

LINEAR AND NONLINEAR ANALYSIS OF SPACE TRUSSES  
RELATIVE TO CYLINDRICAL COORDINATES

by

Jeffrey A. Perrier

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE  
in  
Civil Engineering

APPROVED:

---

S. M. Holzer, Chairman

---

A. E. Somers, Jr.

---

R. M. Barker

December, 1982  
Blacksburg, Virginia

## ACKNOWLEDGEMENTS

The author would like to gratefully acknowledge Dr. S. M. Holzer for his guidance, comments, and suggestions, and for his time in proof-reading parts of the thesis.

Appreciation is given to Dr. L. T. Watson for his helpful suggestions, and to Dr. A. E. Somers, Jr. and Dr. R. M. Barker for their helpful comments and for serving on the committee.

The author gratefully acknowledges the financial support provided by the Department of Civil Engineering and the use of the computer facilities of Virginia Polytechnic Institute and State University.

Comments and assistance from fellow graduate students, particularly Edgar E. Lamma, III, G. Stuart Matthis, II, and Douglas G. Fitzpatrick, are also appreciated.

## TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS . . . . .	ii
LIST OF FIGURES . . . . .	iv
LIST OF TABLES . . . . .	v
1. INTRODUCTION . . . . .	1
2. STRUCTURAL MODELS . . . . .	3
2.1 Cylindrical Coordinate Systems . . . . .	3
2.2 Radial Symmetry . . . . .	3
2.3 Linear Element Model . . . . .	7
2.4 Linear System Model . . . . .	18
2.5 Nonlinear Element Model . . . . .	19
2.6 Nonlinear System Model . . . . .	30
2.7 Stability of Equilibrium . . . . .	30
2.8 Solutions of Systems of Linear Algebraic Equations	31
3. DISCUSSION OF RESULTS . . . . .	34
3.1 Linear Analysis . . . . .	34
3.1.1 Structure 1 . . . . .	34
3.1.2 Structure 2 . . . . .	36
3.1.3 Comparison of Effort . . . . .	36
3.2 Nonlinear Analysis . . . . .	42
3.2.1 Structure 1 . . . . .	42
3.2.2 Structure 2 . . . . .	47
4. CONCLUSION AND RECOMMENDATIONS . . . . .	53
APPENDIX A - REFERENCES . . . . .	57
APPENDIX B - LINEAR PROGRAM . . . . .	60
APPENDIX C - NONLINEAR PROGRAM . . . . .	89
VITA . . . . .	179
ABSTRACT	

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
2.1	Cylindrical Coordinate System . . . . .	4
2.2	"Pie Slice" of a Structure . . . . .	6
2.3	Linear Truss Element . . . . .	8
2.4	Linear Model Coordinate Transformations . . . . .	12
2.5	Nonlinear Truss Element . . . . .	20
2.6	Nonlinear Model Coordinate Transformations . . . . .	27
3.1	Structure 1 . . . . .	35
3.2	"Pie Slices" of Structure 1 . . . . .	37
3.3	Structure 2 . . . . .	38
3.4	Percent of Effort vs. Percent of Structure . . . . .	41
3.5	Nonlinear Load vs. Vertical Displacement of Compression Ring Curve for Structure 1 Showing Locations of Bifurcation Points . . . . .	44
3.6	Buckled Configuration of Structure 1 . . . . .	46
3.7	Percent of Effort vs. Percent of Structure . . . . .	49
3.8	"Pie Slice" of Structure 2 . . . . .	51

LIST OF TABLES

<u>Table</u>		<u>Page</u>
3.1	Computational Effort Required in Linear Analysis . . . . .	40
3.2	Computational Effort Required in Nonlinear Analysis . . . . .	48

## CHAPTER 1

### INTRODUCTION

The purpose of this study is to formulate linear and nonlinear three-dimensional truss element models in cylindrical coordinates, to implement these element models using existing solution techniques, and to investigate the utility of such element models in the analysis of two radially symmetric lamella domes. The linear model is used to perform a joint displacement and element force analysis while the nonlinear model is used to trace the nonlinear load vs. displacement path, to locate bifurcation and limit points, and, frequently, to branch off of the fundamental equilibrium path onto bifurcation paths. The modified Riks/Wempner method [8,15] for tracing nonlinear equilibrium paths is used in the nonlinear model because of its ability to trace equilibrium paths through limit points into the post-critical range and because of its good convergence characteristics.

The reason for formulating structural models in cylindrical coordinates is three-fold.

1. In the analysis of essentially round structures, generation of input data and interpretation of results is much easier when joint forces, constraints, and displacements are expressed as radial, tangential, or vertical rather than in terms of three arbitrary global directions.

2. Under certain conditions, a cylindrical coordinate system allows imposition of the appropriate boundary conditions to use a "pie slice" of a radially symmetric structure to represent the entire structure.
3. When existing methods for branching onto bifurcation paths fail, noting which bifurcation points are suppressed by which "pie slice" configurations provides, at least, qualitative information regarding geometries associated with bifurcation buckling modes.

The only paper located by the author in which a similar model formulation is used is a dissertation by Jagannathan [9] who performs a nonlinear analysis on radially symmetric structures using "pie slices" but does not include the details of his element model formulation.

## CHAPTER 2

### STRUCTURAL MODELS

In this chapter, the linear and nonlinear truss models used in this study are developed. The cylindrical coordinate system is described and radial symmetry, stability of equilibrium, and solution of systems of linear algebraic equations are discussed.

#### 2.1 Cylindrical Coordinate Systems

The system model is assembled in such a way that the joint forces, displacements, and constraints are expressed relative to distinct joint coordinate systems which together form an overall cylindrical, rather than cartesian, coordinate system.

Fig. 2.1 is the plan view of a reticulated dome showing the orientations of the joint coordinate systems. The overall global coordinate system, shown in dashed lines, is still required to specify the joint coordinates and is used to calculate the element deformations. Since all coordinate systems used in this study are right-handed, both the global and joint 3-directions are down.

#### 2.2 Radial Symmetry

The degree of radial symmetry exhibited by a structure can be designated  $C_n$  after Glockner [4] where  $n$  is effectively the number of identical "pie slices" into which the structure can be divided.

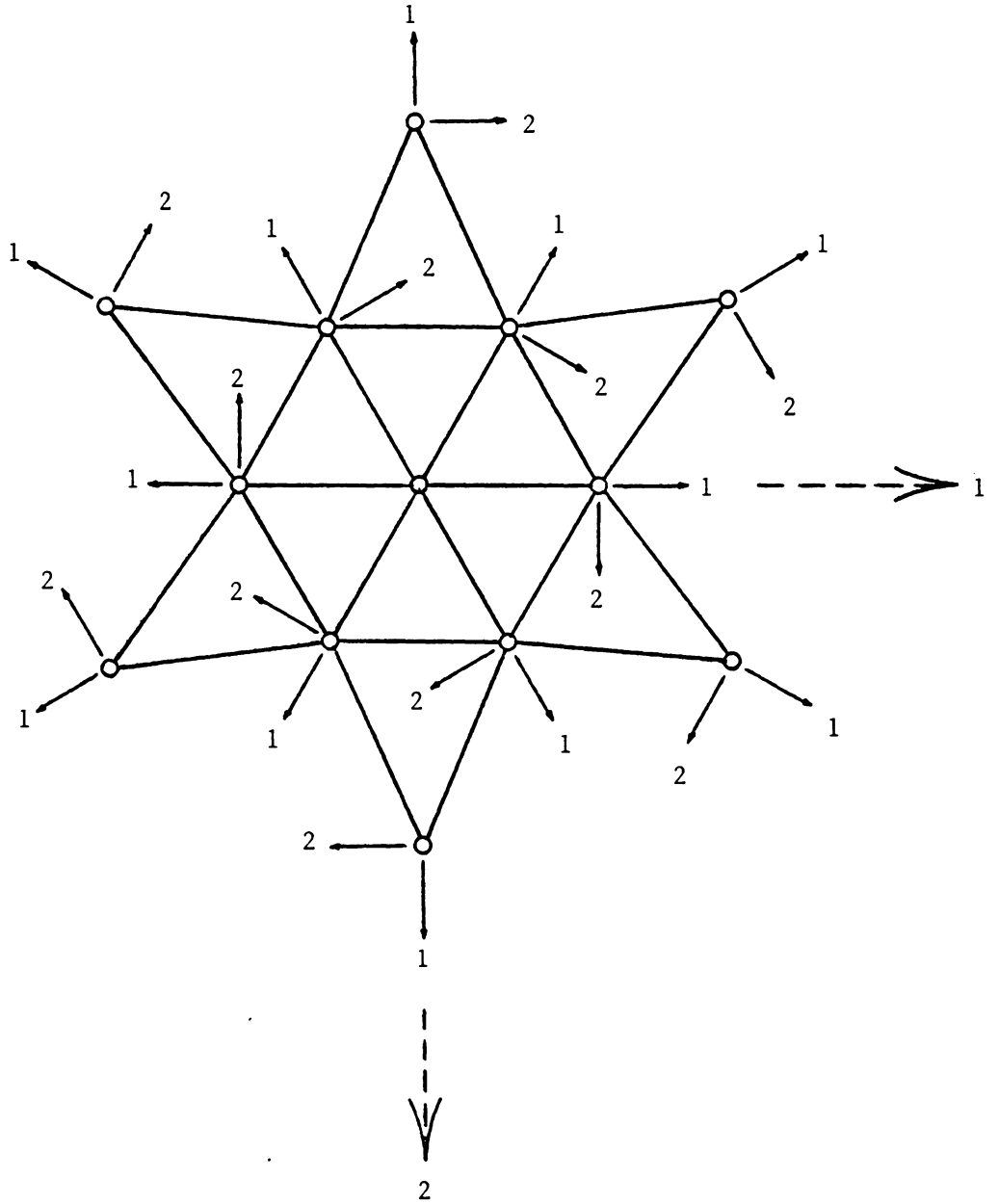


Figure 2.1: Cylindrical Coordinate System

Some structures exhibit several degrees of radial symmetry. For example, a structure which can be divided into four identical parts can also be divided into two identical parts. Therefore, such a structure would exhibit both  $C_4$  and  $C_2$  radial symmetry. The dome shown in Fig. 2.1 exhibits  $C_6$ ,  $C_3$ , and  $C_2$  radial symmetry. Fig. 2.2 illustrates a "pie slice" equal to one sixth of the dome in Fig 2.1 which could be used to represent the entire structure provided the loading condition also exhibited  $C_6$  symmetry. Slices equal to one third of one half of the structure could also be used if the loads exhibited  $C_3$  or  $C_2$  symmetry respectively. For a "pie slice" to be used to represent the entire structure, both the loads and the structure must exhibit the same degree of symmetry and the "pie slice" itself must exhibit bisymmetry. This latter condition must be met to insure that the tangential constraints applied to the "pie slice" do not restrict any "twisting" of the entire structure which could result from non-bisymmetric slices.

Note that, in Fig. 2.2, since its orientation is arbitrary, the joint coordinate system at the joint lying on the radial axis of symmetry is taken to coincide with the global coordinate axes. Joints 1, 2, and 3 are not constrained against motion in the 3-direction and the original constraint conditions at joint 4 are maintained. For a rigorous discussion of the rules governing division of structures into symmetric parts, see Glockner [4] and Holzer [7]. Let it suffice to say here that 1) the cross-sectional areas of members lying on one cutting plane are reduced by one half, 2) the cross-sectional areas of members lying on the intersection of two

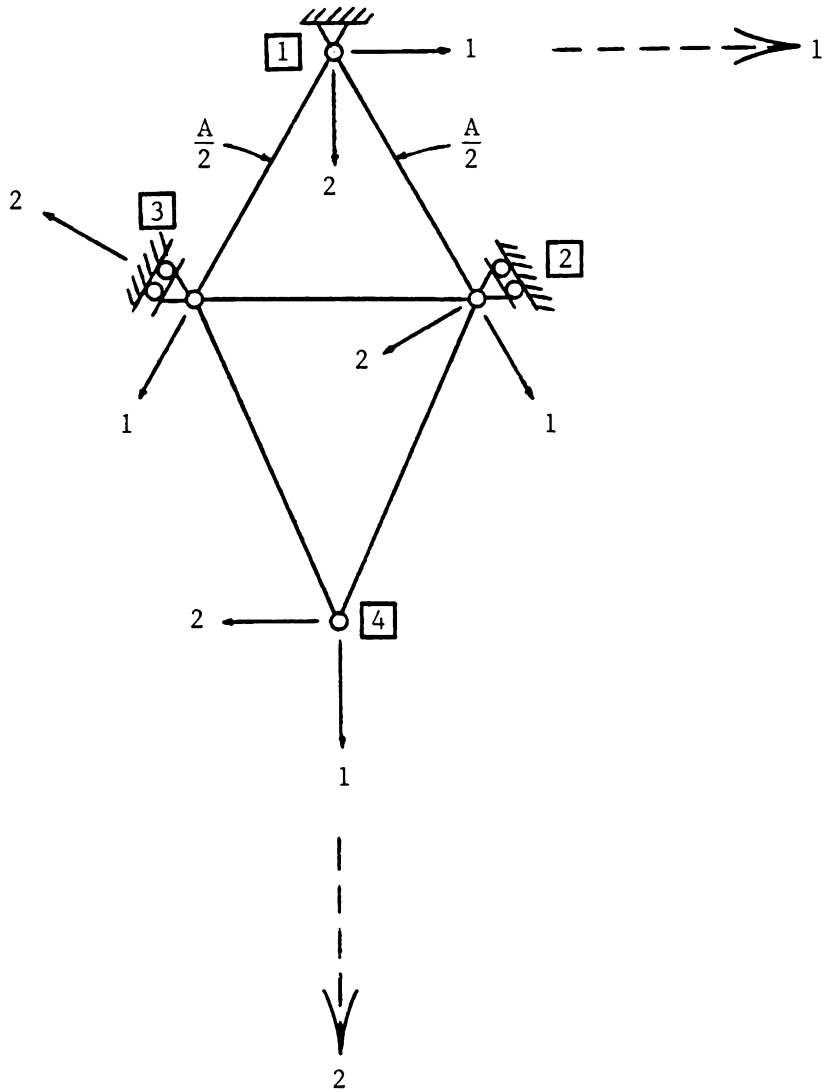


Figure 2.2: "Pie Slice" of a Structure

cutting planes (i.e. the radial axis of symmetry) are divided by the number of slices into which the structure was divided (the subscript,  $n$ , in the expression for the degree of radial symmetry,  $C_n$ ), 3) the magnitudes of loads applied to joints lying on one cutting plane are reduced by one half, and 4) the magnitudes of loads applied to joints lying on the intersection of two cutting planes are divided by the number of slices into which the structure was divided.

### 2.3 Linear Element Model

The linear truss element model, Fig. 2.3, is from Holzer [5,7] and can be expressed in local element coordinates as

$$f = kd \quad (2.1)$$

where

$$f = \begin{bmatrix} f_a \\ f_b \end{bmatrix} \quad (2.1a)$$

$$k = \gamma \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (2.1b)$$

$$\gamma = \frac{AE}{L} \quad (2.1c)$$

$$d = \begin{bmatrix} d_a \\ d_b \end{bmatrix} \quad (2.1d)$$

$A$  = the cross-sectional area of the element

$E$  = the modulus of elasticity of the element material

$L$  = the element length

The coordinate transformation process is a specialization of

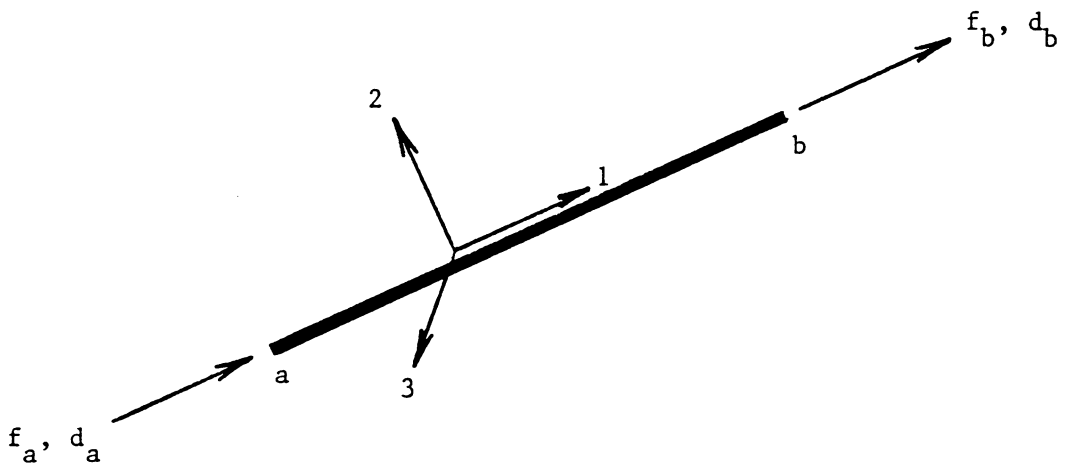


Figure 2.3: Linear Truss Element

that given by Holzer [7] for use in the analysis of problems with independent joint coordinate systems and similar to that employed by Jagannathan [9] in the analysis of geometrically nonlinear problems.

Since the joint coordinate systems may be different at each end of the element (Figs. 2.1 and 2.2), two transformation matrices,  $\lambda$  and  $\hat{\lambda}$  (which accomplish the transformation from local to cylindrical coordinates at the a- and b-ends of the element respectively), are required and are defined such that

$$\begin{aligned} d_a &= \lambda D_a \\ d_b &= \hat{\lambda} D_b \\ F_a &= \lambda^T f_a \\ F_b &= \hat{\lambda}^T f_b \end{aligned} \tag{2.2}$$

where  $d_a$ ,  $f_a$  and  $d_b$ ,  $f_b$  are the element end displacements and forces at the a- and b-ends respectively relative to local element coordinates, and  $D_a$ ,  $F_a$  and  $D_b$ ,  $F_b$  are the element end displacements and forces at the a- and b-ends respectively relative to cylindrical coordinates. Eqs. 2.2 can be expressed more compactly as

$$\begin{aligned} d &= \Lambda D \\ F &= \Lambda^T f \end{aligned} \tag{2.3}$$

where

$$D = \begin{bmatrix} D_a \\ D_b \end{bmatrix} \quad (2.3a)$$

$$F = \begin{bmatrix} F_a \\ F_b \end{bmatrix} \quad (2.3b)$$

$$\Lambda = \begin{bmatrix} \lambda & 0 \\ 0 & \hat{\lambda} \end{bmatrix} \quad (2.3c)$$

Eqs. 2.1 and 2.3 can then be combined to form the linear truss element model in cylindrical coordinates

$$F = KD \quad (2.4)$$

where

$$K = \Lambda^T k \Lambda \quad (2.4a)$$

The transformation matrices,  $\lambda$  and  $\hat{\lambda}$ , are defined as

$$\begin{aligned} \lambda &= [c_1 \ c_2 \ c_3] \\ \hat{\lambda} &= [\hat{c}_1 \ \hat{c}_2 \ \hat{c}_3] \end{aligned} \quad (2.5)$$

where  $c_i$  and  $\hat{c}_i$ ,  $i = 1, 2, 3$ , are the direction cosines of the element local 1-axis relative to the joint  $i$ -axis at the a- and b-ends respectively.

The direction cosines of the element local 1-axis relative to the global coordinate axes,  $c_i$ ,  $i = 1, 2, 3$ , are defined as

$$c_i = \frac{L_i}{L}, \quad i = 1, 2, 3 \quad (2.6)$$

where  $L_i$  is the component of the element length vector in the global  $i$ -direction.

Referring to Fig. 2.4, which is the projection of an arbitrary element onto the global 1,2-plane, expressions for  $c_i$  and  $\hat{c}_i$ ,  $i = 1, 2, 3$ , are easily formulated. The dashed lines represent the global coordinate system, the solid lines represent the joint coordinate system, and the arrowhead on the element indicates the direction of the element local 1-axis. Detailed formulations are given only for the expressions for  $c_i$ ,  $i = 1, 2, 3$ .

The direction cosine of the element local 1-axis relative to the joint 1-axis is the ratio of the component of the element length vector in the joint 1-direction to the total element length

$$c_1 = \frac{L_{12}}{L} \cos \phi_1 = \frac{L_{12}}{L} \sin \phi_2 \quad (2.7)$$

where

$$L_{12} = [L_1^2 + L_2^2]^{1/2} \quad (2.7a)$$

Noting that

$$\phi_2 = \theta + \psi \quad (2.8)$$

and employing the angle sum relation for the sine function yields

$$\begin{aligned} c_1 &= \frac{L_{12}}{L} [\sin \theta \cos \psi + \cos \theta \sin \psi] \\ &= \frac{L_{12}}{L} \left[ \begin{pmatrix} X_{2a} \\ R_a \end{pmatrix} \begin{pmatrix} L_2 \\ L_{12} \end{pmatrix} + \begin{pmatrix} X_{1a} \\ R_a \end{pmatrix} \begin{pmatrix} L_1 \\ L_{12} \end{pmatrix} \right] \end{aligned}$$

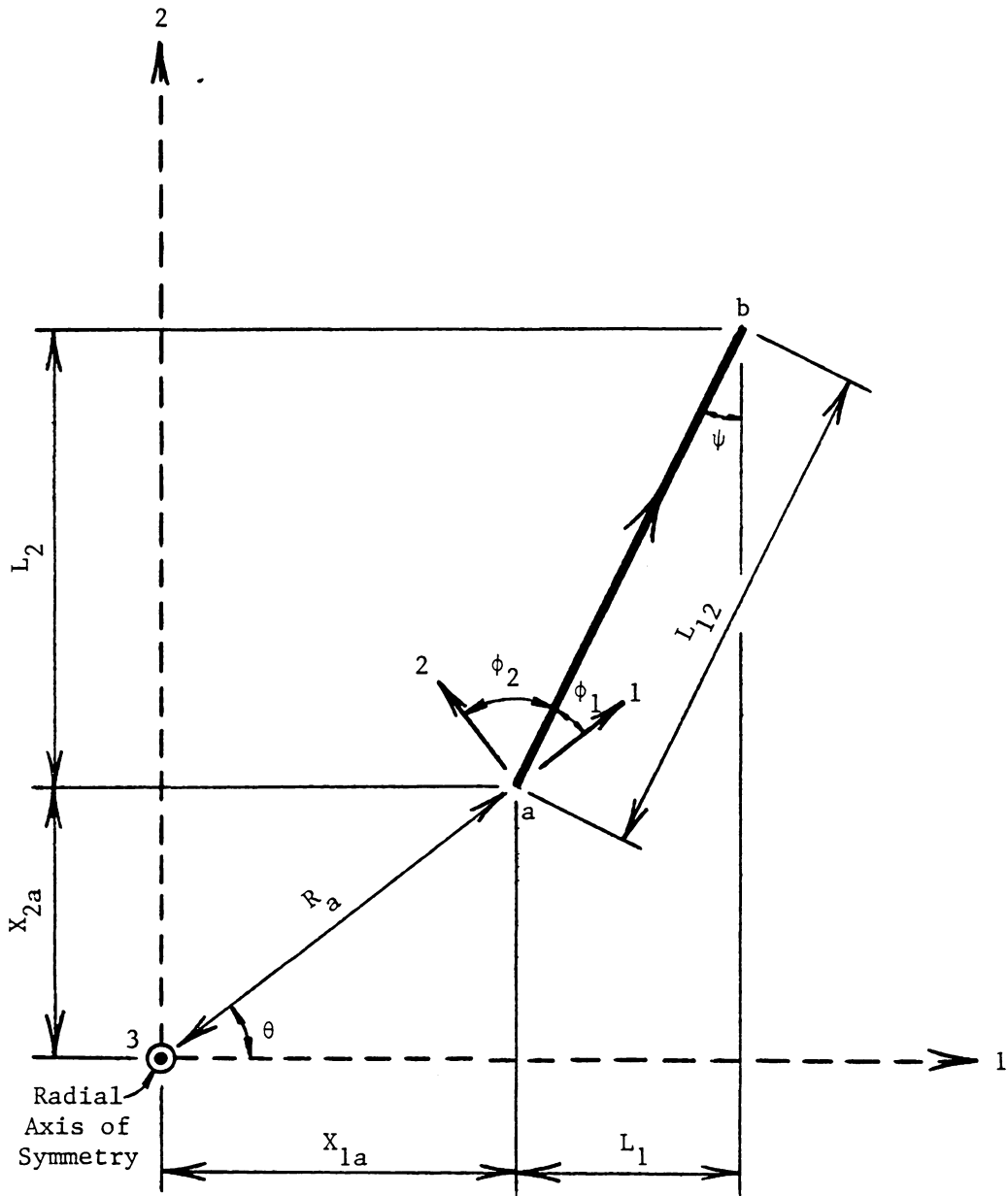


Figure 2.4: Linear Model Coordinate Transformations

$$= \frac{X_{2a}}{R_a} \left( \frac{L_2}{L} \right) + \frac{X_{1a}}{R_a} \left( \frac{L_1}{L} \right) \quad (2.9)$$

Inserting Eq. 2.6 into the appropriate locations of Eq. 2.9 and rearranging terms finally yields

$$c_1 = \frac{X_{1a}}{R_a} c_1 + \frac{X_{2a}}{R_a} c_2 \quad (2.10)$$

Similarly, by definition

$$c_2 = \frac{L_{12}}{L} \cos \phi_2 \quad (2.11)$$

and employing the angle sum relation for the cosine function yields

$$c_2 = \frac{-X_{2a}}{R_a} c_1 + \frac{X_{1a}}{R_a} c_2 \quad (2.12)$$

Since the joint and global 3-axes coincide,

$$c_3 = c_3 \quad (2.13)$$

for all elements

The computation of the joint direction cosines at the a-end can be represented in matrix form as

$$\lambda^T = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{1}{R_a} \begin{bmatrix} X_{1a} & X_{2a} & 0 \\ -X_{2a} & X_{1a} & 0 \\ 0 & 0 & R_a \end{bmatrix} \quad (2.14)$$

where

$$R_a = [X_{1a}^2 + X_{2a}^2]^{1/2} \quad (2.14a)$$

Similarly, for the b-end

$$\hat{c}_1 = \frac{X_{1b}}{R_b} c_1 + \frac{X_{2b}}{R_b} c_2 \quad (2.15)$$

$$\hat{c}_2 = \frac{-X_{2b}}{R_b} c_1 + \frac{X_{1b}}{R_b} c_2 \quad (2.16)$$

$$\hat{c}_3 = c_3 \quad (2.17)$$

or, in matrix form,

$$\hat{\lambda}^T = \begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \\ \hat{c}_3 \end{bmatrix} = \frac{1}{R_b} \begin{bmatrix} X_{1b} & X_{2b} & 0 \\ -X_{2b} & X_{1b} & 0 \\ 0 & 0 & R_b \end{bmatrix} \quad (2.18)$$

where

$$R_b = [X_{1b}^2 + X_{2b}^2]^{1/2} \quad (2.18a)$$

Inserting Eqs. 2.1b and 2.3c into Eq. 2.4a yields

$$K = \gamma \begin{bmatrix} \lambda^T \lambda & -\lambda^T \hat{\lambda} \\ -\hat{\lambda}^T \lambda & \hat{\lambda}^T \hat{\lambda} \end{bmatrix} \quad (2.19)$$

Finally, inserting Eqs 2.5, 2.13, and 2.17 into Eq. 2.19 yields the cylindrical element stiffness matrix

$$K = \begin{bmatrix} g_1 & g_2 & g_3 & -g_4 & -g_5 & -g_3 \\ & g_6 & g_7 & -g_8 & -g_9 & -g_7 \\ & & g_{10} & -g_{11} & -g_{12} & -g_{10} \\ & & & g_{13} & g_{14} & g_{11} \\ & & & & g_{15} & g_{12} \\ \text{sym.} & & & & & g_{10} \end{bmatrix} \quad (2.20)$$

where

$$\begin{aligned} g_1 &= \gamma c_1^2 & g_6 &= \gamma c_2^2 & g_{11} &= \gamma \hat{c}_1 c_3 \\ g_2 &= \gamma c_1 c_2 & g_7 &= \gamma c_2 c_3 & g_{12} &= \gamma \hat{c}_2 c_3 \\ g_3 &= \gamma c_1 c_3 & g_8 &= \gamma c_2 \hat{c}_1 & g_{13} &= \gamma \hat{c}_1^2 \\ g_4 &= \gamma c_1 \hat{c}_1 & g_9 &= \gamma c_2 \hat{c}_2 & g_{14} &= \gamma \hat{c}_1 \hat{c}_2 \\ g_5 &= \gamma c_1 \hat{c}_2 & g_{10} &= \gamma c_3^2 & g_{15} &= \gamma \hat{c}_2^2 \end{aligned}$$

In radially symmetric structures, certain regularly oriented members are frequently encountered for which significant simplifications can be made in the cylindrical element stiffness matrix.

