node
c
c=====
c
c   Array names:
c
c   Arrays brought in from calling subroutine:
c
c   lc(mnodee,nelem) ..... list of element connectivity
c   nodcon(nnode) ..... vector list of number of connected
c                       nodes to current node
c   nodlis(nnode*maxall) ... vector list of node numbers of nodes
c                       connected to the current node
c   nodnew(nnode) ..... vector list of new node numbers for
c                       current renumbering scheme
c   nodold(nnode) ..... vector list of old node numbers for
c                       current renumberig scheme
c   nodren(nnode) ..... vector list of renumbered node
c                       numbers

```



```

c      numet(nelem) ..... vector list of element types
c
c=====
c
c      Element types:
c
c      1  Truss:  2 nodes
c              3 d.o.f. per node
c
c      2  Beam:   2 nodes
c              6 d.o.f. per node
c
c      3  Plate:  4 nodes
c              6 d.o.f. per node
c
c=====

      dimension lc(mnodee,1),nodcon(1),nodlis(1),nodnew(1),nodold(1),
      &nodren(1),numet(1)

c=====
c
c      Section to determine nodal connectivity.
c
c=====
c
c      a. Do essential initialization.
c
c          ibanwd=0
c          do 1 inode=1,nnode
c              nodcon(inode)=0
c          1 continue
c
c      b. Loop over all the elements.
c
c          do 5 jelem=1,nelem
c
c      c. Set the number of nodes for the current element.  Truss (type
c      one) and beam (type two) elements have two nodes per element.
c      Plates (type three) have four nodes per element.
c
c          numnod=2
c          if (numet(jelem).eq.3) numnod=4
c
c      d. Loop over all the nodes of the current element.
c
c          do 4 inode=1,numnod
c
c      e. Get the node number of the node that is under consideration.
c
c          jcurrt=lc(inode,jelem)
c

```

```

c f. Set pointer offset used for start address in the vector of nodal
c   conectivity ("nodlis"), where the node numbers of the nodes that
c   are connected to the current node under consideration will be
c   stored.
c
      jpoint=(jcurrt-1)*maxall
c
c g. Loop over all the nodes of the current element again to
c   determine which nodes are connected to the node under
c   consideration through the current element.
c
      do 3 jnode=1,numnod
c
c h. Check to make sure that the connectivity of the node under
c   consideration with itself is not considered.
c
        if (jnode.ne.inode) then
c
c i. Identify a related node to the one under consideration.
c
          jrelno=lc(jnode,jelem)
c
c j. Ascertain whether a relationship between the node under
c   consideration and the related node has already been
c   established. If it has, go to the appropriate section.
c
          numlis=nodcon(jcurrt)
          if (numlis.ne.0) then
            do 2 list=1,numlis
              if (nodlis(jpoint+list).eq.jrelno) goto 3
            2      continue
          endif
c
c k. If a relationship does not exist, add the related node to the
c   list of nodes connected to the current node under consideration.
c   Also check to see if the maximum allowable connectivy has been
c   exceeded. If it has, set a flag to indicate that no reordering
c   is to be performed and return to the main program.
c
          nodcon(jcurrt)=nodcon(jcurrt)+1
          if (nodcon(jcurrt).gt.maxall) then
            ireord=1
            return
          endif
          nodlis(jpoint+nodcon(jcurrt))=jrelno
c
c l. Check to see if the current nodal relationship increases the
c   bandwidth. If it does, set the new bandwidth.
c
          itest=iabs(jcurrt-jrelno)
          if (itest.gt.ibanwd) ibanwd=itest
c

```

```

c m. End of section to do necessary housekeeping for the current node
c   under consideration and the current other node of the current
c   element.
c
c       endif
c
c n. Loop back and do the necessary housekeeping on the next node of
c   the current element as attached to the current node under
c   consideration.
c
c       3      continue
c
c o. Loop back and consider the next node of the current element.
c
c       4      continue
c
c p. Loop back and consider the next element.
c
c       5      continue

c=====
c
c Section to reorder the node numbers.
c
c=====
c
c q. Initialize the minimum bandwidth to the current bandwidth and
c   initialize the renumbered nodes to their present number.
c
c       minmax=ibanwd
c       ibanw2=ibanwd
c       do 6 inode=1,nnode
c           nodren(inode)=inode
c       6 continue
c
c r. Loop over all the nodes. The origin of the new numbering system
c   is located at each node in turn so that the number of different
c   renumbering schemes (actual or attempted) is equal to the number
c   of nodes. The new numbering scheme starts at the node whose old
c   number is "inode".
c
c       do 11 inode=1,nnode
c
c s. As the new numbering scheme progresses, two arrays are set up
c   which are largely used as pointers to eliminate the need to
c   search through "nodlis". The first, "nodnew", contains the new
c   node numbers, with their locations in the array being equal to
c   the old node numbers. The second, "nodold", contains the old
c   node numbers, with their locations in the array being equal to
c   the new node numbers. For each new numbering scheme, they are
c   both initialized to zero.
c

```

```

        do 7 jnode=1,nnode
            nodnew(jnode)=0
            nodold(jnode)=0
7       continue
c
c t. Initialize for a new node numbering scheme.
c
        max=0
        nodold(1)=inode
        nodnew(inode)=1
        knode=1
c
c u. Loop over all the nodes for renumbering.
c
        do 9 jnode=1,nnode
c
c v. Set the number of related nodes to the node under consideration
c    and set the offset pointer for the vector of nodal connectivity.
c
            numrel=nodcon(nodold(jnode))
            jpoint=(nodold(jnode)-1)*maxall
c
c w. Loop over all the related nodes of the node under consideration.
c
            do 8 jrel=1,numrel
c
c x. Set the current related node number.
c
                nodrel=nodlis(jpoint+jrel)
c
c y. Check to see if the related node has already been renumbered.
c
                if (nodnew(nodrel).eq.0) then
c
c z. If the node has not been renumbered, give it the next possible
c    node number.
c
                    knode=knode+1
                    nodold(knode)=nodrel
                    nodnew(nodrel)=knode
c
c aa. Calculate the new bandwidth based on the renumbered node under
c    consideration and its current renumbered related node.
c
                ibanw2=iabs(jnode-knode)
c
c ab. If the current bandwidth is equal to or greater than the
c    current minimum bandwidth, abandon the current scheme and try
c    the next.
c
                if (ibanw2.ge.minmax) goto 11
c

```

