(2)

SUSTAINABILITY SURVIVABILITY SUPPORTABILITY

TECHNICAL REPORT
NATICK/TR-93/017

AD ___________

A NONLINEAR DYNAMIC SPHERICAL MEMBRANE MODEL

by
Richard John Benney

January 1993

Final Report
August 1991 - July 1992

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

UNITED STATES ARMY NATICK
RESEARCH, DEVELOPMENT AND ENGINEERING CENTER
NATICK, MASSACHUSETTS 01760-5000

AERO-MECHANICAL ENGINEERING DIRECTORATE

DISCLAIMERS

The findings contained in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of trade names in this report does not constitute an official endorsement or approval of the use of such items.

DESTRUCTION NOTICE

For Classified Documents:

Follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX.

For Unclassified/Limited Distribution Documents:

Destroy by any method that prevents disclosure of contents or reconstruction of the document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE January 1993

3. REPORT TYPE AND DATES COVERED Final Report Aug 91 to July 92

4. TITLE AND SUBTITLE

A NONLINEAR DYNAMIC SPHERICAL MEMBRANE MODEL

5. FUNDING NUMBERS

PE 62786D
PR 1L162786D283
TA AJ
WU HOO

6. AUTHOR(S)

RICHARD JOHN BENNEY

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Natick RD&E Center
ATTN: SATNC-UE
Natick, MA 01760-5017

8. PERFORMING ORGANIZATION REPORT NUMBER

NATICK/TR-93/017

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A static set of nonlinear spherical membrane equations are modified to model the dynamic response of a spherical membrane to a time dependent pressure distribution. The two governing partial differential equations (PDE's) were converted into a system of nonlinear first order differential equations which are finite differenced in space and solved numerically. The model is a first step towards modelling the dynamic response of parachutes during the complex opening phase.

The ultimate end use for a numerical structural dynamic canopy model is to couple it with a numerical model of the fluid medium surrounding the canopy. This coupling problem is a concurrent effort at the U.S. Army Natick Research, Development & Engineering Center. The project involves coupling the computational fluid dynamics (CFD) code SALE (Simplified Arbitrary Lagrangian Eulerian) to a modified version of the spherical membrane model Fortran programs described in this report. The spherical membrane model coupled to the CFD code SALE will be a major step towards the solution of the opening problem of parachutes. The solution of the opening problem will provide essential information to aid in the design of high-speed, low-altitude airdrop systems.

14. SUBJECT TERMS COMPUTATIONAL FLUID DYNAMICS(CFD)
PARACHUTE MODELS PARACHUTES VIBRATIONAL ANALYSIS
STRUCTURAL ANALYSIS ANNULAR PARACHUTES

15. NUMBER OF PAGES 122

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

CONTENTS

Figures & Tables v

Preface vii

List of Symbols viii

Introduction 1

Formulation and Analysis 3

Reformulation for SLATEC Software 6

Fortran Program Descriptions 9

MATLAB Program Descriptions 16

Boundary Condition Options and Examples 18

Example # 1 20

Example # 2 26

Example # 3 32

Related Topics 39

Discussion of Model Tests and Limitations 40

Conclusion 42

References 44

Appendix A. Development of Dynamic Membrane Equations . . . 45

Appendix B. Development of Finite Difference Equations . . 47

Appendix C. Development of Symmetry Boundary Condition . . 48

Appendix D. Development of Infinite Mass Boundary Condition 49

Appendix E. Comparison with Static Results Presented By Stoker 51

Appendix F. Static Comparison with NISA Finite Element Model 53

Appendix G. Vent and Annular Canopy Boundary Conditions . . . 57

Appendix H. Static Fortran Programs 60

Appendix I. Dynamic Fortran Programs 74
Appendix J. Static MATLAB Program 99
Appendix K. Dynamic MATLAB Program 106

FIGURES

Figure 1. Spherical Membrane Model 3

Figure 2. Flow Chart Outline for Static and Dynamic Programs 13

Example # 1 = Figures 3-11

Figure 3. Membrane Shape at Five Different Time Steps 21

Figure 4. Tangential Deflections Versus Theta at Five Times . 21

Figure 5. Normal Deflections Versus Theta at Five Times . . . 22

Figure 6. Normal Deflections Versus Time 22

Figure 7. Normal Velocities Versus Time 23

Figure 8. Hoop Strain Versus Theta at Five Times 23

Figure 9. Meridional Strain Versus Theta at Five Times 24

Figure 10. Hoop Stress Versus Theta at Five Times 24

Figure 11. Meridional Stress Versus Theta at Five Times . . . 25

Example # 2 = Figures 12-20

Figure 12. Membrane Shape at Five Different Time Steps 27

Figure 13. Tangential Deflections Versus Theta at Five Times . 27

Figure 14. Normal Deflections Versus Theta at Five Times . . . 28

Figure 15. Normal Deflections Versus Time 28

Figure 16. Normal Velocities Versus Time 29

Figure 17. Hoop Strain Versus Theta at Five Times 29

Figure 18. Meridional Strain Versus Theta at Five Times 30

Figure 19. Hoop Stress Versus Theta at Five Times 30

Figure 20. Meridional Stress Versus Theta at Five Times 31

Example # 3 = Figures 20-31

Figure 21. Static Solutions up to Initial Pressure Distribution 33

Figure 22. Membrane Shape at Five Different Time Steps 33

Figure 23. Tangential Deflections Versus Theta at Five Times . 34
Figure 24. Normal Deflections Versus Theta at Five Times . . . 34
Figure 25. Normal Deflections Versus Time 35
Figure 26. Normal Velocities Versus Time 35
Figure 27. Hoop Strain Versus Theta at Five Times 36
Figure 28. Meridional Strain Versus Theta at Five Times . . . 36
Figure 29. Hoop Stress Versus Theta at Five Times 37
Figure 30. Meridional Stress Versus Theta at Five Times . . . 37
Figure 31. Static Solutions up to Final Pressure Distribution 38
Figure D-1 Infinite Mass Boundary Condition at θ_{max} 49
Figure E-1 Comparison of Figures 2a,b, and c from Stoker Text . 53
Figure F-1 Element & Node Point Layout for NISA Model 55
Figure F-2 Comparison of U Deflection Versus Theta (150) . . . 55
Figure F-3 Comparison of W Deflection Versus Theta (150) . . . 56
Figure F-4 Comparison of U Deflection Versus Theta (660) . . . 56
Figure F-5 Comparison of W Deflection Versus Theta (660) . . . 57
Figure G-1 Membrane Shapes at Different Stepped up Pressures . 59
Figure G-2 Tangential Deflection Versus Theta 59
Figure G-3 Normal Deflections Versus Theta 60

TABLES

Table 1. Constant Input Parameters for Example Problems . . 20
Table E-1. Definition of K in Stoker Text 52

PREFACE

The work described in this report on A Nonlinear Dynamic Spherical Membrane Model was undertaken during the period August 1991 to July 1992. The funding was Program Element 62786D, Project No. 1L162786D283, Task No. AJ, and Work Unit Accession No. HOO. This work was performed by the Engineering Technology Division (ETD) of the Aero-Mechanical Engineering Directorate (AMED).

The author wishes to express his appreciation to Dr. Earl Steeves of ETD for his help in this effort.

LIST OF SYMBOLS

A	Partial Derivative of u wrt θ
B	2nd Partial Derivative of u wrt θ Squared
C	Partial Derivative of w wrt θ
C_u	Constant Tangential Damping Ratio
C_w	Constant Normal Damping Ratio
D	2nd Partial Derivative of w wrt θ Squared
E	Young's Modules
h	Membrane Thickness
M	Number of Nonlinear Algebraic Equations
N	Number of Nonlinear ODE's
nth	Number of Nodes Used to Model Sphere
p	Pressure
R	Membrane Radius
t	Time
u	Tangential Deflection
w	Normal Deflection
z_1	Vector of Tangential Deflections
z_2	Vector of Tangential Velocities
z_3	Vector of Normal Deflections
z_4	Vector of Normal Velocities
Z	Vector (z_1, z_2, z_3, z_4)
ε_ϕ	Hoop Strain
ε_θ	Meridional Strain
ν	Poisson's Ratio
ϕ	Locates a Meridian Plane
ρ	Density of Membrane
σ_ϕ	Hoop Stress
σ_θ	Meridional Stress
θ	Defines Position on Meridional Line
θ_{max}	Defines Bottom Location of Sphere
θ_{min}	Defines Top Location of Sphere

A NONLINEAR DYNAMIC SPHERICAL MEMBRANE MODEL

Introduction

The ultimate end use for a structural dynamic canopy model is to predict the complex opening phase of a parachute. Predicting the opening phase of a parachute must involve the coupling of the structural model representing the canopy (fabric and lines) with the fluid medium surrounding the canopy. The coupling problem is a concurrent effort at the U.S. Army Natick Research, Development & Engineering Center. The effort involves coupling the computational fluid dynamics (CFD) code SALE (Simplified Arbitrary Lagrangian Eulerian) to a modified version of the spherical membrane model described in this report. The spherical membrane model is used in the program as a large subset of subroutines. The spherical membrane programs need the pressure distribution on the sphere surface and a time step as input. The programs return the corresponding deflections and velocities of the membrane as output. The spherical membrane model coupled to the CFD code SALE will be a major step towards the solution of the opening problem of parachutes. This solution is essential in the design of high-speed and low-altitude airdrop systems.

A nonlinear spherical membrane model was developed by Stoker (see reference 6) for the solution of static problems. The model was modified to include inertia terms. In this report, the resulting partial differential equations (PDE's) were converted into a system of nonlinear first order ordinary differential equations (ODE's). These equations are solved numerically. The dynamic response of any portion of a spherical membrane can be determined for a given time-dependent pressure distribution and a set of initial conditions. The model is a first step towards modelling the dynamic response of parachutes during the complex opening phase.

The six governing equations for six unknowns presented by Stoker are converted into two governing PDE's through manipulations and the addition of inertia terms. In the model all of the parameters used to describe the sphere along with the pressure distribution are assumed to act on the undeformed spherical shape. Therefore, the description of these equations can represent small to moderately large displacements. The PDE's are written in terms of the unknown tangential and normal components of deflection and various derivatives of each. The PDE's are second order in time and second order in space. The two PDE's are converted to four PDE's each of which is first order in time and second order in space. These four PDE's are finite differenced in space to yield a nonlinear system of first order ordinary differential equations.

The number of ODE's is dependent on the number of nodes used to represent the spherical membrane. The number of ODE's to be solved is equal to four times the number of nodes used. The resulting ODE's are incorporated into a Fortran program which calls the subroutine "DDEBDF" (see reference 5) (DDEBDF is part of the SLATEC library obtained from the National Energy Software Center). The number of nodes used to represent the canopy is a user-defined parameter.

Two separate main Fortran programs were written, one for static solutions (which yields the membrane deflections for a given time independent pressure load) and one for the dynamic response. The programs take input in dimensional form but solve the nondimensional form of the equations. A variety of different boundary conditions can be chosen. The options include a "pinned-top pinned-bottom", "symmetry-top pinned-bottom", and a "symmetry-top infinite mass bottom". The programs generate output in a matrix format which is readable by MATLAB. The MATLAB software has many capabilities but is only used for postprocessing with this problem. Two MATLAB programs were written which process the results of a run from matrix format into graphical results.

The static solutions presented by Stoker were reproduced. The static Fortran program is solving a system of nonlinear algebraic equations (time independent). The subroutine DNSQE.f (part of the SLATEC library) is used to solve these equations. The "pinned-bottom pinned-top" boundary conditions were also modelled using NISA (see reference 4) for comparison to thin shell finite elements.

A normal and a tangential damping term were added to the dynamic model to check that time independent pressure distributions would damp out to the static solution. A variety of dynamic runs were made using different time steps, different numbers of nodes, different node distributions, etc., to check the results for consistency.

The Fortran programs in a modified form are currently being coupled to the CFD code SALE by Natick personnel. The programs are capable of computing large deflections from the undeformed spherical shape. These large deflections are not accurately modelling a physically real membrane, and the model was not chosen with the intention of doing so. The limitations on accurately modeling large deflections include: 1. the material is considered linearly elastic 2. the loads are applied to the undeformed spherical shape 3. compressive stresses are permitted and 4. all deformations are based on the undeformed geometry. The model can however reproduce these large "numerical" deflections which will enable engineers involved with the coupling problem to investigate the problems associated with large amplitude deflections in rapid motion. This experience is expected to ensure an easier coupling of future parachute structural dynamic models with modified CFD codes.

Formulation and Analysis

The static nonlinear spherical membrane equations developed by Stoker can be written in terms of the tangential and normal deflections. The section of a sphere is represented by a single meridional curve in the global X,Y coordinate system as shown in figure 1.

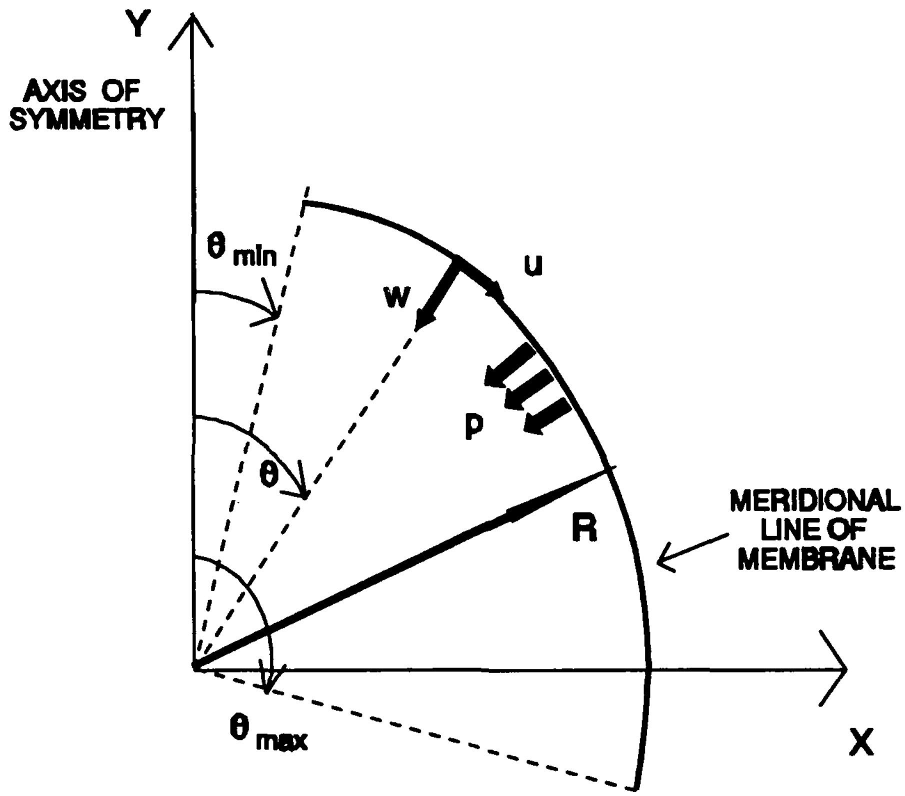

Figure 1. Spherical Membrane Model

A "vent" size can be prescribed by the angle θ_{min} (set to zero for no vent) and the "skirt edge" is defined by the angle θ_{max}. The six governing equations developed by Stoker are presented in Appendix A and are modified by including inertia terms and manipulating the equations into the following two partial differential equations. These equations are merely Newton's second law written in the tangential and normal directions. The resulting PDE's representing dynamic equilibrium in the tangential and normal directions are shown in equation (1) and (2) respectively.

$$
\begin{aligned}
&\frac{\partial^2 u}{\partial\theta^2}-\frac{\partial w}{\partial\theta}+\frac{1}{R}\frac{\partial^2 w}{\partial\theta^2}\frac{\partial w}{\partial\theta}+\frac{u}{R}\frac{\partial^2 w}{\partial\theta^2}+\frac{w}{R}\frac{\partial w}{\partial\theta}+\frac{wu}{R}-\frac{1}{2R^2}\left(\frac{\partial w}{\partial\theta}\right)^3-\\
&\frac{3}{2}\frac{u^2}{R^2}\frac{\partial w}{\partial\theta}-\frac{3}{2}\frac{u}{R^2}\left(\frac{\partial w}{\partial\theta}\right)^2-\frac{u^3}{2R^2}+\frac{\partial u}{\partial\theta}\cot\theta-u\cot^2\theta+\frac{\cot\theta}{2R}\left(\frac{\partial w}{\partial\theta}\right)^2+\\
&\frac{\cot\theta}{R}\frac{\partial w}{\partial\theta}u+\frac{\cot\theta u^2}{2R}+\nu\left[-u-\frac{\partial w}{\partial\theta}-\frac{2u\cot\theta}{R}\frac{\partial w}{\partial\theta}+\frac{w}{R}\frac{\partial w}{\partial\theta}-\right.\\
&\left.\frac{3u^2\cot\theta}{2R}+\frac{uw}{R}-\frac{\cot\theta}{2R}\left(\frac{\partial w}{\partial\theta}\right)^2\right]=C_u\frac{\partial u}{\partial t}+\frac{1-\nu^2}{E}\rho R^2\frac{\partial^2 u}{\partial t^2}
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
&\frac{\cot\theta}{R}\frac{\partial w}{\partial\theta}\frac{\partial u}{\partial\theta}-\frac{w\cot\theta}{R}\frac{\partial w}{\partial\theta}+\frac{\cot\theta}{2R^2}\left(\frac{\partial w}{\partial\theta}\right)^3+\frac{3}{2}\frac{u\cot\theta}{R^2}\left(\frac{\partial w}{\partial\theta}\right)^2+\\
&\frac{3}{2}\frac{u^2\cot\theta}{R^2}\frac{\partial w}{\partial\theta}+\frac{u\cot\theta}{R}\frac{\partial u}{\partial\theta}-\frac{uw\cot\theta}{R}+\frac{\cot\theta}{2R^2}u^3+\frac{1}{R}\frac{\partial^2 u}{\partial\theta^2}\frac{\partial w}{\partial\theta}-\\
&\frac{1}{2R}\left(\frac{\partial w}{\partial\theta}\right)^2+\frac{3}{2R^2}\frac{\partial^2 w}{\partial\theta^2}\left(\frac{\partial w}{\partial\theta}\right)^2+\frac{2u}{R^2}\frac{\partial^2 w}{\partial\theta^2}\frac{\partial w}{\partial\theta}+\frac{3}{2R^2}\left(\frac{\partial w}{\partial\theta}\right)^2\frac{\partial u}{\partial\theta}+\frac{2u}{R^2}\frac{\partial u}{\partial\theta}\frac{\partial w}{\partial\theta}+\\
&\frac{u}{R}\frac{\partial^2 u}{\partial\theta^2}-\frac{u}{R}\frac{\partial w}{\partial\theta}+\frac{3u^2}{2R^2}\frac{\partial^2 w}{\partial\theta^2}+\frac{3u^2}{2R^2}\frac{\partial u}{\partial\theta}+\frac{1}{R}\frac{\partial u}{\partial\theta}\frac{\partial^2 w}{\partial\theta^2}-\frac{w}{R}\frac{\partial^2 w}{\partial\theta^2}+\frac{1}{R}\left(\frac{\partial u}{\partial\theta}\right)^2-\\
&\frac{w}{R}\frac{\partial u}{\partial\theta}+\frac{\partial u}{\partial\theta}+u\cot\theta-2w+\frac{u}{R}\frac{\partial w}{\partial\theta}+\frac{u^2}{2R}+\nu\left[\frac{-3u}{R}\frac{\partial w}{\partial\theta}-\frac{w\cot\theta}{R}\frac{\partial w}{\partial\theta}-\right.\\
&\frac{3u^2}{2R}-\frac{wu\cot\theta}{R}+\frac{\cot\theta}{R}\frac{\partial u}{\partial\theta}\frac{\partial w}{\partial\theta}-\frac{3}{2R}\left(\frac{\partial w}{\partial\theta}\right)^2+\frac{2u\cot\theta}{R}\frac{\partial u}{\partial\theta}+\frac{u\cot\theta}{R}\frac{\partial^2 w}{\partial\theta^2}-\\
&\left.\frac{w}{R}\frac{\partial^2 w}{\partial\theta^2}-\frac{w}{R}\frac{\partial u}{\partial\theta}+\frac{\partial u}{\partial\theta}+u\cot\theta-2w\right]=C_w\frac{\partial w}{\partial t}+\frac{1-\nu^2}{E}\left[\rho R^2\frac{\partial^2 w}{\partial t^2}-\frac{R^2 p}{h}\right]
\end{aligned}
\tag{2}
$$

The static version of these equations is obtained by setting the right hand side of each equation to zero. This is equivalent to setting the velocities and accelerations to zero (no time dependence).

These equations are transformed into a nondimensional version by applying the nondimensional (ND) parameters shown in equation (3). The nondimensional version of the governing equations are shown in equations (4) and (5) (the "ND" subscripts are dropped for convenience).

These governing partial differential equations are second order in time and second order in space. When solved, these equations yield the normal and tangential velocities and displacements as a function of theta and time. If the velocity and accelerations in these equations are set equal to zero, the resulting solution

$$w_{ND}=\frac{w}{R} \qquad u_{ND}=\frac{u}{R}$$

$$\sigma_{xND}=\sigma_x\frac{1-\nu^2}{E}$$

$$p_{ND}=p\frac{R(1-\nu^2)}{Eh} \tag{3}$$

$$t_{ND}=t\left(\frac{1-\nu^2}{E}\rho R^2\right)^{-0.5}$$

$$\frac{\partial^2 u}{\partial\theta^2}-\frac{\partial w}{\partial\theta}+\frac{\partial^2 w}{\partial\theta^2}\frac{\partial w}{\partial\theta}+u\frac{\partial^2 w}{\partial\theta^2}+w\frac{\partial w}{\partial\theta}+wu-\frac{1}{2}\left(\frac{\partial w}{\partial\theta}\right)^3-$$
$$\frac{3}{2}u^2\frac{\partial w}{\partial\theta}-\frac{3}{2}u\left(\frac{\partial w}{\partial\theta}\right)^2-\frac{u^3}{2}+\frac{\partial u}{\partial\theta}\cot\theta-u\cot^2\theta+\frac{\cot\theta}{2}\left(\frac{\partial w}{\partial\theta}\right)^2+$$
$$\cot\theta\frac{\partial w}{\partial\theta}u+\frac{\cot\theta u^2}{2}+\nu\left[-u-\frac{\partial w}{\partial\theta}-2u\cot\theta\frac{\partial w}{\partial\theta}+w\frac{\partial w}{\partial\theta}-\right. \tag{4}$$
$$\left.\frac{3u^2\cot\theta}{2}+uw-\frac{\cot\theta}{2}\left(\frac{\partial w}{\partial\theta}\right)^2\right]=C_u\frac{\partial u}{\partial t}+\frac{\partial^2 u}{\partial t^2}$$

$$\cot\theta\frac{\partial w}{\partial\theta}\frac{\partial u}{\partial\theta}-w\cot\theta\frac{\partial w}{\partial\theta}+\frac{\cot\theta}{2}\left(\frac{\partial w}{\partial\theta}\right)^3+\frac{3}{2}u\cot\theta\left(\frac{\partial w}{\partial\theta}\right)^2+$$
$$\frac{3}{2}u^2\cot\theta\frac{\partial w}{\partial\theta}+u\cot\theta\frac{\partial u}{\partial\theta}-uw\cot\theta+\frac{\cot\theta}{2}u^3+\frac{\partial^2 u}{\partial\theta^2}\frac{\partial w}{\partial\theta}-$$
$$\frac{1}{2}\left(\frac{\partial w}{\partial\theta}\right)^2+\frac{3}{2}\frac{\partial^2 w}{\partial\theta^2}\left(\frac{\partial w}{\partial\theta}\right)^2+2u\frac{\partial^2 w}{\partial\theta^2}\frac{\partial w}{\partial\theta}+\frac{3}{2}\left(\frac{\partial w}{\partial\theta}\right)^2\frac{\partial u}{\partial\theta}+2u\frac{\partial u}{\partial\theta}\frac{\partial w}{\partial\theta}+$$
$$u\frac{\partial^2 u}{\partial\theta^2}-u\frac{\partial w}{\partial\theta}+\frac{3u^2}{2}\frac{\partial^2 w}{\partial\theta^2}+\frac{3u^2}{2}\frac{\partial u}{\partial\theta}+\frac{\partial u}{\partial\theta}\frac{\partial^2 w}{\partial\theta^2}-w\frac{\partial^2 w}{\partial\theta^2}+\left(\frac{\partial u}{\partial\theta}\right)^2- \tag{5}$$
$$w\frac{\partial u}{\partial\theta}+\frac{\partial u}{\partial\theta}+u\cot\theta-2w+u\frac{\partial w}{\partial\theta}+\frac{u^2}{2}+\nu\left[-3u\frac{\partial w}{\partial\theta}-w\cot\theta\frac{\partial w}{\partial\theta}-\right.$$
$$\frac{3u^2}{2}-wu\cot\theta+\cot\theta\frac{\partial u}{\partial\theta}\frac{\partial w}{\partial\theta}-\frac{3}{2}\left(\frac{\partial w}{\partial\theta}\right)^2+2u\cot\theta\frac{\partial u}{\partial\theta}+u\cot\theta\frac{\partial^2 w}{\partial\theta^2}-$$
$$\left.w\frac{\partial^2 w}{\partial\theta^2}-w\frac{\partial u}{\partial\theta}+\frac{\partial u}{\partial\theta}+u\cot\theta-2w\right]=C_w\frac{\partial w}{\partial t}+\frac{\partial^2 w}{\partial t^2}-p(\theta,t)$$

yields the static normal and tangential displacements as a function of theta. A system of subroutines (SLATEC Library, see reference 5) are used to solve these equations numerically at a user defined number of node points on the sphere. The equations are reformulated into a form which is compatible with the SLATEC subroutines. The reformulation and SLATEC program input will be discussed in the next section.

Reformulation for SLATEC Software

The governing equations are first transformed from two PDE's each of which is second order in time and space to four PDE's each of which is first order in time and second order in space. A new set of variables are defined in equation (6) to reformulate the equations. The partial derivatives are redefined to simplify the equations as shown in equation (7). The four governing ODE's in terms of these new variables are shown in equations (8) and (9).

$$
\begin{aligned}
z_1 &= u \qquad & z_3 &= w \\
z_2 &= \frac{\partial u}{\partial t} \qquad & z_4 &= \frac{\partial w}{\partial t}
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
A &= \frac{\partial u}{\partial \theta} = \frac{\partial z_1}{\partial \theta} \qquad & B &= \frac{\partial^2 u}{\partial \theta^2} = \frac{\partial^2 z_1}{\partial \theta^2} \\
C &= \frac{\partial w}{\partial \theta} = \frac{\partial z_3}{\partial \theta} \qquad & D &= \frac{\partial^2 w}{\partial \theta^2} = \frac{\partial^2 z_3}{\partial \theta^2}
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
&\frac{\partial z_1}{\partial t} = z_2 \\
&\frac{\partial z_2}{\partial t} = -[B - C + DC + z_1 D + z_3 C + z_3 z_1 - \frac{1}{2}(C)^3 - \\
&\frac{3}{2} z_1^2 C - \frac{3}{2} z_1 (C)^2 - \frac{z_1^3}{2} + A\cot\theta - z_1 \cot^2\theta + \frac{\cot\theta}{2}(C)^2 + \\
&\cot\theta C z_1 + \frac{\cot\theta z_1^2}{2} + \nu[-z_1 - C - 2 z_1 \cot\theta C + z_3 C - \\
&\frac{3 z_1^2 \cot\theta}{2} + z_1 z_3 - \frac{\cot\theta}{2}(C)^2] - C_u z_2]
\end{aligned}
\tag{8}
$$

Next, the spherical membrane is represented by a set of node points located along a meridional curve of the membrane in the global XY plane. The node points are numbered from node number one at theta minimum to node number "nth" at theta maximum. The value of "nth" is equal to the total number of nodes used to represent the membrane. The value of derivatives with respect to theta in the

$$\frac{\partial z_3}{\partial t}=z_4$$

$$\frac{\partial z_4}{\partial t}=-[\cot\theta CA-w\cot\theta C+\frac{\cot\theta}{2}(C)^3+\frac{3}{2}z_1\cot\theta(C)^2+$$
$$\frac{3}{2}z_1^2\cot\theta C+z_1\cot\theta A-z_1z_3\cot\theta+\frac{\cot\theta}{2}z_1^3+BC-$$
$$\frac{1}{2}(C)^2+\frac{3}{2}D(C)^2+2z_1DC+\frac{3}{2}(C)^2A+2z_1AC+ \qquad (9)$$
$$z_1B-z_1C+\frac{3z_1^2}{2}D+\frac{3z_1^2}{2}A+AD-z_3D+(A)^2-$$
$$z_3A+A+z_1\cot\theta-2z_3+z_1C+\frac{z_1^2}{2}+\nu[-3z_1C-z_3\cot\theta C-$$
$$\frac{3z_1^2}{2}-z_3z_1\cot\theta+\cot\theta AC-\frac{3}{2}(C)^2+2z_1\cot\theta A+z_1\cot\theta D-$$
$$z_3D-z_3A+A+z_1\cot\theta-2z_3]-C_v z_4+p(\theta,t)]$$

governing equations are approximated with second order accurate finite difference equations (FDE's) in terms of the deflections at, and spacing between the node points. Four FDE's are used extensively in the analysis with a variable grid spacing. The FDE's are derived in Appendix B by starting with a second order Lagrange Polynomial. The second order Lagrange Polynomial fits a second order polynomial through three consecutive node points. The variable grid option is used to allow for grid clustering near boundaries where rapid changes in the dependent variable occur and to give more flexibility to potential users involved in coupling these equations to grids generated for and used with the CFD program SALE. The grid clustering options are discussed in more detail in the "Fortran Program Descriptions" section of this report.

Dynamic case:

The FDE's when applied to the derivatives with respect to theta in equations (8) and (9) convert the four partial differential equations into a system of "N" nonlinear first order ordinary differential equations (ODE's). These ODE's are nonlinear in space and first order in time. The value of N is dependent on the number of nodes used to represent the sphere and is given by N=4*nth where "nth" is defined as the total number of node points. The value of nth is user defined; however, changing its value requires the program to be recompiled. The variables z_1, z_2, z_3, and z_4 are now vectors of length nth. Each element of the vector z_i (i=1-4) has a unique value at each node point on the spherical membrane at each time step. These values are the solution of the governing system of ODE's. This resulting system of equations can be written in an acceptable form for the SLATEC subroutine DDEBDF.f. The subroutine DDEBDF.f uses the backwards differentiation formulas of orders one through five to integrate a system of first order ordinary differential equations. The equations must be written in the format shown in equation (10). DDEBDF.f requires a separate subroutine be written which defines the differential equations. A set of initial conditions must also be specified.

$$\frac{DZ}{Dt}=DF(t,Z) \qquad (10)$$

$$where \quad Z^T=(z_1,z_2,z_3,z_4)$$

Static Case:

The static equilibrium equations are obtained by setting the time derivative terms in equations (8) and (9) to zero (note that z_2 and z_4 are identically zero). Applying the FDE's to the derivatives yields a system of "M" nonlinear algebraic equations. The value of M is equal to 2*nth where nth is the user defined total number of nodes representing the canopy. The resulting system of nonlinear algebraic equations can be written in an acceptable form for the SLATEC subroutine DNSQE.f. The purpose of subroutine DNSQE.f is to find a zero of a system of M nonlinear functions in M variables by a modification of the Powell hybrid method. The right hand side of the governing system of equations can be defined as a new function for which DNSQE.f attempts to minimize. The governing equations must be written in a separate subroutine. An initial estimate of the deflections must also be supplied.

The solution of either the dynamic or static equations also depends on the boundary conditions applied at the ends of the membrane. A variety of boundary condition options were programmed. Each boundary condition option will be discussed in the "Boundary Condition Options" section of this report.

<u>Fortran Program Descriptions</u>

The static and dynamic solutions to the spherical membrane problem are determined numerically with two separate Fortran programs. The programs are capable of solving the spherical membrane problem for a wide range of input parameters and forcing functions. The programs described in this section were written to test a wide variety of problems so that the version used for the "coupled problem" would be both error free and have tested multiple user options. Whenever possible, the common variables used in each program have been assigned the same name. The ultimate end use of these programs is to couple the dynamic program to the CFD code SALE and use the static program to generate initial conditions for the deflections. The static program is also used to check the results of a dynamic run that is damped and has reached equilibrium. This section gives a brief overview of the program features followed by a flow chart which outlines the contents of each program. The programs are located in Appendixes H and I.

The Fortran programs have a variety of boundary condition options. The static problem requires two boundary conditions at each boundary node. These conditions are obtained from Stokers derivation which uses the principle of minimum potential energy to derive the static equations of equilibrium. One set of conditions for the static case is to specify both a tangential and normal deflection at both boundary node points. The dynamic program requires conditions on velocities and displacements at the end point nodes. The dynamic program also requires initial conditions for both tangential and normal deflections and velocities at every node point. The boundary condition options available in the programs are: 1. "pinned-bottom pinned-top": This option restricts all deflection of the node points located at theta minimum and theta maximum. The tangential and normal deflections at the boundary node points are set to zero and remain zero for all pressure loads and/or time. 2. "pinned-bottom symmetry-top": The value of theta minimum is set to zero degrees for this option. The boundary conditions at the peak of the sphere are symmetric, which requires the tangential displacement to be zero and the slope of the meridional curve to remain zero. The mathematically equivalent statements involving neighboring node points are derived in Appendix C. The displacement of the node point at theta maximum is restricted to zero. 3. "linear solution": This option is simply the linear version of the equations solved for a constant applied pressure. The static solution was shown by Stoker to be [$w(\theta)=(1-v)R^2p/(2Eh)$]. The dynamic program solves the dynamic version which is equivalent to a simple mass-spring-damper system. All of the normal deflections are equal at any given time. 4. "infinite mass bottom symmetry-top": The value of theta minimum is set to zero for this option and the symmetry boundary conditions are used (see appendix C). The boundary node at theta maximum is restricted to motion along a circular path in the global "X,Y" coordinate system. The radius of the circle is defined as the distance from the undeformed bottom edge of the canopy at theta maximum to the point of intersection with the global "Y" axis along a line that is tangent to the bottom edge of the undeformed canopy. This condition requires the value of theta maximum to be greater than 90 degrees. This model is a closer representation of a parachute. The lines can be visualized as a conical cone that is fixed at the apex (intersection with the "Y" axis) and connected to the bottom edge node at theta maximum on the spherical membrane. The development of this boundary condition is presented in Appendix D.

Both a variably spaced grid option and a user-defined number of node points option are incorporated into each program. The user defined number of nodes "nth" is defined in a "parameter" statement in both the main programs and subroutines. Therefore the programs must be recompiled to change this option. The programs have been run with as few as eleven node points and as many as one hundred node points to represent the sphere. The variably spaced grid option allows the user to define the node point locations on the undeformed sphere with unequal spacing. This option will allow the user of the coupled codes to use CFD grid generators and/or the ability to cluster nodes in areas of interest. Three clustering options (see reference 3) were used to check this capability. The first option clusters nodes at theta minimum, the second clusters nodes at theta maximum, and the third option clusters nodes at both theta minimum and theta maximum. The degree of clustering for each option is controlled by the input constant "beta". The degree of clustering increases as beta approaches a value of one from above. The clustering options were found to have a negligible effect on the solution of any given problem provided the degree of clustering is reasonable (the program will not converge properly if for example 90 percent of the nodes are located along a 10 percent length of the membrane).

The dimensions and material property of the spherical membrane are also user defined and input as dimensional quantities. Any section of a sphere can be modelled. The user must define the value of theta minimum in degrees, which must be greater than or equal to zero degrees. The value of theta maximum must be less than 180 degrees and greater than theta minimum. Test runs with theta minimum equal to zero and theta maximum equal to five degrees, and theta minimum equal to zero degrees and theta maximum equal to 179 degrees have converged. The user must also define a constant membrane thickness, a constant value for Young's Modules of the membrane. Also, for the dynamic program values of the material density, tangential damping constant and normal damping constant must be given. A wide variety of these properties were tested. Note that the program solves the nondimensional version of the governing equations so that certain lumped nondimensional parameters are influencing the solution (see nondimensional parameters equation # 3).

The pressure distribution on the canopy as a function of theta was assumed for test runs of the model. Three options are included for testing the model. The first option is a linear pressure distribution which generates a linearly changing pressure along the canopy surface from pressure values supplied at theta minimum and theta maximum. The second option is a parabolic pressure distribution. This option requires as input the pressures at theta minimum, theta maximum, and the pressure at a user-defined "interior" value of theta. The option generates a second order polynomial fit to these three data points. The second order polynomial is then used to calculate the values of the pressures at any other value of theta. The third option is a user-defined pressure distribution with which the user manually assigns values of the pressure at each node point.

The static program uses the prescribed pressure distribution in one of two ways. The static program can start with a value of zero pressure on the sphere and linearly increase the pressure at each node point to the previously defined final value in a user-defined number of steps (the steps are necessary to guarantee convergence). The static program can also start from a previously calculated solution and the previously defined final pressure distribution and calculate solutions for stepped-up values of the pressures of equal magnitude for each node point for a user-defined number of steps. The dynamic program uses the pressure distribution as a function of theta in one of four ways. 1. The pressure distribution can be considered independent of time and therefore act as a step load. 2. The pressure distribution can start at zero and linearly increase to the final user-defined pressure distribution over a user-defined quantity of time. The pressure remains constant (with time) after this time has been reached. 3. The pressures at the starting time are the user-defined pressure distribution. The pressures at each node are linearly decreased to a state of zero pressure and remain at zero after a user defined time. 4. This option is the same as the second option with one addition. The user defines a constant pressure. The user also defines a time at which all node point pressures will jump to the constant pressure for all future times. These options allowed for a wide variety of pressure distributions as a function of theta and pressure fluctuations with respect to time.

The Fortran programs both use the SLATEC library of subroutines to solve the governing systems of equations. The static program calls the subroutine DNSQE.f, which solves a system of nonlinear algebraic equations. The dynamic program calls the subroutine DDEBDF.f, which solves a system of nonlinear first-order differential equations.