The first type of element considered is the *radial element* which is defined as any element which is parallel to or the extension of which intersects the radial axis of symmetry except for those elements which have precisely one end point on the axis. Elements which lie entirely on the radial axis of symmetry, however, are considered to be radial elements. For radial elements, the joint coordinate systems at both ends coincide so that

$$\hat{c}_1 = c_1 \quad (2.21)$$

and there is no tangential component of the element length vector so that

$$\hat{c}_2 = c_2 = 0 \quad (2.22)$$

Elements with precisely one end point on the radial axis of symmetry do not qualify as radial elements because the orientation of the joint coordinate system at the joint on the axis is arbitrary and, therefore, the two joint coordinate systems do not necessarily coincide.

Inserting Eqs. 2.21 and 2.22 into Eq. 2.20 yields the cylindrical stiffness matrix for a radial element

$$K = \begin{bmatrix} g_1 & 0 & g_3 & -g_1 & 0 & -g_3 \\ & 0 & 0 & 0 & 0 & 0 \\ & & g_{10} & -g_3 & 0 & -g_{10} \\ & & & g_1 & 0 & g_3 \\ & & & & 0 & 0 \\ \text{sym.} & & & & & g_{10} \end{bmatrix} \quad (2.23)$$

where the  $g$  functions are defined in Eq. 2.20.

The second commonly encountered element is the *ring element* which is defined as an element which has both end points equidistant from and lying in a plane perpendicular to the radial axis of symmetry. For a ring element, the joint coordinate systems are oriented relative to each other such that

$$\hat{c}_1 = -c_1 \quad (2.24)$$

$$\hat{c}_2 = c_2 \quad (2.25)$$

$$c_3 = 0 \quad (2.26)$$

Inserting Eqs. 2.24 - 2.26 into Eq. 2.20 yields the cylindrical stiffness matrix for a ring element

$$K = \begin{bmatrix} g_1 & g_2 & 0 & g_1 & -g_2 & 0 \\ & g_6 & 0 & g_2 & -g_6 & 0 \\ & & 0 & 0 & 0 & 0 \\ & & & g_1 & -g_2 & 0 \\ & & & & g_6 & 0 \\ \text{sym.} & & & & & 0 \end{bmatrix} \quad (2.27)$$

where the  $g$  functions are defined in Eq. 2.20.

The third type of element, which is not as frequently encountered as the previous two, is the *skewed ring element*. This element is the same as a regular ring element except that the two end points do not lie in a plane perpendicular to the radial axis of symmetry. In this case, Eq. 2.26 does not hold. Inserting Eqs. 2.24 and 2.25 into Eq. 2.20 yields the cylindrical stiffness matrix for a skewed ring element.

$$K = \begin{bmatrix} g_1 & g_2 & g_3 & -g_1 & -g_2 & -g_3 \\ & g_6 & g_7 & g_2 & -g_6 & -g_7 \\ & & g_{10} & g_3 & -g_7 & -g_{10} \\ & & & g_1 & -g_2 & -g_3 \\ & & & & g_6 & g_7 \\ \text{sym.} & & & & & g_{10} \end{bmatrix} \quad (2.28)$$

where the  $g$  functions are defined in Eq. 2.20.

#### 2.4 Linear System Model

The element models are then assembled to form the system model after Holzer [5,7] using the code-number technique to add the appropriate entries of the individual element stiffness matrices into the appropriate locations of the system stiffness matrix,  $K$ . This then yields the system model

$$Kq = Q \quad (2.29)$$

where  $q$  is the generalized displacement vector and  $Q$  is the generalized load vector.  $Q$  is specified relative to cylindrical coordinates and, when solved for,  $q$  will also be in cylindrical coordinates.

The element forces are then easily calculated. Employing the code-number technique,  $D$  for each element is obtained from  $q$ .  $d$  is then calculated via Eq. 2.3. Finally,  $f$  is obtained from Eq. 2.1.

The joint forces and reactions are just as easily calculated.  $F$  is obtained via Eq. 2.3, then, using the member incidence information stored in the computer, the total force at each joint is compiled. These joint forces and reactions are also in cylindrical

coordinates.

## 2.5 Nonlinear Element Model

The geometrically nonlinear truss element model used in this study, Fig. 2.5a, is directly formulated relative to global coordinates using the principle of virtual work after Holzer [6], Holzer et al. [8], Lamma [12], and Vu [15]. Since large displacements are considered, the virtual work equations must be formulated relative to the deformed element configuration.

The virtual work associated with a virtual displacement is the sum of the external and internal work which must vanish at an equilibrium configuration

$$\delta W = \delta W_e + \delta W_i = 0 \quad (2.30)$$

The internal work can be expressed as negative the change in internal strain energy,  $U$ , which yields, from Eq. 2.30

$$\delta W = \delta W_e - \delta U = 0 \quad (2.31)$$

Referring to Fig. 1.5b, the element displacement can be broken into two components--a rigid body translation and a rotation with axial deflection about the a-end. Since, for a system in equilibrium, no work is required to accomplish a rigid body translation, the generalized displacement vector can be taken as  $u$  where

$$u = D_b - D_a \quad (2.32)$$

and the virtual displacements need only consist of variations at

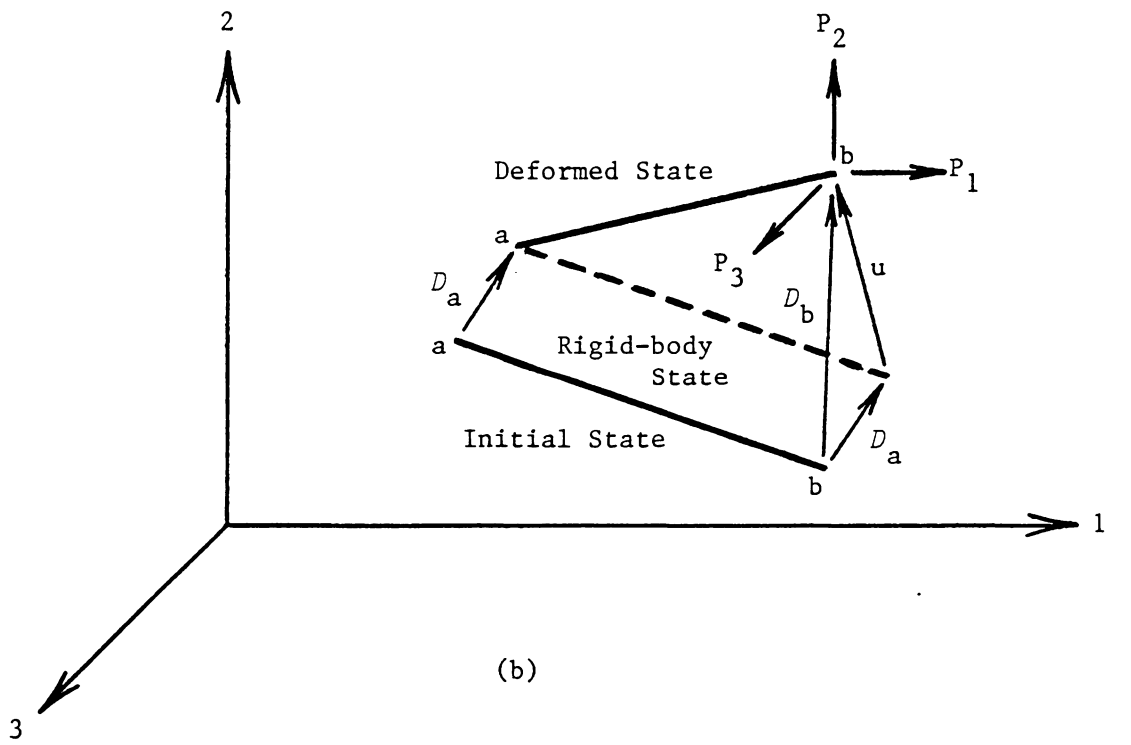
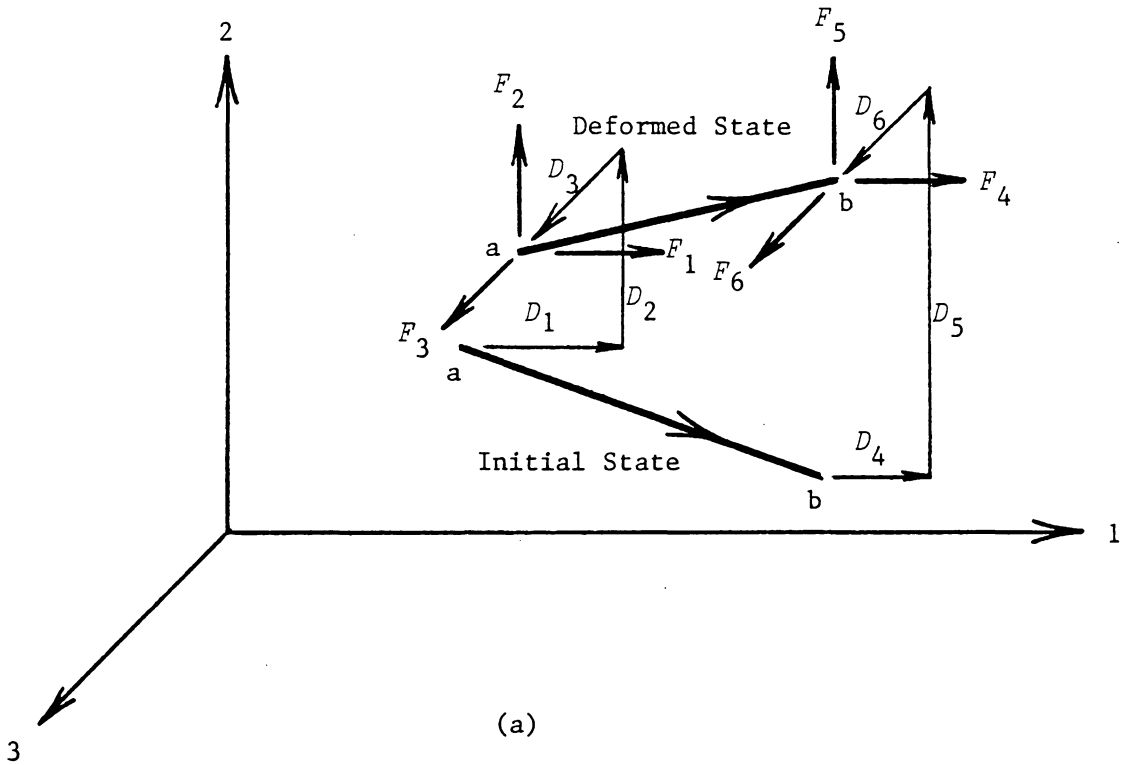


Figure 2.5: Nonlinear Truss Element

the b-end,  $\delta u_i$ ,  $i = 1, 2, 3$ . The external work can therefore be expressed as

$$\delta W_e = \sum_{i=1}^3 P_i \delta u_i \quad (2.33)$$

and the variation in internal strain energy as

$$\delta U = \sum_{i=1}^3 \frac{\partial U}{\partial u_i} \delta u_i \quad (2.34)$$

where  $P_i$  is the force in the  $i$ -direction at the b-end required to equilibrate the deformed state. The total virtual work associated with a virtual displacement can then be expressed as

$$\delta W = \sum_{i=1}^3 \left( P_i - \frac{\partial U}{\partial u_i} \right) \delta u_i = 0 \quad (2.35)$$

Since virtual displacements are arbitrary, for Eq. 2.35 to be true, the quantity in parentheses must equal zero for all  $i$ , yielding

$$P_i = \frac{\partial U}{\partial u_i}, \quad i = 1, 2, 3 \quad (2.36)$$

To trace a nonlinear equilibrium path requires that, given one equilibrium point, a load-displacement increment be taken yielding an approximation to the next equilibrium point. An iterative procedure is then used to resolve the force imbalances arising from the approximation and to converge on a new equilibrium point. The procedure is then repeated until a desired portion of the equilibrium

has been traced. To formulate the incremental load-displacement relation, a Taylor series expansion of  $P_i$  at a displacement  $u + \Delta u$  is taken, yielding

$$P_i(u + \Delta u) = P_i(u) + \sum_{j=1}^3 \frac{\partial P_i}{\partial u_j} \Delta u_j + \dots \quad (2.37)$$

Rearranging terms and neglecting those with higher order derivatives yields an approximate expression for the incremental force,  $\Delta P_i$ , associated with an incremental displacement,  $\Delta u$ ,

$$\Delta P_i = P_i(u + \Delta u) - P_i(u) \approx \sum_{j=1}^3 \frac{\partial P_i}{\partial u_j} \Delta u_j \quad (2.38)$$

Inserting Eq. 2.36 into Eq. 2.38 yields

$$\Delta P_i \approx \sum_{j=1}^3 \frac{\partial^2 U}{\partial u_i \partial u_j} \Delta u_j, \quad i = 1, 2, 3 \quad (2.39)$$

Eq. 2.39 can be expressed in matrix form as

$$\Delta P \approx \hat{K} \Delta u \quad (2.40)$$

where

$$\Delta P = \begin{bmatrix} \Delta P_1 \\ \Delta P_2 \\ \Delta P_3 \end{bmatrix} \quad (2.40a)$$

$$\Delta u = \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \Delta u_3 \end{bmatrix} \quad (2.40b)$$

$$\hat{K} = [\hat{K}_{ij}] = \left[ \frac{\partial^2 U}{\partial u_i \partial u_j} \right] \quad (2.40c)$$

Formulating an expression for U from the geometric and material properties of the element and assuming that Hooke's Law applies yields the terms of  $\hat{K}$  for a general element

$$\hat{K} = \begin{bmatrix} h_1 & h_2 & h_4 \\ & h_3 & h_5 \\ \text{sym.} & & h_6 \end{bmatrix} \quad (2.41)$$

where

$$h_1 = \gamma \left[ c_1^{*2} + \frac{e}{L^*} (1 - c_1^{*2}) \right]$$

$$h_2 = \gamma \left[ c_1^* c_2^* \left( 1 - \frac{e}{L^*} \right) \right]$$

$$h_3 = \gamma \left[ c_2^{*2} + \frac{e}{L^*} (1 - c_2^{*2}) \right]$$

$$h_4 = \gamma \left[ c_1^* c_3^* \left( 1 - \frac{e}{L^*} \right) \right]$$

$$h_5 = \gamma \left[ c_2^* c_3^* \left( 1 - \frac{e}{L^*} \right) \right]$$

$$h_6 = \gamma \left[ c_3^{*2} + \frac{e}{L^*} (1 - c_3^{*2}) \right]$$

$$\gamma = \frac{AE}{L}$$

L = undeformed element length

$L^*$  = deformed element length

$e = L^* - L$  = element elongation

$$c_i^* = \frac{L_i^*}{L_i}, \quad i = 1, 2, 3$$

= direction cosine of the element local 1-axis in the deformed state relative to the global i-axis

$L_i^*$  = component of the deformed element length vector in the global i-direction

To formulate the incremental force-displacement relation for the entire element, the transformation matrix,  $C$ , is defined as

$$C = [-I \quad I] \quad (2.42)$$

where  $I$  is the identity matrix of order 3. The relationship between  $\Delta u$  and  $\Delta D$  (an incremental version of Eq. 2.32) may then be expressed as

$$\Delta u = C \Delta D \quad (2.43)$$

where

$$\Delta D = \begin{bmatrix} \Delta D_a \\ \Delta D_b \end{bmatrix} \quad (2.43a)$$

To preserve equilibrium of the deformed element, the relationship between  $\Delta P$  and the incremental total element force vector,  $\Delta F$ , may be expressed as

$$\Delta F = C^T \Delta P \quad (2.44)$$

Inserting Eqs. 2.43 and 2.44 into Eq. 2.40 yields

$$\Delta F = K \Delta D \quad (2.45)$$

where

$$K = C^T \hat{K} C = \begin{bmatrix} \hat{K} & -\hat{K} \\ -\hat{K} & \hat{K} \end{bmatrix} \quad (2.45a)$$

$K$  is called the *global element tangent stiffness matrix* and will be used to trace the geometrically nonlinear equilibrium path of space trusses.

Since the element tangent stiffness matrix was formulated directly relative to global coordinates, the coordinate transformation used in the nonlinear model must be from global to cylindrical coordinates rather than from local to cylindrical as was used in the formulation of the linear model. To accomplish this transformation, the transformation matrix,  $\Lambda$ , is defined such that

$$\Delta D = \Lambda \Delta D \quad (2.46)$$

$$\Delta F = \Lambda^T \Delta F$$

where  $\Delta D$  and  $\Delta F$  are the element nodal displacement and force vectors relative to cylindrical coordinates. Note that  $\Lambda$  in this case differs from that used in the formulation of the linear model.

$\Lambda$  can be partitioned thus

$$\Lambda = \begin{bmatrix} \lambda & 0 \\ 0 & \hat{\lambda} \end{bmatrix} \quad (2.47)$$

where  $\lambda$  and  $\hat{\lambda}$  perform the transformation between global and cylindrical coordinates at the a- and b-ends of the element respectively.

$\lambda$  and  $\hat{\lambda}$  are defined as

$$\lambda = \begin{bmatrix} c_1 & c_2 & 0 \\ -c_2 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.48)$$

$$\hat{\lambda} = \begin{bmatrix} \hat{c}_1 & \hat{c}_2 & 0 \\ -\hat{c}_2 & \hat{c}_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where  $c_i$  and  $\hat{c}_i$  are the direction cosines of the joint 1-axis at the a- and b-ends of the element, respectively, relative to the global i-axis.

Referring to Fig. 2.6, which is the projection of an arbitrary joint coordinate system onto the global 1,2-plane, expressions for  $c_i$  and  $\hat{c}_i$ ,  $i = 1, 2$ , are easily formulated. The formulation which follows assumes that the joint coordinate system shown lies at the a-end of the element being considered. By definition

$$c_1 = \cos \phi_1 = \frac{X_1}{R} \quad (2.49)$$

$$c_2 = \cos \phi_2 = \frac{X_2}{R}$$

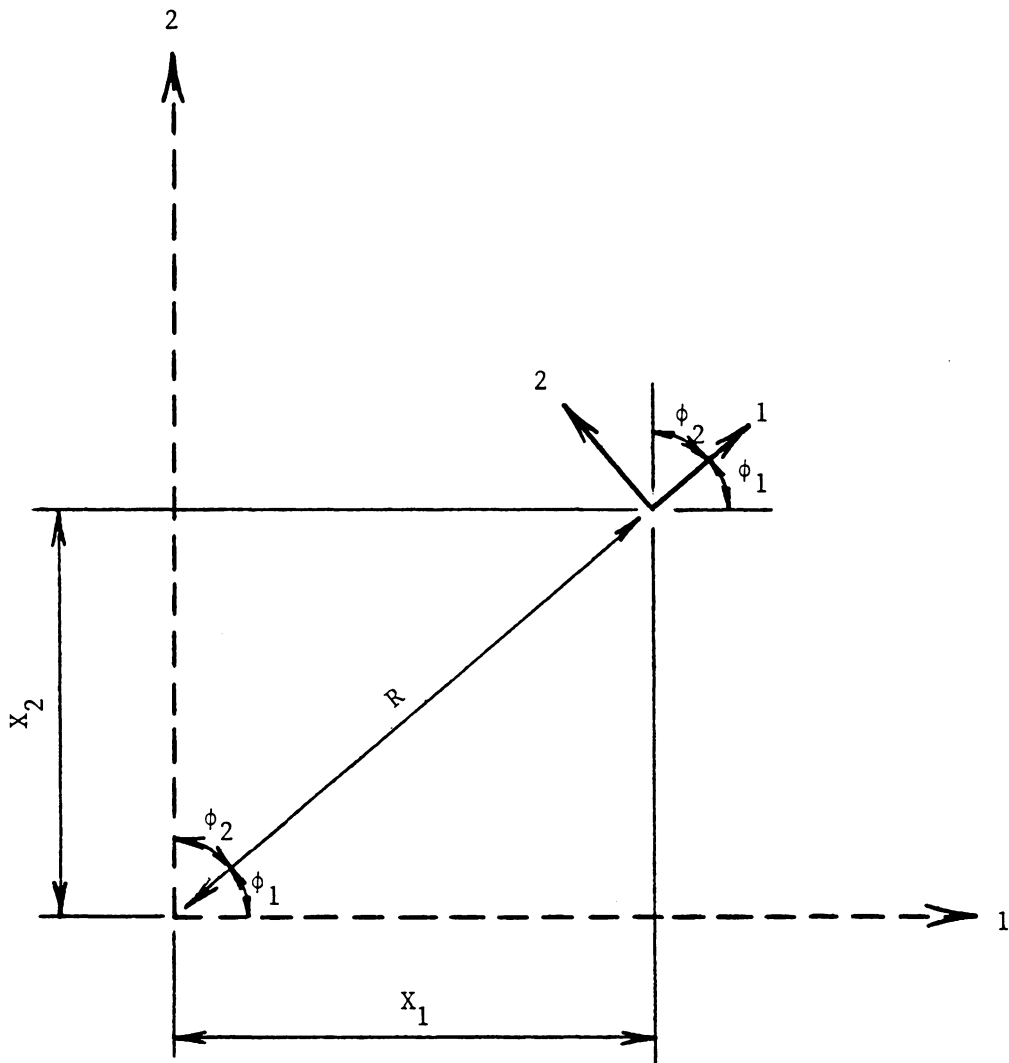


Figure 2.6: Nonlinear Model Coordinate Transformations

where

$$R = [X_1^2 + X_2^2]^{1/2} \quad (2.49a)$$

When the joint coordinate system lies at the b-end, the formulation is the same except that  $\hat{c}_i$  is substituted for  $c_i$  in Eq. 2.49. When the direction cosines are actually calculated in the computer, even though more than one element may be incident at a particular joint, the calculation is done only once for each joint and the general direction cosines are stored in vectors of dimension equal to the number of joints. These cosines are then accessed through the member incidence information stored in the computer as either  $c_i$  or  $\hat{c}_i$  and used in the individual element coordinate transformations.

Inserting Eqs. 2.46 into Eq. 2.45 yields the incremental force-displacement relation relative to cylindrical coordinates

$$\Delta F = K\Delta D \quad (2.50)$$

where  $\Delta F$  and  $\Delta D$  are the cylindrical incremental element force and displacement vectors respectively and

$$K = \Lambda K \Lambda^T \quad (2.50a)$$

$K$  is the *cylindrical element tangent stiffness matrix*. Incorporating Eqs. 2.41, 2.45a, 2.47, and 2.48 into Eq. 2.50a yields

$$K = \begin{bmatrix} H_1 & H_2 & H_3 & H_4 & H_5 & -H_3 \\ & H_6 & H_7 & H_8 & H_9 & -H_7 \\ & & H_{10} & H_{11} & H_{12} & -H_{10} \\ & & & H_{13} & H_{14} & -H_{11} \\ & & & & H_{15} & -H_{12} \\ \text{sym.} & & & & & H_{10} \end{bmatrix} \quad (2.51)$$

where

$$H_1 = c_1^2 h_1 + 2c_1 c_2 h_2 + c_2^2 h_3$$

$$H_2 = -c_1 c_2 h_1 + (c_1^2 - c_2^2) h_2 + c_1 c_2 h_3$$

$$H_3 = c_1 h_4 + c_2 h_5$$

$$H_4 = -c_1 \hat{c}_1 h_1 - (c_1 \hat{c}_2 + c_2 \hat{c}_1) h_2 - c_2 \hat{c}_2 h_3$$

$$H_5 = c_1 \hat{c}_2 h_1 - (c_1 \hat{c}_1 - c_2 \hat{c}_2) h_2 - c_2 \hat{c}_1 h_3$$

$$H_6 = c_2^2 h_1 - 2c_1 c_2 h_2 + c_1^2 h_3$$

$$H_7 = -c_2 h_4 + c_1 h_5$$

$$H_8 = c_2 \hat{c}_1 h_1 + (c_2 \hat{c}_2 - c_1 \hat{c}_1) h_2 - c_1 \hat{c}_2 h_3$$

$$H_9 = -c_2 \hat{c}_2 h_1 + (c_2 \hat{c}_1 + c_1 \hat{c}_2) h_2 - c_1 \hat{c}_1 h_3$$

$$H_{10} = h_6$$

$$H_{11} = -\hat{c}_1 h_4 - \hat{c}_2 h_5$$

$$H_{12} = \hat{c}_2 h_4 - \hat{c}_1 h_5$$

$$H_{13} = \hat{c}_1^2 h_1 + 2\hat{c}_1 \hat{c}_2 h_2 + \hat{c}_2^2 h_3$$

$$H_{14} = -\hat{c}_1 \hat{c}_2 h_1 + (\hat{c}_1^2 - \hat{c}_2^2) h_2 + \hat{c}_1 \hat{c}_2 h_3$$

$$H_{15} = \hat{c}_2^2 h_1 - 2\hat{c}_1 \hat{c}_2 h_2 + \hat{c}_1^2 h_3$$

and  $h_i$ ,  $i = 1, 2, \dots, 6$ , are defined in Eq. 2.41.

## 2.6 Nonlinear System Model

The cylindrical system tangent stiffness matrix is assembled from the individual cylindrical element tangent stiffness matrices using the code-number technique after Holzer [5,7]. A series of points on the nonlinear equilibrium path can then be generated after Holzer et al. [8] and Vu [15] using one of several methods. These methods are 1) the Newton/Raphson method, 2) the Riks/Wempner method, and 3) the modified Riks/Wempner method. Although the program listed in Appendix C is capable of utilizing all three of these methods, since the modified Riks/Wempner method without updating has been shown to be the most efficient scheme (see Holzer et al. [8] and Vu [15]), it has been used exclusively in this study.

## 2.7 Stability of Equilibrium

For a conservative n-degree-of-freedom system, the total potential energy function can be defined after Holzer et al. [8] as

$$V(q, \lambda) = \Omega(q, \lambda) + U(q) \quad (2.52)$$

where  $q$  is the generalized displacement vector,  $\lambda$  is the loading parameter,  $\Omega$  is the potential energy of the external loads, and  $U$  is the internal strain energy. If the loads are configuration independent, the stability of equilibrium can be determined from the second variation of  $V$ . Specifically

$$\delta^2 V(q, \lambda) = \sum_{i=1}^n \sum_{j=1}^n \frac{\partial^2 U}{\partial q_i \partial q_j} \delta q_i \delta q_j \quad (2.53)$$

which can be expressed in matrix form as

$$\delta^2 V = \delta q^T K \delta q \quad (2.54)$$

where  $K$  is the cylindrical system tangent stiffness matrix. Equilibrium is stable when  $\delta^2 V > 0$ , critical when  $\delta^2 V = 0$ , and unstable when  $\delta^2 V < 0$ . These three conditions correspond to  $K$  being positive definite, positive semidefinite, and indefinite, negative semidefinite, or negative definite respectively. Correspondingly, equilibrium is stable when all eigenvalues of  $K$  are positive, critical when at least one eigenvalue is zero, and unstable when at least one eigenvalue is negative. A critical point can be either a bifurcation point or a limit point. When, at a critical point, more than one eigenvalue vanishes, the critical point is called "coincident". Otherwise, it is called "simple".

## 2.8 Solution of Systems of Linear Algebraic Equations

If a symmetric matrix,  $K$ , is positive definite, there exists a unique factorization (see Vu [15])

$$K = LDL^T \quad (2.55)$$

where  $L$  is a unit lower triangular matrix and  $D$  is a positive diagonal matrix. The half-band width of  $L$  is the same as the half-band width of  $K$ , therefore, no additional storage is required to perform this decomposition. A system of linear algebraic equations of the form

$$Kq = Q \quad (2.56)$$

can then be solved using a three-step procedure as follows:

1. Solve  $Lx = Q$  for  $x$
2. Solve  $Dy = x$  for  $y$
3. Solve  $L^T q = y$  for  $q$

Note that all three steps can be executed with very little computational effort and that the  $LDL^T$  decomposition is not destroyed in the process, allowing later solution of subsequent systems of equations having the same coefficient matrix. This is especially useful in the solution of linear problems where the results of several loading conditions are desired and in the solution of nonlinear problems where the tangent stiffness matrix is not updated with each iteration.

When tracing a nonlinear equilibrium path into the unstable region, however, the tangent stiffness matrix is not always positive definite. In this case, the  $LDL^T$  decomposition discussed previously may not exist or may be numerically unstable. Therefore, an unconditionally stable decomposition of  $K$  is required. In this study,

the LINPACK equation solver [3] for symmetric indefinite matrices was used. This solver decomposes  $K$  to the form

$$K = UDU^T \quad (2.57)$$

where  $D$  is a block diagonal matrix with blocks of order 1 or 2 and  $U$  is the product of elementary unit upper triangular and permutation matrices. Regretfully, LINPACK does not have a routine capable of exploiting both bandedness and symmetry of an indefinite matrix simultaneously. Therefore, with larger problems, the equation solver used with the linear model was also used with the nonlinear model unless difficulties arose which could be attributed to instability of the  $LDL^T$  decomposition.

