National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents

NL-339 (r. 88/04) c

Canada

Interlaminar Stresses in Tapered Laminates

Jérôme Daoust

A Thesis

in

The Department

of

Mechanical Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montréal, Québec, Canada

February 1989

© Jérôme Daoust, 1989

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-49125-6

Canada

ABSTRACT

Interlaminar Stresses in Tapered Laminates

Jérôme Daoust, Ph.D.
Concordia University, 1989

Tapered laminates by internal ply drop-offs have been studied. The study consists of two parts: two dimensional and three dimensional. For the two dimensional part, the drop-off is modeled as a triangular hole of a given length/height ratio in an anisotropic plate with a tensile load at any angle. A two dimensional approximate closed form solution is obtained for the stress distribution around the drop-off region through the use of complex curvilinear coordinates.

For cases that the closed form solution can not handle, an extensive finite element program has been developed. A 20 node, three dimensional element that can handle orthotropic material properties through a displacement formulation is used. The material direction can be offset from the element direction. The program permits optimization of dimensions, angles and materials.

A microcomputer environment was prefered over the usual mainframe. This choice required more effort to work around the microcomputer's limitations but greater portability of the program prevailed.

For the two dimensional problem, the closed form solution agrees with finite element results. Also finite element results with a certain mesh size are found to match results from experiments performed on tapered laminates made of laminae of 0°, 45° and 90° orientations. Loadings were both tensile and bending. Interlaminar stresses are found to be of large magnitude. Optimal drop-off locations for minimum interlaminar stresses are found. The effects of loading condition, drop-off shape, drop-off resin content and width to thickness ratio on interlaminar stresses are analyzed.

An expression for the prediction of tapered laminate strength due to combined loading conditions is established. Fiber orientation for layers in contact with the drop-off wedge is optimized for minimum interlaminar stresses. Crack growth simulation is performed by consecutive removal of failed elements.

ACKNOWLEDGEMENTS

The author would like to express his sincere appreciation to Dr. S.V. Hoa for his guidance and supervision.

Special thanks to my wife, Sylvie, and to my parents for their encouragements.

Thanks are due to the Industrial Material Research Institute for financial support and to Bell helicopters for material and discussions.

TABLE OF CONTENTS

VOLUME I

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF SYMBOLS

Main symbols are defined here. Others are defined locally.

A	area
β	fiber angle
[BF]	$\{e'\}=[BF]\{\delta\}$
[BG]	$\{e\}=[BG]\{\delta\}$
c_i	extrapolation function coefficient
{c}	extrapolation function coefficient vector
[C]	elastic constraint matrix in material coordinates
e_{ij}	tensorial strain in i-j direction
{e}	tensorial strain in global coordinates
{e'}	tensorial strain in local coordinates
$\{\bar{e}\}$	tensorial strain in material coordinates
E_i	Young's modulus along direction i
[E]	$\{\tau\}_{nodes}=[E]\{c\}$
[EI']	Elastic constraint matrix in local coordinates
{f}	nodal traction forces
G_{ij}	shear modulus in direction i-j
i,j,k	unit vectors in x,y,z coordinates
[J]	Jacobian matrix
[k]	stiffness matrix
L,T	longitudinal and transverse directions

LIST OF SYMBOLS (Continued)

[L]	matrix of local unit vectors
m,n	cosine and sine of β
[N]	vector of nodal shape function
N_i	nodal shape function
P_i	direction vector i
[P],[Q],[S]	$[P]\{c\}=[Q]\{\tau\}_{opt}$ $[S]=[P]^{-1}[Q]$
R	position vector
{r}	nodal reaction forces
S	maximum shear stress in LT plane
$\bar{t}$	traction
[TR]	transformation matrix
u,v,w	displacement along x,y,z directions
{u},{v},{w}	nodal displacement along in global coordinates
V	volume
V_1, V_2, V_3	unit vectors in local coordinates
x,y,z	global coordinates
{x},{y},{z}	global nodal coordinates
x',y',z'	local coordinate directions
X	maximum stress in L direction
{Δ},{δ}	nodal displacements in global coordinates
δU	internal strain energy variation
δV	potential energy variation

LIST OF SYMBOLS (Continued)

Δ	determinant of a matrix
ε_{ij}	engineering strain in i-j direction
$\{\varepsilon\}$	engineering strain in global coordinates
$\nu_{12}, \nu_{13}, \nu_{23}$	Poisson's ratios
Ω	integration domain
ψ_i	extrapolation shape function
ρ, ϑ	curvilinear coordinates
σ	value of ζ on unit circle (ρ=1) (chapter 2)
σ_{ij}	stress in i-j direction (chapter 3 to 5)
σ^*_{ij}	normalized stress in i-j direction (divided by σ_x)
$\sigma_{u_{ij}}$	ultimate stress in i-j direction
$\{\sigma\}$	stress vector in global coordinates
$\{\sigma'\}$	stress vector in local coordinates
$\{\bar{\sigma}\}$	stress vector in material coordinates
τ_{opt}	stress at an optimum sampling point
τ_S	stress at a sampling point
$\{\tau\}_{opt}$	vector of stress at optimal sampling points
$\{\tau\}_{nodes}$	nodal stresses
ξ, η, ζ	normalized local coordinate direction ($-1 \leq \leq 1$)
ξ_i, η_i, ζ_i	nodal local coordinates
ζ	curvilinear variable = $\rho . e^{i\vartheta}$

LIST OF SYMBOLS (Continued)

1,2,3	principal material directions

operators:

$\{A\}.\{B\}$	dot product for vectors A and B
$\{A\}\times\{B\}$	vector product for vectors A and B
x'	local coordinate direction
y'	derivative of y
{ }	identifies a column vector
[ ]	identifies a matrix or a row vector
$[\ ]^T$	transpose of a matrix
$\frac{\partial(\ )}{\partial x_i}$	partial derivation with respect to i direction
δ	infinitesimal variation operator
Σ	summation operator
$[D_{1_N}/D_2]_S$	A laminate of N layers at angle D_1 and one layer at angle D_2, symmetric
$\bar{z}$	conjugate of z (z=x+iy). $\bar{z}$=x-iy

GLOSSARY OF COMPUTER TERMS

access time: Time required to access data from storage media.

batch mode: Mode in which the execution of work is controlled through previously written commands in a file.

buffers: An area of memory allocated as temporary storage, usually for keeping latest accessed information.

bit: A binary digit with a value of either 0 or 1.

byte: A sequence of 8 bits.

code: Instructions to a compiler. Program.

math coprocessor: A special chip to enhance calculation speed.

compiler: A program which translates code written in high-level language into the computer's language.

computer size: Microcomputers are intended for personal use. Minicomputers are the main system of many universities. Super-computers are used for extensive calculations.

configuration: The manner in which a computer is related to different components.

directory: A work area on a disk.

element library: A list of available element formulations.

file: A collection of data that can be stored or retrieved from a disk.

graphic card: A special circuit board for driving graphic display monitors.

GLOSSARY OF COMPUTER TERMS (Continued)

high-level language: A programming language which more resembles human language than computer language. *PASCAL* is one.

heap: An area of memory reserved for the dynamic allocation of variable.

integer: A numeric variable that is a whole number.

interactive mode: A mode permitting a user to directly control the work process.

kB: 1024 bytes

landscape plot: Representation of three dimensional data as a landscape seen from a viewpoint.

loop: A set of statements which are executed repeatedly.

material database: A special feature of the currently developed program allowing to add, edit, remove materials and their properties. All materials are later referenced through the database.

MB: 1024 kB

mesh: The geometric arrangement of finite elements.

MHz: The clock speed unit of a computer. Clock speed is an indication of execution speed.

preprocessing: Treating mesh-generating commands from a defined language prior to solving for unknowns.

GLOSSARY OF COMPUTER TERMS (Continued)

postprocessing: Treatment of results. Postprocessor are intended to help data analysis.

RAM: Random-Access Memory. Memory devices that can be read from and written to.

real: A number representing by decimal point and/or scientific notation.

submatrix: A continuous part of a matrix. The size is that of a node's number of degrees of freedom.

window: The active or visual part of a whole.

CHAPTER 1

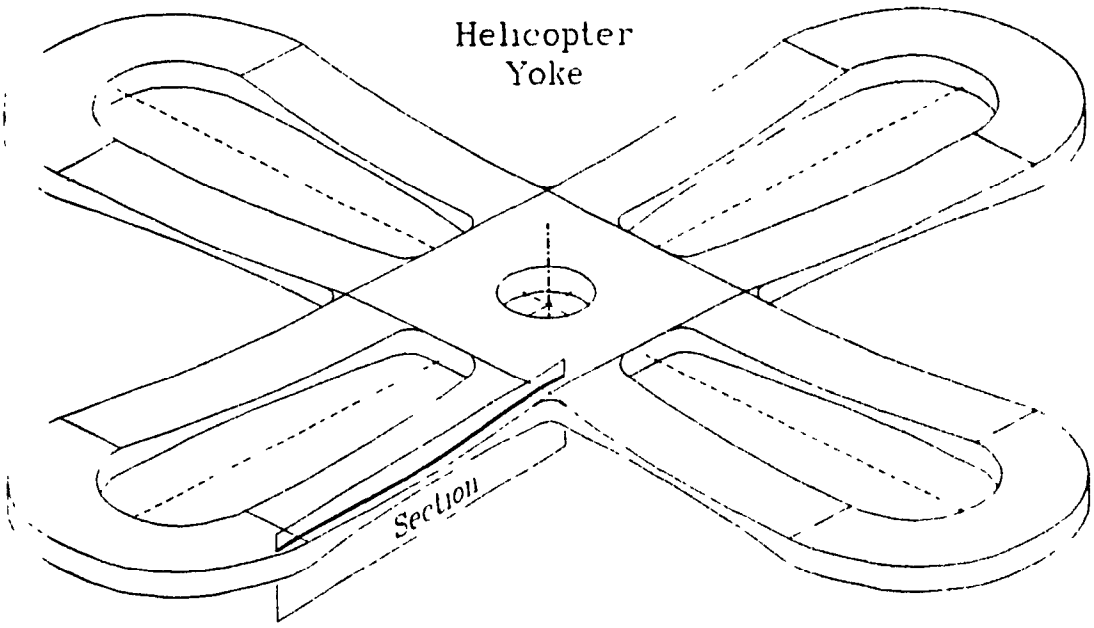
INTRODUCTION

1.1 Objectives

Tapered laminates are finding new applications in flywheels [Kulkarni, Stone, Toland (1978)] and helicopter yokes (Bell/Textron) (Figure 1.1-1.2). A taper is obtained by an external or internal ply drop-off (Figure 1.3). Such drop-offs give rise to interlaminar stresses which are known to account for delamination.

The objectives of this thesis are:

- to obtain an analytical solution for stress around a discontinuity of any triangular shape in an anisotropic plate for cases where plane stress or plane strain assumptions can be made.
- to develop a finite element program for three dimensional stress analysis of tapered laminates on a microcomputer and to generate an understanding of the factors which produce interlaminar stresses.
- to optimize laminates for interlaminar stress reduction.



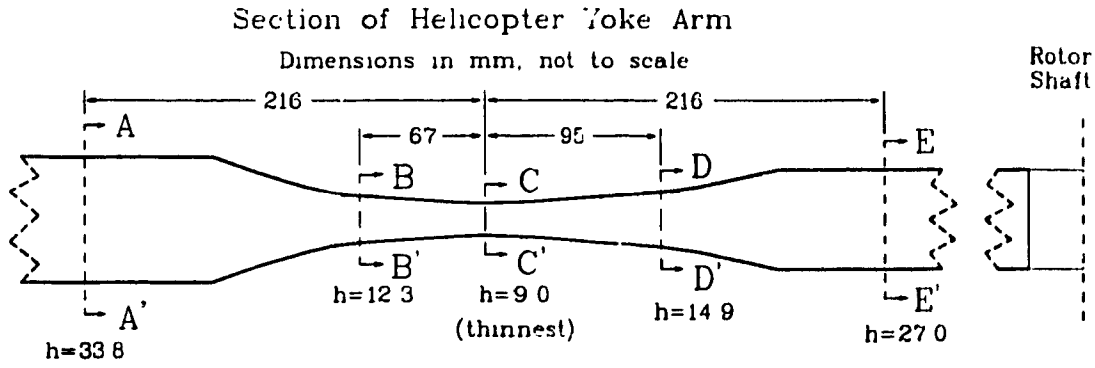


FIGURE 1.1

Helicopter Yoke

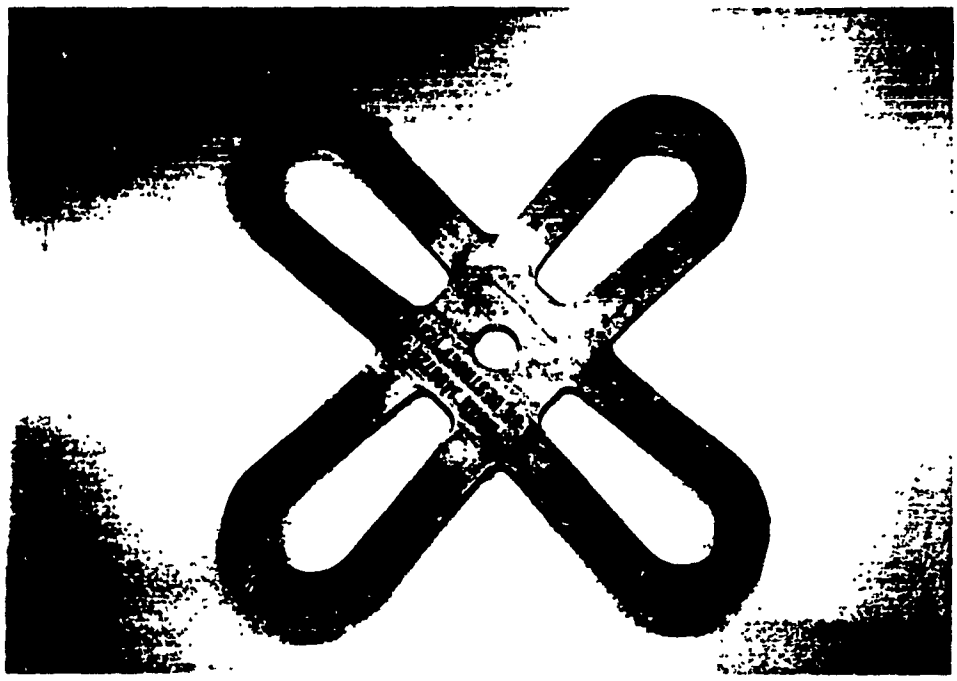


FIGURE 1.2

Photograph of Helicopter Yoke

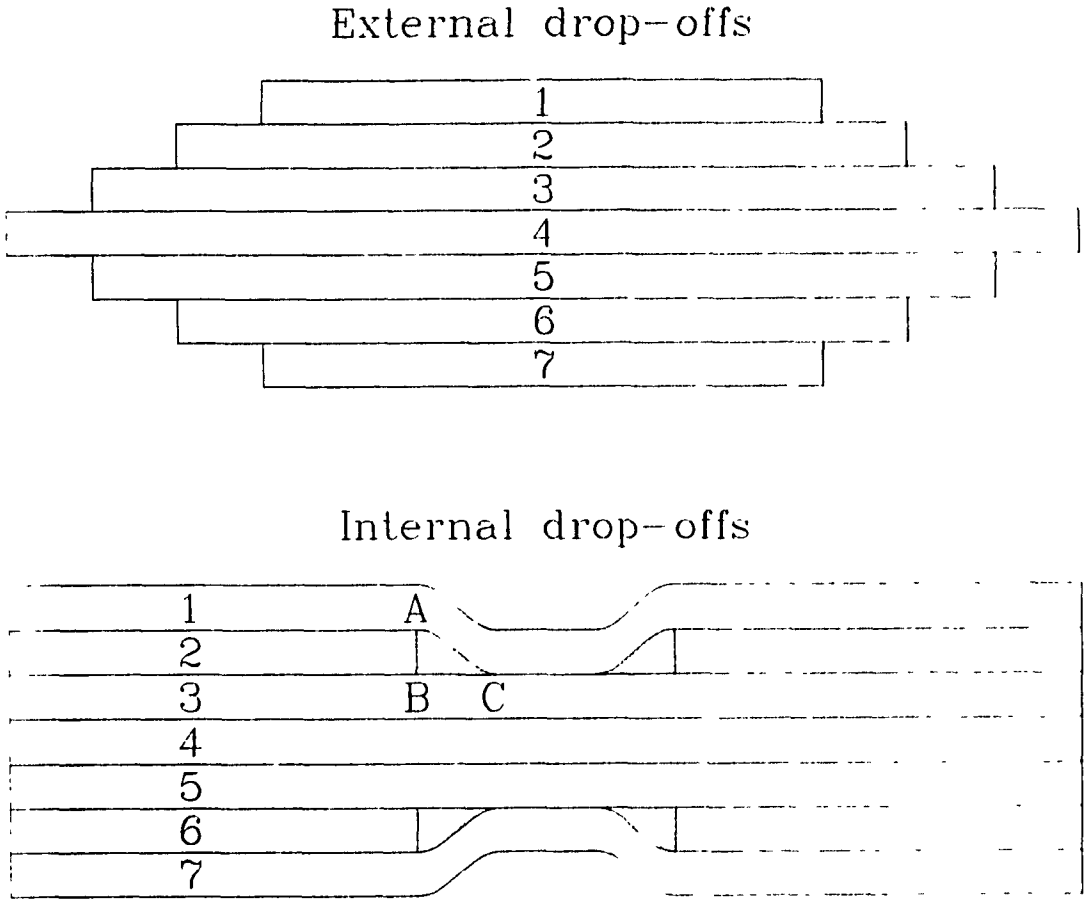


FIGURE 1.3

Drop-off Type

1.2 Background on Interlaminar Stress and Relevant Literature

Up to the present time, few works on stress analysis of composite tapered laminates have been done and these have used mainly the finite element method. Tapered laminates with external drop-offs can be found in the composite flywheel developed by Kulkarni, Stone and Toland (1978). The radial taper was obtained by machining a constant-thickness cylinder leading to external drop-offs. The tapered disk was made of graphite/epoxy (Celanese Celion 6000/5213). A two dimensional axisymmetric finite element analysis was performed assuming plane stress. Neglecting the stepping nature of the drop-off, no interlaminar failure was predicted. The strength predicted using finite elements was higher than experimental results. This is probably due to the fact that interlaminar stresses were not taken into account.

Wu and Webber (1986) and Wu (1987) have analysed tapered laminates with external drop-off using a two dimensional element. They have found that interlaminar stresses (peel and shear) account for delamination and peak in the corner region of the external step. Peel stress values indicated singularity.

In the field of interlaminar stresses within plates, Kassapoglou and Lagace (1987) obtained a closed form solution for the problem of interlaminar stresses at a straight free edge of angle-ply and cross-ply laminates. The solutions were derived by

using the force balance method and the principle of minimum complementary energy. In the case of tapered laminates with drop-offs, the importance of interlaminar stresses arising from the free-edge is second to those arising from the drop-off hole as failure locations from the finite element technique used in this thesis will show.

As shown in Figure 2.2, a real drop-off in a structure like a helicopter yoke does not have any regular shape like a rectangle or a triangle. However, for mathematical treatment of the problem, it is essential to have some mathematical expression for the configuration of the hole in the drop-off region. A shape that most closely resembles the hole in Figure 2.2 is that of a triangle, not with sharp corners but the corners are blunted.

From other considerations, laminates can have different layup configurations like $0^{\circ}/45^{\circ}/-45^{\circ}/90^{\circ}$ etc. In order to provide a solid foundation for the development of the theory, laminates of unidirectional lamina orientation will be considered first. For these unidirectional laminates subjected to uniaxial loading, the problem can be reduced to a plane strain problem and approximate analytical solutions can be obtained. For laminates having multidirectional laminae or for laminates subjected to multidirectional loadings, plane strain or plane stress assumptions can not be made. The problem is fully three dimensional. For those cases, three dimensional finite elements

have to be used to obtain a solution. Real structures such as those of a helicopter yoke can contain fibers with different orientations and planes.

The work in this thesis will therefore consist of three parts. In the first part, laminates made of unidirectional laminae, containing a triangular shaped hole and subjected to uniaxial loading will be modeled as a two dimensional problem. Analytical solution will be obtained for these cases. Subsequently, three dimensional finite elements will be developed and used to calculate stresses in complicated laminates and loadings. Also the analytical results will be compared with finite element results for the two dimensional problems. Finally, the finite element method will be used to generate results and to optimize laminates for minimum interlaminar stresses.

For the two dimensional problems, an equilateral triangle shape function was developed by Savin (1961) which obtained a closed form solution for isotropic material. Lekhnitskii (1968) obtained a solution for an orthotropic plate containing an equilateral triangular hole subjected to a load parallel or perpendicular to one of the triangle side.

Both authors used complex curvilinear coordinates to express the shape of the triangular hole. This seems to be an appropriate approach to the problem. Both previous two-dimensional closed

form solutions are restricted to equilateral triangles.

Pipes and Pagano (1970) showed that interlaminar shear stresses are required to allow shear transfer between layers of the laminate. The drop-off which separates two layers will disable shear transfer and will create a concentration where layers meet. This concentration accounts for the lost shear transfer.

Wu and Webber (1986) found that interlaminar stresses in external drop-off can be reduced by 25% to 50% by including a resin fillet at the corner of the step.

1.3 Thesis Overview

Real drop-offs do not have a distinct shape, but assuming a triangular hole can provide insight on interlaminar stresses around a possible real drop-off. The two dimensional closed form solution will apply to tapered laminates under tension. This solution is developed in chapter 2. The solution is also used to validate the use of finite element in tapered laminates. The finite element method will be able to solve three dimensional models under any loading condition: tension, bending and torsion.

Chapter 3 describes the finite element formulation, used in

chapter 4 to compare with the closed form solution and with experimental results. Chapter 5 uses the appropriate mesh size found, to study many parameters affecting interlaminar stresses. Crack growth simulation is done for tension and bending loading conditions. Optimization of drop-off location and fiber angle at drop-off is performed. Chapter 6 describes the features of the finite element program which was used. In chapter 7, general conclusions about the study are presented along with an outlook for future work.