The Fortran programs discussed above are run interactively. They produce a variety of output to the screen, which indicates progress and intermediate results. The static program also writes to files that are readable for new static runs and for initial conditions on deflections readable by the dynamic program. The programs also write to files that are set up in a MATLAB readable matrix format. These output files are easily read by the MATLAB software. Two separate MATLAB programs, one for static output and one for dynamic output, were written and are usable for postprocessing. These programs and MATLAB's capabilities will be discussed in the next section.

Dynamic Fortran Program Flow Chart Outline

The flow chart outline for both the static and dynamic Fortran programs is shown in Figure 2. A description of each letter's role in the flow chart for the dynamic program is given below. The dynamic Fortran programs are located in Appendix J.

A. Input dimensional parameters: radius of sphere, Poisson's ratio, Young's modules, theta minimum, theta maximum, membrane thickness, density of membrane material, tangential damping ratio, normal damping ratio, and the type of boundary conditions. The boundary condition options include: 1. pinned-bottom pinned-top 2. pinned-bottom symmetry-top 3. linear solution (normal deflections only) and 4. infinite mass bottom symmetry-top, etc.

B. Input the type of clustering used to define the node point positions on the undeformed sphere. The options are uniformally spaced grid, node point clustering near theta minimum, node point clustering near theta maximum, or node point clustering at both theta minimum and theta maximum. The clustering options all depend on the user-defined parameter beta which controls the degree of clustering.

C. Input the pressure distribution on the sphere as a function of theta. The options available are: 1. linear pressure distribution as a function of theta. 2. parabolic pressure distribution defined by the two end node pressures and a user-defined interior value of theta and corresponding pressure. 3. Individual entry of each nodal pressure value.

D. Input time information including starting time, time step (value of time in seconds ahead of starting time at which a solution is requested), finishing time.

E. Input pressure versus time option. The options available are: 1. Start with all pressures equal to zero and increase them linearly to the described pressure versus theta function at a user-defined time. Also, keep the pressures constant for all future times. 2. Start with the defined pressure versus theta function and linearly decrease the pressures to zero at a user defined time and keep the pressures zero for all future times. 3. Same as option 1. but also define a constant pressure versus theta value to which all pressures "jump" at a user-defined time.

F. Input initial conditions for displacements and velocities in the

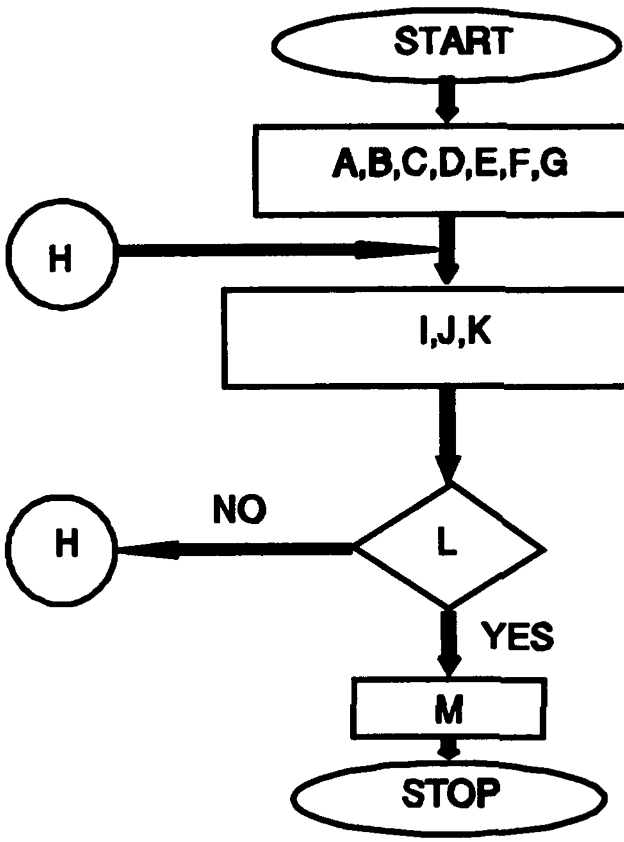

Figure 2. Flow Chart Outline for Static & Dynamic Programs

normal and tangential directions at every node point. The options are: 1. Set all displacements and velocities equal to zero (starting from an undeformed geometry at rest). 2. Read in a set of displacements and velocities either from a previous run or from output produced from the static Fortran program.

G. Input the value of three node points for which the accelerations as a function of time will be saved. Input the number of time steps to save, and the total number of loops for the run. Note: either the final time or the total number of loops will terminate the program, whichever comes first. Also the "bb" matrix for MATLAB postprocessing is written to the output file (Note: the MATLAB matrices and program are described later in this report).

H. Enter the main DO LOOP.

I. Call the SLATEC program DDEBDF.f linked to the correct set of equations which are based on the prescribed boundary conditions. The subroutine DDEBDF.f needs the following input, which is automatically determined by the Fortran program for each loop: 1. A subroutine name where the governing ordinary differential equations are located (The ODE's and subroutine must be written in a predescribed format). 2. Current deflections and velocities at each node point. 3. Current time and desired output time.

J. Extract the desired output if requested for this time step. This includes the current global node point location, current displacements and velocities at every node point, current pressure values at each node point, hoop and meridional strain at each node point and the hoop and meridional stress at each node point. Update maximum and minimum values. Write this information to matrix "bb" for postprocessing and write state of progress and current time to screen.

K. Update the pressure distributions for the next call (if the pressure distribution is a function of time). Update all the parameters that have changed for the next call.

L. If current time is greater than or equal to the "final time" or if the total number of passes through the loop has been reached, then go to M, otherwise go to H.

M. Write final values of deflections, pressures, strains and stresses to the screen. Write the MATLAB matrix "aa" and terminate execution.

Static Fortran Program Flow Chart Outline

The flow chart outline for the static program is shown in Figure 2. A description of each letter's role in the flow chart for the static program is given below. The static Fortran programs are located in Appendix I.

A. Input dimensional parameters: radius of sphere, Poisson's ratio, Young's modules, theta minimum, theta maximum, membrane thickness, and the type of boundary conditions. The boundary condition options include: 1. pinned-bottom pinned-top 2. pinned-bottom symmetry-top 3. infinite mass bottom symmetry-top.

B. Input the type of clustering used to define the node point positions on the undeformed sphere. The options are uniformally spaced grid, clustering near theta minimum, node point clustering near theta maximum, or node point clustering at both theta minimum and theta maximum. The clustering options are all dependent on a user-defined parameter that controls the degree of clustering.

C. Input the pressure distribution on the sphere as a function of theta. The options available are: 1. linear pressure distribution as a function of theta. 2. parabolic pressure distribution defined by the two end node pressures and a user-defined interior value of theta and corresponding pressure. 3. Individual entry of each nodal pressure value.

D,E,F. Input an initial guess for displacements in the normal and tangential directions at every node point. The options are: 1. Set all displacements equal to zero (starting from an undeformed geometry), 2. Read in a set of displacements from a previous run.

G. Input the value of the pressure step. The pressure step is used to increase or decrease the value of the pressures after each call so that solutions for a variety of stepped up pressure functions can be found with one execution of the program. Input the number of solutions to be saved for postprocessing and the total number of passes through the solution loop. Also the "ff" matrix for MATLAB postprocessing is opened (Note: the MATLAB matrices and program are described in the next section of the report).

H. Enter the main DO LOOP.

I. Call the SLATEC program DNSQE.f linked to the correct set of equations which are based on the prescribed boundary conditions. The subroutine DNSQE.f needs the following input which is automatically determined by the Fortran program for each loop: 1. A subroutine name where the governing system of nonlinear algebraic equations are located (The equations and subroutine must be written in a predescribed format). 2. Current "guess" deflections at each node point. 3. Current pressure distribution at each node point.

J. Extract the desired output if requested for this pressure function. This includes the current global node point location, current pressure values at each node point, hoop and meridional strain at each node point and the hoop and meridional stress at each node point. The static program also calculates the total vertical force at each node point. This calculation is preformed in two ways for comparison, first by integrating the pressure distribution over the undeformed sphere, and second by computing the vertical component of force based on the computed meridional stresses at each node point.

K. Update the pressure distributions for the next call. Update all the parameters that have changed for the next call.

L. If the total number of passes through the loop has been reached, then go to M. else go to H.

M. Write final values of deflections, pressures, strains and stresses to the screen. Write the MATLAB matrix "hh" and terminate executions.

MATLAB Program Descriptions

The Fortran programs used to solve the static and dynamic spherical membrane problems generate a large quantity of numerical data. The data generated from a run must be analyzed in a logical and efficient manner. The software package MATLAB was chosen for postprocessing the results from both Fortran programs. The dynamic program results presented the largest challenge because the data are saved for a large quantity of time steps. The user must be able to extract information of interest including deformed shape, strains, and stresses all as a function of time. This requires the ability of graphic animation to present the motion of the membrane as a function of time.

This animation capability was generated in the dynamic MATLAB program. MATLAB is capable of plotting a curve on to a fixed coordinate system. The data plotted can be read from any portion of a preloaded matrix. The curve can be "erased" by replotting it with the "invisible" option in MATLAB. Therefore, to create animation, a curve is plotted then erased for one time step, then plotted and erased for the next time step, etc. The inclusion of a "pause" statement before the erasing of each curve allows the user to stop the animation at any time step. The MATLAB software is run from the STARDENT mini supercomputer. The animation showing the results from a dynamic program run appears as uninterrupted motion to the human eye for runs with less than 30 node points. A list of the output saved by the dynamic and static program in MATLAB matrix format and the capability of the two MATLAB programs is discussed below. It should be noted that the MATLAB programs are a valuable tool in the debugging of the Fortran programs. The MATLAB programs will also serve as the postprocesser of choice for future parachute structural models including the dynamic aspects of the canopy in the coupled problem. A brief outline of the plotting sequence from the dynamic and static MATLAB program is given below.

Dynamic MATLAB Program:

The first graph is a listing of various input parameters of the run. Next, the overall global shape of the canopy (as a two-dimensional meridional line figure) is plotted and the user can view the deformed shape as a function of time. The tangential and normal deflections as a function of theta can also be viewed in animation at each time step. The MATLAB program then allows the user to plot the tangential, normal or resulting displacements, velocities and pressures at any user-defined node point as a function of time. The next section of the program plots hoop and meridional strains and stresses as a function of theta in animation with respect to time. The program also plots the pressure distribution as a function of theta in animation with respect to time. The program sets appropriate axes based on minimum and maximum values that are tracked during execution of the dynamic Fortran program. The dynamic MATLAB program is located in Appendix

K.

<u>Static MATLAB Program:</u>
The first graph is a listing of various input parameters of the run. Next, the overall global shape of the canopy (as a two-dimensional meridional line figure) is plotted and the user can view the deformed shape as a function of different statically applied pressure distributions. The tangential and normal deflections as a function of theta can also be viewed at different pressure distributions. The user can also plot the hoop and meridional strains and stresses in the membrane as a function of theta at different statically applied pressure distributions. The program sets appropriate axes based on minimum and maximum values that are tracked during execution of the static Fortran program. The static MATLAB program is located in Appendix J.

Boundary Condition Options and Examples

This section is a brief discussion of different boundary conditions available with both Fortran programs. An example of output from each boundary condition is presented in the next three sections.

"Pinned Top" - "Pinned Bottom":
This boundary condition restricts the movement of the node points at theta minimum and theta maximum on the canopy to zero deflection. Numerically this is imposed in the dynamic program by setting the velocities and accelerations of the two end point nodes to zero. For the static program the deflections at the two nodes are restricted to zero deflection. The canopy is pinned at these points for all time. The spherical membrane model does not include bending, so large gradients can occur at these pinned points. Clustering node points near a pinned boundary provides better resolution.

"Symmetry Top" - "Pinned Bottom":
This boundary condition restricts the movement of the node point at theta maximum to zero deflection and allows for a symmetry condition at theta minimum (note that theta minimum must be specified as zero degrees for this boundary condition). Numerically the theta maximum boundary condition is imposed in the dynamic program by setting the velocities and accelerations at node point number "nth" to zero and in the static program by setting the deflection at theta maximum equal to zero.

For the static case at theta minimum equal to zero degrees the symmetry boundary is obtained by setting the tangential deflection (which is equivalent to the global "X" deflection) of node point number one equal to zero. The normal deflection condition at node point number one is replaced with a zero slope equation. This condition yields a relationship between node points number one and two through an FDE representing the slope at the peak node and set equal to zero.

For the dynamic program we need conditions on velocities and accelerations at theta equal to zero degrees. The tangential velocity and accelerations at theta minimum must be zero to keep the node point on the global "Y" axis and preserve symmetry. The condition on the normal velocity and the normal acceleration are simply the governing differential equations modified by imposing the zero slope condition. These equations must include the fact that the tangential deflection at node number one is equal to zero. The process becomes difficult due to the singularity at theta equal to zero degrees in the governing equations. The singularity is removed by applying L'Hopitals rule to each term in the form of zero over zero. The derivation is shown in Appendix C.

<u>"Symmetry Top" - "Infinite Mass Bottom":</u>
This boundary condition must have theta minimum equal to zero. The symmetry top boundary condition described in the last paragraph is used at node number one. The boundary node at theta maximum is restricted to motion along a circular path in the global "X,Y" coordinate system. The radius of the circle is defined as the distance from the undeformed bottom edge of the canopy at theta maximum to the point of intersection with the global "Y" axis along a line that is tangent to the bottom edge of the undeformed canopy. This condition requires the value of theta maximum to be greater than 90 degrees. The lines can be visualized as a conical cone that is fixed at the apex (intersection with the "Y" axis) and connected to node number "nth" at theta maximum on the spherical membrane. The radius of the circle (line length) is a constant defined by the undeformed geometry. The canopy is required to remain tangent to the lines for all time. The development of this boundary condition is presented in Appendix D.

Example 1

Table one contains the dimensional input parameters that are kept constant for the three examples that follow.

TABLE 1 Constant Input Parameters for Example Problems

Input Parameter Description	Input Value & Dimensions
R = Radius of Sphere	300 inches
h = Thickness of Membrane	0.005 inches
E = Young's Modules	30000. psi
ν = Poisson's Ratio	0.3
ρ = Density of Membrane Material	9.0E-06 $lbs*sec^2/inches^4$
C_u=C_w= Damping Constants	0.3

The dynamic program was run with the following input. The number of node points used to represent the canopy is 30, theta minimum is 40 degrees, theta maximum is 140 degrees, clustering at both ends with beta equal to 1.2. The pressure distribution applied is a step pressure at each node point that is constant with time. The pressure varies parabolically with theta. The pressure distribution is symmetric about theta equal to 90 degrees. The pressure has a value of 0.0 psi at theta minimum and theta maximum. The pressure reaches a maximum value of -0.2 psi at theta equal to 90 degrees. The run started at time equal to 0.0 seconds and ran through 50000 time steps of 0.000001 seconds each. The program saved data at 100 time steps and finished at time equal to 0.05 seconds. The value of 0.05 seconds corresponds to a nondimensional time of 10.01. The displacements and velocities were set to zero at every node point for initial conditions at time equal to zero. Therefore, the membrane starts from rest in an undeformed state. Figure 3 shows the undeformed shape and initial conditions shape superimposed. Four nondimensional deformed shapes are shown at the first four saved nondimensional time steps corresponding to iteration numbers 500, 1000, 1500, and 2000. The nondimensional tangential and normal deflections at each of these times are shown in Figures 4 and 5, respectively. The symmetry of the problem is clearly evident.

A plot of the nondimensional normal deflections and nondimensional normal velocities for three separate node points as a function of time is shown in Figures 6 and 7, respectively. The three tracked nodes are node points 5, 10, and 15, which correspond to theta values of 48.05, 64.88, and 88.28 degrees, respectively. The amplitudes are slowly damping out due to the nonzero damping ratios. The hoop and meridional strains and stresses as a function of theta for the first four nondimensional time steps are shown in Figures 8 to 11.

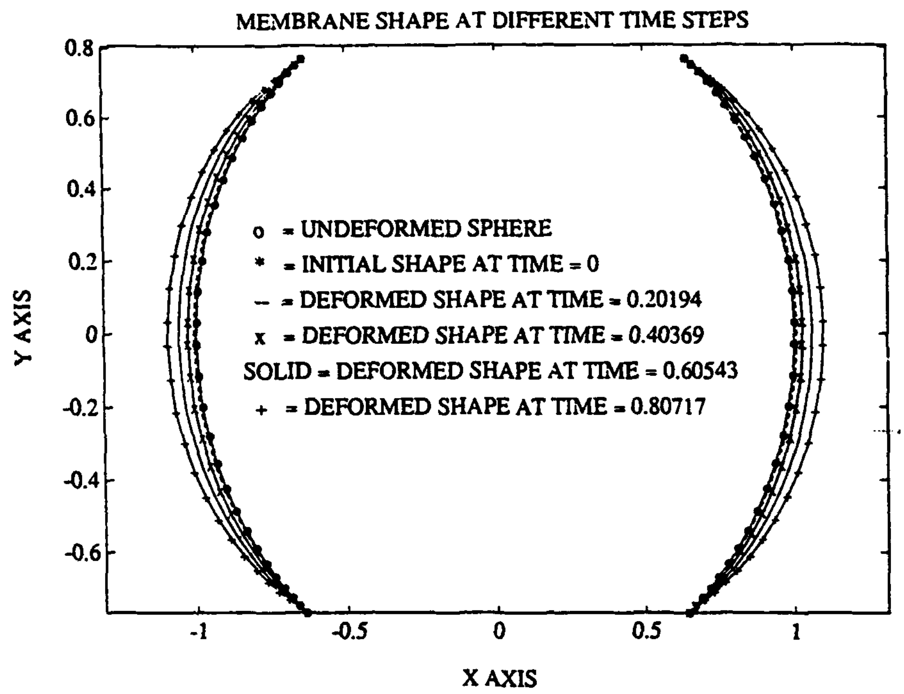

Figure 3. Membrane Shape at Five Different Time Steps

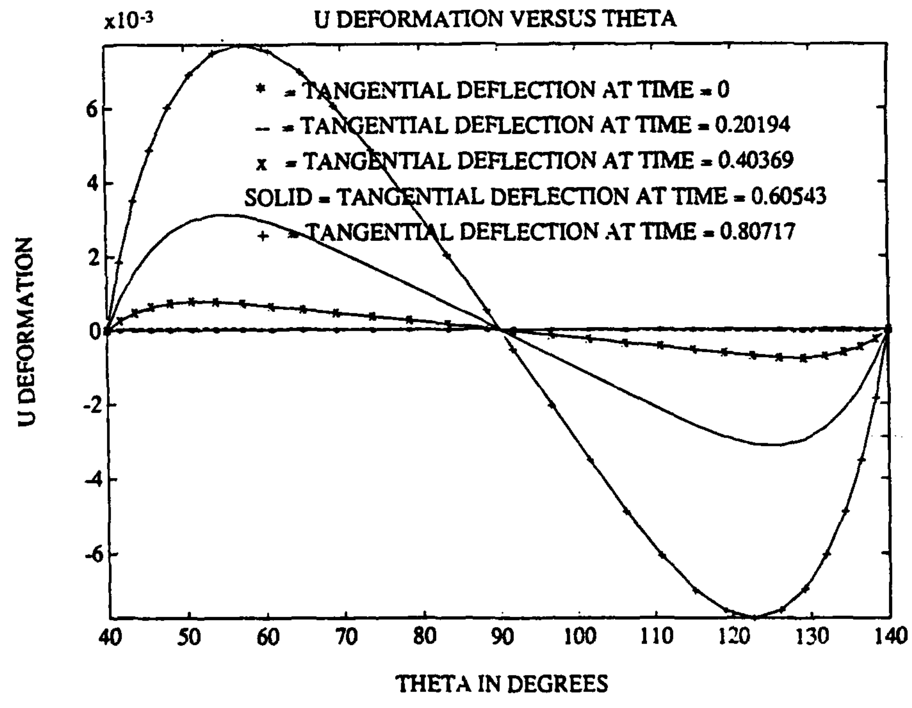

Figure 4. Tangential Deflections Versus Theta at Five Times

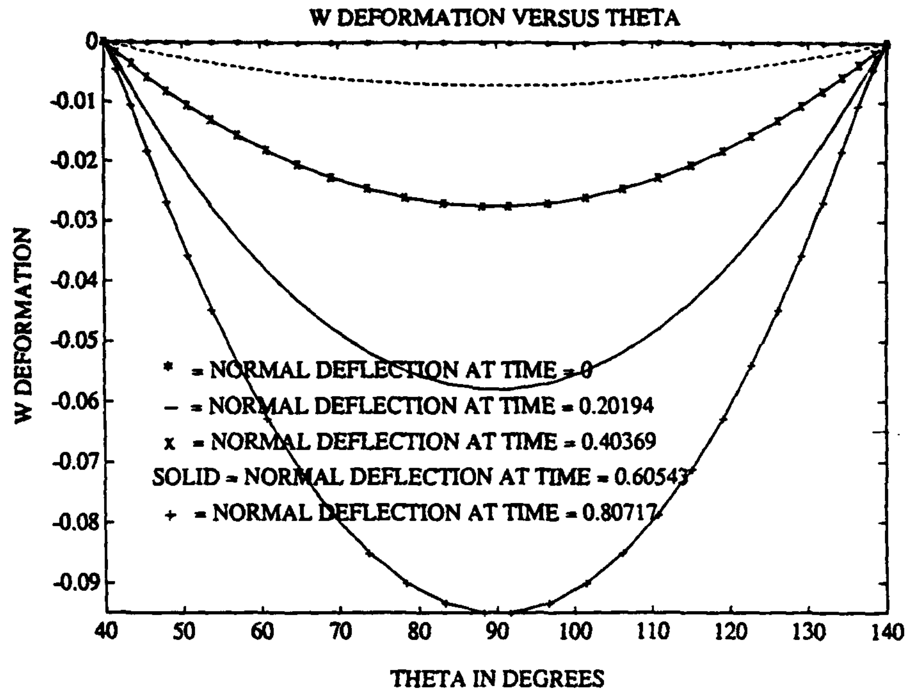

Figure 5. Normal Deflections Versus Theta at Five Times

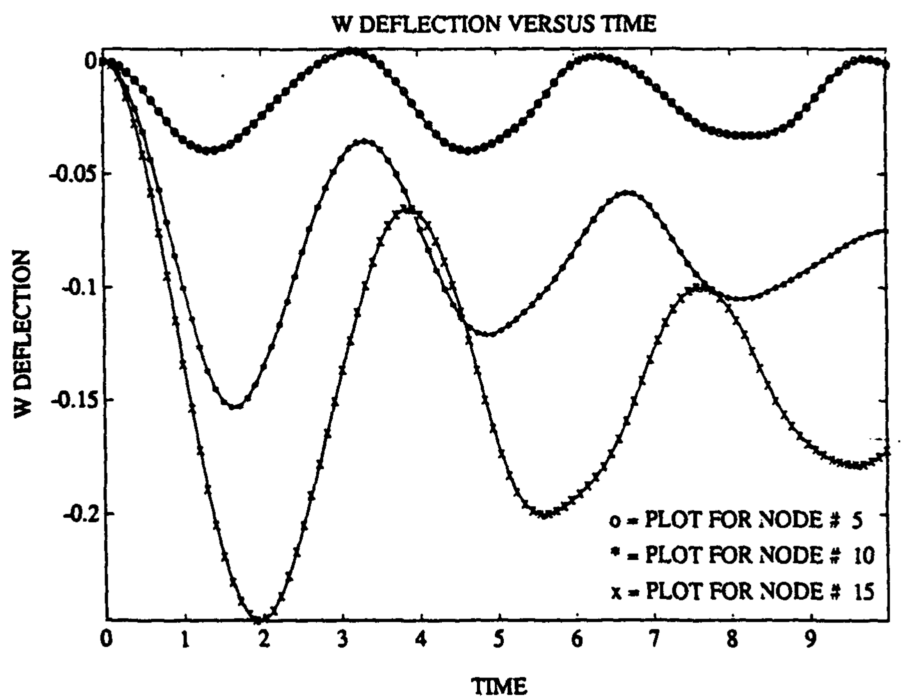

Figure 6. Normal Deflections Versus Time

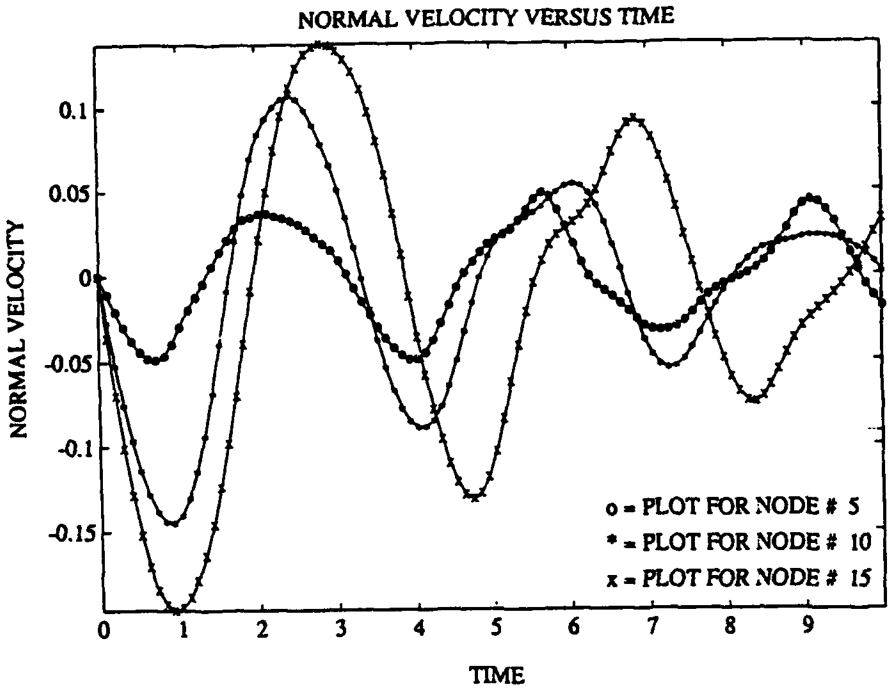

Figure 7. Normal Velocities Versus Time

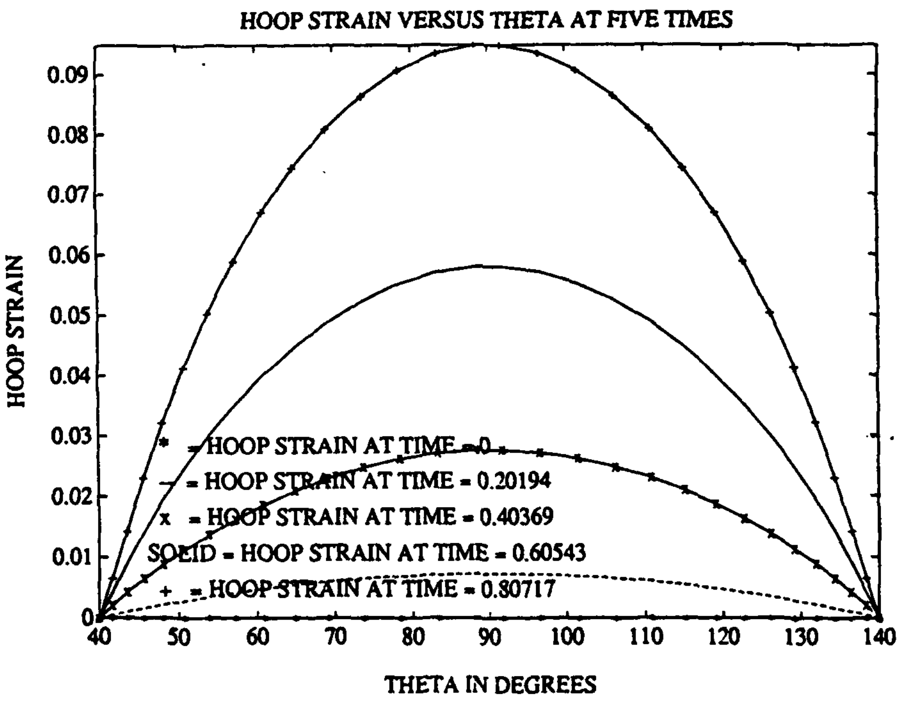

Figure 8. Hoop Strain Versus Theta at Five Times

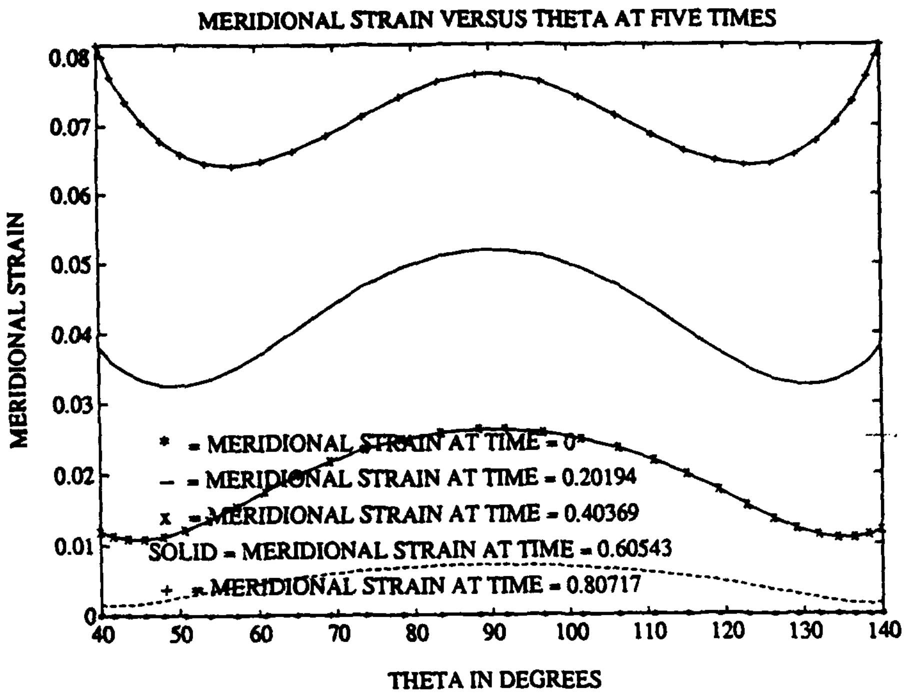

Figure 9. Meridional Strain Versus Theta at Five Times

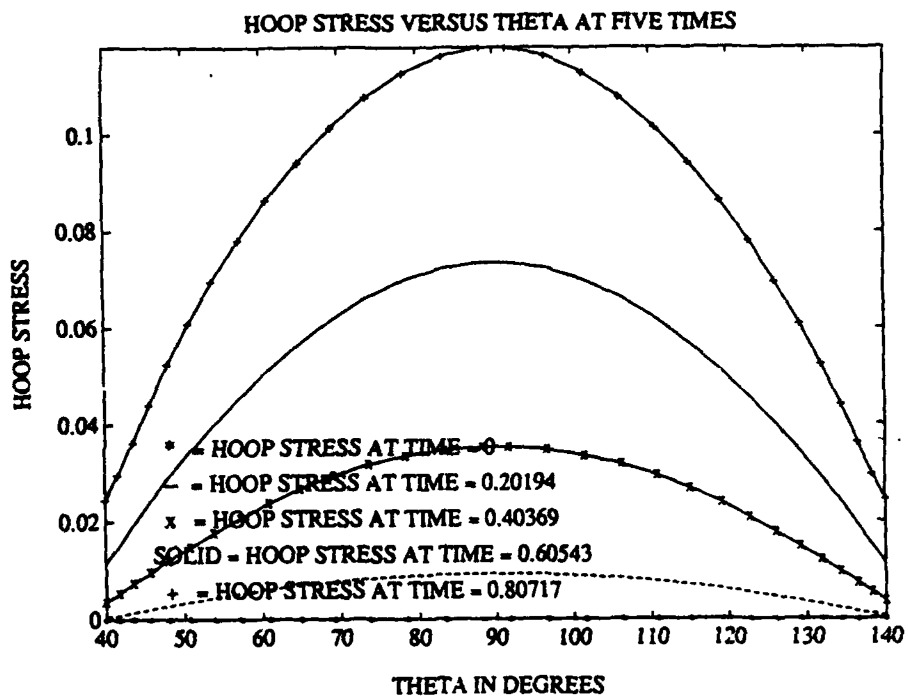

Figure 10. Hoop Stress Versus Theta at Five Times

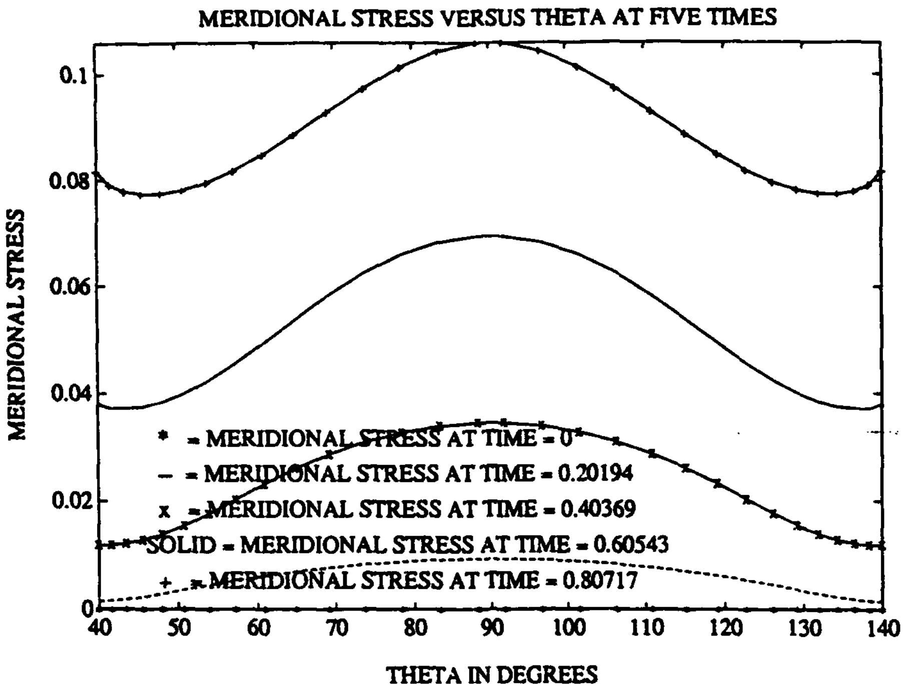

Figure 11. Meridional Stress Versus Theta at Five Times

Example 2

The dynamic program with the "symmetry top"-"pinned bottom" was run with the following input. The number of node points used to represent the canopy is 20, theta minimum is 0 degrees, theta maximum is 110 degrees, clustering at theta maximum was used with beta equal to 1.2. The pressure distribution applied is a step pressure at each node point that is constant with time. The pressure varies parabolically with theta. The pressure has a value of -0.2 psi at theta minimum, -0.1 psi at theta equal to 70 degrees and 0.0 psi at theta maximum. The run started at time equal to 0.0 seconds and ran through 100,000 time steps of 0.000001 seconds each. The program saved data at 100 time steps and finished at time equal to 0.1 seconds. The value of 0.1 seconds corresponds to a nondimensional time of 20.02. The displacements and velocities were set to zero at every node point for initial conditions. Therefore the membrane starts from rest in an undeformed state. Figure 12 shows the undeformed shape and initial conditions shape superimposed. Four nondimensional deformed shapes are shown at the first four saved nondimensional time steps corresponding to iteration numbers 1000, 2000, 3000, and 4000. The nondimensional tangential and normal deflections at each of these times are shown in Figures 13 and 14, respectively. A plot of the nondimensional normal deflections and nondimensional normal velocities for three separate node points as a function of time is shown in Figures 15 and 16, respectively. The three tracked nodes are node points 5, 10, and 15 which correspond to theta values of 32.63, 67.83, and 93.47 degrees, respectively. The amplitudes are slowly damping out due to the nonzero damping ratios. The hoop and meridional strains and stresses as a function of theta for the first four nondimensional time steps are shown in Figures 17 to 20.