## CHAPTER 3

### DISCUSSION OF RESULTS

Linear and nonlinear analyses were performed on each of two trusses. The results of the nonlinear analysis were compared to those in refs. 12 and 15. The results of the linear analysis were compared to those obtained using a program previously written and verified by the author. This earlier program operated solely in global coordinates and has been incorporated into the linear analysis program as an option.

When analyzing "pie slices" using either the linear or nonlinear program, the member cross sections and joint loads were altered, where necessary, per sec. 2.2.

#### 3.1 Linear Analysis

The computer program used in the linear analysis was written by the author and is capable of operating in either global or cylindrical coordinates. A program listing and users' guide can be found in Appendix B.

##### 3.1.1 Structure 1

The first structure analyzed was a 24-bar lamella dome (see Fig. 3.1). All element cross-sectional areas were taken as  $1.0 \text{ cm}^2$  and the modulus of elasticity as  $2,110,000 \text{ g/cm}^2$ . The loading condition consisted of a 1000 g downward load at each free joint.

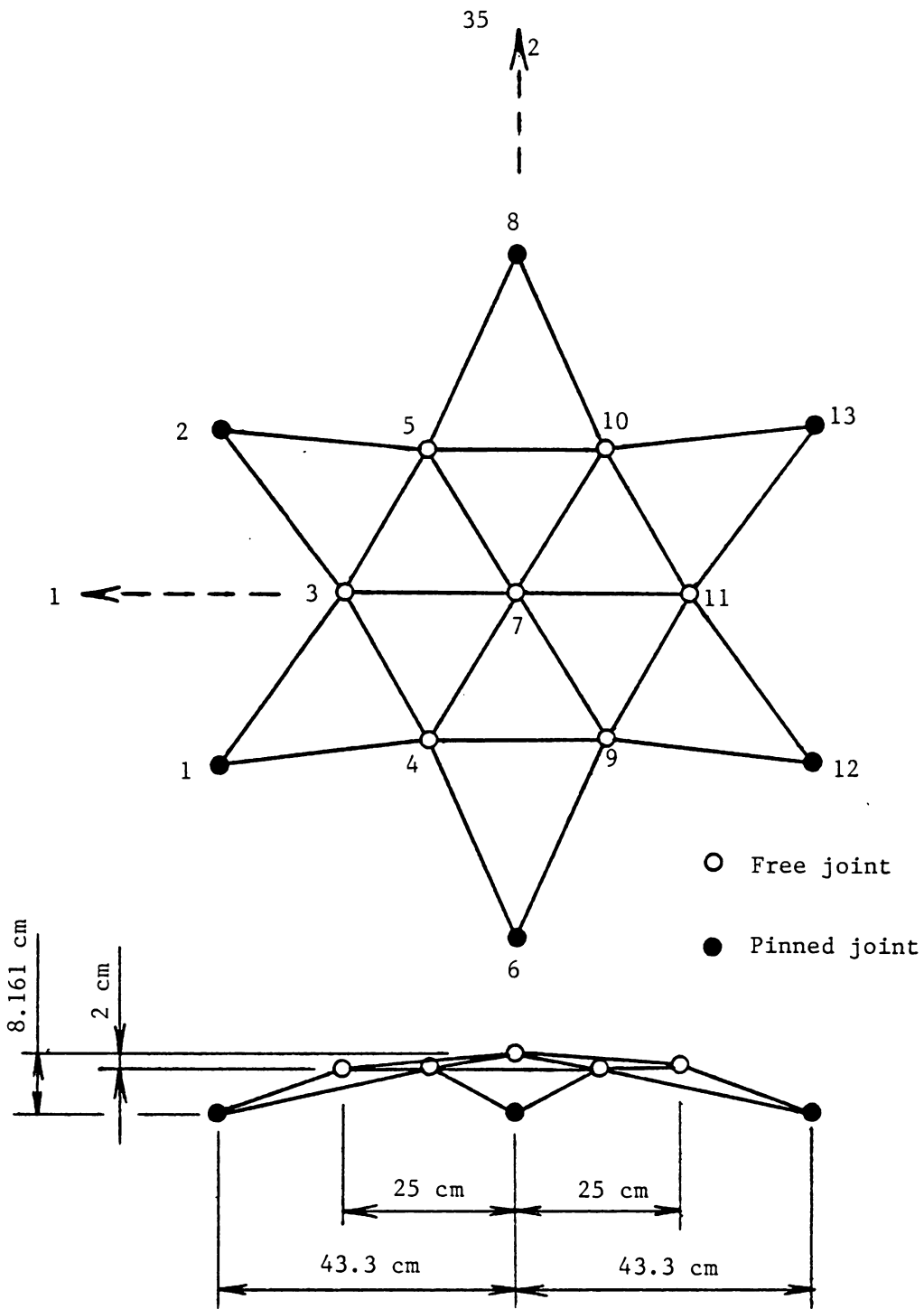


Figure 3.1: Structure 1

The structure and loading condition exhibit  $C_6$  radial symmetry and, therefore, five "pie slices" equal to  $\frac{1}{6}, \frac{2}{6}, \dots, \frac{5}{6}$  of the structure (see Fig. 3.2) were analyzed.

All "pie slice" analyses yielded results identical to those obtained from analysis of the entire structure.

### 3.1.2 Structure 2

The second structure analyzed was a 70-bar lamella dome with 70 degrees of freedom (see Fig. 3.3). All joints lie on a spherical cap of 1200 in. radius with the first ring  $10^\circ$  from the radial axis of symmetry and the second ring  $20^\circ$ . All element cross-sectional areas were taken as  $1.0 \text{ in.}^2$  and the modulus of elasticity as 30,000 ksi. All joints on the outer ring were constrained in the 3-direction, joint 16 was constrained in the 1- and 2-directions, and joint 31 was constrained in the 2-direction. The loading condition consisted of downward 1.0 kip loads at joints 8, 9, 12, 13, 15, 16, 17, 19, 20, 23, and 24.

The structure and loading condition exhibit  $C_{10}$  radial symmetry and, therefore, nine "pie slices" equal to  $\frac{1}{10}, \frac{2}{10}, \dots, \frac{9}{10}$  of the structure were analyzed.

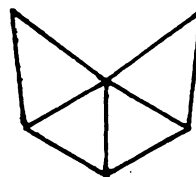
As with the 24-bar dome, all "pie slice" analyses yielded results identical to those obtained from analysis of the entire structure.

### 3.1.3 Comparison of Effort

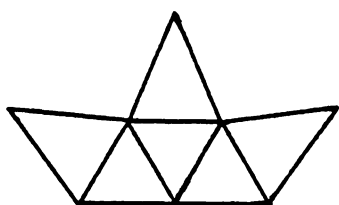
A comparison of the relative effort required to analyze a "pie slice" of a structure vs. the effort required to analyze the



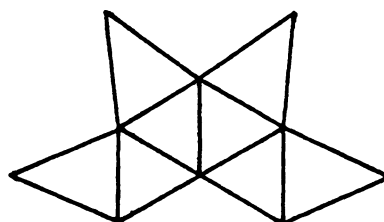
(a)



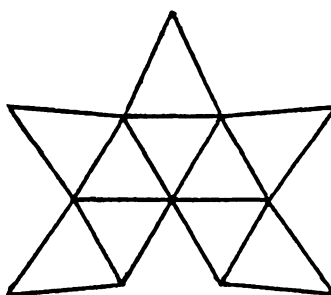
(b)



(c)



(d)



(e)

Figure 3.2: "Pie Slices" of Structure 1

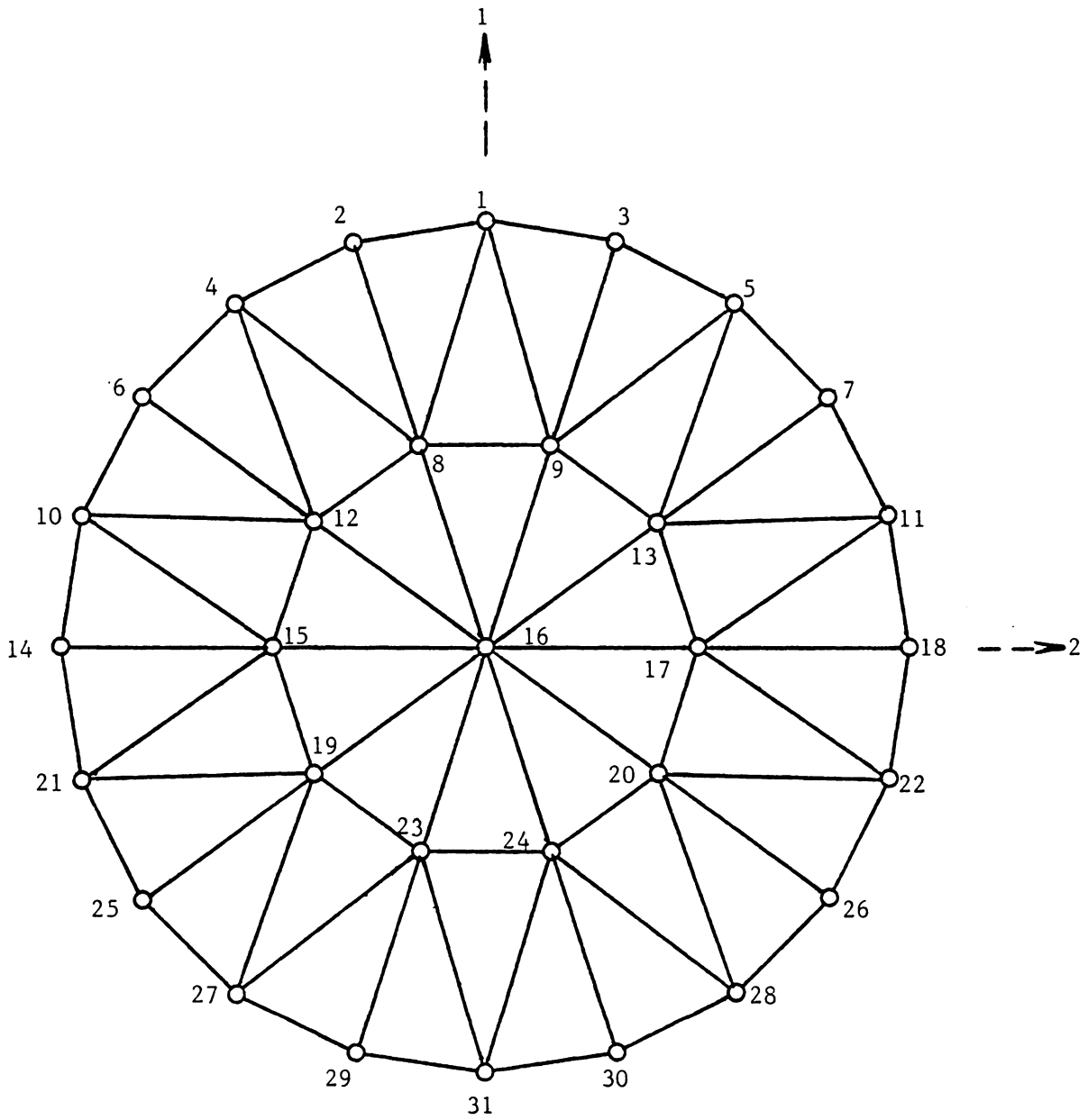


Figure 3.3: Structure 2

entire structure was made for structures 1 and 2. Acknowledging that certain FORTRAN statements require more execution time than others, the number of statements executed in performing an analysis was, nevertheless, used as the criterion for comparison of computational effort. For a structure exhibiting  $C_n$  symmetry, a "pie slice",  $n$  of which would form the whole structure, was considered to represent  $\frac{1}{n} \times 100$  percent of the structure. Consequently,  $m$  of these "pie slices" combined to form a larger "pie slice" were considered to represent  $\frac{m}{n} \times 100$  percent of the structure. Since, when analyzing a "pie slice", the joint lying on the radial axis of symmetry must be constrained in the 1- and 2-directions, for purposes of comparison of effort, this joint was likewise constrained when analyzing the entire structure.

The number of statements executed for the various portions of structures 1 and 2 are displayed in Table 3.1. The percent of computational effort is defined as the ratio of the number of statements executed in the analysis of a "pie slice" to the number executed in the analysis of the entire structure times one hundred. The half-band width and the number of degrees of freedom (D.O.F.) are included for reference. A "wave front" scheme [5] was used in numbering the joints. When analyzing a "pie slice", the "wave front" was taken to move in the direction of the axis of bisymmetry.

Fig. 3.4 is a plot of the data in Table 3.1. As can be seen, for the two structures analyzed, the percent reduction in the amount of effort required was roughly equal to the percent reduction in the size of the structure.

TABLE 3.1  
 COMPUTATIONAL EFFORT REQUIRED IN LINEAR ANALYSIS

<u>24-Bar Dome</u>				
% of Structure	No. of D.O.F.	Half-Band Width	Statements Executed	% of Effort
100	19	10	9965	100.0
83.3	17	10	8541	85.7
66.7	14	10	6684	67.1
50	11	9	4842	48.6
33.3	8	8	3380	33.9
16.7	5	5	1641	16.5
<u>70-Bar Dome</u>				
% of Structure	No. of D.O.F.	Half-Band Width	Statements Executed	% of Effort
100	70	22	64211	100.0
90	65	22	59207	92.2
80	58	22	51409	80.1
70	51	22	44681	69.6
60	44	22	37792	58.9
50	37	22	31637	49.3
40	30	19	22295	34.7
30	23	18	15582	24.3
20	16	11	7694	12.0
10	9	8	3484	5.4

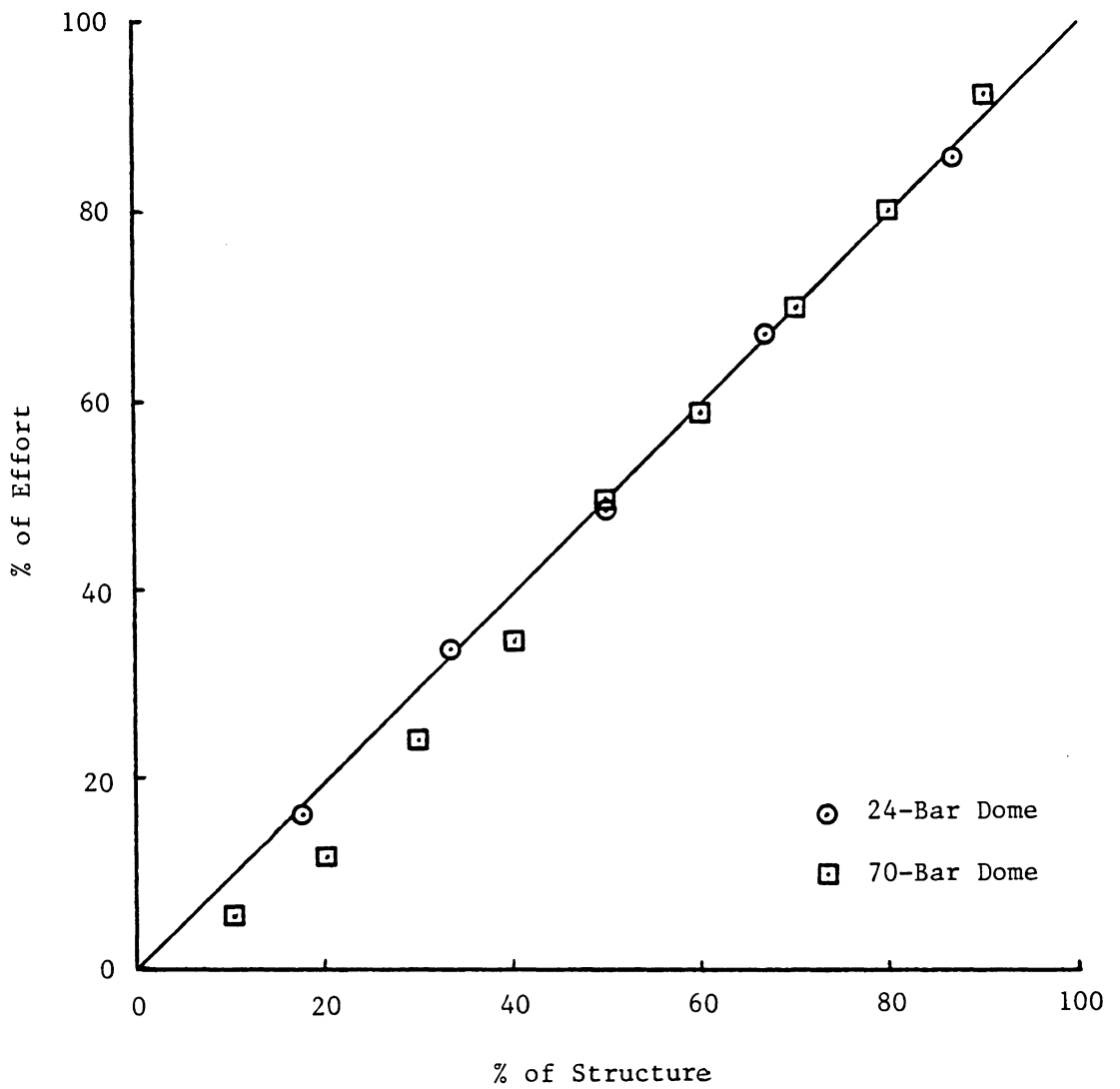


Figure 3.4: Percent of Effort vs. Percent of Structure

### 3.2 Nonlinear Analysis

The computer program used in the nonlinear analysis is an extension by the author of a program written by Lamma [12]. The program is capable of tracing the nonlinear fundamental equilibrium path of an arbitrary truss structure with arbitrary loads, converging on bifurcation and limit points, and, in many cases, branching off of the fundamental equilibrium path onto a bifurcation path. These varying tasks may be performed in either global or cylindrical coordinates. Either of two equation solvers may be used in tracing equilibrium paths. The first is a very economical solver after Cook [2] which exploits both bandedness and symmetry of the tangent stiffness matrix, and the second is the less efficient LINPACK solution routine for symmetric indefinite systems of equations. A program listing and users' guide can be found in Appendix C.

#### 3.2.1 Structure 1

The first structure analyzed was the 19 degree of freedom, 24-bar lamella dome analyzed linearly in sec. 3.1.1 (see Fig. 3.1). This structure was chosen because it has been extensively analyzed in previous papers (see refs. 12 and 15) and a well-established basis for comparison of results exists.

All members have the same cross-sectional area and modulus of elasticity. The loading condition,  $Q$ , is expressed as

$$Q = \lambda \bar{Q} \quad (3.1)$$

where  $\bar{Q}$  is the generalized loading vector and  $\lambda$  is a scaling factor.

When incrementing the load while tracing the equilibrium path,  $\bar{Q}$  is held constant while  $\lambda$  is incremented. Load magnitudes for both structures analyzed in this study are expressed nondimensionally in terms of  $\bar{\lambda}$  where

$$\bar{\lambda} = \frac{\lambda \times 10^4}{AE} \quad (3.2)$$

The generalized loading condition consisted of downward unit loads at joints 3, 4, 5, 9, 10, and 11.

Tracing the fundamental nonlinear equilibrium path of the entire structure located three bifurcation points before reaching the first limit point. These three points occurred at values of  $\bar{\lambda}$  equal to 7.592, 9.359, and 15.520 which are verified within a reasonable computational error by ref. 12 (see Fig 3.5). Branching onto the bifurcation paths at the first and second bifurcation points revealed that the buckling mode exhibited  $C_3$  radial symmetry at the first bifurcation point and  $C_2$  radial symmetry at the second. The program was unable to converge on a bifurcation path at the third bifurcation point. The buckling configuration at the first bifurcation point consisted of a buckling of the compression ring with joints 3, 9, and 10 moving together as one group, and joints 4, 5, and 11 as another. This differs from the buckling mode described by Lamma [12] for this bifurcation point. The buckling configuration at the second bifurcation point consisted of a "tilting" of the compression ring around the global 1-axis.

Tracing the fundamental equilibrium path of a "pie slice" equal to one sixth of structure 1 located only one bifurcation

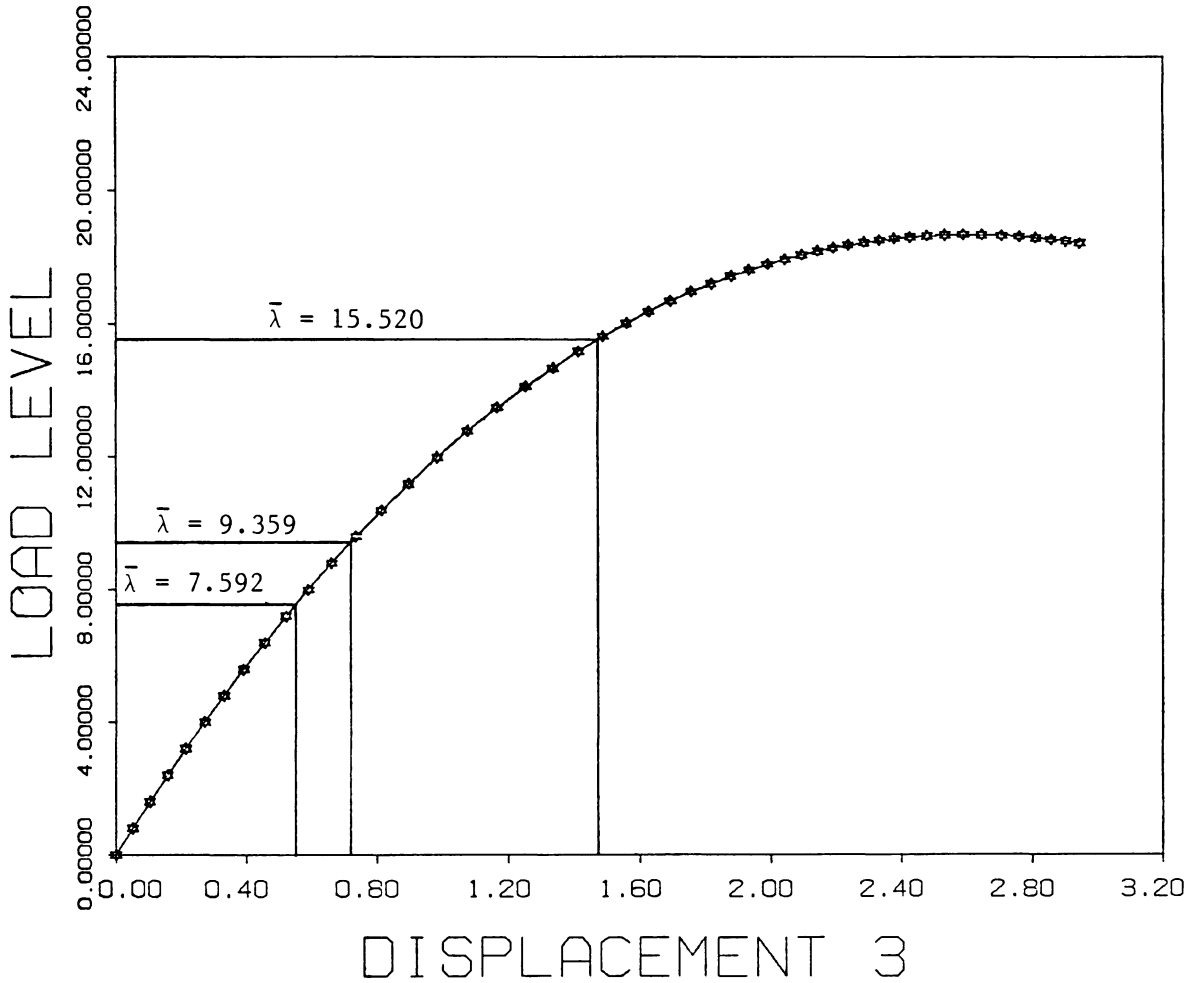


Figure 3.5: Nonlinear load vs. vertical displacement of compression ring curve for structure 1 showing locations of bifurcation points.

point before reaching the limit point. This bifurcation point duplicates the first bifurcation point found by analysis of the entire structure. Branching onto the bifurcation path of the "pie slice" also duplicated the buckling mode revealed by analysis of the whole structure, with the single ring element "tilting" to one side. When extrapolating this buckling configuration around the entire structure, alternating "pie slices" tip in opposite directions to produce the overall "zig-zag" buckled configuration for the compression ring (see Fig. 3.6).

Tracing the fundamental equilibrium path of a "pie slice" equal to two sixths of the structure located two bifurcation points before reaching the limit point. The first bifurcation point, as expected, again duplicates the first bifurcation point found by analysis of the entire structure. The second bifurcation point, however, occurred at  $\bar{\lambda} \approx 11.800$  where none had been encountered in the two previous analyses. Although the program was unable to converge on the associated bifurcation path, the author strongly suspects that the buckling configuration consists of the entire "pie slice" tipping to one side. Since an odd number of these "pie slices" would be required to form the entire structure, this buckling mode would, therefore, be inhibited in an analysis of the entire structure.

A comparison of the computational effort required to analyze "pie slices" vs. the effort to analyze the entire structure was also made with the nonlinear model. However, due to high computing costs and difficulties which arose in the analysis of structure 2,

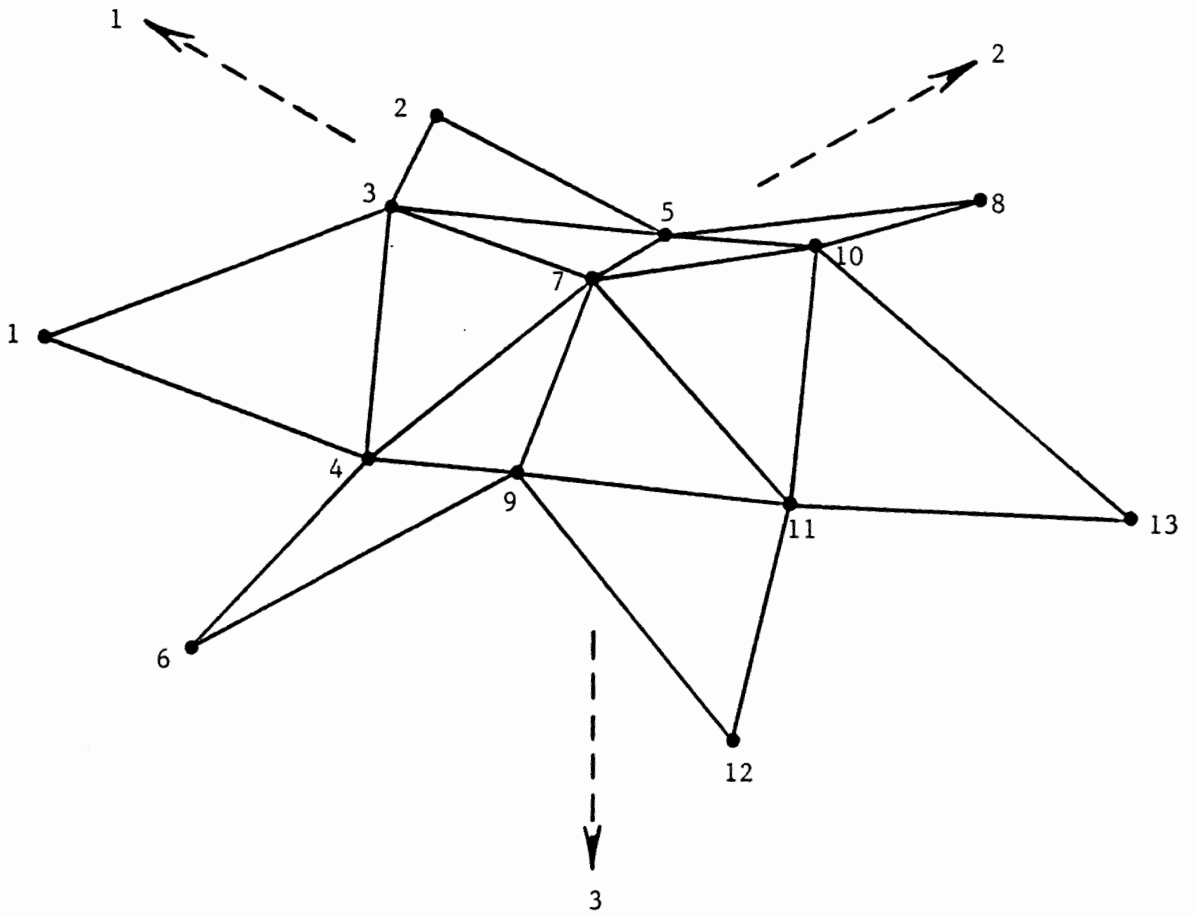


Figure 3.6: Buckled Configuration of Structure 1

the comparison was made only with the 24-bar dome. The fundamental equilibrium paths of the entire structure and of five "pie slices" equal to  $\frac{1}{6}$ ,  $\frac{2}{6}$ , ...,  $\frac{5}{6}$  of the structure were traced until the first bifurcation point was passed at  $\bar{\lambda} = 7.592$ . The initial load increment,  $\Delta\bar{\lambda}_0$ , was set at 0.800 and the tangent vector was scaled per refs. 8, 12, and 15 for four iterations. The determinant of the undeformed tangent stiffness matrix was provided before execution began. The results of this comparison are displayed in Table 3.2 and Fig. 3.7. The LINPACK equation solver was used to trace the equilibrium paths, so the half-band widths and the joint numbering scheme are of no consequence. Again, the percent reduction in the computational effort required was roughly equal to the percent reduction in the size of the structure. Note, however, that the data points may suggest a slightly parabolic relation between effort and size of structure. This is probably because, in a nonlinear analysis, a greater portion of the total computational effort is expended in solving systems of simultaneous equations than is likewise expended in a linear analysis, and that there is generally a second order relation between the size of a system of equations and the number of operations required to solve it [10].