CHAPTER 2

CLOSED FORM SOLUTION OF STRESS DISTRIBUTION

The helicopter composite yoke, made by Bell helicopter Textron Canada Inc., constitutes the main part of the rotor hub assembly (Figure 1.1-1.2) which is used to take the centrifugal, beamwise, chordwise shear and bending forces from the blades. The yoke is a laminate of some 130 layers, comprising of S2 glass rovings, tape and fabric. The thickness varies from 2.72 cm to 0.91 cm and back to 3.38 cm within a distance of 30 cm going from the inner to outer direction (Figures 1.1, 2.1). This construction is achieved by terminating many laminas inside the laminate. Those lamina drop-offs will create interlaminar stresses which can produce delamination.

FIGURE 2.1

Photograph of Helicopter Yoke Section

In order to study the drop-off geometry in this example of a tapered laminate, a strip was cut from the edge of an arm of the yoke. Figures 2.2 - 2.4 show a drop-off region (dark area) under 3 levels of enlargement. Looking at the first enlargement, the sandwiched layer has 45° fiber (90° points toward the viewer) and the other 2 layers have 0° fiber orientation. The resin appears as black in the micrograph. There is some porosity seen in this picture (2 places).

The micrographs do not allow for a clear determination of the drop-off geometry. Still, the existence of a drop-off is confirmed by an abrupt change in the density of the fiber bundles in the sandwiched layer. A simple triangular drop-off geometry is assumed. This geometry can be reproduced in specimens for testing purposes.

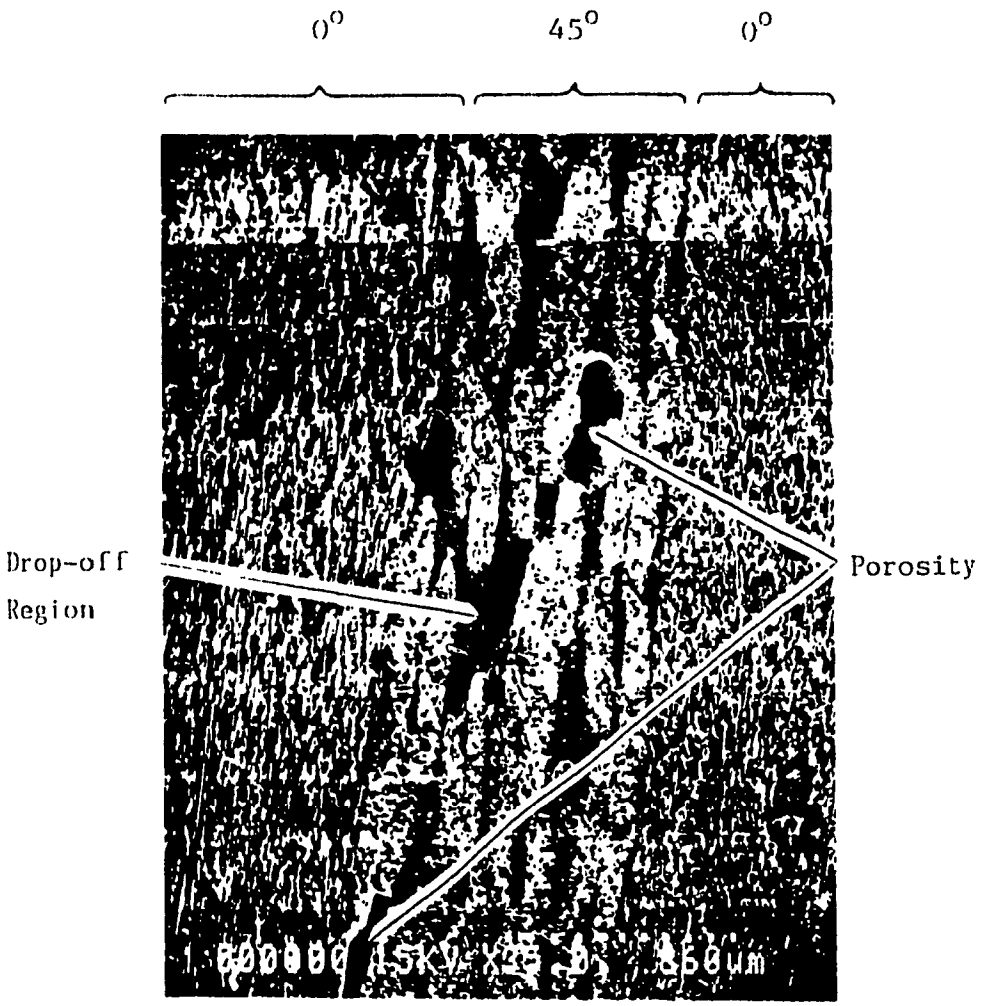


FIGURE 2.2

Micrograph of Helicopter Yoke, 35X

FIGURE 2.3

Micrograph of Helicopter Yoke, 101X

FIGURE 2.4

Micrograph of Helicopter Yoke, 1000X

In this chapter, a closed form solution for the two dimensional stress distribution around a triangular hole is investigated. The desired loading is a stress P in direction a (Figure 2.5) from the x axis of the plate. Savin (1961) and Lekhnitskii (1968) have respectively obtained solutions for equilateral triangular holes in isotropic and orthotropic plates through complex variables. Savin's solution is limited to an equilateral triangular hole in isotropic material. Lekhnitskii's formulation is limited to a load P which is longitudinal or normal to the x axis of an equilateral triangular hole. The closed form solution presented here will account for anisotropic material and accept different length/height ratios of the triangular hole as well as different orientations of the load. In the conclusion of this chapter, Table 2.8 presents the advantages of each author's solution.

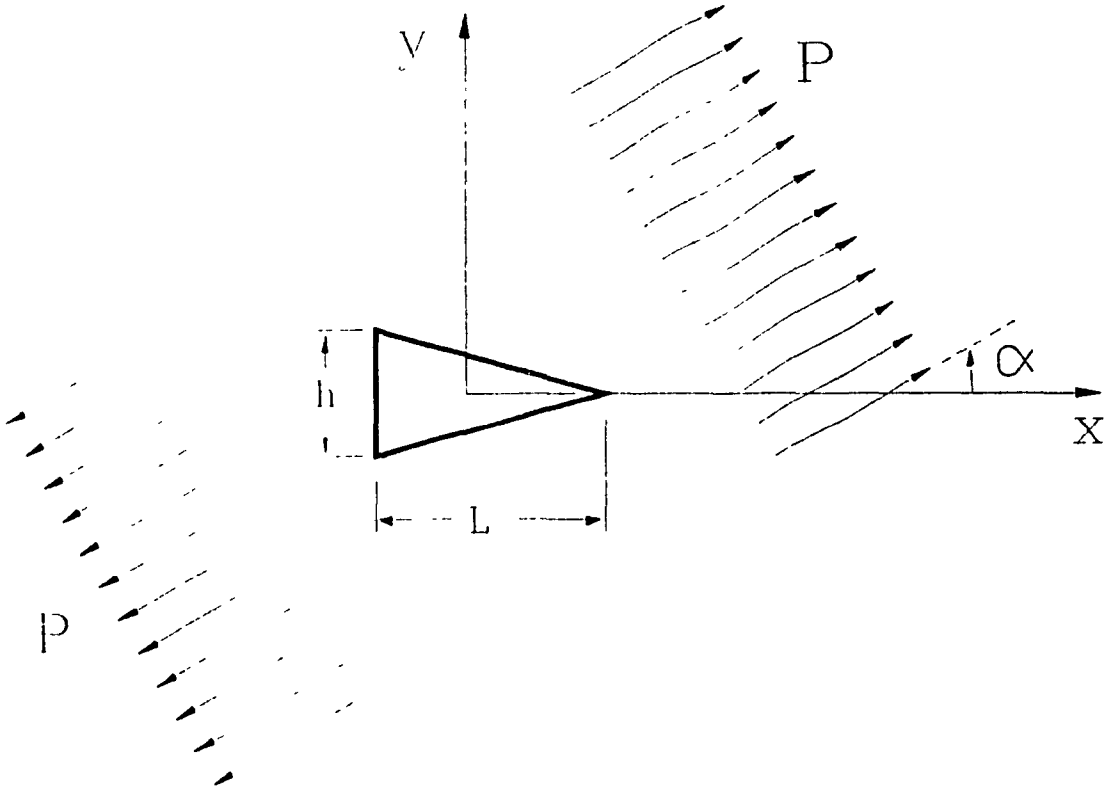


FIGURE 2.5

Loading Direction

2.1 Material Properties

To obtain the material properties of glass/epoxy given in Table 2.1, specimens were made and tested according to the American Standard Testing Method:

-Standard D4255-83 for the inplane shear properties

-Standard D3039-76 for the tensile properties

-Standard D3410-75 for the compressive properties

-Standard D2344-84 for interlaminar shear strength

Compressive results from ASTM standard D3410-75 were duplicated by simple compression of short specimens failing before buckling loads were reached.

TABLE 2.1

Material Properties

CE 9000 Glass/Epoxy	Epoxy, Anhydride-Cured
E_1 = 47.4 GPa	E = 3.45 GPa*
E_2 = 16.2 GPa	ν = 0.35*
G_{12} = 7.0 GPa	$G = \frac{E}{2+2\nu}$ = 1.28 GPa
G_{23} = 1.28 GPa (Epoxy)	X = 72.4 MPa**
ν_{12} = 0.26 (ScotchPly 1002)*	$S = \frac{X}{2}$ = 36.2 MPa
ν_{23} = 0.35 (Epoxy)	
σ_{u_1} = 1283 MPa (Tension), 725 MPa (Compression)	
σ_{u_2} = 27.0 MPa (Tension), 179 MPa (Compression)	
$\sigma_{u_{12}}$ = 56.1 MPa	
$\sigma_{u_{13}}$ = 46.6 MPa	* taken from Tsai & Hahn (1980)
$\sigma_{u_{23}}$ = 36.2 MPa (Epoxy)	** taken from Lubin (1982)

2.2 Triangle Shape Function

A closed form solution for the two dimensional stress distribution around a triangular hole is investigated. The desired loading is a stress P in direction α (Figure 2.5) from the x axis of the plate.

Complex curvilinear coordinate ζ is used. ζ is defined in terms of coordinates ρ and ϑ (Figure 2.6) which constitute a mapping of x and y coordinates. ρ=1 defines the unit circle.

$$\zeta = \rho.e^{i\vartheta} = \rho.\sigma \tag{2.1}$$

$$e^{in\vartheta} = \cos(n\vartheta) + i.\sin(n\vartheta) \tag{2.2}$$

$$e^{-in\vartheta} = \cos(n\vartheta) - i.\sin(n\vartheta) \tag{2.3}$$

The following equilateral triangle shape function was obtained [Savin (1961) Equation (1.32)]. $\omega(\zeta)$ is the complex coordinate of the boundary.

$$\omega(\zeta) = c\left[\frac{1}{\zeta} + \frac{\zeta^2}{3} + \frac{\zeta^5}{45} + \frac{\zeta^8}{162} + \frac{7\zeta^{11}}{2673} + \ldots\right] \tag{2.4}$$

Where c is a constant, equal to the radius of the unit circle R (Figure 2.6). The first term produces a circle. The author's method starts by multiplying the other terms by a constant to enable control over the bluntness of the triangular shape:

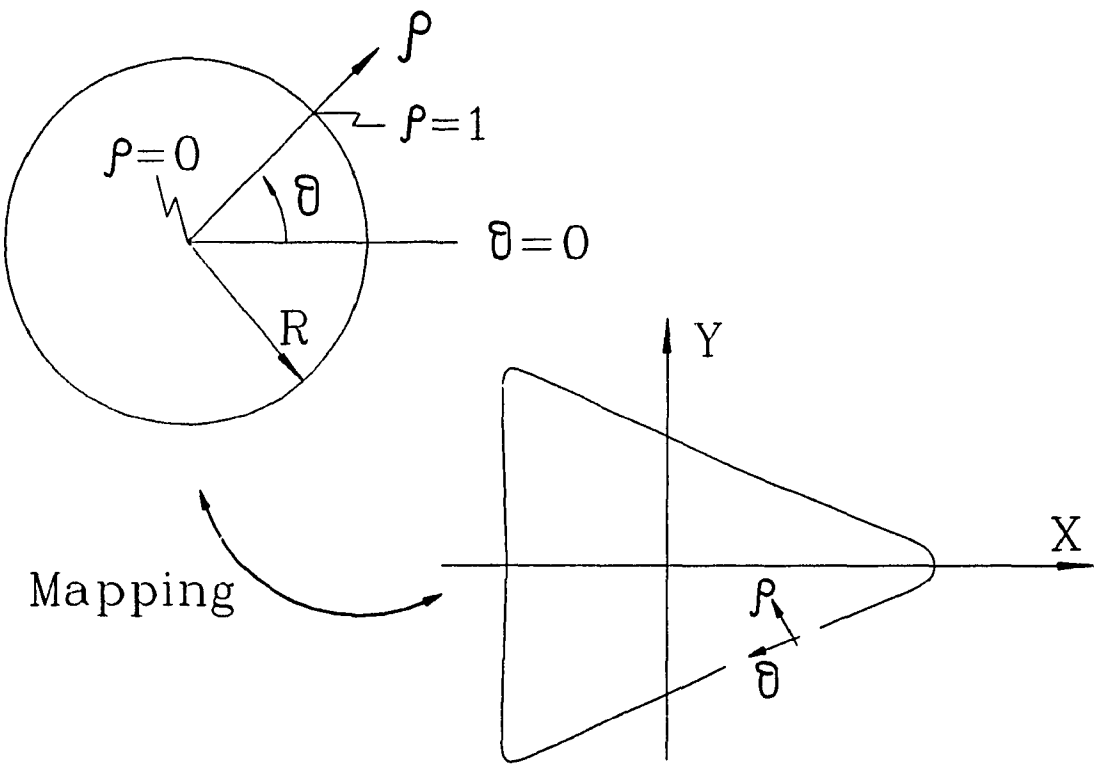


FIGURE 2.6

Curvilinear Coordinate Mapping

$$\omega(\zeta) = R\left[\frac{1}{\zeta} + \varepsilon\left[\zeta^2 + \frac{\zeta^5}{15} + \frac{\zeta^8}{54} + \frac{7\zeta^{11}}{891} + \ldots\right]\right] \qquad (2.5)$$

where $-\frac{1}{3} \leq \varepsilon \leq \frac{1}{3}$

The limits upon ε were obtained from a graphical method. If $\frac{1}{3} < |\varepsilon|$, the mapping is no longer a 1 to 1 function. Negative values of ε produce concave triangle shapes. ε=0 corresponds to a circle.

x is the real part of $\omega(\zeta)$, noted:

$$x = Re[\omega(\zeta)] \qquad (2.6)$$

$$y = Im[\omega(\zeta)]$$

y is the imaginary part of $\omega(\zeta)$. If the first two terms of Equation (2.5) are used one has:

$$\omega(\zeta) = R\left[\frac{1}{\zeta} + \varepsilon\zeta^2\right] \qquad (2.7)$$

where R is the radius of the unit circle. Substituting Equations (2.1) (with ρ=1), (2.2) and (2.3) into Equation (2.7) yields:

$$\omega(\zeta) = R(\cos\theta + \varepsilon.\cos2\theta) - i.R(\sin\theta - \varepsilon.\sin2\theta) \qquad (2.8)$$

$$\longrightarrow x = R(\cos\theta + \varepsilon.\cos2\theta) \qquad (2.9)$$

$$y = -R(\sin\theta - \varepsilon.\sin2\theta) \qquad (2.10)$$

In order to obtain different Length/Height ratios, the x coordinate of Equation (2.9) is multiplied by a variational

constant r, after which the $\omega(\zeta)$ function is reconstructed.

$$x = rR(\cos\theta + \varepsilon.\cos 2\theta) \tag{2.11}$$

$$\omega(\zeta) = rR(\cos\theta + \varepsilon.\cos 2\theta) - i.R(\sin\theta - \varepsilon.\sin 2\theta) \tag{2.12}$$

and

$$r = \frac{2L}{\sqrt{3}h} \quad \text{(for an equilateral triangle: } \frac{L.\sqrt{3}}{h}=\frac{2}{2} \rightarrow r=1\text{)} \tag{2.13}$$

L and h correspond to the length and height of the triangular hole (Figure 2.5). Using the following identities:

$$\cos(n\theta) = \frac{1}{2}\left[\zeta^n + \frac{1}{\zeta^n}\right] \tag{2.14}$$

$$\sin(n\theta) = -\frac{i}{2}\left[\zeta^n - \frac{1}{\zeta^n}\right] \tag{2.15}$$

Equation (2.12) can be written as:

$$\omega(\zeta) = \frac{R}{2}\left[\frac{a}{\zeta} + b\zeta + \varepsilon a\zeta^2 + \frac{\varepsilon b}{\zeta^2}\right] \tag{2.16}$$

where $a = r+1$ and $b = r-1$ (2.17)

From Equation (2.16) one recognizes a pattern similar to Equation (2.7). Extending the number of terms used in Equation (2.5) one arrives at:

$$\omega(\zeta) = \frac{R}{2}\left\{ a\left[\frac{1}{\zeta} + \varepsilon \left[\zeta^2 + \frac{\zeta^5}{15} + \frac{\zeta^8}{54} + \frac{7\zeta^{11}}{891} \right] \right] + b\left[\zeta + \varepsilon \left[\frac{1}{\zeta^2} + \frac{1}{15\zeta^5} + \frac{1}{54\zeta^8} + \frac{1}{891\zeta^{11}} \right] \right] \right\} \qquad (2.18)$$

2 terms → 4 → 6 → 8 → 10

Notice that the second bracket is the conjugate of the first. The second bracket has the physical meaning of contributing to the non-equilateral shape. For an equilateral triangle (r=1 → a=2, b=0) the first bracket returns the same result as Equation (2.5). An equal number of corresponding terms must be taken from each bracket (dotted lines) as the terms in each bracket are arranged in a decreasing order of their contribution. Figure 2.7 shows the effect of ε and of the quantity of terms used for $\omega(\zeta)$. For equilateral triangles (r=1 or $\frac{h}{L}\approx 1.15$) with $\varepsilon=\frac{1}{3}$, the radius of curvature at the summits of the triangle are as follows:

4 terms used: radius = $\frac{R}{21}$

6 terms used: radius = $\frac{R}{58.7}$

8 terms used: radius = $\frac{R}{110.4}$

10 terms used: radius = $\frac{R}{174.3}$ (2.19)

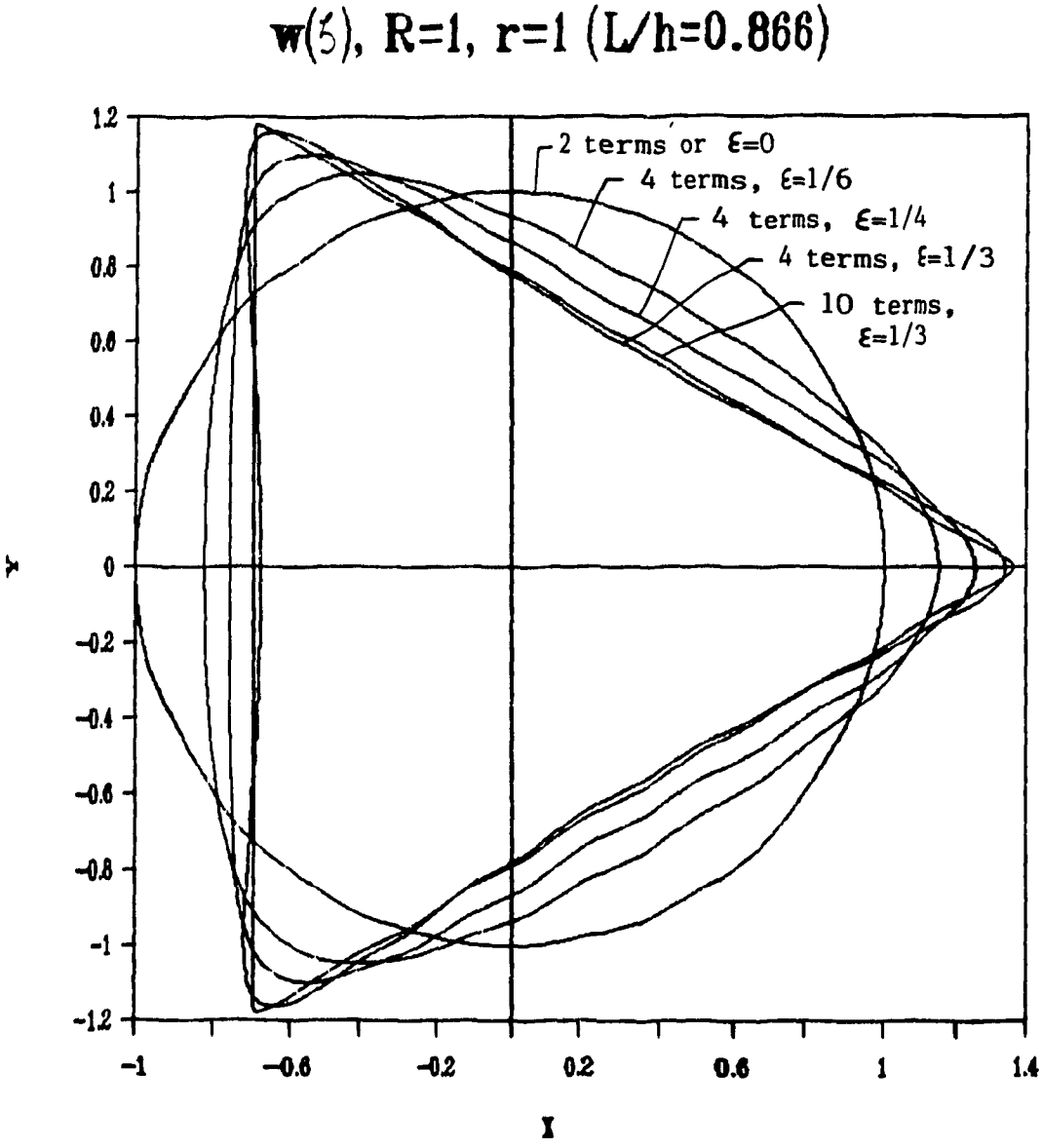


FIGURE 2.7

Shapes for Equilateral Triangles with Different Degree of Bluntness at the Apex

Numbers in Equation (2.19) were obtained numerically by the radius of a circle defined by three points on the hole boundary near $\vartheta=0$. For non-equilateral triangles ($r\neq1$), the radius of curvature of the summit intercepting the x axis was calculated numerically. It was found that the radius of curvature is proportional to the h/L ratio of the triangle.