```

c ac. If the current bandwidth is greater than the largest bandwidth
c encountered so far for the current scheme, reset the value of
c the largest bandwidth for the current scheme.
c
c         if (ibanw2.gt.max) max=ibanw2
c
c ad. End of section that rennumbers an attached node to the current
c node under consideration.
c
c         endif
c
c ae. Loop back and do the next attached node to the current node
c under consideration.
c
c         8 continue
c
c af. Loop back and consider the next base node.
c
c         9 continue
c
c ag. At this point a completely new numbering scheme has been
c completed which is superior to any previous numbering schemes.
c The bandwidth is kept for comparision with subsequent
c numbering schemes. The new numbering scheme is then saved
c in the vector "nodren".
c
c         minmax=max
c         do 10 lnode=1,nnode
c             nodren(lnode)=nodnew(lnode)
c         10 continue
c
c ah. Loop back and try next numbering scheme.
c
c         11 continue
c
c ai. After all schemes have been tried, return to the main program
c with the optimum node numbering scheme stored in the vector
c "nodren".
c
c         return
c         end

```

APPENDIX D

PROGRAM LISTINGS FOR THE FRONTAL SOLVER

```

      subroutine prefrr(idn,ndofn,ispr,spr,ide,msprg,sprf,iorder,
&ide2,nsprct)

```

```

C=====
C
C Subroutine to do preliminary housekeeping before frontal solution.
C
C=====
C
C coded by: Christopher D. Pionke
C coded for: SST Systems Inc.
C
C June, 1986
C
C This subroutine is called by the main program to perform some
C preliminary chores that are necessary to integrate the original
C main program to the frontal solver coded in subroutine "front".
C It is called only once and is used whether or not the front
C reduction technique is employed.
C
C=====
C
C Variable names:
C
C Variables brought in from main program:
C
C   mdofe .... maximum number of d.o.f.s per element
C   mdofn .... maximum number of d.o.f.s per node
C   nelelem .... number of elements
C   ndof ..... total number of d.o.f.s
C   nnode .... total number of nodes
C   nspr ..... number of nodes with support springs
C   nsprct ... number of d.o.f.s with non-zero spring stiffnesses
C
C Variables used only in this subroutine:
C
C   idof ..... index over all the d.o.f.s
C   idofe .... index counter over element d.o.f.s
C   ielem .... index counter over element numbers
C   isprg .... index counter over spring numbers
C   jdofn .... index counter over d.o.f.s per node
C   jelem .... index counter over element numbers
C   kdofe .... index counter over element d.o.f.s
C   klast .... index number of the last appearance of a d.o.f.
C   kspr ..... node number index of node with spring
C   lelem .... index of element number
C   nudofn ... index counter over number of d.o.f.s per node
C
C=====
C
C Array names:
C

```

```

c   Arrays brought in from main program:
c
c   ide(mdofe,nelem) .. element d.o.f. vectors
c   ide2(mdofe*nelem) .. working copy of element d.o.f. vectors
c   idn(mdofn,nnode) .. nodal d.o.f. vectors
c   ispr(nspr) ..... spring node numbers
c   iorder(nelem) ..... vector list of indexes for the vector of
c                       element numbers in the order that the
c                       elements will be brought into the front
c   msprg(nsprct) ..... list of d.o.f.s with non-zero spring
c                       stiffnesses
c   ndofn(nnode) ..... vector of number of d.o.f.s per node
c   spr(mdofn,nspr) ... array of spring stiffnesses on a nodal
c                       basis
c   sprf(nsprgct) ..... vector of d.o.f. spring stiffnesses on a
c                       d.o.f. basis
c
c=====

      common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
      &nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
      &ndof,nres,neqn,mdofn,mnodee,mdofe,mprop,mgeom,moffe,iai,iao

      dimension idn(mdofn,1),ndofn(1),ispr(1),spr(mdofn,1),
      &ide(mdofe,1),msprg(1),sprf(1),iorder(1),ide2(1)

c=====
c
c Section to transfer spring stiffnesses from the array "spr( , )"
c to the vector "sprf( )". This puts the spring stiffnesses on a
c d.o.f. basis instead of a nodal basis.
c
c=====
c
c a. Check first to see if there are any springs for this problem.
c
c   if (nspr.gt.0) then
c
c b. If there are springs, initialize the number of d.o.f.s with
c    springs to zero.
c
c   nsprct=0
c
c c. Loop over the total number of nodes with springs.
c
c   do 2 isprg=1,nspr
c
c d. Set "kspr" to the node number of the current spring number.
c
c   kspr=ispr(isprg)
c
c e. Set "nudofn" to the number of d.o.f.s of the current node.

```



```

c
      nudofn=ndofn(kspr)
c
c f. Loop over the number of d.o.f.s of the current node.
c
      do 1 jdofn=1,nudofn
c
c g. Check to see if the spring stiffness for the current d.o.f. is
c    a non-zero quantity.
c
      if (spr(jdofn,isprg).ne.0.0) then
c
c h. If the spring stiffness is non-zero, increment the counter of
c    the number of d.o.f.s with spring stiffnesses.
c
      nsprct=nsprct+1
c
c i. Transfer the spring stiffness in the array "spr(jndof,isprg)"
c    to the spring stiffness vector "sprf(nsprct)".
c
      sprf(nsprct)=spr(jdofn,isprg)
c
c j. Put the global d.o.f. number of the newly stored spring
c    stiffness in "msprg(nsprct)".
c
      msprg(nsprct)=idn(jdofn,kspr)
c
c k. End of non-zero spring stiffness section.
c
      endif
c
c l. Continue the loop over the number of d.o.f.s per node.
c
      1      continue
c
c m. Continue the loop over the total number of springs.
c
      2      continue
c
c n. End of spring stiffness transfer "if" condition.
c
      endif

c=====
c
c Change the sign to negative of the last appearance of each
c d.o.f. number in the element degree of freedom array "ide( , )".
c The negative sign will indicate when a d.o.f. can be eliminated
c from the front in the frontal solver. To speed up the process and
c not interfere with the arrays of the main program, a working copy
c of the the array "ide( , )" will be made in vector form.
c

```

```

c=====
c
c o. First make a working copy of the element d.o.f. array in vector
c   form.
c
c       kdofe=0
c
c       do 4 ielem=1,nelem
c         do 3 idofe=1,mdofe
c           kdofe=kdofe+1
c           ide2(kdofe)=ide(idofe,ielem)
c       3 continue
c       4 continue
c
c p. Loop over all the global d.o.f. numbers.
c
c       do 7 idof=1,ndof
c
c q. Set flag for last appearance of the current d.o.f. number
c   to the zero position.
c
c       klast=0
c
c r. Loop over all the elements.
c
c       do 6 jelem=1,nelem
c
c s. Get the element index number from the vector "iorder( )".
c
c       ielem=iorder(jelem)
c
c t. Set index of "ide2( )" at the beginning of the d.o.f. numbers
c   for the current element.
c
c       kdofe=(ielem-1)*mdofe
c
c u. Loop over the maximum number of d.o.f.s per element.
c
c       do 5 idofe=1,mdofe
c
c v. Set index to the current d.o.f. number of the current element.
c
c       kdofe=kdofe+1
c
c w. Check to see if the current d.o.f. number of the current element
c   is equal to the d.o.f. number of the outer loop.
c
c       if (ide2(kdofe).eq.idof) then
c
c x. If the above condition is true, set "klast" to the current index
c   number of the vector "ide2( , )".
c
c

```