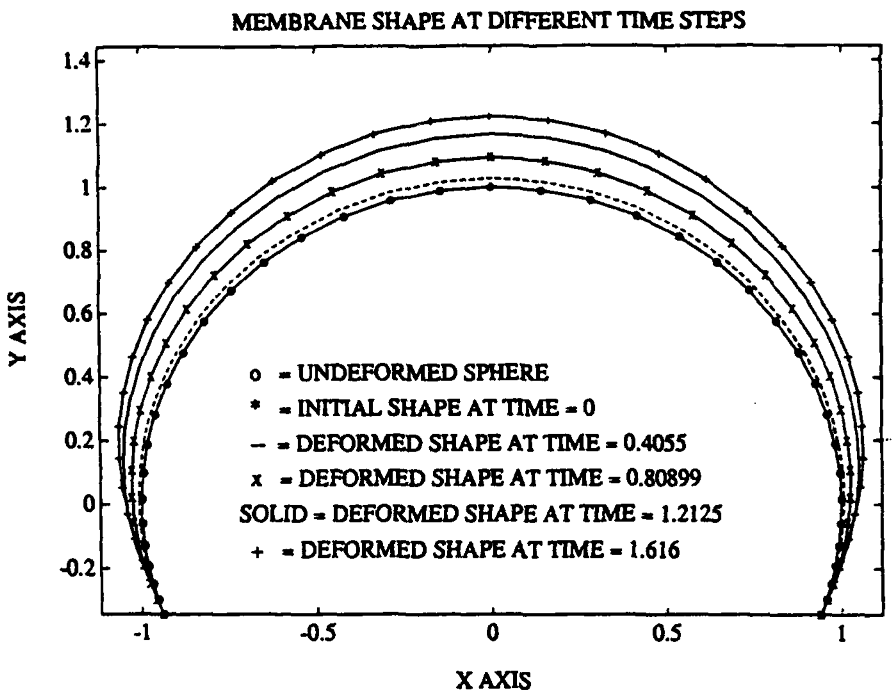

Figure 12. Membrane Shape at Five Different Time Steps

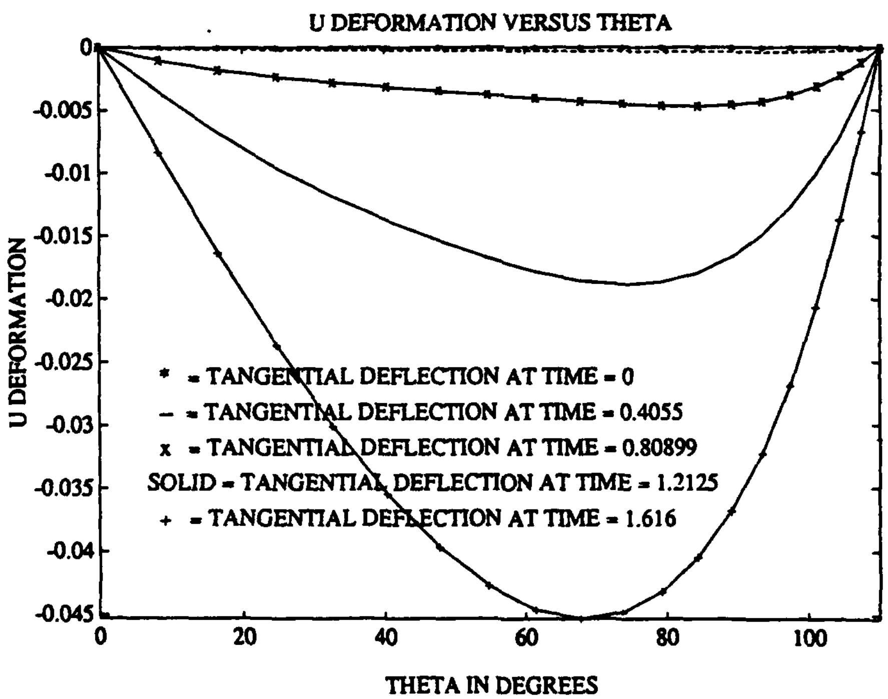

Figure 13. Tangential Deflections Versus Theta at Five Times

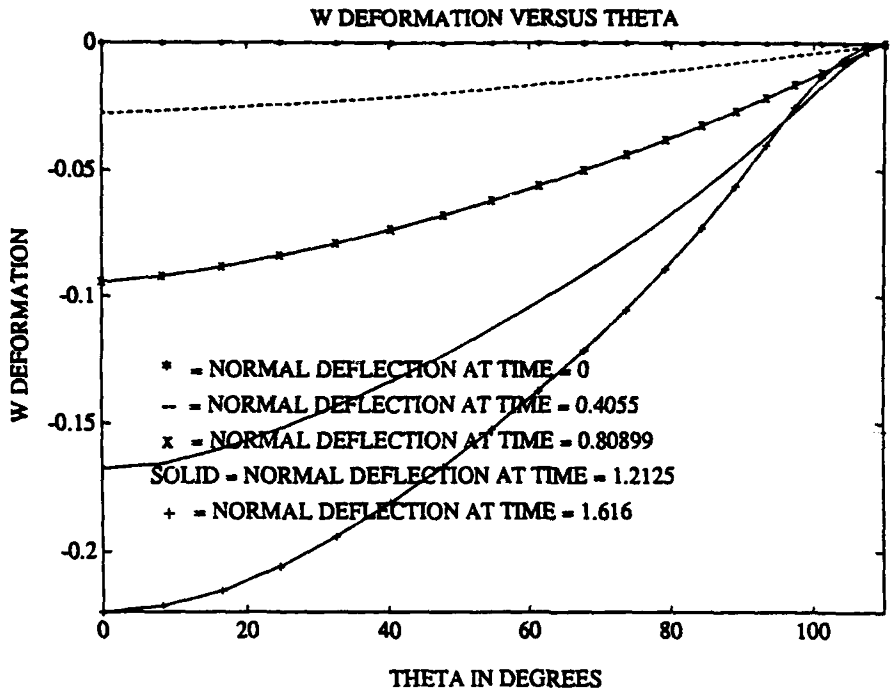

Figure 14. Normal Deflec⁺ions Versus Theta at Five Times

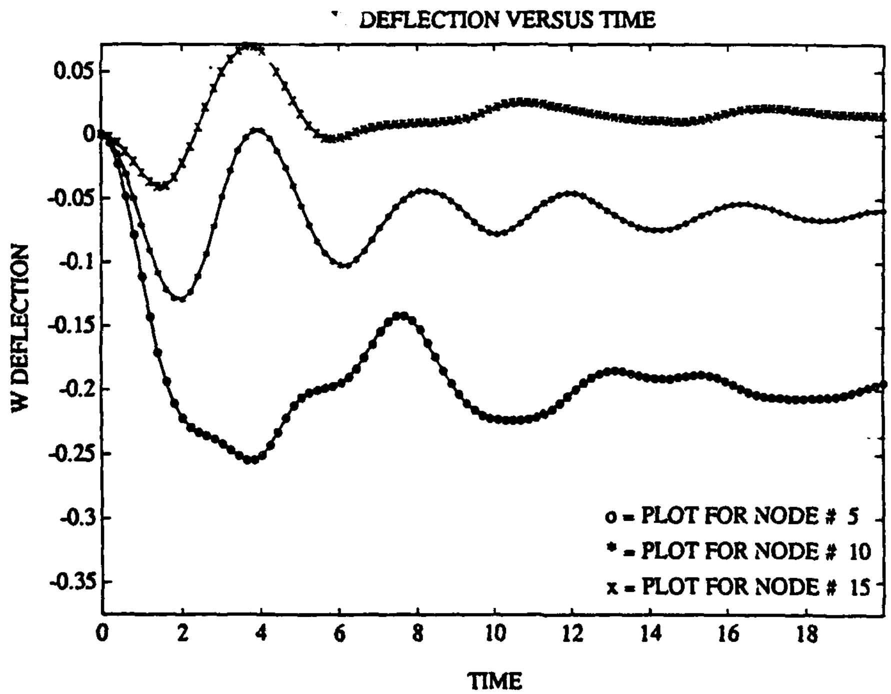

Figure 15. Normal Deflections Versus Time

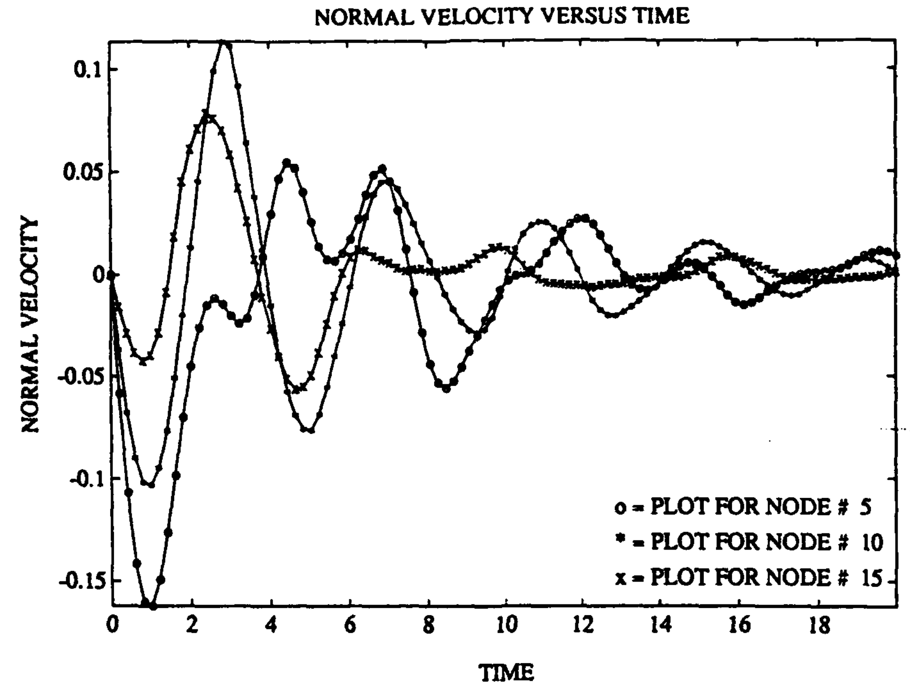

Figure 16. Normal Velocities Versus Time

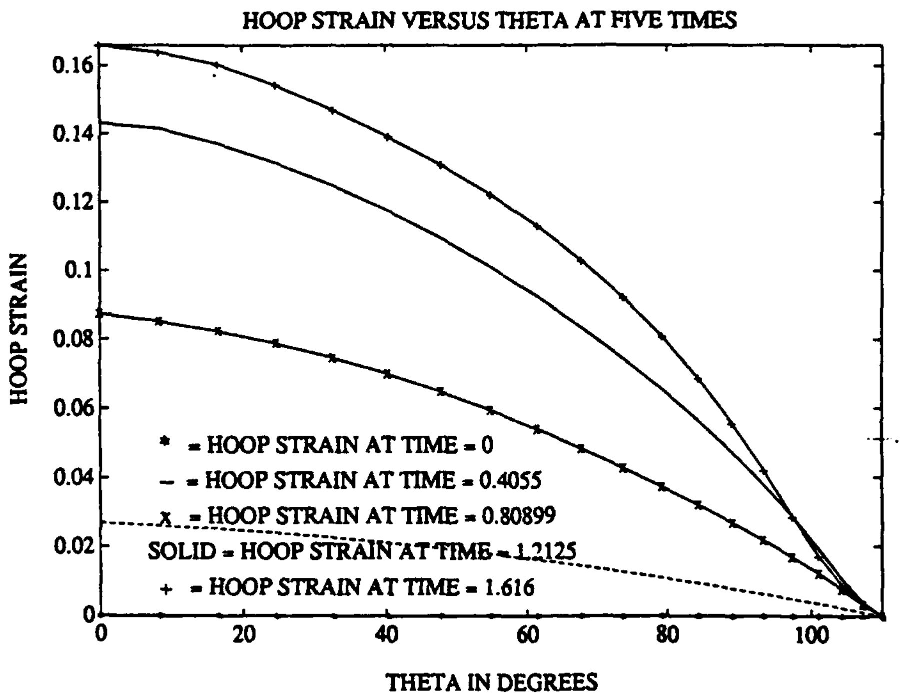

Figure 17. Hoop Strain Versus Theta at Five Times

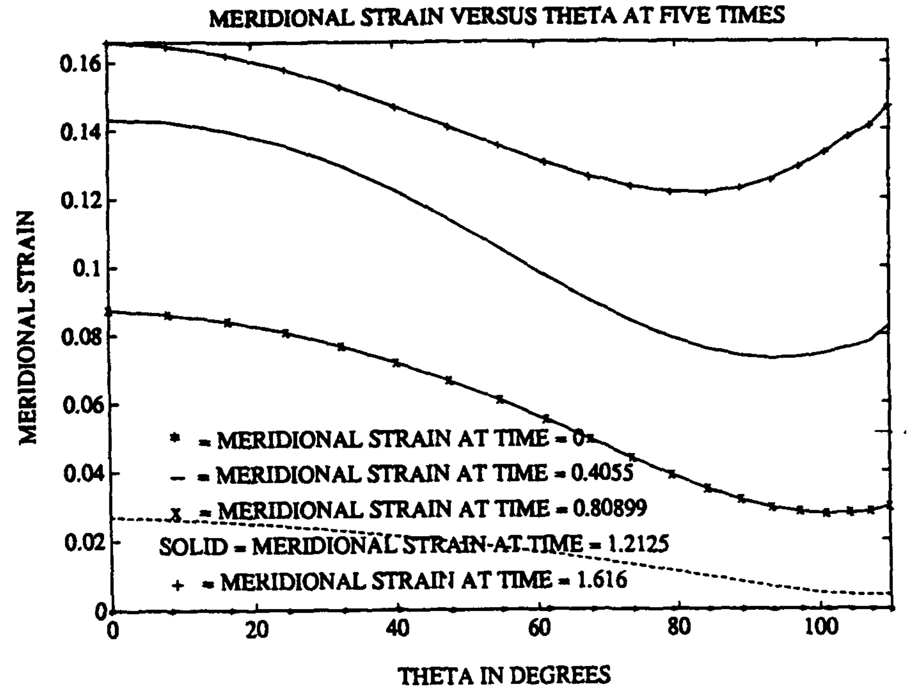

Figure 18. Meridional Strain Versus Theta at Five Times

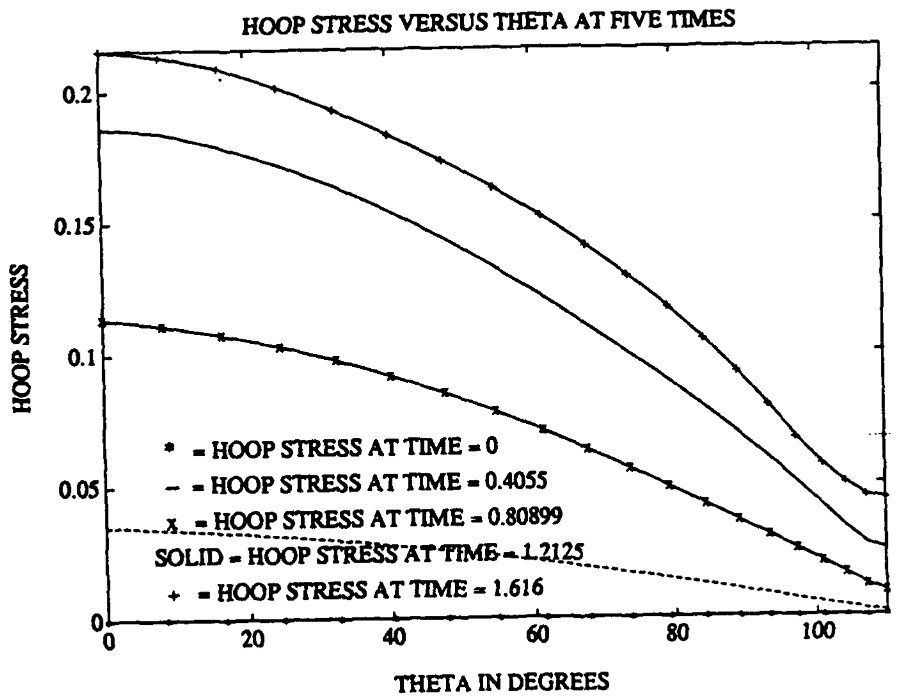

Figure 19. Hoop Stress Versus Theta at Five Times

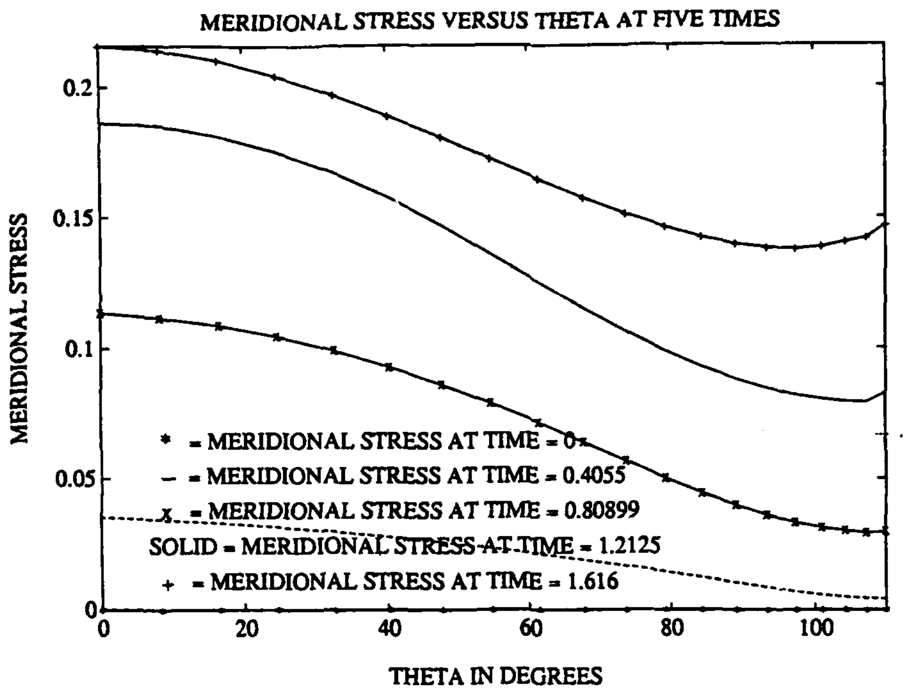

Figure 20. Meridional Stress Versus Theta at Five Times

Example 3

The dynamic program was run with the following input. The number of node points used to represent the canopy is 20, theta minimum is 0 degrees, theta maximum is 110 degrees, clustering at theta maximum with beta equal to 1.1 was used. The pressure distribution applied is a step pressure at each node point that is constant with time. The pressure varies linearly with theta. The pressure has a value of -0.6 psi at theta minimum and -0.3 psi at theta maximum. The run started at time equal to 0.0 seconds and ran through 150,000 time steps of 0.000001 seconds each. The program saved data at 100 time steps and finished at time equal to 0.15 seconds. The value of 0.15 seconds corresponds to a nondimensional time of 30.03. The velocities were set to zero at every node point for initial conditions. The initial conditions on displacement at every node point were generated by the static program. The pressure distribution used to generate the initial shape was also linear. The pressure varied from -0.4 psi at theta minimum to 0.1 psi at theta maximum.

Figure 21 shows the deformed shapes generated by the static program at eight intermediate pressure steps with the pressure distribution used to generate the initial conditions. Therefore the initial conditions are that the membrane starts from rest in a prescribed deformed state. Figure 22 shows the undeformed shape and initial conditions shape as separate curves along with four nondimensional deformed shapes at the first four saved nondimensional time steps corresponding to iteration numbers 1500, 3000, 4500, and 6000.

The nondimensional tangential and normal deflections at each of these times are shown in Figures 23 and 24, respectively. Plots of the nondimensional normal deflections and nondimensional normal velocities for three separate node points are shown as a function of time in Figures 25 and 26, respectively. The three tracked nodes are node points 5, 10, and 15, which correspond to theta values of 37.5, 74.73, and 97.79 degrees, respectively. The amplitudes are slowly damping out due to the nonzero damping ratios.

The hoop and meridional strains and stresses for four nondimensional time steps are shown as a function of theta in Figures 27 to 30. Figure 31 shows the deformed shape of the membrane at different stepped up pressures. The final shape corresponds to the pressure distribution used in the dynamic run.

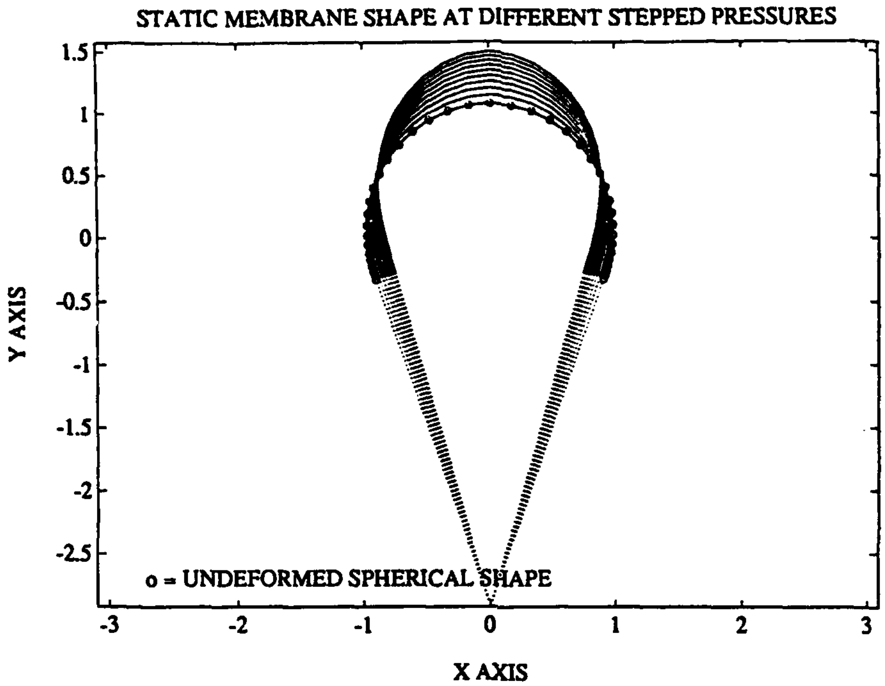

Figure 21. Static Solutions up to Initial Pressure Distribution

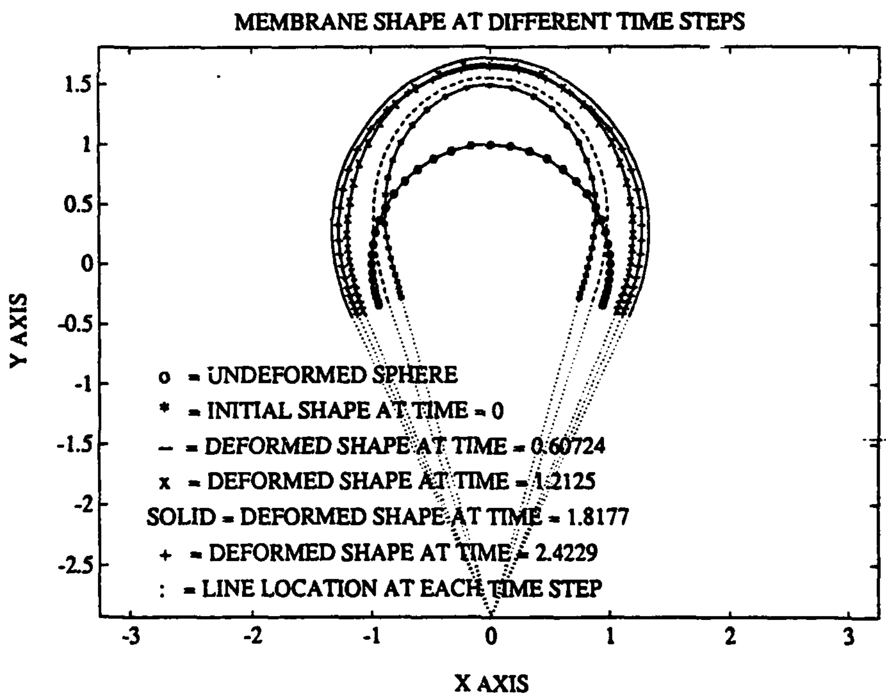

Figure 22. Membrane Shape at Five Different Time Steps

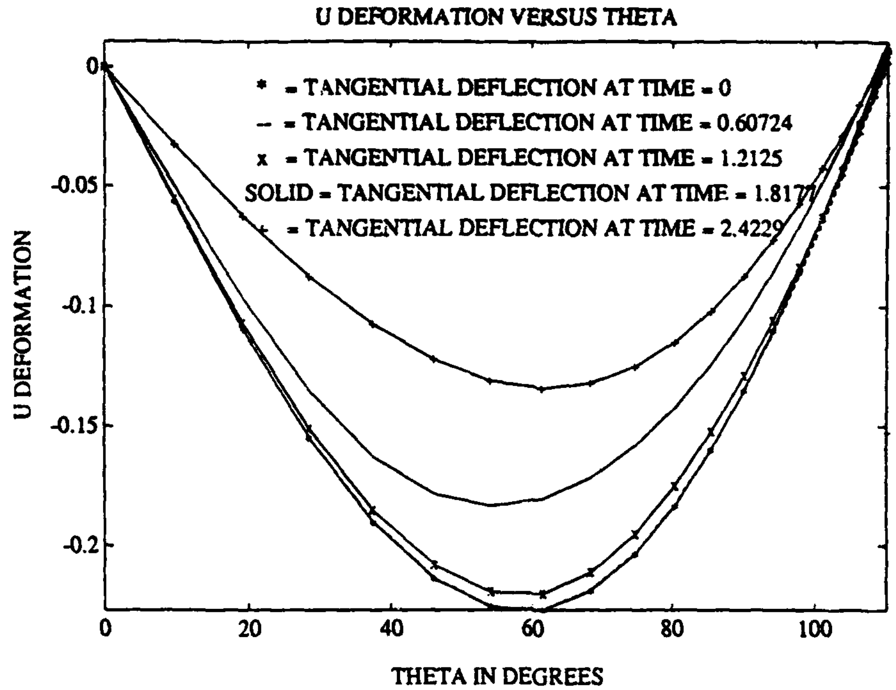

Figure 23. Tangential Deflections Versus Theta at Five Times

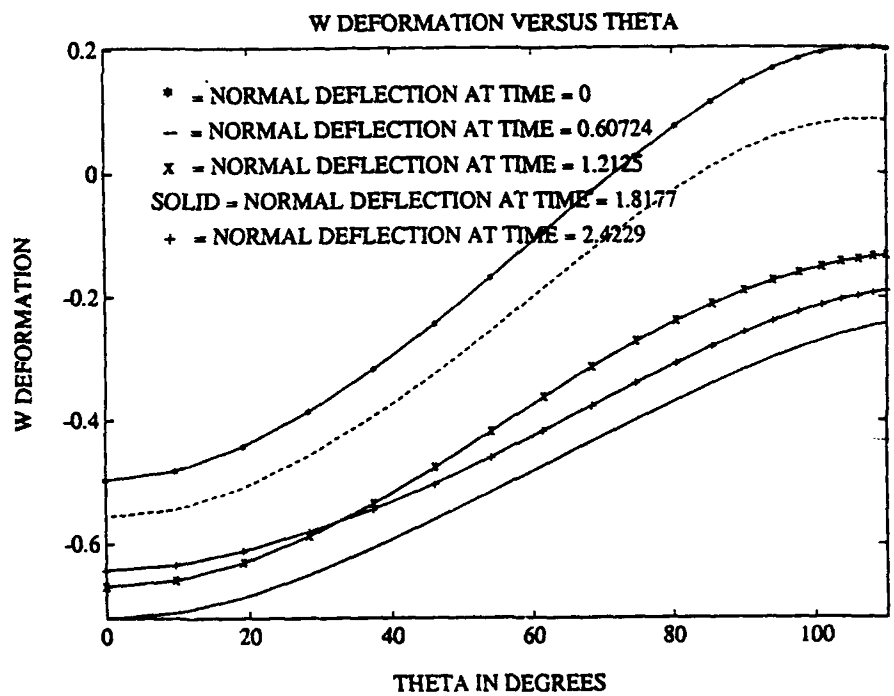

Figure 24. Normal Deflections Versus Theta at Five Times

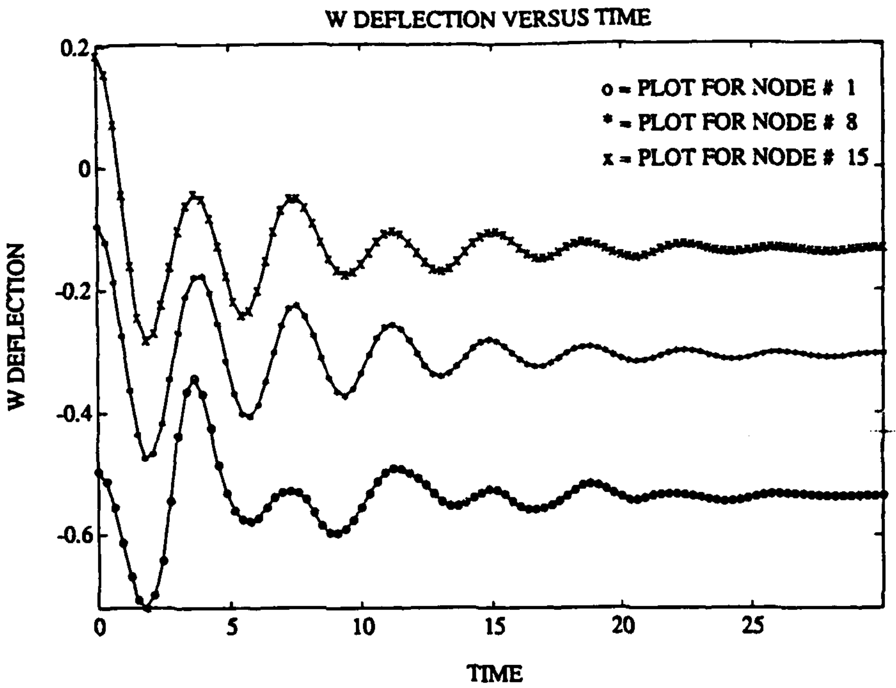

Figure 25. Normal Deflections Versus Time

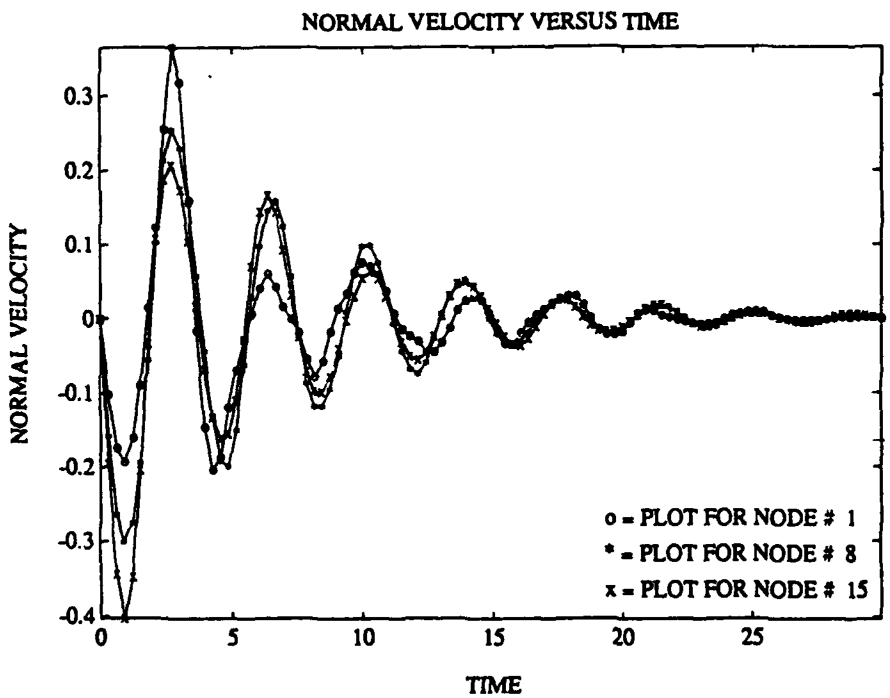

Figure 26. Normal Velocities Versus Time

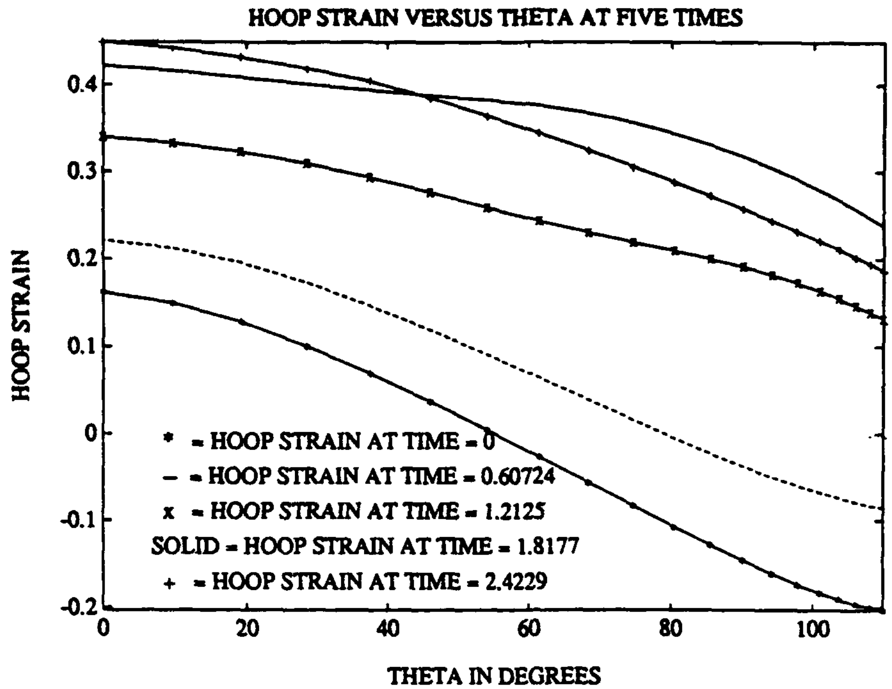

Figure 27. Hoop Strain Versus Theta at Five Times

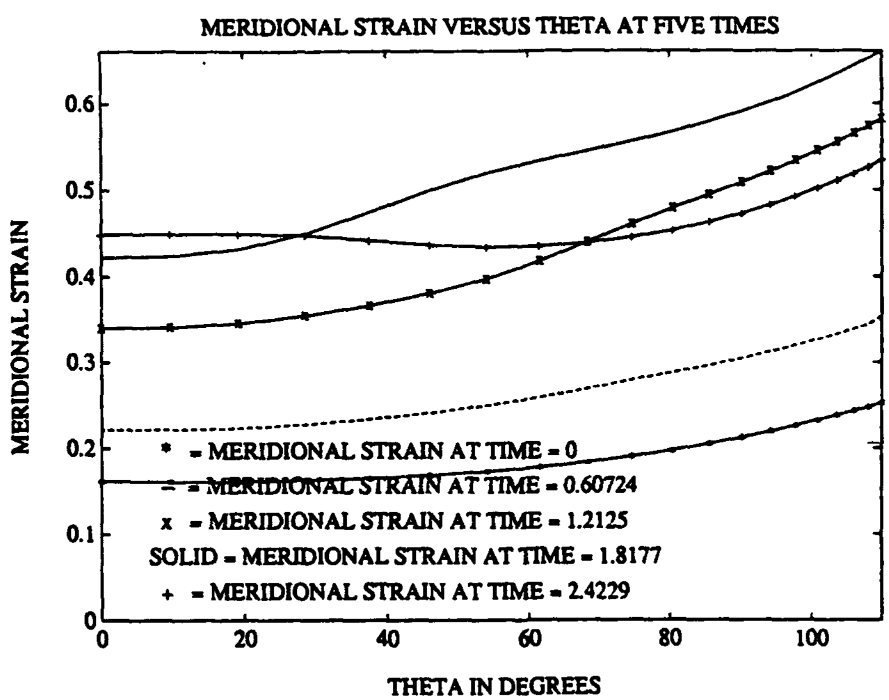

Figure 28. Meridional Strain Versus Theta at Five Times

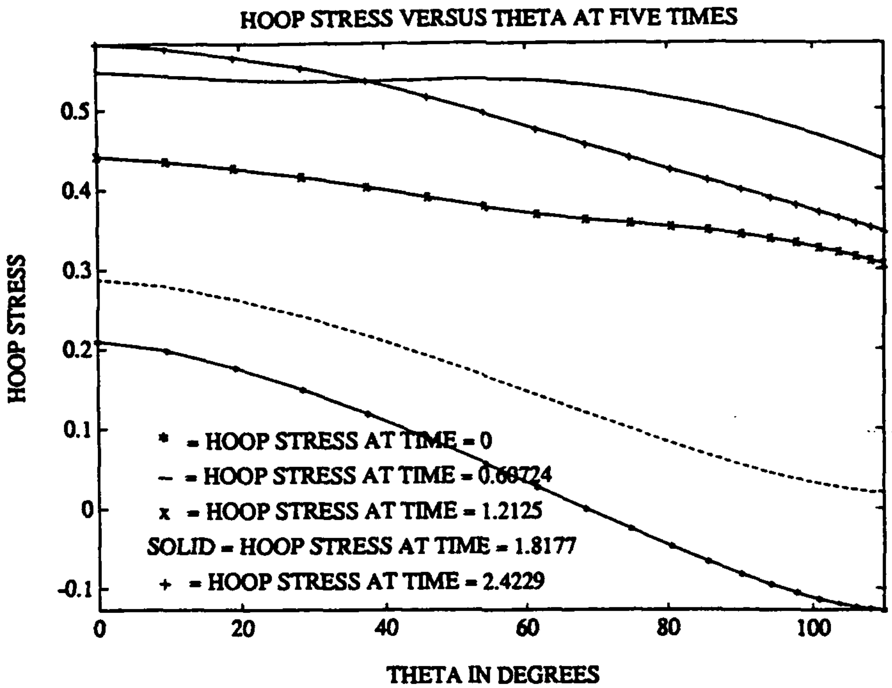

Figure 29. Hoop Stress Versus Theta at Five Times

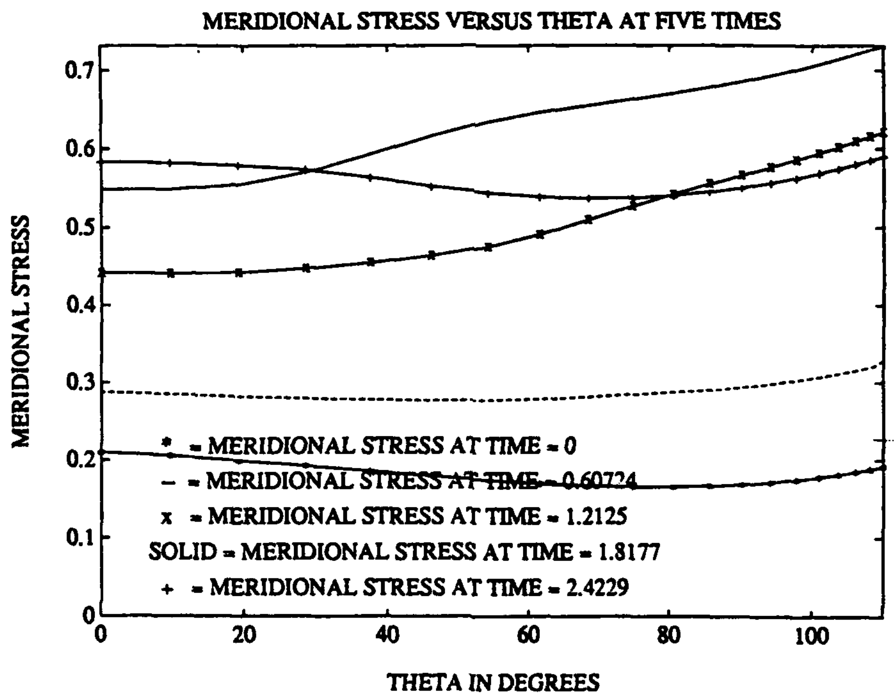

Figure 30. Meridional Stress Versus Theta at Five Times

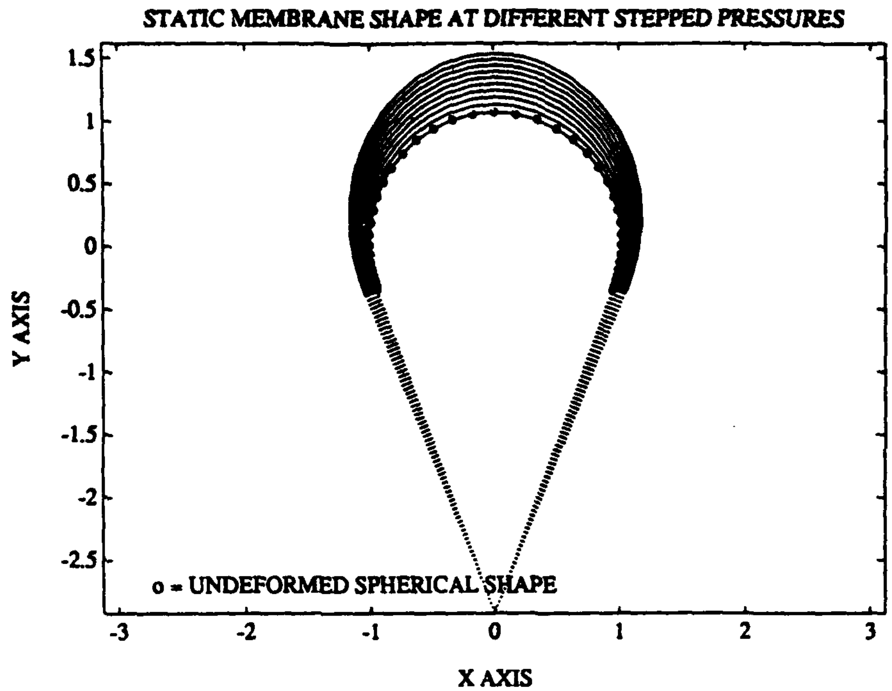

Figure 31. Static Solutions up to Final Pressure Distribution

Related Topics

The spherical membrane model is not expected to predict the entire structural dynamic opening of a canopy. However, the model is expected to provide valuable insight into problems associated with coupling and requirements for future models. This section of the report presents some other modifications and comparisons that were made while working with the spherical membrane model.