In general, real drop-offs have been observed to be more blunt than the four term triangular shape with $\varepsilon=\frac{1}{3}$ as in Figure 2.7. Peak stresses obtained with this triangular shape are expected to be higher than at a real drop-off, producing conservative results.

2.3 Analytical Functions in Isotropic Material

The stress in curvilinear coordinates can be expressed as a function of two analytical functions, $\phi(\zeta)$ and $\psi(\zeta)$ [Savin (1961) Equation (1.17)]:

$$\sigma_\rho + \sigma_\vartheta = 2\left[\Phi(\zeta) + \overline{\Phi(\zeta)}\right] \tag{2.20}$$

$$\sigma_\vartheta - \sigma_\rho + 2i\sigma_{\rho\vartheta} = \frac{2\zeta^2}{\rho^2\,\overline{\omega'(\zeta)}}\left[\overline{\omega(\zeta)}\Phi'(\zeta)+\omega'(\zeta)\Psi(\zeta)\right] \tag{2.21}$$

where $$\Phi(\zeta) = \frac{\phi'(\zeta)}{\omega'(\zeta)} \tag{2.22}$$

and $$\Psi(\zeta) = \frac{\psi'(\zeta)}{\omega'(\zeta)} \tag{2.23}$$

To obtain σ_x σ_y σ_{xy} from σ_ρ σ_ϑ $\sigma_{\rho\vartheta}$ or the inverse, the following relations are used to transform the stress according to a rotation of angle λ (Figure 2.8) which is the angle of the ρ axis with respect to the x axis. The orientation of the ρ axis can be obtained numerically by using Equation (2.18), separating it into its real and imaginary part to obtain x and y and giving a small positive variation to ρ.

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} = \begin{bmatrix} m^2 & n^2 & -2mn \\ n^2 & m^2 & 2mn \\ mn & -mn & m^2-n^2 \end{bmatrix} \begin{Bmatrix} \sigma_\rho \\ \sigma_\vartheta \\ \sigma_{\rho\vartheta} \end{Bmatrix} \quad \text{where } m=\cos\lambda \quad n=\sin\lambda \tag{2.24}$$

$$\begin{Bmatrix} \sigma_\rho \\ \sigma_\vartheta \\ \sigma_{\rho\vartheta} \end{Bmatrix} = \begin{bmatrix} m^2 & n^2 & 2mn \\ n^2 & m^2 & -2mn \\ -mn & mn & m^2-n^2 \end{bmatrix} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} \quad \text{where } m=\cos\lambda \quad n=\sin\lambda \tag{2.25}$$

For the loading condition shown in Figure 2.5 the analytical function has the following shape [Savin (1961) Equation (2.20)]:

$$\phi(\zeta) = \frac{P}{4}.\omega(\zeta) + \phi_o(\zeta) \tag{2.26}$$

$$\psi(\zeta) = -\frac{P}{2}.e^{-2i\alpha}.\omega(\zeta) + \psi_o(\zeta) \tag{2.27}$$

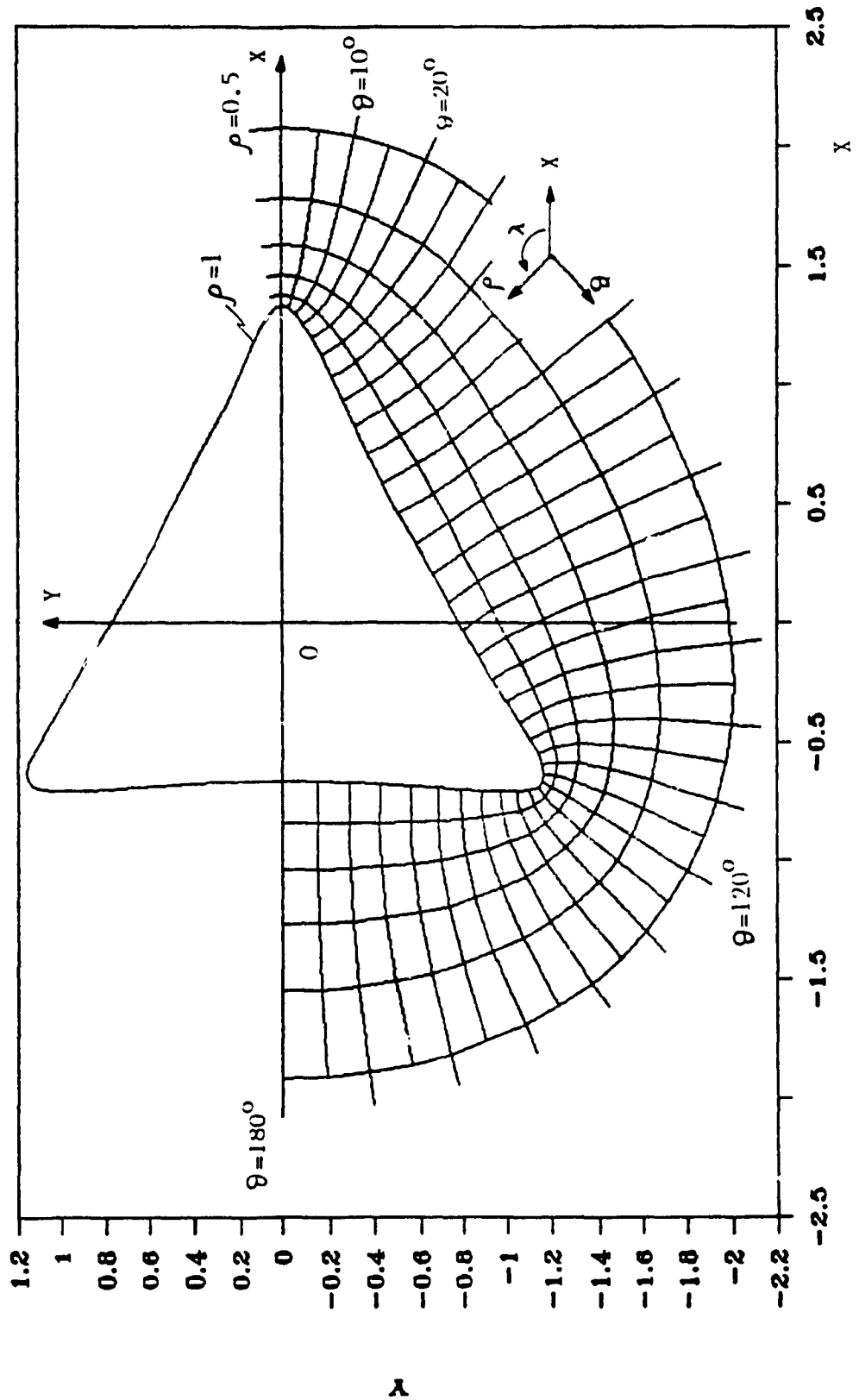


FIGURE 2.8

x, y, ρ, ϑ Coordinates (R=1, r=1, ε=1/3, 4 terms)

where $$\phi_o(\zeta) = \alpha_1\zeta + \alpha_2\zeta^2 + \alpha_3\zeta^3 + \ldots \tag{2.28}$$

$$\psi_o(\zeta) = \beta_0 + \beta_1\zeta + \beta_2\zeta^2 + \beta_3\zeta^3 + \ldots \tag{2.29}$$

$\phi_o(\zeta)$ and $\psi_o(\zeta)$ are evaluated by the following equations [Savin (1961) Equation (2.6)]:

$$\phi_o(\zeta) + \frac{1}{2\pi i}\int_\gamma \frac{\omega(\sigma)}{\overline{\omega'(\sigma)}}\,\overline{\phi_o'(\sigma)}\,\frac{d\sigma}{\sigma-\zeta} + \overline{\beta_o} = \frac{1}{2\pi i}\int_\gamma \frac{f_1^\circ + if_2^\circ}{\sigma-\zeta}\,d\sigma \tag{2.30}$$

$$\psi_o(\zeta) + \frac{1}{2\pi i}\int_\gamma \frac{\overline{\omega(\sigma)}}{\omega'(\sigma)}\,\phi_o'(\sigma)\,\frac{d\sigma}{\sigma-\zeta} = \frac{1}{2\pi i}\int_\gamma \frac{f_1^\circ - if_2^\circ}{\sigma-\zeta}\,d\sigma \tag{2.31}$$

These equations are the integrated form of the solution to a plate with a free edge hole and its conjugate. The integrals are taken over the boundary γ of the hole.

Evaluation of the integrals on the right hand side of Equations (2.30) and (2.31)

For the loading condition shown in Figure 2.5, one has [Savin (1961) Equation (2.21)]:

$$f_1^\circ + if_2^\circ = -\frac{P}{2}\left[\omega(\sigma) - e^{2i\alpha}\,\overline{\omega(\sigma)}\right] \tag{2.32}$$

$$f_1^\circ - if_2^\circ = -\frac{P}{2}\left[\overline{\omega(\sigma)} - e^{-2i\alpha}\,\omega(\sigma)\right] \tag{2.33}$$

Where $(f_1^\circ + if_2^\circ)$ denotes the derivative of the boundary conditions

of functions $\phi_o(\zeta)$ and $\psi_o(\zeta)$ which modify the original stress state (plate without hole) to produce the stress state of the plate with a hole [Savin (1961) pages 42-43]. Using Equation (2.16) for replacing $\omega(\sigma)$ in Equations (2.32) and (2.33) gives:

$$f_1^\circ + if_2^\circ = \frac{-PR}{4}\left[\left(\varepsilon\sigma^2+\frac{1}{\sigma}\right)(a-be^{2i\alpha}) + \left(\sigma+\frac{\varepsilon}{\sigma^2}\right)(b-ae^{2i\alpha})\right] \qquad (2.34)$$

$$f_1^\circ - if_2^\circ = \frac{-PR}{4}\left[\left(\varepsilon\sigma^2+\frac{1}{\sigma}\right)(b-ae^{-2i\alpha}) + \left(\sigma+\frac{\varepsilon}{\sigma^2}\right)(a-be^{-2i\alpha})\right] \qquad (2.35)$$

In order to evaluate the integrals on the right hand side of Equations (2.30) and (2.31), the following identities, known as Cauchy integrals [Muskhelishvili (1963) Equation (70.1) and (70.2)], will be required:

$$\frac{1}{2\pi i}\int_\gamma \sigma^n \frac{d\sigma}{\sigma-\zeta} = \zeta^n \quad \text{where} \quad 0\le n \qquad (2.36)$$

$$\frac{1}{2\pi i}\int_\gamma \frac{1}{\sigma^n}\frac{d\sigma}{\sigma-\zeta} = 0 \quad \text{where} \quad 0<n \qquad (2.37)$$

Multiplying both sides of Equations (2.34) and (2.35) by $\frac{d\sigma}{\sigma-\zeta}$, integrating over γ and dividing by $2\pi i$ yields after taking into account Equations (2.36) and (2.37):

$$\frac{1}{2\pi i}\int_\gamma \frac{f_1^\circ+if_2^\circ}{\sigma-\zeta}\,d\sigma = \frac{-PR}{4}\left[\varepsilon\zeta^2(a-be^{2i\alpha}) + \zeta\,(b-ae^{2i\alpha})\right] \qquad (2.38)$$

$$\frac{1}{2\pi i}\int_\gamma \frac{f_1^\circ-if_2^\circ}{\sigma-\zeta}\,d\sigma = \frac{-PR}{4}\left[\varepsilon\zeta^2(b-ae^{-2i\alpha}) + \zeta\,(a-be^{-2i\alpha})\right] \qquad (2.39)$$

For an axial load (α=0) the integrals reduce to:

$$\frac{1}{2\pi i}\int_{\gamma}\frac{f_1^{\circ}+if_2^{\circ}}{\sigma-\zeta}\,d\sigma = \frac{-PR}{2}\left[\,\varepsilon\zeta^2 - \zeta\,\right] \tag{2.40}$$

$$\frac{1}{2\pi i}\int_{\gamma}\frac{f_1^{\circ}-if_2^{\circ}}{\sigma-\zeta}\,d\sigma = \frac{-PR}{2}\left[\,-\varepsilon\zeta^2 + \zeta\,\right] \tag{2.41}$$

Evaluation of integrals on the left hand side of Equations (2.30) and (2.31)

To evaluate the integrals on the left hand side of Equations (2.30) and (2.31), the derivative and the conjugate of $\omega(\sigma)$ [Equation (2.16)] are found and substituted into the second term on the left hand side of Equation (2.30) and (2.31) and a polynomial d'vision is carried out. Here are the results:

$$\omega(\zeta) = \frac{R}{2}\left[\,a\left(\frac{1}{\zeta} + \varepsilon\zeta^2\right) + b\left(\zeta + \frac{\varepsilon}{\zeta^2}\right)\right] \tag{2.42}$$

$$\omega'(\zeta) = \frac{R}{2}\left[\,a\left(\frac{-1}{\zeta^2} + 2\varepsilon\zeta\right) + b\left(1 - \frac{2\varepsilon}{\zeta^3}\right)\right] \tag{2.43}$$

$$\overline{\omega(\zeta)} = \frac{R}{2}\left[\,a\left(\zeta + \frac{\varepsilon}{\zeta^2}\right) + b\left(\frac{1}{\zeta} + \varepsilon\zeta^2\right)\right] \tag{2.44}$$

$$\overline{\omega'(\zeta)} = \frac{R}{2}\left[\,a\left(-\zeta^2 + \frac{2\varepsilon}{\zeta}\right) + b\left(1 - 2\varepsilon\zeta^3\right)\right] \tag{2.45}$$

$$\frac{\omega(\sigma)}{\overline{\omega'(\sigma)}} = -\frac{a}{2b\sigma} + \frac{(a^2-2b^2)}{4\varepsilon b^2\sigma^2} - \frac{a(a^2-2b^2)}{8\varepsilon^2 b^3\sigma^3} + \ldots \tag{2.46}$$

$$\frac{\overline{\omega(\sigma)}}{\omega'(\sigma)} = \frac{\varepsilon b\sigma^5 + a\sigma^4 + b\sigma^2 + \varepsilon a\sigma}{2\varepsilon a\sigma^4 + b\sigma^3 - a\sigma - 2\varepsilon b} \tag{2.47}$$

As will be shown later, subsequent terms in Equations (2.46) and (2.47) are not necessary for the evaluation of the integrals on the left hand side of Equations (2.30) and (2.31). Now for the remaining terms in these integrals, one has from (2.28) and (2.29):

$$\phi_o'(\zeta) = \alpha_1 + 2\alpha_2\zeta + 3\alpha_3\zeta^2 + \ldots \tag{2.48}$$

$$\overline{\phi_o'(\zeta)} = \overline{\alpha_1} + \frac{2\overline{\alpha_2}}{\zeta} + \frac{3\overline{\alpha_3}}{\zeta^2} + \ldots \tag{2.49}$$

Using Equation (2.46) together with Equation (2.49) for the integral on the left hand side of Equation (2.30)

$$\frac{1}{2\pi i}\int_\gamma \frac{\omega(\sigma)}{\overline{\omega'(\sigma)}}\,\overline{\phi_o'(\sigma)}\,\frac{d\sigma}{\sigma-\zeta} \tag{2.50}$$

$$= \frac{1}{2\pi i}\int_\gamma \left[-\frac{a}{2b\sigma} + \frac{(a^2-2b^2)}{4\varepsilon b^2\sigma^2} + \ldots\right]\left[\overline{\alpha_1} + \frac{2\overline{\alpha_2}}{\sigma} + \ldots\right]\frac{d\sigma}{\sigma-\zeta}$$

Using Equations (2.36) and (2.37) it can be seen that:

$$\frac{1}{2\pi i}\int_{\gamma} \frac{\omega(\sigma)}{\overline{\omega'(\sigma)}}\,\overline{\phi_o'(\sigma)}\,\frac{d\sigma}{\sigma-\zeta} = 0 \qquad (2.51)$$

Using Equations (2.28), (2.50) and (2.38), Equation (2.30) can now be written as:

$$(\alpha_1\zeta+\alpha_2\zeta^2+\alpha_3\zeta^3)+0+\overline{\beta_o} = \frac{-PR}{4}\left[\ \varepsilon\zeta^2(a-be^{2i\alpha})+\zeta\ (b-ae^{2i\alpha})\right] \qquad (2.52)$$

from which the constants α_i and $\overline{\beta_o}$ can be determined as:

$$\alpha_1 = \frac{-PR}{4}(b-ae^{2i\alpha}) \qquad (2.53)$$

$$\alpha_2 = \frac{-PR\varepsilon}{4}\ (a-be^{2i\alpha}) \qquad (2.54)$$

$$\alpha_3 = 0 \quad \text{and} \quad \overline{\beta_0} = 0 \qquad (2.55)$$

Inserting Equations (2.53) to (2.55) into Equation (2.28) and using Equation (2.16), Equation (2.26) can be written as:

$$\phi(\zeta) = \frac{PR}{8}\left[\zeta^2\varepsilon(2be^{2i\alpha}-a) + \zeta(2ae^{2i\alpha}-b) + \frac{a}{\zeta} + \frac{b\varepsilon}{\zeta^2}\right] \qquad (2.56)$$

For $\alpha=0$: $$\phi(\zeta) = \frac{PR}{8}\left[\ \zeta^2\varepsilon(r-3) + \zeta(r+3) + \frac{a}{\zeta} + \frac{b\varepsilon}{\zeta^2}\right] \qquad (2.57)$$

For $r=1$: $$\phi(\zeta) = \frac{PR}{4}\left[\ -\varepsilon\zeta^2 + 2\zeta e^{2i\alpha} + \frac{1}{\zeta}\right] \qquad (2.58)$$

For $r=1$, $\alpha=0$: $$\phi(\zeta) = \frac{PR}{4}\left[\ -\varepsilon\zeta^2 + 2\zeta + \frac{1}{\zeta}\right] \qquad (2.59)$$

For the case of an equilateral triangle hole (r=1) and $\varepsilon=\frac{1}{3}$, Equations (2.58) and (2.59) agreed with Savin's solution [Savin (1961) Equation (2.55) and (2.56)]. The procedure to handle Equation (2.31) follows the same way as that for Equation (2.30). Inserting Equations (2.47) and (2.48) into the first integral of Equation (2.31) yields:

$$\frac{1}{2\pi i}\int_{\gamma}\frac{\overline{\omega(\sigma)}}{\omega'(\sigma)}\,\phi_o'(\sigma)\,\frac{d\sigma}{\sigma-\zeta}$$
$$=\frac{1}{2\pi i}\int_{\gamma}\left[\frac{\varepsilon b\sigma^5+a\sigma^4+b\sigma^2+\varepsilon a\sigma}{2\varepsilon a\sigma^4+b\sigma^3-a\sigma-2\varepsilon b}\right]\left[\alpha_1+2\alpha_2\sigma\right]\frac{d\sigma}{\sigma-\zeta} \qquad (2.60)$$

Using Equations (2.36) and (2.37) one has:

$$\frac{1}{2\pi i}\int_{\gamma}\frac{\overline{\omega(\sigma)}}{\omega'(\sigma)}\,\phi_o'(\sigma)\frac{d\sigma}{\sigma-\zeta}=\left[\alpha_1+2\alpha_2\zeta\right]\left[\frac{\varepsilon b\zeta^5+a\zeta^4+b\zeta^2+\varepsilon a\zeta}{2\varepsilon a\zeta^4+b\zeta^3-a\zeta-2\varepsilon b}\right] \qquad (2.61)$$

Replacing α_1 and α_2 in Equation (2.61) by their value [Equations (2.53), (2.54)] yields:

$$\frac{1}{2\pi i}\int_{\gamma} \frac{\overline{\omega(\sigma)}}{\omega'(\sigma)}\,\phi_o'(\sigma)\,\frac{d\sigma}{\sigma-\zeta} = -\frac{PR}{4}\left[D\zeta+E\right] \tag{2.62}$$

where

$$B = 2\varepsilon a\zeta^4+b\zeta^3-a\zeta-2\varepsilon b$$

$$C = b-ae^{2i\alpha}$$

$$D = a-be^{2i\alpha}$$

$$E = \frac{1}{B}\left[2D\varepsilon^2 b\zeta^6+C\varepsilon b\zeta^5+\left[Ca-Db\right]\zeta^4+2D\varepsilon b\zeta^3+\left[Cb+2D\varepsilon^2 a+Da\right]\zeta^2+\left[C\varepsilon a+2D\varepsilon b\right]\zeta\right]$$

Inserting Equations (2.29), (2.62) and (2.39) into Equation (2.31) one has:

$$\beta_0 + \beta_1\zeta + \beta_2\zeta^2 + \beta_3\zeta^3 - \frac{PR}{4}\left[D\zeta+E\right] = -\frac{PR}{4}\left[\varepsilon F\zeta^2+G\zeta\right] \tag{2.63}$$

where $F = b-ae^{-2i\alpha}$

$G = a-be^{-2i\alpha}$

Matching terms of similar power of ζ yields

$$\beta_0 = \frac{PR}{4}\,E \tag{2.64}$$

$$\beta_1 = -\frac{PR}{4}\,(G-D) \tag{2.65}$$

$$\beta_2 = -\frac{PR}{4}\,\varepsilon F \tag{2.66}$$

$$\beta_3 = 0 \tag{2.67}$$

This yields:

$$\psi_o(\zeta) = \beta_0 + \beta_1\zeta + \beta_2\zeta^2 \tag{2.68}$$

Substituting Equations (2.64) to (2.68) and Equation (2.16) into Equation (2.27) yields:

$$\psi(\zeta) = -\frac{PR}{4}\left[\varepsilon\left(F+ae^{-2i\alpha}\right)\zeta^2+\left(G-D+be^{-2i\alpha}\right)\zeta-E+\frac{ae^{-2i\alpha}}{\zeta}+\frac{bce^{-2i\alpha}}{\zeta^2}\right] \quad (2.69)$$

$$\text{For } r=1: \quad \psi(\zeta) = -PR\left[\frac{e^{-2i\alpha}}{2\zeta} + \frac{e^{2i\alpha}\zeta^3-(2\varepsilon^2+1)\zeta+\varepsilon e^{2i\alpha}}{4\varepsilon\zeta^3-2}\right] \quad (2.70)$$

$$\text{For } r=1,\ \varepsilon=\frac{1}{3}: \quad \psi(\zeta) = -PR\left[\frac{e^{-2i\alpha}}{2\zeta} + \frac{9e^{2i\alpha}\zeta^3-11\zeta+3e^{2i\alpha}}{12\zeta^3-18}\right] \quad (2.71)$$

$$\text{For } r=1,\ \varepsilon=\frac{1}{3},\ \alpha=0: \quad \psi(\zeta) = -PR\left[\frac{1}{2\zeta} + \frac{9\zeta^3-11\zeta+3}{12\zeta^3-18}\right] \quad (2.72)$$

For the case of an equilateral triangle hole (r=1) with $\varepsilon=\frac{1}{3}$, Equations (2.71) and (2.72) agreed with Savin's solution [Savin (1961) Equation (2.55) and (2.56)].