### 3.2.2 Structure 2

The second structure analyzed was the 70 degree of freedom, 70-bar lamella dome analyzed in sec. 3.1.2 (see Fig. 3.3). All members have the same cross-sectional area and modulus of elasticity. The loading condition,  $Q$ , is as expressed in Eq. 3.1 and load mag-

TABLE 3.2  
 COMPUTATIONAL EFFORT REQUIRED IN NONLINEAR ANALYSIS

---

<u>24-Bar Dome</u>			
% of Structure	No. of D.O.F.	Statements Executed	% of Effort
100	19	431684	100.0
83.3	17	353177	81.8
66.7	14	275593	63.8
50	11	198041	45.9
33.3	8	128609	29.8
16.7	5	69633	16.1

---

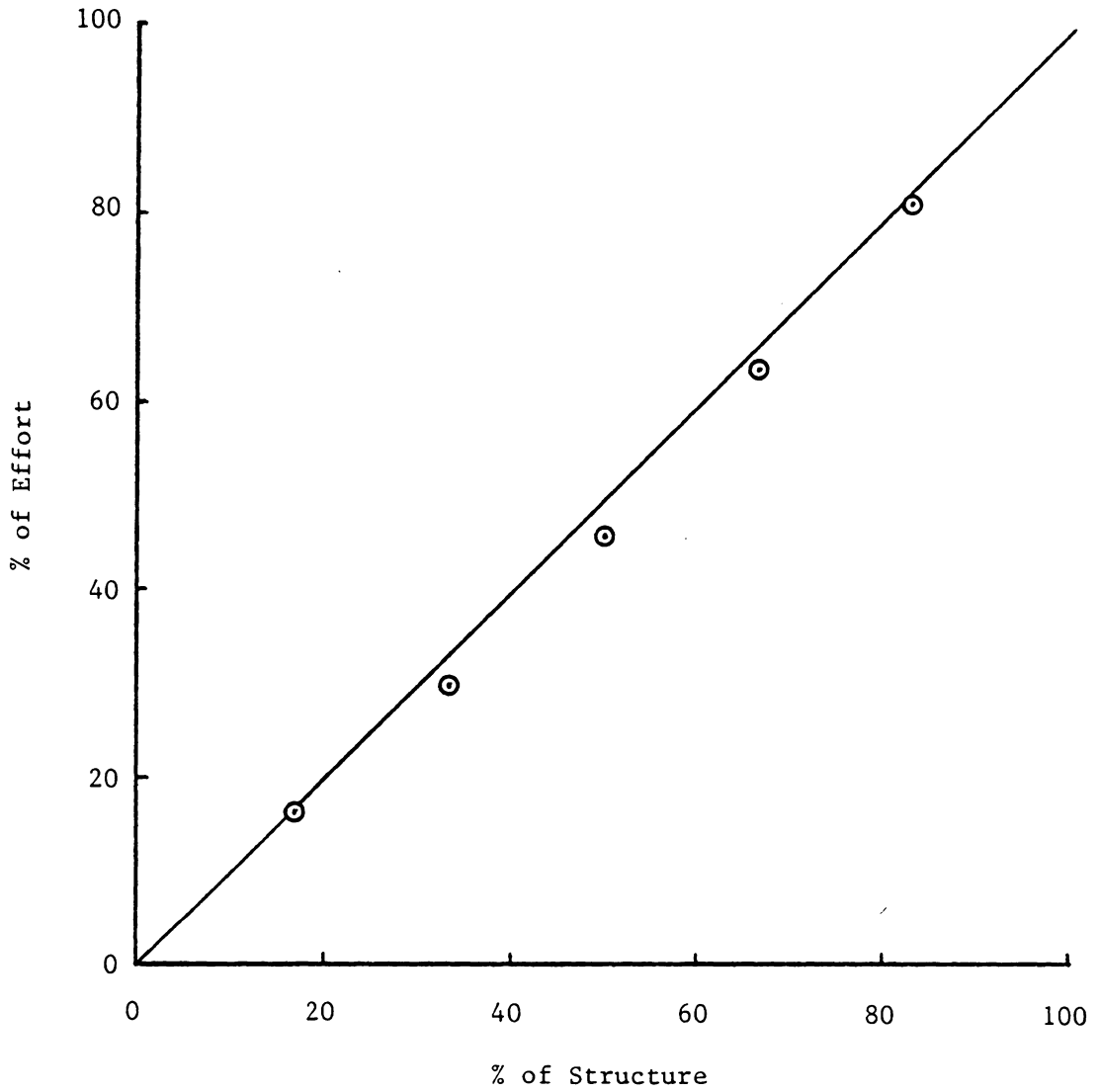


Figure 3.7: Percent of Effort vs. Percent of Structure

nitudes are again expressed in terms of  $\bar{\lambda}$ , which is defined in Eq. 3.2.

The generalized loading condition consisted of downward unit loads at joints 8, 9, 12, 13, 15, 17, 19, 20, 23, and 24.

Tracing the fundamental equilibrium path of the entire structure encountered five bifurcation points before reaching the first limit point at  $\bar{\lambda} = 6.686$ . The first bifurcation point occurred at  $\bar{\lambda} = 2.932$ . Due to problems with the solution algorithm in converging on the remaining bifurcation points and to the high computing costs associated with a structure of this size, the remaining bifurcation were not determined to a high degree of accuracy.

Analyzing a "pie slice" equal to one tenth of the structure yielded no bifurcation points before reaching the first limit point.

Analyzing a "pie slice" equal to two tenths of the structure (see Fig. 3.8) yielded two bifurcation points before reaching the first limit point. The first bifurcation point duplicated the first bifurcation point found by analysis of the entire structure. The second occurred at a value of  $\bar{\lambda} \approx 4.000$  where none had been previously encountered. Again, the author strongly suspects that the associated buckling mode consists of the entire "pie slice" tipping to one side, which would be impossible in an analysis of the entire structure.

The program was unable to converge on the bifurcation path at the first bifurcation point. However, suspecting that, as with the 24-bar dome, the associated buckling mode consisted of a buckling of the compression ring in a "zig-zag" fashion, a 0.001%

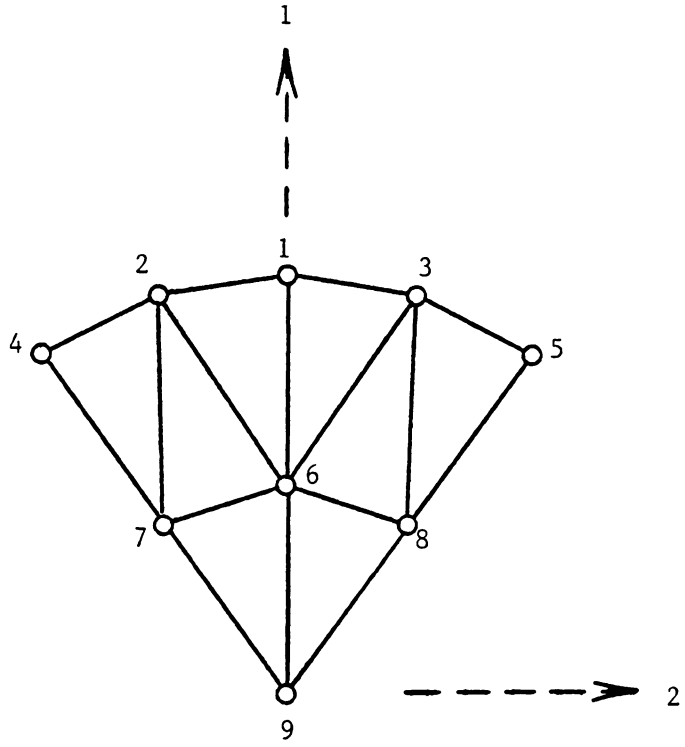


Figure 3.8: "Pie Slice" of Structure 2

imperfection in the vertical coordinate of joint 6 was introduced (actually, since the joint coordinates were entered in spherical coordinates, joint 6 was placed on a spherical cap of radius 1199.988 in. The resulting imperfection in the radial coordinate was negligible). Analysis of this imperfect slice yielded a limit point at  $\bar{\lambda} = 2.884$ , thus confirming the suspicion. However, the author is unable to offer any explanation as to why the one-tenth "pie slice" did not model this buckling mode as did the one-sixth "pie slice" with the 24-bar dome.

Since no single bifurcation point was modelled by all "pie slices" of the structure, a comparison of effort was not made with the 70-bar dome.

## CHAPTER 4

### CONCLUSION AND RECOMMENDATIONS

In this study, the element and system models for linear and geometrically nonlinear three-dimensional trusses were developed and implemented relative to cylindrical coordinates. These models were each used in the analysis of two radially symmetric domes with radially symmetric loading conditions.

The most important benefits derived from using these models are as follows:

1. Analysis of "pie slices" of a structure required considerably less effort than did analysis of the entire structure in both computer time and in operator time to develop data files.
2. Noting which bifurcation points were inhibited by which "pie slice" configurations provided qualitative information concerning buckled configurations when "branching" methods failed, and provided guidance in introducing imperfections to verify suspected buckling modes.

The primary drawback to using these models was that if a buckling mode cannot be modelled by a particular "pie slice" configuration, the bifurcation point corresponding to that mode will be suppressed in the "pie slice" analysis. Therefore, at least a rough analysis of the entire structure is usually required to insure

that all of the bifurcation points have been determined and to permit identification of extraneous bifurcation points introduced by the "pie slice".

The author would like to offer several recommendations concerning the further application of the structural models developed in this study.

1. Investigate further the suppression of bifurcation points by "pie slice" configurations. Primarily, try to offer some explanation for why some buckling modes which exhibit  $C_{2n}$  radial symmetry can be modelled with a "pie slice" equal to one  $n^{\text{th}}$  of the structure (as with structure 1) and others require a "pie slice" equal to one  $2n^{\text{th}}$  of the structure (as with structure 2).
2. Investigate the possibility of expressing arbitrary loads in terms of combinations of radially symmetric loads. In a nonlinear analysis, determine the effect this would have on the locations of bifurcation and limit points. If, indeed, such an expression is possible, investigate the economics of doing so.

The author would also like to offer several suggestions concerning further use of the nonlinear solution technique regardless of which model formulation was employed.

1. A frequent problem encountered was that, when using the modified Riks/Wempner method, while solving the resulting quadratic equation (see refs. 8 and 15) the value under the radical would become negative, resulting in an error. The actual physical meaning of this should be investigated

so that ways to avoid the problem might be suggested. This would make the modified Riks/Wempner method more consistently reliable.

2. Locate or write a solution routine for banded symmetric indefinite matrices with accompanying routines for calculating the determinant and number of negative eigenvalues of the system tangent stiffness matrix. Implement and investigate this routine.
3. The present method for converging on bifurcation points (see ref. 12) consists of noting, via the number of negative eigenvalues, when a bifurcation point has been passed and then reducing the length and changing the sign of the tangent vector until the bifurcation point has been located with sufficient accuracy. However, reducing the length of a vector in  $n + 1$  dimensional space (where  $n$  is the number of D.O.F.) does not insure that the resulting load magnitude does not fall in a region of the equilibrium path that has already been determined not to contain the bifurcation point. Therefore, the method often does not converge rapidly and, occasionally, with very tight convergence tolerances, for all practical purposes, does not converge at all. The author, instead, suggests that the Newton/Raphson method (see ref. 6) might perform better in converging on bifurcation points. Decreasing the size of  $\Delta\lambda$  rather than  $\Delta s$  systematically would provide absolute control of one of the parameters, thus insuring that the

method will converge on the bifurcation point directly and to any desired degree of accuracy. Although less computationally efficient *per iteration*, the author believes that the Newton/Raphson method would provide overall greater efficiency in locating bifurcation points.

4. More often than not, the present branching technique is not capable of converging on a bifurcation path. More research should be conducted in an effort to improve the branching capabilities.

APPENDIX A

REFERENCES

1. Bathe, K.-J. and Wilson, E. L., Numerical Methods in Finite Element Analysis, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.
2. Cook, R. D., Concepts and Applications of Finite Element Analysis, John Wiley and Sons, Inc., New York, N.Y., 1974.
3. Dongarra, J. J., Moler, C. B., Bunch, J. R., and Stewart, G. W., LINPACK Users' Guide, Siam, Philadelphia, Pa., 1979.
4. Glockner, P. G., "Symmetry in Structural Mechanics", Journal of the Structural Division, ASCE, Vol. 99, No. ST1, January, 1973.
5. Holzer, S. M., "Lecture Notes on CE 4001, 4002 -- Matrix Structural Analysis", VPI & SU, Blacksburg, Va., Fall and Winter Quarters, 1979 - 1980.
6. Holzer, S. M., "Lecture Notes on CE 6020 -- Dynamics of Structures", VPI & SU, Blacksburg, Va., Winter Quarter, 1981.
7. Holzer, S. M., Matrix Analysis of Structures -- Computer-Based Analysis and Design, to be published.
8. Holzer, S. M., Watson, L. T., and Vu, P. D., "Stability Analysis of Lamella Domes", Long Span Roof Structures, Proceedings of a Symposium Held at the 1981 Annual Convention and Exhibit, Sponsored by the Committee on Special Structures of the Committee on Metals of the Structural Division, ASCE, St. Louis, Mo., October 26 - 30, 1981.
9. Jagannathan, D. S., Nonlinear Analysis of Reticulated Space Trusses, Ph.D. Dissertation, University of Minnesota, March, 1974.
10. Johnson, L. W. and Riess, R. D., Numerical Analysis, Addison-Wesley Publishing Co., Reading, Mass., 1977.
11. Kolman, B., Elementary Linear Algebra, Macmillan Publishing Co., New York, N.Y., 1970.
12. Lamma, E. E., Tracing the Fundamental and Secondary Equilibrium Paths of Geometrically Nonlinear Space Trusses Using the Modified Riks/Wempner Method, M.S. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Va., August, 1982.
13. Langhaar, H. L., Energy Methods in Applied Mechanics, John Wiley and Sons, Inc., New York, N.Y., 1962.

14. Popov, E. P., Mechanics of Materials, 2nd ed., Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.
15. Vu, P. D., Tracing Nonlinear Equilibrium Paths by the Modified Riks/Wempner Method, M.S. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Va., October, 1981.
16. Wylie, C. R., Advanced Engineering Mathematics, 2nd ed., McGraw-Hill Book Co., New York, N.Y., 1975.

APPENDIX B  
LINEAR PROGRAM

C THIS PROGRAM PERFORMS A LINEAR STRESS AND DISPLACEMENT ANALYSIS ON SPACE  
C TRUSSES. IT CAN OPERATE IN BOTH GLOBAL AND CYLINDRICAL COORDINATES AND  
C ACCEPTS LOADS AND JOINT CONSTRAINTS IN WHICHEVER COORDINATE SYSTEM IS  
C BEING EMPLOYED. THE CYLINDRICAL COORDINATE SYSTEM IS INTENDED FOR, BUT  
C NOT RESTRICTED TO, RADIALY SYMMETRIC STRUCTURES AND ALLOWS FOR IMPOSI-  
C TION OF THE APPROPRIATE BOUNDARY CONDITIONS TO ANALYZE A REPRESENTATIVE  
C "PIE SLICE" OF THE STRUCTURE.

C THE GLOBAL COORDINATE AXES ARE A RIGHT-HANDED SYSTEM. WHEN WORKING  
C WITH A RADIALY SYMMETRIC STRUCTURE IN CYLINDRICAL COORDINATES, THE GLO-  
C BAL 3-AXIS MUST BE PLACED ALONG THE AXIS OF RADIAL SYMMETRY WITH THE  
C 1-DIRECTION BEING RADIALY OUT.

C JOINT COORDINATES MAY BE ENTERED IN TERMS OF CARTESIAN (X1, X2, X3) OR  
C SPHERICAL (R, THETA, PHI) COORDINATES. WHEN ENTERED IN SPHERICAL COOR-  
C DINATES, THE CARTESIAN COORDINATES ARE CALCULATED THUS:

$$\begin{aligned} X1 &= R * \cos(\text{THETA}) * \sin(\text{PHI}) \\ X2 &= R * \sin(\text{THETA}) * \sin(\text{PHI}) \\ X3 &= R * (1. - \cos(\text{PHI})) \end{aligned}$$

C WHEN WORKING IN CYLINDRICAL COORDINATES, FOUR TYPES OF ELEMENTS ARE  
C RECOGNIZED AND MUST BE IDENTIFIED BY THE USER PRIOR TO DATA ENTRY. THESE  
C TYPES ARE:

- C 1) RADIAL ELEMENT—AN ELEMENT WHICH LIES ON A LINE INTERSECTING THE  
C AXIS OF RADIAL SYMMETRY OR WHICH IS PARALLEL TO OR LIES ON  
C THAT AXIS EXCEPT FOR THOSE ELEMENTS WHICH HAVE PRECISELY ONE  
C END POINT ON THE AXIS.
- C 2) RING ELEMENT—AN ELEMENT WHICH HAS BOTH END POINTS EQUIDISTANT FROM  
C AND LYING IN A PLANE PERPENDICULAR TO THE RADIAL AXIS OF SYMMETRY.







C C1H THE DIRECTION COSINE OF THE MEMBER RELATIVE TO THE JOINT  
 C 1-AXIS AT THE B-END (THE END INCIDENT TO THE JOINT WHOSE  
 C NUMBER LIES IN THE "TO" COLUMN IN THE MEMBER PROPERTIES LIST-  
 C ING)  
 C  
 C C2H THE DIRECTION COSINE OF THE MEMBER RELATIVE TO THE JOINT  
 C 2-AXIS AT THE B-END  
 C  
 C CG1 THE DIRECTION COSINE OF THE MEMBER RELATIVE TO THE GLOBAL  
 C 1-AXIS  
 C  
 C CG2 THE DIRECTION COSINE OF THE MEMBER RELATIVE TO THE GLOBAL  
 C 2-AXIS  
 C  
 C CG3 THE DIRECTION COSINE OF THE MEMBER RELATIVE TO THE GLOBAL  
 C 3-AXIS  
 C  
 C E THE MODULUS OF ELASTICITY OF THE MEMBER MATERIAL  
 C  
 C ID THE ID NUMBER OF THE MEMBER  
 C  
 C IHBW THE HALF BAND WIDTH OF THE SYSTEM STIFFNESS MATRIX  
 C  
 C JCODE THE LISTING IN ORDER OF JOINT NUMBER OF THE SYSTEM GENERAL DIS-  
 C PLACEMENT NUMBER CORRESPONDING TO THE 1-, 2-, AND 3-DIRECTION  
 C IN TERMS OF WHICHEVER COORDINATE SYSTEM IS BEING EMPLOYED. A  
 C ZERO ENTRY INDICATES THAT THAT DIRECTION IS CONSTRAINED.  
 C  
 C LENGTH THE LENGTH OF THE MEMBER  
 C  
 C MCODE THE LISTING IN ORDER OF ELEMENT NUMBER OF THE SYSTEM GENERAL  
 C DISPLACEMENT NUMBER CORRESPONDING TO THE 1-, 2-, AND 3-DIR-





```

C*****
C                               MAIN PROGRAM                               *
C*****
  DIMENSION SS(19,19)
  DIMENSION Q(19),LOCATE(19,2)
  DIMENSION A(24),E(24),XL(24),GAMMA(24),CG1(24),CG2(24),
1CG3(24),C1(24),C2(24),C1H(24),C2H(24),MCODE(24,6),
2MINC(24,2),ID(24)
  DIMENSION P(13,3),X1(13),X2(13),X3(13),JCODE(13,3)

C
C   READ AND WRITE CONTROL PARAMETERS AND FLAGS.
C

  READ(1,*) NE,NJ,NACT,JOINTS,ICOORD
  WRITE(2,15) NE,NJ,NACT
15 FORMAT(///' NE=',I3,4X,'NJ=',I3,4X,'NACT=',I3//)
  CALL DATA(X1,X2,X3,A,E,JCODE,MINC,ID,ICOORD,JOINTS,NJ,NE)
  CALL PROCES(X1,X2,X3,CG1,CG2,CG3,C1,C2,C1H,C2H,GAMMA,XL,MCODE,
1JCODE,MINC,ID,NEQS,NJ,NE,IHBW,A,E,ICOORD)
  CALL LOKATE(JCODE,LOCATE,NJ,NEQS)
    DO 30 I=1,NEQS
      DC 30 J=1,IHBW
30      SS(I,J)=0.
  CALL STIFF(SS,CG1,CG2,CG3,C1,C2,C1H,C2H,GAMMA,MCODE,ID,NEQS,
1IHBW,ICOORD,NE)
  NCOUNT=NACT
100 LCNUM=NACT-NCOUNT+1
  WRITE(2,20)LCNUM
20 FORMAT(///' * * * LOADING CONDITION',I3,2X,'* * *')
  CALL ACTION(Q,JCODE,NEQS,NJ)
  CALL SOLVE(SS,Q,LOCATE,NACT,NEQS,IHBW,NCOUNT)
  CALL FORCE(CG1,CG2,CG3,C1,C2,C1H,C2H,Q,GAMMA,MCODE,ID,NJ,NE,
1ICOORD,NEQS,P,MINC)

```

```
NCOUNT = NCOUNT-1  
IF(NCOUNT.GT.0)GOTO 100  
STOP
```

```

C*****
C                               SUBROUTINE ACTION                               *
C*****
      SUBROUTINE ACTION(Q,JCODE,NEQS,NJ)
      DIMENSION Q(NEQS),JCODE(NJ,3)
      DO 1 I=1,NEQS
1      Q(I)=0.
C
C      READ AND WRITE THE EXTERNAL JOINT FORCES.
C
      WRITE(2,4)
4      FORMAT(///' EXTERNAL JOINT FORCES')
      WRITE(2,5)
5      FORMAT(4X,'JNUM',3X,'JDIR',6X,'FORCE')
45     READ(1,*)JNUM,JDIR,FORCE
C
C      TEST FOR LAST CARD AND PLACE THE EXTERNAL JOINT FORCES IN THE FORCE
C      VECTOR.
C
      IF(JNUM.EQ.0)GOTO 50
      K=JCODE(JNUM,JDIR)
      Q(K)=FORCE
      WRITE(2,20)JNUM,JDIR,FORCE
      GOTO 45
20     FORMAT(2I7,F13.2)
50     RETURN

```

```

C*****
C                               SLBROUTINE DATA                               *
C*****
  SUBROUTINE DATA(X1,X2,X3,A,E,JCODE,MINC,ID,ICoord,JOINTS,NJ,NE)
  DIMENSION X1(NJ),X2(NJ),X3(NJ),A(NE),E(NE),JCODE(NJ,3),MINC(NE,2),
  1 ID(NE)
  GOTO(21,22),ICoord
 21 WRITE(2,23)
  GOTO 25
 22 WRITE(2,24)
 23 FORMAT(' THE DISPLACEMENTS AND JOINT FORCES ARE COMPUTED RELATIVE
  1 TO GLOBAL COORDINATES.'//)
 24 FORMAT(' THE DISPLACEMENTS AND JOINT FORCES ARE COMPUTED RELATIVE
  1 TO CYLINDRICAL',/, ' COORDINATES.'//)
C
C   READ AND WRITE JOINT COORDINATES.
C
 25 WRITE(2,20)
 20 FORMAT(' JOINT COORDINATES')
  WRITE(2,100)
100 FORMAT(' JOINT',5X,'X1',8X,'X2',8X,'X3')
  GOTO(1,2),JOINTS
  1   DO 41 I=1,NJ
      READ(1,*)R,THETA,PHI
      THETA =THETA*.01745329
      PHI=PHI*.01745329
      X1(I)=R*COS(THETA)*SIN(PHI)
      X2(I)=R*SIN(THETA)*SIN(PHI)
      X3(I)=R*(1.-COS(PHI))
 41  WRITE(2,101)I,X1(I),X2(I),X3(I)
      GOTO 90
  2   DO 40 I=1,NJ

```

```

      READ(1,*)X1(I),X2(I),X3(I)
40    WRITE(2,101)I,X1(I),X2(I),X3(I)
101  FORMAT(I4,2X,3F10.4)

```

C  
C  
C

READ MEMBER INCIDENCE MATRIX.

```

90    DO 50 I=1,NE
      READ(1,*) MINC(I,1),MINC(I,2)
      IF(MINC(I,2).GT.MINC(I,1))GOTO 50
      TEMP=MINC(I,2)
      MINC(I,2)=MINC(I,1)
      MINC(I,1)=TEMP
50    CONTINUE

```

C  
C  
C

READ AND WRITE MEMBER AND MATERIAL PROPERTIES.

```

      WRITE(2,18)
18    FORMAT(///' MEMBER PROPERTIES')
      GOTO(61,62),ICoord
61    WRITE(2,30)
30    FORMAT(4X,'MEMBER',5X,'AREA',10X,'E',7X,'FROM',2X,'TO')
      DO 63 I=1,NE
      READ(1,*)A(I),E(I)
63    WRITE(2,103)I,A(I),E(I),MINC(I,1),MINC(I,2)
103  FORMAT(I8,F12.4,F14.4,3X,I3,2X,I3)
      GOTO 65
62    WRITE(2,19)
19    FORMAT(4X,'MEMBER',3X,'ID',7X,'AREA',10X,'E',7X,'FROM',2X,'TO')
      DO 60 I=1,NE
      READ(1,*) ID(I),A(I),E(I)
60    WRITE(2,102)I,ID(I),A(I),E(I),MINC(I,1),MINC(I,2)
102  FORMAT(I8,I7,F12.4,F14.4,3X,I3,2X,I3)

```

```

C
C   CONSTRUCT JCODE
C
C   INITIALIZE JCODE TO UNITY
C
65   DO 5 I=1,NJ
      DO 5 J=1,3
          5   JCODE(I,J)=1
C
C   TEST FOR LAST CARD AND PLACE ZEROS FOR CONSTRAINTS.
C
70  READ(1,*) JNUM,JDIR
      IF(JDIR.EQ.4)GOTO 75
      IF(JNUM.EQ.0)GOTO 80
      JCODE(JNUM,JDIR)=0
      GOTO 70
75  JCODE(JNUM,1)=0
      JCODE(JNUM,2)=0
      JCODE(JNUM,3)=0
      GOTO 70
80  RETURN

```

```

C*****
C                               SUBROUTINE FORCE                               *
C*****
      SUBROUTINE FORCE(CG1,CG2,CG3,C1,C2,C1H,C2H,Q,GAMMA,MCODE,ID,NJ,NE,
      1ICoord,NEQS,P,MINC)
      DIMENSION CG1(NE),CG2(NE),CG3(NE),C1(NE),C2(NE),C1H(NE),C2H(NE),Q(
      1NEQS),GAMMA(NE),P(NJ,3),D(6),MCODE(NE,6),ID(NE),MINC(NE,2)
      DO 10 I=1,NJ
      DO 10 J=1,3
10      P(I,J)=0.
      WRITE(2,99)
99      FORMAT(///' ELEMENT FORCES'/)
      WRITE(2,101)
101     FORMAT(4X,'ELEMENT',6X,'FORCE'/)
      IF(ICoord.EQ.1)IDI=5
      DO 20 I=1,NE

C
C      COMPUTE D FROM Q USING MCODE.
C
      DO 30 J=1,6
      K=MCODE(I,J)
      IF(K.EQ.0)GOTO 40
      D(J)=Q(K)
      GOTO 30
40      D(J)=0.
30      CONTINUE
      C3I=CG3(I)
      GAMMAI=GAMMA(I)
      GOTO(200,210),ICoord
200     C1I=CG1(I)
      C2I=CG2(I)
      GOTO 220

```

C  
C  
C  
C

COMPUTE THE ELEMENT FORCES MAKING APPROPRIATE SIMPLIFICATIONS FOR THE  
VARIOUS ID NUMBERS.

```
210  C1I=C1(I)
      C2I=C2(I)
      IDI=ID(I)
220  GOTO(60,70,80,90,100),IDI
60    D14=D(1)-D(4)
      D36=D(3)-D(6)
      F1=GAMMAI*(D14*C1I+D36*C3I)
      GOTO 43
70    D14=D(1)+D(4)
      D25=D(2)-D(5)
      F1=GAMMAI*(D14*C1I+D25*C2I)
      GOTO 43
80    D14=D(1)+D(4)
      D25=D(2)-D(5)
      D36=D(3)-D(6)
      F1=GAMMAI*(D14*C1I+D25*C2I+D36*C3I)
      GOTO 43
90    C1HI=C1H(I)
      C2HI=C2H(I)
      D36=D(3)-D(6)
      F1=GAMMAI*(C1I*D(1)+C2I*D(2)+C3I*D36-C1HI*D(4)-C2HI*D(5))
      GOTO 43
100   D14=D(1)-D(4)
      D25=D(2)-D(5)
      D36=D(3)-D(6)
      F1=GAMMAI*(D14*C1I+D25*C2I+D36*C3I)
43    IF(F1)44,48,49
44    F1=-F1
```

```

WRITE(2,45)I,F1
45  FORMAT(I8,4X,F13.7,3X,'TENSION')
    F1=-F1
    GOTO 52
48  WRITE(2,53)I,F1
53  FORMAT(I8,4X,F13.7)
    GOTO 52
49  WRITE(2,54)I,F1
54  FORMAT(I8,4X,F13.7,3X,'COMPRESSION')