Static Comparison with NISA Finite Element Model:

The spherical membrane equations are nonlinear. The nonlinear term is introduced in the meridional strain equation presented in Appendix A equation A-1. The effect of this nonlinear term will become evident in this short section. The spherical membrane equations were incorporated into Fortran packages for solution. These programs are source codes which are easily modified. All future structural dynamic models must provide source codes if they are to be coupled with CFD codes. A search is on going for existing software with modifiable source codes that are capable of solving the structural dynamic behavior of canopies. So far, no codes have been found that can solve this problem. A few simple, statically loaded thin-shell finite element models were run using the NISA (see reference 4) finite element package. These runs were made with two purposes in mind. (1) To compare the static spherical membrane results to thin-shell finite element results. (2) To determine what elements were capable of modelling the axisymmetric problem and determine NISA's capabilities. The results of these runs are presented in Appendix F.

Vent and Annular Canopy Boundary Conditions:

Other boundary conditions than the three already mentioned were investigated briefly in an attempt to model other types of canopy designs. The ability to include a vent that is not pinned is an important feature to include in any parachute model. This feature should also be capable of modelling a large vent opening which leads to the modelling of annular parachutes. An accurate model of an annular parachute or a vent model must be capable of including lines that are connected to a variety of locations. The spherical membrane model does not have this capability but the inclusion of a movable peak node could be a useful option in the coupling problem. The usual procedure for a "free" or cantilever boundary condition in plates or shells is to prescribe a moment-free and shear-free end condition. However, these options are not available in membrane theory because bending is not included in the theory. An approximation was made to model an annular canopy boundary condition at the first node point for any theta minimum value. This option was included with the pinned bottom and infinite mass bottom boundary conditions for the static Fortran programs. The condition was coupled with only the pinned bottom boundary condition for the dynamic Fortran programs. A brief description of the approximation along with an example of static output is presented in Appendix G.

Discussion of Model Tests and Limitations

This section gives a brief review of the model's potential and some limitations. The spherical membrane model was not chosen with the intention of modelling the entire opening process of a canopy. The model was chosen because of its similarity to canopies and for the purpose of being coupled with the computational fluid dynamics package SALE. The Fortran programs are expected to be useful tools in debugging the problems associated with coupling to nonlinear sets of equations through a boundary. The Fortran programs are numerically capable of solving the spherical membrane problem with "large" deflections. These results are inaccurate for any physically real membrane. The spherical membrane equations are developed and based on deformations and pressure loadings on the undeformed membrane surface. The accuracy of the equations becomes increasingly inaccurate as deformations grow from the undeformed shape.

The Fortran programs described in this report were tested extensively. One sequence of tests ran the dynamic program with all input the same except for the number of node points used to represent the membrane. The programs converge with damping and oscillate through the same path with 11,20,30,60, and 100 node points being used to represent the membrane. The time for the runs became excessive as the number of ODE's being solved increased from 44 to 400. Another set of tests ran the dynamic program with all parameters the same except for the degree of clustering used at one or both ends of the membrane. Again, the program converged to the same results. The resolution of the solution near heavy clustering or for a large number of node points would of course increase for these tests. Also, many dynamic runs were made by starting at some deformed position and applying some type of time-dependent load for a given length of time. Then at a prescribed time the load would be held constant with time and the membrane allowed to damp out. The final damped out deflection was checked against the solution obtained using the static Fortran program with the same final pressure distribution. The final deflections were always the same.

The spherical membrane programs do have limitations. One problem was encountered while executing the dynamic program with the pinned top pinned bottom boundary conditions. The membrane initial conditions require that all node points start from rest and the deflections are an inflated shape of moderate deflection. The membrane was not loaded (pressure distribution set equal to zero for all nodes and all time). The membrane was damped and the purpose of the test run was to see if the membrane would damp out its motion to the undeformed spherical shape. The run will work for small initial displacements but will not work for moderate or large initial displacements. The problem is similar to that encountered in a snap-through problem. The membrane will move towards the undeformed shape and move into a state of pure compression. At some critical value starting with the node points next to the pinned end nodes (for initial displacements that are generated from a uniform pressure distribution) the solution becomes unstable and the deflections grow until the program crashes. The critical value and a more detailed analysis of this phenomena was not conducted because this set of initial conditions and loading conditions is not expected in the coupling problem.

Other problems are associated with the fact that all calculations are based on the undeformed geometry of the spherical membrane. This includes the inability to accurately determine the vertical load versus time graph for moderate and large deflections. The pressure distribution is applied to the undeformed sphere but the coupled problem will be applying pressures based on the current deformed geometry. Other problems and limitations will inevitably be found during the coupling process.

Conclusion

The major purpose of this report is to present a structural dynamic canopy model being used by the U.S. Army Natick Research, Development and Engineering Center as part of a larger coupling problem. The coupling of the spherical membrane model with the computational fluid dynamics code SALE is expected to aid in the prediction of the complex opening problem of parachutes. The solution of the opening problem will provide essential information to aid in the design of high-speed and low-altitude airdrop systems.

A static set of nonlinear spherical membrane equations are modified to model the dynamic response of a spherical membrane to a time-dependent pressure distribution. The two governing partial differential equations were converted into a system of nonlinear first order differential equations, which were finite differenced in space and solved numerically. The Fortran programs described in the report are capable of computing large deflections from the undeformed spherical shape. These large deflections are not an accurate model of a physical membrane, and the model was not chosen with that intention. The model can however reproduce these large "numerical" deflections which will enable Natick personnel to investigate the problems associated with large amplitude deflections and rapid motion in the coupling problem.

A set of Fortran programs have been written to solve the spherical membrane model equations. The programs have been tested with a wide range of input parameters and several examples are presented. The programs are also capable of solving the spherical membrane problem with a user-defined number of arbitrarily spaced node points. These features are incorporated to permit a relatively small number of modifications for the coupling of the codes with the significantly more complicated computational fluid dynamics package SALE. The major conclusions of this report are listed below.

(1) A spherical membrane model has been chosen as a first step to represent the structural dynamic behavior of canopies.

(2) A set of Fortran programs capable of solving a large variety of dynamic and static spherical membrane problems have been written and tested with a variety of different input parameters. A modified set of these programs is currently being coupled to the CFD code SALE at Natick.

(3) A MATLAB program has been written for postprocessing the results of the Fortran programs. The MATLAB program will also be a useful tool for future structural dynamic canopy models and for the structural results from the coupling problem.

(4) The spherical membrane model is not capable of solving the full opening process of a parachute because the model includes

compressive stresses and is not accurately modeling large deflections. However, the model is expected to provide insight into the difficulties of the coupling problem.

References

1. Amsden, A.A., Ruppel, H.M., and Hirt, C.W. "SALE: A Simplified ALE Computer Program for Fluid Flow at All Speeds." Los Alamos Scientific Laboratory Report No LA-8095, 1980.

2. Ferziger, Joel H. "Numerical Methods for Engineering Application." John Wiley and Sons. N.Y. 1981.

3. Hoffmann, Klaus A. "Computational Fluid Dynamics for Engineers." Engineering Education System™ P.O. Box 8148 Austin, TX 78713-8148, 1989.

4. NISA, Engineering Mechanics Research Corporation. P.O. Box 696, Troy, Michigan 48099.

5. SLATEC Library, A collection of FORTRAN mathematical subprograms available through the National Energy Software Center (NESC)

6. Stoker, J.J. "Nonlinear Elasticity." Gordon and Breach. N.Y. 1968.

Appendix A. Development of Dynamic Membrane Equations

This Appendix section describes the equations (A-1)-(A-4) of this report. The addition of inertia terms and damping terms to the static equilibrium equations developed by Stoker are discussed first. The Stoker equilibrium equations state that the sum of the forces in the tangential and normal direction of any differential element of the spherical membrane must sum to zero. The equations developed by Stoker are shown in equations (A-1)-(A-4) when the right hand side of the equilibrium equations is set to zero and all partial derivatives with respect to theta are changed to ordinary derivatives. Stoker derives these equations by applying the principle of minimum potential energy. The addition of inertia terms requires that the sum of the forces be equal to the mass times the respective component of acceleration of the differential element. The acceleration is defined as the second partial derivative of the corresponding displacement with respect to time. The mass is computed as the product of the membrane material density and the volume of the differential element. The damping terms are included as motion resistors. They apply restoring forces linear in magnitude to the respective velocity at any location on the sphere. The damping forces are not modelling any realistic damping process but are included to help stabilize the solution numerically. The damping also allows the user to compare the results of a completely damped out motion from a time-independent load to the solution obtained from the same load applied to the static problem. Equations (A-1)-(A-4) are six governing equations for six unknowns which describe the dynamic response of the spherical membrane.

Stress Strain

$$E\varepsilon_\theta=\sigma_\theta-\nu\sigma_\phi \qquad E\varepsilon_\phi=\sigma_\phi-\nu\sigma_\theta \tag{A-1}$$

Strain Displacement

$$\varepsilon_\theta=\frac{1}{R}\left(\frac{\partial u}{\partial\theta}-w\right)+\frac{1}{2R^2}\left(\frac{\partial w}{\partial\theta}+u\right)^2 \qquad \varepsilon_\phi=\frac{1}{R}(u\cot\theta-w) \tag{A-2}$$

Equilibrium: u-direction

$$\frac{\partial}{\partial\theta}(\sigma_\theta\sin\theta)-\frac{\sigma_\theta\sin\theta}{R}\left(\frac{\partial w}{\partial\theta}+u\right)-\sigma_\phi\cos\theta=C_u\frac{\partial u}{\partial t}+\rho R\sin\theta\frac{\partial^2 u}{\partial t^2} \tag{A-3}$$

Next, the six governing equations are reduced to two partial differential equations. The first step taken is to rewrite the

Equilibrium: w-direction

$$\frac{1}{R}\frac{\partial}{\partial\theta}[\sigma_\theta \sin\theta(\frac{\partial w}{\partial\theta}+u)]+(\sigma_\theta+\sigma_\phi+\frac{Rp}{h})\sin\theta=C_w\frac{\partial w}{\partial t}+\rho R\sin\theta\frac{\partial^2 w}{\partial t^2} \quad \text{(A-4)}$$

stress strain equations in terms of strains. Next, substitute the values of strains in terms of displacements from the strain displacement equations. The resulting two equations define each stress in terms of displacements. These two equations are substituted into the two equilibrium equations to yield the two governing partial differential equations involving only the tangential and normal deflections as unknowns. These equations are shown in equations one and two of the "Formulation and Analysis" section of the report.

Appendix B. Development of Finite Difference Equations

This Appendix derives the Second order accurate FDE's with a variably spaced nodal grid point option. The derivation of the FDE's starts with the standard second order Lagrange Polynomial shown in equation (B-1) (see reference 2).

$$f(x)=f(x_{i-1})\frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})}+f(x_i)\frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})}+ \\ f(x_{i+1})\frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} \tag{B-1}$$

Note that for this general derivation x is the independent variable, f is the dependent variable and h is the spacing defined as $h_i=x_{i+1}-x_i$. The Lagrange Polynomial is differentiated with respect to (x) and evaluated at the desired nodes. Evaluation at $x=x_i$ yields the central difference formula. Evaluation at $x=x_{i-1}$ yields the forward difference formula, and evaluation at $x=x_{i+1}$ yields the backwards difference formula. The backward, central and forward difference formulas for the first derivative are shown in equation (B-2) where H is defined by $H=h_{i+1}+h_i$.

$$f'(x_{i+1})=f(x_{i-1})\frac{h_{i+1}}{h_iH}-f(x_i)\frac{H}{h_ih_{i+1}}+f(x_{i+1})\frac{2h_{i+1}+h_i}{h_{i+1}H}$$

$$f'(x_i)=-f(x_{i-1})\frac{h_{i+1}}{h_iH}+f(x_i)\frac{1}{h_i-h_{i+1}}+f(x_{i+1})\frac{h_i}{h_{i+1}H} \tag{B-2}$$

$$f'(x_{i-1})=-f(x_{i-1})\frac{2h_i+h_{i+1}}{h_iH}+f(x_i)\frac{H}{h_ih_{i+1}}-f(x_{i+1})\frac{h_i}{h_{i+1}H}$$

The second derivative can be computed from the Lagrange Polynomial by differentiating with respect to x twice. The result is a constant second derivative between any given three adjacent node points. The second derivative is shown in equation (B-3).

$$f''(x_{i-1}\le x\le x_{i+1})=f(x_{i-1})\frac{2}{h_iH}-f(x_i)\frac{2}{h_ih_{i+1}}+f(x_{i+1})\frac{2}{h_{i+1}H} \tag{B-3}$$

Appendix C. Development of Symmetry Boundary Condition

This Appendix derives the mathematical and numerical statements representing the symmetry boundary condition at theta minimum equal to zero degrees. As previously mentioned, the tangential velocity and acceleration at the peak node must be set to zero for the dynamic program. The tangential displacement must be set equal to zero for the static Fortran program. The value of the normal acceleration at node number one is found by applying the equilibrium equation at that point. The problem is that many of the terms in the equilibrium equation are undefined at theta=0.0. For example cot(0) is undefined. The undefined terms are in the form of zero/zero so L'Hopital's Rule is applied to each of the problem terms as follows;

$$\begin{gathered}
\cot\theta \frac{du}{d\theta}\frac{dw}{d\theta} \rightarrow \frac{du}{d\theta}\frac{d^2w}{d\theta^2} \\
w\cot\theta \frac{dw}{d\theta} \rightarrow w\frac{d^2w}{d\theta^2} \\
\cot\theta \left(\frac{dw}{d\theta}\right)^3 \rightarrow 0 \\
u\cot\theta \left(\frac{dw}{d\theta}\right)^2 \rightarrow 0 \\
u^2\cot\theta \frac{dw}{d\theta} \rightarrow 0 \\
u\cot\theta \frac{du}{d\theta} \rightarrow \left(\frac{du}{d\theta}\right)^2 \\
uw\cot\theta \rightarrow w\frac{du}{d\theta} \\
u^3\cot\theta \rightarrow 0 \\
u\cot\theta \rightarrow \frac{du}{d\theta} \\
u\cot\theta \frac{d^2w}{d\theta^2} \rightarrow \frac{du}{d\theta}\frac{d^2w}{d\theta^2}
\end{gathered} \qquad \text{(C-1)}$$

The first term shown in equation (C-1) is derived in equation (C-2) to illustrate the method.

$$\cot\theta \frac{du}{d\theta}\frac{dw}{d\theta} \rightarrow \frac{\cos\theta \frac{du}{d\theta}\frac{dw}{d\theta}}{\sin\theta} \rightarrow \frac{\frac{d}{d\theta}\left[\cos\theta \frac{du}{d\theta}\frac{dw}{d\theta}\right]}{\frac{d}{d\theta}[\sin\theta]} \rightarrow \frac{du}{d\theta}\frac{d^2w}{d\theta^2}\bigg|_{[\theta=0]} \qquad \text{(C-2)}$$

The other terms are derived by the same procedure. The acceleration at theta minimum equal to zero is rewritten by applying the symmetry boundary conditions using the expressions from equation C-1.

Appendix D. Development of Infinite Mass Boundary Condition

This Appendix derives the mathematical and numerical statements representing the infinite mass boundary condition at theta maximum. Figure D-1 shows a schematic of the spherical membrane with imaginary lines. The lines have a fixed length L and the model is fixed at point A to restrict rigid body motion.

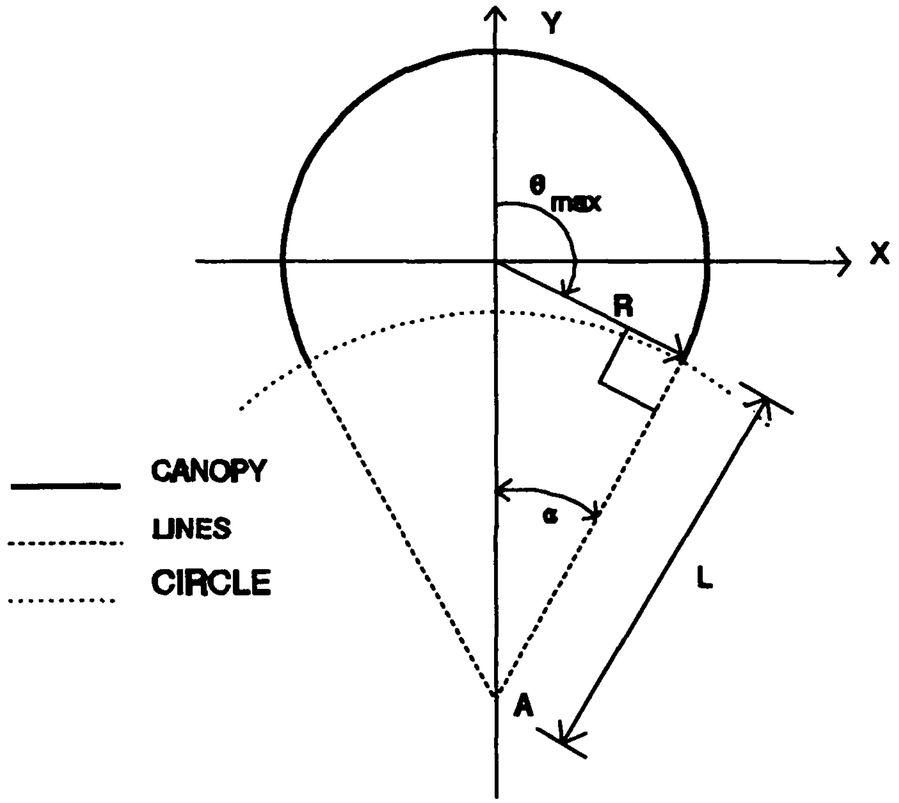

Figure D-1. Infinite Mass Boundary Condition at θ_{max}

The point A defines the center of a circle with a radius of length L. The point A is defined by the intersection of the line (tangent to the bottom edge of the membrane) with the global Y axis. The spherical membrane still has its undeformed geometric center at the origin of the global XY coordinate system. The infinite mass boundary condition requires the bottom edge node of the sphere to follow the circular arc defined by point A and radius L. The membrane must also intersect the circular arc at a normal angle. This condition is equivalent to restricting discontinuities at the connection between the bottom node point and the imaginary lines. These conditions require that the value of theta maximum be greater than 90 degrees. The line length L is defined by the angle theta maximum and the radius of the undeformed spherical membrane. Equation (D-1) is the equation of the circular arc centered at point A of radius L.

$$X^2+\left[Y+\frac{R}{\sin(\theta max-90)}\right]^2=\frac{R^2}{\tan(\theta max-90)^2} \qquad \text{(D-1)}$$

Static Program Implementation:
The infinite mass boundary condition was added as an option to the static Fortran program by writing the conditions in two equations. The two equations are minimized by the SLATEC subroutines. The first equation requires node number "nth" to follow the prescribed circular arc in equation D-1. The equation is written in terms of tangential and normal displacements and the radius squared term is subtracted from both sides. This yields one condition that must be minimized. The second condition is obtained by subtracting the slope of the lines from the slope of node point "nth" which is calculated from a backwards difference formula using node points "nth", "nth-1", and "nth-2". The difference in slopes must be zero if the "tangent" boundary condition is to be met. This equation can be written in terms of tangential and normal deflections.

Dynamic Program Implementation:
The infinite mass boundary condition applied to the dynamic problem was harder to derive. The boundary conditions are in terms of displacements but the program needs conditions on velocities and accelerations. The problem was solved by actually solving the dynamic program for every node except node "nth". The system of equations needed is reduced to 4*(nth-1). The components of velocity at node point "nth" are interpolated from the velocities at node points "nth-1", "nth-2", and "nth-3". The displacements at node point "nth" are calculated within the user-supplied function evaluation subroutine used by the SLATEC software. The value of the displacements at node point "nth" are needed to evaluate the acceleration function of node point "nth-1" within the function evaluation subroutine. The value of the *displacements at node point* "nth" is calculated within this subroutine as described in the "Static Program Implementation" section. However, a fifth order backwards finite difference equation is used to compute the slope at node point "nth". The resulting two nonlinear equations for the current tangential and normal displacements at node point "nth" are solved by calling the DNSQE.f subroutine within every call to the function evaluation subroutine.

Appendix E. Comparison with Static Results Presented by Stoker

Stoker (see reference 5) gives results to a sample spherical membrane problem. The same example problem was run with the static Fortran program described here and the results compared. The input parameters used by Stoker are defined in one nondimensional parameter K. K is defined in equation (E-1).

$$K=\frac{pR}{Eh}=-1.56E-03 \tag{E-1}$$

The pressure is taken as a constant value for all values of theta. The value of theta minimum equals 0.0 degrees and the value of theta maximum is 11.4591 degrees. The problem can be visualized as a dome with pinned edges and a constant internal pressure. The values needed for the static Fortran program are dimensional. Therefore the values of the thickness, radius, and Young's modules where specified and the corresponding value of the constant pressure was determined from equation (E-1). The value of Poisson's ratio was also kept the same as Stokers value of 0.3. A listing of the input values chosen is shown in Table E-1.

TABLE E-1 Definition of K in Stoker Text

Input Parameter Description	Input Value & Dimensions
R = Radius of Sphere	300 inches
h = Thickness of Membrane	0.005 inches
E = Young's Modules	30000. psi
p = Pressure	-0.00078 psi

The static Fortran program was run with these values and the results were converted into graphs of the same form as the graphs presented by Stoker. Figure E-1 shows the graphical results from the static Fortran program run. Figure E-1 combines the three plots presented by Stoker. The plots in Figure E-1 are identical to the plots presented by Stoker.

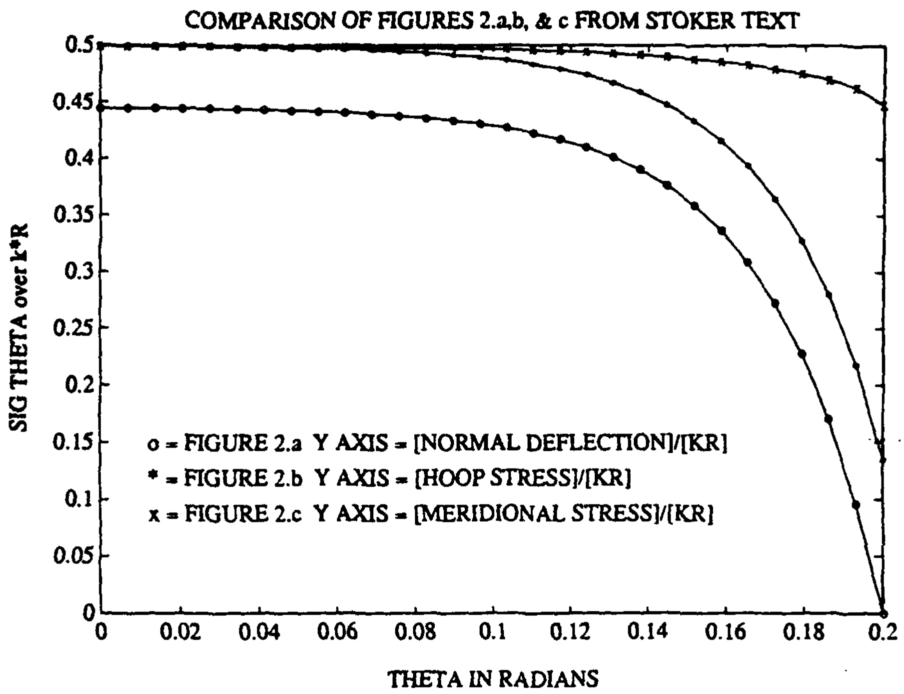

Figure E-1. Comparison of Figures 2a,b, and c from Stoker Text

Appendix F. Static Comparison with NISA Finite Element Model

The finite element code NISA (see reference 4) was used to model the spherical membrane with "pinned bottom" - "pinned top" boundary conditions. The NISA library of elements does not include an element to model "membranes". The library does include shell elements. The shell elements include bending and are therefore stiffer than the membrane model. Also, the runs made with NISA are linear and the comparison highlights the nonlinear contribution to the solution of the spherical membrane equations at moderate loads. Axisymmetric NISA elements (NKTP=36) were attempted for the problem but were unable to converge to a smooth boundary region near the pinned ends. The "smoothest" results were obtained by modelling the sphere with "general 3-D shell elements" (NKTP=20). The best of these elements for the model was found to be the six node elements (NORDR=9). The NISA model was run with different constant pressure distributions. The radius, thickness, Poisson's ratio, and Young's modules are the same as those shown in Table E-1. The value of theta maximum is 90 degrees and the value of theta minimum is 20 degrees. The model consisting of general 3-D shell elements takes the symmetry of the problem into consideration by modelling only one quarter of the sphere. One typical finite element grid using 150 elements and 341 node points is shown in Figure F-1.

This grid was used to solve the static program with a constant pressure distribution of -0.001 psi for all values of theta. The NISA-predicted displacements were converted into nondimensional tangential and normal displacements. The results of the NISA run are compared with the results using the static Fortran program in Figures F-2 and F-3. The results are quite close. However, the NISA results appear to have trouble converging smoothly near the boundary points.

A second run was made using a similar grid to that shown in Figure F-1 but with more elements. The grid used 2189 node points and 660 elements. The comparison run was made using the same input as the previous run except the pressure distribution was increased by a factor of one hundred to -0.1 psi for all theta. A comparison of the resulting deflections is shown in Figures F-4 and F-5. The linear NISA results are obvious by comparison to the previous example with 1/100th of the load. The NISA run deflections increased proportionally to the applied load. The nonlinear effect of the spherical membrane equations is also evident. The membrane deflections do not increase linearly with the applied load. The membrane solutions also appear to be less stiff then the NISA solutions, which could be due to the bending contributions included in the NISA elements. The tangential deflection versus theta curves illustrate this difference.

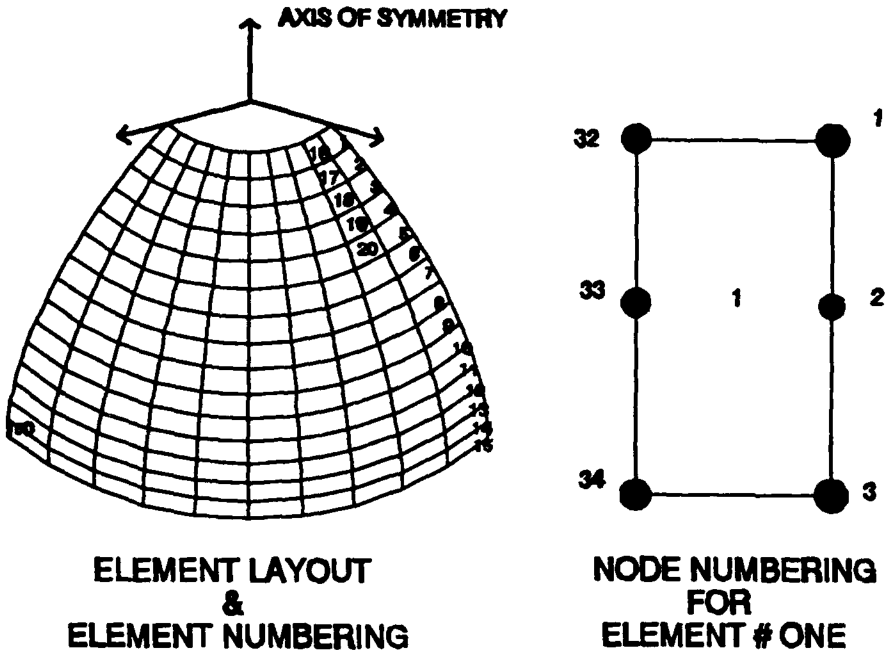

Figure F-1. Element & Node Point Layout for NISA Model

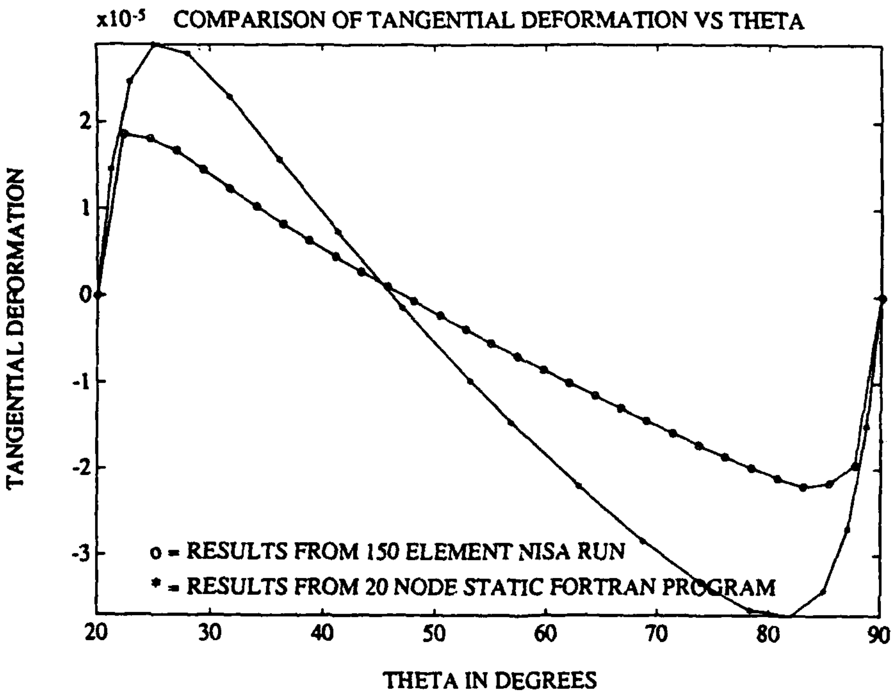

Figure F-2. Comparison of U deflection Versus Theta (150)

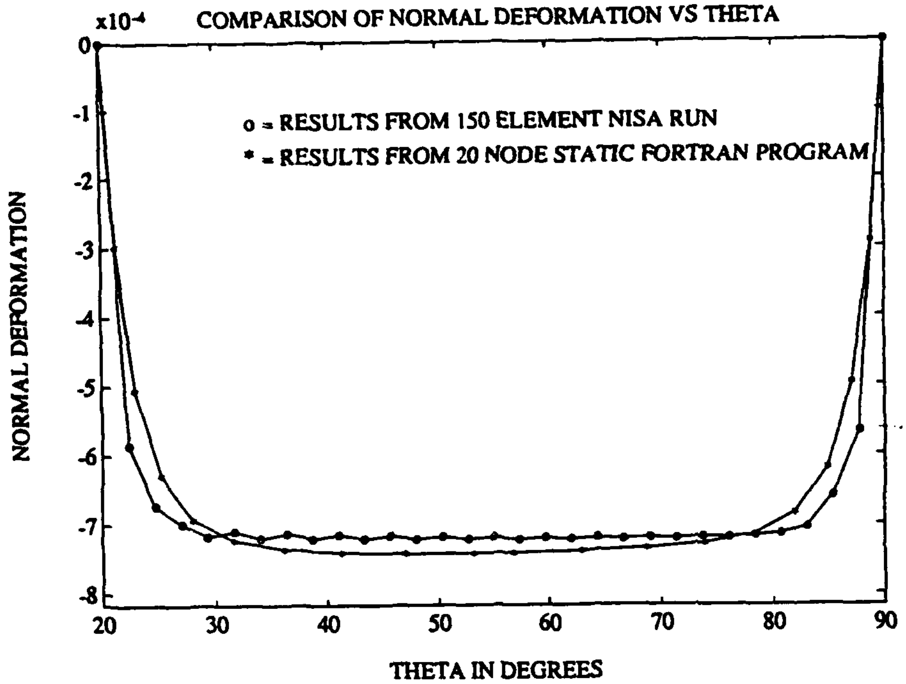

Figure F-3. Comparison of W Deflection Versus Theta (150)

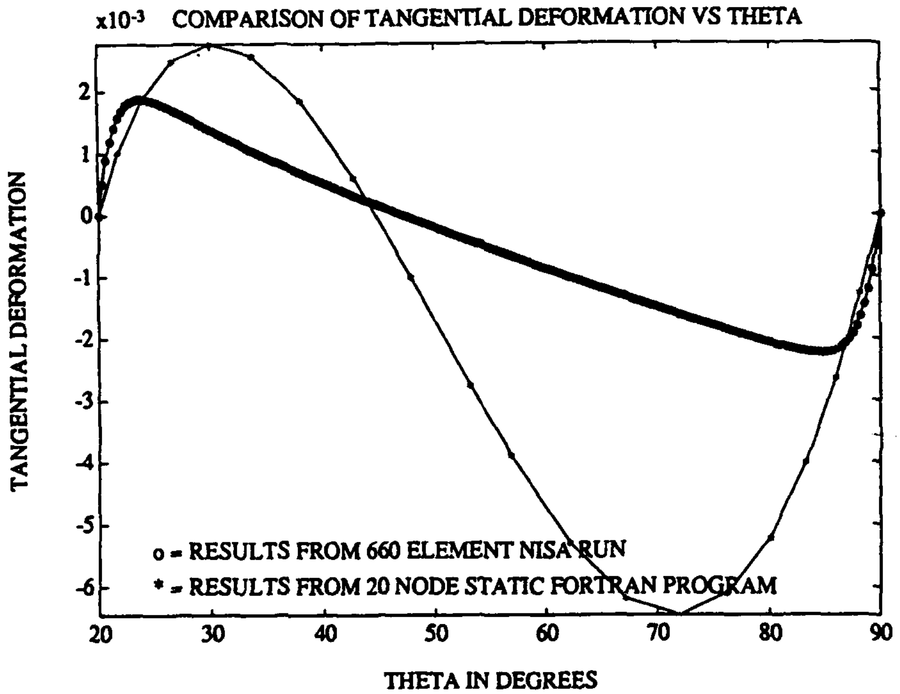

Figure F-4. Comparison of U deflection Versus Theta (660)

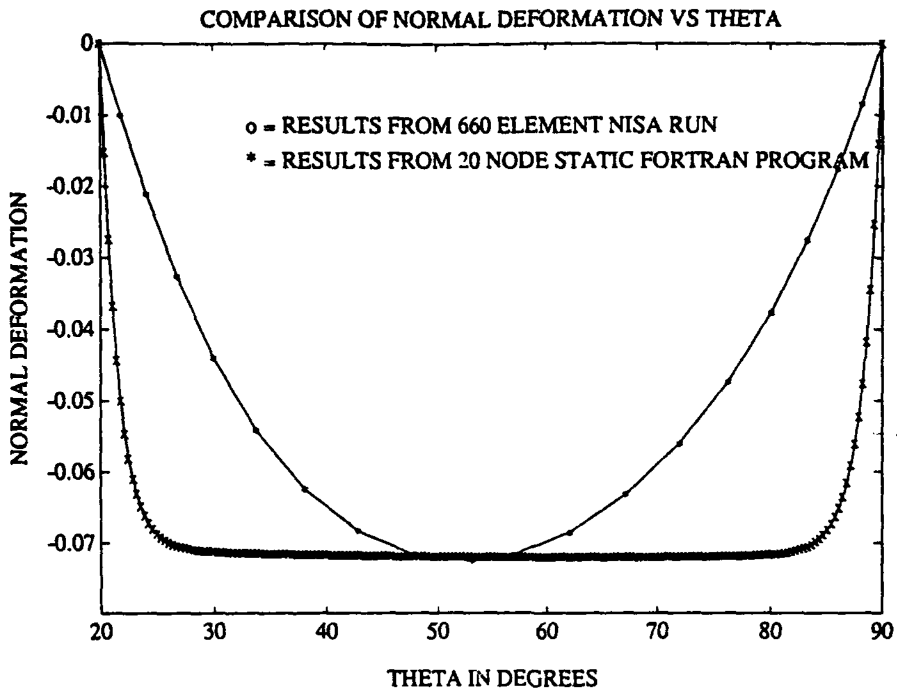

Figure F-5. Comparison of W Deflection Versus Theta (660)

Appendix G. Vent and Annular Canopy Boundary Conditions

This Appendix describes the approximations used to model a vent and or an annular canopy. The method is specific to the spherical membrane equations and an example of the approximation will be given. The allowable boundary conditions at each end from the derivation by Stoker reduce to specifying a condition on both the normal and tangential deflections at each end node point. The first condition that must be satisfied for a vent or annular canopy is that the value of the meridional stress must be zero at the peak node point. Physically this says that there is no meridional load applied at that node point. Mathematically this condition is stated in equation (G-1).

$$\sigma_\theta\big|_{(at\ \theta_{min})}=\frac{du}{d\theta}-w+\frac{1}{2}\left(\frac{dw}{d\theta}+u\right)^2+\nu\left(u\cot\theta-w\right)=0 \qquad \text{(G-1)}$$

The second boundary condition at theta minimum is difficult to determine. A variety of conditions were attempted. The best result (based solely on the fact that the solutions yielded the most continuous curves) was to extrapolate the value of the normal deflection of the node point at theta minimum. This was accomplished by fitting a second order polynomial to the normal displacements of node points two, three, and four. The resulting equation in terms of theta was used to extrapolate the value of the normal deflection at node point number one. The actual physical interpretation of this condition was not determined.