2.4 Analytical Functions in Anisotropic Material

For the loading conditions shown in Figure 2.5, the stress around a hole in an anisotropic plate can be expressed in the following manner [Savin (1961) Equation (3.25)]:

$$\sigma_x = P\cos^2\alpha + 2.\mathrm{Re}[s_1^2\phi_o'(z_1)+s_2^2\psi_o'(z_2)] \quad (2.73)$$

$$\sigma_y = P\sin^2\alpha + 2.\mathrm{Re}[\phi_o'(z_1)+\psi_o'(z_2)] \quad (2.74)$$

$$\sigma_{xy} = P\sin\alpha.\cos\alpha - 2.\mathrm{Re}[s_1\phi_o'(z_1)+s_2\psi_o'(z_2)] \quad (2.75)$$

For anisotropic materials, the stress - strain relation can be expressed in the following manner:

$$\sigma_x = A_{11}\varepsilon_x + A_{12}\varepsilon_y + A_{13}\varepsilon_z + A_{14}\gamma_{yz} + A_{15}\gamma_{xz} + A_{16}\gamma_{xy} \qquad (2.76)$$

$$\sigma_y = A_{21}\varepsilon_x + A_{22}\varepsilon_y + A_{23}\varepsilon_z + A_{24}\gamma_{yz} + A_{25}\gamma_{xz} + A_{26}\gamma_{xy}$$

$$\sigma_z = A_{31}\varepsilon_x + A_{32}\varepsilon_y + A_{33}\varepsilon_z + A_{34}\gamma_{yz} + A_{35}\gamma_{xz} + A_{36}\gamma_{xy}$$

$$\tau_{yz} = A_{41}\varepsilon_x + A_{42}\varepsilon_y + A_{43}\varepsilon_z + A_{44}\gamma_{yz} + A_{45}\gamma_{xz} + A_{46}\gamma_{xy}$$

$$\tau_{xz} = A_{51}\varepsilon_x + A_{52}\varepsilon_y + A_{53}\varepsilon_z + A_{54}\gamma_{yz} + A_{55}\gamma_{xz} + A_{56}\gamma_{xy}$$

$$\tau_{xy} = A_{61}\varepsilon_x + A_{62}\varepsilon_y + A_{63}\varepsilon_z + A_{64}\gamma_{yz} + A_{65}\gamma_{xz} + A_{66}\gamma_{xy}$$

$$A_{ij} = A_{ji} \quad (i, j = 1, 2, 3, 4, 5, 6)$$

where A_{ij} are components of the stiffness matrix. By inverting the matrix of A_{ij} coefficients, one obtains components of the compliance c_{ij} where:

$$[c] = [A]^{-1} \qquad (2.77)$$

The origin of s_1 and s_2 in Equations (2.73) to (2.75) comes from the equation of equilibrium for an anisotropic body [Lekhnitskii (1968) Equation (7.1) where s_1 and s_2 are noted as μ_1 and μ_2]. The equation of equilibrium can be written as:

$$a_{22}\frac{\partial^4 U}{\partial x^4} - 2a_{26}\frac{\partial^4 U}{\partial x^3\,\partial y} + (2a_{12}+a_{66})\frac{\partial^4 U}{\partial x^2\,\partial y^2} \qquad (2.78)$$

$$- 2a_{16}\frac{\partial^4 U}{\partial x\,\partial y^3} + a_{11}\frac{\partial^4 U}{\partial y^4} = 0$$

where $U = U(x,y)$ is a stress function. Compliance components a_{ij} are given in Equation (2.84). A plane stress or a plane strain case can be considered by using the appropriate values of a_{ij}.

This equation can be integrated in its general form by writing it symbolically with the use of four linear differential operators of the first order:

$$D_1 D_2 D_3 D_4 U = 0 \tag{2.79}$$

D_i (i=1,2,3,4) designates the operation:

$$D_i = \frac{\partial}{\partial y} - s_i \frac{\partial}{\partial x} \tag{2.80}$$

where s_i are the roots of the characteristic equation

$$a_{11}s^4 - 2a_{16}s^3 + (2a_{12}+a_{66})s^2 - 2a_{26}s + a_{22} = 0 \tag{2.81}$$

The roots of this equation can be written as:

$$\begin{aligned} s_1 &= \alpha_1 + i\beta_1 \qquad 0 \leq \beta_1 \\ s_2 &= \alpha_2 + i\beta_2 \qquad 0 \leq \beta_2 \\ s_3 &= \alpha_1 - i\beta_1 \\ s_4 &= \alpha_2 - i\beta_2 \end{aligned} \tag{2.82}$$

For isotropic materials $a_{16}=a_{26}=0$ so that Equation (2.81) is reduced to

$$a_{11}s^4 + (2a_{12}+a_{66})s^2 + a_{22} = 0 \tag{2.83}$$

which is the same as a second degree equation in terms of s^2. For

isotropic material one finds that $\beta_1=\beta_2$.

The constants in Equation (2.81) are as follows:

For a plane strain case ($\varepsilon_z = \gamma_{xz} = \gamma_{yz} = 0$):

$$\begin{aligned}
a_{11} &= (c_{11}c_{33}-c_{13}^2)/c_{33} \\
a_{12} &= (c_{12}c_{33}-c_{13}c_{23})/c_{33} \\
a_{16} &= (c_{16}c_{33}-c_{13}c_{36})/c_{33} \\
a_{22} &= (c_{22}c_{33}-c_{23}^2)/c_{33} \\
a_{26} &= (c_{26}c_{33}-c_{23}c_{36})/c_{33} \\
a_{66} &= (c_{66}c_{33}-c_{36}^2)/c_{33}
\end{aligned} \tag{2.84}$$

For a plane stress case: $a_{ij} = c_{ij}$

where c_{ij} are components of [c] in equation (2.77)

Evaluation of $\phi_o(z_1)$ and $\psi_o(z_2)$

The analytical functions $\phi_o(z_1)$ and $\psi_o(z_2)$ seen in Equations (2.73) through (2.75) are obtained by substituting ζ with z_1 and z_2 respectively in $\Phi_o(\zeta)$ and $\Psi_o(\zeta)$. The need for two variables, z_1 and z_2 , comes from the fact that anisotropic materials will produce four s_i roots [Equation (2.82)] dependent upon two constants, β_1 and β_2. Analytical functions for isotropic materials only depend upon one variable as $\beta_1=\beta_2$. The analytical functions are expressed as [Savin (1961) Equation (3.18)]:

$$\Phi_o(\zeta) = \frac{i}{4\pi(s_1-s_2)} \int_\gamma (s_2 f_1^\circ - f_2^\circ) \frac{\sigma+\zeta}{\sigma-\zeta} \frac{d\sigma}{\sigma} \qquad (2.85)$$

$$\Psi_o(\zeta) = \frac{-i}{4\pi(s_1-s_2)} \int_\gamma (s_1 f_1^\circ - f_2^\circ) \frac{\sigma+\zeta}{\sigma-\zeta} \frac{d\sigma}{\sigma} \qquad (2.86)$$

where the boundary condition functions [Savin (1961) Equation (3.20)] arising from the hole's contribution to an unweakened plate are

$$f_1^\circ = -2Re[B^* z_1 + (B'^* + iC'^*) z_2] \qquad (2.87)$$

$$f_2^\circ = -2Re[B^* s_1 z_1 + s_2 (B'^* + iC'^*) z_2] \qquad (2.88)$$

and

$$B^* = P \frac{\cos^2\alpha + (\alpha_2^2+\beta_2^2)\sin^2\alpha + \alpha_2 \sin 2\alpha}{2\,[(\alpha_2-\alpha_1)^2+(\beta_2^2-\beta_1^2)]} \qquad (2.89)$$

$$B'^* = P \frac{[(\alpha_1^2-\beta_1^2)-2\alpha_1\alpha_2]\sin^2\alpha - \cos^2\alpha - \alpha_2 \sin 2\alpha}{2\,[(\alpha_2-\alpha_1)^2+(\beta_2^2-\beta_1^2)]} \qquad (2.90)$$

$$C'^* = \frac{P}{2\beta_2\,[(\alpha_2-\alpha_1)^2+(\beta_2^2-\beta_1^2)]} \Big[(\alpha_1-\alpha_2)\cos^2\alpha \qquad (2.91)$$

$$+ [\alpha_2(\alpha_1^2-\beta_1^2) - \alpha_1(\alpha_2^2-\beta_2^2)] \sin^2\alpha$$

$$+ [(\alpha_1^2-\beta_1^2) - (\alpha_2^2-\beta_2^2)] \sin\alpha.\cos\alpha \Big]$$

z_1 is obtained by the affine transformation

$$z_1 = x + s_1 . y \qquad (2.92)$$

Using x and y in Equations (2.11) and (2.10), one has:

$$z_1 = R[r(\cos\theta + \varepsilon.\cos 2\theta) - s_1(\sin\theta - \varepsilon.\sin 2\theta)] \qquad (2.93)$$

z_1 can be expressed in terms of ζ by using Equations (2.14) and (2.15).

$$z_1 = \frac{R}{2}\left[a_1\zeta + \frac{b_1}{\zeta} + \varepsilon b_1\zeta^2 + \frac{a_1\varepsilon}{\zeta^2} \right] \tag{2.94}$$

where $a_j = r+i.s_j$

$$b_j = r-i.s_j \qquad j=1,2 \tag{2.95}$$

similarly,

$$z_2 = \frac{R}{2}\left[a_2\zeta + \frac{b_2}{\zeta} + \varepsilon b_2\zeta^2 + \frac{a_2\varepsilon}{\zeta^2} \right] \tag{2.96}$$

Substituting Equations (2.94) and (2.96) into Equation (2.87) one obtains:

$$f_1^\circ = -2.\mathrm{Re}\left[k_1\zeta + \frac{k_2}{\zeta} + k_3\zeta^2 + \frac{k_4}{\zeta^2} \right] \tag{2.97}$$

$$\begin{aligned} k_1 &= \frac{R}{2}\left[B^* a_1 + (B'^* + i.C'^*)a_2\right] \\ k_2 &= \frac{R}{2}\left[B^* b_1 + (B'^* + i.C'^*)b_2\right] \\ k_3 &= \varepsilon k_2 \\ k_4 &= \varepsilon k_1 \end{aligned} \tag{2.98}$$

Substituting Equations (2.94) and (2.96) into Equation (2.88) one obtains:

$$f_2^\circ = -2.\mathrm{Re}\left[k_5\zeta + \frac{k_6}{\zeta} + k_7\zeta^2 + \frac{k_8}{\zeta^2} \right] \tag{2.99}$$

$$k_5 = \frac{R}{2}\,[B^{*}s_1a_1+s_2(B'^{*}+i.C'^{*})a_2]$$

$$k_6 = \frac{R}{2}\,[B^{*}s_1b_1+s_2(B'^{*}+i.C'^{*})b_2]$$

$$k_7 = \varepsilon k_6$$

$$k_8 = \varepsilon k_5 \qquad (2.100)$$

In order to evaluate the integrals of Equations (2.85) and (2.86), the Schwarz formula is required [Muskhelishvili (1963) Equation (77.7)]:

$$F(\zeta) = \frac{1}{2\pi i}\int_{\gamma} U(\vartheta)\,\frac{\sigma+\zeta}{\sigma-\zeta}\,\frac{d\sigma}{\sigma} + i\alpha_o \qquad (2.101)$$

Where $U(\vartheta)$ is the value of the real part of the function $F(\zeta)$ on the contour of the unit circle; α_o is some real constant. The Schwarz formula allows one to obtain:

$$\frac{i}{4\pi(s_1-s_2)}\int_{\gamma} 2a.\mathrm{Re}\left[\,k_1\sigma^{n}+\frac{k_2}{\sigma^{n}}\,\right]\frac{\sigma+\zeta}{\sigma-\zeta}\,\frac{d\sigma}{\sigma} = \frac{-a}{s_1-s_2}\,(k_1+\overline{k_2})\zeta^{n} \qquad (2.102)$$

Substituting ζ for σ (ρ=1) in Equations (2.97) and (2.99) one can integrate Equations (2.85) and (2.86):

$$\Phi_o(\zeta)=\frac{\zeta}{s_1-s_2}\left[s_2(k_1+\overline{k_2})-(k_5+\overline{k_6})\right]+\frac{\zeta^{2}\varepsilon}{(s_1-s_2)}\left[s_2(k_2+\overline{k_1})-(k_6+\overline{k_5})\right] \qquad (2.103)$$

$$\Psi_o(\zeta)=\frac{\zeta}{s_2-s_1}\left[s_1(k_1+\overline{k_2})-(k_5+\overline{k_6})\right]+\frac{\zeta^{2}\varepsilon}{(s_2-s_1)}\left[s_1(k_2+\overline{k_1})-(k_6+\overline{k_5})\right] \qquad (2.104)$$

2.4.1 Analytical Technique

The last step is to obtain ζ in terms of z_1 to substitute into Equation (2.103) and ζ in terms of z_2 for Equation (2.104). When attempting to express ζ in terms of z_1 from Equation (2.94), the root for a fourth degree equation must be obtained. Here, the analytical technique is used to obtain $\zeta=f(z_1)$. Considering a general fourth degree equation:

$$\zeta^4 + \Gamma_1\zeta^3 + \Gamma_2\zeta^2 + \Gamma_3\zeta + \Gamma_4 = 0 \qquad (2.105)$$

The zeros of this equation are the roots of the following equation [Zaguskin (1961) and Spiegel (1981) Equation (9.7), noting the error of $\pm$ instead of $\mp$ in the third term]:

$$\zeta^2 + \frac{1}{2}\left[\Gamma_1 \pm \sqrt{\Gamma_1^2-4\Gamma_2+4y_1}\right]\zeta + \frac{1}{2}\left[y_1 \mp \sqrt{y_1^2-4\Gamma_4}\right] = 0 \qquad (2.106)$$

where y_1 is a real root of:

$$y^3 - \Gamma_2 y^2 + (\Gamma_1\Gamma_3-4\Gamma_4)y + (4\Gamma_2\Gamma_4-\Gamma_3^2-\Gamma_1^2\Gamma_4) = 0 \qquad (2.107)$$

The $\mp$ sign in the second term indicates that the opposite sign of the $\pm$ in the first term must be used. The third degree Equation (2.107) is solved in the following manner [Spiegel (1981) Equation (9.3)]:

$$y^3 + \mathbb{B}_1 y^2 + \mathbb{B}_2 y + \mathbb{B}_3 = 0 \qquad (2.108)$$

$$y_1 = \mathbb{S}+\mathbb{T}-\mathbb{B}_1/3 \qquad (2.109)$$

$$\mathbb{S} = [\mathbb{R}+(\mathbb{Q}^3+\mathbb{R}^2)^{1/2}]^{1/3} \qquad (2.110)$$

$$\mathbb{T} = [\mathbb{R}-(\mathbb{Q}^3+\mathbb{R}^2)^{1/2}]^{1/3} \qquad (2.111)$$

$$\mathbb{Q} = (3\mathbb{B}_2-\mathbb{B}_1^2)/9 \qquad (2.112)$$

$$\mathbb{R} = (9\mathbb{B}_1\mathbb{B}_2-27\mathbb{B}_3-2\mathbb{B}_1^3)/54 \qquad (2.113)$$

In this case one has:

$$\mathbb{B}_1 = -\Gamma_2$$

$$\mathbb{B}_2 = \Gamma_1\Gamma_3 - 4\Gamma_4$$

$$\mathbb{B}_3 = 4\Gamma_2\Gamma_4 - \Gamma_3^2 - \Gamma_1^2\Gamma_4 \qquad (2.114)$$

Substituting Equation (2.114) into Equation (2.112) yields:

$$\mathbb{Q} = (3\Gamma_1\Gamma_3 - 12\Gamma_4 - \Gamma_2^2)/9 \qquad (2.115)$$

$$\begin{aligned}\mathbb{Q}^3 = 1/729.(&27\Gamma_1^3\Gamma_3^3 - 1728\Gamma_4^3 - \Gamma_2^6 - 324\Gamma_1^2\Gamma_3^2\Gamma_4 \\ &-27\Gamma_1^2\Gamma_2^2\Gamma_3^2 - 432\Gamma_2^2\Gamma_4^2 + 1296\Gamma_1\Gamma_3\Gamma_4^2 \\ &+9\Gamma_1\Gamma_2^4\Gamma_3 - 36\Gamma_2^4\Gamma_4 + 216\Gamma_1\Gamma_2^2\Gamma_3\Gamma_4)\end{aligned} \qquad (2.116)$$

Substituting Equation (2.114) into Equation (2.113):

$$\mathbb{R} = 1/54.(-9\Gamma_1\Gamma_2\Gamma_3 - 72\Gamma_2\Gamma_4 + 27\Gamma_3^2 + 27\Gamma_1^2\Gamma_4 + 2\Gamma_2^3) \qquad (2.117)$$

$$\begin{aligned}\mathbb{R}^2 = 1/2916.(&81\Gamma_1^2\Gamma_2^2\Gamma_3^2 + 5184\Gamma_2^2\Gamma_4^2 + 729\Gamma_3^4 + 729\Gamma_1^4\Gamma_4^2 + 4\Gamma_2^6 \\ &+1296\Gamma_1\Gamma_2^2\Gamma_3\Gamma_4 - 486\Gamma_1\Gamma_2\Gamma_3^3 - 486\Gamma_1^3\Gamma_2\Gamma_3\Gamma_4 - 36\Gamma_1\Gamma_2^4\Gamma_3 \\ &-3888\Gamma_2\Gamma_3^2\Gamma_4 - 3888\Gamma_1^2\Gamma_2\Gamma_4^2 - 288\Gamma_2^4\Gamma_4 + 1458\Gamma_1^2\Gamma_3^2\Gamma_4 \\ &+108\Gamma_2^3\Gamma_3^2 + 108\Gamma_1^2\Gamma_2^3\Gamma_4)\end{aligned} \qquad (2.118)$$

$$\begin{aligned}\mathbb{Q}^3 + \mathbb{R}^2 = 1/108.(&4\Gamma_1^3\Gamma_3^3 - 256\Gamma_4^3 + 6\Gamma_1^2\Gamma_3^2\Gamma_4 - \Gamma_1^2\Gamma_2^2\Gamma_3^2 \\ &+128\Gamma_2^2\Gamma_4^2 + 192\Gamma_1\Gamma_3\Gamma_4^2 - 16\Gamma_2^4\Gamma_4 + 80\Gamma_1\Gamma_2^2\Gamma_3\Gamma_4 + 27\Gamma_3^4 \\ &+27\Gamma_1^4\Gamma_4^2 - 18\Gamma_1\Gamma_2\Gamma_3^3 - 18\Gamma_1^3\Gamma_2\Gamma_3\Gamma_4 - 144\Gamma_2\Gamma_3^2\Gamma_4 \\ &-144\Gamma_1^2\Gamma_2\Gamma_4^2 + 4\Gamma_2^3\Gamma_3^2 + 4\Gamma_1^2\Gamma_2^3\Gamma_4)\end{aligned} \qquad (2.119)$$

Transforming Equation (2.94) into a fourth degree equation:

$$\zeta^4 + \left[\frac{a_1}{b_1\varepsilon}\right]\zeta^3 + \left[\frac{-2z_1}{R\varepsilon b_1}\right]\zeta^2 + \frac{\zeta}{\varepsilon} + \left[\frac{a_1}{b_1}\right] = 0 \qquad (2.120)$$

$$\text{let} \quad c_1 = a_1/b_1 \quad d_1 = \frac{-2}{R\varepsilon b_1} \qquad (2.121)$$

one can substitute

$$\Gamma_1 = \frac{c_1}{\varepsilon} \quad \Gamma_2 = d_1 z_1 \quad \Gamma_3 = \frac{1}{\varepsilon} \quad \Gamma_4 = c_1 \qquad (2.122)$$