```

        klast=kdofe
    endif
c
c y. Continue the loop over the element d.o.f.s.
c
c     5      continue
c
c z. Continue loop over all the elements.
c
c     6      continue
c
c aa. If the flag for the last appearance of a global d.o.f. number
c     has changed from the zero location, change the sign of the
c     element d.o.f. number stored at the "klast" position of
c     "ide( , )" to negative.
c
c         if (klast.ne.0) ide2(klast)=-ide2(klast)
c
c ab. Continue the loop over all the global d.o.f. numbers.
c
c     7      continue
c
c ac. Return to the main program.
c
c         return
c         end

```

```

subroutine ordnre(ume,iorder)

```

```

c=====
c
c Subroutine to determine the element order calls for the frontal
c solution for the non-frontwidth reduction case.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c This subroutine is called by the main program to determine the
c order of the indexes for the vector of element numbers "ume( )",
c that will result in the elements being called in ascending
c numerical order. The order of the indexes is stored in vector
c "iorder( )". It is only called once.
c
c=====
c
c Variable names:

```

```

c
c   Variables brought in from main program:
c
c       nelem ..... number of elements
c
c   Variables used only in this subroutine:
c
c       idum ..... temporary storage
c       ielem .... index counter over element numbers
c       jelem .... index counter over element numbers
c       lelem .... index of element number
c
c=====
c
c   Array names:
c
c   Arrays brought in from main program:
c
c       iorder(nelem) ..... vector list of the indexes for "nume( )"
c                               that will list the element numbers in
c                               ascending order
c       nume(nelem) ..... vector list of element numbers
c
c=====

      common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
&nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
&ndof,nres,neqn,mdofn,mnodee,mdofo,mprop,mgeom,moffe,iai,iao

      dimension nume(1),iorder(1)
c
c a. Initialize an element index.
c
c       lelem=1
c
c b. Loop over all the elements.
c
c       do 3 ielem=1,nelem
c
c c. Check sign of "nume(ielem)" to see if it is negative.
c
c       1  if (nume(ielem).lt.0) then
c
c d. If "nume(ielem)" is negative, increment "lelem" and
c     check the next value of "nume(ielem)".
c
c         lelem=lelem+1
c         goto 1
c       endif
c
c e. Set "idum" to the element number stored in "nume(lelem)".
c

```

```

        idum=nume(1elem)
c
c f. Loop over all the elements again.
c
        do 2 jelem=1,nelem
c
c g. Check the element number stored in "nume(jelem)" to see if it is
c   less than the element number stored in "idum" and greater than
c   zero.
c
        if (nume(jelem).le.idum.and.nume(jelem).gt.0) then
c
c h. If the above condition is true, put the index of "nume(jelem)"
c   in "iorder(ielem)" and put the element number stored in
c   "nume(jelem)" in "idum".
c
            iorder(ielem)=jelem
            idum=nume(jelem)
        endif
c
c i. Continue the inner loop over all the elements.
c
        2 continue
c
c j. Set the element number stored in "nume(iorder(ielem))" to
c   negative to indicate that its index has been stored in
c   "iorder(ielem)".
c
            nume(iorder(ielem))=-nume(iorder(ielem))
c
c k. Continue the outer loop over all the elements.
c
        3 continue
c
c *****
c
c l. Change the sign of all the values of "nume( )" used above back
c   to positive.
c
        do 4 ielem=1,nelem
            nume(ielem)=iabs(nume(ielem))
        4 continue
c
c m. Return to the main program.
c
        return
        end

        subroutine ordred(nume,numet,lc,iorder,itrip,nodcon,nodlis,
&nodnew,nodold,nodren,maxall,ibanwd,maxmin)

```

```

C=====
C
C Subroutine to do frontwidth reduction.
C
C=====
C
C coded by: Christopher D. Pionke
C coded for: SST Systems Inc.
C
C June, 1986
C
C Based on:
C
C   Sloan, S. W., and M. F. Randolph, "Automatic Element Reordering
C   for Finite Element Analysis with Frontal Solution Schemes",
C   International Journal for Numerical Methods in Engineering,
C   Vol 19, 1983, pp. 1153-1181.
C
C This subroutine is used to determine the order of element calls
C for the wavefront solver that will result in a reduction of the
C maximum and average wavefront.
C
C=====
C
C Variable names:
C
C   Variables brought in from main program:
C
C   maxall ... maximum allowable number of nodes that can be
C             attached to a particular node
C   mdofn .... maximum d.o.f.s per node
C   mnodee ... maximum number of nodes per element
C   nelem .... number of elements
C   nnode .... number of nodes
C
C   Variables used only in this subroutine:
C
C   inode .... index counter over node numbers
C   ireord ... flag to indicate to stop reordering process
C   jdofn .... index counter over d.o.f.s per node
C   jelem .... index counter over element numbers
C   jnode .... index counter over node numbers
C   kount .... counter and index of renumbered elements
C   nodeno ... node number of current node
C   numnod ... number of nodes in current element
C
C=====
C
C Array names:
C
C   Arrays brought in from main program:
C

```

```

c      iorder(nelem) ..... list of index for "nume( )" to give
c                           element numbers in an order to
c                           minimize frontwidth
c      itrip(nelem) ..... flag to indicate if current element
c                           has been renumbered
c      lc(mnodee,nelem) ..... list of element connectivity
c      nodcon(nnode) ..... vector list of number of connected
c                           nodes to current node
c      nodlis(nnode*maxall) ... vector list of node numbers of nodes
c                           connected to the current node
c      nodnew(nnode) ..... vector list of new node numbers for
c                           current renumbering scheme
c      nodold(nnode) ..... vector list of old node numbers for
c                           current renumberig scheme
c      nodren(nnode) ..... vector list of renumbered node
c                           numbers
c      nume(nelem) ..... vector list of element numbers
c      numet(nelem) ..... vector list of element types
c
c=====
c
c  Subroutines called:
c
c      renum ..... renumbers nodes to reduce bandwidth
c
c=====
c
c  Element types:
c
c      1  Truss:  2 nodes
c                3 d.o.f.s per node
c
c      2  Beam:   2 nodes
c                6 d.o.f.s per node
c
c      3  Plate:  4 nodes
c                6 d.o.f.s per node
c
c=====

      common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
&nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
&ndof,nres,neqn,mdofn,mnodee,mdofe,mprop,mgeom,moffe,iai,iao

      dimension nume(1),numet(1),lc(mnodee,1),iorder(1),itrip(1),
&nodcon(1),nodlis(1),nodnew(1),nodold(1),nodren(1)

c=====
c
c Determine the order of the index for the vector of element
c numbers, "nume( )", that will result in the elements being called
c in an order that minimizes the frontwidth. The order of the

```