A static example using this boundary condition with nine stepped up pressures is presented next. The example uses the infinite mass boundary conditions at theta maximum defined as 120 degrees. The value of theta minimum is 20 degrees. The input parameters are the same as the parameters listed in Table E-1. Also, clustering at both ends was used with beta equal to 1.1. The pressure is constant for all theta and steps to a final value of -0.3 psi. The deformed shapes are shown in Figure G-1. The tangential and normal deflections versus theta are shown in figures G-2 and G-3, respectively.

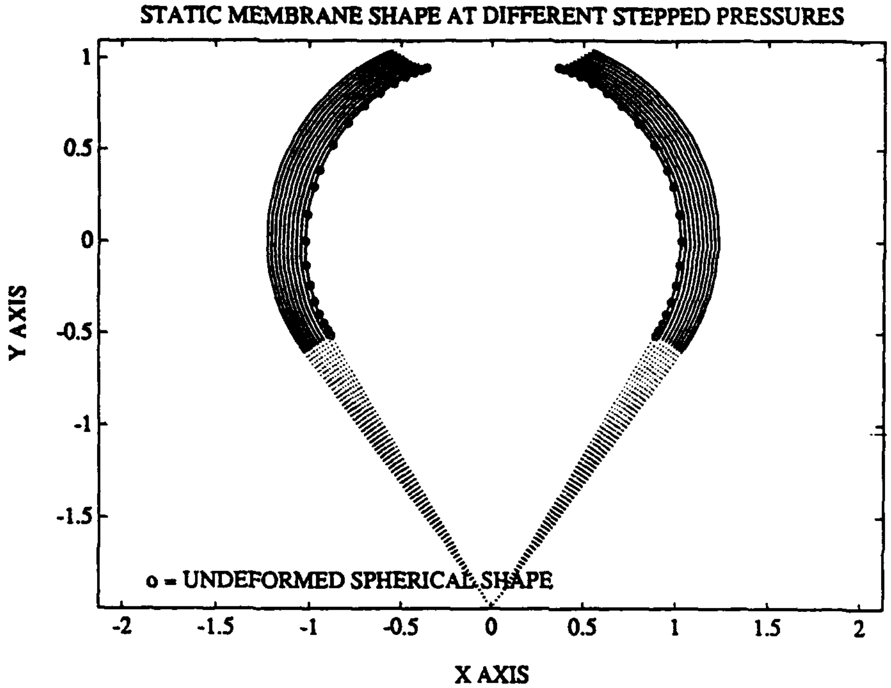

Figure G-1. Static Deformed Shapes at Stepped up Pressures

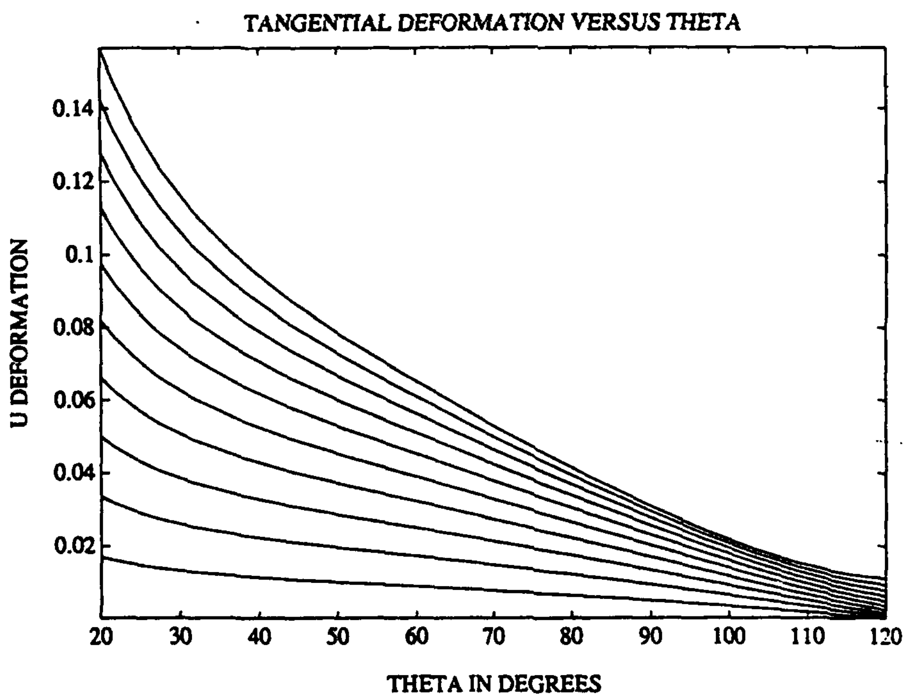

Figure G-2. Tangential Deflections Versus Theta

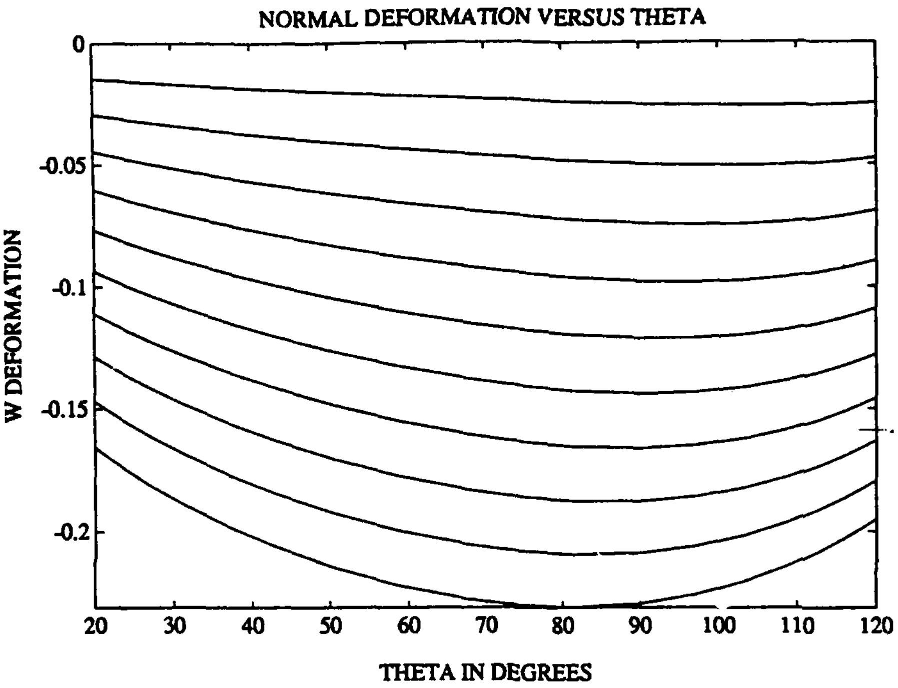

Figure G-3. Normal Deflections Versus Theta

Appendix H. Static Fortran Programs

```
c    ****************************************************
c         STATIC PROGRAM AND ATTACHED SUBROUTINES
c    ****************************************************
c
c         ******* NON DIMENSIONALIZED VERSION    *******
c      FORTRAN PROGRAM to solve static version of nonlinear
c      membrane equations see "Nonlinear
c      Elasticity" written by J.J. Stoker 1968 page 32.
c       THE PROGRAM generates three MATLAB readable
c       matrices which can be loaded into MATLAB.
c       The MATLAB program STATIC.m generates graphical
c       results of the run.
c
        parameter (nth=20)
        parameter (nnn=nth+nth)
        parameter (lwa=(6*nnn**2+13*nnn)/2)
c
        implicit double precision(a-h,o-z)
        double precision fvec(nth+nth),y(nth+nth),wa(lwa),tol
        real*8 el,epi(nth)
        real*8 eth(nth),h,hh(nth),nu,p(nth),parnd(nth),pi
        real*8 pres(nth),presinc,r
        real*8 pinc,pfact,pnow(nth)
        real*8 spi(nth),spimax,sth(nth),sthmax
        real*8 x(nth),xmax,xx,zz
        integer i,ibc,info,iopt,n,nprint
        common/prop/ nu,parnd,x,hh,pi,xmax,xmin
        external ndfcn,ndfcnpt,ndfcnim
        external ndstress
c
        open(12,file='ndstatic.m')
c        *****LIST OF FILES/PROGRAMS CALLED OR OPENED*************
c        open ndstatic.m
c        call ndfcn.f
c        call ndfcnpt.f
c        call ndfcnim.f
c        call ndstress.f
c        open NDSTART
c        open NDCONTINUE
c      *******************  DEFINITIONS  *********************
c        aparab=constant used to generate a parabolic
c               pressure distribution ipress=2
c        beta  =parameter used to generate a clustered grid
c               pos. beta>1.0
c        binter=intercept value used to define linear pressure
c               distribution ipress=1
c        bparab=constant used to generate a parabolic pressure
c               distribution ipress=2
c        cc    =used as variable in grid generation
c        cparab=constant used to generate a parabolic pressure
c               distribution ipress=2
c        dd    =used as variable in grid generation
c        el    =Youngs Modulus divided by
c               (one minus Poisson's ratio squared)
c        em    =Youngs Modulus
c        epi(i)=strain "epsilon phi" at node "i"
```

```
c       eth(i)=strain "epsilon theta" at node "i"
c       epimax=max. value of epi for run
c       epimin=min. value of epi for run
c       ethmax=max. value of eth for run
c       ethmin=min. value of eth for run
c       gamma =used in grid generation for iopt=4
c       h      =membrane thickness (constant)
c       hh(i) =the difference in radians from theta at
c              node "i" to theta at node "i-1"
c       i,j,k=integer counter for various loops
c       ibc    =parameter which defines which B.C. to use
c       iopt   =variable grid options four types
c       ipress=pressure vs. theta options three types
c       jkl    =integer = 1/2 of nth (used in grid
c              generation for iopt=4)
c       NDSTART=file created by ndstatic.f contains two
c             columns by nth rows (init. def. used by dampall.f)
c       nth    =total number of nodes on the membrane
c       nu     =Poissons ratio
c       parnd(i)=non-dimensional pressure=p(i)*r/(el*h)
c       pi     =double precision value for pi=3.14.....
c       p(i)   =pressure at node point i, a function of theta
c       pslope=slope of press. vs. theta for linear
c              pressure distribution ipress=1
c       py1,px1 etc. =used to generate parabola
c                    equation when ipress=2
c       r      =undeformed radius of sphere in inches
c       sdefxmin=next four min and max values of deformed sphere
c       sdefymin=
c       sdefxmax=
c       sdefymax=
c       sth(i)=sigma theta at node i
c       spi(i)=sigma phi at node i
c       spimax=max. value of spi for run
c       spimin=min. value of spi for run
c       sthmax=max. value of sth for run
c       sthmin=min. value of sth for run
c       sumin=min. value of u-deflections for the entire run
c       sumax=max. "     "      "
c       swmin=min.  "     "  w-def.  "
c       swmax=max.  "
c       wmax   =
c       x(i)   =value of theta at node "i" (theta=0.0 at top
c               of sphere) max value is xmax
c       xmax   =maximum value of theta in ****degrees****
c       xmin   =value of theta at top of sphere annular
c               case in degrees
c       y(i)   =u-def. for 1 to nth,w-def. for nth+1
c       zbot   =used in parabolic pressure distribution ipress=2
c       ztop   =used in parabolic pressure distribution ipress=2
c       zz     =used as variable in grid generation
c      *********  MATERIALS AND DIMENSIONS  *********************
       write(*,*)'ENTER value of ibc'
       write(*,*)'    1=symmetry-top pinned-bottom'
       write(*,*)'    2=pinned-top pinned-bottom'
       write(*,*)'    3=infinite mass B.C.'
       read(*,*)ibc
       write(*,*)
c
       write(*,*)'ENTER THE PARAMETERS WHEN ASKED "0" =DEFAULT'
       write(*,*)
       write(*,*)'ENTER THE RADIUS IN INCHES (300)'
```

```
      read(*,*)r
      if(r.eq.0.0)r=300.0
      write(*,*)'ENTER POISSON RATIO (0.3)'
      read(*,*)nu
      if(nu.eq.0.0)nu=0.3
      write(*,*)'ENTER YOUNGS MODULUS (30000.0)'
      read(*,*)em
      if(em.eq.0.0)em=30000.0
      el=em/(1.-nu**2.)
      if(ibc.ne.1.and.ibc.ne.3)then
       write(*,*)'ENTER XMIN THETAMIN AT TOP OF SPHERE (DEG) (0)'
       read(*,*)xmin
      else
      endif
      write(*,*)'ENTER XMAX THETAMAX AT EDGE OF SPHERE (DEG) (90)'
      read(*,*)xmax
      if(xmax.eq.0.0)xmax=90.0
      write(*,*)'ENTER THE MEMBRANE THICHNESS INCHES (0.005)'
      read(*,*)h
      if(h.eq.0.0)h=0.005
      write(*,*)'ENTER THE VALUE OF TOL (1.0E-12)'
      read(*,*)tol
      if(tol.eq.0.0)tol=1.0E-12
      pi=4.*DATAN(1.D00)
c
c        ********  grid generation section  ***********
       write(*,*)'There are FOUR built in grid options'
       write(*,*)'ENTER 1= uniform spacing'
       write(*,*)'ENTER 2= clustering at peak node (node=1)'
       write(*,*)'ENTER 3= clustering at edge node (node=nth)'
       write(*,*)'ENTER 4= clustering at both ends'
       read(*,*)iopt1
       hh(1)=(xmax-xmin)*(pi/180.)/(nth-1)
c
       if(iopt1.eq.1)then
        x(1)=xmin*(pi/180.)
        do 7 i=2,nth,1
         hh(i)=hh(1)
         x(i)=x(i-1)+hh(i)
  7     continue
        hh(1)=hh(2)
       else
       endif
c
       if(iopt1.eq.2)then
        write(*,*)'ENTER BETA more clustering as beta gos to 1'
        read(*,*)beta
        zz=0.0
        do 8 i=nth,1,-1
         gamma=zz/((xmax-xmin)*(pi/180.))
         dd=((beta+1.)/(beta-1.))**gamma
         cc=1.-beta*(dd-1.)/(dd+1.)
         x(i)=cc*(xmax-xmin)*(pi/180.)
         zz=zz+hh(1)
         x(i)=xmin*(pi/180.)+x(i)
  8     continue
       else
       endif
c
       if(iopt1.eq.3)then
        write(*,*)'ENTER BETA more clustering as beta gos to 1'
        read(*,*)beta
```

```
         zz=0.0
         do 9 i=1,nth,1
          gamma=zz/((xmax-xmin)*(pi/180.))
          dd=((beta+1.)/(beta-1.))**gamma
          cc=beta*(dd-1.)/(dd+1.)
          x(i)=cc*(xmax-xmin)*(pi/180.)
          zz=zz+hh(1)
          x(i)=xmin*(pi/180.)+x(i)
 9       continue
        else
        endif
c
        if(iopt1.eq.4)then
         write(*,*)'ENTER BETA more clustering as beta gos to 1'
         read(*,*)beta
         write(*,*)'ENTER JKL =1/2 of nth nth=',nth
         read(*,*)jkl
         hh(1)=(xmax-xmin)*(pi/180.)/(nth-1)
         zz=0.0
         do 119 i=jkl+1,nth,1
          gamma=zz/(0.5*(xmax-xmin)*(pi/180.)-hh(1)/2.)
          dd=((beta+1.)/(beta-1.))**gamma
          cc=beta*(dd-1.)/(dd+1.)
          x(i)=cc*(0.5*(xmax-xmin)*(pi/180.)-hh(1)/2.)
          zz=zz+hh(1)
          x(nth-i+1)=((xmax-xmin)/2.0+xmin)*(pi/180.)-hh(1)/2.-x(i)
          x(i)=((xmax-xmin)/2.0+xmin)*(pi/180.)+hh(1)/2.+x(i)
 119     continue
        else
        endif
c
         do 6 i=2,nth,1
          hh(i)=x(i)-x(i-1)
 6       continue
         hh(1)=hh(2)
c     PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
c              PRESSURE SECTION
        write(*,*)'ENTER the type of pressure distribution'
        write(*,*)' 1=linear pressure along meridian'
        write(*,*)' 2=parabolic pressure along meridian'
        read(*,*)ipress
c
         if(ipress.eq.1)then
          write(*,*)'Enter value of the pressure at theta=xmin'
          read(*,*)pmin
          write(*,*)'Enter value of the pressure at theta=xmax'
          read(*,*)pmax
          pslope=(pmax-pmin)/((xmax-xmin)*(pi/180.))
          binter=pmin-pslope*xmin*(pi/180.)
          do 678 i=1,nth,1
           p(i)=pslope*x(i)+binter
           parnd(i)=p(i)*r/(e1*h)
 678      continue
         else
         endif
c
         if(ipress.eq.2)then
          write(*,*)'Enter value of the pressure at theta=xmin'
          read(*,*)pmin
          write(*,*)'Enter value of the pressure at theta=xmax'
          read(*,*)pmax
          write(*,*)'Enter value of theta for the third point'
```

```
          read(*,*)thetamid
          write(*,*)'Enter value of the pressure at third point'
          read(*,*)pmid
          py1=pmin
          py2=pmid
          py3=pmax
          px1=xmin*(pi/180.)
          px2=thetamid*(pi/180.)
          px3=xmax*(pi/180.)
          ztop=(py1-py3)*(px2-px3)-(py2-py3)*(px1-px3)
          zbot=(px1**2.-px3**2.)*(px2-px3)-
     1         (px2**2.-px3**2.)*(px1-px3)
          aparab=ztop/zbot
          bparab=(aparab*(px3**2.-px2**2.)+py2-py3)/(px2-px3)
          cparab=py3-aparab*px3**2.-bparab*px3
          do 679 i=1,nth,1
           p(i)=aparab*x(i)**2.+bparab*x(i)+cparab
           parnd(i)=p(i)*r/(el*h)
 679      continue
         else
         endif
c      PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
c       write matrix of undeformed geometry in MATLAB format
         write(12,*)'gg=['
         do 333 i=1,nth,1
          write(12,*)x(i)
 333     continue
         write(12,*)'];'
c      *******************************************************
c      INPUT INITIAL GUESS
       do 5 i=2,nth,1
        y(i)=-0.00000001
        y(i+nth)=-0.0000001
 5     continue
         y(1)=0.0
         y(nth)=0.0
        if(xmin.eq.0.0)then
         y(1+nth)=-0.0000001
        else
         y(1+nth)=0.0
        endif
        y(2*nth)=0.0
c      *******************************************************
        sumin=1.0
        sumax=0.0
        swmin=1.0
        swmax=0.0
        sdefxmin=100.0
        sdefymin=100.0
        sdefxmax=-100.0
        sdefymax=-100.0
        sthmax=-100.0
        spimax=-100.0
        sthmin=100.0
        spimin=100.0
        ethmax=-100.0
        epimax=-100.0
        ethmin=100.0
        epimin=100.0
        pmin=100.0
        pmax=-100.0
c     ##################  CALL SET UP  #####################
```

```
      iopt=2
      n=nth+nth
      nprint=-1
c
      write(*,*)'TWO OPTIONS HERE:'
      write(*,*)'1. Enter pressure step which is added'
      write(*,*)'   to the prescribed pressure at every'
      write(*,*)'   node after every loop'
      write(*,*)'2. Enter 99 to increment pressure to final'
      write(*,*)'   prescribed pressure distribution linearly'
      read(*,*)presinc
c
      write(*,*)'START RUN FROM LAST STOP?? YES=1'
      read(*,*)lll
      if(lll.eq.1)then
       open(15,file='NDCONTINUE')
       do 888 i=1,nth,1
        read(15,*)y(i),y(i+nth)
  888  continue
       close(15)
      else
      endif
      write(*,*)'SAVE THE END DATA FOR A NEW RUN?? YES=1'
       read(*,*)kkl
      if(kkl.eq.1)then
       open(15,file='NDCONTINUE')
       open(13,file='NDSTART')
      else
      endif
c
      write(*,*)'ENTER THE NUMBER OF CALLS INTEGER'
      read(*,*)kkll
c
       pinc=1./kkll
       pfact=pinc
       if(presinc.eq.99)then
        do 57 i=1,nth,1
         pnow(i)=parnd(i)
         parnd(i)=pnow(i)*pfact
         pres(i)=parnd(i)
 57     continue
       else
       endif
c
      llkksave=kkll
      numb=kkll
      kdef=int(kkll/numb)
      lcount=kdef
      write(12,*)'ff=['
c
      do 25 j=1,kkll,1
       umax=0.0
       wmax=0.0
c
c         ibc=1=symmetry-top pinned-bottom
      if(ibc.eq.1)then
       call dnsqe(ndfcn,jac,iopt,n,y,fvec,tol,nprint,info,wa,lwa)
      else
      endif
c         ibc=2=pinned-top pinned-bottom
      if(ibc.eq.2)then
       call dnsqe(ndfcnpt,jac,iopt,n,y,fvec,tol,nprint,info,wa,lwa)
```

```
      else
      endif
c         ibc=3=infinite mass B.C.
      if(ibc.eq.3)then
       call dnsqe(ndfcnim,jac,iopt,n,y,fvec,tol,nprint,info,wa,lwa)
      else
      endif
c
      write(*,*)'info=',info,'   loop # ',j
c      SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES
      if((lcount.ge.kdef).or.j.eq.kk11)then
c
      call ndstress(x,y,hh,nu,ibc,epi,eth,
     +spi,sth,ethmax,epimax,ethmin,epimin,
     +sthmax,spimax,sthmin,spimin)
c
       do 1119 i=1,nth,1
        sxdef=dsin(x(i))+y(i)*dcos(x(i))-y(i+nth)*dsin(x(i))
        sydef=dcos(x(i))-y(i)*dsin(x(i))-y(i+nth)*dcos(x(i))
        write(12,1157)sxdef,sydef,y(i),y(i+nth),
     +pres(i),epi(i),eth(i),spi(i),sth(i)
        if(sydef.gt.sdefymax)sdefymax=sydef
        if(sxdef.gt.sdefxmax)sdefxmax=sxdef
        if(sydef.lt.sdefymin)sdefymin=sydef
        if(sxdef.lt.sdefxmin)sdefxmin=sxdef
        if(y(i).lt.sumin)sumin=y(i)
        if(y(i).gt.sumax)sumax=y(i)
        if(y(i+nth).lt.swmin)swmin=y(i+nth)
        if(y(i+nth).gt.swmax)swmax=y(i+nth)
        if(pres(i).lt.pmin)pmin=pres(i)
        if(pres(i).gt.pmax)pmax=pres(i)
 1119   continue
       llcount1=llcount1+1
       lcount=1
      else
       lcount=lcount+1
      endif
 1157 format(1x,e13.6,2x,e13.6,2x,e13.6,2x,e13.6,2x,
     +e13.6,2x,e13.6,2x,e13.6,2x,e13.6,2x,e13.6,2x)
c      SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPE
      write(*,4)
  4   format(1x,'node#',2x,'theta(deg)',8x,'u(i)/r',10x,
     1 'w(i)/r',3x,'p(i)*r/(el*h)')
      do 10 _=1,nth,1
       if(abs(y(i)).lt.1.0E-09)y(i)=0.0
       if(abs(y(i+nth)).lt.1.0E-09)y(i+nth)=0.0
       xx=x(i)*(180./pi)
       write(*,111)i,xx,y(i),y(i+nth),pres(i)
       if(abs(y(i)).gt.umax)umax=abs(y(i))
       if(abs(y(i+nth)).gt.wmax)wmax=abs(y(i+nth))
  10  continue
  11  format(1x,i4,2x,f8.3,2x,e14.5,2x,e14.5,2x,e14.5,2x,e14.5)
 111  format(1x,i4,4x,f8.3,3x,e14.5,4x,e14.5,4x,e14.5)
c
      if(j.ne.kk11)then
       if(presinc.eq.99)then
        pfact=pfact+pinc
        do 81 k=1,nth,1
         parnd(k)=pnow(k)*pfact
         pres(k)=parnd(k)
  81    continue
       else
```

```
          do 26 k=1,nth,1
           p(k)=p(k)+presinc
           parnd(k)=p(k)*r/(el*h)
           pres(k)=parnd(k)
  26      continue
         endif
        else
        endif
          if(j.eq.kkll.and.kkl.eq.1)then
           do 777 i=1,nth,1
           write(15,*)y(i),y(i+nth)
         write(13,89)y(i),y(i+nth)
  777      continue
  89      format(1x,e13.6,3x,e13.6)
          else
          endif
  25    continue
        write(12,*)'];'
c      ##########################################################
c      ###  next write final stress and strain to screen ######
       write(*,136)
  136   format(1x,'NODE #',4x,'THETA',4x,'EPSILON THETA',6x
     1   ,'EPSILON PHI',6x,'SIGMA THETA',9x,'SIGMA PHI')
        do 130 i=1,nth,1
          xx=x(i)*(180./pi)
          write(*,135)i,xx,eth(i),epi(i),sth(i),spi(i)
  130    continue
  135    format(1x,i4,3x,f6.2,4x,e13.6,4x,e13.6,4x,e13.6,5x,e13.6)
c        $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c         NEXT: write parameter matrix in MATLAB format
c         NOTE: Not all slots are used!!
  154    format(1x,e12.5)
  153    format(1x,i9)
         write(12,*)'hh=['
         write(12,153)nth
         write(12,153)ibc
         write(12,154)r
         write(12,154)em
         write(12,154)x(1)*(180./pi)
         write(12,154)x(nth)*(180./pi)
         write(12,154)beta
         write(12,153)ioptl
         write(12,154)0.
         write(12,154)0.
         write(12,154)0.
         write(12,154)h
         write(12,154)nu
         write(12,153)llcountl
         write(12,154)sumin
         write(12,154)sumax
         write(12,154)swmin
         write(12,154)swmax
         write(12,154)pmin
         write(12,154)pmax
         write(12,154)ethmin
         write(12,154)ethmax
         write(12,154)epimin
         write(12,154)epimax
         write(12,154)sthmin
         write(12,154)sthmax
         write(12,154)spimin
         write(12,154)spimax
```

```
      write(12,154)sdefxmin
      write(12,154)sdefxmax*1.05
      write(12,154)sdefymin
      write(12,154)sdefymax*1.05
      write(12,153)0.
      write(12,153)0.
      write(12,154)t
      write(12,154)t
      write(12,154)t
      write(12,153)llkksave
      write(12,153)ipress
      write(12,153)lptopt
      write(12,153)0.
      write(12,*)'];'
     stop
     end
c  ******************************************************
c      SUBROUTINE NDFCN.F
c  ******************************************************
      subroutine ndfcn(n,y,fvec,iflag)
      parameter (nth=20)
      implicit double precision(a-h,o-z)
      double precision y(nth+nth),fvec(nth+nth)
      real*8 a,b,c,co,d,nu,parnd(nth),pi
      real*8 x(nth),hh(nth),xmax
      integer i,n,iflag
      common/prop/ nu,parnd,x,hh,pi,xmax,xmin
c              BOUNDARY CONDITIONS
c      u(1)=y(1)=0.0,w(nth)=y(2*nth)=0.0,u(nth)=
c      y(nth)=0.0,slope of w(at1)=0.0
      fvec(1)=y(1)
      i=1
      co=0.0
      a=y(2)/hh(1)
      b=0.0
      c=0.0
      d=2.*(y(nth+2)-y(nth+1))/hh(1)**2.
c          next with limits:L'Hopitals Rule
      fvec(i+nth)=(a*d)-(y(i+nth)*d)+(a**2.)
     1 -(a*y(i+nth))+(3.*d*y(i)**2.)/(2.)
     2 +(3.*a*y(i)**2.)/(2.)+(a*d)-
     3 y(i+nth)*d+a**2.-(y(i+nth)*a)+a+a
     4 -2.*y(i+nth)+y(i)**2./(2.)+
     5 nu*(-(y(i+nth)*d)-(y(i)**2.)
     6 /(2.)-(y(i+nth)*a)+(a*d)+(2.*
     7 a**2.)+(a*d)-(y(i+nth)*d)-(y(i+nth)*a)
     8 +a+a-2.*
     9 y(i+nth))+parnd(i)
c
      fvec(nth)=y(nth)
      fvec(2*nth)=y(2*nth)
c     NEXT INTERIOR NODES
      do 10 i=2,nth-1,1
      co=dcos(x(i))/dsin(x(i))
      a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))
     1 +y(i)*(1./hh(i)-1./hh(i+1))+
     2 y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
      b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
     1 2.*y(i)/(hh(i)*hh(i+1))+
     2 y(i+1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
      c=-y(i-1+nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)
     1 +y(i+nth)*(hh(i)**-1.-
```

```
     2  hh(i+1)**-1.)+y(i+1+nth)*((hh(i)/hh(i+1))*(hh(i)
     3  +hh(i+1))**-1.)
      d=y(i-1+nth)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.
     1  -2.*y(i+nth)/(hh(i)*
     2  hh(i+1))+y(i+1+nth)*2.*(hh(i+1)**-1.)*((hh(i)
     3  +hh(i+1))**-1.)
c     next equation is 9a
      fvec(i)=b-c+c*d+(y(i))*d
     1  +(y(i+nth))*c+(y(i+nth)*y(i))-
     2  (1/(2.))*c**3.-(3.*y(i)/(2.))*c**2.-
     3  ((3.*y(i)**2.)/(2.))*c-y(i)**3.
     4  /(2.)+a*co-y(i)*co**2.+(co/(2.))*c**2.+((y(i)*
     5  co))*c+((y(i)**2.)*co)/(2.)+nu*(-y(i)
     6  -c-(2.*y(i)*co*c)
     7  +(y(i+nth)*c)+(y(i)*y(i+nth))
     8  -(co*c**2.)/(2.)-(3.*(y(i)**2.)*co)
     9  /(2.))
c      next equation is 9b
      fvec(i+nth)=(co*a*c)-(co*y(i+nth)*c)
     1  +(co*c**3.)/(2.)+(3.*co*y(i)
     2  *c**2.)/(2.)+(3.*co*c*y(i)**2.)/(2.)+(y(i)*co*a)
     3  -(co*y(i)*y(i+nth))+(co*y(i)**3.)/(2.)
     4  +(b*c)-c**2./
     5  (2.)+(3.*d*c**2.)/(2.)+(2.*y(i)*d*c)+(3.*a*c**
     6  2.)/(2.)+(2.*y(i)*a*c)+(y(i)*b)-(y(i)*c)+(3.
     7  *d*y(i)**2.)/(2.)+(3.*a*y(i)**2.)/(2.)+(a*d)-(
     8  y(i+nth)*d)+a**2.-(y(i+nth)*a)+a+y(i)*co
     9  -2.*y(i+nth)+(y(i)*c)+y(
     1  i)**2./(2.)+nu*((-y(i)*c)-(y(i+nth)*co*c)-(y(i)**2.)
     2  /(2.)-(y(i+nth)*y(i)*co)+(co*a*c)-(c**2.)/(2.)+(2.*
     3  y(i)*co*a)+(y(i)*co*d)-(y(i+nth)*d)-(y(i+nth)*a)
     4  +a+y(i)*co-2.*
     5  y(i+nth))+parnd(i)
 10   continue
      return
      end
C   ****************************************************
C     SUBROUTINE NDFCNPT.F
C   ****************************************************
      subroutine ndfcnpt(n,y,fvec,iflag)
      parameter (nth=20)
      implicit double precision(a-h,o-z)
      double precision y(nth+nth),fvec(nth+nth)
      real*8 a,b,c,co,d,nu,parnd(nth),pi
      real*8 x(nth),hh(nth),xmax
      integer i,n,iflag
      common/prop/ nu,parnd,x,hh,pi,xmax,xmin
c          B.C. for pinned both sides
      fvec(1)=y(1)
      fvec(1+nth)=y(1+nth)
      fvec(nth)=y(nth)
      fvec(2*nth)=y(2*nth)
c     NEXT INTERIOR NODES
      do 10 i=2,nth-1,1
      co=dcos(x(i))/dsin(x(i))
      a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))
     1  +y(i)*(1./hh(i)-1./hh(i+1))+
     2  y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
      b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
     1  2.*y(i)/(hh(i)*hh(i+1))+
     2  y(i+1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
      c=-y(i-1+nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)
```

```
     1  +y(i+nth)*(hh(i)**-1.-
     2  hh(i+1)**-1.)+y(i+1+nth)*((hh(i)/hh(i+1))*(hh(i)
     3  +hh(i+1))**-1.)
      d=y(i-1+nth)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.
     1  -2.*y(i+nth)/(hh(i)*
     2  hh(i+1))+y(i+1+nth)*2.*(hh(i+1)**-1.)*((hh(i)
     3  +hh(i+1))**-1.)
c     next equation is 9a
      fvec(i)=b-c+c*d+(y(i))*d
     1  +(y(i+nth))*c+(y(i+nth)*y(i))-
     2  (1/(2.))*c**3.-(3.*y(i)/(2.))*c**2.-
     3  ((3.*y(i)**2.)/(2.))*c-y(i)**3.
     4  /(2.)+a*co-y(i)*co**2.+(co/(2.))*c**2.+((y(i)*
     5  co))*c+((y(i)**2.)*co)/(2.)+nu*(-y(i)
     6  -c-(2.*y(i)*co*c)
     7  +(y(i+nth)*c)+(y(i)*y(i+nth))
     8  -(co*c**2.)/(2.)-(3.*(y(i)**2.)*co)
     9  /(2.))
c      next equation is 9b
      fvec(i+nth)=(co*a*c)-(co*y(i+nth)*c)
     1  +(co*c**3.)/(2.)+(3.*co*y(i)
     2  *c**2.)/(2.)+(3.*co*c*y(i)**2.)/(2.)+(y(i)*co*a)
     3  -(co*y(i)*y(i+nth))+(co*y(i)**3.)/(2.)
     4  +(b*c)-c**2./
     5  (2.)+(3.*d*c**2.)/(2.)+(2.*y(i)*d*c)+(3.*a*c**
     6  2.)/(2.)+(2.*y(i)*a*c)+(y(i)*b)-(y(i)*c)+(3.
     7  *d*y(i)**2.)/(2.)+(3.*a*y(i)**2.)/(2.)+(a*d)-(
     8  y(i+nth)*d)+a**2.-(y(i+nth)*a)+a+y(i)*co
     9  -2.*y(i+nth)+(y(i)*c)+y(
     1  i)**2./(2.)+nu*((-y(i)*c)-(y(i+nth)*co*c)-(y(i)**2.)
     8  /(2.)-(y(i+nth)*y(i)*co)+(co*a*c)-(c**2.)/(2.)+(2.*
     2  y(i)*co*a)+(y(i)*co*d)-(y(i+nth)*d)-(y(i+nth)*a)
     3  +a+y(i)*co-2.*
     4  y(i+nth))+parnd(i)
 10   continue
       return
       end
c     ***************************************************
c      SUBROUTINE NDFCNIM.F
c     ***************************************************
       subroutine ndfcnim(n,y,fvec,iflag)
       parameter (nth=20)
       implicit double precision(a-h,o-z)
       double precision y(nth+nth),fvec(nth+nth)
       real*8 a,b,c,co,d,nu,parnd(nth),pi
       real*8 x(nth),hh(nth),xmax
       real*8 theta1,theta2,theta3,xnot,ynot
       real*8 xnotm1,ynotm1,xnotm2,ynotm2,ybelow
       real*8 radius,xnth,xnthm1,xnthm2,ynth
       real*8 ynthm1,ynthm2,hi,hip1,slope
       integer i,n,iflag
       common/prop/ nu,parnd,x,hh,pi,xmax,xmin
c               BOUNDARY CONDITIONS
c       u(1)=y(1)=0.0,w(nth)=y(2*nth)=0.0,u(nth)=
c       y(nth)=0.0,slope of w(at1)=0.0
       fvec(1)=y(1)
       i=1
       co=0.0
       a=y(2)/hh(1)
       b=0.0
       c=0.0
       d=2.*(y(nth+2)-y(nth+1))/hh(1)**2.
```

```fortran
c            next with limits:L'Hopitals Rule
      fvec(i+nth)=(a*d)-(y(i+nth)*d)
     1  +(co*c**3.)/(2.)+(3.*co*y(i)
     2  *c**2.)/(2.)+(3.*co*c*y(i)**2.)/(2.)+(a**2.)
     3  -(a*y(i+nth))+(co*y(i)**3.)/(2.)
     4  +(b*c)-c**2./
     5  (2.)+(3.*d*c**2.)/(2.)+(2.*y(i)*d*c)+(3.*a*c**
     6  2.)/(2.)+(2.*y(i)*a*c)+(y(i)*b)-(y(i)*c)+(3.
     7  *d*y(i)**2.)/(2.)+(3.*a*y(i)**2.)/(2.)+(a*d)-(
     8  y(i+nth)*d)+a**2.-(y(i+nth)*a)+a+a
     9  -2.*y(i+nth)+(y(i)*c)+y(
     1  i)**2./(2.)+nu*((-y(i)*c)-(y(i+nth)*d)-(y(i)**2.)
     2  /(2.)-(y(i+nth)*a)+(a*d)-(c**2.)/(2.)+(2.*
     3  a**2.)+(a*d)-(y(i+nth)*d)-(y(i+nth)*a)
     4  +a+a-2.*
     5  y(i+nth))+parnd(i)
c        next B.C. at edge
       if(xmax.le.90.0)write(*,*)'xmax is to small for I.M. B.C.'
c      first establish constants
         theta1=(xmax-90.0)*(pi/180.)
         theta2=theta1-hh(nth)
         theta3=theta2-hh(nth-1)
         xnot=dcos(theta1)
         ynot=dsin(theta1)
         xnotm1=dcos(theta2)
         ynotm1=dsin(theta2)
         xnotm2=dcos(theta3)
         ynotm2=dsin(theta3)
         ybelow=-1./ynot
         radius=(xnot/ynot)
c
         xnth=(1.-y(2.*nth))*xnot-y(nth)*ynot
         ynth=(-1.+y(2.*nth))*ynot-y(nth)*xnot
         xnthm1=(1.-y(2.*nth-1))*xnotm1-y(nth-1)*ynotm1
         ynthm1=(-1.+y(2.*nth-1))*ynotm1-y(nth-1)*xnotm1
         xnthm2=(1.-y(2.*nth-2))*xnotm2-y(nth-2)*ynotm2
         ynthm2=(-1.+y(2.*nth-2))*ynotm2-y(nth-2)*xnotm2
c
         hi=xnthm1-xnth
         hip1=xnthm2-xnthm1
         slope=-ynth*((2*hi+hip1)/(hi*(hi+hip1)))+
     1         ynthm1*((hi+hip1)/(hi*hip1))-
     2         ynthm2*(hi/(hip1*(hi+hip1)))
c  NEXT:Node nth must travel in a circle of radius="radius"
         fvec(nth)=xnth**2.+(ynth-ybelow)**2.-radius**2.
c  NEXT:Slope at node nth based on three nodes must=slope of "line"
         fvec(2.*nth)=((radius**2.-xnth**2.)**0.5)/xnth-slope
c       NEXT INTERIOR NODES
        do 10 i=2,nth-1,1
        co=dcos(x(i))/dsin(x(i))
       a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))
     1  +y(i)*(1./hh(i)-1./hh(i+1))+
     2  y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
      b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
     1  2.*y(i)/(hh(i)*hh(i+1))+
     2  y(i+1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
      c=-y(i-1+nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)
     1  +y(i+nth)*(hh(i)**-1.-
     2  hh(i+1)**-1.)+y(i+1+nth)*((hh(i)/hh(i+1))*(hh(i)
     3  +hh(i+1))**-1.)
      d=y(i-1+nth)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.
     1  -2.*y(i+nth)/(hh(i)*
```

```
     2  hh(i+1))+y(i+1+nth)*2.*(hh(i+1)**-1.)*((hh(i)
     3  +hh(i+1))**-1.)
c       next equation is 9a
        fvec(i)=b-c+c*d+(y(i))*d
     1  +(y(i+nth))*c+(y(i+nth)*y(i))-
     2  (1/(2.))*c**3.-(3.*y(i)/(2.))*c**2.-
     3  ((3.*y(i)**2.)/(2.))*c-y(i)**3.
     4  /(2.)+a*co-y(i)*co**2.+(co/(2.))*c**2.+((y(i)*
     5  co))*c+((y(i)**2.)*co)/(2.)+nu*(-y(i)
     6  -c-(2.*y(i)*co*c)
     7  +(y(i+nth)*c)+(y(i)*y(i+nth))
     8  -(co*c**2.)/(2.)-(3.*(y(i)**2.)*co)
     9  /(2.))
c         next equation is 9b
        fvec(i+nth)=(co*a*c)-(co*y(i+nth)*c)
     1  +(co*c**3.)/(2.)+(3.*co*y(i)
     2  *c**2.)/(2.)+(3.*co*c*y(i)**2.)/(2.)+(y(i)*co*a)
     3  -(co*y(i)*y(i+nth))+(co*y(i)**3.)/(2.)
     4  +(b*c)-c**2./
     5  (2.)+(3.*d*c**2.)/(2.)+(2.*y(i)*d*c)+(3.*a*c**
     6  2.)/(2.)+(2.*y(i)*a*c)+(y(i)*b)-(y(i)*c)+(3.
     7  *d*y(i)**2.)/(2.)+(3.*a*y(i)**2.)/(2.)+(a*d)-(
     8  y(i+nth)*d)+a**2.-(y(i+nth)*a)+a+y(i)*co
     9  -2.*y(i+nth)+(y(i)*c)+y(
     1  i)**2./(2.)+nu*((-y(i)*c)-(y(i+nth)*co*c)-(y(i)**2.)
     2  /(2.)-(y(i+nth)*y(i)*co)+(co*a*c)-(c**2.)/(2.)+(2.*
     3  y(i)*co*a)+(y(i)*co*d)-(y(i+nth)*d)-(y(i+nth)*a)
     4  +a+y(i)*co-2.*
     5  y(i+nth))+parnd(i)
 10     continue
          return
          end
c     ***************************************************
c         SUBROUTINE NDSTRESS.F
c     ***************************************************
         subroutine ndstress(x,y,hh,nu,ibc,epi,eth,
        1spi,sth,ethmax,epimax,ethmin,epimin,
        2sthmax,spimax,sthmin,spimin)
          parameter (nth=20)
          implicit double precision (a-h,o-z)
          real*8 a,c
          real*8 epi(nth),eth(nth),spi(nth),sth(nth)
          real*8 hh(nth),nu,x(nth)
          real*8 sthmax,spimax,sthmin,spimin
          real*8 ethmax,epimax,ethmin,epimin
          dimension y(nth+nth)
          integer i,ibc
c         first for node # 1
          if(ibc.ne.1.and.ibc.ne.3)then
           a=-y(1)*((2*hh(2)+hh(3))/(hh(2)*(hh(2)+hh(3))))+
        1  y(2)*((hh(2)+hh(3))/(hh(2)*hh(3)))-y(3)
        2  *((hh(2))/(hh(3)*(hh(2)+hh(3))))
           c=-y(1+nth)*((2.*hh(2)+hh(3))/(hh(2)*(hh(2)+hh(3))))+
        1  y(2+nth)*((hh(2)+hh(3))/(hh(2)*hh(3)))-
        2  y(3+nth)*(hh(2)/(hh(3)*(hh(2)+hh(3))))
           eth(1)=(a-y(1+nth))+(0.5)*(c+y(1))**2.
           epi(1)=y(1)*(dcos(x(1))/dsin(x(1)))-y(1+nth)
           sth(1)=eth(1)+nu*epi(1)
           spi(1)=epi(1)+nu*eth(1)
          else
           a=y(2)/hh(1)
           eth(1)=a-y(1+nth)+(0.5)*(y(1))**2.
```

```
      epi(1)=a-y(1+nth)
      sth(1)=eth(1)+nu*epi(1)
      spi(1)=epi(1)+nu*eth(1)
     endif
     do 30 i=2,nth-1,1
    a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))+
   1  y(i)*(1./hh(i)-1./hh(i+1))+
   2  y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
    c=-y(i-1+nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)+
   1  y(i+nth)*(hh(i)**-1.-hh(i+1)**-1.)+
   2  y(i+1+nth)*((hh(i)/hh(i+1))*(hh(i)+hh(i+1))**-1.)
     eth(i)=a-y(i+nth)+(0.5)*(c+y(i))**2.
     epi(i)=y(i)*(dcos(x(i))/dsin(x(i)))-y(i+nth)
     sth(i)=eth(i)+nu*epi(i)
     spi(i)=epi(i)+nu*eth(i)
 30  continue
c     next for node # nth
     a=y(nth-2)*(hh(nth)/(hh(nth-1)*(hh(nth)+hh(nth-1))))-
   1y(nth-1)*((hh(nth)+hh(nth-1))/(hh(nth)*hh(nth-1)))+y(nth)
   2*((2*hh(nth)+hh(nth-1))/(hh(nth)*(hh(nth-1)+hh(nth))))
     c=y(2*nth-2)*(hh(nth)/(hh(nth-1)*(hh(nth)+hh(nth-1))))-
   1y(2*nth-1)*((hh(nth)+hh(nth-1))/(hh(nth)*hh(nth-1)))+
   2y(2*nth)*((2*hh(nth)+hh(nth-1))/(hh(nth)*(hh(nth-1)
   3+hh(nth))))
     eth(nth)=a-y(2*nth)+(0.5)*(c+y(nth))**2.
     epi(nth)=y(nth)*(dcos(x(nth))/dsin(x(nth)))-y(2*nth)
     sth(nth)=eth(nth)+nu*epi(nth)
     spi(nth)=epi(nth)+nu*eth(nth)
     do 40 i=1,nth,1
      if(sth(i).gt.sthmax)sthmax=sth(i)
      if(spi(i).gt.spimax)spimax=spi(i)
      if(sth(i).lt.sthmin)sthmin=sth(i)
      if(spi(i).lt.spimin)spimin=spi(i)
      if(eth(i).gt.ethmax)ethmax=eth(i)
      if(epi(i).gt.epimax)epimax=epi(i)
      if(eth(i).lt.ethmin)ethmin=eth(i)
      if(epi(i).lt.epimin)epimin=epi(i)
 40  continue
     return
     end
```