Equations (2.117) and (2.119) can be simplified by using Equation (2.122):

$$\mathbb{R} = V_{11} + V_{21}z_1 + V_{31}z_1^3 \qquad (2.123)$$

$$\mathbb{Q}^3 + \mathbb{R}^2 = V_{41} + V_{51}z_1 + V_{61}z_1^2 + V_{71}z_1^3 + V_{81}z_1^4 \qquad (2.124)$$

$$V_{11} = \frac{1}{2\varepsilon^2}\left[1 + c_1^3\right]$$

$$V_{21} = \frac{-c_1 d_1}{6}\left[8 + \frac{1}{\varepsilon^2}\right]$$

$$V_{31} = 1/27.\, d_1^3$$

$$V_{41} = \frac{1}{108}\left[\left[\frac{27}{\varepsilon^4}\right]c_1^6 + \left[\frac{4}{\varepsilon^6} + \frac{6}{\varepsilon^4} + \frac{192}{\varepsilon^2} - 256\right]c_1^3 + \frac{27}{\varepsilon^4}\right]$$

$$V_{51} = -c_1 d_1\left[1 + c_1^3\right]\left[\frac{1}{6\varepsilon^4} + \frac{4}{3\varepsilon^2}\right]$$

$$V_{61} = \frac{c_1^2 d_1^2}{108}\left[128 - \frac{1}{\varepsilon^4} + \frac{80}{\varepsilon^2}\right]$$

$$V_{71} = \frac{d_1^3}{27\varepsilon^2}\left[1 + c_1^3\right]$$

$$V_{81} = \frac{-4}{27} c_1 d_1^4 \qquad (2.125)$$

Here, the last subscript, 1, is referring to z_1. The solution is identical for z_2, only the last subscript should be changed to 2,

for example:

$$V_{42} = \frac{1}{108}\left[\left(\frac{27}{\varepsilon^4}\right)c_2^6 + \left(\frac{4}{\varepsilon^6} + \frac{6}{\varepsilon^4} + \frac{192}{\varepsilon^2} - 256\right)c_2^3 + \frac{27}{\varepsilon^4}\right]$$

$$\text{with} \quad c_2 = a_2/b_2 \quad d_2 = \frac{-2}{R\varepsilon b_2}$$

The root of Equation (2.108) is now found:

$$y_1 = \left[(V_{11}+V_{21}z_1+V_{31}z_1^3) + (V_{41}+V_{51}z_1+V_{61}z_1^2+V_{71}z_1^3+V_{81}z_1^4)^{1/2}\right]^{1/3} + \left[(V_{11}+V_{21}z_1+V_{31}z_1^3) - (V_{41}+V_{51}z_1+V_{61}z_1^2+V_{71}z_1^3+V_{81}z_1^4)^{1/2}\right]^{1/3} + \frac{d_1 z_1}{3} \tag{2.126}$$

By using Equations (2.106), (2.122) and (2.126) yields ζ in terms of z_1:

$$\zeta = \frac{1}{4}\left[-m_1 \pm \left(m_1^2 - 8n_1\right)^{1/2}\right] \tag{2.127}$$

$$m_1 = \frac{c_1}{\varepsilon} \pm \left(\frac{c_1^2}{\varepsilon^2} - 4d_1z_1 + 4y_1\right)^{1/2}$$

$$n_1 = y_1 \mp \left(y_1^2 - 4c_1\right)^{1/2} \quad \text{(inverse sign of m1)} \tag{2.128}$$

Equation (2.127) can be used with Equation (2.103) to obtain $\phi_o(z_1)$. Replacing a_1 and b_1 with a_2 and b_2 respectively in Equation (2.121) will produce ζ in terms of z_2. Then, $\psi_o(z_2)$ is known. Stresses σ_x σ_y σ_{xy} can be evaluated by numerically differentiating $\phi_o(z_1)$ and $\psi_o(z_1)$ and using Equations (2.73) to

(2.75). Derivation with respect to z_j is done as if z_j was a real number. The following rule can then be used:

$$f'(z) \underset{\lim \Delta x \to 0}{=} \frac{f(z+\Delta x) - f(z)}{\Delta x} \quad \text{where } z=x+iy \tag{2.129}$$

example:

$$f(z) = az^2 + bz + c \tag{2.130}$$

$$f(z+\Delta x) = a(x^2+\Delta x^2-y^2+2x\Delta x+2xyi+2\Delta xyi) + b(x+\Delta x+iy) +c$$

$$f(z) = a(x^2 -y^2 +2xyi) + b(x +iy) +c$$

$$f(z+\Delta x)-f(z) = a(\Delta x^2+2x\Delta x+2\Delta xyi) + b(\Delta x)$$

$$f'(z) = [f(z+\Delta x)-f(z)]/\Delta x$$

$$= a(\Delta x+2x+2yi)+b = a(2x+2yi)+b \quad \text{as } \Delta x \to 0$$

$$= 2az+b \quad \text{(known solution)}$$

Using Equation (2.127) to analytically obtain the roots of Equation (2.120) has the following limitation: y_1 must be a real number. Giving a range of ϑ and ρ, values of y_1 are calculated using Equation (2.126), it was found that this limitation reduces the valid domain to the approximate region (using the conditions of $\varepsilon=\frac{1}{3}$, $\alpha=0$, CE 9000 material) of only two points: $\rho=1$ and $\vartheta=0^\circ$ or 180°. In addition, in order to obtain $\sigma_\rho=0$ at these two locations the appropriate signs for Equations (2.127) and (2.128) were found numerically and are shown in Table 2.2. If other signs are used, no sensible solution can be obtained. Obviously, there are stresses outside this region, but the roots of Equation (2.120) must be obtained numerically.

TABLE 2.2

Specific Signs for Analytical $\zeta=f(z)$ ($\varepsilon=1/3$, $\alpha=0$, CE 9000, ρ 1)

	$\vartheta=0^{\circ}$	$\vartheta=180^{\circ}$
Sign for ζ in Equation (2.127)	$\pm \rightarrow -$	$\pm \rightarrow -$
Sign for m_1 in Equation (2.128)	$\pm \rightarrow -$	$\pm \rightarrow +$
Sign for n_1 in Equation (2.128)	$\mp \rightarrow +$	$\mp \rightarrow -$

2.4.2 Numerical Technique

A difficulty in using any numerical method is in controlling the root selection. The roots can be controlled through the Bairstow method [Burden and Faires (1985)]. The Bairstow method is used to obtain the roots of a polynomial with real coefficients. Here the method is extended to polynomials with complex coefficients. A polynomial as in Equation (2.120) can be written as

$$p(\zeta) = A_0.\zeta^N + A_1.\zeta^{N-1} + \ldots + A_{N-1}.\zeta + A_N = 0 \qquad (2.131)$$

$p(\zeta)$ is a polynomial with complex coefficients:

$$A_j = x_j + i.y_j \qquad (2.132)$$

Equation (2.131) can be expressed as

$$p(\zeta) = (\zeta^2 - r\zeta - s).q(\zeta) + \mathcal{R}\zeta + \mathcal{S} \qquad (2.133)$$

Our objective is to obtain r, s and $q(\zeta)$ with $\mathcal{R}, \mathcal{S} \to 0$. Where $\mathcal{R}$, $\mathcal{S}$, r and s are complex numbers. One set of roots will be

$$\zeta = \frac{1}{2}.[r \pm (r^2+4s)^{1/2}] \qquad (2.134)$$

The root resulting from the + sign in Equation (2.134) is known as the first root, the other one is known as the second root. With a guess on the initial values of r_0 and δ_0 the procedure is the following:

Repeat the following until $|\Delta r|+|\Delta\delta| \leq$ Desired Precision

$$\begin{aligned}
&\text{Set } b_{-1} = b_{-2} = c_{-1} = c_{-2} = 0 \\
&\text{For k=0 to N do} \\
&\quad b_k = A_k + r \cdot b_{k-1} + \delta \cdot b_{k-2} \\
&\quad c_k = b_k + r \cdot c_{k-1} + \delta \cdot c_{k-2} \qquad (2.135) \\
&\text{Improve } r, \delta \ (m^{th} \text{ iteration, } \Delta r = r_m - r_{m-1} \quad \Delta\delta = \delta_m - \delta_{m-1}): \\
&\quad J = c_{N-2}^2 - c_{N-1} \cdot c_{N-3} \\
&\quad r_m = r_{m-1} - (b_{N-1} \cdot c_{N-2} - b_N \cdot c_{N-3})/J \\
&\quad \delta_m = \delta_{m-1} + (b_{N-1} \cdot c_{N-1} - b_N \cdot c_{N-2})/J \qquad (2.136)
\end{aligned}$$

Obtain roots with 2.125

Obtain the deflated polynomial $q(\zeta) = Q_0 \cdot \zeta^{N-2} + \ldots + Q_{N-2}$:

$$\begin{aligned}
&\text{For k=0 to N-2 do} \qquad (2.137) \\
&\quad Q_k = b_k
\end{aligned}$$

Starting with the expression of $p(\zeta)$ in Equation (2.131), the above algorithm gives the values of r, δ and an expression for $q(\zeta)$ such that $\mathcal{R}=\mathcal{S}=0$. The above r, δ will give two roots ζ of $p(\zeta)$ as shown in Equation (2.134). In order to find the remaining roots of $p(\zeta)$ which are contained in $q(\zeta)$, the same algorithm is applied to $q(\zeta)$ in the same manner as it was applied to $p(\zeta)$.

Using the above procedure, inputting a set of coefficients $A_0 - A_N$ will produce $\frac{N}{2}$ pairs (for even N) of (r_1, δ_1), (r_2, δ_2), etc. which can be used to find the N roots of $p(\zeta)$.

A benefit of finding the roots of Equation (2.120) numerically is that the degree of the polynomial is not restricted like an analytical method, therefore a higher order triangle shape function could be used [see Equation (2.18)].

2.4.3 Root Selection

Both methods still require a selection of the appropriate root out of the four available. Root selection was not discussed by Savin (1961) and Lekhnitskii (1968).

For root selection, it is important to start the solution of the problem at points on the boundary where the criterion of $\sigma_\rho=0$ can be used. Out of the four roots, only one will satisfy this condition.

For points away from the boundary ($\rho<1$), there is no criterion for the stress. However, assuming stress continuity, it is important to use small increments in ρ (with constant θ). This way, values of n, Δ obtained from the previous solution at the neighbouring point can be used as input in the algorithm [Equations (2.135) to (2.137)] to obtain slightly modified values of n, Δ. These in turn will give the stress solution for the point under consideration.

Stress Determination

After σ_x, σ_y, σ_{xy} are obtained from Equations (2.73) to (2.75), stress σ_ρ, σ_ϑ, $\sigma_{\rho\vartheta}$ can be obtained using Equation (2.25).

2.4.4 Example

Consider a plate with a triangular hole as shown in Figure 2.5 made of CE 9000 material and α=0, P=1Pa, $\varepsilon=\frac{1}{3}$, R=1, r=1. It is desired to obtain the stress distribution around the hole.

For the material properties given in Table 2.1, one can obtain the a_{ij} coefficients from Equation (2.84) for a plane strain case with CE 9000 material:

$$
\begin{aligned}
a_{11} &= 2.06096E\text{-}11 \ Pa^{-1} \\
a_{12} &= -7.40506E\text{-}12 \ Pa^{-1} \\
a_{16} &= 0 \ Pa^{-1} \\
a_{22} &= 5.41667E\text{-}11 \ Pa^{-1} \\
a_{26} &= 0 \ Pa^{-1} \\
a_{66} &= 1.42857E\text{-}10 \ Pa^{-1}
\end{aligned}
\qquad (2.138)
$$

Using Equation (2.81) and the numerical technique, one obtains

$$
\begin{aligned}
s_1 &= 0 + i(2.3992) \\
s_2 &= 0 + i(0.6757)
\end{aligned}
\qquad (2.139)
$$

For comparison purpose, the same Equation (2.81) but using different materials was also solved. Table 2.3 presents results for these other materials. Note that any isotropic material will produce the same set of s_1 and s_2.

TABLE 2.3

Material Properties and s_1 and s_2 for Different Materials

	Isotropic Steel	CE 9000 Glass/Epoxy	Plywood[1]	T300 5208[2] Graphite/Epoxy
E_L(MPa)	207	47.4	11.79	181
E_T(MPa)	207	16.2	5.89	10.3
ν_{LT}	0.3	0.26	0.071	0.28
ν_{TT}	0.3[3]	0.35[4]	0.3[5]	0.35[4]
G_{LT}(MPa)	79.3	7.0	0.69	7.17
G_{TT}(MPa)	79.3[6]	1.28[4]	0.5[5]	1.28[4]
Plane Strain				
s_1	i	2.3992i	4.1032i	4.8940i
s_2	i	0.6757i	0.3293i	0.8042i
Plane Stress				
s_1	i	2.3962i	4.1019i	4.8939i
s_2	i	0.7138i	0.3449i	0.8566i

Notes (for material properties only)

1: taken from Lekhnitskii (1968) Equation (11.9) and (11.10)
2: taken from Tsai & Hahn (1980) table 1.7
3: $\nu_{TT} = \nu_{LT}$ for isotropic material
4: taken from epoxy material [Lubin (1982)]
5: assumed (but does not influence s_1 and s_2 significantly)
6: $G_{TT} = G_{LT}$ for isotropic material

The numerical method is selected to calculate the stresses. For each selected point on the hole boundary (ρ=1) the following procedure is used to evaluate the stress state:

1- select a coordinate: (ϑ, ρ) on the hole boundary (ρ=1).

2- obtain (x,y) coordinates by using $\omega(\zeta)=f(\zeta)$ Equation (2.18). Only 4 terms are used in this example.

3- obtain z_1 and z_2 with Equation such as (2.92).

4- Use the numerical algorithm to obtain values of η and α which produce all the 4 roots of ζ in Equation (2.120). There are 4 roots of ζ for z_1 and 4 roots of ζ for z_2.

5- obtain 4 roots of ζ with Equation (2.134).

6- obtain derivative of each of the 4 roots of ζ with respect to z_1 and z_2 by considering $z_1^+=z_1+\Delta x$ and using the previously obtained η and α numbers for each ζ root (step 4) so that the resulting ζ^+ will contribute to the derivative of ζ: $\zeta'=(\zeta^+-\zeta)/\Delta x$. The value of $|\Delta\eta|+|\Delta\alpha|$ in the numerical algorithm [Equation (2.135) - (2.137)] should be a few orders of magnitude smaller than the variation of η and α produced by Δx.

7- obtain $\phi_o'(z_1)$ and $\psi_o'(z_2)$ by the derivation of Equations (2.103) and (2.104) with respect to ζ.

8- evaluate σ_x , σ_y , and σ_{xy} with Equations (2.73) to (2.75).

9- transform σ_x , σ_y , σ_{xy} into σ_ϑ , σ_ρ , $\sigma_{\vartheta\rho}$ with Equation (2.25)

10- select one pair of values of η and α (step 4) and either the first or second root for ζ so that σ_ρ=0 [Equation (2.134)].

These values will be used later for the determination of stresses at $\rho<1$.

Following the above procedure, the problem is solved. Table 2.4 presents the values of η and δ for the roots of $\zeta=f(z_1)$ and $\zeta=f(z_2)$ using the numerical algorithm.

TABLE 2.4

η, δ Values for ζ Roots at $\vartheta=0^\circ$, $\rho=1$, $\alpha=0$, P=1, Equilateral, $\varepsilon=\frac{1}{3}$

	$\zeta=f(z_1)$		$\zeta=f(z_2)$	
	η_1	δ_1	η_2	δ_2
1	**1.158288**	**-0.158288**	1.145448	0.070933
2	0.076591	2.600497	**-1.726041**	**-2.726041**

Out of the 4 possible combinations of η and δ for $\zeta=f(z_1)$ and $\zeta=f(z_2)$ at $\vartheta=0^\circ$, $\rho=1$, using the first line of Table 2.4 for η_1 and δ_1 and the last for η_2 and δ_2 (highlighted) produced the lowest absolute value of σ_ρ^* (normalized stress $\sigma_\rho \div P$) for $\varepsilon=\frac{1}{3}$ and CE9000 glass/epoxy material. The error on σ_ρ^* was in the order of 1×10^{-6} which corresponds to the accuracy of η and δ in table 2.4.

Table 2.5 gives the selected roots for three critical locations: $\vartheta=0^\circ$, $\vartheta=\vartheta_0$ and $\vartheta=180^\circ$. ϑ_0 is the angle at which the tangent to the hole boundary is parallel to the x axis. It is

also the location where σ_ϑ is maximum [Lekhnitskii (1968)]. This location is obtained by $\partial y/\partial\vartheta=0$ applied to Equation (2.16):

$$\cos\vartheta_0-2\varepsilon.\cos2\vartheta_0=0 \qquad \text{(4 term triangle)} \qquad (2.140)$$

TABLE 2.5

ζ Roots for Critical Points

(ρ=1, ε=1/3, CE9000 glass/epoxy, α=0, 4 term shape)

	ϑ	0°	$\vartheta_0=115.17^\circ$	180°
$\zeta=f(z_1)$	n	1.1583	-0.3020+0.8827i	-0.8687
	a	-0.1583	0.0322-0.1211i	0.1313
	ζ	1.0000	-0.4253+0.9050i	-1.0000
	root[1]	first	first	second
$\zeta=f(z_2)$	n	-1.7260	0.4926+0.9003i	-1.0681
	a	2.7260	-0.0329+0.0589i	-0.0681
	ζ	1.0000	-0.4253+0.9050i	-1.0000
	root[1]	first	first	second

1: first and second root of ζ refers to the sign + or - used in (2.134)

Table 2.6 presents the values of stresses for each 10° increment of ϑ (where $\sigma'_{ij}=\sigma_{ij}\div P$). The highest stress concentration factor is at $\vartheta_0=115.17^\circ$, ρ=1, where $\sigma'_\vartheta=\sigma'_x=12.131$. σ'_y reaches 1.569 at $\vartheta=120^\circ$, ρ=1 while σ'_{xy} reached -2.718 at $\vartheta=120^\circ$, ρ=1. Stress plots are shown in Figures 2.9a to 2.14b.

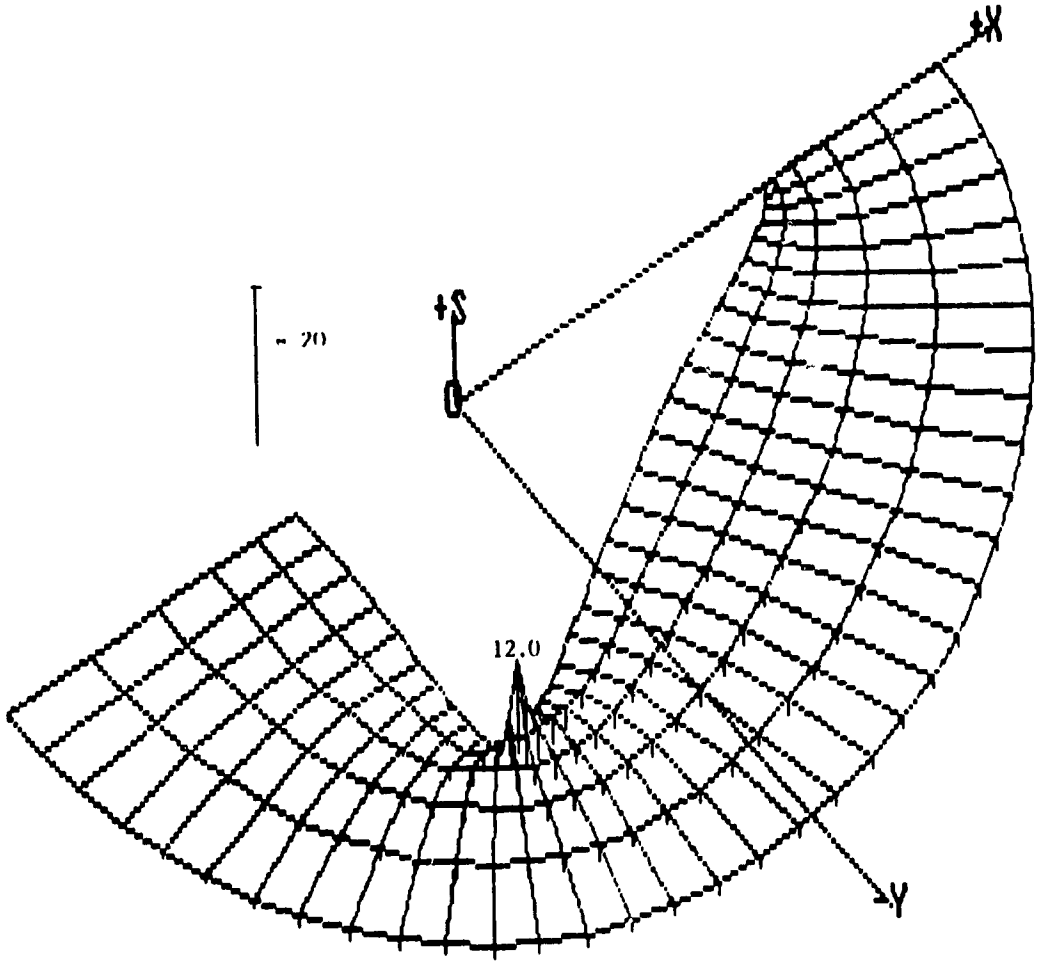


FIGURE 2.9a

σ_x^* for $\varepsilon=1/3$, R=1, $\alpha=0$, Equilateral, CE 9000 material
($\Delta\theta=5^\circ$, $\Delta\rho=0.1$)

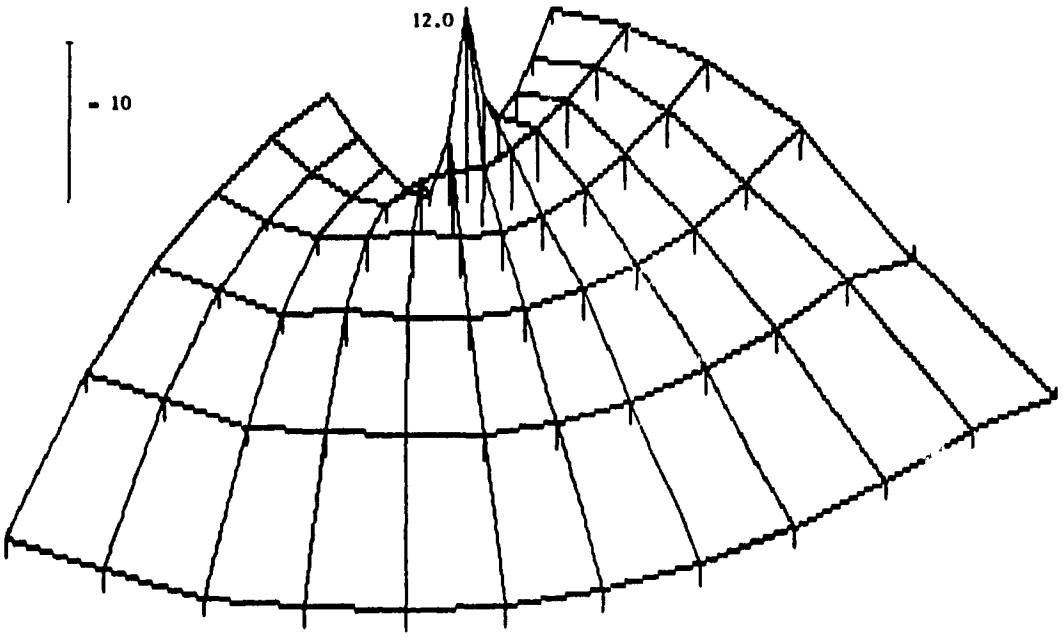


FIGURE 2.9b

σ_x^* Enlargement ($90^\circ \leq \theta \leq 145^\circ$)