```

C  
C  
C

COMPUTE THE JOINT FORCES FROM THE ELEMENT FORCES AND MINC.

```

52  J=MINC(I,1)
    K=MINC(I,2)
    P(J,1)=P(J,1)+C1I*F1
    GOTO(55,56,57,58,59),IDI
55  P(J,3)=P(J,3)+C3I*F1
    P(K,1)=P(K,1)-C1I*F1
    P(K,3)=P(K,3)-C3I*F1
    GOTO 20
56  P(J,2)=P(J,2)+C2I*F1
    P(K,1)=P(K,1)+C1I*F1
    P(K,2)=P(K,2)-C2I*F1
    GOTO 20
57  P(J,2)=P(J,2)+C2I*F1
    P(J,3)=P(J,3)+C3I*F1
    P(K,1)=P(K,1)+C1I*F1
    P(K,2)=P(K,2)-C2I*F1
    P(K,3)=P(K,3)-C3I*F1
    GOTO 20
58  P(J,2)=P(J,2)+C2I*F1
    P(J,3)=P(J,3)+C3I*F1

```

```

      P(K,1)=P(K,1)-C1HI*F1
      P(K,2)=P(K,2)-C2HI*F1
      P(K,3)=P(K,3)-C3I*F1
      GOTO 20
59     P(J,2)=P(J,2)+C2I*F1
      P(J,3)=P(J,3)+C3I*F1
      P(K,1)=P(K,1)-C1I*F1
      P(K,2)=P(K,2)-C2I*F1
      P(K,3)=P(K,3)-C3I*F1
20     CONTINUE
      WRITE(2,46)
46     FORMAT(///' JOINT FORCES')
      WRITE(2,51)
51     FORMAT(4X,'JOINT',6X,'1-DIR.',11X,'2-DIR.',11X,'3-DIR.'/)
      DO 50 J=1,NJ
50     WRITE(2,47)J,(P(J,K),K=1,3)
47     FORMAT(I7,2X,3F17.7)
      RETURN
      END

```

```
C*****
C                               SUBROUTINE LOKATE                               *
C*****
      SUBROUTINE LOKATE(JCODE,LOCATE,NJ,NEQS)
      DIMENSION JCODE(NJ,3),LOCATE(NEQS,2)
        DO 10 I=1,NJ
          DO 10 J=1,3
            JC=JCODE(I,J)
            IF(JC.EQ.0)GOTO 10
            LOCATE(JC,1)=I
            LOCATE(JC,2)=J
10      CONTINUE
      RETURN
```

```

C*****
C                               SUBROUTINE PROCES                               *
C*****
      SUBROUTINE PROCES(X1,X2,X3,CG1,CG2,CG3,C1,C2,C1H,C2H,GAMMA,XL,MCOD
1E,JCODE,MINC,ID,NEQS,NJ,NE,IHBW,A,E,ICORD)
      DIMENSION X1(NJ),X2(NJ),X3(NJ),CG1(NE),CG2(NE),CG3(NE),C1(NE),C2(N
1E),C1H(NE),C2H(NE),GAMMA(NE),MCODE(NE,6),JCODE(NJ,3),MINC(NE,2),ID
2(NE),XL(NE),A(NE),E(NE)

C
C      GENERATE JCODE AND DETERMINE THE NUMBER OF DEGREES OF FREEDOM.
C
      NEQS=0
      DO 10 I=1,NJ
        DO 10 J=1,3
          NEQS=NEQS+JCODE(I,J)
          JCODE(I,J)=JCODE(I,J)*NEQS
10      CONTINUE
      WRITE(2,1)
      1  FORMAT(///' JCODE' )
        DO 20 I=1,NJ
20      WRITE(2,2)(JCODE(I,J),J=1,3)
      2  FORMAT(3I5)
      WRITE(2,501)NEQS
501  FORMAT(///' NEQS=' ,I3)
      MAXID=0

C
C      COMPUTE MEMBER PROPERTIES AND GENERATE MCODE WITHIN THE SAME DO-LOOP.
C
C      GENERATE MCODE FROM JCODE WITH INFORMATION CONTAINED IN MINC.
C
      DO 40 I=1,NE
        J=MINC(I,1)

```

```

      K=MINC(I,2)
      DO 41 L=1,3
      MCODE(I,L)=JCODE(J,L)
41      MCODE(I,L+3)=JCODE(K,L)

C
C
C
C
C
      COMPUTE MEMBER PROPERTIES--LENGTH, CG1, CG2, CG3, C1, C2, C1H, C2H,
      AND GAMMA, TAKING ID NUMBER INTO ACCOUNT AND MAKING THE APPROPRIATE
      SIMPLIFICATIONS.

      XL1=X1(K)-X1(J)
      XL2=X2(K)-X2(J)
      XL3=X3(K)-X3(J)
      XL(I)=SQRT(XL1**2+XL2**2+XL3**2)
      XLONG=XL(I)
      CG1(I)=XL1/XLONG
      CG2(I)=XL2/XLONG
      GAMMA(I)=A(I)*E(I)/XLONG
      GOTO(200,300),ICoord
200      CG3(I)=XL3/XLONG
      GOTO 110
300      X1J=X1(J)
      X2J=X2(J)
      R=SQRT(X1J**2+X2J**2)
      CG1I=CG1(I)
      CG2I=CG2(I)
      IF(R.NE.0.)GOTO 61
      C1(I)=CG1I
      C2(I)=CG2I
      GOTO 65
61      C1(I)=X1J/R*CG1I+X2J/R*CG2I
      C2(I)=-X2J/R*CG1I+X1J/R*CG2I
65      IDI=ID(I)

```

```

      GOTO(70,80,70,90),IDI
70    XL3=X3(K)-X3(J)
      CG3(I)=XL3/XLONG
      GOTO 110
80    CG3(I)=0.
      GOTO 110
90    XL3=X3(K)-X3(J)
      CG3(I)=XL3/XLONG
      X1K=X1(K)
      X2K=X2(K)
      R=SQRT(X1K**2+X2K**2)
      IF(R.NE.0.)GOTO 100
      C1H(I)=CG1I
      C2H(I)=CG2I
      GOTO 110
100   C1H(I)=X1K/R*CG1I+X2K/R*CG2I
      C2H(I)=-X2K/R*CG1I+X1K/R*CG2I

C
C
C      COMPUTE AND WRITE THE HALF BAND WIDTH.

110   J=0
42    J=J+1
      IS=MCODE(I,J)
      IF(IS.EQ.C)GOTO 42
      J=7
44    J=J-1
      IL=MCODE(I,J)
      IF(IL.EQ.0)GOTO 44
      IE=IL-IS
40    IF(IE.GT.MAXID)MAXID=IE
      IHBW=MAXID+1
      WRITE(2,8) IHBW

```

```

      8 FORMAT(///' IHBW=',I3)
C
C   WRITE OUT MEMBER PROPERTIES.
C
      WRITE(2,3)
      3 FORMAT(///' MCODE')
        DO 50 I=1,NE
50     WRITE(2,4)(MCODE(I,J),J=1,6)
      4 FORMAT(6I5)
        GOTO(51,52),ICCOORD
51     WRITE(2,7)
      7 FORMAT(///' MEMBER',3X,'LENGTH',8X,'CG1',10X,'CG2',10X,'CG3',6X,'G
      GAMMA')
        DO 55 I=1,NE
55     WRITE(2,161)I,XL(I),CG1(I),CG2(I),CG3(I),GAMMA(I)
161    FORMAT(I5,F11.2,3F13.7,F11.4)
      RETURN
52     WRITE(2,6)
      6 FORMAT(///' MEMBER',3X,'LENGTH',8X,'CG1',10X,'CG2',10X,'CG3',11X,'
      1C1',11X,'C2',10X,'C1H',10X,'C2H',6X,'GAMMA')
        DO 60 I=1,NE
          IDI=ID(I)
          GOTO(120,130,140,150),IDI
120     WRITE(2,121)I,XL(I),CG1(I),CG2(I),CG3(I),C1(I),C2(I),GAMMA(I)
          GOTO 60
130     WRITE(2,131)I,XL(I),CG1(I),CG2(I),CG3(I),C1(I),C2(I),GAMMA(I)
          GOTO 60
140     WRITE(2,141)I,XL(I),CG1(I),CG2(I),CG3(I),C1(I),C2(I),GAMMA(I)
          GOTO 60
150     WRITE(2,151)I,XL(I),CG1(I),CG2(I),CG3(I),C1(I),C2(I),C1H(I),C2H
      1 (I),GAMMA(I)
60     CONTINUE

```

```
121 FORMAT(I5,F11.2,5F13.7,6X,'RADIAL ELEMENT',6X,F11.4)
131 FORMAT(I5,F11.2,5F13.7,7X,'RING ELEMENT',7X,F11.4)
141 FORMAT(I5,F11.2,5F13.7,5X,'SKEW RING ELEMENT',4X,F11.4)
151 FORMAT(I5,F11.2,7F13.7,F11.4)
      RETURN
```

```

C*****
C                               SUBROUTINE SOLVE                               *
C*****
C      SUBROUTINE SOLVE(SS,Q,LOCATE,NACT,NEQS,IHBW,NCOUNT)
C      DIMENSION SS(NEQS,IHBW),Q(NEQS),LOCATE(NEQS,2)
C      IF(NACT.GT.NCOUNT)GOTO 800
C
C      REDUCE THE STIFFNESS MATRIX.
C
C      700      DO 790 N=1,NEQS
C              DO 780 L=2,IHBW
C                IF(SS(N,L).EQ.0.)GOTO 780
C                I=N+L-1
C                C=SS(N,L)/SS(N,1)
C                J=0
C                DO 750 K=L,IHBW
C                  J=J+1
C                  SS(I,J)=SS(I,J)-C*SS(N,K)
C                  SS(N,L)=C
C      780      CONTINUE
C
C      REDUCE FORCE VECTOR.
C
C      790      CONTINUE
C      800      DO 830 N=1,NEQS
C              DO 820 L=2,IHBW
C                IF(SS(N,L).EQ.0.)GOTO 820
C                I=N+L-1
C                Q(I)=Q(I)-SS(N,L)*Q(N)
C      820      CONTINUE
C
C      BACK-SOLVE FOR THE GENERALIZED DISPLACEMENTS.

```

C

```
830      Q(N)=Q(N)/SS(N,1)
        DO 860 M=2,NEQS
        N=NEQS+1-M
        DO 850 L=2,IHBW
        IF(SS(N,L).EQ.0.)GOTO 850
        K=N+L-1
        Q(N)=Q(N)-SS(N,L)*Q(K)
850      CONTINUE
860      CONTINUE
        WRITE(2,61)
61      FORMAT(///' GENERALIZED DISPLACEMENTS')
        DO 900 I=1,NEQS
900      WRITE(2,60) I,Q(I),LOCATE(I,1),LOCATE(I,2)
60      FORMAT(4X,'Q(',I3,') =',F11.7,3X,'(JOINT',I3,',',I1,')-DIRECTION)'
1)
        RETURN
```

```

C*****
C                               SUBROUTINE STIFF                               *
C*****
      SUBROUTINE STIFF(SS,CG1,CG2,CG3,C1,C2,C1H,C2H,GAMMA,MCODE,ID,NEQS,
      IHBW,ICoord,NE)
      DIMENSION SS(NEQS,IHBW),CG1(NE),CG2(NE),CG3(NE),C1(NE),C2(NE),C1H(
      INE),C2H(NE),GAMMA(NE),G(15),INDEX(6,6,5),MCODE(NE,6),ID(NE)
      INTEGER INDEX/1,0,3,-1,0,-3,6*0,3,0,10,-3,0,-10,-1,0,-3,1,0,3,6*0,
      1-3,0,-10,3,0,10,1,2,0,1,-2,0,2,6,0,2,-6,7*0,1,2,0,1,-2,0,-2,-6,0,-
      22,6,7*0,1,2,3,1,-2,-3,2,6,7,2,-6,-7,3,7,10,3,-7,-10,1,2,3,1,-2,-3,
      3-2,-6,-7,-2,6,7,-3,-7,-10,-3,7,10,1,2,3,-4,-5,-3,2,6,7,-8,-9,-7,3,
      47,10,-11,-12,-10,-4,-8,-11,13,14,11,-5,-9,-12,14,15,12,-3,-7,-10,1
      51,12,10,1,2,3,-1,-2,-3,2,6,7,-2,-6,-7,3,7,10,-3,-7,-10,-1,-2,-3,1,
      62,3,-2,-6,-7,2,6,7,-3,-7,-10,3,7,10/
      DO 1 I=1,NEQS
        DO 1 J=1,IHBW
1          SS(I,J)=0.
      IF(ICoord.EQ.1)IDI=5
      DO 10 I=1,NE
        GAMMAI=GAMMA(I)
        C3I=CG3(I)
        GOTO(200,300),ICoord
200       C1I=CG1(I)
          C2I=CG2(I)
          GOTO 100
300       C1I=C1(I)
          C2I=C2(I)
          IDI=ID(I)
C
C      COMPUTE THE APPROPRIATE G FACTORS FOR THE ELEMENT BASED ON ITS ID NUMBER.
C
100      G(1)=GAMMAI*C1I**2

```

```

      GOTO(110,120,130,140,130),IDI
110   G(3)=GAMMAI*C1I*C3I
      G(10)=GAMMAI*C3I**2
      GOTO 150
120   G(2)=GAMMAI*C1I*C2I
      G(6)=GAMMAI*C2I**2
      GOTO 150
130   G(2)=GAMMAI*C1I*C2I
      G(3)=GAMMAI*C1I*C3I
      G(6)=GAMMAI*C2I**2
      G(7)=GAMMAI*C2I*C3I
      G(10)=GAMMAI*C3I**2
      GOTO 150
140   C1HI=C1H(I)
      C2HI=C2H(I)
      G(2)=GAMMAI*C1I*C2I
      G(3)=GAMMAI*C1I*C3I
      G(4)=GAMMAI*C1I*C1HI
      G(5)=GAMMAI*C1I*C2HI
      G(6)=GAMMAI*C2I**2
      G(7)=GAMMAI*C2I*C3I
      G(8)=GAMMAI*C2I*C1HI
      G(9)=GAMMAI*C2I*C2HI
      G(10)=GAMMAI*C3I**2
      G(11)=GAMMAI*C3I*C1HI
      G(12)=GAMMAI*C3I*C2HI
      G(13)=GAMMAI*C1HI**2
      G(14)=GAMMAI*C1HI*C2HI
      G(15)=GAMMAI*C2HI**2

```

C  
C  
C

PLACE G FACTORS IN BANDED SYSTEM STIFFNESS MATRIX USING INFORMATION CONTAINED IN MCODE AND INDEX.

C

```
150      DO 20 JM=1,6  
        J=MCODE(I,JM)  
        IF(J.EQ.0)GOTO 20  
          DO 30 KM=JM,6  
            K=MCODE(I,KM)  
            IF(K.EQ.0)GOTO 30  
            KB=K-J+1  
            L=INDEX(JM,KM,IDI)  
            IF(L)60,30,40  
60          L=-L  
            SS(J,KB)=SS(J,KB)-G(L)  
            GOTO 30  
40          SS(J,KB)=SS(J,KB)+G(L)  
30          CONTINUE  
20          CONTINUE  
10          CONTINUE  
          RETURN
```

APPENDIX C  
NONLINEAR PROGRAM

```

C*****
C*                                     GENERAL INFORMATION                               *
C*****
C
C
C   THIS PROGRAM USES THE MODIFIED RIKS/WEMPNER METHOD TO TRACE THE
C   EQUILIBRIUM PATHS OF GEOMETRICALLY NONLINEAR SPACE TRUSSES. IT
C   IS ASSUMED THAT THE USER IS FAMILIAR WITH THE METHOD.
C
C-----
C
C   THIS PROGRAM MAKES USE OF FIVE DIFFERENT FILES.
C
C FILE 1  CONTAINS ALL INPUT DATA
C
C FILE 2  CONTAINS OUTPUT OF THE DATA AND THE NORMAL VECTOR USED IN
C          BRANCHING ONTO A SECONDARY EQUILIBRIUM PATH.
C
C FILE 3  CONTAINS OUTPUT OF THE LOAD LEVEL AND DISPLACEMENT VECTOR
C          OF THE EQUILIBRIUM POINTS.
C
C FILE 4  CONTAINS OUTPUT OF THE LOAD LEVEL AND DISPLACEMENT VECTOR
C          OF THE EQUILIBRIUM POINTS OBTAINED IN THE DETERMINATION
C          OF THE CRITICAL POINT.
C
C FILE 7  CONTAINS OUTPUT OF THE LOAD LEVEL AND DISPLACEMENT VECOR
C          OF ALL EQUILIBRIUM POINTS IN A SUITABLE FORMAT FOR INPUT
C          INTO A PLUTTING PROGRAM.
C-----
C

```

C THE EXECUTION OF THIS PROGRAM CAN BE BROKEN INTO SEVEN DIFFERENT  
C CATEGORIES.  
C  
C TYPE 1 N NUMBER OF POINTS ARE OBTAINED ON THE EQUILIBRIUM PATH.  
C  
C TYPE 2 ONE REQUIRES THE PROGRAM TO EXECUTE UNTIL A STABLE POINT  
C IS ENCOUNTERED.  
C  
C TYPE 3 N NUMBER OF POINTS ARE OBTAINED AFTER THE FIRST CRITICAL  
C POINT. IT IS RECOMMENDED THAT N BE LESS THAN 5.  
C  
C TYPE 4 BRANCHING ONTO SECONDARY EQUILIBRIUM PATHS. THE SEARCH FOR  
C THE CRITICAL POINT WILL BEGIN AT POINT N AND M POINTS ARE  
C OBTAINED ON THE BRANCHING PATH.  
C  
C TYPE 5 DETERMINATION OF A CRITICAL POINT. THE SEARCH BEGINS AT  
C POINT N.  
C  
C TYPE 6 RUNNING THE PROGRAM TO OBTAIN THE FIRST CRITICAL POINT.  
C THE NUMBER OF POINTS, N, SHOULD BE LARGE ENOUGH FOR THE  
C PROGRAM TO REACH A CRITICAL POINT.  
C  
C TYPE 7 STARTING THE PROGRAM AT A KNOWN EQUILIBRIUM POINT ON THE  
C EQUILIBRIUM PATH. THIS CAN BE DONE IN COMBINATION WITH  
C TYPES 1-6.  
C

C THE FOLLOWING VARIABLES IN THE INPUT DATA ARE TABULATED TO AID  
 C THE USER IN DETERMINING INPUT VALUES.

C	N	E	E	B	M	B	E	E	E	L	N	D	E	D		
C	P	N	N	R	P	P	P	P	P	A	N	D	E	D		
C	T	O	D	D	A	O	S	S	S	M	N	D	E	X	S	
C	Y	I	L	S	N	I	I	L	L	L	M	N	E	X	S	
C	P	N	I	T	C	N	N	N	N	D	E	T	P	B		
C	C	E	T	M	B	H	T	1	2	3	A	G	U	O	R	Q0
C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C 1	N	1,2	2	2	-#	@	2	S	S	@	+	+	C	C	@	+
C 2	@	2	2	1	-#	@	2	S	S	@	+	+	C	C	@	+
C 3	N	1,2	1	2	-#	@	2	S	S	@	+	+	C	C	@	+
C 4	@	1,2	2	2	N	M	2	S	S	S	+	+	C	C	S	+
C 5	@	2	2	2	N	@	1	S	S	S	+	+	C	C	@	+
C 6	N	2	2	2	-#	@	3	S	S	S	+	+	C	C	@	+
C 7					(CAN BE USED IN 1-6)					S	S	S	S			S

C WHERE N AND M ARE DEFINED ABOVE  
 C S = SPECIFY  
 C # = ARBITRARY NUMBER  
 C @ = NOT APPROPRIATE, BUT SPECIFY  
 C Q0 = INITIAL DISPLACEMENT VECTOR (NOT ENTERED IF ZERO)  
 C + = ZERO IF NOT USING TYPE 7  
 C C = CAN BE SPECIFIED  
 C 1,2 = 1 OR 2

```

C
C*****
C*                               INPUT GUIDE                               *
C*****
C
C   FOR VARIABLE INPUT, (I) REPRESENTS AN INTEGER VARIABLE
C                               (R) REPRESENTS A REAL VARIABLE.
C
C   ***** ALL INPUT IS FORMAT FREE *****
C
C   _____
C
C   DATA SET
C
C   1      INPUT OF THE FOLLOWING VARIABLES
C
C           NM (I)
C           NJ (I)
C           SYSTEM (I)
C           UPDATE (I)
C           IHBW (I)
C           SCALE (I)
C           LHAT (I)
C           LMAX (I)
C           NPOINT (I)
C           METHOD (I)
C           CONT (I)
C           ENDLIM (I)
C           ENDSTB (I)
C           INTERST (I)
C           BRANCH (I)
C           IMAX (I)
C           JMAX (I)

```



C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C  
C

JOINT ARE CONSTRAINED. FOLLOW THIS SET WITH A  
ZERO JOINT NUMBER AND JOINT DIRECTION.

JNUM , JDIR  
(I) (I)

5 MEMBER DATA IN INCREASING MEMBER NUMBER. A NEGATIVE  
AREA FOR MEMBER 1 INDICATES THAT A AND E ARE 1.0 FOR  
ALL MEMBERS AND THE REMAINING MEMBERS NEED ONLY SUPPLY  
MINC.

A , E , MINC  
(R) (R) (I)(I)

6 LOAD DATA. FOLLOW WITH A ZERO FOR JNUM, JDIR, AND P1

JNUM , JDIR , P1  
(I) (I) (R)

7 DISPLACEMENT VECTOR. FOLLOW SET 6 WITH A NEGATIVE  
JOINT NUMBER IF THERE IS NO DISPLACEMENT VECTOR.

```

C*****
C*                                     ARRAY DIMENSIONING                               *
C*****
C
C   COMMON /BLOCK1/ SS(NEQS,IHBW),DQ(NEQS),DQ1(NEQS),DQI(NEQS),
C                   DISTR(NEQS),LOAD(NEQS),Q(NEQS),R(NEQS),Q1(NEQS),
C                   R1(NEQS),KPVT(NEQS)
C   COMMON /BLOCK2/ A(NM),CG1(NM),CG2(NM),CG3(NM),E(NM),XL(NM),
C                   XLD(NM),MCODE(NM,6),MINC(NM,2)
C   COMMON /BLOCK3/ X(NJ),Y(NJ),Z(NJ),C1(NJ),C2(NJ),JCODE(NJ,3)
C   COMMON /BLOCK4/ DIMENSIONS ARE THE SAME FOR ALL PROBLEMS.
C
C   FOR SUBROUTINE EIGEN:  XTEMP(NEQS)
C
C   AT THE BEGINNING OF MAINLINE, LDA = DIMENSIONED VALUE OF NEQS.
C
C   NM = NUMBER OF MEMBERS
C   NJ = NUMBER OF JOINTS
C   NEQS = NUMBER OF EQUATIONS (DEGREES OF FREEDOM)
C   IHBW = HALF-BAND WIDTH
C
C   NOTES:  C1 AND C2 ARE ONLY USED WHEN THE PROGRAM IS OPERATING IN
C           CYLINDRICAL COORDINATES.  TO AVOID WASTING STORAGE SPACE
C           DIMENSION (1) WHEN OPERATING IN GLOBAL COORDINATES.
C
C           KPVT IS USED ONLY WHEN USING THE LINPACK EQUATION SOLVER
C           TO AVOID WASTING STORAGE SPACE, DIMENSION (1) WHEN USING
C           THE BANDED EQUATION SOLVER.
C
C           WHEN USING THE LINPACK EQUATION SOLVER, DIMENSION
C           SS(NEQS,NEQS).
C

```

```

C*****
C*                                     PROGRAM VARIABLES                                     *
C*****
C
C
C      ***** FOR ALL VARIABLES, 1=YES, 2=NO *****
C
C
C          A(I)   AREA OF MEMBER I.
C      A1,B1,D1  COEFFICIENTS OF THE QUADRATIC EQUATION IN
C                SUBROUTINE SPHERE.
C
C          BPOINT  ALLOWS CONVERGENCE ON THE BIFURCATION
C                POINT WITHOUT BRANCHING ONTO ANOTHER PATH.
C
C          BRANCH  DETERMINES IF THE PROGRAM IS TO BRANCH OFF
C                OF THE FUNDAMENTAL PATH ONTO THE BRANCHING
C                PATH.
C                -# = DO NOT BRANCH
C                0 = BRANCH IMMEDIATELY
C                # = BEGIN BRANCHING ROUTINE AT POINT #
C      CG1(I),CG2(I),CG3(I)  DEFORMED GLOBAL DIRECTION COSINES FOR MEM-
C                BER I.
C
C          C1(I),C2(I)  THE GLOBAL DIRECTION COSINES FOR THE 1-AXIS
C                FOR JOINT I WHEN WORKING IN CYLINDRICAL CO-
C                ORDINATES.
C
C          CONT    DETERMINES IF THE PROGRAM SHOULD CONTINUE
C                TO EXECUTE WHEN THE PROGRAM HAS BECOME
C                MORE STABLE (ALL NEGATIVE EIGENVALUES HAVE
C                DISAPPEARED).
C
C          D(I)    DISPLACEMENT FOR AN ELEMENT IN DIRECTION I,
C                OR MEMBER END FORCES USED TO COMPUTE R(I).
C
C          DET     THE DETERMINANT OF THE TANGENT STIFFNESS
C                MATRIX LESS THE SCALE FACTOR.
C

```

C           DETO    DET FOR THE UNDEFORMED TRUSS.  IF DETO IS  
C                    INPUT AS 0, IT WILL BE CALCULATED.  
C                    (INPUT THE DETERMINANT WHEN STARTING AT A  
C                    KNOWN POINT ON THE EQUILIBRIUM PATH)  
C           DETNW   THE NEW DETERMINANT WHEN CONVERGING ON THE  
C                    BIFURCATION POINT.  
C           DETOLD   THE OLD DETERMINANT WHEN CONVERGING ON THE  
C                    BIFURCATION POINT.  
C   DISTR(I)   LOAD DISTRIBUTION VECTOR.  
C           DLAMD   CHANGE IN LOAD LEVEL.  
C           DLAMD1   CHANGE IN LOAD LEVEL AT THE FIRST STEP.  
C           DQ(I)    CHANGE IN DISPLACEMENT.  
C           DQ1(I)   SOLUTION TO  $K DQKI = DISTR$  FOR ITERATION 1.  
C           DQI(I)   SOLUTION TO  $K DQKI = DISTR$ .  
C           DS       THE LENGTH OF THE TANGENT VECTOR.  
C           DSBR     THE INITIAL CHANGE IN LOAD LEVEL WHEN  
C                    BRANCHING.  
C           E(I)     YOUNG'S MODULUS FOR MEMBER I.  
C           ELONG    ELONGATION OF A MEMBER.  
C   ENDLIM       STOPS EXECUTION AT NPOINT POINTS BEYOND THE  
C                    FIRST CRITICAL POINT.  
C   ENDSTB       STOPS EXECUTION WHEN THE TRUSS HAS BECOME  
C                    STABLE.  
C   EPSLN1       CONVERGENCE CRITERIA FOR THE UNBALANCED FORCE  
C                    TO OBTAIN THE POINT ON THE EQUILIBRIUM PATH.  
C   EPSLN2       CONVERGENCE CRITERIA FOR THE CHANGE IN  
C                    DISPLACEMENT TO OBTAIN THE POINT ON THE  
C                    EQUILIBRIUM PATH.  
C   EPSLN3       FOR CONVERGENCE TO THE BIFURCATION POINT,  
C                    THIS IS THE TOLERANCE OF THE RATIO OF THE  
C                    DETERMINANT OF THE STIFFNESS MATRIX AT SOME  
C                    POINT TO THE DETERMINANT OF THE STIFFNESS





C	MCODE(I,J)	DOF IN DIRECTION J FOR MEMBER I.
C	METHOD	PROCEDURE USED IN THE PROGRAM.
C		1=NORMAL PLANE ITERATION
C		2=SPHERE ITERATION
C		3=NEWTON-RAPHSON
C	MINC(I,J)	MEMBER INCIDENCE FOR MEMBER I AT END J.
C	MPOINT	NUMBER OF POINTS WANTED AFTER BRANCHING.
C	NCOUNT	THE NUMBER OF THE POINT ON THE EQUILIBRIUM
C		PATH.
C	NDIGIT	THE NUMBER OF SIGNIFICANT DIGITS WANTED FOR
C		THE EIGENVALUE IN SUBROUTINE POWINV.
C	NEQS	NUMBER OF EQUATIONS.
C	NJ	NUMBER OF JOINTS.
C	NM	NUMBER OF MEMBERS.
C	NNEG	NUMBER OF NEGATIVE EIGENVALUES FOR SS.
C	NPOINT	NUMBER OF POINTS WANTED ON THE EQUILIBRIUM
C		PATH.
C	PHI	ANGLE OF ROTATION ABOUT THE Z-AXIS IN POLAR
C		COORDINATES.
C	Q(I)	DISPLACEMENT FOR DOF I.
C	Q1(I)	DISPLACEMENT FOR DOF I BEFORE THE FIRST STEP.
C	R(I)	UNBALANCED FORCE VECTOR.
C	RI(I)	THE EIGENVECTOR USED TO OBTAIN THE NORMAL
C		VECTOR.
C	RADIUS	THE VECTOR LENGTH IN POLAR COORDINATE DATA.
C	SCALE	DETERMINES IF THE TANGENT VECTOR IS TO BE
C		SCALED.
C	SS(I,J)	THE SYSTEM TANGENT STIFFNESS MATRIX. (STORED
C		IN BANDED FORM)
C	SYSTEM	COORDINATE SYSTEM USED FOR INPUT OF JOINT
C		LOCATIONS. 1=SPHERICAL
C		2=RECTANGULAR

```

C          THETA   ANGLE FROM THE Z-AXIS IN POLAR COORDINATES.
C          UPDATE  DETERMINES IF THE STIFFNESS MATRIX IS TO BE
C                  UPDATED EVERY ITERATION.
C          X(I),Y(I),Z(I)  GLOBAL JOINT COORDINATES.
C                  RECTANGULAR:  X=X, Y=Y,      Z=Z
C                  SPHERICAL:    X=R, Y=THETA, Z=PHI
C          X1(I),Y1(I),Z1(I)  VARIABLES USED FOR CALCULATIONS IN
C                  SUBPROGRAMS.
C          XL(I)    UNDEFORMED LENGTH OF MEMBER I.
C          XLAMDA(I)  TEMPORARY STORAGE FOR Q(I) FOR THE CASE
C                  WHERE THE ASSUMPTION IN SUBROUTINE EIGEN IS
C                  WRONG.
C          XLD(I)   DEFORMED LENGTH OF MEMBER I.
C          XTEMP    TEMPORARY STORAGE FOR LAMDA FOR THE CASE
C                  WHERE THE ASSUMPTION IN SUBROUTINE EIGEN IS
C                  WRONG.