Appendix I. Dynamic Fortran Programs

```
c     ****************************************************
c          DYNAMIC PROGRAM AND ATTACHED SUBROUTINES
c     ****************************************************
c
c       *************** NONDIMENSIONAL VERSION **************
c       FORTRAN PROGRAM to solve dynamic version of nonlinear
c       membrane equations see "Nonlinear
c       Elasticity" written by J.J. Stoker 1968 page 32.
c       The equations are solved by calling the subroutine
c       DDEBDF (SLATEC LIBRARY) etc. which solve a system of
c       first order differential equations. The program
c       DDEBDF.F calls one of the following subroutines
c       depending on the prescribed boundary conditions
c       dampdf.f, dampdfl.f, dampdfpt.f, or dampdfim.f
c       (ibc=1,2,3 or 4).
c       These subroutines are (user supplied) and are
c       located in this directory. Theses subroutines
c       call the functions dampea.f,dampeb.f and dampfcn.f in
c       this directory. The program can handle a variable
c       grid and has 3 clustering options. Parameters are
c       input for each run. The number of nodes used to solve
c       the problem can be changed by changing the
c       parameter statement for (nth) in this program
c       and in dampdf.f , dampdfl.f ,dampdfpt.f,dampdfim.f,
c       dampfcn.f, and dampstress.f. Dampstress.f is used to
c       calculate stresses and strains at various times.
c
c       NOTE: The parameters are entered in dimensional form
c       but output and internal computations are done in
c       nondimensional form. Also, three pressure
c       distributions as a function of theta are included.
c       These are (1) linear (2) parabolic (3) user defined
c       at each node. The pressure can be changed as a function
c       of time three separate ways. (1) linearly from p=0.0
c       to pfinal (2) linearly from pstart to p=0.0
c       (3) same as one with a time option which sets
c       all p(i)=constant
c
c       OUTPUT: The program generates "THREE" matrices aa bb
c               and cc. aa and cc are saved in
c               the file AAMatrix.m. This file can be
c               loaded into the MATLAB software. The bb matrix
c               is saved in bb.mat and can be loaded in
c               MATLAB with the command "load bb"
c               The MATLAB program DYNAMIC.m is used to
c               view the results of a run.
c
        parameter (nth=20)
        parameter (lrw=250+10*4*nth+16.*nth*nth)
        parameter (liw=55+4.*nth)
c
        implicit double precision (a-h,o-z)
        real*8 atol,beta,cc,cu,cw,dd
        real*8 el,em,epi(nth),eth(nth),gamma
        real*8 h,hh(nth),nu,parnd(nth),pi
        real*8 pf(nth),p(nth),r,rho,rtol,rwork(lrw)
        real*8 spi(nth),spimax,sth(nth),sthmax,tstep,tfinal
        real*? sumax,swmax,xmax,xmin,x(nth),xx(3),xx1,zz
```

```
      real*8 y1,y2,y3,y4,y5,y6,y7,y8,xend
      real*8 tin,tout,toutin
      dimension rpar(500),y(4*nth)
      dimension yt(4*nth-4)
      integer cont,i,ibc,idid,info(20),iopt,ipar(nth)
      integer iwork(liw),j,k,neq
c
      common/prop1/ x,y1,y2,y3,y4,y5,y6,y7,y8,xend
c
      external dampdf,dampdf1,dampdfpt
      external dampfcn,dampdfim
c      LIST OF FILES/PROGRAMS CALLED OR OPENED ***********
c      external=dampdf.f (for ibc=1)
c      external=dampdf1.f (for ibc=2)
c      external=dampdfpt.f (for ibc=3)
c      external=dampdfim.f (for ibc=4)
c         called from above two programs
c               function=dampea.f
c               function=dampeb.f
c      external=dampfcn.f (for ibc=4)
c      opened=NDSTART
c      opened=AAMatrix.m
c      opened=bb.mat
c      called=ddebdf.f (equation solver)
c      called=dampstress.f calculates stresses and strains
c      *************************  DEFINITIONS  ***********
c      aparab=constant used to generate a parabolic
c             pressure distribution ipress=2
c      atol  ="absolute" error tolerance used in DDEBDF.F
c      beta  =parameter used to generate a clustered grid
c             pos. beta>1.0
c      binter=intercept value used to define linear pressure
c             distribution ipress=1
c      bparab=constant used to generate a parabolic pressure
c             distribution ipress=2
c      cc    =used as variable in grid generation
c      cont  =used to control output to screen
c      cparab=constant used to generate a parabolic pressure
c             distribution ipress=2
c      cu    =damping ratio in u-direction
c      cw    =damping ratio in w-direction
c      dampdf=External subroutine
c      dampdfim=External subroutine
c      dampdf1=External subroutine
c      dampdfpt=External subroutine
c      dampea=External function
c      dampeb=External function
c      dampstress=External subroutine
c      dd    =used as variable in grid generation
c      deltat=tout-tin  used to determine u and w vel at
c             node nth when using i.m. b.c.
c      e1    =Youngs Modulus divided by
c             (one minus Poisson's ratio squared)
c      em    =Youngs Modulus
c      epi(i)=strain "epsilon phi" at node "i"
c      eth(i)=strain "epsilon theta" at node "i"
c      epimax=max. value of epi for run
c      epimin=min. value of epi for run
c      ethmax=max. value of eth for run
c      ethmin=min. value of eth for run
c      gamma =used in grid generation for iopt=4
c      h     =membrane thickness (constant)
```

```
c       hh(i) =the difference in radians from theta at
c              node "i" to theta at node "i-1"
c       i,j,k,m=integer counter for various loops
c       ibc    =parameter which defines which B.C. to use
c       idid   =integer value sent back by DDEBDF.F to
c              report status of run
c       info(i)=input parameters used in DDEBDF.F
c       iopt   =variable grid options four types
c       ipress=pressure vs. theta options three types
c       iwork =integer work array used by DDEBDF.F
c       jkl    =integer = 1/2 of nth (used in grid
c              generation for iopt=4)
c       kdef   =value used to determine when to print
c              to b[] matrix
c       lcount=counter used with kdef to print output
c              to b[]matrix
c       liw    =dimension of iwork used by DDEBDF.F
c       llcount1=number of sets of data saved in bb[] matrix
c       llkk   =user input for total # of calls to DDEBDF.F
c       llkksave=llkk sent to defsave.m
c       lpcount=counter used to determine when to print to
c               screen every lpcounter loops
c       lpvstime=parameter for pressure vs. time
c                (if=1 then pressure changes with time)
c       lptopt=three pressure vs time options
c       lrw    =dimension of rwork array used by DDEBDF.F
c       NDSTART=files created by ndstatic.f contains two
c                 columns by nth rows (init. def.)
c       neq    =number of equations to be solved by DDEBDF.F
c       nth    =total number of nodes on the membrane
c       nu     =Poissons ratio
c       numb   =# of "sets" of output to be saved
c       parnd(i)=non-dimensional pressure=p(i)*r/(el*h)
c       pi     =double precision value for pi=3.14.....
c       p(i)   =pressure, a function of theta and time psi
c       pf(i) =pressure final when lpvstime=1
c       pfin   =final "constant vs theta" pressure
c              used when lptopt=3
c       pfthree=time for which p(i)=0.0 when lpopt=3
c       pft=time input used in lptopt options
c       pmax   =used to define pressure lin. and para.
c              also used to save pmaximum
c       pmin   =used to define pressure lin. and para.
c              also used to save pminimum
c       ppa=parameter used to change pressure each time step
c       ppstep= parameter used to change pparam each time step
c       pslope=slope of press. vs. theta for linear
c              pressure distribution ipress=1
c       py1,px1 etc. =used to generate parabola
c                   equation when ipress=2
c       r      =undeformed radius of sphere in inches
c       rho    =density of material (lbf*sec**2)/in**4
c       rpar(i)=used to pass info to and from DDEBDF.F
c       rtol   =tolerence parameter used by DDEBDF "relative error"
c       rwork(lrw)=work array used in DDEBDF.F
c       sdefxmin=next four min and max values of deformed sphere
c       sdefymin=
c       sdefxmax=
c       sdefymax=
c       sumin=min. value of u-deflections for the entire run
c       sumax=max. "       "        "
c       swmin=min.  "      "  w-def.  "
```

```
c       swmax=max.  "
c       sth(i)=sigma theta at node i
c       spi(i)=sigma phi at node i
c       spimax=max. value of spi for run
c       spimin=min. value of spi for run
c       sthmax=max. value of sth for run
c       sthmin=min. value of sth for run
c       t     =current time ND
c       test3 =used when lpvstime=1 to determine if
c              a restart is needed info(1)=0
c       tester=used when lpvstime=1 to determine if
c              a restart is needed info(1)=0
c       tfinal=time at which run stops if larger
c              than time from loop 100
c       tfinish=dimensional time used for aa[]matrix
c       tin   =time used to start call
c       tout  =current attempted output time ND
c       toutin=time we want to reach during this
c              run through loop
c       toutsave=saves original value of tout for aa[]matrix
c       tstart=saves starting time of run
c       tstep =value added to tout for each call to DDERKF
c       wmax  =
c       x(i)  =value of theta at node "i" (theta=0.0 at top
c              of sphere) max value is xmax
c       xmax  =maximum value of theta in ****degrees****
c       xmin  =value of theta at top of sphere annular
c              case in degrees
c       xoldu =value of u-def at node nth tstep before
c              present (used with i.m. b.c.)
c       xoldw =value of w-def at node nth tstep before
c              present (used with i.m. b.c.)
c       y(i)  =u-def. for 1 to nth,u-vel. for nth+1
c              to 2*nth, etc.
c       yt(i) =used in i.m. b.c. u-def. for 1 to
c              nth-1,u-vel. for nth to 2*nth-1, etc.
c       zbot  =used in parabolic pressure distribution ipress=2
c       ztop  =used in parabolic pressure distribution ipress=2
c       zz    =used as variable in grid generation
c      *******************************************************
c      ***********  NEXT: BOUNDARY CONDITIONS ****************
c      *******************************************************
c      ibc used to call the correct subroutine for the required
c           boundary conditions
c      ibc=1 is for pinned boundary conditions at node
c          nth (calls dampdf.f)
c      ibc=2 is for linear case (calls dampdfl.f)
c      ibc=3 is for pinned b.c. at "top" and "bottom"
c          nodes (calls dampdfpt.f)
c      ibc=4 is for infinite mass b.c. (not annular) calls
c          dampdfim.f which calls dampfcn.f
c      ibc=5 is for FUTURE B.C.
       write(*,*)'CHOOSE BOUNDARY CONDITION??'
       write(*,*)'ENTER 1 = pinned boundary at node nth'
       write(*,*)'ENTER 2 = linear case boundary conditions'
       write(*,*)'ENTER 3 = pinned b.c. at top and bottom annular'
       write(*,*)'ENTER 4 = infinite mass B.C.'
       write(*,*)'ENTER 5 = FUTURE!!!'
       read(*,*)ibc
c      **********************************************************
c      ***************  MATERIALS AND DIMENSIONS  ***************
c      **** NOTE: parameters are input in dimensional form  *****
```

```
c     ****************************************************************
      write(*,*)'ENTER THE PARAMETERS,  "0" FOR (**) DEFAULT'
      write(*,*)
      write(*,*)'ENTER THE RADIUS IN INCHES (300)'
      read(*,*)r
      if(r.eq.0.0)r=300.0
      write(*,*)'ENTER NU (0.3)'
      read(*,*)nu
      if(nu.eq.0.0)nu=0.3
      write(*,*)'ENTER YOUNGS MOD. psi  (30000.0)'
      read(*,*)em
      if(em.eq.0.0)em=30000.0
      el=em/(1.-nu**2.)
      if(ibc.eq.3)then
       write(*,*)'ENTER XMIN THETAMIN AT TOP OF SPHERE (DEG) (0)'
       read(*,*)xmin
      else
       xmin=0.0
      endif
      write(*,*)'ENTER XMAX THETAMAX AT EDGE OF SPHERE (DEG) (90)'
      read(*,*)xmax
       if(xmax.eq.0.0)xmax=90.0
       write(*,*)'ENTER THE MEMBRANE THICKNESS INCHES (0.005)'
       read(*,*)h
       if(h.eq.0.0)h=0.005
       write(*,*)'ENTER THE MASS DENSITY OF THE MEMBRANE (9.0E-06)'
       read(*,*)rho
       if(rho.eq.0.0)rho=9.0E-06
       write(*,*)'ENTER THE U-DAMPING RATIO '
       read(*,*)cu
       rpar(6)=cu
       write(*,*)'ENTER THE W-DAMPING RATIO '
       read(*,*)cw
       rpar(7)=cw
       pi=4.*DATAN(1.D00)
c     ****************************************************************
c     GRIDS...GRIDS...GRIDS...GRIDS...GRIDS...GRIDS...GRIDS...
       write(*,12)
  12   format(1x,'***********************************************')
       write(*,*)'THERE ARE FOUR TYPES OF GRID SPACING OPTIONS:'
       write(*,*)'ENTER 1= uniform spacing'
       write(*,*)'ENTER 2= clustering at peak node (node=1)'
       write(*,*)'ENTER 3= clustering at edge node (node=nth)'
       write(*,*)'ENTER 4= clustering at both ends'
       read(*,*)iopt
       hh(1)=(xmax-xmin)*(pi/180.)/(nth-1)
c
       if(iopt.eq.1)then
        x(1)=xmin*(pi/180.)
        hh(1)=(xmax-xmin)*(pi/180.)/(nth-1)
        do 5 i=2,nth,1
         hh(i)=hh(1)
         x(i)=x(i-1)+hh(i)
  5     continue
       else
       endif
c
       if(iopt.eq.2)then
        write(*,*)'ENTER BETA more clustering as beta gos to 1'
        read(*,*)beta
        zz=0.0
        do 8 i=nth,1,-1
```

```
          gamma=zz/((xmax-xmin)*(pi/180.))
          dd=((beta+1.)/(beta-1.))**gamma
          cc=1.-beta*(dd-1.)/(dd+1.)
          x(i)=cc*(xmax-xmin)*(pi/180.)
          zz=zz+hh(1)
          x(i)=xmin*(pi/180.)+x(i)
          write(*,*)i,x(i)*(180./pi)
 8       continue
        else
        endif
c
        if(iopt.eq.3)then
         write(*,*)'ENTER BETA more clustering as beta gos to 1'
         read(*,*)beta
         zz=0.
         do 9 i=1,nth,1
          gamma=zz/((xmax-xmin)*(pi/180.))
          dd=((beta+1.)/(beta-1.))**gamma
          cc=beta*(dd-1.)/(dd+1.)
          x(i)=cc*(xmax-xmin)*(pi/180.)
          zz=zz+hh(1)
          x(i)=xmin*(pi/180.)+x(i)
          write(*,*)i,x(i)*(180./pi)
 9       continue
        else
        endif
c
        if(iopt.eq.4)then
         write(*,*)'ENTER BETA more clustering as beta gos to 1'
         read(*,*)beta
         write(*,*)'ENTER JKL =1/2 of nth nth=',nth
         read(*,*)jkl
         hh(1)=(xmax-xmin)*(pi/180.)/(nth-1)
         zz=0.0
         do 119 i=jkl+1,nth,1
          gamma=zz/(0.5*(xmax-xmin)*(pi/180.)-hh(1)/2.)
          dd=((beta+1.)/(beta-1.))**gamma
          cc=beta*(dd-1.)/(dd+1.)
          x(i)=cc*(0.5*(xmax-xmin)*(pi/180.)-hh(1)/2.)
          zz=zz+hh(1)
          x(nth-i+1)=((xmax-xmin)/2.0+xmin)*(pi/180.)-hh(1)/2.-x(i)
          x(i)=((xmax-xmin)/2.0+xmin)*(pi/180.)+hh(1)/2.+x(i)
 119     continue
         do 219 i=1,nth,1
          write(*,*)i,x(i)*(180./pi)
 219     continue
        else
        endif
c        next: establish grid spacing
         do 6 i=2,nth,1
          hh(i)=x(i)-x(i-1)
 6       continue
        hh(1)=hh(2)
c       GRIDS...GRIDS...GRIDS...GRIDS...GRIDS...GRIDS...GRIDS
c       PVSTHETA...PVSTHETA...PVSTHETA...PVSTHETA...PVSTHETA
        write(*,*)'*****  PRESSURE SECTION  *****'
        write(*,*)'ENTER the type of pressure distribution'
        write(*,*)'   1=linear pressure distribution'
        write(*,*)'   2=parabolic pressure distribution'
        write(*,*)'   3=YOU DEFINE THE PRESSURE AT EACH NODE'
        read(*,*)ipress
c
```

```
        if(ipress.eq.1)then
         write(*,*)'Enter the value of the pressure at theta=xmin'
         read(*,*)pmin
         write(*,*)'Enter the value of the pressure at theta=xmax'
         read(*,*)pmax
         pslope=(pmax-pmin)/((xmax-xmin)*(pi/180.))
         binter=pmin-pslope*xmin*(pi/180.)
         do 678 i=1,nth,1
          p(i)=pslope*x(i)+binter
          parnd(i)=p(i)*r/(el*h)
  678    continue
        else
        endif
c
        if(ipress.eq.2)then
         write(*,*)'Enter the value of the pressure at theta=xmin'
         read(*,*)pmin
         write(*,*)'Enter the value of the pressure at theta=xmax'
         read(*,*)pmax
         write(*,*)'Enter the value of theta for the third point'
         read(*,*)thetamid
         write(*,*)'Enter the value of the pressure at third point'
         read(*,*)pmid
         py1=pmin
         py2=pmid
         py3=pmax
         px1=xmin*(pi/180.)
         px2=thetamid*(pi/180.)
         px3=xmax*(pi/180.)
         ztop=(py1-py3)*(px2-px3)-(py2-py3)*(px1-px3)
         zbot=(px1**2.-px3**2.)*(px2-px3)-
     1        (px2**2.-px3**2.)*(px1-px3)
         aparab=ztop/zbot
         bparab=(aparab*(px3**2.-px2**2.)+py2-py3)/(px2-px3)
         cparab=py3-aparab*px3**2.-bparab*px3
         do 679 i=1,nth,1
          p(i)=aparab*x(i)**2.+bparab*x(i)+cparab
          parnd(i)=p(i)*r/(el*h)
  679    continue
        else
        endif
c
        if(ipress.eq.3)then
         write(*,*)'Enter the pressure at each node when asked'
         do 677 i=1,nth,1
          write(*,*)'ENTER the pressure at node # ',i
          read(*,*)p(i)
          parnd(i)=p(i)*r/(el*h)
  677    continue
        else
        endif
c      PVSTHETA...PVSTHETA...PVSTHETA...PVSTHETA...PVSTHETA
c      PVSTIME....PVSTIME....PVSTIME....PVSTIME....PVSTIME
      write(*,12)
      write(*,*)'DO YOU WANT THE PRESSURE TO CHANGE WITH TIME???'
      write(*,*)'IF YES ENTER  1     IF NO ENTER ANY OTHER #'
      read(*,*)lpvstime
      if(lpvstime.eq.1)then
       write(*,*)'THE FOLLOWING OPTIONS ARE AVAILABLE'
       write(*,12)
       write(*,*)'1=START with all p(i)=0.0 and linearly increase'
       write(*,*)'the pressure values to p-final at user defined t'
```

```
      write(*,*)
      write(*,*)'2=START from p(i)=given above and decrease the'
      write(*,*)'pressure to p(i)=zero at user defined time'
      write(*,*)
      write(*,*)'3=same as #1 but all p(i) go to p(i)=user defined'
      write(*,*)'constant at a user defined time'
      write(*,12)
      write(*,*)'ENTER THE OPTION NUMBER'
      read(*,*)lptopt
c
       if(lptopt.eq.1)then
        do 543 i=1,nth,1
         pf(i)=parnd(i)
         parnd(i)=0.0
  543   continue
        write(*,*)'ENTER THE TIME AT WHICH PFINAL IS TO BE REACHED'
        read(*,*)pft
       else
       endif
c
       if(lptopt.eq.2)then
        do 544 i=1,nth,1
         pf(i)=parnd(i)
  544   continue
        write(*,*)'ENTER THE TIME AT WHICH P=0.0 IS TO BE REACHED'
        read(*,*)pft
       else
       endif

       if(lptopt.eq.3)then
        do 545 i=1,nth,1
         pf(i)=parnd(i)
         parnd(i)=0.0
  545   continue
        write(*,*)'ENTER THE TIME AT WHICH PFINAL IS TO BE REACHED'
        read(*,*)pft
        write(*,*)'ENTER THE TIME AT WHICH P IS TO JUMP TO pfin'
        read(*,*)pfthree
        write(*,*)'ENTER THE PRESSURE pfin for nodes after above time'
        read(*,*)pfin
        pfin=pfin*r/(el*h)
       else
       endif
     else
     endif
c     PVSTIME....PVSTIME....PVSTIME....PVSTIME....PVSTIME....
c     $$$$$$$  next some set up for call to DDEBDF.F   $$$$$$$$$
c     $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
       t=0.0
       tstart=t
       write(*,*)'ENTER THE VALUE OF TOUT (0.00002)'
       read(*,*)tout
       if(tout.eq.0.0)tout=0.00002
       toutsave=tout
       write(*,*)'ENTER THE VALUE OF TSTEP (0.00002)'
       read(*,*)tstep
       if(tstep.eq.0.0)tstep=0.00002
       write(*,*)'ENTER THE VALUE OF TFINAL (0.1)'
       read(*,*)tfinal
       if(tfinal.eq.0.0)tfinal=0.1
       info(1)=0
       info(2)=0
```

```
      info(3)=0
      write(*,*)'ENTER THE VALUE OF RTOL (0.000000001)'
      read(*,*)rtol
      if(rtol.eq.0.0)rtol=0.000000001
      write(*,*)'ENTER THE VALUE OF ATOL (0.000000001)'
      read(*,*)atol
      if(atol.eq.0.0)atol=0.000000001
      rpar(2)=nu
      ipar(1)=nth
c     $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c                 NEXT:MORE ON PRESSURE VS TIME
c     PVSTIME....PVSTIME....PVSTIME....PVSTIME....PVSTIME....
      if(lpvstime.eq.1)then
       if(lptopt.eq.1.or.lptopt.eq.3)then
        pfthree=pfthree*((rho*r**2.)/el)**-0.5
        ppstep=(tout/pft)
        ppa=ppstep
c
        if(ppstep.ge.1.0)then
         write(*,*)'THE VALUE OF ppstep IS GREATER THAN ONE'
         write(*,*)' therefore pressures are pf(i) at t=0.0'
         do 13 i=1,nth,1
          parnd(i)=pf(i)
 13      continue
        else
        endif
       endif
c
       if(lptopt.eq.2)then
        ppa=1.0-1.*(pft/tout)**-1.0
        ppstep=-1.*(pft/tout)**-1.0
       else
       endif
      else
      endif
c     PVSTIME....PVSTIME....PVSTIME....PVSTIME....PVSTIME....
c        NEXT: nondimensionalize the times
      tout=tout*((rho*r**2.)/el)**-0.5
      tstep=tstep*((rho*r**2.)/el)**-0.5
      tfinal=tfinal*((rho*r**2.)/el)**-0.5
c     INIT.COND...INIT.COND...INIT.COND...INIT.COND...INIT.COND..
      do 10 i=1,nth,1
       y(i)=0.0
       y(i+nth)=0.0
       y(i+nth*2)=0.0
       y(i+nth*3)=0.0
       rpar(i+3*nth)=x(i)
       rpar(i+2*nth)=hh(i)
       rpar(i+nth)=parnd(i)
 10   continue
c      next  NDSTART is a data file generated
c      from the program "ndstatic.f"
c      you may start the run with this deflection
c      and zero initial velocity
       write(*,*)'      ENTER INITIAL GUESS FOR DEFLECTIONS'
       write(*,*)'ENTER 0=all nodes start from undeformed position'
       write(*,*)'ENTER 1=start in deformed state defined in STARTND'
       read(*,*)jj
       if(jj.eq.1)then
        open(13,file='NDSTART')
        do 149 i=1,nth,1
         read(13,*)y(i),y(i+2*nth)
```

```
          write(*,*)y(i),y(i+2*nth)
  149     continue
        else
        endif
c      INIT.COND...INIT.COND...INIT.COND...INIT.COND...INIT.COND..
c      INIT.COND.LINEAR...INIT.COND.LINEAR...INIT.COND.LINEAR
c      LLLLLL if ibc=2 prescribe values of w at time = 0.0 LLLLLLL
       if(ibc.eq.2)then
        write(*,*)'YOU HAVE CHOSEN IBC=2 LINEAR BC '
        write(*,*)'YOU CAN PRESCRIBE THE LINEAR SOLUTION FOR W'
        write(*,*)'START WITH LINEAR SOLUTION?? yes=99,no=any #'
        read(*,*)j
        if(j.eq.99)then
         do 11 i=1,nth,1
          y(i+2*nth)=((1.-nu)*p(i)*r**2.)/(2.*em*h)
c             next nondimensionalize
          y(i+2*nth)=y(i+2*nth)*1./r
  11     continue
         write(*,*)'the value of y(3+2*nth)=',y(3+2*nth)
        else
        endif
       else
       endif
c      INIT.COND.LINEAR...INIT.COND.LINEAR...INIT.COND.LINEAR
        write(*,*)'ENTER # of time steps to save'
        read(*,*)numb
c
        open(14,file='bb.mat')
        open(15,file='AAMatrix.m')
c
        sumin=1.0
        sumax=0.0
        swmin=1.0
        swmax=0.0
        sdefxmin=100.0
        sdefymin=100.0
        sdefxmax=-100.0
        sdefymax=-100.0
        sthmax=-100.0
        spimax=-100.0
        sthmin=100.0
        spimin=100.0
        ethmax=-100.0
        epimax=-100.0
        ethmin=100.0
        epimin=100.0
        pmin=100.0
        pmax=-100.0
c
       call dampstress(x,y,hh,nu,ibc,epi,eth,
     +spi,sth,ethmax,epimax,ethmin,epimin,
     +sthmax,spimax,sthmin,spimin)
c
        write(14,157)t,0.0,0.0,0.0,
     +0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
        do 1118 i=1,nth,1
         write(14,157)dsin(x(i)),dcos(x(i)),y(i),y(i+2*nth),
     +y(i+nth),y(i+3*nth),rpar(i+nth),epi(i),eth(i),
     +spi(i),sth(i),rwork(20+i+nth),rwork(20+i+3*nth)
          llcount1=1
 1118   continue
 1157    format(1x,i4,2x,e13.5,2x,e13.5,2x,e13.5,2x,e13.5)
```

```
157     format(1x,e13.5,2x,e13.5,2x,e13.5,2x,e13.5,2x,e13.5,
     +2x,e13.5,2x,e13.5,2x,e13.5,2x,e13.5,2x,e13.5,2x,e13.5,
     +2x,e13.5,2x,e13.5)
c       $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c       $$$$$$$$  CALL EQUATION SOLVER DDEBDF.F $$$$$$$$$$$$$$$$$$
c       $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        xoldu=y(nth)
        xoldw=y(3*nth)
        neq=4*nth
        write(*,*)'ENTER THE TOTAL # OF LOOPS FOR RUN '
        read(*,*)llkk
        llkksave=llkk
         kdef=int(llkk/numb)
         lcount=kdef
         cont=0
         lpcount=25
c       ***** NEXT :: BIGLOOP  *********************************
        do 100 j=1,llkk,1
c          NEXT::RESTART NEEDED IF PRESSURE CHANGING WITH TIME
         if(lpvstime.eq.1.and.test3.eq.1.0)info(1)=0
c
c       111111111111111  PINNED BOTTOM BOUNDARY CONDITION 1111111
        if(ibc.eq.1)then
         tin=t
         toutin=tout
         call ddebdf(dampdf,neq,t,y,tout,info,rtol,atol,idid,
     1             rwork,lrw,iwork,liw,rpar,ipar,djac)
c           next line continues integration past 500 if nessecary
         if(idid.eq.-1)info(1)=1
        else
        endif
c       1111111111111111111111111111111111111111111111111111111111
c       2222222222222  LINEAR SOLUTION "BOUNDARY CONDITION" 2222222
        if(ibc.eq.2)then
         tin=t
         toutin=tout
           call ddebdf(dampdfl,neq,t,y,tout,info,rtol,atol,idid,
     1             rwork,lrw,iwork,liw,rpar,ipar,djac)
c           next line continues integration past 500 if nessecary
          if(idid.eq.-1)info(1)=1
        else
        endif
c       2222222222222222222222222222222222222222222222222222222222
c       333333  PINNED TOP AND BOTTOM BOUNDARY CONDITION 3333333333
        if(ibc.eq.3)then
         tin=t
         toutin=tout
         call ddebdf(dampdfpt,neq,t,y,tout,info,rtol,atol,idid,
     1             rwork,lrw,iwork,liw,rpar,ipar,djac)
c           next line continues integration past 500 if nessecary
         if(idid.eq.-1)info(1)=1
        else
        endif
c       3333333333333333333333333333333333333333333333333333333333
c       444444  INFINITE MASS BOUNDARY CONDITION  4444444444444444
c
        if(ibc.eq.4.and.ntestit.ne.1)then
         xx(1)=y(nth)
         xx(2)=y(3*nth)
         rpar(4)=xx(1)
         rpar(5)=xx(2)
         ntestit=1
```

```
      else
      endif
c
      if(ibc.eq.4)then
       neq=4*nth-4
        do 1543 m=1,4*nth-1,1
         lj=m
         if(m.ge.nth)lj=m-1
         if(m.ge.2*nth)lj=m-2
         if(m.ge.3*nth)lj=m-3
         if(m.eq.nth)goto 1543
         if(m.eq.2*nth)goto 1543
         if(m.eq.3*nth)goto 1543
         yt(lj)=y(m)
 1543   continue
       tin=t
       toutin=tout
 9441  call ddebdf(dampdfim,neq,t,yt,tout,info,rtol,atol,idid,
     1          rwork,lrw,iwork,liw,rpar,ipar,djac)
c         next line continues integration past 500 if nessecary
       if(idid.eq.-1)then
        info(1)=1
        jcount=jcount+1
       if(jcount.eq.2)goto 999
        goto 9441
       else
        jcount=0.
         xx(1)=rpar(4)
         xx(2)=rpar(5)
        do 1544 m=1,4*nth-1,1
         lj=m
         if(m.ge.nth)lj=m-1
         if(m.ge.2*nth)lj=m-2
         if(m.ge.3*nth)lj=m-3
         if(m.eq.nth)goto 1544
         if(m.eq.2*nth)goto 1544
         if(m.eq.3*nth)goto 1544
         y(m)=yt(lj)
 1544   continue
          deltat=tout-tin
          y(nth)=xx(1)
          y(2*nth)=(xx(1)-xoldu)/deltat
          y(3*nth)=xx(2)
          y(4*nth)=(xx(2)-xoldw)/deltat
          xoldu=xx(1)
          xoldw=xx(2)
       endif
      else
      endif
c     44444444444444444444444444444444444444444444444444444444
c      SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES
c      SHAPES section saves values of u and w deflections
c             and deformed shapes of membrane at "numb"
c             intermediate time steps during the run
c             also saves current time and pressures,
c             u and w velocities and
c             stress and strains at all nodes
        tfinish=t*((rho*r**2.)/el)**0.5
      if((lcount.ge.kdef).or.j.eq.llkk)then
       write(14,157)t,0.0,0.0,0.0,
     +0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
c
```

```fortran
      call dampstress(x,y,hh,nu,ibc,epi,eth,
     +spi,sth,ethmax,epimax,ethmin,epimin,
     +sthmax,spimax,sthmin,spimin)
        do 1119 i=1,nth,1
         sxdef=dsin(x(i))+y(i)*dcos(x(i))-y(i+2*nth)*dsin(x(i))
         sydef=dcos(x(i))-y(i)*dsin(x(i))-y(i+2*nth)*dcos(x(i))
         write(14,157)sxdef,sydef,y(i),y(i+2*nth),
     +y(i+nth),y(i+3*nth),rpar(i+nth),epi(i),eth(i),
     +spi(i),sth(i),rwork(20+nth+i),rwork(20+3*nth+i)
         if(sydef.gt.sdefymax)sdefymax=sydef
         if(sxdef.gt.sdefxmax)sdefxmax=sxdef
         if(sydef.lt.sdefymin)sdefymin=sydef
         if(sxdef.lt.sdefxmin)sdefxmin=sxdef
         if(y(i).lt.sumin)sumin=y(i)
         if(y(i).gt.sumax)sumax=y(i)
         if(y(i+2*nth).lt.swmin)swmin=y(i+2*nth)
         if(y(i+2*nth).gt.swmax)swmax=y(i+2*nth)
         if(rpar(i+nth).lt.pmin)pmin=rpar(i+nth)
         if(rpar(i+nth).gt.pmax)pmax=rpar(i+nth)
c
 1119    continue
        llcount1=llcount1+1
        lcount=1
       else
        lcount=lcount+1
       endif
c       SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES SHAPES
        if(lpcount.ge.25)then
         write(*,*)'LOOP# =  ',j,' CURRENT TIME= ',t,' IDID=',idid
         lpcount=0
        else
         lpcount=lpcount+1
        endif
       if(cont.eq.0)then
        write(*,15)
        do 20 i=1,nth,1
         xz=x(i)*(180./pi)
         write(*,25)i,xz,y(i),y(i+2*nth),y(i+nth),
     1             y(i+3*nth),rpar(i+nth)
  20    continue
        write(*,*)
        write(*,*)'TOUT=',tout,'  CURRENT TIME=',t
        write(*,*)'CONTINUE IN TIME??  0=yes  1=stop other noprint'
        read(*,*)cont
       else
       endif
        if(cont.eq.1)goto 999
        if(t.ge.tfinal)goto 999
c
        tout=tout+tstep
c        PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
c                       NEXT:PRESSURE CHANGE WITH TIME
       if(lpvstime.eq.1)then
        if(lptopt.eq.1)then
          if(ppa.gt.1.0)then
           ppa=1.0
           lpvstime=0.0
          else
          endif
          test3=1.0
         do 31 k=1,nth,1
          parnd(k)=ppa*pf(k)
```

```
          rpar(k+nth)=parnd(k)
 31      continue
         ppa=ppa+ppstep
        else
        endif
c
        if(lptopt.eq.2)then
          if(ppa.lt.0.0)then
           ppa=0.0
           lpvstime=0.0
          else
          endif
          test3=1.0
         do 131 k=1,nth,1
          parnd(k)=ppa*pf(k)
          rpar(k+nth)=parnd(k)
 131     continue
          ppa=ppa+ppstep
        else
        endif
c
        if(lptopt.eq.3)then
          if(tester.eq.0.0)then
           if(ppa.ge.1.0)then
            ppa=1.0
            test3=0.0
            do 341 k=1,nth,1
             parnd(k)=ppa*pf(k)
             rpar(k+nth)=parnd(k)
 341        continue
            tester=99.0
           else
            do 331 k=1,nth,1
             parnd(k)=ppa*pf(k)
             rpar(k+nth)=parnd(k)
 331        continue
            test3=1.0
            ppa=ppa+ppstep
           endif
          else
          endif
         if(t.ge.pfthree)then
          do 136 k=1,nth,1
           parnd(k)=pfin
           rpar(k+nth)=pfin
 136      continue
          test3=test3+1.0
c          lpvstime=0.0
         else
         endif
        else
        endif
       else
       endif
c        PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
  100  continue
c      BIGLOOP...BIGLOOP...BIGLOOP...BIGLOOP...BIGLOOP...
  15   format(1x,'node #',8x,'THETA',5x,'U(def)/r',
     1   8x,' W(def)/r',8x,'ndvel(U)',8x,'ndvel(W)',
     2   8x,'ndpress')
  25   format(1x,i4,3x,f6.2,3x,e13.6,3x,e13.6,3x,e13.6,
     +3x,e13.6,3x,e13.6)
```

```
c       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c       $$$$$$$$$$$$$$$$  STRESS AND STRAIN CALCULATIONS  $$$$$$$
 999  write(*,36)
  36  format(1x,'NODE #',4x,'THETA',4x,'EPSILON THETA',6x
     1  ,'EPSILON PHI',6x,'SIGMA THETA',9x,'SIGMA PHI')
c
c        NEXT write final values in matrix for plotting
c
        call dampstress(x,y,hh,nu,ibc,epi,eth,
     +spi,sth,ethmax,epimax,ethmin,epimin,
     +sthmax,spimax,sthmin,spimin)
        write(15,*)'cc=['
        do 40 i=1,nth,1
         xx1=x(i)*(180./pi)
         write(*,35)i,xx1,eth(i),epi(i),sth(i),spi(i)
         write(15,37)xx1,eth(i),epi(i),sth(i),spi(i)
  40    continue
  35    format(1x,i4,3x,f6.2,4x,e13.6,4x,e13.6,4x,e13.6,5x,e13.6)
  37    format(1x,e12.6,3x,e12.6,3x,e12.6,3x,e12.6,3x,e12.6)
        write(15,*)'];'
c        $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c         NEXT: write parameter matrix
  154    format(1x,e12.5)
  153    format(1x,i9)
         write(15,*)'aa=['
         write(15,153)nth
         write(15,153)ibc
         write(15,154)r
         write(15,154)em
         write(15,154)x(1)*(180./pi)
         write(15,154)x(nth)*(180./pi)
         write(15,154)beta
         write(15,153)iopt
         write(15,154)cu
         write(15,154)cw
         write(15,154)rho
         write(15,154)h
         write(15,154)nu
         write(15,153)llcount1
         write(15,154)sumin
         write(15,154)sumax
         write(15,154)swmin
         write(15,154)swmax
         write(15,154)pmin
         write(15,154)pmax
         write(15,154)ethmin
         write(15,154)ethmax
         write(15,154)epimin
         write(15,154)epimax
         write(15,154)sthmin
         write(15,154)sthmax
         write(15,154)spimin
         write(15,154)spimax
         write(15,154)sdefxmin
         write(15,154)sdefxmax*1.05
         write(15,154)sdefymin
         write(15,154)sdefymax*1.05
         write(15,153)0.0
         write(15,153)0.0
         write(15,154)toutsave
         write(15,154)tstart
         write(15,154)tfinish
```

```
         write(15,153)llkksave
         write(15,153)ipress
         write(15,153)lptopt
         write(15,153)0.0
         write(15,*)'];'
c       stop
        end
C   ****************************************************
C       SUBROUTINE DAMPDF.f
C   ****************************************************
        subroutine dampdf(t,y,yp,rpar,ipar)
        parameter (nth=20)
        implicit double precision (a-h,o-z)
        real*8 a,b,c,co,d,nu,parnd(nth)
        real*8 x(nth),hh(nth),pp,uu,ww
        real*8 cu,cw,uvel,wvel
        dimension y(4*nth),yp(4*nth),rpar(500)
        integer i,ipar(nth)
c       yp(1--nth)=u-velocity
c       yp(nth+1--2*nth)=u-acc.
c       yp(2*nth+1--3*nth)=w-velocity
c       yp(3*nth+1--4*nth)=w-acc.
        nu=rpar(2)
        cu=rpar(6)
        cw=rpar(7)
c       first set parameters (same as dampall.f) and velocities
        do 5 j=1,nth,1
         x(j)=rpar(j+3*nth)
         hh(j)=rpar(j+2*nth)
         parnd(j)=rpar(j+nth)
         yp(j)=y(j+nth)
         yp(j+2*nth)=y(j+3*nth)
  5     continue
c        #####  PEAK NODE BOUNDARY CONDITIONS ##################
        a=y(2)/hh(1)
        b=0.0
        c=0.0
c       next forward difference for d=d^2w/dtheta^2
c        note:equal spacing here hh(1)=hh(2)
         d=(2.*(y(2+2*nth)-y(1+2*nth)))/hh(1)**2.
c       next velocity and acc. in u-dir (yp(1)=u-vel.,
c        yp(1+nth)=u-acc. at theta=0.0)
        yp(1)=0.0
        yp(1+nth)=0.0
        co=0.0
        pp=parnd(1)
c        def. in u-dir at node 1 = 0.0
        uu=0.0
        ww=y(1+2*nth)
        wvel=y(1+3*nth)
c       next velocity and accelleration in w-dir
       yp(1+2*nth)=y(1+3*nth)
c       next: equation from applying L'Hopitals Rule
c       to "dampeb.f" equation
       yp(1+3*nth)=(a*d)-(ww*d)+(a**2.)-(a*ww)+(a*d)-(
     1   ww*d)+a**2.-(ww*a)+a+a-2.*ww+nu*(-(ww*d)
     2   -(ww*a)+(a*d)+(2.*a**2.)+(a*d)-(ww*d)-
     3   (ww*a)+a+a-2.*ww)+pp-cw*wvel
c      ######## next pinned edge conditions at maxtheta   ########
c       next velocity and acceleration at node # nth =
c            zero in u-dir and w-dir
       yp(nth)=0.0
```

```
      yp(2*nth)=0.0
      yp(3*nth)=0.0
      yp(4*nth)=0.0
c     $$$$$$$$$$$  NEXT CALCULATE YP FOR INTERIOR NODES $$$$$$$
        do 10 i=2,nth-1,1
        co=dcos(x(i))/dsin(x(i))
c       a,b,c,d are central difference formulas for
c         du/dtheta,d**2u/dtheta**2, etc
       a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))+
     1  y(i)*(1./hh(i)-1./hh(i+1))+
     2  y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
       b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
     1  2.*y(i)/(hh(i)*hh(i+1))+
     2  y(i+1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
       c=-y(i-1+2*nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)+
     1  y(i+2*nth)*(hh(i)**-1.-hh(i+1)**-1.)+
     2  y(i+1+2*nth)*((hh(i)/hh(i+1))*(hh(i)+hh(i+1))**-1.)
       d=y(i-1+2*nth)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
     1  2.*y(i+2*nth)/(hh(i)*hh(i+1))+
     2  y(i+1+2*nth)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
        pp=parnd(i)
        uu=y(i)
        ww=y(i+2*nth)
        uvel=y(i+nth)
       yp(i+nth)=dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
        wvel=y(i+3*nth)
       yp(i+3*nth)=dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
 10    continue
        return
        end
c    *****************************************************
c       SUBROUTINE DAMPDFL.f
c    *****************************************************
        subroutine dampdfl(t,y,yp,rpar,ipar)
        parameter (nth=20)
        implicit double precision (a-h,o-z)
        real*8 a,b,c,co,d,nu,parnd(nth)
        real*8 x(nth),hh(nth),pp,uu,ww
        real*8 cu,cw,uvel,wvel
        dimension y(4*nth),yp(4*nth),rpar(500)
        integer i,j,ipar(nth)
c       yp(1--nth)=u-velocity
c       yp(nth+1--2*nth)=u-acc.
c       yp(2*nth+1--3*nth)=w-velocity
c       yp(3*nth+1--4*nth)=w-acc.
        nu=rpar(2)
        cu=rpar(6)
        cw=rpar(7)
c      first set parameters (same as dampall.f)
        do 5 j=1,nth,1
         x(j)=rpar(j+3*nth)
         hh(j)=rpar(j+2*nth)
         parnd(j)=rpar(j+nth)
         yp(j)=y(j+nth)
         yp(j+2*nth)=y(j+3*nth)
  5     continue
c        #########  FIRST NODES BOUNDARY CONDITION ##########
        a=y(2)/hh(1)
        b=0.0
        c=0.0
c       next forward difference for (d) note******
c        equal spacing here(symmetry)
```

```
      d=(2.*(y(2+2*nth)-y(1+2*nth)))/hh(1)**2.
c     next velocity and acc. in u-dir (yp(1)=u-vel.,
c       yp(1+nth)=u-acc. at theta=0.0)
      yp(1)=0.0
      yp(1+nth)=0.0
      co=0.0
      pp=parnd(1)
c      def in u-dir at node 1 = 0.0
      uu=0.0
      ww=y(1+2*nth)
      wvel=y(1+3*nth)
c     next velocity and accelleration in w-dir
     yp(1+2*nth)=y(1+3*nth)
c        next equation from L'Hopitals Rule applied to dampeb.f
     yp(1+3*nth)=(a*d)-(ww*d)+(a**2.)-(a*ww)
    1   +(a*d)-(ww*d)+a**2.-(ww*a)+a+a-2.*ww
    2   +nu*(-(ww*d)-(ww*a)+(a*d)+(2.
    3   *a**2.)+(a*d)-(ww*d)-(ww*a)+a+a-2.*
    4   ww)+pp-cw*wvel
c     ######## next edge conditions at maxtheta  ###########
c     here=backwards difference equations with equal spacing
      i=nth
      a=y(nth-2)*(hh(nth)/(hh(nth-1)*(hh(nth)+hh(nth-1))))-
    1  y(nth-1)*((hh(nth)+hh(nth-1))/(hh(nth)*hh(nth-1)))+
    2  y(nth)*(1./hh(nth-1)+hh(nth)/(hh(nth-1)*
    3  (hh(nth-1)+hh(nth))))
      b=y(i-2)*2.*(hh(i-1)**-1.)*(hh(i-1)+hh(i))**-1.-
    1   2.*y(i-1)/(hh(i-1)*hh(i))+
    2   y(i)*2.*(hh(i)**-1.)*((hh(i-1)+hh(i))**-1.)
      c=y(3*nth-2)*(hh(nth)/(hh(nth-1)*(hh(nth)+hh(nth-1))))-
    1  y(3*nth-1)*((hh(nth)+hh(nth-1))/(hh(nth)*hh(nth-1)))+
    2  y(3*nth)*(1./hh(nth-1)+hh(nth)/(hh(nth-1)*(hh(nth-1)
    3  +hh(nth))))
      d=y(i-2+2*nth)*2.*(hh(i-1)**-1.)*(hh(i-1)+hh(i))**-1.-
    1  2.*y(i-1+2*nth)/(hh(i-1)*hh(i))+
    2  y(i+2*nth)*2.*(hh(i)**-1.)*((hh(i-1)+hh(i))**-1.)
      pp=parnd(nth)
      uu=y(nth)
      ww=y(3*nth)
      uvel=y(2*nth)
      wvel=y(4*nth)
      co=dcos(x(nth))/dsin(x(nth))
     yp(2*nth)=dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
     yp(4*nth)=dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
     yp(nth)=y(2*nth)
     yp(3*nth)=y(4*nth)
c     ##################################################
c     $$  NEXT CALCULATE YP FOR THE INTERIOR NODES $$$$$$$$
      do 10 i=2,nth-1,1
      co=dcos(x(i))/dsin(x(i))
     a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))+
    1  y(i)*(1./hh(i)-1./hh(i+1))+
    2  y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
     b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
    1  2.*y(i)/(hh(i)*hh(i+1))+
    2  y(i+1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
     c=-y(i-1+2*nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)+
    1  y(i+2*nth)*(hh(i)**-1.-hh(i+1)**-1.)+
    2  y(i+1+2*nth)*((hh(i)/hh(i+1))*(hh(i)+hh(i+1))**-1.)
     d=y(i-1+2*nth)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
    1  2.*y(i+2*nth)/(hh(i)*hh(i+1))+
    2  y(i+1+2*nth)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
```

```
c                 next equation call function ea.f=eq. 9a
      pp=parnd(i)
      uu=y(i)
      ww=y(i+2*nth)
      uvel=y(i+nth)
      yp(i+nth)=dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
c                 next equation is 9b=eb
      wvel=y(i+3*nth)
      yp(i+3*nth)=dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
 10   continue
      return
      end
c    ****************************************************
c     SUBROUTINE DAMPDFPT.f
c    ****************************************************
      subroutine dampdfpt(t,y,yp,rpar,ipar)
      parameter (nth=20)
      implicit double precision (a-h,o-z)
      real*8 a,b,c,co,d,nu,parnd(nth)
      real*8 x(nth),hh(nth),pp,uu,ww
      real*8 cu,cw,uvel,wvel,rpar(500)
      dimension y(nth+nth+nth+nth),yp(nth+nth+nth+nth)
      integer i,ipar(nth)
c     yp(1--nth)=u-velocity
c     yp(nth+1--2*nth)=u-acc.
c     yp(2*nth+1--3*nth)=w-velocity
c     yp(3*nth+1--4*nth)=w-acc.
      nu=rpar(2)
      cu=rpar(6)
      cw=rpar(7)
      do 5 j=1,nth,1
       x(j)=rpar(j+3*nth)
       hh(j)=rpar(j+2*nth)
       parnd(j)=rpar(j+nth)
       yp(j)=y(j+nth)
       yp(j+2*nth)=y(j+3*nth)
  5   continue
c    ######  FIRST NODES BOUNDARY CONDITIONS ##########
c     the velocity and accelleration at node # 1 =
c     zero in u-dir and w-dir
     yp(1)=0.0
     yp(nth+1)=0.0
     yp(2*nth+1)=0.0
     yp(3*nth+1)=0.0
c    ######## next edge conditions at maxtheta #########
c     next velocity and acceleration at node #
c     nth = zero in u-dir and w-dir
     yp(nth)=0.0
     yp(2*nth)=0.0
     yp(3*nth)=0.0
     yp(4*nth)=0.0
c    $$$$$$$$  NEXT CALCULATE YP FOR INTERIOR NODES $$$$
      do 10 i=2,nth-1,1
      co=dcos(x(i))/dsin(x(i))
c     a,b,c,d are central difference formulas for
c     du/dtheta,d**2u/dtheta**2, etc
     a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))+
    1  y(i)*(1./hh(i)-1./hh(i+1))+
    2  y(i+1)*((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
     b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
    1  2.*y(i)/(hh(i)*hh(i+1))+
    2  y(i+1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
```

```
     c=-y(i-1+2*nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)+
    1  y(i+2*nth)*(hh(i)**-1.-hh(i+1)**-1.)+
    2  y(i+1+2*nth)*((hh(i)/hh(i+1))*(hh(i)+hh(i+1))**-1.)
     d=y(i-1+2*nth)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
    1  2.*y(i+2*nth)/(hh(i)*hh(i+1))+
    2  y(i+1+2*nth)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
c             next equation call function dampea.f=eq. 9a
      pp=parnd(i)
      uu=y(i)
      ww=y(i+2*nth)
      uvel=y(i+nth)
     yp(i+nth)=dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
c                      next equation is 9b=eb
      wvel=y(i+3*nth)
     yp(i+3*nth)=dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
 10  continue
      return
      end
c    **************************************************
c     SUBROUTINE DAMPDFIM.f
c    **************************************************
      subroutine dampdfim(t,y,yp,rpar,ipar)
      parameter (nth=20)
      parameter (lwa=33)
      implicit double precision (a-h,o-z)
      real*8 a,b,c,co,d,nu,parnd(nth)
      real*8 x(nth),hh(nth),pp,uu,ww
      real*8 cu,cw,uvel,wvel,fvec(3),xx(3)
      real*8 y1,y2,y3,y4,wa(lwa)
      dimension y(4*nth-4),rpar(500)
      dimension yp(4*nth-4)
      integer i,ipar(nth)
      common/prop1/ x,y1,y2,y3,y4,y5,y6,y7,y8,xend
      external dampfcn
c     yp(1--nth-1)=u-velocity
c     yp(nth--2*nth-2)=u-acc.
c     yp(2*nth-1--3*nth-3)=w-velocity
c     yp(3*nth-2--4*nth-4)=w-acc.
      nu=rpar(2)
      cu=rpar(6)
      cw=rpar(7)
c    ******NOTE:   SOLVING 4*(nth-1) EQUATIONS y's are different
c                  see calling statement in dampall.f
      do 5 j=1,nth,1
       x(j)=rpar(j+3*nth)
       hh(j)=rpar(j+2*nth)
       parnd(j)=rpar(j+nth)
       if(j.eq.nth)goto 5
       yp(j)=y(j+nth-1)
       yp(j+2*nth-2)=y(j+3*nth-3)
  5   continue
c      #########  FIRST NODES BOUNDARY CONDITIONS ##########
c      #########  first at theta=0 degrees "peak" #######
      a=y(2)/hh(1)
      b=0.0
      c=0.0
c     next forwards difference for d=d**w/dtheta**2
c      note****** for equal spacing here
       d=(2.*(y(2*nth)-y(2*nth-1)))/hh(1)**2.
c     next velocity and acc. in u-dir (yp(1)=u-vel.,
c       yp(1+nth)=u-acc. at theta=0.0)
      yp(1)=0.0
```

```
      yp(nth)=0.0
      co=0.0
      pp=parnd(1)
c       def in u-dir at node 1 = 0.0
      uu=0.0
      ww=y(2*nth-1)
      wvel=y(3*nth-2)
c     next velocity and accelleration in w-dir
     yp(2*nth-1)=y(3*nth-2)
c       next equation was generated by applying L'Hopitals
c       Rule to eq. dampeb.f
     yp(3*nth-2)=(a*d)-(ww*d)+(a**2.)-(a*ww)+(a*d)-(
    1   ww*d)+a**2.-(ww*a)+a+a-2.*ww+nu*(-(ww*d)
    2   -(ww*a)+(a*d)+(2.*a**2.)+(a*d)-(ww*d)-
    3   (ww*a)+a+a-2.*ww)+pp-cw*wvel
c     $$$$$$$$$$$$  NEXT CALCULATE YP FOR INTERIOR NODES
      do 10 i=2,nth-2,1
      co=dcos(x(i))/dsin(x(i))
     a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+
    1  hh(i+1))))+y(i)*(1./hh(i)-1./hh(i+1))+y(i+1)*
    2  ((hh(i)/hh(i+1))*(1./(hh(i)+hh(i+1))))
     b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
    1  2.*y(i)/(hh(i)*hh(i+1))+y(i+1)*2.*(hh(i+1)**-1.)
    2  *((hh(i)+hh(i+1))**-1.)
     c=-y(i+2*nth-3)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)+
    1  y(i+2*nth-2)*(hh(i)**-1.-hh(i+1)**-1.)+
    2  y(i+2*nth-1)*((hh(i)/hh(i+1))*(hh(i)+hh(i+1))**-1.)
     d=y(i+2*nth-3)*2.*(hh(i)**-1.)*(hh(i)+hh(i+1))**-1.-
    1  2.*y(i+2*nth-2)/(hh(i)*hh(i+1))+
    2  y(i+2*nth-1)*2.*(hh(i+1)**-1.)*((hh(i)+hh(i+1))**-1.)
      pp=parnd(i)
      uu=y(i)
      ww=y(i+2*nth-2)
      uvel=y(i+nth-1)
      wvel=y(i+3*nth-3)
     yp(i+nth-1)=dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
     yp(i+3*nth-3)=dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
 10  continue
c    ##### next edge conditions at maxtheta  ###########
c    next solve for u and w based on circle and slope b.c.'s
c    note "end" is an imaginary node
       y1=y(nth-1)
       y2=y(nth-2)
       y5=y(nth-3)
       y6=y(nth-4)
       y3=y(3*nth-3)
       y4=y(3*nth-4)
       y7=y(3*nth-5)
       y8=y(3*nth-6)
       iiopt=2
        n=3
       if(xx(3).eq.0.0)then
        xx(1)=rpar(4)
        xx(2)=rpar(5)
        xx(3)=dsin(x(nth))*(1.-xx(2))+xx(1)*dcos(x(nth))
       else
       endif
       tol2=1.0E-08
       iinfo=0.0
       nprint=-1.0
       xend=x(nth)
c     next solve a set of 3 nonlinear algebraic equations
```

```
c       to "locate" node nth position
         call dnsqe(dampfcn,jac,iiopt,n,xx,fvec,tol2,nprint,
     +iinfo,wa,lwa)
c       xx(1)=u-deflection,xx(2)=w-deflection
        rpar(4)=xx(1)
        rpar(5)=xx(2)
        uend=rpar(4)
        wend=rpar(5)
        xend=x(nth)
        hhend=hh(nth)
        i=nth-1
        co=dcos(x(i))/dsin(x(i))
c       a,b,c,d are central difference formulas for
c        du/dtheta,d**2u/dtheta**2, etc
       a=-y(i-1)*((hhend/hh(i))*(1./(hh(i)+hhend)))+
     1  y(i)*(1./hh(i)-1./hhend)+
     2  uend*((hh(i)/hhend)*(1./(hh(i)+hhend)))
       b=y(i-1)*2.*(hh(i)**-1.)*(hh(i)+hhend)**-1.-
     1  2.*y(i)/(hh(i)*hhend)+
     2  uend*2.*(hhend**-1.)*((hh(i)+hhend)**-1.)
       c=-y(i+2*nth-3)*((hhend/hh(i))*(hh(i)+hhend)**-1.)+
     1  y(i+2*nth-2)*(hh(i)**-1.-hhend**-1.)+
     2  wend*((hh(i)/hhend)*(hh(i)+hhend)**-1.)
       d=y(i+2*nth-3)*2.*(hh(i)**-1.)*(hh(i)+hhend)**-1.-
     1  2.*y(i+2*nth-2)/(hh(i)*hhend)+
     2  wend*2.*(hhend**-1.)*((hh(i)+hhend)**-1.)
        pp=parnd(i)
        uu=y(i)
        ww=y(i+2*nth-2)
        uvel=y(i+nth-1)
        wvel=y(i+3*nth-3)
        yp(i+nth-1)=dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
        yp(i+3*nth-3)=dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
       return
       end
c     ******************************************************
c       SUBROUTINE DAMPFCN.f
c     ******************************************************
c        SUBROUTINE DAMPFCN.F IS CALLED FROM DAMPDFIM.F
c        "INFINITE MASS B.C." USED to DETERMINE VALUES
c        AT NODE POINT "NTH"
        subroutine dampfcn(n,xx,fvec,iflag)
        parameter (nth=20)
        implicit double precision(a-h,o-z)
        double precision xx(3),fvec(3)
        real*8 x(nth),ynthm1,ynthm2
        real*8 ynthm3,ynthm4,co,si
        real*8 radius,y1,y2,y3,y4,y5,y6,y7,y8,xend
        real*8 x1,x2,x3,x4
        integer n,iflag
        common/prop1/ x,y1,y2,y3,y4,y5,y6,y7,y8,xend
c        note:xend=x(nth)
c        xnth=si*(1.-xx(2))+xx(1)*co
c        ynth=co*(1.-xx(2))-xx(1)*si
c       CIRCLE          BOUNDARY CONDITION
        si=dsin(xend)
        co=dcos(xend)
        radius=si/co
        fvec(1)=(si*(1.-xx(2))+xx(1)*co)**2.+
      +((1.-xx(2))*co-xx(1)*si-1./co)**2.-(radius)**2.
c       SLOPE           BOUNDARY CONDITION
        x1=dsin(x(nth-1))*(1.-y3)+y1*dcos(x(nth-1))
```

```
      ynthm1=dcos(x(nth-1))*(1.-y3)-y1*dsin(x(nth-1))
      x2=dsin(x(nth-2))*(1.-y4)+y2*dcos(x(nth-2))
      ynthm2=dcos(x(nth-2))*(1.-y4)-y2*dsin(x(nth-2))
      x3=dsin(x(nth-3))*(1.-y7)+y5*dcos(x(nth-3))
      ynthm3=dcos(x(nth-3))*(1.-y7)-y5*dsin(x(nth-3))
      x4=dsin(x(nth-4))*(1.-y8)+y6*dcos(x(nth-4))
      ynthm4=dcos(x(nth-4))*(1.-y8)-y6*dsin(x(nth-4))
c     note:fvec(2) used to reduce size of fvec(3) equation
      fvec(2)=xx(3)-(si*(1.-xx(2))+xx(1)*co)
      fvec(3)=-(-1./dcos(xend)+co*(1.-xx(2))-xx(1)*si
     1)/(si*(1.-xx(2))+xx(1)*co)
     2+(co*(1.-xx(2))-xx(1)*si)*((xx(3)-x2)*(xx(3)-
     2x3)*(xx(3)-x4)+(xx(3)-
     3x1)*(xx(3)-x3)*(xx(3)-x4)+(xx(3)-x1)*(xx(3)-
     4x2)*(xx(3)-x4)+(xx(3)-x1)*(xx(3)-x2)*(xx(3)-x3))
     5/((xx(3)-x1)*(xx(3)-x2)*(xx(3)-x3)*(xx(3)-x4))
     6+ynthm1*((xx(3)-x2)*(xx(3)-x3)*(xx(3)-x4))
     7/((x1-xx(3))*(x1-x2)*(x1-x3)*(x1-x4))
     8+ynthm2*((xx(3)-x1)*(xx(3)-x3)*(xx(3)-x4))
     9/((x2-xx(3))*(x2-x1)*(x2-x3)*(x2-x4))
     1+ynthm3*((xx(3)-x1)*(xx(3)-x2)*(xx(3)-x4))
     2/((x3-xx(3))*(x3-x1)*(x3-x2)*(x3-x4))
     3+ynthm4*((xx(3)-x1)*(xx(3)-x2)*(xx(3)-x3))
     4/((x4-xx(3))*(x4-x1)*(x4-x2)*(x4-x3))
      return
      end
c    ****************************************************
c     SUBROUTINE DAMPSTRESS.f
c    ****************************************************
     subroutine dampstress(x,y,hh,nu,ibc,epi,eth,
     +spi,sth,ethmax,epimax,ethmin,epimin,
     +sthmax,spimax,sthmin,spimin)
      parameter (nth=20)
      implicit double precision (a-h,o-z)
      real*8 a,c
      real*8 epi(nth),eth(nth),spi(nth),sth(nth)
      real*8 hh(nth),nu,x(nth)
      real*8 sthmax,spimax,sthmin,spimin
      real*8 ethmax,epimax,ethmin,epimin
      dimension y(nth+nth+nth+nth)
      integer i,ibc
c     first for node # 1
      if(ibc.eq.3)then
        if(x(1).eq.0.)then
         a=-y(1)*((2*hh(2)+hh(3))/(hh(2)*(hh(2)+hh(3))))+
     1  y(2)*((hh(2)+hh(3))/(hh(2)*hh(3)))-y(3)
     2  *((hh(2))/(hh(3)*(hh(2)+hh(3))))
         c=-y(1+2*nth)*((2.*hh(2)+hh(3))/(hh(2)*(hh(2)+hh(3))))+
     1  y(2+2*nth)*((hh(2)+hh(3))/(hh(2)*hh(3)))-
     2  y(3+2*nth)*(hh(2)/(hh(3)*(hh(2)+hh(3))))
         eth(1)=(a-y(1+2*nth))+(1./2.)*(c+y(1))**2.
         epi(1)=(a-y(1+2*nth))
         sth(1)=(eth(1)+nu*epi(1))
         spi(1)=(epi(1)+nu*eth(1))
        else
         a=-y(1)*((2*hh(2)+hh(3))/(hh(2)*(hh(2)+hh(3))))+
     1  y(2)*((hh(2)+hh(3))/(hh(2)*hh(3)))-y(3)
     2  *((hh(2))/(hh(3)*(hh(2)+hh(3))))
         c=-y(1+2*nth)*((2.*hh(2)+hh(3))/(hh(2)*(hh(2)+hh(3))))+
     1  y(2+2*nth)*((hh(2)+hh(3))/(hh(2)*hh(3)))-
     2  y(3+2*nth)*(hh(2)/(hh(3)*(hh(2)+hh(3))))
         eth(1)=(a-y(1+2*nth))+(1./2.)*(c+y(1))**2.
```

```
         epi(1)=(y(1)*(dcos(x(1))/dsin(x(1)))-y(1+2*nth))
         sth(1)=(eth(1)+nu*epi(1))
         spi(1)=(epi(1)+nu*eth(1))
        endif
       else
        co=0.0
        a=y(2)/hh(1)
       eth(1)=a-y(1+2*nth)+(0.5)*y(1)**2.
       epi(1)=a-y(1+2*nth)
       sth(1)=eth(1)+nu*epi(1)
       spi(1)=epi(1)+nu*eth(1)
       endif
       do 30 i=2,nth-1,1
      a=-y(i-1)*((hh(i+1)/hh(i))*(1./(hh(i)+hh(i+1))))+
     1  y(i)*(1./hh(i)-1./hh(i+1))+y(i+1)*((hh(i)/
     2  hh(i+1))*(1./(hh(i)+hh(i+1))))
      c=-y(i-1+2*nth)*((hh(i+1)/hh(i))*(hh(i)+hh(i+1))**-1.)+
     1  y(i+2*nth)*(hh(i)**-1.-hh(i+1)**-1.)+
     2  y(i+1+2*nth)*((hh(i)/hh(i+1))*(hh(i)+hh(i+1))**-1.)
       eth(i)=a-y(i+2*nth)+(0.5)*(c+y(i))**2.
       epi(i)=y(i)*(dcos(x(i))/dsin(x(i)))-y(i+2*nth)
       sth(i)=eth(i)+nu*epi(i)
       spi(i)=epi(i)+nu*eth(i)
 30    continue
c       next for node # nth
       a=y(nth-2)*(hh(nth)/(hh(nth-1)*(hh(nth)+hh(nth-1))))-
     1  y(nth-1)*((hh(nth)+hh(nth-1))/(hh(nth)*hh(nth-1)))+y(nth)
     2  *((2*hh(nth)+hh(nth-1))/(hh(nth)*(hh(nth-1)+hh(nth))))
       c=y(3*nth-2)*(hh(nth)/(hh(nth-1)*(hh(nth)+hh(nth-1))))-
     1  y(3*nth-1)*((hh(nth)+hh(nth-1))/(hh(nth)*hh(nth-1)))+
     2  y(3*nth)*((2*hh(nth)+hh(nth-1))/(hh(nth)*(hh(nth-1)
     3  +hh(nth))))
       eth(nth)=a-y(3*nth)+(0.5)*(c+y(nth))**2.
       epi(nth)=y(nth)*(dcos(x(nth))/dsin(x(nth)))-y(3*nth)
       sth(nth)=eth(nth)+nu*epi(nth)
       spi(nth)=epi(nth)+nu*eth(nth)
       do 40 i=1,nth,1
        if(sth(i).gt.sthmax)sthmax=sth(i)
        if(spi(i).gt.spimax)spimax=spi(i)
        if(sth(i).lt.sthmin)sthmin=sth(i)
        if(spi(i).lt.spimin)spimin=spi(i)
        if(eth(i).gt.ethmax)ethmax=eth(i)
        if(epi(i).gt.epimax)epimax=epi(i)
        if(eth(i).lt.ethmin)ethmin=eth(i)
        if(epi(i).lt.epimin)epimin=epi(i)
 40    continue
       return
       end
C   ***************************************************
C      FUNCTION DAMPEA.f
C   ***************************************************
C      FUNCTION DAMPEA.F IS THE EQUATION FOR U-ACCELERATION
       function dampea(a,b,c,d,co,ww,uu,nu,cu,uvel)
       implicit double precision (a-h,o-z)
       real*8 a,b,c,d,co,ww,uu,nu,cu,uvel
       dampea=b-c+c*d+(uu)*d+(ww)*c+(ww*uu)-
     1  (1/2.)*c**3.-(3.*uu/2.)*c**2.-
     2  ((3.*uu**2.)/2.)*c-uu**3.
     3  /2.+a*co-uu*co**3.+(co/2.)*c**2.+((uu*
     4  co))*c+((uu**2.)*co)/2.+nu*(-uu-c-(2.*uu*co*c)
     5  +(ww*c)+(uu*ww)-(co*c**2.)/2.-(3.*(uu**2.)*co)
     6  /2.)-cu*uvel
```

```
      return
      end
c   *****************************************************
c     FUNCTION DAMPEB.f
c   *****************************************************
c     FUNCTION DAMPEB.F IS THE EQUATION FOR W-ACCELERATION
      function dampeb(a,b,c,d,co,ww,uu,nu,pp,cw,wvel)
      implicit double precision (a-h,o-z)
      real*8 a,b,c,d,co,ww,uu,nu,pp,cw,wvel
     dampeb=(co*a*c)-(co*ww*c)+(co*c**3.)/2.+(3.*co*uu
     1   *c**2.)/2.+(3.*co*c*uu**2.)/2.+(uu*co*a)
     2   -(co*uu*ww)+(co*uu**3.)/2.+(b*c)-c**2./
     3   (2.)+(3.*d*c**2.)/2.+(2.*uu*d*c)+(3.*a*c**
     4   2.)/2.+(2.*uu*a*c)+(uu*b)-(uu*c)+(3.
     5   *d*uu**2.)/2.+(3.*a*uu**2.)/2.+(a*d)-(
     6   ww*d)+a**2.-(ww*a)+a+uu*co-2.*ww+(uu*c)+uu
     7   **2./2.+nu*((-uu*c)-(ww*co*c)-(uu**2.)
     8   /2.-(ww*uu*co)+(co*a*c)-(c**2.)/2.+(2.*uu
     9   *co*a)+(uu*co*d)-(ww*d)-(ww*a)+a+uu*co-2.*
     :   ww)+pp-cw*wvel
      return
      end
```