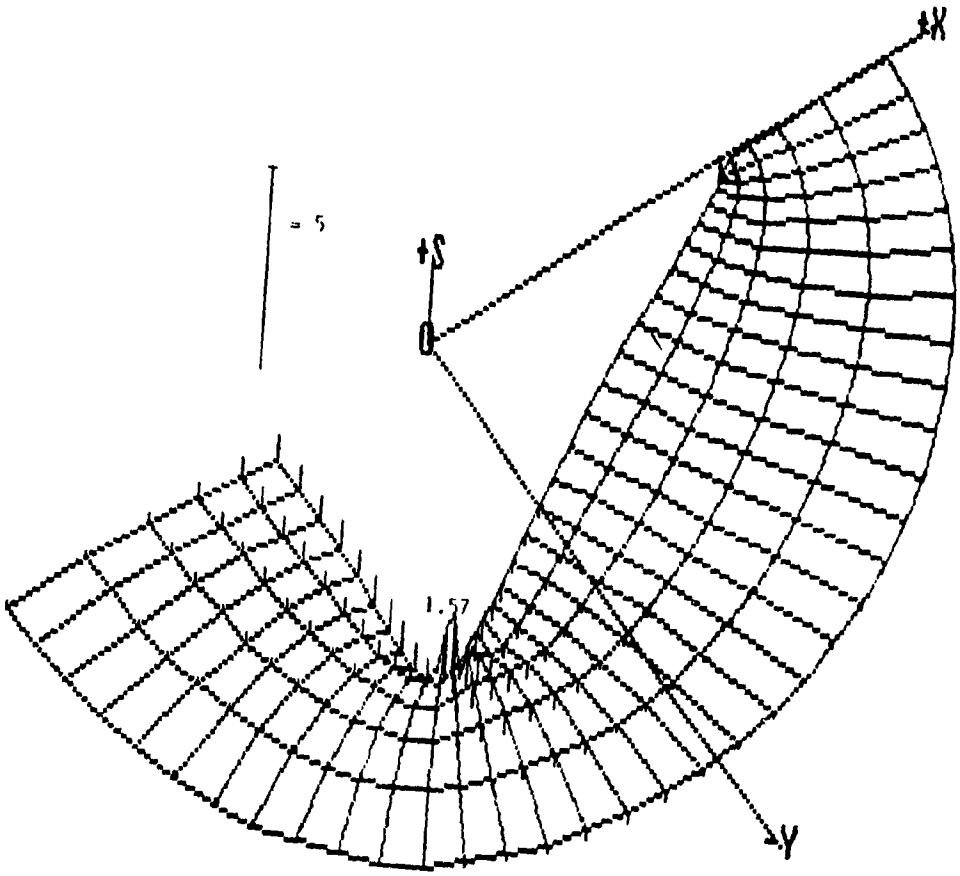


FIGURE 2.10a

σ_y^* for ε=1/3, R=1, α=0, Equilateral, CE 9000 material
($\Delta\theta$=5°, $\Delta\rho$=0.1)

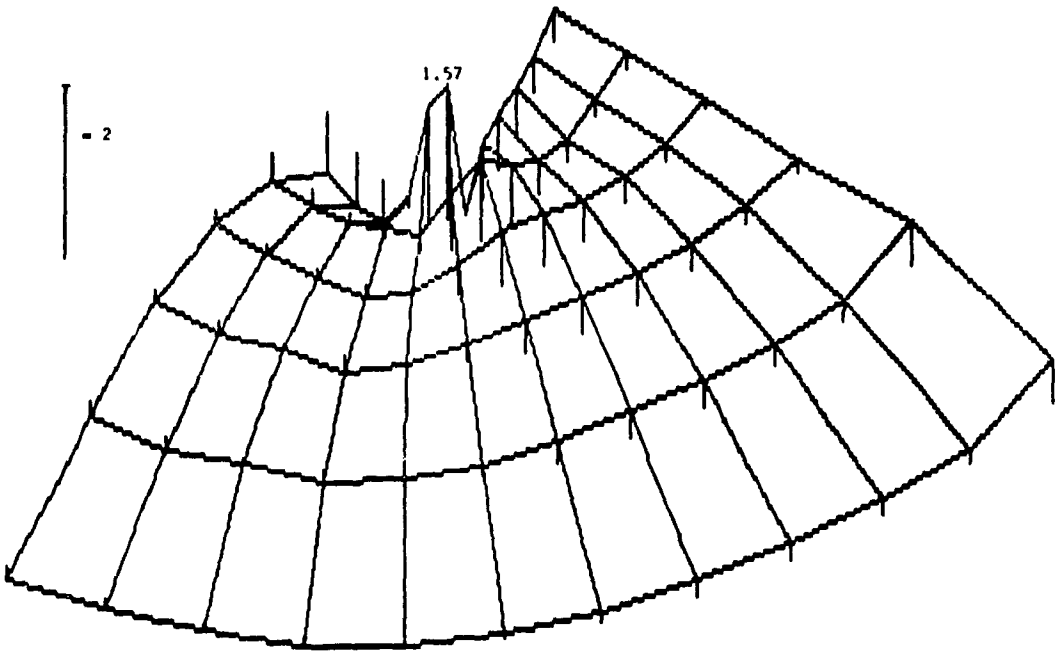


FIGURE 2.10b

σ_y^* Enlargement ($90^\circ \leq \theta \leq 145^\circ$)

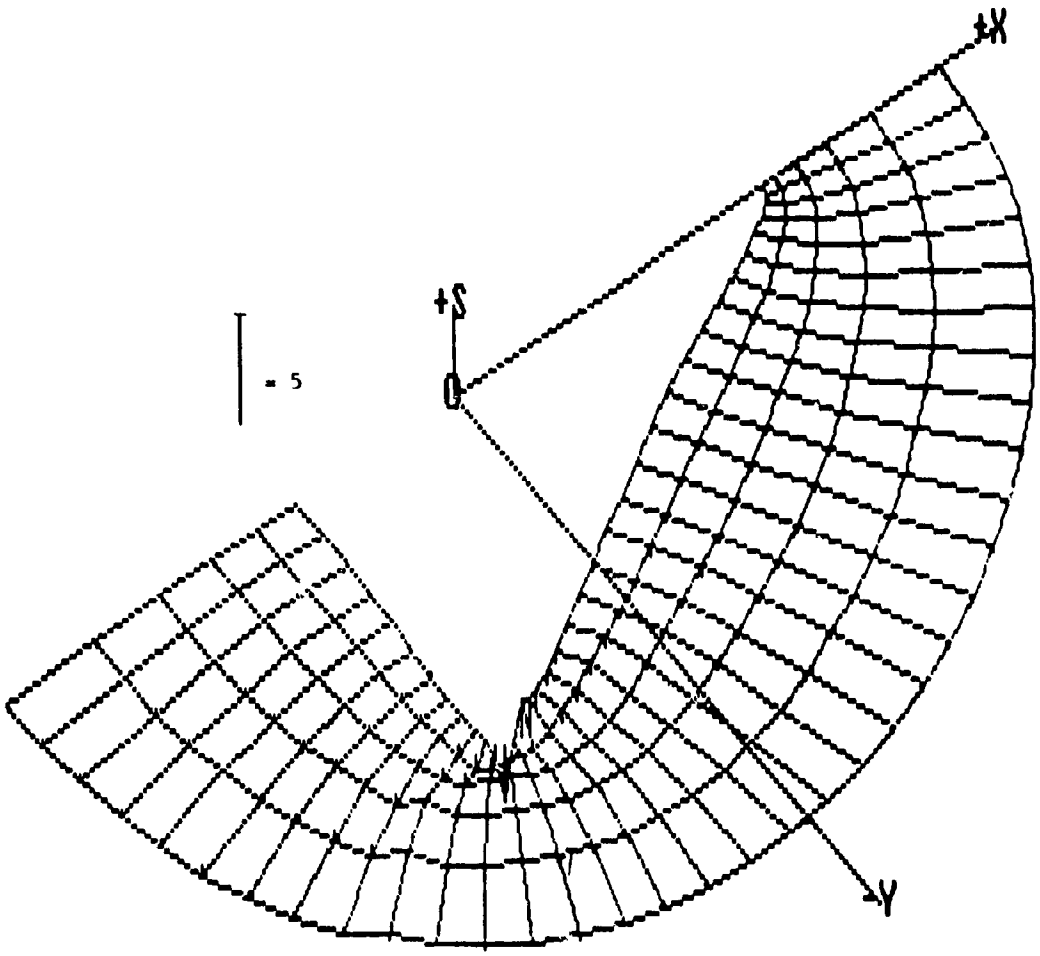


FIGURE 2.11a

σ^*_{xy} for $\varepsilon=1/3$, R=1, $\alpha=0$, Equilateral, CE 9000 material
($\Delta\vartheta=5^\circ$, $\Delta\rho=0.1$)

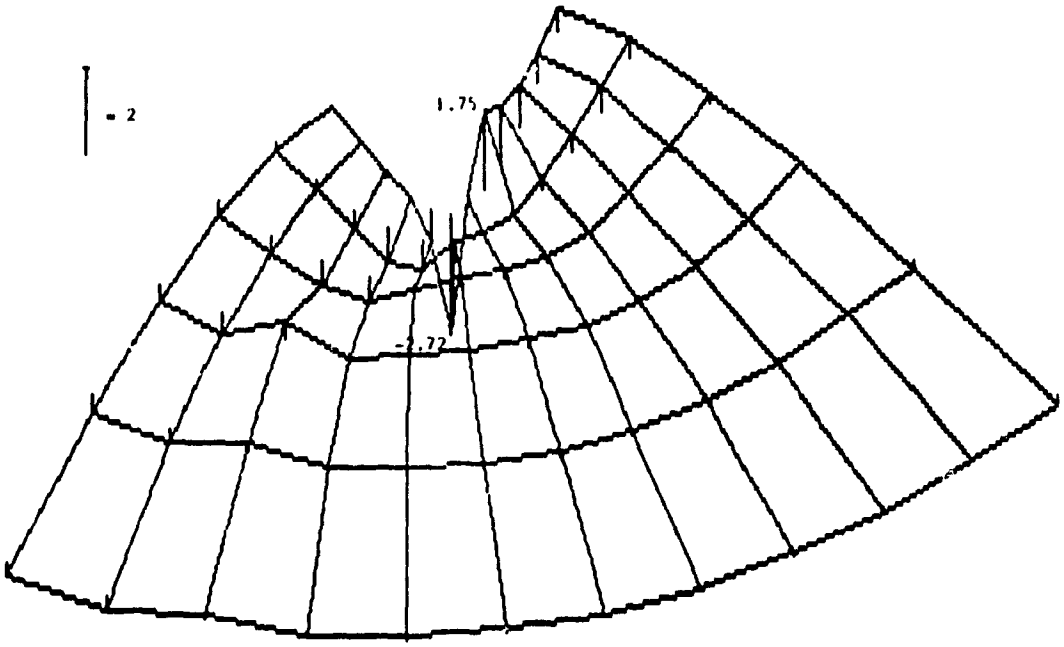


FIGURE 2.11b

σ^{*}_{xy} Enlargement ($90^{\circ} \leq \theta \leq 145^{\circ}$)

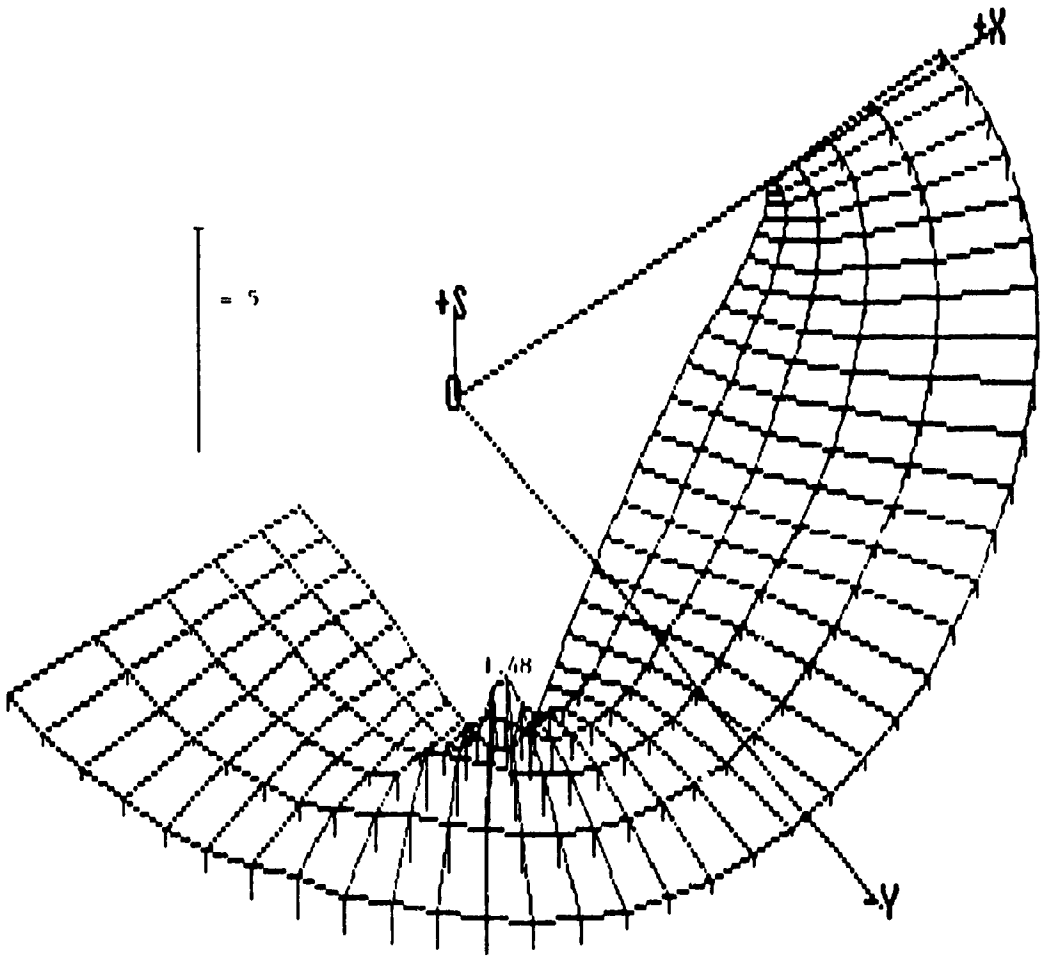


FIGURE 2.12a

σ_ρ^* for $\varepsilon=1/3$, $R=1$, $\alpha=0$, Equilateral, CE 9000 material
($\Delta\theta=5^\circ$, $\Delta\rho=0.1$)

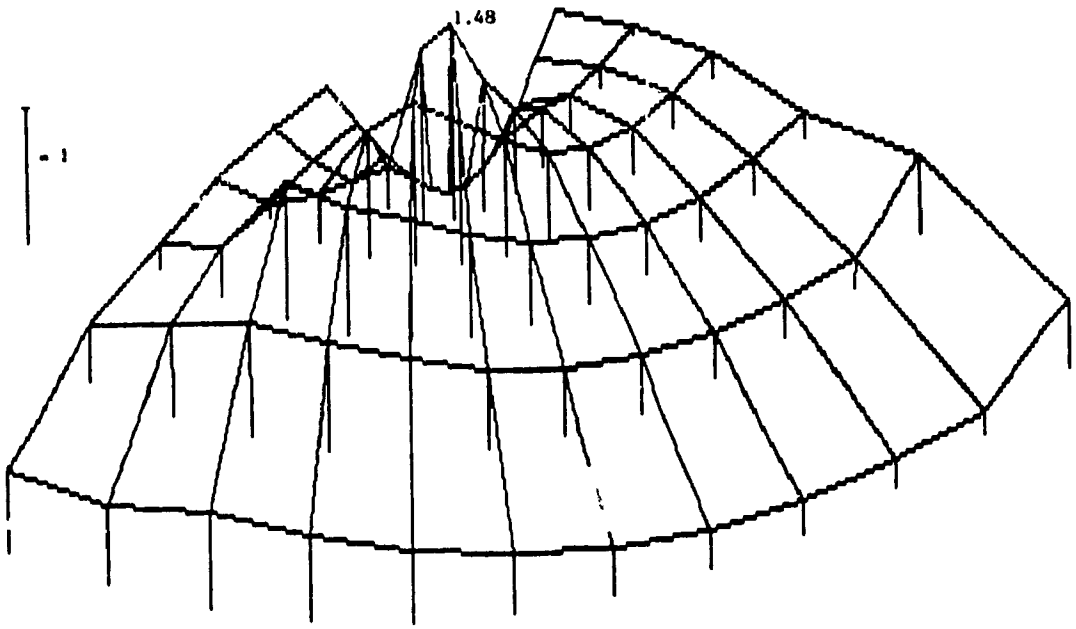


FIGURE 2.12b

σ_ρ^* Enlargement ($90^\circ \leq \phi \leq 145^\circ$)

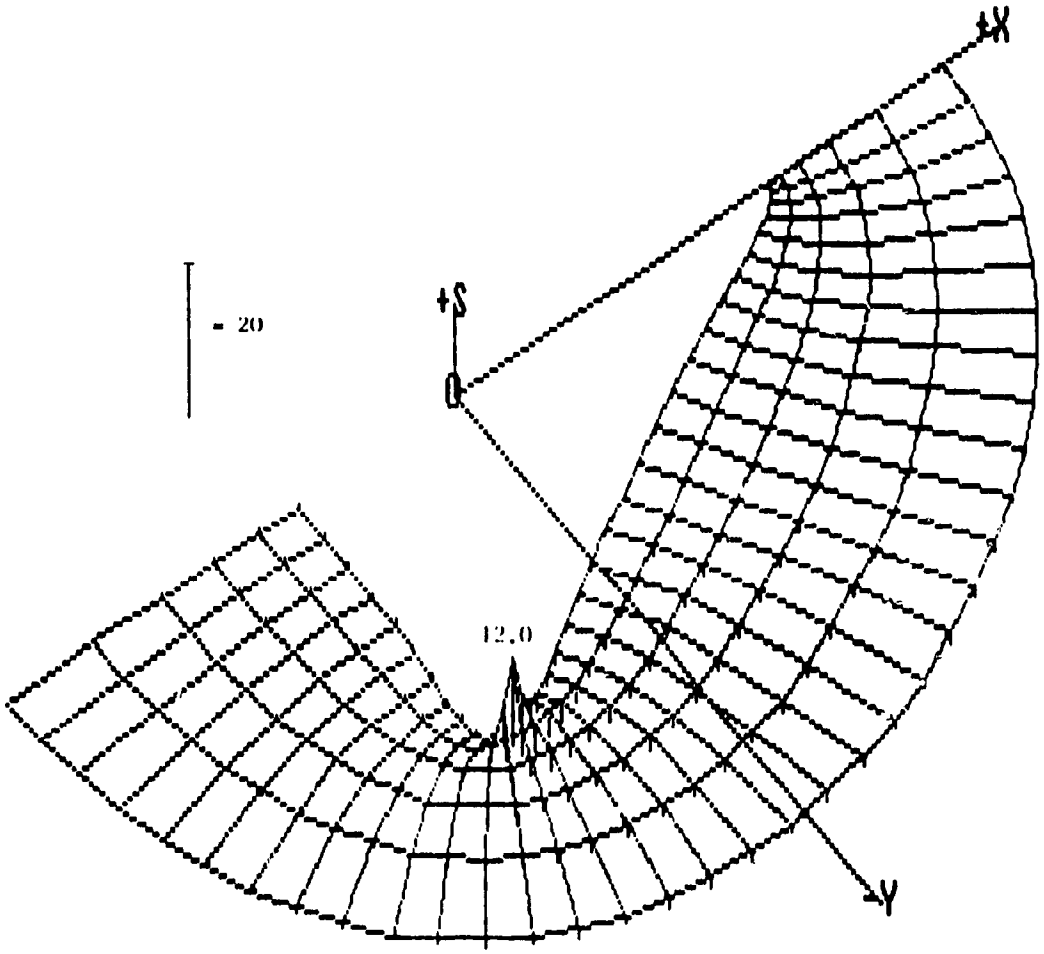


FIGURE 2.13a

σ^*_θ for ε=1/3, R=1, α=0, Equilateral, CE 9000 material
($\Delta\theta$=5°, $\Delta\rho$=0.1)