```

c indexes are stored in "iorder( )".
c
c=====
c
c a. Set flag for reordering to the positive condition.
c
c     ireord=0
c
c b. Go to subroutine "renum" and determine a new nodal numbering
c     that will minimize the bandwidth. The new node numbering will
c     be in the vector "nodren". The index of the vector will
c     correspond to the old node number and the value will correspond
c     to the new node number. For example "nodren(3)=7" means old node
c     three is now new node number seven.
c
c     call renum(lc,nodcon,nodlis,nodnew,nodold,nodren,numet,ireord,
c     &nelem,nnode,mnodee,maxall,ibanwd,maxmin)
c
c c. Check condition of reordering flag and do the required action.
c
c     if (ireord.eq.0) then
c
c d. If the flag is still positive, start by initializing a counter
c     over the element indexes. Initialize the vector that holds the
c     indexes of the elements in the order necessary to minimize the
c     frontwidth and initialize a flag vector.
c
c     kount=0
c     do 1 jelem=1,nelem
c         iorder(jelem)=0
c         itrip(jelem)=0
c 1    continue
c
c e. Loop over all the nodes.
c
c     do 4 inode=1,nnode
c
c f. Loop over all the elements.
c
c     do 3 jelem=1,nelem
c
c g. Check to make sure that the current element has not already been
c     renumbered.
c
c     if (itrip(jelem).eq.0) then
c
c h. If the current element has not been renumbered, set the number
c     of nodes for the current element. Truss (type one) and beam
c     (type two) elements have two nodes. Plates (type three) have
c     four nodes.
c
c     numnod=2

```



```

        if (numet(jelem).eq.3) numnod=4
c
c i. Loop over all the nodes of the current element.
c
        do 2 jnode=1,numnod
c
c j. Get original node number of the current node of the current
c    element.
c
        nodeno=lc(jnode,jelem)
c
c k. Get new node number corresponding to the old node number of the
c    current node of the current element.
c
        nodeno=nodren(nodeno)
c
c l. Check to see if the new node number of the current node of
c    the current element is the same as the the node number on the
c    outer most loop.
c
        if (nodeno.eq.inode) then
c
c m. If the above is true, set the current element as the next
c    element in the front. Also check to see if all elements have
c    been set. If they have, jump out of the loop and continue the
c    program.
c
            kount=kount+1
            iorder(kount)=jelem
            itrip(jelem)=1
            if (kount.eq.nelem) goto 5
        endif
c
c n. Loop back and get next node of the current element.
c
        2      continue
c
c o. End of section for current element.
c
        endif
c
c p. Loop back and get next element.
c
        3      continue
c
c q. Loop back and get next basis node.
c
        4      continue
c
c r. Jump spot for inner loop given above when the element
c    renumbering is finished.
c

```

```

5    continue
c
c    else
c
c s. If no reordering is to be done, go to subroutine "ordnre" to
c    determine the original element order.
c
c        call ordnre(numel,iorder)
c
c    endif
c
c t. Return to the main program.
c
c        return
c        end

c        subroutine front(s,x,y,z,dn,ac,numel,numet,lc,isecl,imat,ioff,el,r,
c        &offset,geom,prop,msprg,sprf,iorder,nappr,ide2,lfrmax,nsprct,
c        &facflg)

c=====
c
c Subroutine to do frontal solution.
c
c=====
c
c coded by: Christopher D. Pionke
c coded for: SST Systems Inc.
c
c June, 1986
c
c Based on:
c
c    Hinton, E., and D. R. J. Owen. Finite Element Programming.
c    London: Academic Press, 1977.
c
c    The algorithm is found in chapter eight, pages 170-206.
c
c This subroutine is called by the main program to solve for the
c nodal displacements. It is called once for each loadcase and
c is used for both the frontwidth and non-frontwidth reduction
c techniques.
c
c=====
c
c Variable names:
c
c Variables brought in from main program:
c
c    facflg ... =.false. => reduce stiffness vector (1st loadcase)
c              =.true.  => do not reduce stiffness vector (2nd and

```

```

c                subsequent loadcases)
c    mdofe .... maximum number of d.o.f.s per element
c    mgeom .... maximum number of properties per section
c    moffe .... maximum number of offset distances per element
c    mprop .... maximum number of properties per material
c    nelem .... number of elements
c    ndof .... total number of d.o.f.s
c    neqn .... number of free d.o.f.s
c    nmat .... total number of materials
c    nnode .... total number of nodes
c    noff .... number of elements with offsets
c    nsec .... total number of sections
c    nsprct ... number of d.o.f.s with non-zero spring stiffnesses
c
c    Variables used only in this subroutine:
c
c    dum ..... dummy variable used in reading from disk
c    eqrhs .... right hand side load term after reduction
c    i ..... index counter over the elements
c    iappr .... index counter over the total number of d.o.f.s
c    idest .... index of location in front of the current d.o.f.
c    idofe .... index counter over d.o.f.s per element
c    idum ..... temporary storage and for dummy reads from disk
c    idum1 .... dummy variable used in reading from disk
c    ielem .... index counter over element numbers
c    ielva .... index counter over the eliminated variables
c    ifind .... flag to indicate whether or not an open position
c                in the front was found
c    ifront ... index counter over positions in the front
c    isprg .... index counter over the number of d.o.f.s with
c                spring stiffnesses
c    itots .... index counter over the stiffness vector
c    izeo4 ... integer*4 zero
c    i4 ..... index for a integer function
c    jdest .... index of location in front of the current d.o.f.
c    jdofe .... index counter over d.o.f.s per element
c    jfront ... index counter over positions in the front
c    jjj ..... dummy index counter
c    jnode .... index counter over the nodes of an element
c    j4 ..... index for a integer function
c    kdofn .... index counter over the d.o.f.s of a node
c    kelva .... number of eliminated variables
c    kevab .... counter of total element variables
c    kexis .... flag to indicate if the current d.o.f. is already
c                in the front
c    kfront ... index counter over positions in the front
c    kmat .... material number of active element
c    koff .... offset number of active element
c    ksec .... section number of active element
c    lfrmax ... largest number of unknowns in the front at any time
c    lfront ... index counter over positions in the front
c    mfront ... maximum number of variables allowed in front

```