```

```

C*****
C*          SUBROUTINE DESCRIPTIONS          *
C*****

```

```

C          BPATH  THIS SUBROUTINE FINDS THE CRITICAL POINT GIVEN THE TOLERANCE
C                  EPSLN3.  IT IS ASSUMED THAT THE POINT FROM WHICH THE SEARCH
C                  IS MADE IS RELATIVELY 'CLOSE' TO THE CRITICAL POINT.  THIS
C                  ALLOWS ONE TO ASSUME A LINEAR RELATIONSHIP BETWEEN DS AND THE
C                  DETERMINANT OF THE TANGENT STIFFNESS MATRIX. (AT A CRITICAL
C                  POINT, THE DETERMINANT IS ZERO.)  EPSLN3 SPECIFIES THE LOSS
C                  IN STIFFNESS OF THE TRUSS.
C
C          CONVG  THIS SUBROUTINE DETERMINES IF THE RIKS/WEMPNER ITERATION
C                  HAS CONVERGED ON THE EQUILIBRIUM PATH.

```

C  
 C DATA THIS SUBROUTINE INPUTS THE DATA FOR THE PROGRAM FROM FILE  
 C NUMBER 1. SOME INITIALIZATION OF VARIABLES OCCURS HERE ALSO.  
 C  
 C DEFORM THIS SUBROUTINE DETERMINES THE MEMBERS DEFORMED LENGTHS AND  
 C DEFORMED COSINES FOR THE PRESENT CONFIGURATION.  
 C  
 C DTNEG1 THIS SUBROUTINE CALCULATES THE NUMBER OF NEGATIVE EIGEN-  
 C VALUES AND THE DETERMINANT OF THE TANGENT STIFFNESS MATRIX  
 C USING THE BANDED SOLVER. THIS IS PERFORMED AFTER THE FORWARD  
 C REDUCTION OF THE STIFFNESS MATRIX.  
 C  
 C DTNEG2 THIS SUBROUTINE PERFORMS THE SAME FUNCTION AS DTNEG1 USING THE  
 C LINPACK EQUATION SOLVER.  
 C  
 C EIGEN THIS SUBROUTINE AUTOMATES THE PROGRAM SO THAT IT WILL  
 C DETERMINE THE DIRECTION OF TRAVEL WHEN CRITICAL POINTS ARE  
 C ENCOUNTERED. SEVERAL CONTROL VARIABLES ARE USED HERE AND  
 C THEY CAUSE THE SUBROUTINE TO APPEAR COMPLEX.  
 C  
 C F THIS FUNCTION DETERMINES THE DOT PRODUCT OF SEVERAL VECTORS.  
 C FOR THE ARGUMENTS IN THE ARGUMENT LIST  
 C 
$$F = X1 \cdot ( Y1 + K * Z1 )$$
  
 C  
 C INIT THIS SUBROUTINE DETERMINES THE INITIAL TANGENT VECTOR FOR  
 C THE ITERATION TO BEGIN. IT ALSO CALCULATES DS OR DLAMD  
 C FOR THE INITIAL CONFIGURATION. (EITHER DS OR DLAMD IS  
 C SPECIFIED IN INPUT)  
 C  
 C NEWTON THIS SUBROUTINE USES THE NEWTON-RAPHSON METHOD TO DETERMINE  
 C THE EQUILIBRIUM PATH.  
 C

C NORMAL THIS SUBROUTINE DETERMINES DLAMD AND DQ FOR THE 'NORMAL  
 C PLANE' ITERATION.  
 C  
 C POWINV THIS SUBROUTINE USES THE INVERSE POWER METHOD TO DETERMINE  
 C AN EIGENVALUE AND EIGENVECTOR OF THE TANGENT STIFFNESS  
 C MATRIX. THIS METHOD CONVERGES ON THE EIGENVALUE NEAREST TO  
 C ALPHA.  
 C  
 C RESID THIS SUBROUTINE DETERMINES THE RESIDUAL FORCE VECTOR FROM THE  
 C APPLIED LOAD AND THE JOINT FORCES OBTAINED FROM THE DISPLACE-  
 C MENT VECTOR.  
 C  
 C RWC THIS SUBROUTINE USES THE MODIFIED RIKS/WEMPNER METHOD TO  
 C OBTAIN THE EQUILIBRIUM PATH.  
 C  
 C SOLVE1 THIS SUBROUTINE SOLVES A SET OF SIMULTANEOUS EQUATIONS USING  
 C THE BANDED STIFFNESS MATRIX. THE FIRST CALL TO SOLVE (AFTER  
 C THE STIFFNESS MATRIX HAS BEEN CALCULATED) DOES FORWARD  
 C REDUCTION OF THE MATRIX. ALL SUBSEQUENT CALLS TO SOLVE DO  
 C THE FORWARD REDUCTION OF CONSTANTS AND BACK-SUBSTITUTION  
 C (UNTIL STIFF IS CALLED AGAIN).  
 C  
 C SOLVE2 THIS SUBROUTINE PERFORMS THE SAME FUNCTIONS AS SOLVE1 EXCEPT  
 C USING THE LINPACK EQUATION SOLVER. SUBROUTINES SSIFA, SSISL,  
 C SSWAP, AND SAXPY ARE CALLED FROM HERE, AS ARE FUNCTIONS SDOT  
 C AND ISAMAX.  
 C  
 C SPHERE THIS SUBROUTINE DETERMINES DLAMD AND DQ FOR THE 'SPHERE'  
 C ITERATION.  
 C  
 C STIFF THIS SUBROUTINE CALCULATES THE BANDED TANGENT STIFFNESS  
 C MATRIX.

C  
C STORE THIS SUBROUTINE STORES A COMBINATION OF SEVERAL VECTORS INTO  
C ANOTHER VECTOR. FOR THE ARGUMENTS IN THE ARGUMENT LIST  
C  $X1 = Y1 + V * Z1$   
C  
C VECTOR THIS SUBROUTINE DETERMINES THE NORMAL VECTOR FROM THE  
C EIGENVECTOR IN POWINV. THIS ALLOWS ONE TO BRANCH ONTO  
C SECONDARY EQUILIBRIUM PATHS.  
C

```

C*****
C*                                     MAINLINE                                     *
C*****
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE, SYSTEM, UPDATE, CONT, ENDLIM, ENDSTB, BRANCH,
1     EXP, EXPO, EXPOLD, EXPNEW, BPOINT
  REAL*8 LOAD, LAMDA, LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70), DQ(70), DQ1(70), DQI(70), DISTR(70),
1     LOAD(70), Q(70), R(70), Q1(70), R1(70), KPVT(70)
  COMMON /BLOCK2/ A(70), CG1(70), CG2(70), CG3(70), E(70), XL(70), XLD(70)
1     , MCODE(70,6), MINC(70,2)
  COMMON /BLOCK3/ X(31), Y(31), Z(31), C1(31), C2(31), JCODE(31,3)
  COMMON /BLOCK4/ D(6), DLAMD, DLAMD1, DS, ELONG, EPSLN1, EPSLN2, EPSLN3,
1     DETO, DET, DETOLD, DETNEW, LAMDA, LAMDA1, DSBR,
2     IHBW, JFLAG, JHBW, L, IFLAG, LHAT, LMAX, ITRACE(2000),
3     NCOUNT, NEQS, NJ, NM, NNEG, NPOINT, SCALE, SYSTEM, K15,
4     UPDATE, CONT, ENDLIM, ENDSTB, INTEST, METHOD, BRANCH,
5     EXP, EXPO, EXPOLD, EXPNEW, IMAX, JMAX, MPOINT, NDIGIT,
6     IHELP, BPOINT, LFLAG, ICOORD, NUMBER, LDA, ISOLVE

  LDA=70
  CALL DATA
C  WRITE OUT THE INITIAL (STARTING) POINT DATA.
  L=LHAT
  K15=0
  LFLAG=2
  I=0
  NUMBER=1
  WRITE(3,1) I, LAMDA
1  FORMAT(13, ' LAMDA =', G24.16, ' ', 83(' '*))
  WRITE(3,2)
2  FORMAT(' DISPLACEMENT')

```

```

WRITE(3,3) (Q(I),I=1,NEQS)
3 FORMAT(5G24.16)
WRITE(7,42) LAMDA
42 FORMAT(F15.9)
WRITE(7,41) (Q(I),I=1,NEQS)
NCOUNT=0
C CHECK TO SEE IF THE PROGRAM IS TO START BRANCHING FROM THE
C POINT ON THE PATH THAT WAS INPUT.
IF(BRANCH.NE.0) GO TO 13
C $$$ A $$$
K15=K15+1
ITRACE(K15)='A'
CALL DEFORM
CALL STIFF
CALL VECTOR
IMETH=METHOD
METHOD=2
CALL RWC
METHOD=IMETH
GO TO 80
C CHECK TO SEE IF THE DETERMINANT OF THE UNDEFORMED TRUSS WAS
C INPUT.
13 IF(DETO.NE.0.DO) GO TO 10
C $$$ B $$$
K15=K15+1
ITRACE(K15)='B'
C DETERMINE THE INITIAL DETERMINANT.
CALL DEFORM
CALL STIFF
IF(ISOLVE.EQ.1)CALL SOLVE1
IF(ISOLVE.EQ.2)CALL SOLVE2
CALL EIGEN

```

```

        DET0=DET
        EXP0=EXP
        NCOUNT=1
        GO TO 30
10 NCOUNT=NCOUNT+1
C     $$$ C $$$
        K15=K15+1
        ITRACE(K15)='C'
C     CHECK TO SEE IF THE BRANCHING ROUTINE IS TO BEGIN.
        IF(BRANCH.NE.NCOUNT-1) GO TO 15
C     $$$ D $$$
20     K15=K15+1
        ITRACE(K15)='D'
        CALL BPATH
        CALL DEFORM
        CALL STIFF
        CALL VECTOR
        IMETH=METHOD
        METHOD=1
        CALL RWC
        METHOD=IMETH
C     CHECK TO SEE IF DS HAS THE CORRECT SIGN. (DS HAS THE SIGN
C     OF DLAMD.)
        IF(DS.GT.0.DO.AND.LAMDA.LT.LAMDA1) DS=-DS
        IF(DS.LT.0.DO.AND.LAMDA.GT.LAMDA1) DS=-DS
        GO TO 80
C     FIND DQKI FOR THE FIRST ITERATION.
15 CALL DEFORM
        CALL STIFF
C     NOTE: CALLING SOLVE AFTER STIFF ONLY CAUSES THE FORWARD
C     REDUCTION OF THE CONSTANTS.
        IF(ISOLVE.EQ.1)CALL SOLVE1

```

```

      IF(ISOLVE.EQ.2)CALL SOLVE2
C     CHECK TO SEE IF THERE IS A BIFURCATION OR LIMIT POINT.
      CALL EIGEN
C     CHECK TO SEE IF THE CRITICAL POINT IS TO BE LOCATED.
      IF(LFLAG.EQ.2)GOTO 12
      DS=DS/2.00
      BPOINT=1
      GOTO 20
C     $$$ E $$$
12    K15=K15+1
      ITRACE(K15)='E'
30    CALL STORE(LOAD,DISTR,LOAD,0.00)
C     SOLVE K DQKI = DISTR.
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SOLVE2
C     GET THE TANGENT VECTOR.
      CALL INIT
C     CALL THE SUBROUTINE THAT CORRESPONDS TO THE METHOD DESIRED.
      IF(METHOD.EQ.3) GO TO 60
      CALL RWC
      GO TO 70
60    CALL NEWTON
C     WRITE OUT THE POINT DATA.
70    WRITE(7,42) LAMDA
      WRITE(7,41) (Q(I),I=1,NEQS)
41    FORMAT(13F10.6)
80    WRITE(3,1) NCOUNT,LAMDA
      WRITE(3,2)
      WRITE(3,3) (Q(I),I=1,NEQS)
      WRITE(3,32) L
32    FORMAT(' NUMBER OF ITERATIONS =',I3)
      WRITE(3,31) DS

```

```

31 FORMAT('    DS =',G24.16)
C   CONTINUE IF 1) A STABLE POINT IS WANTED (ENDSTB=1)
C   OR 2) A LIMIT OR BIFURCATION POINT IS WANTED
C   (ENDLIM=1)
C   OR 3) NCGOUNT NOT=NPOINT, ENDLIM=2, AND ENDSTB=2.
      IF((NCGOUNT.NE.NPOINT.AND.ENDSTB.NE.1.AND.ENDLIM.NE.1).OR.
1    ENDSTB.EQ.1.OR.ENDLIM.EQ.1) GO TO 10
      IF(ISOLVE.EQ.1)CALL DTNEG1
      IF(ISOLVE.EQ.2)CALL DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
      WRITE(3,11) NNEG,EXP,DET
11   FORMAT(' NUMBER OF NEGATIVE EIGENVALUES =',I3,/, ' DETERMIN',
1     'ANT * 1.D',I3, ' =',G24.16,/)
      WRITE(2,1001)
1001  FORMAT('/', ' TRACE OF PROGRAM PATH',/)
      WRITE(2,1002) (ITRACE(I),I=1,K15)
1002  FORMAT(1X,10A2,5X,10A2,5X,10A2,5X,10A2,5X,10A2,5X)
      WRITE(2,4) NCGOUNT
4     FORMAT('/',/, ' NUMBER OF POINTS =',I4,////, ' ***** NORMAL ',
1     'COMPLETION *****')
      STOP

```

```

C*****
C*                                     BPATH                                     *
C*****
SUBROUTINE BPATH
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
C     $$$ S $$$
K15=K15+1
ITRACE(K15)='S'
KOUNT=0
ISCALE=SCALE
SCALE=2
CALL DEFORM
CALL STIFF
IF(ISOLVE.EQ.1)CALL SOLVE1
IF(ISOLVE.EQ.2)CALL SOLVE2
IF(ISOLVE.EQ.1)CALL DTNEG1

```

```

      IF (ISOLVE.EQ.2) CALL DTNEG2 (SS, LDA, NEQS, KPVT, DET, EXP, NNEG)
10  K=NNEG
      DETOLD=DET
      EXPOLD=EXP
      CALL STORE (LOAD, DISTR, LOAD, 0.00)
      IF (ISOLVE.EQ.1) CALL SOLVE1
      IF (ISOLVE.EQ.2) CALL SOLVE2
      CALL INIT
      CALL RWC
      CALL DEFORM
      CALL STIFF
      IF (ISOLVE.EQ.1) CALL SOLVE1
      IF (ISOLVE.EQ.2) CALL SOLVE2
      IF (ISOLVE.EQ.1) CALL DTNEG1
      IF (ISOLVE.EQ.2) CALL DTNEG2 (SS, LDA, NEQS, KPVT, DET, EXP, NNEG)
C   CHECK TO SEE IF THE CRITICAL POINT WAS PASSED.
      IF (K.EQ.NNEG) GO TO 10
C   CHECK TO SEE IF THE CRITICAL POINT IS FOUND.
15  ALPH=DABS (DET / (DETO * 10 ** (EXPO - EXP)))
      IF (ALPH.LT.EPSLN3) GO TO 30
C   CHECK TO SEE IF THE MAXIMUM NUMBER OF ITERATIONS
C   HAS BEEN REACHED.
      IF (KOUNT.LT.IMAX) GO TO 20
      WRITE (2, 1001)
1001  FORMAT (//, ' TRACE OF PROGRAM PATH', /)
      WRITE (2, 1002) (ITRACE (I), I=1, K15)
1002  FORMAT (1X, 10A2, 5X, 10A2, 5X, 10A2, 5X, 10A2, 5X, 10A2)
      WRITE (2, 11)
11   FORMAT (//, ' ***** ITERATION LIMIT EXCEEDED TO LOCATE',
1     ' THE CRITICAL POINT *****')
      STOP
20   KOUNT=KOUNT+1

```

```

C      PREDICT THE NEXT DS THAT WILL GIVE THE CRITICAL POINT.
C      IT IS ASSUMED THAT EXPOLD=EXP.
      DS=-(DET*DS)/(DET-DETOLD)
      DETOLD=DET
      EXPOLD=EXP
      CALL STORE(LOAD,DISTR,LOAD,0.DO)
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SOLVE2
      CALL INIT
      CALL RWC
      CALL DEFORM
      CALL STIFF
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SOLVE2
      IF(ISOLVE.EQ.1)CALL DTNEG1
      IF(ISOLVE.EQ.2)CALL DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
      WRITE(4,1) KOUNT,LAMDA
1      FORMAT(I3,' *** LAMDA =',G24.16,' ',79('**'))
      WRITE(4,2)
2      FORMAT(' DISPLACEMENT')
      WRITE(4,3) (Q(I),I=1,NEQS)
3      FORMAT(5G24.16)
      WRITE(4,32)NNEG,EXP,DET
32     FORMAT(' NUMBER OF NEGATIVE EIGENVALUES =',I3,/,
1       ' DETERMINANT * 1.0',I3,' =',G24.16,/)
      WRITE(4,33) DS,DLAMD
33     FORMAT(' DS =',G24.16,/, ' DLAMD =',G24.16)
      WRITE(4,34) L
34     FORMAT(' NUMBER OF ITERATIONS =',I3)
      GO TO 15
C      $$$ T $$$
30     K15=K15+1

```

```
      ITRACE(K15)='T'  
      SCALE=ISCALE  
      WRITE(2,31) KOUNT  
31     FORMAT(///,' NUMBER OF ITERATIONS FOR BIF. POINT =',I3)  
C     STOP EXECUTION IF ONLY THE CRITICAL POINT WAS WANTED.  
      IF(BPOINT.EQ.2) RETURN  
        WRITE(2,1001)  
        WRITE(2,1002) (ITRACE(I),I=1,K15)  
        WRITE(2,1003)  
1003    FORMAT(///,' BIFURCATION POINT FOUND')  
      STOP
```

```

C*****
C*                                     CONV*
C*****
SUBROUTINE CONV*
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICoord,NUMBER,LDA,ISOLVE
IFLAG=2
C     CONVERGENCE OCCURS IF THE NORM OF THE REIDUAL FORCES IS LESS
C     THAN EPSLN1 AND IF THE NORM OF THE DISPLACEMENT VECTOR IS LESS
C     THAN EPSLN2.
IF(DSQRT(F(R,R,R,0)).LT.EPSLN1.AND.
1     DSQRT(F(DQ,DQ,R,0)).LT.EPSLN2) IFLAG=1
RETURN

```

```

C*****
C*                                     DATA                                     *
C*****
SUBROUTINE DATA
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
C INPUT PROGRAM CONTROL VARIABLES
READ(1,*)NM,NJ,SYSTEM,UPDATE,IHBW,SCALE,LHAT,LMAX,NPOINT,METHOD,
1     CONT,ENDLIM,ENDSTB,INTEST,BRANCH,IMAX,JMAX,MPOINT,IHHELP,
2     BPOINT,ICOORD,ISOLVE
WRITE(2,1)NM,NJ,SYSTEM,UPDATE,IHBW,SCALE,LHAT,LMAX,NPOINT,METHOD,
1     CONT,ENDLIM,ENDSTB,INTEST,BRANCH,IMAX,JMAX,MPOINT,
2     IHHELP,BPOINT,ICOORD,ISOLVE
1 FORMAT('      NM =',I5,',',I5,',',I5,'      NJ =',I5,',',I5,',',I5,' SYSTEM =',I5,',',I5,',',I5,
1     ' UPDATE =',I5,',',I5,',',I5,' IHBW =',I5,',',I5,',',I5,' SCALE =',I5,',',I5,',',I5,
2     ' LHAT =',I5,',',I5,',',I5,' LMAX =',I5,',',I5,',',I5,' NPOINT =',I5,',',I5,',',I5,
3     ' METHOD =',I5,',',I5,',',I5,' CONT =',I5,',',I5,',',I5,' ENDLIM =',I5,',',I5,',',I5,

```

```

4      * ENDSTB =',15,/, ' INTEST =',15,/, ' BRANCH =',15,/,
5      *   IMAX =',15,/, '   JMAX =',15,/, ' MPOINT =',15,/,
6      *   IHELP =',15,/, ' BPOINT =',15,/, ' ICOORD =',15,/,
7      *   ISOLVE =',15,/)
C      INPUT THE SOLUTION TOLERANCES, THE INITIAL DELTA LAMDA VALUE,
C      LAMDA VALUE, AND THE NUMBER OF NEGATIVE EIGENVELUES.
      READ(1,*) EPSLN1, EPSLN2, EPSLN3, NDIGIT, DS, DLAMD, LAMDA, NNEG, DETO,
1      EXPO, DSBR
      WRITE(2,2) EPSLN1, EPSLN2, EPSLN3, NDIGIT, DS, DLAMD, LAMDA, NNEG, DETO,
1      EXPO, DSBR
2  FORMAT(' EPSLN1 =',G24.16,/, ' EPSLN2 =',G24.16,/,
1      ' EPSLN3 =',G24.16,/, ' NDIGIT =',15,/,
2      '   DS =',G24.16,/, '   DLAMD =',G24.16,/,
3      '   LAMDA =',G24.16,/, '   NNEG =',15,/,
4      '   DETO =',G24.16,/, '   EXPO =',15,/,
5      '   DSBR =',G24.16,/)
C      READ IN THE JOINT COORDINATES.
      READ(1,*) (X(I),Y(I),Z(I),I=1,NJ)
      IF(SYSTEM.EQ.1) WRITE(2,12)
12     FORMAT(/, ' JOINT',6X, 'R',7X, 'THETA',6X, 'PHI')
      IF(SYSTEM.EQ.2) WRITE(2,13)
13     FORMAT(/, ' JOINT',6X, 'X',9X, 'Y',9X, 'Z')
      WRITE(2,4) (I,X(I),Y(I),Z(I),I=1,NJ)
4     FORMAT(I4,2X,3F10.4)
C      SET JCODE EQUAL TO ZERO.
      DO 95 I=1,NJ
          DO 85 J=1,3
              JCODE(I,J)=0
85     CONTINUE
95     CONTINUE
C      READ IN THE CONSTRAINT DATA.
      WRITE(2,86)

```

```

86 FORMAT(/, ' CONSTRAINED JOINT DATA', /, ' JOINT NO. JOINT DIR.', /)
15 READ(1,*) JNUM, JDIR
   IF(JNUM.EQ.0) GO TO 25
      WRITE(2,87) JNUM, JDIR
87   FORMAT(I6, I11)
      IF(JDIR.EQ.4) GO TO 27
         JCODE(JNUM, JDIR)=1
         GO TO 15
27   JCODE(JNUM, 1)=1
      JCODE(JNUM, 2)=1
      JCODE(JNUM, 3)=1
      GO TO 15
C   (IF SPHERICAL COORDINATES CONVERT TO CARTESIAN COORDINATES.
C   CONVERT DEGREE ANGLES TO RADIAN ANGLES)
25 IF(SYSTEM.NE.1) GO TO 35
      DO 70 I=1, NJ
         RADIUS=X(I)
         THETA=Y(I)*.1745329251994330D-01
         PHI=Z(I)*.1745329251994330D-01
         X(I)=RADIUS*DSIN(PHI)*DCOS(THETA)
         Y(I)=RADIUS*DSIN(PHI)*DSIN(THETA)
         Z(I)=-1.0D0*RADIUS*DCOS(PHI)
70   CONTINUE
C   READ IN THE MEMBER PROPERTIES.
35 WRITE(2,5)
   5 FORMAT(/, 2X, 'ELEMENT', 9X, 'A', 9X, 'E', 5X, 'A-END', 5X, 'B-END')
   READ(1,*) A(1), E(1), MINC(1,1), MINC(1,2)
   IF(A(1).GT.0) GO TO 120
      DO 110 I=1, NM
         A(I)=1.0D0
         E(I)=1.0D0
110  CONTINUE

```

```

        READ(1,*) (MINC(I,1),MINC(I,2),I=2,NM)
        GO TO 130
120 READ(1,*) (A(I),E(I),MINC(I,1),MINC(I,2),I=2,NM)
C      COMPUTE JCODE AND NEQS.
130 K=0
      DO 30 I=1,NJ
        DO 20 J=1,3
          IF(JCODE(I,J).EQ.0) GO TO 10
          JCODE(I,J)=0
          GO TO 20
10      K=K+1
          JCODE(I,J)=K
20      CONTINUE
30      CONTINUE
      NEQS=K
      WRITE(7,101) NEQS
101  FORMAT(I4)
      JHBW=0
C      CALCULATE MCODE, XL, AND THE REQUIRED HALF-BAND WIDTH, JHBW.
      DO 80 I=1,NM
        JA=MINC(I,1)
        JB=MINC(I,2)
        XDIF= X(JB)-X(JA)
        YDIF= Y(JB)-Y(JA)
        ZDIF= Z(JB)-Z(JA)
        XL(I)=DSQRT(XDIF*XDIF+YDIF*YDIF+ZDIF*ZDIF)
        MIN=1000
        MAX=0
        DO 90 J=1,3
          MCODE(I,J)=JCODE(JA,J)
          MCODE(I,J+3)=JCODE(JB,J)
          IF(JCODE(JA,J).LT.MIN.AND.JCODE(JA,J).NE.0) MIN=JCODE(JA,J)

```

```

                IF(JCODE(JB,J).LT.MIN.AND.JCODE(JB,J).NE.0) MIN=JCODE(JB,J)
                IF(JCODE(JA,J).GT.MAX) MAX=JCODE(JA,J)
                IF(JCODE(JB,J).GT.MAX) MAX=JCODE(JB,J)
90      CONTINUE
        IF(JHBW.LT.MAX-MIN+1) JHBW=MAX-MIN+1
        WRITE(2,6) I,A(I),E(I),MINC(I,1),MINC(I,2)
        6      FORMAT(I6,5X,2F10.2,I6,I10)
80      CONTINUE
        WRITE(2,11) NEQS
11      FORMAT(////,' NUMBER OF EQUATIONS =',I4)
C      CHECK TO SEE IF THE SPECIFIED HALF-BAND WIDTH IS TOO SMALL.
        IF(IHBW.GE.JHBW) GO TO 100
        WRITE(2,8)
        8      FORMAT(////,' ***** HALF-BAND WIDTH IS TOO SMALL *****')
                STOP
100     IHBW=JHBW
        WRITE(2,9) IHBW
        9      FORMAT(////,' HALF-BAND WIDTH =',I4)
        WRITE(2,7)
        7      FORMAT(//,' JOINT NUMBER JOINT DIRECTION FORCE')
C      INITIALIZE THE DISTRIBUTION VECTOR, DISPLACEMENT VECTOR, AND
C      JOINT DIRECTION COSINES TO ZERO.
        DO 40 I=1,NEQS
            DISTR(I)=0.0D00
            Q(I)=0.0D00
40      CONTINUE
        DO 45 I=1,NJ
            C1(I)=0.0D0
            C2(I)=0.0D0
45      CONTINUE
C      INPUT THE CONSTANT LOAD DISTRIBUTION VECTOR AND THE INITIAL
C      DISPLACEMENT VECTOR.