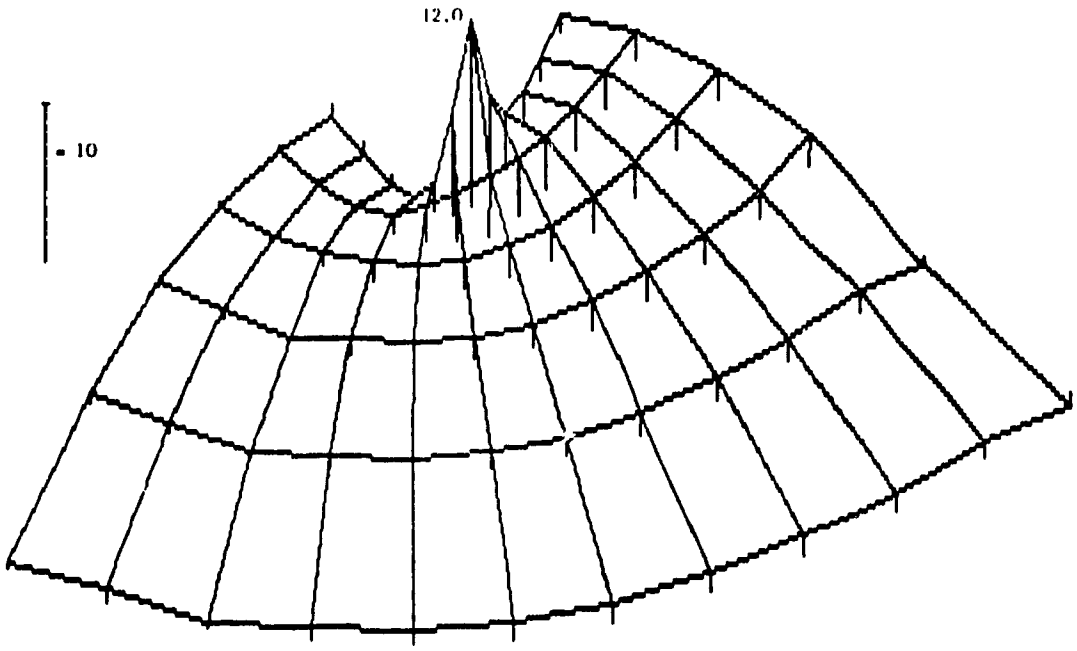


FIGURE 2.13b

σ^*_ϑ Enlargement ($90^\circ \leq \theta \leq 145^\circ$)

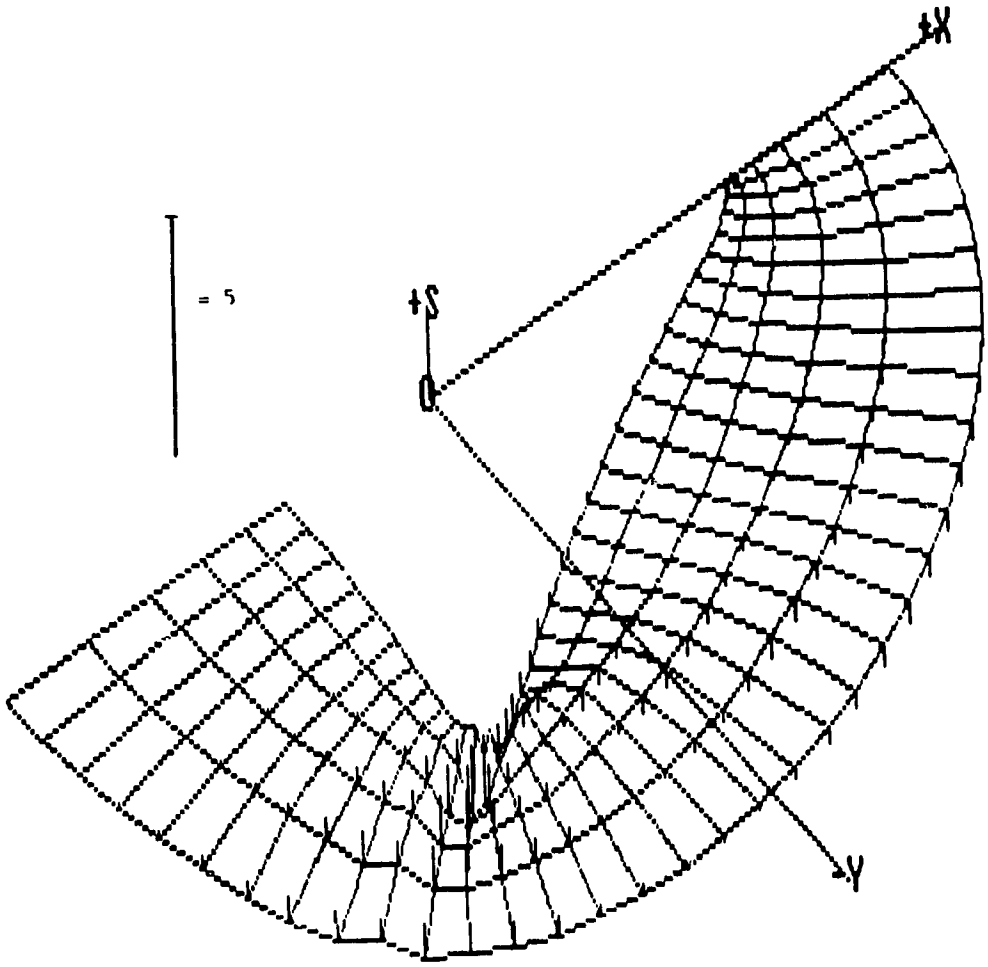


FIGURE 2.14a

$\sigma^*_{\rho\vartheta}$ for $\varepsilon=1/3$, $R=1$, $\alpha=0$, Equilateral, CE 9000 material
($\Delta\vartheta=5^\circ$, $\Delta\rho=0.1$)

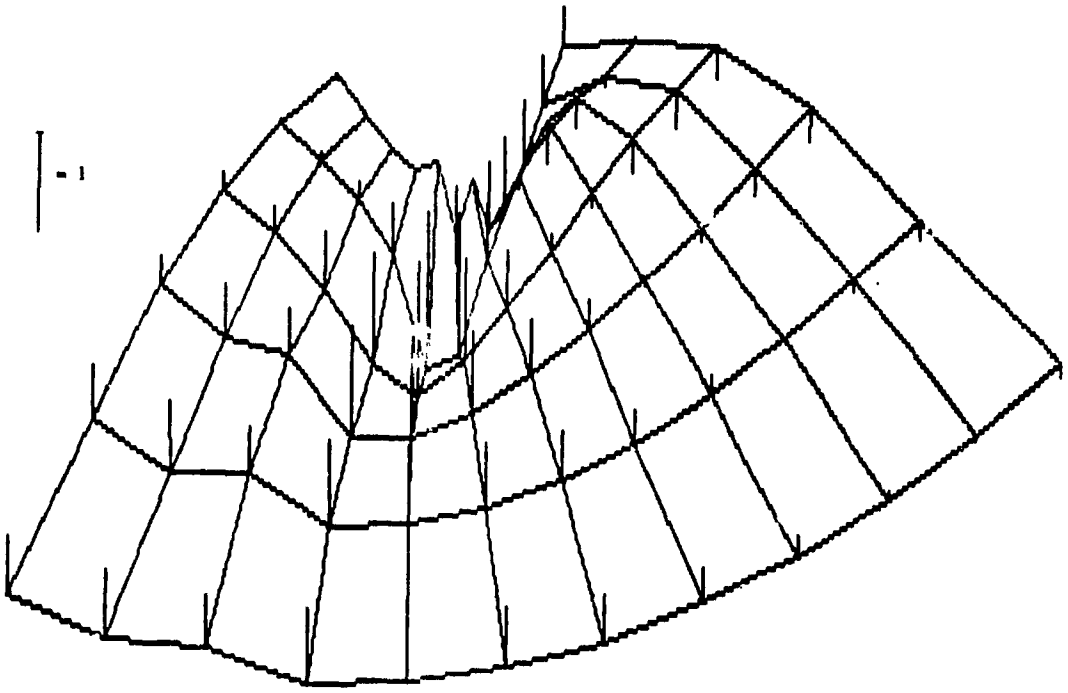


FIGURE 2.14b

$\sigma^{*}_{\rho\vartheta}$ Enlargement ($90^{\circ} \leq \vartheta \leq 145^{\circ}$)

The stress disturbance region is approximately confined to $100^{\circ}<\vartheta<135^{\circ}$ and $0.8<\rho\leq 1$. While peak normal stresses are higher than shear stresses, failure may occur in shear mode due to low interlaminar shear strength.

TABLE 2.6

Stress for CE 9000 glass/epoxy, ε=1/3, ρ=1

ϑ	$\sigma^{\cdot}_{x}$	$\sigma^{\cdot}_{y}$	$\sigma^{\cdot}_{xy}$	$\sigma^{\cdot}_{\rho}$	$\sigma^{\cdot}_{\vartheta}$	$\sigma^{\cdot}_{\rho\vartheta}$
0°	-0.000	-0.617	0.000	0.000	-0.617	0.000
10°	-0.093	-0.074	-0.033	0.000	-0.167	0.078
20°	0.048	0.015	0.027	0.000	0.063	-0.018
30°	0.143	0.035	0.071	0.000	0.178	-0.042
40°	0.206	0.052	0.103	0.000	0.258	-0.067
50°	0.260	0.074	0.139	0.000	0.334	-0.088
60°	0.320	0.107	0.184	0.000	0.426	-0.126
70°	0.399	0.155	0.249	0.000	0.554	-0.181
80°	0.527	0.229	0.347	0.000	0.756	-0.265
90°	0.784	0.348	0.522	0.000	1.132	-0.402
100°	1.502	0.538	0.898	0.000	2.039	-0.632
110°	5.302	0.577	1.750	0.000	5.879	-0.692
115.17°	12.131	0.000	0.005	0.000	12.131	0.002
120°	4.708	1.569	-2.718	0.000	6.278	-0.498
130°	0.002	0.051	-0.011	0.000	0.053	0.008
140°	-0.000	-0.643	-0.010	0.000	-0.643	0.010
150°	-0.003	-0.676	-0.044	0.000	-0.679	0.046
160°	-0.002	-0.647	-0.039	0.000	-0.649	0.041
170°	-0.001	-0.625	-0.021	0.000	-0.625	0.022
180°	-0.000	-0.617	-0.000	0.000	-0.617	0.000

Stresses can be evaluated at $\rho<1$ by using the selected values of n_1, s_1, n_2, s_2 at $\rho=1$ for the same ϑ. Decreasing ρ from

1, the stress at a neighboring point is evaluated by using η_1, a_1, η_2, a_2 of the previous point as an initial guess and using the same sign in Equation (2.134) which produced the appropriate ζ root. By using small increments of ρ, continuity is maintained for the selected roots for ζ. By this manner, the stress plots can be produced for the region surrounding the triangular hole.

Keeping $\varepsilon=\frac{1}{3}$ and CE 9000 glass/epoxy, a change in the $\frac{L}{h}$ ratio from 0.866 (equilateral triangle) to 10 drops the peak stress, σ_ϑ^* ($\vartheta=115.17^\circ$ $\rho=1$) from 12.13 to 1.96 (Figure 2.15).

Figure 2.16 shows the effect of ε upon σ_ϑ^* at $\rho=1$ with isotropic material and $\alpha=0$. Figure 2.17 shows the effect of materials (Table 2.3) upon σ_ϑ^* at $\rho=1$ with $\varepsilon=\frac{1}{3}$ and $\alpha=0$. No simple measure of anisotropy ($s_2 \div s_1$, $E_L \div E_T$, $Q_{yy} \div Q_{xx}$) was found to be able to predict the σ_ϑ^* peak levels which are: 21.63, 17.04, 12.13 and 8.28 for Graphite/Epoxy, Plywood, CE 9000 and isotropic streel respectively.

Figure 2.18 shows the effect of α (loading angle) upon σ_ϑ^* at $\rho=1$, $\vartheta=0^\circ$, 115.17° and 180° for a $\varepsilon=\frac{1}{3}$ triangular hole in an isotropic plate [Agreed with Tables 5 and 6 of Savin (1961) under Equation (2.59) which uses Equation (2.56) of the same author].

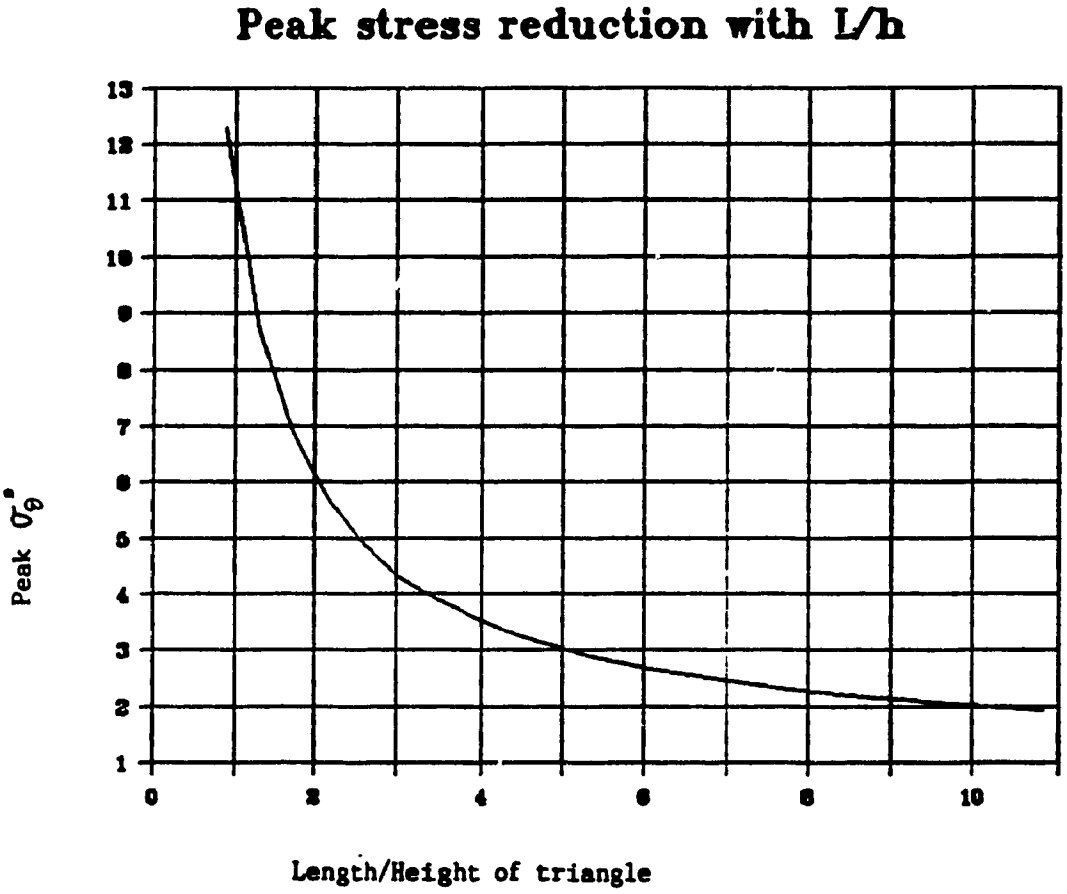


FIGURE 2.15

Peak σ_θ^* Reduction with L/h, ε=1/3, CE 9000

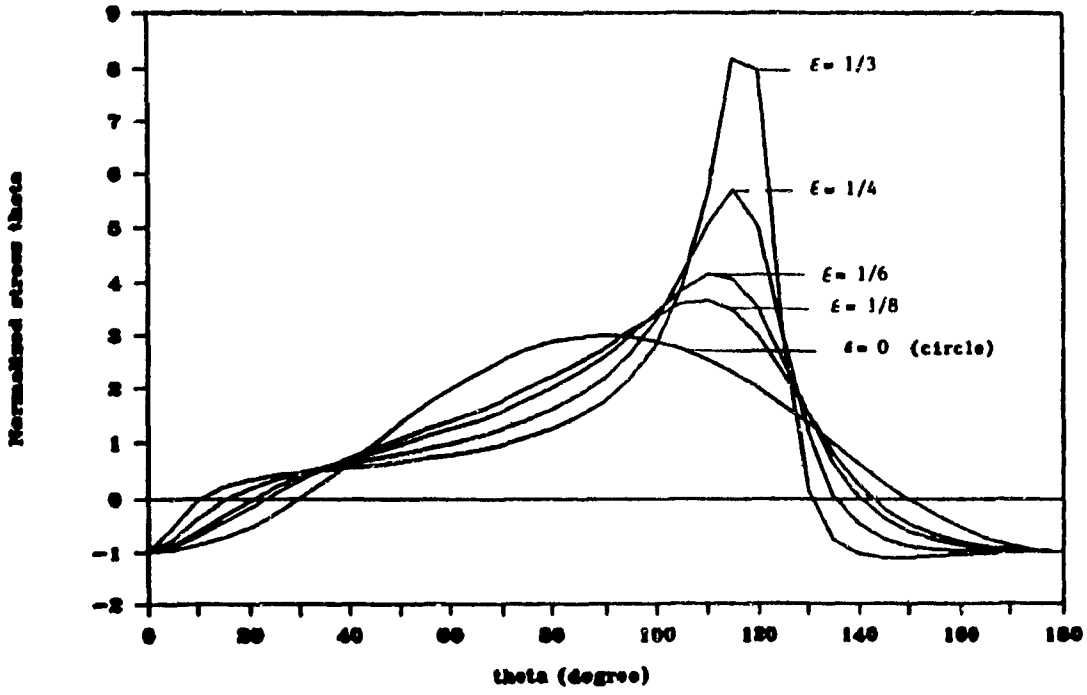


FIGURE 2.16

ε Effect upon σ_θ^* , Isotropic Material

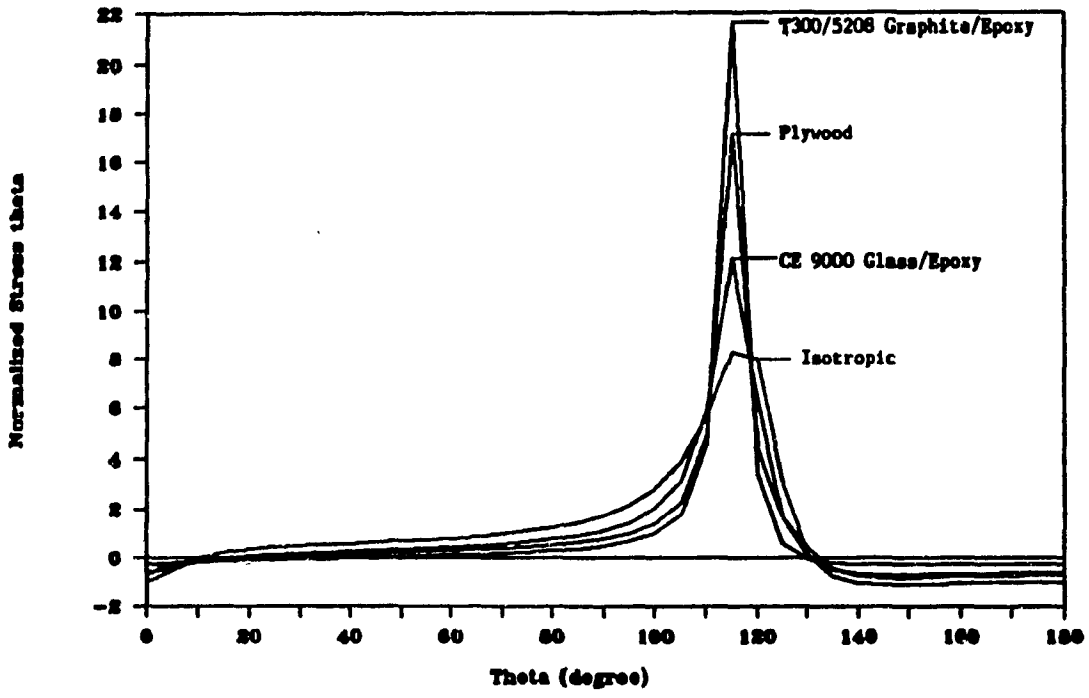


FIGURE 2.17

Material Effect upon σ^*_θ , ε=1/3

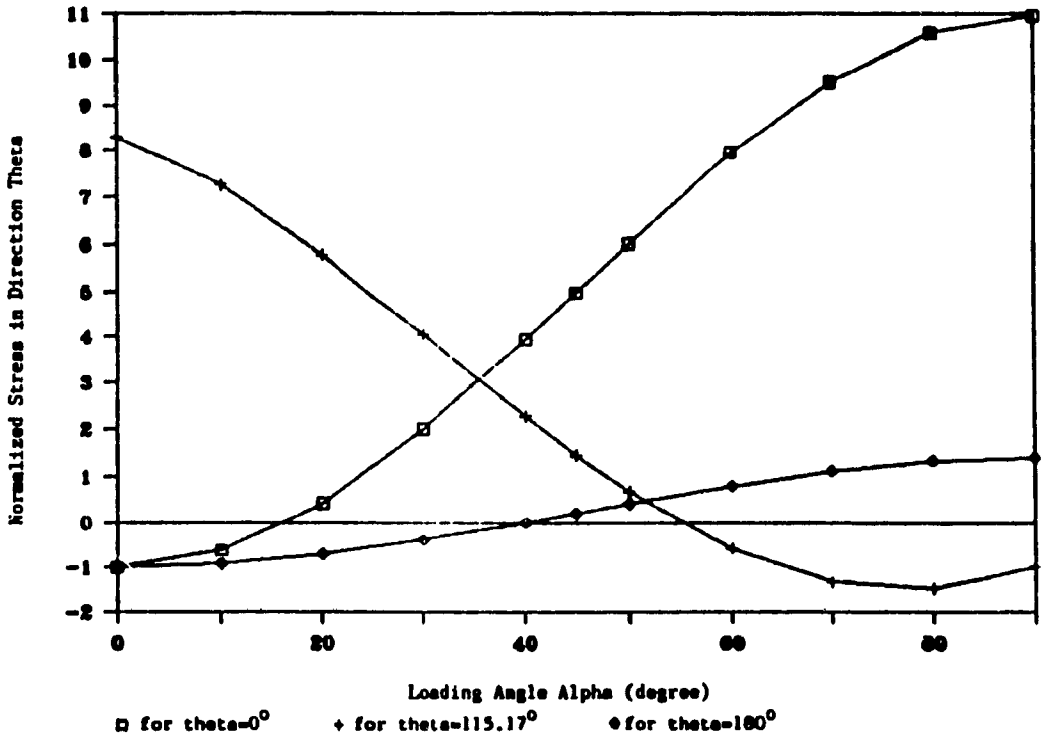


FIGURE 2.18

α Effect upon σ_θ^* at ρ=1 and θ=0°, θ=115.17°, θ=180°
(isotropic material, ε=1/3, r=1)

2.5 Solution Comparison

The closed form solution presented should match that of known simpler cases, like the solution given by Lekhnitskii (1968). Lekhnitskii's solution is recalled (case where $\alpha=0$) [Lekhnitskii (1968) Equations (54.5), (54.8), (54.6)]:

$$(\sigma_\vartheta)_A = \frac{P}{k}\left[-1 + \frac{\varepsilon^2}{1-2\varepsilon} gn - \frac{\varepsilon^3}{1-2\varepsilon} n\,(dh+gl+dgn) \right] \qquad (2.141)$$

point A is at $\vartheta=0^\circ$

$$(\sigma_\vartheta)_D = P\left[1+\frac{n}{B}\left[\sin\vartheta_0 - 2\varepsilon.\sin 2\vartheta_0 + \varepsilon^2 h.\sin\vartheta_0 + \varepsilon^3(dgk-hl)\sin 2\vartheta_0\right]\right] \qquad (2.142)$$

point D is where the boundary is parallel to the x axis ($\vartheta=\vartheta_0$)

$$(\sigma_\vartheta)_C = \frac{P}{k}\left[-1 + \frac{\varepsilon^2}{1+2\varepsilon} gn + \frac{\varepsilon^3}{1+2\varepsilon} n\,(dh+gl+dgn) \right] \qquad (2.143)$$

point C is at $\vartheta=180^\circ$

Other constants are given by [Lekhnitskii (1968) Equations (53.6), (52.9) to (52.11)]:

$$B = \sin\vartheta + 2\varepsilon.\sin 2\vartheta \qquad (2.144)$$

$$k = -\mu_1\mu_2 \quad [\mu_1=\beta_1 \quad \mu_2=\beta_2 \text{ in Equation (2.82)}] \qquad (2.145)$$

$$n = -i(\mu_1+\mu_2) \qquad (2.146)$$

$$g = \frac{8\,(1-k)}{(1+k+n)^2} \qquad (2.147)$$

$$h = \frac{2}{(1+k+n)^2}\left[(1-n)^2+k(k+2n-6)\right] \qquad (2.148)$$

$$d = \frac{4}{1+k+n} \qquad (2.149)$$

$$l = \frac{2}{1+k+n}\,(1-k-n) \qquad (2.150)$$

Table 2.7 compares both closed form solutions using properties given in Table 2.3 under plane strain conditions. For isotropic material, plane strain gives the same result as plane stress because s_1 and s_2 remain the same.