Appendix J. Static MATLAB Program

```
% ******************************************************************
%   THIS IS THE MATLAB PROGRAM "STATIC.m" EXECUTE AFTER
%          LOADING THE FILE ndstatic.m IN MATLAB.
% ******************************************************************
clg
axis([0,100,0,100])
hold on
g1=sprintf('TOTAL # OF NODES= %g',hh(1,1));
text(10,100,g1)
g2=sprintf('RADIUS OF SPHERE= %g',hh(3,1));
text(10,95,g2)
text(40,95,'INCHES')
g3=sprintf('THICKNESS OF SPHERE= %g',hh(12,1));
text(10,90,g3)
text(45,90,'INCHES')
g4=sprintf('YOUNGS MODULUS FOR THE MEMBRANE IS= %g',hh(4,1));
text(10,85,g4)
text(60,85,'PSI')
g5=sprintf('THETA MIN.= %g',hh(5,1));
text(10,80,g5)
text(30,80,'DEGREES')
g6=sprintf('THETA MAX.= %g',hh(6,1));
text(10,75,g6)
text(30,75,'DEGREES')
g7=sprintf('POISSONS RATIO= %g',hh(13,1));
text(10,70,g7)
g8=sprintf('ROOM HERE= %g',hh(11,1));
g9=sprintf('IBC TYPE OF BOUNDARY CONDITIONS= %g',hh(2,1));
text(10,65,g9)
g10=sprintf('IOPT TYPE OF CLUSTERING= %g',hh(8,1));
text(10,60,g10)
g11=sprintf('BETA IS THE CLUSTERING PARAMETER= %g',hh(7,1));
text(10,55,g11)
g17=sprintf('THE TOTAL # OF STEPS SAVED IS %g',hh(14,1));
text(10,50,g17)
g18=sprintf('IPRESS TYPE OF PRESSURE DISTRIBUTION= %g',hh(39,1));
text(10,45,g18)
pause
clg
hold off
%    *****************************************************************
%      NEXT  DEFORMED SHAPES AT EACH PRESSURE DISTRIBUTION
%    *****************************************************************
text(0.1,0.5,'STATIC ANIMATION?? YES:ENTER 1 ,NO:RETURN','sc')
ee=input(' ');
clg
if ee==1,
nth=[hh(1,1)];
nn=[nth];
if hh(2,1)==6,hh(2,1)=3;,end
if hh(2,1)==7,hh(2,1)=3;,end
if hh(2,1)==3,
kk(1,1)=[-1./sin((hh(6,1)-90.0)*(3.14159/180.))];
axis([-hh(30,1) hh(30,1) kk(1,1) hh(32,1)])
else
axis([-hh(30,1) hh(30,1) hh(31,1) hh(32,1)])
end
```

```
plot(ff(1:nth,1),ff(1:nth,2),'o',ff(1:nth,1),ff(1:nth,2))
hold on
title('DEFORMED AND UNDEFORMED SPHERE')
xlabel('X-AXIS')
ylabel('Y_AXIS')
%
if hh(2,1)==3,
dd(2,1)=[0.0];
dd(3,1)=[ff(nn,1)];
dd(2,2)=[-1./sin((hh(6,1)-90.0)*(3.14159/180.))];
dd(1,2)=[ff(nn,2)];
dd(1,1)=[-ff(nn,1)];
dd(3,2)=[ff(nn,2)];
plot(dd(:,1),dd(:,2),':')
end
%
plot(-ff(1:nth,1),ff(1:nth,2),'o',-ff(1:nth,1),ff(1:nth,2))
nn=[nn+nth];
pause
for i=2:hh(14,1)
plot(ff(nn-nth+1:nn,1),ff(nn-nth+1:nn,2))
plot(-ff(nn-nth+1:nn,1),ff(nn-nth+1:nn,2))
plot(-ff(1:nth,1),ff(1:nth,2),'o',-ff(1:nth,1),ff(1:nth,2))
plot(ff(1:nth,1),ff(1:nth,2),'o',ff(1:nth,1),ff(1:nth,2))
if hh(2,1)==3,
dd(2,1)=[0.0];
dd(3,1)=[ff(nn,1)];
dd(2,2)=[-1./sin((hh(6,1)-90.0)*(3.14159/180.))];
dd(1,2)=[ff(nn,2)];
dd(1,1)=[-ff(nn,1)];
dd(3,2)=[ff(nn,2)];
if i==2,ll=dd;,end
plot(hh(30,1),kk(1,1)+(i/(hh(14,1)))*(hh(32,1)-kk(1,1)),'o')
plot(dd(:,1),dd(:,2),':')
else
plot(hh(30,1),hh(31,1)+(i/(hh(14,1)))*(hh(32,1)-hh(31,1)),'o')
end
pause
plot(ff(nn-nth+1:nn,1),ff(nn-nth+1:nn,2),'i')
plot(-ff(nn-nth+1:nn,1),ff(nn-nth+1:nn,2),'i')
if hh(2,1)==3,
end
nn=[nn+nth];
end
pause
hold off
clg
%    ***************************************************************
%      NEXT  U VERSUS THETA CURVES AT EACH PRESSURE DISTRIBUTION
%    ***************************************************************
axis([ hh(5,1) hh(6,1) hh(15,1) hh(16,1) ])
nn=[nth];
gg1=gg*(180./3.14159);
plot(gg1(:,1),ff(nn-nth+1:nn,3))
hold on
title('U/R DEFORMATION VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('U/R DEFORMATION')
nn=[nn+nth];
for j=2:hh(14,1)
plot(gg1(:,1),ff(1:nth,3))
plot(gg1(:,1),ff(nn-nth+1:nn,3))
```

```
plot(hh(6,1),hh(15,1)+(j/(hh(14,1)))*(hh(16,1)-hh(15,1)),'o')
pause
plot(gg1(:,1),ff(nn-nth+1:nn,3),'i')
nn=[nn+nth];
end
pause
hold off
clg
%    ****************************************************************
%       NEXT  W VERSUS THETA CURVES AT EACH PRESSURE DISTRIBUTION
%    ****************************************************************
axis([ hh(5,1) hh(6,1) hh(17,1) hh(18,1)])
nn=[nth];
plot(gg1(:,1),ff(nn-nth+1:nn,4))
hold on
title('W/R DEFORMATION VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('W/R DEFORMATION')
nn=[nn+nth];
for i=2:hh(14,1)
plot(gg1(:,1),ff(1:nth,4))
plot(gg1(:,1),ff(nn-nth+1:nn,4))
plot(hh(6,1),hh(17,1)+(i/(hh(14,1)))*(hh(18,1)-hh(17,1)),'o')
pause
plot(gg1(:,1),ff(nn-nth+1:nn,4),'i')
nn=[nn+nth];
end
pause
hold off
clg
end
ee=[0];
%    ****************************************************************
ee=[1];
while ee>0.1
clg
text(0.1,0.5,'DISP. VS PRESSURE PLOT? YES:NODE # ,NO RETURN','sc')
ee=input(' ');
if ee>0.0,
nn=[nth];
for j=1:2
if j==2,subplot(221),end
ns=[0];
for i=1:hh(14,1)
fv(i)=abs(ff(ee+ns,3));
et(i)=abs(ff(ee+ns,5));
ns=[ns+nth];
end
fv=fv';
axis([ min(fv) max(fv) min(et) max(et) ])
plot(fv,et,'o',fv,et)
xlabel('U/R DISP.')
ylabel('PRESSURE.')
g31=sprintf('PRESSURE VS. U/R DISP. FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g31,'sc'),end
if j==2,text(0.01,0.97,g31,'sc'),end
if j==1,pause,end
if j==2,subplot(222),end
ns=[0];
for i=1:hh(14,1)
fv(i)=abs(ff(ee+ns,4));
et(i)=abs(ff(ee+ns,5));
```

```
ns=[ns+nth];
end
fv=fv';
axis([ min(fv) max(fv) min(et) max(et) ])
plot(fv,et,'o',fv,et)
xlabel('W/R DISP.')
ylabel('PRESSURE')
g32=sprintf('W/R DISP. VS PRESSURE FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g32,'sc'),end
if j==2,text(0.51,0.97,g32,'sc'),end
if j==1,pause,end
if j==2,subplot(223),end
ns=[0];
for i=1:hh(14,1)
fv(i)=((ff(ee+ns,3)^2.+ff(ee+ns,4)^2.)^0.5);
et(i)=abs(ff(ee+ns,5));
ns=[ns+nth];
end
fv=fv';
axis([ min(fv) max(fv) min(et) max(et) ])
plot(fv,et,'o',fv,et)
xlabel('RES. DIS.')
ylabel('PRESSURE')
g35=sprintf('PRESSURE VS RESULTANT DISP. FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g35,'sc'),end
if j==2,text(0.01,0.97,g35,'sc'),end
if j==1,pause,end
end
pause
else
ee=[-1.0];
end
end
ee=[1];
et=[0];
fv=[0];
ex=[0];
clg
hold off
%     ****************************************************************
%               NEXT STRESS PLOTS FOR EACH PRESSURE DISTRIBUTION
%     ****************************************************************
text(0.1,0.5,'VIEW STRESS PLOTS???   YES:ENTER 1  ,NO: RETURN  ','sc')
ee=input(' ');
clg
if ee==1,
%     ****************************************************************
pp(1,1)=[6];
nth=[hh(1,1)];
nn=[nth];
axis([hh(5,1) hh(6,1) hh(23,1) hh(24,1)])
plot(gg1(:,1),ff(1:nth,pp(1,1)),'o',gg1(:,1),ff(1:nth,pp(1,1)))
hold on
title('EPSILON PHI VS THETA')
xlabel('THETA DEG.')
ylabel('EPI')
nn=[nn+nth];
for i=2:hh(14,1)
plot(gg1(:,1),ff(nn-nth+1:nn,pp(1,1)))
plot(hh(6,1),hh(23,1)+(i/(hh(14,1)))*(hh(24,1)-hh(23,1)),'o')
pause
plot(gg1(:,1),ff(nn-nth+1:nn,pp(1,1)),'i')
```

```
nn=[nn+nth];
end
pause
hold off
clg
%     ****************************************************************
pp(1,1)=[7];
nth=[hh(1,1)];
nn=[nth];
axis([hh(5,1) hh(6,1) hh(21,1) hh(22,1)])
plot(ggl(:,1),ff(1:nth,pp(1,1)),'o',ggl(:,1),ff(1:nth,pp(1,1)))
hold on
title('EPSILON THE VS THETA')
xlabel('THETA DEG.')
ylabel('ETH')
nn=[nn+nth];
for i=2:hh(14,1)
plot(ggl(:,1),ff(nn-nth+1:nn,pp(1,1)))
plot(hh(6,1),hh(21,1)+(i/(hh(14,1)))*(hh(22,1)-hh(21,1)),'o')
pause
plot(ggl(:,1),ff(nn-nth+1:nn,pp(1,1)),'i')
nn=[nn+nth];
end
pause
hold off
clg
%     ****************************************************************
pp(1,1)=[8];
nth=[hh(1,1)];
nn=[nth];
axis([hh(5,1) hh(6,1) hh(27,1) hh(28,1)])
plot(ggl(:,1),ff(1:nth,pp(1,1)),'o',ggl(:,1),ff(1:nth,pp(1,1)))
hold on
title('SIGMA-PHI VS THETA')
xlabel('THETA DEG.')
ylabel('SPI')
nn=[nn+nth];
for i=2:hh(14,1)
plot(ggl(:,1),ff(nn-nth+1:nn,pp(1,1)))
plot(hh(6,1),hh(27,1)+(i/(hh(14,1)))*(hh(28,1)-hh(27,1)),'o')
pause
plot(ggl(:,1),ff(nn-nth+1:nn,pp(1,1)),'i')
nn=[nn+nth];
end
pause
hold off
clg
%     ****************************************************************
pp(1,1)=[9];
nth=[hh(1,1)];
nn=[nth];
axis([hh(5,1) hh(6,1) hh(25,1) hh(26,1)])
plot(ggl(:,1),ff(1:nth,pp(1,1)),'o',ggl(:,1),ff(1:nth,pp(1,1)))
hold on
title('SIGMA-THETA VS THETA')
xlabel('THETA DEG.')
ylabel('STH')
nn=[nn+nth];
for i=2:hh(14,1)
plot(ggl(:,1),ff(nn-nth+1:nn,pp(1,1)))
plot(hh(6,1),hh(25,1)+(i/(hh(14,1)))*(hh(26,1)-hh(25,1)),'o')
pause
```

```
plot(gg1(:,1),ff(nn-nth+1:nn,pp(1,1)),'i')
nn=[nn+nth];
end
pause
hold off
clg
end
gg1=gg*(180./3.14159);
hold off
clg
%     ****************************************************************
%          NEXT  FINAL STRESS AND STRAIN CURVES
%     ****************************************************************
for i=1:2
if i==2,subplot(221),end
al(1)=hh(23,1);
al(2)=hh(24,1);
axis([hh(5,1) hh(6,1) al(1) al(2)])
tt=[nth*hh(14,1)];
plot(gg1(:,1),ff(tt-nth+1:tt,6),'o',gg1(:,1),ff(tt-nth+1:tt,6))
title('EPI VS THETA')
xlabel('THETA DEG.')
ylabel('EPI.')
if i==1,pause,clg,end
%     ****************************************************************
if i==2,subplot(222),end
al(1)=hh(21,1);
al(2)=hh(22,1);
axis([hh(5,1) hh(6,1) al(1) al(2)])
plot(gg1(:,1),ff(tt-nth+1:tt,7),'o',gg1(:,1),ff(tt-nth+1:tt,7))
title('ETH VS THETA')
xlabel('THETA IN DEG.')
ylabel('ETH.')
if i==1,pause,clg,end
%     ****************************************************************
if i==2,subplot(223),end
al(1)=hh(27,1);
al(2)=hh(28,1);
axis([hh(5,1) hh(6,1) al(1) al(2)])
plot(gg1(:,1),ff(tt-nth+1:tt,8),'o',gg1(:,1),ff(tt-nth+1:tt,8))
title('SPI VS THETA')
xlabel('THETA IN DEG.')
ylabel('SPI.')
if i==1,pause,clg,end
%     ****************************************************************
if i==2,subplot(224),end
al(1)=hh(25,1);
al(2)=hh(26,1);
axis([hh(5,1) hh(6,1) al(1) al(2)])
plot(gg1(:,1),ff(tt-nth+1:tt,9),'o',gg1(:,1),ff(tt-nth+1:tt,9))
hold on
title('STH VS THETA')
xlabel('THETA IN DEG.')
ylabel('STH.')
hold off
pause
end
clg
%     ****************************************************************
%                   NEXT PRESSURE CURVES VERSUS THETA
%     ****************************************************************
axis([ hh(5,1) hh(6,1) hh(19,1) hh(20,1) ])
```

```
nn=[nth];
plot(gg1(:,1),ff(nn-nth+1:nn,5))
hold on
title('PRESSURE VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('PRESSURE')
nn=[nn+nth];
for j=2:hh(14,1)
plot(gg1(:,1),ff(nn-nth+1:nn,5))
plot(hh(6,1),hh(19,1)+(j/(hh(14,1)-1))*(hh(20,1)-hh(19,1)),'o')
pause
plot(gg1(:,1),ff(nn-nth+1:nn,5),'i')
nn=[nn+nth];
end
pause
hold off
clg
```