```

```

50 READ(1,*) JNUM,JDIR,P1
C   CHECK TO SEE IF THIS IS THE END OF THE LOAD DATA.
   IF(JNUM.LT.0) GO TO 60
C   CHECK TO SEE IF THE DISPLACEMENT VECTOR IS TO BE INPUT.
   IF(JNUM.EQ.0) GO TO 200
   WRITE(2,18) JNUM,JDIR,P1
18  FORMAT(I8,I15,G23.10)
   K=JCODE(JNUM,JDIR)
C   CHECK TO SEE IF THERE IS A DATA ERROR.
   IF(K.NE.0) GO TO 55
   WRITE(2,51)
51  FORMAT(/,' DATA ERROR IN FORCE INPUT',/)
   STOP
55  DISTR(K)=P1
   GO TO 50
200 WRITE(2,21)
21  FORMAT(/,' DISPLACEMENT VECTOR',/)
   READ(1,*) (Q(I),I=1,NEQS)
   WRITE(2,22) (Q(I),I=1,NEQS)
22  FORMAT(5G24.16)
C   STOP IF THE PROGRAM IS RUN FOR A DATA CHECK.
60 IF(INTEST.NE.1) RETURN
   WRITE(2,19)
19  FORMAT(/,' ***** PROGRAM IS STOPPED FOR DATA TEST *****')
   STOP

```

```

C*****
C*                                     DEFORM                                     *
C*****
SUBROUTINE DEFORM
IMPLICIT REAL*8 (A-H,G-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1   EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1   LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1   ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1   DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2   IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3   NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4   UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5   EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6   IHELP,BPOINT,LFLAG,ICoord,NUMBER,LDA,ISOLVE
C   CALCULATE THE DEFORMED LENGTHS AND COSINES.
DO 30 I=1,NM
  DO 20 J=1,6
    IF(MCODE(I,J).EQ.0) GO TO 10
    N=MCODE(I,J)
    D(J)=Q(N)
    GO TO 20
10   D(J)=0.0D00
20   CONTINUE
    JA=MINC(I,1)
    JB=MINC(I,2)

```

```

      D63=D(6)-D(3)
      GOTO(21,22),ICOORD
21    D41=D(4)-D(1)
      D52=D(5)-D(2)
      GOTO 23
22    C1I=C1(JA)
      C2I=C2(JA)
      C1HI=C1(JB)
      C2HI=C2(JB)
      D41=C1HI*D(4)-C2HI*D(5)-C1I*D(1)+C2I*D(2)
      D52=C2HI*D(4)+C1HI*D(5)-C2I*D(1)-C1I*D(2)
23    XLD1=X(JB)-X(JA)+D41
      XLD2=Y(JB)-Y(JA)+D52
      XLD3=Z(JB)-Z(JA)+D63
      XLD(I)=DSQRT(XLD1*XLD1+XLD2*XLD2+XLD3*XLD3)
      CG1(I)=XLD1/XLD(I)
      CG2(I)=XLD2/XLD(I)
      CG3(I)=XLD3/XLD(I)
30   CONTINUE
      IF(ICOORD.EQ.1.OR.NUMBER.EQ.2)RETURN
      DO 40 I=1,NJ
      RAD=DSQRT(X(I)**2+Y(I)**2)
      IF(RAD.NE.0.D0)GOTO 50
      C1(I)=1.D0
      C2(I)=0.D0
      GOTO 40
50    C1(I)=X(I)/RAD
      C2(I)=Y(I)/RAD
40   CONTINUE
      NUMBER=2
      RETURN

```

```

C*****
C*                               DTNEG1                               *
C*****
SUBROUTINE DTNEG1
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
  NNEG=0
  EXP=0
  DET=1.00
  DO 20 I=1,NEQS
    IF(SS(I,1).LT.0.00) NNEG=NNEG+1
    DET=DET*SS(I,1)
    IF(DABS(DET).LT.1.030) GO TO 10
    EXP=EXP-30
    DET=DET*1.0-30
    GO TO 20
10  IF(DABS(DET).GT.1.0-30) GO TO 20

```

EXP=EXP+30  
DET=DET\*1.D30

20 CONTINUE  
RETURN

```

C*****
C                                     DTNEG2                                     *
C*****
SUBROUTINE DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
IMPLICIT REAL*8 (A-H,C-Z)
REAL*8 SS(LDA,NEQS)
INTEGER KPVT(1),EXP
NNEG=0
DET=1.00
EXP=0
TEN=10.00
T=0.00
DO 130 K=1,NEQS
D=SS(K,K)
IF(KPVT(K).GT.0)GOTO 50
IF(T.NE.0.00)GOTO 30
T=DABS(SS(K,K+1))
D=(D/T)*SS(K+1,K+1)-T
GOTO 40
30 CONTINUE
D=T
T=0.00
40 CONTINUE
50 CONTINUE
IF(D.LT.0.00)NNEG=NNEG+1
DET=D*DET
IF(DET.EQ.0.00)GOTO 110
70 IF(DABS(DET).GE.1.00)GOTO 80
DET=TEN*DET
EXP=EXP-1
GOTO 70
80 CONTINUE

```

```
90    IF(DABS(DET).LT.TEN)GOTO 100
      DET=DET/TEN
      EXP=EXP+1
      GOTO 90
100   CONTINUE
110   CONTINUE
130   CONTINUE
      EXP=-EXP
      RETURN
```

```

C*****
C*                                     EIGEN                                     *
C*****
SUBROUTINE EIGEN
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
  DIMENSION XTEMP(70)
C     $$$ J $$$
  K15=K15+1
  ITRACE(K15)='J'
  K=NNEG
  IF(ISOLVE.EQ.1)CALL DTNEG1
  IF(ISOLVE.EQ.2)CALL DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
  WRITE(3,11) NNEG,EXP,DET
11  FORMAT(' NUMBER OF NEGATIVE EIGENVALUES =',I3,/, ' DETERMINANT ',
1     '* 1.D',I3,' =',G24.16,/)
C     RETURN IF BRANCHING IS TO OCCUR.

```

```

      IF(BRANCH.EQ.NCOUNT-1) RETURN
C     RETURN IF THERE ARE (AND WERE) NO NEGATIVE EIGENVALUES.
      IF(K.EQ.0.AND.NNEG.EQ.0) RETURN
C     $$$ K $$$
      K15=K15+1
      ITRACE(K15)='K'
C     CHECK TO SEE IF THE PROGRAM WAS RUN WITH ENDLIM=1.
C     IF SO, BRANCH IF THERE IS (ARE) NEGATIVE EIGENVALUE(S).
90    IF(NNEG.LE.0.OR.ENDLIM.EQ.2) GO TO 30
C     $$$ L $$$
      K15=K15+1
      ITRACE(K15)='L'
C     STOP IF NO MORE POINTS ARE WANTED.
      IF(NPOINT.GT.0) GO TO 20
          WRITE(2,1001)
1001   FORMAT(/, ' TRACE OF PROGRAM PATH',/)
          WRITE(2,1002) (ITRACE(I),I=1,K15)
1002   FORMAT(1X,10A2,5X,10A2,5X,10A2,5X,10A2,5X,10A2)
          WRITE(2,4) NCOUNT-1
          4   FORMAT(/, ' NUMBER OF POINTS =',I3,/, ' ***** PROGRAM IS',
1           ' STOPPED AT OR PAST A LIMIT OR BIFURCATION',
2           ' POINT AS WAS REQUESTED *****')
          STOP
20     NPOINT=NPOINT-1
C     RETURN IF THE NUMBER OF NEGATIVE EIGENVALUES IS THE SAME.
30    IF(K.EQ.NNEG) RETURN
C     CHECK TO SEE IF THE TRUSS IS BECOMING STABLE.
      IF(K.LT.NNEG) GO TO 50
C     $$$ M $$$
      K15=K15+1
      ITRACE(K15)='M'
      WRITE(2,1) NCOUNT-1

```

```

1   FORMAT(/, ' AT POINT', I3, ', THE TRUSS IS BECOMING MORE ',
1   ' STABLE')
C   STOP IF THE TRUSS IS STABLE AND CONT=2.
   IF(NNEG.NE.0.OR.CONT.EQ.1) GO TO 60
   WRITE(2,1001)
   WRITE(2,1002) (ITRACE(I), I=1, K15)
   WRITE(2,3) NCOUNT-1
3   FORMAT(/, ' ***** PROGRAM IS STOPPED AT POINT', I3, ', A ',
1   ' STABLE POINT *****')
   STOP
C   AT THIS POINT, A LIMIT OR BIFURCATION POINT WAS ENCOUNTERED.
50  WRITE(2,2) NCOUNT-2, NCOUNT-1
   2 FORMAT(/, ' A LIMIT POINT OR BIFURCATION POINT WAS ENCOUNTERED', /
1   ' BETWEEN POINT', I3, ' AND POINT', I3, ', ', //)
C   CHECK TO SEE IF THE CRITICAL POINT IS TO BE LOCATED.
60  IF(BPOINT.NE.3) GO TO 55
   LFLAG=1
   RETURN
C   $$$ N $$$
55  K15=K15+1
   ITRACE(K15)='N'
   LFLAG=2
C   RETURN IF THE NEWTON-RAPHSON METHOD IS USED.
   IF(METHOD.EQ.3) RETURN
   IF(NCOUNT.NE.1) GO TO 61
   CALL STORE(LOAD, DISTR, LOAD, 0.00)
   IF(ISOLVE.EQ.1) CALL SOLVE1
   IF(ISOLVE.EQ.2) CALL SOLVE2
   DS=DLAMD*DSQRT(F(LOAD, LOAD, LOAD, 0)+1.00)
C   $$$ D $$$
61  K15=K15+1
   ITRACE(K15)='D'

```

```

        IF(IHELP.EQ.2) GO TO 505
        IF(IHELP.EQ.3) RETURN
        IHELP=2
        RETURN
C      RECORD THE DISPLACEMENT VECTOR AND THE LOAD LEVEL.
505    CALL STORE(XTEMP,Q,Q,0.DO)
        XLAMDA=LAMDA
        CALL STORE(LOAD,DISTR,LOAD,0.DO)
C      SOLVE K DQKI = DISTR.
        IF(ISOLVE.EQ.1)CALL SOLVE1
        IF(ISOLVE.EQ.2)CALL SOLVE2
C      ASSUME DS = -DS.
        DS=-1.DO*DS
C      PREVENT SCALING AND ENLARGE DS TO ENSURE THAT THE ASSUMPTION
C      THAT DS=-DS WILL PROVE WRONG IF IT IS WRONG (THE PROCEDURE MAY
C      'BACK' ON ITSELF).
        ISCALE=SCALE
C      PREVENT SCALING SO THAT DS REMAINS UNCHANGED.
        SCALE=2
        DS=1.100*DS
C      GET THE TANGENT VECTOR.
        CALL INIT
C      RETAIN THE VARIABLES AS THEY WERE.
        SCALE=ISCALE
        CALL RWC
        CALL DEFORM
        CALL STIFF
        IF(ISOLVE.EQ.1)CALL SOLVE1
        IF(ISOLVE.EQ.2)CALL SOLVE2
C      CALCULATE THE NUMBER OF NEGATIVE EIGENVALUES.
        K=NNEG
        IF(ISOLVE.EQ.1)CALL DTNEGL

```

```

C      IF(ISOLVE.EQ.2)CALL DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
C      CHECK TO SEE IF THE NUMBER OF EIGENVALUES HAS DECREASED.
C      IF(K.LE.NNEG) GO TO 80
C      $$$ P $$$
C      K15=K15+1
C      ITRACE(K15)='P'
C      THE ASSUMPTION THAT DS = -DS WAS WRONG. RETURN TO THE
C      MAINLINE.
C      CALL STORE(Q,XTEMP,Q,0.00)
C      LAMDA=XLAMDA
C      DS=-1.00*DS
C      NNEG=K
C      RETURN
C      WRITE OUT THE POINT DATA.
80 WRITE(3,9) NCOUNT,LAMDA
   9 FORMAT(I3,' LAMDA =',G24.16,' ',83('*'))
   WRITE(3,5)
   5 FORMAT(' DISPLACEMENT')
   WRITE(3,6) (Q(I),I=1,NEQS)
   WRITE(3,17) L
  17 FORMAT(' NUMBER OF ITERATIONS =',I3)
   6 FORMAT(5G24.16)
   WRITE(3,12) DS
  12 FORMAT(' DS =',G24.16)
   DS=DS/1.100
   WRITE(3,11) NNEG,EXP,DET
   WRITE(7,42) LAMDA
  42 FORMAT(F15.9)
   WRITE(7,41) (Q(I),I=1,NEQS)
  41 FORMAT(13F10.6)
   NCOUNT=NCOUNT+1
C      $$$ Q $$$

```

```

K15=K15+1
ITRACE(K15)='Q'
C RETURN IF BRANCHING IS TO OCCUR.
IF(BRANCH.EQ.NCOUNT-1) RETURN
C CONTINUE IF 1) A STABLE POINT IS WANTED (ENDSTB=1)
C OR 2) A LIMIT OR BIFURCATION POINT IS WANTED
C (ENDLIM=1)
C OR 3) NCOUNT=NPOINT, ENDLIM=2, AND ENDSTB=2.
IF(ENDLIM.EQ.1.OR.ENDSTB.EQ.1.OR.(NCOUNT-1.NE.NPOINT.AND.
1 ENDSTB.NE.1.AND.ENDLIM.NE.1)) GO TO 90
WRITE(2,1001)
WRITE(2,1002) (ITRACE(I),I=1,K15)
WRITE(2,7) NCOUNT-1
7 FORMAT(//,' NUMBER OF POINTS =',I3,//,' ***** NORMAL ',
1 'COMPLETION ****')
STOP

```

```

C*****
C*                                     F                                     *
C*****
FUNCTION F(X1,Y1,Z1,K)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CUNT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CUNT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
DIMENSION X1(NEQS),Y1(NEQS),Z1(NEQS)
C     F IS THE DOT PRODUCT OF X1 ( Y1 + K * Z1 )
F=0.00
IF(K.EQ.0) GO TO 20
DO 10 I=1,NEQS
    F=F+(X1(I)*(Y1(I)+DFLOAT(K)*Z1(I)))
10  CONTINUE
    RETURN
20 DO 30 I=1,NEQS
    F=F+X1(I)*Y1(I)
30  CONTINUE

```

RETURN

```

C*****
C*                               INIT                               *
C*****
SUBROUTINE INIT
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1 EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1 LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ AI(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1 ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ DI(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1 DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2 IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3 NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4 UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5 EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6 IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
C CHECK TO SEE IF THIS IS THE FIRST ITERATION FOR A POINT.
IF(DS.EQ.0.00) GO TO 30
C FIND THE INCREMENT IN LOAD LEVEL GIVEN DS.
IF(SCALE.NE.1) GO TO 20
C SCALE THE TANGENT VECTOR.
DS=DS*DSQRT(DFLOAT(LHAT)/DFLOAT(L))
20 DLAMD=DS/DSQRT(F(LOAD,LOAD,LOAD,0)+1.00)
GO TO 40
C FIND THE LENGTH IN THE TANGENT VECTOR, DS, GIVEN THE INITIAL
C INCREMENT IN LOAD LEVEL.
30 DS=DLAMD*DSQRT(F(LOAD,LOAD,LOAD,0)+1.00)

```

```
40 DO 50 I=1,NEQS
C   FIND THE INITIAL DQ.
      DQ(I)=DLAMD*LOAD(I)
50 CONTINUE
C   SAVE THE INITIAL Q VECTOR AND LAMDA FOR THE SPHERE ITERATION.
      LAMDA1=LAMDA
      CALL STORE(Q1,Q,Q,0.DO)
C   UPDATE THE VARIABLES FOR THE FIRST ITERATION.
      LAMDA=LAMDA+DLAMD
      CALL STORE(Q,Q,DQ,1.DO)
      RETURN
```

```

C*****
C*                                     NEWTON                                     *
C*****
SUBROUTINE NEWTON
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCCUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE

C     $$$ H $$$
K15=K15+1
ITRACE(K15)='H'
C     UPDATE FOR THE FIRST ITERATION.
L=1
CALL DEFORM
CALL RESID
GO TO 50
C     COMPUTE THE STIFFNESS MATRIX IF 1) IT IS THE FIRST ITERATION
C                                         OR 2) UPDATING IS DESIRED.
20 IF(L.NE.1.AND.UPDATE.NE.1) GO TO 30

```

```

C      $$$ I $$$
      K15=K15+1
      ITRACE(K15)='I'
C      UPDATE THE STIFFNESS MATRIX.
      CALL STIFF
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SGLVE2
30    CALL STORE(LOAD,R,R,0.DO)
C      SOLVE K DQKI = R.
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SOLVE2
C      UPDATE THE VARIABLES.
      CALL STORE(DQ,LOAD,Q,0.DO)
      CALL STORE(Q,Q,DQ,1.DO)
      L=L+1
      CALL DEFORM
      CALL RESID
C      CHECK FOR CONVERGENCE.
50    CALL CONVG
      IF(IFLAG.EQ.1) RETURN
C      CHECK TO SEE IF THE MAXIMUM NUMBER OF ITERATIONS HAS BEEN
C      REACHED.
      IF(L.NE.LMAX) GO TO 20
      IF(ISOLVE.EQ.1)CALL DTNEG1
      IF(ISOLVE.EQ.2)CALL DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
      WRITE(3,11) NNEG,EXP,DET
11    FORMAT(' NUMBER OF NEGATIVE EIGENVALUES =',I3,/, ' DETERMIN',
1      ' ANT * 1.0',I3,' =',G24.16,/)
      WRITE(2,1001)
1001  FORMAT('/', ' TRACE OF PROGRAM PATH',/)
      WRITE(2,1002) (ITRACE(I),I=1,K15)
1002  FORMAT(1X,10A2,5X,10A2,5X,10A2,5X,10A2,5X,10A2,5X)

```

```
WRITE(2,1) NCOUNT
1  FORMAT(////,' NUMBER OF POINTS =',I4,////,' ***** ITERATION ',
1  'LIMIT EXCEEDED *****')
STOP
```

```

C*****
C*                                     NORMAL                                     *
C*****
SUBROUTINE NORMAL
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPV(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICORD,NUMBER,LDA,ISOLVE
  DLAMD=-F(DQ1,DQ,DQ,0)/(F(DQ1,DQI,DQ,0)+DLAMD1)
  CALL STORE(DQ,DQ,DQI,DLAMD)
C  UPDATE THE VARIABLES.
  LAMDA=LAMDA+DLAMD
  CALL STORE(Q,Q,DQ,1.DO)
  RETURN

```

```

C*****
C*                                     POWINV                                     *
C*****
      SUBROUTINE POWINV
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1      EXP,EXPO,EXPOLD,EXPNEW,BPOINT
      REAL*8 LOAD,LAMDA,LAMDA1
      CHARACTER*2 ITRACE
      COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1      LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
      COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1      ,MCODE(70,6),MINC(70,2)
      COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
      COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1      DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2      IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3      NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4      UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5      EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6      IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE

C      $$$ U $$$
      K15=K15+1
      ITRACE(K15)='U'
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SOLVE2

C      THIS SUBROUTINE USES THE INVERSE POWER METHOD TO ESTIMATE THE
C      EIGENVECTOR CORRESPONDING TO THE SMALLEST EIGENVALUE.
      KOUNT=0

C      CALCULATE THE INITIAL EIGENVALUE APPROXIMATION.
      DO 10 I=1,NEQS
          LOAD(I)=1.00/DFLOAT(NEQS)

```

```

        R1(I)=LOAD(I)
10  CONTINUE
    IF(ISOLVE.EQ.1)CALL SOLVE1
    IF(ISOLVE.EQ.2)CALL SOLVE2
    TEMP=F(LOAD,R1,R1,0)
    YSCALE=DSQRT(F(LOAD,LOAD,LOAD,0))
    ESTOLD=1.00/TEMP
C   INVERSE POWER METHOD ITERATION WITH SCALING (ALPH=0).
15  KOUNT=KCOUNT+1
    DO 20 I=1,NEQS
        LOAD(I)=LOAD(I)/YSCALE
        R1(I)=LOAD(I)
20  CONTINUE
    IF(ISOLVE.EQ.1)CALL SOLVE1
    IF(ISOLVE.EQ.2)CALL SOLVE2
    TEMP=F(LOAD,R1,R1,0)
    YSCALE=DSQRT(F(LOAD,LOAD,LOAD,0))
    ESTNEW=1.00/TEMP
C   SEE IF THE NUMBER OF ITERATIONS IS THE MAXIMUM ALLOWED.
    IF(KOUNT.GE.JMAX) GO TO 50
        ESTEMP=ESTNEW
201  IF(DABS(ESTNEW).GT.0.00) GO TO 202
        ESTNEW=ESTNEW*10.00
        ESTOLD=ESTOLD*10.00
        GO TO 201
202  I1=IFIX(SNGL(ESTNEW*10**NDIGIT))
        I2=IFIX(SNGL(ESTOLD*10**NDIGIT))
        IF(I1.EQ.I2) GO TO 30
        ESTOLD=ESTEMP
        GO TO 15
30  WRITE(2,1) ESTEMP
1   FORMAT(//,' EIGENVALUE =',G24.16,//,' EIGENVECTOR',/)

```

```

        DO 40 I=1,NEQS
          R1(I)=LOAD(I)/YSCALE
40      CONTINUE
        G10=0.00
        WRITE(2,2) (R1(I),I=1,NEQS),G10
        2      FORMAT(5G24.16)
        WRITE(2,39) KOUNT
39      FORMAT(//,' NUMBER OF ITERATIONS TO FIND THE EIGENVECTOR =',
1        I3,///)
        RETURN
50      WRITE(2,1001)
1001     FORMAT(//,' TRACE OF PROGRAM PATH',/)
        WRITE(2,1002) (ITRACE(I),I=1,K15)
1002     FORMAT(1X,10A2,5X,10A2,5X,10A2,5X,10A2,5X,10A2,5X)
        WRITE(2,3)
        3      FORMAT(//,' ITERATION LIMIT EXCEEDED TO FIND THE EIGENVECTOR')
        STOP

```

```

C*****
C*                               RESID                               *
C*****
SUBROUTINE RESID
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
C     INITIALIZE THE RESIDUAL FORCES TO THE CURRENT LOAD LEVEL.
DO 10 I=1,NEQS
    R(I)=LAMDA*DISTR(I)
10 CONTINUE
C     SUBTRACT THE FORCES THAT COME FROM THE JOINT DISPLACEMENTS FROM
C     THE RESIDUAL FCRCE VECTOR.
DO 20 I=1,NM
    ELONG=XLD(I)-XL(I)
    PP1=A(I)*E(I)*ELONG*CG1(I)/XL(I)
    PP2=A(I)*E(I)*ELONG*CG2(I)/XL(I)
    PP3=A(I)*E(I)*ELONG*CG3(I)/XL(I)

```

```

      D(3)=-1.000*PP3
      D(6)=PP3
      GOTO(11,12),ICoord
11    D(1)=-1.000*PP1
      D(2)=-1.000*PP2
      D(4)=PP1
      D(5)=PP2
      GOTO 13
12    I1=MINC(I,1)
      I2=MINC(I,2)
      C1I=C1(I1)
      C2I=C2(I1)
      C1HI=C1(I2)
      C2HI=C2(I2)
      D(1)=-1.000*PP1*C1I-PP2*C2I
      D(2)=PP1*C2I-PP2*C1I
      D(4)=PP1*C1HI+PP2*C2HI
      D(5)=PP2*C1HI-PP1*C2HI
13    DO 30 J=1,6
      IF(MCODE(I,J).EQ.0) GO TO 30
      ND=MCODE(I,J)
      R(ND)=R(ND)-D(J)
30    CONTINUE
20    CONTINUE
      RETURN

```

```

C*****
C*                                     RWC                                     *
C*****
  SUBROUTINE RWC
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1      EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1      LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1      ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1      DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2      IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3      NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4      UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5      EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6      IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE

C      $$$ F $$$
      K15=K15+1
      ITRACE(K15)='F'
C      SAVE THE FIRST LAMDA AND Q VECTOR FOR THE NORMAL PLANE ITERATION
      CALL STORE(DQ1,DQ,DQ,0.00)
      DLAMD1=DLAMD
      L=1
      CALL DEFORM
      CALL RESID
      GO TO 50
C      COMPUTE THE STIFFNESS MATRIX AND SOLVE K DQKI = DISTR IF IT IS

```

```

C     THE FIRST ITERATION OR IF UPDATING IS WANTED.
20  IF(L.NE.1.AND.UPDATE.NE.1) GO TO 30
C     $$$ G $$$
C     K15=K15+1
C     ITRACE(K15)='G'
C     UPDATE THE STIFFNESS MATRIX.
C     CALL STIFF
C     IF(ISOLVE.EQ.1)CALL SOLVE1
C     IF(ISOLVE.EQ.2)CALL SOLVE2
C     CALL STORE(LOAD,DISTR,LOAD,0.DO)
C     SOLVE K DQKI = DISTR.
C     IF(ISOLVE.EQ.1)CALL SOLVE1
C     IF(ISOLVE.EQ.2)CALL SCLVE2
C     TEMPORARILY STORE DQKI IN DQI.
C     CALL STORE(DQI,LOAD,DQI,0.DO)
30  CALL STORE(LOAD,R,R,0.DO)
C     SOLVE K DQKII = R.
C     IF(ISOLVE.EQ.1)CALL SOLVE1
C     IF(ISOLVE.EQ.2)CALL SOLVE2
C     TEMPORARILY STORE DQKII IN DQ.
C     CALL STORE(DQ,LOAD,DQ,0.DO)
C     CALL THE APPROPRIATE SUBROUTINE TO GET DLAMD.
C     IF(METHOD.EQ.1) CALL NGRMAL
C     IF(METHOD.EQ.2) CALL SPHERE
C     L=L+1
C     CALL DEFORM
C     CALL RESID
C     CHECK FOR CONVERGENCE.
50  CALL CONVG
C     IF(IFLAG.EQ.1) RETURN
C     CHECK TO SEE IF THE MAXIMUM NUMBER OF ITERATIONS HAS BEEN
C     REACHED.