```

c      mtots .... maximum number of terms in the stiffness vector
c      ndofe .... number of d.o.f.s per element
c      ndofn .... number of d.o.f.s per node
c      nfront ... number of current variables in the front
c      ngash .... location in global stiffness vector of current
c                  element stiffness term
c      ngish .... location in global stiffness vector of current
c                  element stiffness term
c      nikno .... nickname number for a particular d.o.f.
c      nloca .... location in global stiffness vector of current
c                  element stiffness term
c      nnodee ... number of nodes per element
c      nnspr .... location in global stiffness vector of a current
c                  d.o.f. spring stiffness
c      nposi .... position of a particular element d.o.f.
c      pivot .... diagonal pivoting term
c
c=====
c
c      Array names:
c
c      Arrays brought in from main program:
c
c      ac(ndof) ..... global combined nodal loads
c      dn(ndof) ..... global nodal displacements
c      el(nelem) ..... element length or thickness
c      geom(mgeom,nsec) ..... section properties
c      ide2(mdofe,nelem) ..... working copy of element d.o.f. vectors
c                          and releases
c      imat(nelem) ..... material number
c      ioff(nelem) ..... offset number
c      iorder(nelem) ..... list of index for nume( ) to give
c                          element numbers in ascending order
c      isec(nelem) ..... section number
c      lc(mnodee,nelem) ..... list of connectivity
c      msprg(nsprct) ..... list of d.o.f.s with non-zero spring
c                          stiffnesses
c      nume(nelem) ..... vector list of element numbers
c      numet(nelem) ..... vector list of element types
c      offset(mnodee*3,noff) .. offsets
c      prop(mprop,nmat) ..... material properties
c      r(9,nelem) ..... element rotation matrix
c      s(mtots) ..... global stiffness vector
c      sprf(ndof) ..... vector of d.o.f. spring stiffnesses
c      x(nnode) ..... x-coordinates of nodes
c      y(nnode) ..... y-coordinates of nodes
c      z(nnode) ..... z-coordinates of nodes
c
c      Arrays used only in this subroutine:
c
c      equat(mfront) ..... storage area of reduced equation
c                          coefficients

```

```

c      gload(mfront) ..... frontal global load vector
c      locel(24) ..... local vector holding element global
c                        d.o.f. numbers
c      nacva(mfront) ..... list of active variable nickname
c                        numbers in front
c      nappr(ndof) ..... flag to signal if a d.o.f. nickname
c                        number has already appeared
c      ndest(24) ..... destination vector of each element
c                        variable
c      sb(12,12) ..... beam stiffness matrix
c      sp(24,24) ..... plate stiffness matrix
c      st(6,6) ..... truss stiffness matrix
c      vecrv(mfront) ..... local vector of running solved
c                        displacements
c
c=====
c
c      Subroutines called:
c
c      ostif2 .... offsets beam stiffness matrix, if offsets exist
c      ostif3 .... offsets plate stiffness matrix, if offsets exist
c      rstif1 .... rotates truss stiffness matrix to global coordinates
c      rstif2 .... rotates beam stiffness matrix to global coordinates
c      rstif3 .... rotates plate stiffness matrix to global coordinates
c      stiff1 .... calculates stiffness matrix for truss element
c      stiff2 .... calculates stiffness matrix for beam element
c      stiff3 .... calculates stiffness matrix for plate element
c      wtime ..... supplied timing routine
c
c=====
c
c      Element types:
c
c      1  Truss:  2 nodes
c                3 d.o.f. per node
c
c      2  Beam:   2 nodes
c                6 d.o.f. per node
c
c      3  Plate:  4 nodes
c                6 d.o.f. per node
c
c=====
c
c      Files and file names:
c
c      * .. screen:      display program progress and statistics
c
c      2 .. scra1.dmp:    scratch file for reduced stiffness equations
c                        and reduced right hand sides of first loadcase
c
c      3 .. scra2.dmp:    scratch file for reduced right hand sides of

```

```

c                      second and subsequent loadcases
c
c=====
c
c a. Set up sizes of maximum front length and maximum stiffness
c     vector length. The first statement is used if the frontwidth
c     reduction technique is not used. The second statement is used
c     for the frontwidth reduction technique. The difference arises
c     due to the extra code and data area needed for the reduction
c     technique.
c
c     parameter ( mfront=222 , mtots=24753 )
c     parameter ( mfront=190 , mtots=18145 )
c
c     common lnn(999),len(999),lsn(99),lmn(99),nerr,nwar,
c     &nnode,nelem,ntruss,nbeam,nplate,nsec,nmat,nrel,noff,nspr,
c     &ndof,nres,neqn,mdofn,mnodee,mdofe,mprop,mgeom,moffe,iai,iao
c
c     real*8 s(mtots),st(6,6),sb(12,12),sp(24,24),equat(mfront),pivot,
c     &gload(mfront),vecrv(mfront),eqrhs,dum
c
c     dimension x(1),y(1),z(1),dn(1),ac(1),nume(1),numet(1),
c     &lc(mnodee,1),isec(1),imat(1),ioff(1),el(1),r(9,1),
c     &offset(moffe,1),geom(mgeom,1),prop(mprop,1),msprg(1),sprf(1),
c     &iorder(1),nappr(1),ide2(mdofe,1)
c
c     dimension nacva(mfront),locel(24),ndest(24)
c
c     logical*2 facflg
c
c     integer*4 nfunc,i4,j4,izero4
c
c a. Set up function giving the address of the stiffness term se(i,j)
c     in the stiffness vector s(nfunc). (This is valid for i <= j).
c
c     nfunc(i4,j4)=(j4*j4-j4)/2+i4
c
c b. Update screen.
c
c     write( *, 901 ) ' Begin wavefront solver
c     call wtime
c
c c. Initialize constants, variables, and arrays. Update screen.
c
c     write( *, 901 ) ' Initialize variables
c
c     izero4=0
c     kelva=0
c     lfrmax=0
c     nfront=0
c
c     do 1 iappr=1,ndof

```