Appendix K. Dynamic MATLAB Program

```
% *****************************************************************
%      THIS IS THE MATLAB PROGRAM "DYNAMIC.m" EXECUTE AFTER
%        LOADING THE FILE AAMatrix.m AND bb.mat IN MATLAB.
% *****************************************************************
clg
axis([0,100,0,100])
hold on
g1=sprintf('TOTAL # OF NODES= %g',aa(1,1));
text(10,100,g1)
g2=sprintf('RADIUS OF SPHERE= %g',aa(3,1));
text(10,95,g2)
text(40,95,'INCHES')
g3=sprintf('THICKNESS OF SPHERE= %g',aa(12,1));
text(10,90,g3)
text(45,90,'INCHES')
g4=sprintf('YOUNGS MODULUS FOR THE MEMBRANE IS= %g',aa(4,1));
text(10,85,g4)
text(60,85,'PSI')
g5=sprintf('THETA MINIMUM= %g',aa(5,1));
text(10,80,g5)
text(30,80,'DEGREES')
g6=sprintf('THETA MAXIMUM= %g',aa(6,1));
text(10,75,g6)
text(30,75,'DEGREES')
g7=sprintf('POISSONS RATIO= %g',aa(13,1));
text(10,70,g7)
g8=sprintf('DENSITY= %g',aa(11,1));
text(10,65,g8)
text(30,65,'LBS*SEC**2/INCHES**4')
g9=sprintf('IBC TYPE OF BOUNDARY CONDITIONS= %g',aa(2,1));
text(10,60,g9)
g10=sprintf('IOPT TYPE OF CLUSTERING= %g',aa(8,1));
text(10,55,g10)
g11=sprintf('BETA CLUSTERING PARAMETER= %g',aa(7,1));
text(10,50,g11)
g12=sprintf('DAMPING RATIO FOR U-EQUATIONS= %g',aa(9,1));
text(10,45,g12)
g13=sprintf('DAMPING RATIO FOR W-EQUATIONS= %g',aa(10,1));
text(10,40,g13)
g17=sprintf('THE TOTAL # OF TIME STEPS SAVED IS %g',aa(14,1));
text(10,35,g17)
g18=sprintf('IPRESS TYPE OF PRESSURE DISTRIBUTION= %g',aa(39,1));
text(10,30,g18)
g19=sprintf('LPTOPT TYPE OF PRES.VS.TIME PARAMETER= %g',aa(40,1));
text(10,25,g19)
g20=sprintf('THE STARTING TIME FOR THE RUN IS %g',aa(36,1));
text(10,20,g20)
text(60,20,'SEC')
g21=sprintf('THE FINAL TIME FOR THIS RUN= %g',aa(37,1));
text(10,15,g21)
text(60,15,'SEC')
pause
clg
hold off
text(0.1,0.5,'ANIMATION???? YES:ENTER 1 NO: RETURN','sc')
ee=input(' ');
clg
```

```
if ee==1,
%   *****************************************************************
%        NEXT DEFORMED SPHERICAL SHAPE IN ANIMATION SEQUENCE
%   *****************************************************************
text(0.1,0.5,'VIEW DEFORMING SPHERE?? NO:ENTER 1 ,YES:RETURN','sc')
zt=input(' ');
clg
nth=[aa(1,1)];
nn=[nth+1];
no=[nn];
if aa(2,1)==4,
gg(1,1)=[-1./sin((aa(6,1)-90.0)*(3.14159/180.))];
axis([-aa(30,1) aa(30,1) gg(1,1) aa(32,1)])
else
axis([-aa(30,1) aa(30,1) aa(31,1) aa(32,1)])
end
plot(bb(2:nth+1,1),bb(2:nth+1,2),'o',bb(2:nth+1,1),bb(2:nth+1,2))
hold on
title('DEFORMED AND UNDEFORMED SPHERE')
xlabel('X-AXIS')
ylabel('Y_AXIS')
plot(-bb(2:nth+1,1),bb(2:nth+1,2),'o',-bb(2:nth+1,1),bb(2:nth+1,2))
nn=[nn+no];
mm=[nn];
for i=2:aa(14,1)-1
if zt==1,break,end
plot(bb(nn-nth+1:nn,1),bb(nn-nth+1:nn,2))
plot(-bb(nn-nth+1:nn,1),bb(nn-nth+1:nn,2))
%
plot(bb(mm-nth+1:mm,1),bb(mm-nth+1:mm,2),'o',...
bb(mm-nth+1:mm,1),bb(mm-nth+1:mm,2))
plot(-bb(mm-nth+1:mm,1),bb(mm-nth+1:mm,2),'o',...
-bb(mm-nth+1:mm,1),bb(mm-nth+1:mm,2))
%
if i==2,
g30=sprintf('THE TIME STEP = %g',bb(nn-nth,1));
text(0.65,0.03,g30,'sc')
end
plot(aa(30,1),aa(31,1)+(i/(aa(14,1)-1))*(aa(32,1)-aa(31,1)),'o')
plot(-bb(2:nth+1,1),bb(2:nth+1,2),'o',-bb(2:nth+1,1),bb(2:nth+1,2))
plot(bb(2:nth+1,1),bb(2:nth+1,2),'o',bb(2:nth+1,1),bb(2:nth+1,2))
if aa(2,1)==4,
dd(2,1)=[0.0];
dd(3,1)=[bb(nn,1)];
dd(2,2)=[-1./sin((aa(6,1)-90.0)*(3.14159/180.))];
dd(1,2)=[bb(nn,2)];
dd(1,1)=[-bb(nn,1)];
dd(3,2)=[bb(nn,2)];
if i==2,ff=dd;,end
plot(dd(:,1),dd(:,2),':')
plot(ff(:,1),ff(:,2),':')
end
pause
plot(bb(nn-nth+1:nn,1),bb(nn-nth+1:nn,2),'i')
plot(-bb(nn-nth+1:nn,1),bb(nn-nth+1:nn,2),'i')
if aa(2,1)==4,
plot(dd(:,1),dd(:,2),'i')
end
nn=[nn+no];
end
hold off
clg
```

```
%   ******************************************************************
%       NEXT U DEFLECTION VERSUS THETA ANIMATED IN TIME
%   ******************************************************************
text(0.1,0.5,'VIEW TANGENTIAL DISP.?? NO:ENTER 1 ,YES:RETURN','sc')
zv=input(' ');
clg
axis([ aa(5,1) aa(6,1) aa(15,1) aa(16,1) ])
nn=[nth+1];
plot(cc(:,1),bb(nn-nth+1:nn,3))
hold on
title('U/R DEFORMATION VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('U/R DEFORMATION')
nn=[nn+no];
for j=2:aa(14,1)-1
if zv==1,break,end
plot(cc(:,1),bb(2:nth+1,3))
plot(cc(:,1),bb(nn-nth+1:nn,3))
plot(aa(6,1),aa(15,1)+(j/(aa(14,1)-1))*(aa(16,1)-aa(15,1)),'o')
pause
plot(cc(:,1),bb(nn-nth+1:nn,3),'i')
nn=[nn+no];
end
pause
hold off
clg
%   ******************************************************************
%       NEXT W DEFLECTION VERSUS THETA ANIMATED IN TIME
%   ******************************************************************
axis([ aa(5,1) aa(6,1) aa(17,1) aa(18,1)])
nn=[nth+1];
plot(cc(:,1),bb(nn-nth+1:nn,4))
hold on
title('W/R DEFORMATION VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('W/R DEFORMATION')
nn=[nn+no];
for i=2:aa(14,1)-1
plot(cc(:,1),bb(2:nth+1,4))
plot(cc(:,1),bb(nn-nth+1:nn,4))
plot(aa(6,1),aa(17,1)+(i/(aa(14,1)-1))*(aa(18,1)-aa(17,1)),'o')
pause
plot(cc(:,1),bb(nn-nth+1:nn,4),'i')
nn=[nn+no];
end
pause
hold off
clg
end
ee=[0];
et=rand(1)
%     *************************************************************
ee=[1];
while ee>0.1
clg
text(0.1,0.5,'DISPLACEMENT VS TIME PLOT? YES:NODE #,NO RETURN','sc')
ee=input(' ')
if ee>0.0,
text(0.1,0.3,'U/R,W/R, or RESULTANT/R ENTER 1,2, or 3','sc')
xf=input('  ');
nn=[no+1];
%   ******************************************************************
```

```
%       NEXT W DEFLECTIONS, VELOCITIES & ACCELERATIONS VERSUS TIME
%   *****************************************************************
if xf==1
for j=1:2
if j==2,subplot(221),end
ns=[0];
nn=[no+1];
et(1)==bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,3);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ min(et) max(et) aa(15,1) aa(16,1) ])
plot(et,fv,'o',et,fv)
xlabel('TIME')
ylabel('U/R DIS.')
g31=sprintf('U/R DISP. VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g31,'sc'),end
if j==2,text(0.01,0.97,g31,'sc'),end
if j==1,pause,end
%   *****************************************************************
%
if j==2,subplot(222),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,5);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ min(et) max(et) min(fv) max(fv) ])
plot(et,fv,'o',et,fv)
xlabel('TIME')
ylabel('U/R VEL.')
g32=sprintf('U/R VEL. VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g32,'sc'),end
if j==2,text(0.51,0.97,g32,'sc'),end
if j==1,pause,end
end
%   *****************************************************************
%   *********    NEXT: U-ACC vs time for node #
if j==2,subplot(223),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,12);
nn=[ns+no];
end
et=et';
fv=fv';
if min(fv)~=max(fv),
```

```
axis([ min(et) max(et) min(fv) max(fv)])
plot(et,fv,'o',et,fv)
title('U/R ACC. VS TIME')
xlabel('TIME')
ylabel('U/R ACC.')
else
text(0.1,0.3,'U/R ACC. vs. TIME is a CONSTANT','sc')
end
g40=sprintf('U/R ACC. VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.72,0.02,g40,'sc'),end
if j==2,text(0.45,0.5,g40,'sc'),end
if j==1,pause,end
%   *****************************************************************
subplot(224)
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,7);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ min(et) max(et) aa(19,1) aa(20,1) ])
plot(et,fv,'o',et,fv)
title('PRESSURE VS TIME')
xlabel('TIME')
ylabel('PRESSURE')
pause
end
%   *****************************************************************
%       NEXT W DEFLECTIONS, VELOCITIES & ACCELERATIONS VERSUS TIME
%   *****************************************************************
if xf==2
for j=1:2
if j==2,subplot(221),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,4);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ 0.0 max(et) aa(17,1) aa(18,1)])
plot(et,fv,'o',et,fv)
xlabel('TIME')
ylabel('W/R DIS.')
g33=sprintf('W/R DISP. VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g33,'sc'),end
if j==2,text(0.01,0.97,g33,'sc'),end
if j==1,pause,end
%   *****************************************************************
if j==2,subplot(222),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
```

```
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,6);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ 0.0 max(et) min(fv) max(fv) ])
plot(et,fv,'o',et,fv)
xlabel('TIME')
ylabel('W/R VEL.')
g34=sprintf('W/R VEL. VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g34,'sc'),end
if j==2,text(0.51,0.97,g34,'sc'),end
if j==1,pause,end
end
%   ****************************************************************
if j==2,subplot(223),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,13);
ns=[ns+no];
end
et=et';
fv=fv';
if min(fv)~=max(fv),
axis([ min(et) max(et) min(fv) max(fv)])
plot(et,fv,'o',et,fv)
title('W/R ACC. VS TIME')
xlabel('TIME')
ylabel('W/R ACC.')
else
text(0.1,0.3,'W/R ACC. vs. TIME is a CONSTANT','sc')
end
g42=sprintf('W/R ACC. VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.72,0.02,g42,'sc'),end
if j==2,text(0.45,0.5,g42,'sc'),end
if j==1,pause,end
%   ****************************************************************
subplot(224)
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,7);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ min(et) max(et) aa(19,1) aa(20,1) ])
plot(et,fv,'o',et,fv)
title('PRESSURE VS TIME')
xlabel('TIME')
ylabel('PRESSURE')
pause
```

```
end
%   ******************************************************************
%        NEXT RESULTANT DEFLECTIONS AND VELOCITIES VS TIME
%   ******************************************************************
if xf==3
for j=1:2
if j==2,subplot(221),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=((bb(ee+1+ns,3)^2.+bb(ee+1+ns,4)^2.)^0.5);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ 0.0 max(et) min(fv) max(fv) ])
plot(et,fv,'o',et,fv)
xlabel('TIME')
ylabel('RESULTANT DISPLACEMENT')
g35=sprintf('RESULTANT DISPLACEMENT VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g35,'sc'),end
if j==2,text(0.01,0.97,g35,'sc'),end
if j==1,pause,end
%   ******************************************************************
if j==2,subplot(222),end
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=((bb(ee+1+ns,5)^2.+bb(ee+1+ns,6)^2.)^0.5);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ 0.0 max(et) min(fv) max(fv) ])
plot(et,fv,'o',et,fv)
xlabel('TIME')
ylabel('RESULTANT VELOCITY')
g36=sprintf('RESULTANT VELOCITY VS TIME FOR NODE NUMBER %g',ee);
if j==1,text(0.3,0.97,g36,'sc'),end
if j==2,text(0.51,0.97,g36,'sc'),end
if j==1,pause,end
%   ******************************************************************
end
subplot(224)
ns=[0];
nn=[no+1];
et(1)=bb(1,1);
for i=1:aa(14,1)-1
et(i)=bb(nn,1);
nn=[nn+no];
fv(i)=bb(ee+1+ns,7);
ns=[ns+no];
end
et=et';
fv=fv';
axis([ min(et) max(et) aa(19,1) aa(20,1) ])
```

```
plot(et,fv,'o',et,fv)
title('PRESSURE VS TIME')
xlabel('TIME')
ylabel('PRESSURE')
pause
end
else
ee=[-1.0];
end
end
ee=[1];
et=[0];
fv=[0];
ex=[0];
clg
hold off
%   ****************************************************************
%                    NEXT STRESS ANIMATION PLOTS
text(0.1,0.5,'STRESS ANIMATION? YES:ENTER 1 ,NO:RETURN','sc')
ee=input(' ');
clg
if ee==1,
%   ****************************************************************
gg(1,1)=[8];
nth=[aa(1,1)];
nn=[nth+1];
no=[nn];
axis([aa(5,1) aa(6,1) aa(23,1) aa(24,1)])
plot(cc(:,1),bb(2:nth+1,gg(1,1)),'o',cc(:,1),bb(2:nth+1,gg(1,1)))
hold on
title('EPSILON PHI VS THETA')
xlabel('THETA in DEGREES')
ylabel('EPSILON PHI')
nn=[nn+no];
for i=2:aa(14,1)-1
plot(aa(6,1),aa(23,1)+(i/(aa(14,1)-1))*(aa(24,1)-aa(23,1)),'o')
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)))
pause
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)),'i')
nn=[nn+no];
end
hold off
clg
%   ****************************************************************
gg(1,1)=[9];
nth=[aa(1,1)];
nn=[nth+1];
no=[nn];
axis([aa(5,1) aa(6,1) aa(21,1) aa(22,1)])
plot(cc(:,1),bb(2:nth+1,gg(1,1)),'o',cc(:,1),bb(2:nth+1,gg(1,1)))
hold on
title('EPSILON THETA VS THETA')
xlabel('THETA in DEGREES')
ylabel('EPSILON THETA')
nn=[nn+no];
for i=2:aa(14,1)-1
plot(aa(6,1),aa(21,1)+(i/(aa(14,1)-1))*(aa(22,1)-aa(21,1)),'o')
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)))
pause
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)),'i')
nn=[nn+no];
end
```

```
hold off
clg
%   ***************************************************************
gg(1,1)=[10];
nth=[aa(1,1)];
nn=[nth+1];
no=[nn];
axis([aa(5,1) aa(6,1) aa(27,1) aa(28,1)])
plot(cc(:,1),bb(2:nth+1,gg(1,1)),'o',cc(:,1),bb(2:nth+1,gg(1,1)))
hold on
title('SIGMA PHI VS THETA')
xlabel('THETA in DEGREES')
ylabel('SIGMA PHI')
nn=[nn+no];
for i=2:aa(14,1)-1
plot(aa(6,1),aa(27,1)+(i/(aa(14,1)-1))*(aa(28,1)-aa(27,1)),'o')
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)))
pause
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)),'i')
nn=[nn+no];
end
hold off
clg
%   ***************************************************************
gg(1,1)=[11];
nth=[aa(1,1)];
nn=[nth+1];
no=[nn];
axis([aa(5,1) aa(6,1) aa(25,1) aa(26,1)])
plot(cc(:,1),bb(2:nth+1,gg(1,1)),'o',cc(:,1),bb(2:nth+1,gg(1,1)))
hold on
title('SIGMA THETA VS THETA')
xlabel('THETA in DEGREES')
ylabel('SIGMA THETA')
nn=[nn+no];
for i=2:aa(14,1)-1
plot(aa(6,1),aa(25,1)+(i/(aa(14,1)-1))*(aa(26,1)-aa(25,1)),'o')
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)))
pause
plot(cc(:,1),bb(nn-nth+1:nn,gg(1,1)),'i')
nn=[nn+no];
end
hold off
clg
end
%
%   ***************************************************************
%    NEXT FINAL STRESS AND STRAIN VALUES VERSUS THETA
for i=1:2
if i==2,subplot(221),end
al(1,1)=aa(21,1);
al(2,1)=aa(22,1);
if aa(21,1)==aa(22,1),al(1,1)=aa(21,1)*0.95;,...
al(2,1)=aa(22,1)*1.05;,end
axis([aa(5,1) aa(6,1) al(1,1) al(2,1)])
plot(cc(:,1),cc(:,2),'o',cc(:,1),cc(:,2))
title('EPSILON THETA VS THETA')
xlabel('THETA in DEGREES')
ylabel('EPSILON THETA')
if i==1,pause,end
%   ***************************************************************
if i==2,subplot(222),end
```

```
al(1,1)=aa(23,1);
al(2,1)=aa(24,1);
if aa(23,1)==aa(24,1),al(1,1)=aa(23,1)*0.95;,...
al(2,1)=aa(24,1)*1.05;,end
axis([aa(5,1) aa(6,1) al(1,1) al(2,1)])
plot(cc(:,1),cc(:,3),'o',cc(:,1),cc(:,3))
title('EPSILON PHI VS THETA')
xlabel('THETA IN DEGREES')
ylabel('EPSILON PHI')
if i==1,pause,end
%   *****************************************************************
if i==2,subplot(223),end
al(1,1)=aa(25,1);
al(2,1)=aa(26,1);
if aa(25,1)==aa(26,1),al(1,1)=aa(25,1)*0.95;,...
al(2,1)=aa(26,1)*1.05;,end
axis([aa(5,1) aa(6,1) al(1,1) al(2,1)])
plot(cc(:,1),cc(:,4),'o',cc(:,1),cc(:,4))
title('SIGMA THETA VS THETA')
xlabel('THETA IN DEGREES')
ylabel('SIGMA THETA')
if i==1,pause,end
%   *****************************************************************
if i==2,subplot(224),end
al(1,1)=aa(27,1);
al(2,1)=aa(28,1);
if aa(27,1)==aa(28,1),al(1,1)=aa(27,1)*0.95;,...
al(2,1)=aa(28,1)*1.05;,end
axis([aa(5,1) aa(6,1) al(1,1) al(2,1)])
plot(cc(:,1),cc(:,5),'o',cc(:,1),cc(:,5))
title('SIGMA PHI VS THETA')
xlabel('THETA IN DEGREES')
ylabel('SIGMA PHI')
pause
end
clg
%   *****************************************************************
%         NEXT PRESSURE VS TIME CURVES
if aa(19,1)==aa(20,1)
nn=[nth+1];
al(1)=aa(19,1)*1.05;
al(2)=aa(20,1)*0.95;
axis([ aa(5,1) aa(6,1) al(1) al(2) ])
plot(cc(:,1),bb(nn-nth+1:nn,5))
title('PRESSURE=CONSTANT WITH TIME VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('PRESSURE')
pause
else
axis([ aa(5,1) aa(6,1) aa(19,1) aa(20,1) ])
nn=[nth+1];
plot(cc(:,1),bb(nn-nth+1:nn,7))
hold on
title('PRESSURE VS. THETA')
xlabel('THETA IN DEGREES')
ylabel('PRESSURE')
nn=[nn+no];
for j=2:aa(14,1)-1
plot(cc(:,1),bb(nn-nth+1:nn,7))
plot(aa(6,1),aa(19,1)+(j/(aa(14,1)-1))*(aa(20,1)-aa(19,1)),'o')
pause
plot(cc(:,1),bb(nn-nth+1:nn,7),'i')
```

```
nn=[nn+no];
end
pause
end
hold off
clg
%    ************************************************************************
```

This document reports research undertaken at the US Army Natick Research, Development and Engineering Center and has been assigned No. NATICK/TR-93/017 in the series of reports approved for publication.