```

```

IF(L.NE.LMAX) GO TO 20
  IF(ISOLVE.EQ.1)CALL DTNEG1
  IF(ISOLVE.EQ.2)CALL DTNEG2(SS,LDA,NEQS,KPVT,DET,EXP,NNEG)
  WRITE(3,11) NNEG,EXP,DET
11  FORMAT(' NUMBER OF NEGATIVE EIGENVALUES =',I3,/, ' DETERMIN',
1    ' ANT * 1.0',I3, ' =',G24.16,/)
  WRITE(2,1001)
1001 FORMAT(/, ' TRACE OF PROGRAM PATH',/)
  WRITE(2,1002) (ITRACE(I),I=1,K15)
1002 FORMAT(1X,10A2,5X,10A2,5X,10A2,5X,10A2,5X,10A2,5X)
  WRITE(2,1) NCOUNT
1  FORMAT(////, ' NUMBER OF POINTS =',I4,////, ' ***** ITERATION ',
1    'LIMIT EXCEEDED *****')
  STOP

```

```

C*****
C*                                     SOLVE1                                     *
C*****
SUBROUTINE SCLVE1
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
C FORWARD REDUCTION OF MATRIX
IF(JFLAG.EQ.1) GO TO 40
DO 10 N=1,NEQS
DO 20 LL=2,IHBW
IF(SS(N,LL).EQ.0.DO) GO TO 20
I=N+LL-1
C=SS(N,LL)/SS(N,1)
J=0
DO 30 K=LL,IHBW
J=J+1
SS(I,J)=SS(I,J)-C*SS(N,K)

```

```

30             CONTINUE
               SS(N,LL)=C
20     CONTINUE
10     CONTINUE
       JFLAG=1
       RETURN
C     FORWARD REDUCTION OF CONSTANTS
40 DO 50 N=1,NEQS
      DO 60 LL=2,IHBM
        IF(SS(N,LL).EQ.0.DO) GO TO 60
        I=N+LL-1
        LOAD(I)=LOAD(I)-SS(N,LL)*LOAD(N)
60     CONTINUE
      LOAD(N)=LOAD(N)/SS(N,1)
50 CONTINUE
C     SOLVE FOR UNKNOWN BY BACK-SUBSTITUTION
DO 80 M=2,NEQS
  N=NEQS+1-M
  DO 70 LL=2,IHBM
    IF(SS(N,LL).EQ.0.DO) GO TO 70
    K=N+LL-1
    LOAD(N)=LOAD(N)-SS(N,LL)*LOAD(K)
70     CONTINUE
80 CONTINUE
   RETURN

```

```

C*****
C*                               SOLVE2                               *
C*****
SUBROUTINE SOLVE2
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
  IF(JFLAG.EQ.1)GOTO 40

C
C     FORWARD REDUCTION OF COEFFICIENT MATRIX
C
  CALL SSIFA(SS,LDA,NEQS,KPVT,INFO)
  IF(INFO.EQ.0)GOTO 30
  WRITE(2,50)
50 FORMAT(///,'* * * MATRIX IS SINGULAR--EXECUTION STOPPED TO AVOID E
  IRROR * * *')
  STOP
30 JFLAG=1

```

```
      RETURN  
C  
C      DETERMINATION OF SOLUTION VECTOR  
C  
40 CALL SSISL(SS,LDA,NEQS,KPVT,LOAD)  
      RETURN
```

```

C*****
C*                                     SOLVE2                                     *
C*****
      SUBROUTINE SOLVE2
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1      EXP,EXPO,EXPOLD,EXPNEW,BPOINT
      REAL*8 LOAD,LAMDA,LAMDA1
      CHARACTER*2 ITRACE
      COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1      LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
      COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1      ,MCODE(70,6),MINC(70,2)
      COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
      COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1      DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2      IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3      NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4      UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5      EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6      IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
      IF(JFLAG.EQ.1)GOTO 40

C
C      FORWARD REDUCTION OF COEFFICIENT MATRIX
C
      CALL SSIFA(SS,LDA,NEQS,KPVT,INFO)
      IF(INFO.EQ.0)GOTO 30
      WRITE(2,50)
50  FORMAT(///,'* * * MATRIX IS SINGULAR—EXECUTION STOPPED TO AVOID E
      RRROR * * *')
      STOP
30  JFLAG=1

```

RETURN

C  
C  
C

DETERMINATION OF SOLUTION VECTOR

40 CALL SSISL(SS,LDA,NEQS,KPVT,LOAD)  
RETURN

```

C*****
C                               SUBROUTINE SSIFA                               *
C*****
      SUBROUTINE SSIFA(SS,LDA,NEQS,KPVT,INFO)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER KPVT(NEQS)
      REAL*8 SS(LDA,NEQS),MULK,MULKM1
      LOGICAL SWAP
      ALPHA=(1.00+DSQRT(17.00))/8.00
      INFO=0
      K=NEQS
10  CONTINUE
      IF(K.EQ.0)GOTO 200
      IF(K.GT.1)GOTO 20
      KPVT(1)=1
      IF(SS(1,1).EQ.0.00)INFO=1
      GOTO 200
20  CONTINUE
      KM1=K-1
      ABSAKK=DABS(SS(K,K))
      IMAX=ISAMAX(K-1,SS(1,K),1,NEQS)
      COLMAX=DABS(SS(IMAX,K))
      IF(ABSAKK.LT.ALPHA*COLMAX)GOTO 30
      KSTEP=1
      SWAP=.FALSE.
      GOTO 90
30  CONTINUE
      ROWMAX=0.00
      IMAXP1=IMAX+1
      DO 40 J=IMAXP1,K
         ROWMAX=DMAX1(ROWMAX,DABS(SS(IMAX,J)))
40  CONTINUE

```

```

      IF(IMAX.EQ.1)GOTO 50
      JMAX=ISAMAX(IMAX-1,SS(1,IMAX),1,NEQS)
      ROWMAX=DMAX1(ROWMAX,DABS(SS(JMAX,IMAX)))
50  CONTINUE
      IF(DABS(SS(IMAX,IMAX)).LT.ALPHA*ROWMAX)GOTO 60
      KSTEP=1
      SWAP=.TRUE.
      GOTO 80
60  CONTINUE
      IF(ABSAKK.LT.ALPHA*COLMAX*(COLMAX/ROWMAX))GOTO 70
      KSTEP=1
      SWAP=.FALSE.
      GOTO 80
70  CONTINUE
      KSTEP=2
      SWAP=IMAX.NE.KM1
80  CONTINUE
90  CONTINUE
      IF(DMAX1(ABSAKK,COLMAX).NE.0.DO)GOTO 100
      KPVT(K)=K
      INFO=K
      GOTO 190
100 CONTINUE
      IF(KSTEP.EQ.2)GOTO 140
      IF(.NOT.SWAP)GOTO 120
      CALL SSWAP(IMAX,SS(1,IMAX),1,SS(1,K),1,NEQS)
      DO 110 JJ=IMAX,K
      J=K+IMAX-JJ
      T=SS(J,K)
      SS(J,K)=SS(IMAX,J)
      SS(IMAX,J)=T
110  CONTINUE

```

```

120 CONTINUE
    DO 130 JJ=1,KM1
        J=K-JJ
        MULK=-SS(J,K)/SS(K,K)
        T=MULK
        CALL SAXPY(J,T,SS(1,K),1,SS(1,J),1,NEQS)
        SS(J,K)=MULK
130 CONTINUE
    KPVT(K)=K
    IF(SWAP)KPVT(K)=IMAX
    GOTO 190
140 CONTINUE
    IF(.NOT.SWAP)GOTO 160
    CALL SSWAP(IMAX,SS(1,IMAX),1,SS(1,K-1),1,NEQS)
    DO 150 JJ=IMAX,KM1
        J=KM1+IMAX-JJ
        T=SS(J,K-1)
        SS(J,K-1)=SS(IMAX,J)
        SS(IMAX,J)=T
150 CONTINUE
    T=SS(K-1,K)
    SS(K-1,K)=SS(IMAX,K)
    SS(IMAX,K)=T
160 CONTINUE
    KM2=K-2
    IF(KM2.EQ.0)GOTO 180
    AK=SS(K,K)/SS(K-1,K)
    AKM1=SS(K-1,K-1)/SS(K-1,K)
    DENOM=1.00-AK*AKM1
    DO 170 JJ=1,KM2
        J=KM1-JJ
        BK=SS(J,K)/SS(K-1,K)

```

```

      BKMI=SS(J,K-1)/SS(K-1,K)
      MULK=(AKMI*BK-BKMI)/DENOM
      MULKMI=(AKMI*BKMI-BK)/DENOM
      T=MULK
      CALL SAXPY(J,T,SS(1,K),1,SS(1,J),1,NEQS)
      T=MULKMI
      CALL SAXPY(J,T,SS(1,K-1),1,SS(1,J),1,NEQS)
      SS(J,K)=MULK
      SS(J,K-1)=MULKMI
170   CONTINUE
180   CONTINUE
      KPVT(K)=1-K
      IF(SWAP)KPVT(K)=-IMAX
      KPVT(K-1)=KPVT(K)
190   CONTINUE
      K=K-KSTEP
      GOTO 10
200   CONTINUE
      RETURN

```

```

C*****
C                               SUBROUTINE SSISL                               *
C*****
      SUBROUTINE SSISL(SS,LDA,NEQS,KPVT,LOAD)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER KPVT(NEQS)
      REAL*8 SS(LDA,NEQS),LOAD(NEQS)
      K=NEQS
10  IF(K.EQ.0)GOTO 80
      IF(KPVT(K).LT.0)GOTO 40
      IF(K.EQ.1)GOTO 30
      KP=KPVT(K)
      IF(KP.EQ.K)GOTO 20
      TEMP=LOAD(K)
      LOAD(K)=LOAD(KP)
      LOAD(KP)=TEMP
20  CONTINUE
      CALL SAXPY(K-1,LOAD(K),SS(1,K),1,LOAD(1),1,NEQS)
30  CONTINUE
      LOAD(K)=LOAD(K)/SS(K,K)
      K=K-1
      GOTO 70
40  CONTINUE
      IF(K.EQ.2)GOTO 60
      KP=IABS(KPVT(K))
      IF(KP.EQ.K-1)GOTO 50
      TEMP=LOAD(K-1)
      LOAD(K-1)=LOAD(KP)
      LOAD(KP)=TEMP
50  CONTINUE
      CALL SAXPY(K-2,LOAD(K),SS(1,K),1,LOAD(1),1,NEQS)
      CALL SAXPY(K-2,LOAD(K-1),SS(1,K-1),1,LOAD(1),1,NEQS)

```

```

60 CONTINUE
   AK=SS(K,K)/SS(K-1,K)
   AKM1=SS(K-1,K-1)/SS(K-1,K)
   BK=LOAD(K)/SS(K-1,K)
   BKM1=LOAD(K-1)/SS(K-1,K)
   DENOM=AK*AKM1-1.00
   LOAD(K)=(AKM1*BK-BKM1)/DENOM
   LOAD(K-1)=(AK*BKM1-BK)/DENOM
   K=K-2
70 CONTINUE
   GOTO 10
80 CONTINUE
   K=1
90 IF(K.GT.NEQS)GOTO 160
   IF(KPVT(K).LT.0)GOTO 120
   IF(K.EQ.1)GOTO 110
   LOAD(K)=LOAD(K)+SDOT(K-1,SS(1,K),1,LOAD(1),1,NEQS)
   KP=KPVT(K)
   IF(KP.EQ.K)GOTO 100
   TEMP=LOAD(K)
   LOAD(K)=LOAD(KP)
   LOAD(KP)=TEMP
100 CONTINUE
110 CONTINUE
   K=K+1
   GOTO 150
120 CONTINUE
   IF(K.EQ.1)GOTO 140
   LOAD(K)=LOAD(K)+SDOT(K-1,SS(1,K),1,LOAD(1),1,NEQS)
   LOAD(K+1)=LOAD(K+1)+SDOT(K-1,SS(1,K+1),1,LOAD(1),1,NEQS)
   KP=IABS(KPVT(K))
   IF(KP.EQ.K)GOTO 130

```

```
    TEMP=LOAD(K)
    LOAD(K)=LOAD(KP)
    LOAD(KP)=TEMP
130 CONTINUE
140 CONTINUE
    K=K+2
150 CONTINUE
    GOTD 90
160 CONTINUE
    RETURN
```

```

C*****
C                               FUNCTION SDOT                               *
C*****
      FUNCTION SDOT(N,SX,INCX,SY,INCY,NEQS)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 SX(NEQS),SY(NEQS)
      STEMP=0.000
      SDOT=0.000
      IF(N.LE.0) GO TO 70
      IF(INCX.EQ.1.AND.INCY.EQ.1) GO TO 20
      IX=1
      IY=1
      IF(INCX.LT.0) IX=(-N+1)*INCX+1
      IF(INCY.LT.0) IY=(-N+1)*INCY+1
      DO 10 I=1,N
      STEMP=STEMP+SX(IX)*SY(IY)
      IX=IX+INCX
      IY=IY+INCY
10     CONTINUE
      SDOT=STEMP
      GO TO 70
20     M=MOD(N,5)
      IF(M.EQ.0) GO TO 40
      DO 30 I=1,M
      STEMP=STEMP+SX(I)*SY(I)
30     CONTINUE
      IF(N.LT.5) GO TO 60
40     MP1=M+1
      DO 50 I=MP1,N,5
      STEMP=STEMP+SX(I)*SY(I)+SX(I+1)*SY(I+1)+SX(I+2)*SY(I+2)+SX(I+3)*SY
1(I+3)+SX(I+4)*SY(I+4)
50     CONTINUE

```

```
60  SDOT=STEMP  
70  CONTINUE  
    RETURN
```

```

C*****
C                               SUBROUTINE SAXPY                               *
C*****
      SUBROUTINE SAXPY(N,SA,SX,INCX,SY,INCY,NEQS)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 SX(NEQS),SY(NEQS),SA
      IF(N.LE.0) RETURN
      IF(SA.EQ.0.0D0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1) GO TO 20
      IX=1
      IY=1
      IF(INCX.LT.0) IX=(-N+1)*INCX+1
      IF(INCY.LT.0) IY=(-N+1)*INCY+1
      DO 10 I=1,N
      SY(IY)=SY(IY)+SA*SX(IX)
      IX=IX+INCX
      IY=IY+INCY
10  CONTINUE
20  M=MOD(N,4)
      IF(M.EQ.0) GO TO 40
      DO 30 I=1,M
      SY(I)=SY(I)+SA*SX(I)
30  CONTINUE
      IF(N.LT.4) RETURN
40  MP1=M+1
      DO 50 I=MP1,N,4
      SY(I)=SY(I)+SA*SX(I)
      SY(I+1)=SY(I+1) + SA*SX(I+1)
      SY(I+2)=SY(I+2) + SA*SX(I+2)
      SY(I+3)=SY(I+3) + SA*SX(I+3)
50  CONTINUE
      RETURN

```

```

C*****
C                               SUBROUTINE SSWAP                               *
C*****
SUBROUTINE SSWAP(N,SX,INCX,SY,INCY,NEQS)
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 SX(NEQS),SY(NEQS)
IF(N.LE.0)RETURN
IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
IX=1
IY=1
IF(INCX.LT.0)IX=(-N+1)*INCX+1
IF(INCY.LT.0)IY=(-N+1)*INCY+1
DO 10 I=1,N
  STEMP=SX(IX)
  SX(IX)=SY(IY)
  SY(IY)=STEMP
  IX=IX+INCX
  IY=IY+INCY
10 CONTINUE
RETURN
20 M=MOD(N,3)
IF(M.EQ.0)GOTO 40
DO 30 I=1,M
  STEMP=SX(I)
  SX(I)=SY(I)
  SY(I)=STEMP
30 CONTINUE
IF(N.LT.3)RETURN
40 MP1=M+1
DO 50 I=MP1,N,3
  STEMP=SX(I)
  SX(I)=SY(I)

```

```
    SY(I)=STEMP  
    STEMP=SX(I+1)  
    SX(I+1)=SY(I+1)  
    SY(I+1)=STEMP  
    STEMP=SX(I+2)  
    SX(I+2)=SY(I+2)  
    SY(I+2)=STEMP  
50  CONTINUE  
    RETURN
```

```

C*****
C                                     FUNCTION ISAMAX                                     *
C*****
      INTEGER FUNCTION ISAMAX(N, SX, INCX, NEQS)
      IMPLICIT REAL*8 (A-H, O-Z)
      REAL*8 SX(NEQS)
      ISAMAX=0
      IF(N.LT.1)RETURN
      ISAMAX=1
      IF(N.EQ.1)RETURN
      IF(INCX.EQ.1)GOTO 20
      IX=1
      SMAX=DABS(SX(1))
      IX=IX+INCX
      DO 10 I=2,N
        IF(DABS(SX(IX)).LE.SMAX)GOTO 5
        ISAMAX=I
        SMAX=DABS(SX(IX))
      5  IX=IX+INCX
     10 CONTINUE
      RETURN
     20 SMAX=DABS(SX(1))
        DO 30 I=2,N
          IF(DABS(SX(I)).LE.SMAX)GOTO 30
          ISAMAX=I
          SMAX=DABS(SX(I))
     30 CONTINUE
      RETURN

```

```

C*****
C*                               SPHERE                               *
C*****
      SUBROUTINE SPHERE
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1      EXP,EXPO,EXPOLD,EXPNEW,BPOINT
      REAL*8 LOAD,LAMDA,LAMDA1
      CHARACTER*2 ITRACE
      COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQ1(70),DISTR(70),
1      LOAD(70),Q(70),R(70),Q1(70),R1(70),KPV(70)
      COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1      ,MCODE(70,6),MINC(70,2)
      COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
      COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1      DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2      IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3      NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4      UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5      EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6      IHELP,BPOINT,LFLAG,ICORD,NUMBER,LDA,ISOLVE
C      FIND THE CONSTANTS FOR THE QUADRATIC EQUATION.
      CALL STORE(LOAD,Q,Q1,-1.DO)
      A1=1.DO+F(DQ1,DQ1,DQ1,0)
      B1=(LAMDA-LAMDA1+F(DQ1,DQ,LOAD,1))*2.DO
      D1=F(DQ,DQ,LOAD,2)
C      FIND THE ROOTS OF THE QUADRATIC EQUATION.
      G2=B1*B1-4.DO*A1*D1
      IF(G2.GT.0.DO) GO TO 20
      WRITE(2,1) G2
1      FORMAT(//,' NEGATIVE VALUE IN SPHERE IS ',G24.16,
1      ' PROGRAM IS STOPPED TO AVOID ERROR.')
```

```

        WRITE(2,1001)
1001  FORMAT('/', ' TRACE OF PROGRAM PATH',/)
        WRITE(2,1002) (ITRACE(I),I=1,K15)
1002  FORMAT(1X,10A2,5X,10A2,5X,10A2,5X,10A2,5X,10A2,5X)
        STOP
20  G2=DSQRT(G2)
     G1=(-B1+G2)/(2.00*A1)
     G2=(-B1-G2)/(2.00*A1)
C   CHOOSE THE ROOT THAT LEADS TO THE SMALLEST INCREMENTAL VECTOR.
     DLAMD=G1
     CALL STORE(L0AD,DQ,DQ1,G2)
     CALL STORE(DQ,DQ,DQ1,G1)
     IF(DSQRT(G1*G1+F(DQ,DQ,DQ,0)).LT.
1    DSQRT(G2*G2+F(L0AD,L0AD,L0AD,0))) GO TO 10
     CALL STORE(DQ,L0AD,L0AD,0.00)
     DLAMD=G2
C   UPDATE THE VARIABLES.
10  LAMDA=LAMDA+DLAMD
     CALL STORE(Q,Q,DQ,1.00)
     RETURN

```

```

C*****
C*                                     STIFF                                     *
C*****
SUBROUTINE STIFF
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1      EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1      LOAD(70),Q(70),R(70),Q1(70),R1(70),KPVT(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1      ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1      DETO,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2      IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3      NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4      UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5      EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6      IHELP,BPOINT,LFLAG,ICCORD,NUMBER,LDA,ISOLVE
  DIMENSION H(15),INDEX(6,6,2)
  DATA INDEX /1,2,4,-1,-2,-4,2,3,5,-2,-3,-5,4,5,6,-4,-5,-6,
1      -1,-2,-4,1,2,4,-2,-3,-5,2,3,5,-4,-5,-6,4,5,6,
2      1,2,3,4,5,-3,2,6,7,8,9,-7,3,7,10,11,12,-10,4,8,
3      11,13,14,-11,5,9,12,14,15,-12,-3,-7,-10,-11,-12,
4      10/
  JFLAG=0
  IF (ISOLVE.EQ.2) IHBW=NEQS
  DO 10 II=1,NEQS
    DO 20 JJ=1,IHBW
      SS(II,JJ)=0.DO

```

```

20 CONTINUE
10 CONTINUE
DO 30 I=1,NM
  AI=A(I)
  EI=E(I)
  XLI=XL(I)
  XLDI=XLD(I)
  CG1I=CG1(I)
  CG2I=CG2(I)
  CG3I=CG3(I)
  ELONG=XLDI-XLI
  H(1)=AI*EI*(CG1I*CG1I+(ELONG/XLDI)*(1.00-CG1I*CG1I))/XLI
  H(2)=AI*EI*CG1I*CG2I*(1.00-ELONG/XLDI)/XLI
  H(3)=AI*EI*(CG2I*CG2I+(ELONG/XLDI)*(1.00-CG2I*CG2I))/XLI
  H(4)=AI*EI*CG1I*CG3I*(1.00-ELONG/XLDI)/XLI
  H(5)=AI*EI*CG2I*CG3I*(1.00-ELONG/XLDI)/XLI
  H(6)=AI*EI*(CG3I*CG3I+(ELONG/XLDI)*(1.00-CG3I*CG3I))/XLI
  IF(ICCORD.EQ.1)GOTO 70
  H1=H(1)
  H2=H(2)
  H3=H(3)
  H4=H(4)
  H5=H(5)
  H6=H(6)
  I1=MINC(I,1)
  I2=MINC(I,2)
  C1I=C1(I1)
  C2I=C2(I1)
  C1HI=C1(I2)
  C2HI=C2(I2)
  H(1)=C1I**2*H1+2.00*C1I*C2I*H2+C2I**2*H3
  H(2)=-1.00*C1I*C2I*H1+(C1I**2-C2I**2)*H2+C1I*C2I*H3

```

```

H(3)=C1I*H4+C2I*H5
H(4)=-1.D0*C1I*C1HI*H1-(C1I*C2HI+C2I*C1HI)*H2-C2I*C2HI*H3
H(5)=C1I*C2HI*H1-(C1I*C1HI-C2I*C2HI)*H2-C2I*C1HI*H3
H(6)=C2I**2*H1-2.D0*C1I*C2I*H2+C1I**2*H3
H(7)=-1.D0*C2I*H4+C1I*H5
H(8)=C2I*C1HI*H1-(C1I*C1HI-C2I*C2HI)*H2-C1I*C2HI*H3
H(9)=-1.D0*C2I*C2HI*H1+(C1I*C2HI+C2I*C1HI)*H2-C1I*C1HI*H3
H(10)=H6
H(11)=-1.D0*C1HI*H4-C2HI*H5
H(12)=C2HI*H4-C1HI*H5
H(13)=C1HI**2*H1+2.D0*C1HI*C2HI*H2+C2HI**2*H3
H(14)=-1.D0*C1HI*C2HI*H1+(C1HI**2-C2HI**2)*H2+C1HI*C2HI*H3
H(15)=C2HI**2*H1-2.D0*C1HI*C2HI*H2+C1HI**2*H3
GOTO(70,170),ISOLVE
70 DO 40 JM=1,6
    J=MCODE(I,JM)
    IF(J.EQ.0) GO TO 40
    DO 50 KM=JM,6
        K=MCODE(I,KM)
        IF(K.EQ.0) GO TO 50
        KB=K-J+1
        IN=INDEX(JM,KM,ICORD)
        IF(IN.LE.0) GO TO 60
        SS(J,KB)=SS(J,KB)+H(IN)
        GO TO 50
60        SS(J,KB)=SS(J,KB)-H(-IN)
50    CONTINUE
40    CONTINUE
    GOTO 30
170 DO 140 JM=1,6
    J=MCODE(I,JM)
    IF(J.EQ.0)GOTO 140

```

```
DO 150 KM=JM,6
K=MCODE(I,KM)
IF(K.EQ.0)GOTO 150
IN=INDEX(JM,KM,ICoord)
IF(IN.LT.0)GOTO 160
SS(J,K)=SS(J,K)+H(IN)
GOTO 150
160 SS(J,K)=SS(J,K)-H(-IN)
150 CONTINUE
140 CONTINUE
30 CONTINUE
RETURN
```

```

C*****
C*                                     STORE                                     *
C*****
SUBROUTINE STORE(X1,Y1,Z1,V)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1     EXP,EXPO,EXPOLD,EXPNEW,BPOINT
REAL*8 LOAD,LAMDA,LAMDA1
CHARACTER*2 ITRACE
COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1     LOAD(70),Q(70),R(70),Q1(70),R1(70),KPV(70)
COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1     ,MCODE(70,6),MINC(70,2)
COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1     DET0,DET,DETOLD,DETNEW,LAMDA,LAMDA1,DSBR,
2     IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3     NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4     UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5     EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6     IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE
DIMENSION X1(NEQS),Y1(NEQS),Z1(NEQS)
STORE Y1 + V * Z1 IN X1.
IF(V.EQ.0.DO) GO TO 20
DO 10 I=1,NEQS
X1(I)=Y1(I)+V*Z1(I)
10 CONTINUE
RETURN
20 DO 30 I=1,NEQS
X1(I)=Y1(I)
30 CONTINUE
RETURN

```

```

C*****
C*                                VECTOR                                *
C*****
SUBROUTINE VECTOR
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER SCALE,SYSTEM,UPDATE,CONT,ENDLIM,ENDSTB,BRANCH,
1      EXP,EXPO,EXPOLD,EXPNEW,BPOINT
  REAL*8 LOAD,LAMDA,LAMDA1
  CHARACTER*2 ITRACE
  COMMON /BLOCK1/ SS(70,70),DQ(70),DQ1(70),DQI(70),DISTR(70),
1      LOAD(70),Q(70),R(70),Q1(70),R1(70),KPV(70)
  COMMON /BLOCK2/ A(70),CG1(70),CG2(70),CG3(70),E(70),XL(70),XLD(70)
1      ,MCODE(70,6),MINC(70,2)
  COMMON /BLOCK3/ X(31),Y(31),Z(31),C1(31),C2(31),JCODE(31,3)
  COMMON /BLOCK4/ D(6),DLAMD,DLAMD1,DS,ELONG,EPSLN1,EPSLN2,EPSLN3,
1      DETO,DET,DETDLD,DETNEW,LAMDA,LAMDA1,DSBR,
2      IHBW,JFLAG,JHBW,L,IFLAG,LHAT,LMAX,ITRACE(2000),
3      NCOUNT,NEQS,NJ,NM,NNEG,NPOINT,SCALE,SYSTEM,K15,
4      UPDATE,CONT,ENDLIM,ENDSTB,INTEST,METHOD,BRANCH,
5      EXP,EXPO,EXPOLD,EXPNEW,IMAX,JMAX,MPOINT,NDIGIT,
6      IHELP,BPOINT,LFLAG,ICOORD,NUMBER,LDA,ISOLVE

C      $$$ R $$$
      K15=K15+1
      ITRACE(K15)='R'
      BRANCH=0
      CALL POWINV

C      FIND THE TANGENT VECTOR TO GET THE NORMAL VECTOR.
      CALL STORE(LOAD,DISTR,LOAD,0.DO)
      IF(ISOLVE.EQ.1)CALL SOLVE1
      IF(ISOLVE.EQ.2)CALL SOLVE2
      IF(NCOUNT.EQ.0) GO TO 70
      DLAMD=DS/DSQRT(F(LOAD,LOAD,LOAD,0)+1.DO)

```

```

      GO TO 80
70 DS=DLAMD*DSQRT(F(LOAD,LOAD,LOAD,0)+1.DO)
80 DO 90 I=1,NEQS
      DQ(I)=LOAD(I)*DLAMD
90 CONTINUE
      LAMDA1=LAMDA
      CALL STORE(Q1,Q,Q,0.DO)
      WRITE(2,32)
32 FORMAT(//,' TANGENT VECTOR',/)
      WRITE(2,2) (DQ(I),I=1,NEQS),DLAMD
      2 FORMAT(5G24.16)
      DLTEMP=DLAMD
      CALL STORE(LOAD,DQ,DQ,0.DO)
C      CALCULATE THE NORMAL VECTOR.
      ALPH=-(F(DQ,DQ,DQ,0)+DLAMD*DLAMD)/F(DQ,R1,R1,0)
      CALL STORE(DQ,DQ,R1,ALPH)
      DS=DSQRT(F(DQ,DQ,DQ,0)+DLAMD*DLAMD)
      G10=DSBR/DS
      DS=DSBR
      DO 65 I=1,NEQS
        DQ(I)=DQ(I)*G10
65 CONTINUE
      DLAMD=DLAMD*G10
C      UPDATE FOR THE NORMAL VECTOR.
      CALL STORE(Q,Q,DQ,1.DO)
      LAMDA=LAMDA+DLAMD
      NPOINT=MPOINT+NCOUNT-1
      WRITE(2,31)
31 FORMAT(//,' NORMAL VECTOR',/)
      WRITE(2,2) (DQ(I),I=1,NEQS),DLAMD
      WRITE(2,49) DS
49 FORMAT(///,' DS =',G24.16)

```

```
C   CALCULATE THE DOT PRODUCT OF THE NORMAL AND TANGENT
C   VECTORS.
      G10=F(Load,DQ,DQ,0)+DLAMD*DLTEMP
      WRITE(2,37) G10
37  FORMAT(///,' DOT PRODUCT =',G24.16,///)
      RETURN
      END
```

**The vita has been removed from  
the scanned document**

LINEAR AND NONLINEAR ANALYSIS OF SPACE TRUSSES  
RELATIVE TO CYLINDRICAL COORDINATES

by

Jeffrey A. Perrier

(ABSTRACT)

The element and system models for linear and nonlinear space trusses were formulated and implemented relative to cylindrical coordinates. Nonlinear equilibrium paths were traced using the modified Riks/Wempner method. Two radially symmetric reticulated domes which exhibit snap-through instability and which were subjected to radially symmetric loads were analyzed both linearly and nonlinearly. Along with the entire structures, "pie slices" of the structures were also analyzed. Accuracy was observed by comparison of element forces or of critical load values to those previously presented for these trusses.

Reduction of computational effort from analysis of "pie slices" was studied, as were suppression and generation of bifurcation buckling modes.

Structural models in cylindrical coordinates were found to greatly reduce computational effort by analysis of "pie slices" of radially symmetric structures with radially symmetric loads, and analysis of "pie slices" provided qualitative information regarding buckling mode geometries when techniques for branching onto bifurcation paths failed.