```

      nappr(iappr)=0
1  continue

      do 2 ifront=1,mfront
        gload(ifront)=0.0
        equat(ifront)=0.0
        vecrv(ifront)=0.0
        nacva(ifront)=0
2  continue

c
c d. Initialize the stiffness vector. This only needs to be done
c   for the first loadcase.
c
      if(.not.facflg) then
        do 3 itots=1,mtots
          s(itots)=0.0
3      continue
      endif

c
c e. Update screen.
c
      call wtime

c=====
c
c  * * * * *  ELEMENT ELIMINATION AND REDUCTION SECTION  * * * * *
c
c=====
c
c f. Update screen.
c
      write( *, 901 ) ' Wavefront reduction
c
c g. Loop over all the elements.
c
      do 26 i=1,nelem
c
c h. Get index number of the active element.
c
        ielem=iorder(i)
c
c i. Set the number of active element variables to zero.
c
        kevab=0
c
c j. Get section property number, material property number, and
c   offset number for the active element.
c
        ksec=isec(ielem)
        kmat=imat(ielem)
        koff=ioff(ielem)
c

```

```

c k. Determine the type of the active element and go to the proper
c   section.
c
c       goto (4,5,6) numet(ielem)
c
c * * *   truss   * * * * *
c
c l-t. Set node and d.o.f. information for truss.
c
c       4       nnodee=2
c              ndofn=3
c              ndofe=6
c
c m-t. If this is the first loadcase, get truss stiffness matrix and
c       then rotate it to global coordinates.
c
c       if (.not.facflg) then
c           call stiff1(st,el(ielem),geom(1,ksec),prop(1,kmat))
c           call rstif1(st,r(1,ielem))
c       endif
c
c n-t. Skip beam and plate calculations and continue with program.
c
c       goto 7
c
c * * *   beam   * * * * *
c
c l-b. Set node and d.o.f information for beam.
c
c       5       nnodee=2
c              ndofn=6
c              ndofe=12
c
c m-b. If this is the first loadcase, get beam stiffness matrix and
c       then rotate it to global coordinates.
c
c       if (.not.facflg) then
c           call stiff2(sb,el(ielem),geom(1,ksec),prop(1,kmat))
c           call rstif2(sb,r(1,ielem))
c
c n-b. If offsets exist for this element, offset the beam stiffness
c       matrix.
c
c           if (koff.gt.0) call ostif2(sb,offset(1,koff))
c
c       endif
c
c o-b. Skip plate information and continue with program.
c
c       goto 7
c
c * * *   plate   * * * * *

```



```

c
c 1-p. Set node and d.o.f information for plate.
c
      6      nnodee=4
            ndofn=6
            ndofe=24
c
c m-p. If this is the first loadcase, get plate stiffness matrix and
c      then rotate it to global coordinates.
c
            if (.not.facflg) then
                call stiff3(sp,x,y,z,lc(1,ielem),r(1,ielem),el(ielem),
&                prop(1,kmat),offset(1,koff),koff)
                call rstif3(sp,r(1,ielem))
c
c n-p. If offsets exist for this element, offset the plate stiffness
c      matrix.
c
            if (koff.gt.0) call ostif3(sp,offset(1,koff))
c
            endif
c
c o-p. Continue with program for plate section and this is where the
c      program will come for the truss and beam sections.
c
      7      continue
c
c *****
c
c p. Loop over all the active element d.o.f.s, determine the element
c      position of each global d.o.f. number and store in variable
c      "nposi". Then store the global d.o.f. number in position
c      "nposi" in the vector "locel(npoise)".
c
            do 9 jnode=1,nnodee
                do 8 kdofn=1,ndofn
                    nposi=(jnode-1)*ndofn+kdofn
                    locel(nposi)=ide2(nposi,ielem)
                8      continue
            9      continue
c
c *****
c
c q. Loop over all the d.o.f.s for the active element.
c
            do 13 jdofe=1,ndofe
c
c r. Set the nickname number to the current active element d.o.f..
c
                nikno=iabs(locel(jdofe))
c
c s. Set a flag to indicate whether or not the current d.o.f. is

```

```

c   already in the front to the negative condition.
c
c       kexis=0
c
c t. Loop over all the active positions in the front.
c
c       do 10 kfront=1,nfront
c
c u. If the d.o.f. is already active: increase the counter of active
c   element variables by one, switch the already active switch to
c   positive, and set the destination of the current d.o.f. to the
c   position in the front where the d.o.f. already exists.
c
c       if (nikno.eq.nacva(kfront)) then
c           kevab=kevab+1
c           kexis=1
c           ndest(kevab)=kfront
c       endif
c   10   continue
c
c v. If the current d.o.f. is not already in the front, then do the
c   following:
c
c       if (kexis.eq.0) then
c
c w. Set the trip flag for finding an open position in the front to
c   negative.
c
c       ifind=0
c
c x. Loop over all the possible positions available in the front.
c
c       do 11 kfront=1,mfront
c
c y. Check each position to see if it is an open position.
c
c       if (nacva(kfront).eq.0) then
c
c z. When the first open position is found: put the nickname number
c   of the current d.o.f. in the open position of the front,
c   increase the counter of active element variables by one, set
c   the position of the current d.o.f. to the location in the front
c   where it was stored, set trip flag for finding an open position
c   in the front to positive, then exit loop and continue with
c   program.
c
c       nacva(kfront)=nikno
c       kevab=kevab+1
c       ndest(kevab)=kfront
c       ifind=1
c       goto 12

```

```

                endif
11      continue
c
c aa. If an open position in the front is not found, write an error
c      message to the screen and exit the program.
c
12      if (ifind.eq.0) then
            write(*,902)
            stop
        endif
c
c ab. If the location of the current d.o.f. in the front was stored
c      outside the current frontwidth, increase the current
c      frontwidth. Also, adjust the maximum frontwidth attained to
c      this point if conditions warrant.
c
            if (ndest(kevab).gt.nfront) then
                nfront=ndest(kevab)
                if(nfront.gt.lfrmax) then
                    lfrmax=nfront
                endif
            endif
c
        endif
c
c ac. End of the loop over all the d.o.f.s of the active element.
c      At this point all the d.o.f.s of the active element have been
c      stored in the front and all the locations have been recorded.
c
13      continue
c
c*****
c
c ad. Loop over all the active element d.o.f.s again.
c
        do 19 idofe=1,ndofe
c
c ae. Store the front location of the current d.o.f. in "idest".
c
            idest=ndest(idofe)
c
c af. Check to see if this is the first appearance of this d.o.f..
c
            if (nappr(nacva(idest)).eq.0) then
c
c ag. If this is the first appearance of this d.o.f.: get the total
c      load for this d.o.f. and put it in the load vector "gload( )",
c      add the spring stiffness for this d.o.f. to the correct
c      location of the global stiffness vector (if this is the first
c      loadcase), then change the first appearance flag for this
c      d.o.f. to positive.
c

```

```

        gload(idest)=dble(ac(nacva(idest)))
        if (.not.facflg) then
            do 14 isprg=1,nsprct
                if(nacva(idest).eq.msprg(isprg)) then
                    i4=idest
                    nnspr=nfunc(i4,i4)
                    s(nnspr)=s(nnspr)+dble(sprf(isprg))
                endif
14          continue
            endif
            nappr(nacva(idest))=1
        endif
c
c ah. Check to see if this is the first loadcase.
c
        if (.not.facflg) then
c
c ai. If this is the first loadcase, loop over all the active element
c      d.o.f.s up to the current d.o.f..
c
            do 18 jdofe=1,idofe
c
c aj. Store the location of the inner loop d.o.f. in "jdest".
c
                jdest=ndest(jdofe)
c
c ak. Get address of global stiffness vector where the element
c      stiffness term corresponding to the indexes "idest" and "jdest"
c      will go. "ngash" will be used if "jdest" is >= to "idest" and
c      "ngish" will be used if "jdest" is < "idest".
c
                i4=idest
                j4=jdest
                ngash=nfunc(i4,j4)
                ngish=nfunc(j4,i4)
c
c al. Determine the type of the active element and go to the proper
c      section.
c
                goto (15,16,17) numet(ielem)
c
c * * *   truss   * * * * *
c
c am-t. If active element is a truss, add the current element d.o.f.
c      stiffness term to the appropriate location in the global
c      stiffness vector.
c
15          if (jdest.ge.idest) then
                s(ngash)=s(ngash)+st(idofe,jdofe)
            else
                s(ngish)=s(ngish)+st(idofe,jdofe)
            endif

```

```

c
c an-t. Skip beam and plate sections and continue program.
c
c         goto 18
c
c * * * beam * * * * *
c
c am-b. If active element is a beam, add the current element d.o.f.
c       stiffness term to the appropriate location in the global
c       stiffness vector.
c
c       16         if (jdest.ge.idest) then
c                   s(ngash)=s(ngash)+sb(idofe,jdofe)
c                   else
c                   s(ngish)=s(ngish)+sb(idofe,jdofe)
c                   endif
c
c an-b. Skip plate section and continue program.
c
c         goto 18
c
c * * * plate * * * * *
c
c am-p. If active element is a plate, add the current element d.o.f.
c       stiffness term to the appropriate location in the global
c       stiffness vector.
c
c       17         if (jdest.ge.idest) then
c                   s(ngash)=s(ngash)+sp(idofe,jdofe)
c                   else
c                   s(ngish)=s(ngish)+sp(idofe,jdofe)
c                   endif
c
c an-p. Continue inner loop over element d.o.f.s.
c
c       18         continue
c
c ao. End of section to assemble the element stiffness matrix into
c       the global stiffness vector.
c
c         endif
c
c ap. Continue the outer loop over element d.o.f.s.
c
c       19         continue
c
c *****
c
c aq. Loop over the active element d.o.f. again for possible
c       elimination.
c
c         do 25 idofe=1,ndofe

```

```

c
c ar. Multiply the nickname number of the current element d.o.f. by
c   -1 and then check its sign to see if it is positive. Since at
c   first the sign of any d.o.f. ready for elimination is negative,
c   it will now be positive. Therefore, only positive nickname
c   numbers will be eliminated.
c
      nikno=-locel(idofe)
      if (nikno.gt.0) then
c
c as. The section inside this if statement will be executed if the
c   current d.o.f. is ready for elimination. Loop over all
c   positions in the front to find the position of the d.o.f. that
c   is being eliminated.
c
      do 23 ifront=1,nfront
        if (nacva(ifront).eq.nikno) then
c
c at. This section is executed when the position of the now to be
c   eliminated d.o.f. is found. If this is the first load case,
c   the global stiffness vector terms that correspond to the
c   coefficients of the equation (that has the about to be
c   eliminated d.o.f. as the diagonal) term has to be found so
c   they can be written to disk. For subsequent loadcases this
c   can be skipped since the reduced stiffness equations are
c   already on disk.
c
          if (.not.facflg) then
c
c au. Loop over all active positions in the front.
c
          do 20 jfront=1,nfront
c
c av. Determine whether the position in the front of the d.o.f. being
c   eliminated is less than or greater than or equal to the current
c   loop index over all the active front positions. From this
c   determination, find the address in the global stiffness vector
c   that corresponds to these two subscripts.
c
          i4=ifront
          j4=jfront
          if (ifront.lt.jfront) then
            nloca=nfunc(i4,j4)
          else
            nloca=nfunc(j4,i4)
          endif
c
c aw. Put the global stiffness term in the location as given above
c   into the equation coefficient vector "equat( )".
c
          equat(jfront)=s(nloca)
c

```

```

c ax. Set the global stiffness term just stored in "equat( )" to zero
c   so that this position can be used later for a new d.o.f..
c
c       s(nloca)=0.0
c
c ay. Continue loop over all the active positions in the front. At
c   the end of this loop "equat( )" will have all the coefficients
c   of the row that has the about to be eliminated d.o.f. as the
c   diagonal term.
c
c       20          continue
c
c az. End of section of finding "equat( )" for the about to be
c   eliminated d.o.f. (if this is the first loadcase).
c
c       endif
c
c ba. Set the load corresponding to the about to be eliminated d.o.f.
c   to the current right hand side and then set the load term to
c   zero so that it can be used for another d.o.f..
c
c       eqrhs=gload(ifront)
c       gload(ifront)=0.0
c
c bb. Increment the counter of eliminated d.o.f.s by one.
c
c       kelva=kelva+1
c
c bc. If this is the first loadcase, write the eliminated d.o.f.
c   nickname number, position in the front, the current frontwidth,
c   the right hand side, and all the coefficients of its equation
c   to disk in file 2.
c
c       if (.not.facflg) then
c           write(2) nikno,ifront,nfront,eqrhs,
c           &          (equat(jjj),jjj=1,nfront)
c       else
c
c bd. If this is not the first loadcase, write the current right hand
c   side to disk file 3. Then read the current nickname number,
c   the position in the front as a dummy variable, the old
c   frontwidth as a dummy variable, the old right hand side as a
c   dummy variable, and all the reduced coefficients of the
c   equation of the current d.o.f..
c
c       write(3) eqrhs
c       read(2) nikno,idum,idum1,dum,
c       &          (equat(jjj),jjj=1,idum1)
c
c       endif
c
c be. Set "pivot" equal to the diagonal term of the equation about to

```

```

c      be eliminated. Then set its location in the coefficient vector
c      "equat( )" to zero for convenience.
c
c          pivot=equat(ifront)
c          equat(ifront)=0.0
c
c      bf. Check to see if the nickname number of the current about to be
c      eliminated d.o.f. is greater than the total number of
c      equations. If it is, then it is a restrained d.o.f. with a
c      fixed displacement. If it is not, then it is a free d.o.f..
c      If the degree of freedom is restrained, nothing is done.
c
c          if (nikno.le.neqn) then
c
c      bg. If the d.o.f. is free, the load vector must be reduced and
c      if this is the first loadcase, the global stiffness vector
c      must be reduced. Loop over all the currently active positions
c      in the front.
c
c          do 22 jfront=1,nfront
c
c      bh. Reduce the load vector.
c
c          gload(jfront)=gload(jfront)-equat(jfront)*eqrhs/
c      &          pivot
c
c      bi. If this is the first loadcase, reduce the global stiffness
c      vector.
c
c          if (.not.facflg) then
c
c      bj. Check to see if time is being wasted on zero operations.
c
c          if (equat(jfront).ne.0.0) then
c
c      bk. This statement removes the function evaluation of the global
c      stiffness vector positions from the innermost expensive loop.
c
c          j4=jfront
c          nloca=nfunc(izero4,j4)
c
c      bl. Loop over all positions in the front up to the outer loop
c      index.
c
c          do 21 lfront=1,jfront
c
c      bm. Get address of current stiffness term about to be reduced.
c
c          ngash=lfront+nloca
c
c      bn. Reduce the current stiffness term.
c

```



```

                                s(ngash)=s(ngash)-equat(jfront)*
                                &                                equat(lfront)/pivot
c
c bo. Continue reduction in the inner loop.
c
    21                                continue
c
c bp. End of section to reduce stiffness terms for a non-zero free
c    d.o.f..
c
                                endif
c
c bq. End of section to reduce stiffness terms for a free d.o.f..
c
                                endif
c
c br. End of loop to reduce the load vector and global stiffness
c    vector for a free d.o.f..
c
    22                                continue
c
c bs. End of reduction section.
c
                                endif
c
c bt. The spot in the front associated with the eliminated variable
c    is made vacant.
c
                                nacva(ifront)=0
c
c bu. End section to eliminate a d.o.f..
c
                                endif
    23                                continue
c
c bv. Check to see if frontwidth can be reduced.
c
    24                                if (nacva(nfront).eq.0) then
                                    nfront=nfront-1
                                    goto 24
                                endif
c
c bw. End of section that checks to see if current d.o.f. can be
c    eliminated.
c
                                endif
c
c bx. Loop back up and check next d.o.f. of the active element to see
c    if it can be eliminated.
c
    25                                continue
c

```

```

c by. Loop back up and repeat the whole process on the next element.
c
c   26 continue
c
c bz. Update screen.
c
c   call wtime

c=====
c
c   * * * * * ENTER BACK SUBSTITUTION PHASE * * * * *
c
c=====
c
c ca. Update screen.
c
c   write( *, 901 ) ' Back substitution
c
c cb. Loop over all the eliminated variables.
c
c   do 28 ielva=1,kelva
c
c cc. Read the variable nickname, position in the front, the
c   frontwidth, the right hand side, and all the left hand side
c   coefficients of the d.o.f. to be solved for.
c
c       backspace 2
c       read(2) nikno,ifront,nfront,eqrhs,
c   &       (equat(jjj),jjj=1,nfront)
c       backspace 2
c
c cd. If this is not the first loadcase, read the new right hand
c   side.
c
c       if (facflg) then
c           backspace 3
c           read(3) eqrhs
c           backspace 3
c       endif
c
c ce. Set the pivot term as the diagonal term.
c
c       pivot=equat(ifront)
c
c cf. Check to see if the current d.o.f. nickname number is greater
c   than the number of equations. If it is, it is a restrained
c   d.o.f.. If it is not, it is a free d.o.f..
c
c       if (nikno.gt.neqn) then
c
c cg. If it is a restrained d.o.f., set the value in "vecrv(ifront)"
c   to zero.

```

```

c          vecrv(iframe)=0.0
c
c          else
c
c ch. If it is a free d.o.f., set the diagonal term in the coefficient
c      vector to zero.
c
c          equat(iframe)=0.0
c
c ci. Then loop over all the active positions in the front, and
c      reduce the right hand side from the already solved
c      displacements.
c
c          do 27 jfront=1,nfront
c              eqrhs=eqrhs-vecrv(jfront)*equat(jfront)
c 27          continue
c
c cj. Solve for the displacement of the current d.o.f. by dividing
c      the reduced right hand side by the pivot term.
c
c          vecrv(iframe)=eqrhs/pivot
c
c ck. End section to solve for displacements.
c
c          endif
c
c cl. If the current d.o.f. is a free d.o.f., store the answer for
c      the displacement of the d.o.f. in the location used by the
c      rest of the program. (The restrained d.o.f. displacements are
c      already stored.)
c
c          if (nikno.le.neqn) dn(nikno)=real(vecrv(iframe))
c
c cm. Loop back and solve for next d.o.f..
c
c      28 continue
c
c cn. Update screen.
c
c          call wtime
c
c co. Format block.
c
c 901 format(a44)
c 902 format(//,' Frontwidth too large, analysis stopped',//)
c
c cp. Return to the main program.
c
c          return
c          end

```

VITA

Christopher D. Pionke was born in Waupun, Wisconsin on December 6, 1956. In September of 1976 he entered The University of Tennessee (UT). During the fifth year of his six year program, he attended the University of Stirling, in Stirling, Scotland as an ISEP scholar to advance his somewhat lax business skills and generally have a good time. In 1981 he returned to UT to finish his dual bachelors degrees in Engineering Physics and Engineering Science. Upon graduating in 1982, Chris accepted a so-so job with Presearch Inc. in Oak Ridge, Tennessee where he met his future wife, the lovely and charming Cynthia M. McLaughlin, a 1982 Civil Engineering graduate of UT.

After a two year stint with Presaerch, during which time Chris attended graduate school part-time, he went back to UT on a full-time basis to complete his Master's degree in Engineering Science. When Chris completed all of his course work, but before he finished this thesis, he accepted a job with Optic-Electronic Corp. in Dallas, Texas. During his two years there, he managed to burn his furniture, propose to his girlfriend, build and buy a house, get married and still not finish this thesis. After his wife Cindy graduated with her Master's degree in Civil Engineering, Chris gratefully quit his job so that she could start supporting him in the manner to which he wanted to become accustomed. Upon finally finishing this thesis and graduating from UT, Chris plans to pursue a Ph.D. at the Georgia Institute of Technology while his wife continues to support him.