TABLE 2.7

Closed Form Solution Comparison

				σ'_ϑ					
				$\vartheta=0^\circ$		$\vartheta=\vartheta_0$		$\vartheta=180^\circ$	
Material	α	ε	ϑ_0	L[1]	D[2]	L	D	L	D
Iso.[3]	0°	0	90°	-1.00	-1.00	3.00	3.00	-1.00	-1.00
Iso.	0°	1/4	111.47°	-1.00	-1.00	5.31	5.33	-1.00	-1.00
Iso.	0°	1/3	115.17°	-1.00	-1.00	8.24	8.28	-1.00	-1.00
Iso.	90°	1/4	111.47°	7.00	6.97	-1.00	-1.00	1.67	1.66
CE 9000	0°	1/4	111.47°	-0.66	-0.62	7.74	7.63	-0.63	-0.62
CE 9000	0°	1/3	115.17°	-0.75	-0.62	12.39	12.13	-0.63	-0.62

1: L stands for the solution of Lekhnitskii (1968).

2: D stands for the solution of Daoust.

3: For the current solution I had to use $s_1 \neq s_2$ for isotropic so that B^* B'^* and C'^* have nonzero denominators in Equations (2.89) to (2.91).
$\varepsilon \neq 0$ is required for d_1 in Equation (2.121).
In both cases, a variation of 0.001 corrects the problem

By examination of Table 2.7, the current solution is confirmed by Lekhnitskii's (1968) solution.

2.6 Summary

The shape of a real drop-off can not be precisely identified, but a triangular hole provides a possible drop-off geometry which can give insight on interlaminar stresses distribution.

Analytical functions have been developed for two dimensional stress distributions around a triangular hole of a given Length/Height ratio in an isotropic or anisotropic plate under a tensile load in any direction (Figure 2.5). The solution can be used for plane stress and plane strain problems. The curvature of the summits of the triangle can be controlled by including more terms in the shape function. The overall bluntness of the hole can also be controlled. Table 2.8 gives the advantages of this solution with respect to previous ones.

TABLE 2.8

Closed Form Solution Features

	Savin (1961)	Lekhnitskii (1968)	Daoust (1988)
Triangle shape:	Equilateral	Equilateral	Any
Can the triangle be blunt?[1]	No	Yes	Yes
Can the triangle be sharp?[2]	Yes	No	Yes
Type of material:[3]	Isotropic	Orthotropic	Anisotropic
Loading angle (α):	0° to 360°	0° and 90°	0° to 360°
Stress solution region:	Anywhere	Boundary	Anywhere

1: A blunt triangle has a high radius of curvature at its summit, example: figure 2.7 with ε=1/6 and 4 terms.

2: A sharp triangle has a low radius of curvature at its summit, example: figure 2.7 with ε=1/3 and 10 terms.

3: Orthotropic identifies that the fibers in the material are along the x or y axis of figure 2.5. Anisotropic material has any fiber orientation in the x-y plane of figure 2.5.

The solution presented in this chapter is not applicable to a bending loading condition, but could be modified for it. In real drop-offs, the fibers follow the triangular shape of the hole. This theory is limited to a single material property direction. The theory can be applied to triangular holes in anisotropic plates which have been cut out after the plate fabrication.

The closed form solution is used to obtain insight upon factors influencing the stress distribution. Keeping $\varepsilon=\frac{1}{3}$ and Gl. 9000 glass/epoxy, a change in the $\frac{L}{h}$ ratio from 0.866 (equilateral

triangle) to 10, drops the peak stress $\sigma_{\vartheta}^{\cdot}$ (ϑ=115.17° ρ=1) from 12.13 to 1.96 (Figure 2.15). Figure 2.16 shows the effect of ε upon $\sigma_{\vartheta}^{\cdot}$ at ρ=1 with isotropic material and α=0. Figure 2.17 shows the effect of materials (Table 2.3) upon $\sigma_{\vartheta}^{\cdot}$ at ρ=1 with $\varepsilon=\frac{1}{3}$ and α=0 Figure 2.18 shows the effect of α (loading angle) upon $\sigma_{\vartheta}^{\cdot}$ at ρ=1, ϑ=0°, 115.17° and 180° for a $\varepsilon=\frac{1}{3}$ triangular hole in an isotropic plate.

CHAPTER 3

PROBLEM FORMULATION

3.1 Introduction

The closed form solution presented in the previous chapter is limited in applications to two dimensional problems in which only two dimensional stress (σ_x, σ_y, σ_{xy}) are obtained. This assumes that the plate is very thin or very thick and that loadings are in plane (Figure 2.5). In real structures, such as helicopter yokes, the dimension in the z direction is finite and sometimes can be significantly large. Stress components σ_z, σ_{xz}, σ_{yz} definitely exist, particularly at the tip of the discontinuity. For these real structures, the closed form solution does not work and numerical techniques such as three dimensional finite elements have to be used. However, the closed form solution is useful in serving as a check for the finite element model. It is assumed here that if agreement is obtained between the finite element model and the closed form solution in the two dimensional case, the model is reasonably correct to be extended to three dimensional problems.

There are many ways to formulate the finite elements: displacements, hybrid stress - displacement formulations. Displacement formulation is the conventional approach where the

displacement field is assumed. After the potential energy is minimized, displacements are obtained approximately. Strains are then calculated by differentiating the displacements. Differentiation magnify the errors. For problems where there are high strain gradients, extremely fine mesh has to be used before an accurate solution can be obtained.

In hybrid stress formulation [Pian (1964)], stresses are assumed. After the minimization process, stresses are obtained without differentiation. The error obtained for stresses are therefore reduced. This approach shows promise for the calculation of local stresses in composites. However, extensive work has to be done before it can be applied successfully to structures made of composite [Discussion with Q.Huang, Ph.D. candidate at Concordia].

Since the hybrid stress formulation is not yet fully proven in composites, the displacement formulated finite element method is used in this thesis. In order to ensure accuracy of the results, results obtained from different meshes are compared with experimental results. Only the mesh that gives the best agreement with experimental values is used in subsequent analysis.

This chapter goes through the details of the element formulation. For the sake of completeness, the whole element formulation is presented in this chapter.

3.2 Virtual Work Theory [Zienkewicz (1977)]

Consider an infinitesimal volume dV of material that is subjected to a stress σ. If the displacement is varied from the deformed configuration by a virtual strain δe (Figure 3.1), the virtual strain energy δU is given as $\delta U = \sigma . \delta e$. For the total volume, the virtual work is

$$\delta U = \int_{\Omega} \{\sigma\}^T . \delta\{e\} . dV \tag{3.1}$$

where Ω is the volume and

$$\{\sigma\} = [C]\{e\} \tag{3.2}$$

$$\{\sigma\} = [\sigma_x \quad \sigma_y \quad \sigma_z \quad \sigma_{xy} \quad \sigma_{xz} \sigma_{yz}]^T$$

$$\{e\} = [e_x \quad e_y \quad e_z \quad e_{xy} \quad e_{xz} e_{yz}]^T$$

(tensorial strain)

[C] is the stiffness matrix

$$\{e\} = [BG]\{\Delta\}$$

$$\{\Delta\} = \text{nodal displacements} \tag{3.3}$$

$$= [u_1 \cdots u_N, \; v_1 \cdots v_N, \; w_1 \cdots w_N]^T$$

[BG] is a differentiating matrix

substituting Equations (3.3) and (3.2) into Equation (3.1) yields

$$\delta U = \int_{\Omega} \{\Delta\}^T . [BG]^T . [C] . [BG] . \delta\{\Delta\} . dV \tag{3.4}$$

$\{\Delta\}^T$ and $\delta\{\Delta\}$ remain constant throughout the integration, giving:

$$\delta U = \{\Delta\}^T \underbrace{\left[\int_\Omega [BG]^T.[C].[BG].dV\right]}_{[K]} \delta\{\Delta\} \qquad (3.5)$$

$$= \{\Delta\}^T.[K].\delta\{\Delta\} \qquad (3.6)$$

= virtual strain energy inside element

If the body is subjected to external load {F}, the virtual work done by these forces is

$$\delta V = -\{F\}.\delta\{\Delta\} \qquad (3.7)$$

= virtual work done by forces on nodes (external or inertia forces)

The variation of the work done by the forces on the node should equal that of the strain energy inside the element:

$$\delta U + \delta V = 0 \qquad (3.8)$$

Substituting Equations (3.6) and (3.7) into (3.8) yields

$$\{\Delta\}^T.[K].\delta\{\Delta\} = \{F\}.\delta\{\Delta\} \quad \rightarrow \quad [K].\{\Delta\} = \{F\} \qquad (3.9)$$

Equation (3.9) shows that the nodal displacement is related to nodal forces through a stiffness matrix [K].

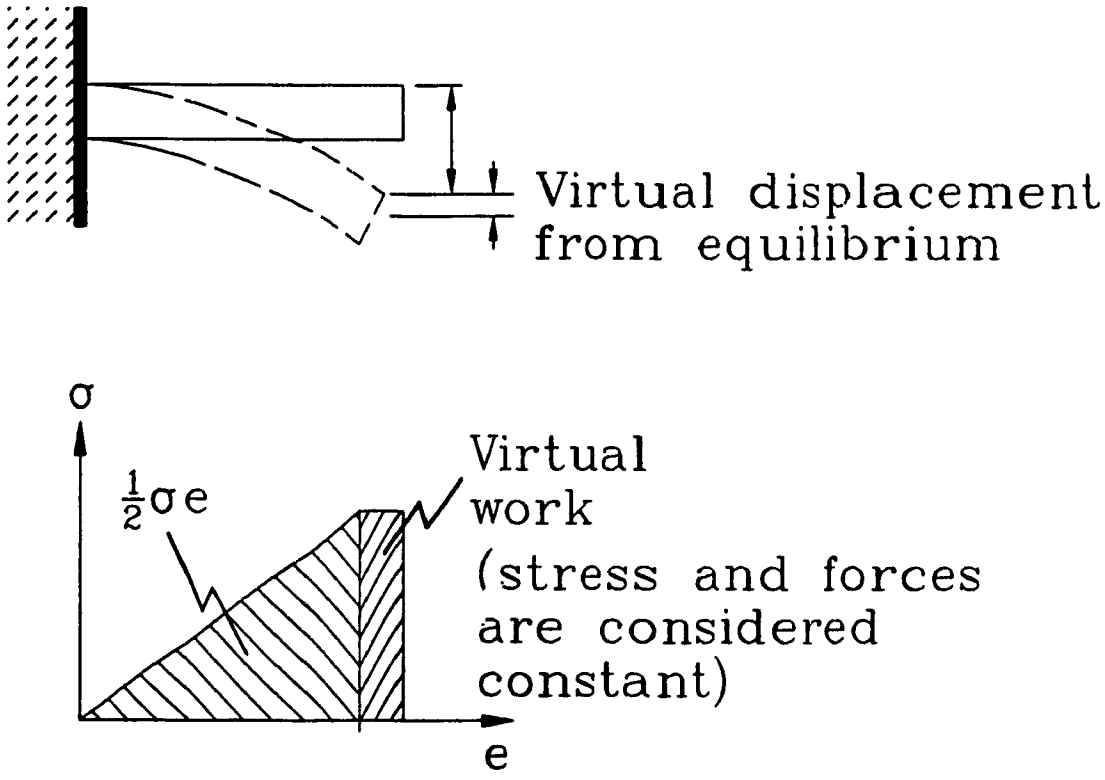


FIGURE 3.1

Virtual Work

3.3 Stiffness Matrix [Hoa, Yu and Sankar (1985)]

In order to obtain linear strain (or stress) variation within the element, a 20 node hexahedron finite element is used (Figure 3.2). This element is isoparametric: same shape function applies for displacements or coordinates of the element. The displacements can be expressed as:

$$u = [N]\{u\}$$
$$v = [N]\{v\}$$
$$w = [N]\{w\} \qquad (3.10)$$

where [N] is the shape function matrix expressed in terms of 20 shape functions:

$$[N] = [N_1, N_2 \ldots N_{20}] \qquad (3.11)$$

and the nodal displacement vectors {u},{v},{w} are expressed in terms of 20 nodal displacements:

$$\{u\} = [u_1, u_2, \ldots, u_{20}]^T \qquad (3.12)$$

and similarly for {v} and {w}.

The N_i at the vertex node in Equation (3.11), with the origin of the coordinates at the centroid as in Figure 3.2, are:

$$N_i = \frac{1}{8}(1+\xi\xi i)(1+\eta\eta_i)(1+\zeta\zeta_i)(\xi\xi_i+\eta\eta_i+\zeta\zeta_i-2) \qquad (3.13)$$

and for typical mid-edge nodes:

$\zeta_i=0,\ \eta_i=\pm1,\ \xi_i=\pm1$

$$N_i = \frac{1}{4}(1-\xi^2)(1+\eta\eta_i)(1+\zeta\zeta_i) \qquad (3.14)$$

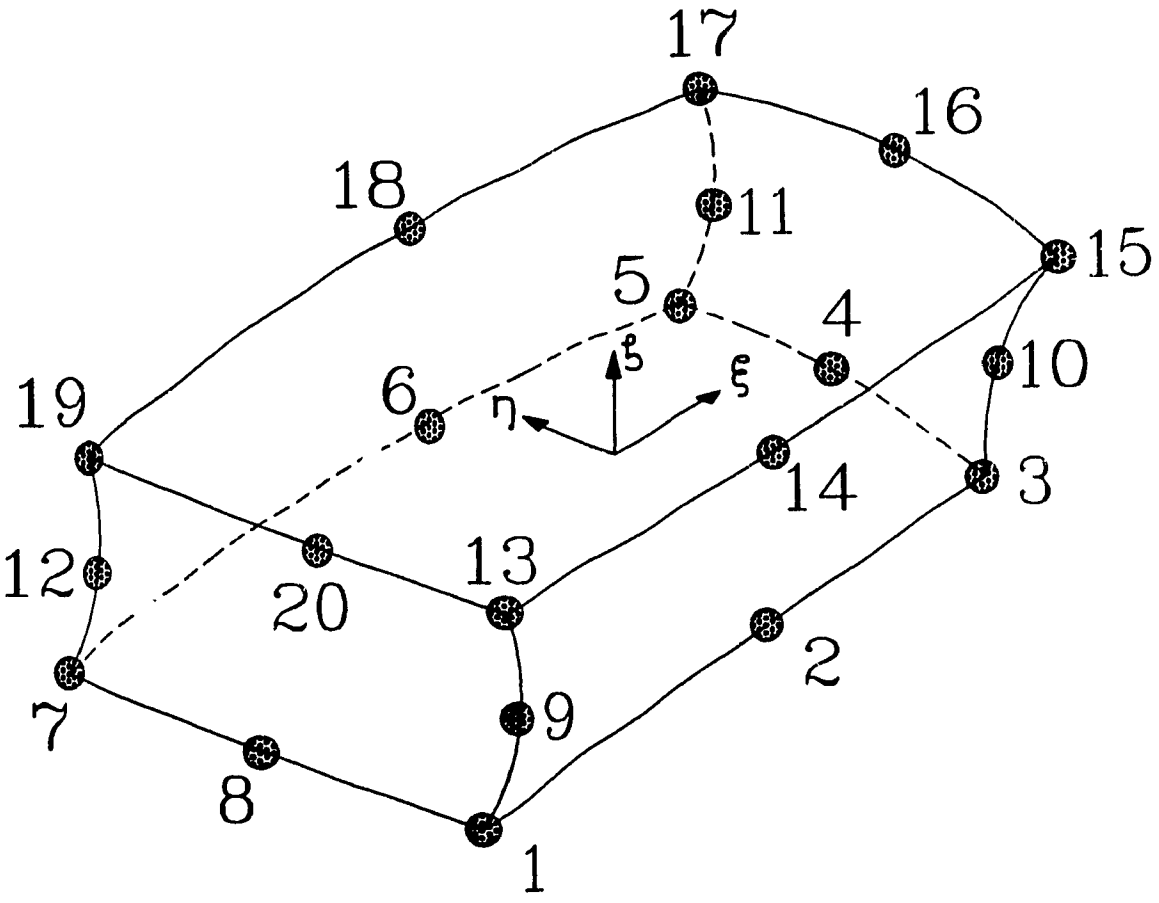


FIGURE 3.2

20 Node Element

The linear strain(tensorial)-displacement equations are

$$e_x = \partial u/\partial x \qquad e_{xy} = \frac{1}{2}\,(\partial u/\partial y + \partial v/\partial x)$$

$$e_y = \partial v/\partial y \qquad e_{xz} = \frac{1}{2}\,(\partial w/\partial x + \partial u/\partial z)$$

$$e_z = \partial w/\partial z \qquad e_{yz} = \frac{1}{2}\,(\partial w/\partial y + \partial v/\partial z) \tag{3.15}$$

$$\{e\} = [e_x\ e_y\ e_z\ e_{xy}\ e_{xz}\ e_{yz}]^T \tag{3.16}$$

Equations (3.10) - (3.14) show that u,v,w are functions of the local coordinates ξ,η,ζ; the strains in Equation (3.15) can be evaluated by utilizing the relation:

$$\begin{bmatrix} \partial N_i/\partial \xi \\ \partial N_i/\partial \eta \\ \partial N_i/\partial \zeta \end{bmatrix} = [J]. \begin{bmatrix} \partial N_i/\partial x \\ \partial N_i/\partial y \\ \partial N_i/\partial z \end{bmatrix} \tag{3.17}$$

where

$$[J]_{3x3} = \begin{bmatrix} \Sigma\left(\frac{\partial N_i}{\partial \xi} x_i\right) & \Sigma\left(\frac{\partial N_i}{\partial \xi} y_i\right) & \Sigma\left(\frac{\partial N_i}{\partial \xi} z_i\right) \\ \Sigma\left(\frac{\partial N_i}{\partial \eta} x_i\right) & \Sigma\left(\frac{\partial N_i}{\partial \eta} y_i\right) & \Sigma\left(\frac{\partial N_i}{\partial \eta} z_i\right) \\ \Sigma\left(\frac{\partial N_i}{\partial \zeta} x_i\right) & \Sigma\left(\frac{\partial N_i}{\partial \zeta} y_i\right) & \Sigma\left(\frac{\partial N_i}{\partial \zeta} z_i\right) \end{bmatrix} \quad \text{where } \Sigma = \sum_{i=1}^{20}$$

$$= \begin{bmatrix} \partial x/\partial \xi & \partial y/\partial \xi & \partial z/\partial \xi \\ \partial x/\partial \eta & \partial y/\partial \eta & \partial z/\partial \eta \\ \partial x/\partial \zeta & \partial y/\partial \zeta & \partial z/\partial \zeta \end{bmatrix} \tag{3.18}$$

The strain-displacement Equation (3.15) can be put in the form

$$\{e\} = [BG].\{\delta\} \tag{3.19}$$

where

$$\{\delta\} = [u_1 \ \cdots \ u_{20},\ v_1 \ \cdots \ v_{20},\ w_1 \ \cdots \ w_{20}]^T \qquad (3.20)$$

The location of a point P(x,y,z) (Figure 3.3) is determined by vector $\vec{R}$:

$$\vec{R} = \vec{I}.x + \vec{J}.y + \vec{K}.z$$

The unit vector along the ξ direction (Figure 3.2) at P can be expressed as

$$\vec{V}_1 = \frac{\vec{I}(\partial x/\partial\xi) + \vec{J}(\partial y/\partial\xi) + \vec{k}(\partial z/\partial\xi)}{\sqrt{(\partial x/\partial\xi)^2+(\partial y/\partial\xi)^2+(\partial z/\partial\xi)^2}} \qquad (3.21)$$

Similarly, the unit vector along the η direction is

$$\vec{V}_2' = \frac{\vec{I}(\partial x/\partial\eta) + \vec{J}(\partial y/\partial\eta) + \vec{k}(\partial z/\partial\eta)}{\sqrt{(\partial x/\partial\eta)^2+(\partial y/\partial\eta)^2+(\partial z/\partial\eta)^2}} \qquad (3.22)$$

Since the ξ and η directions may not be orthogonal at point P in the x,y,z space, vectors $\vec{V}_1$ and $\vec{V}_2'$ may not be normal to each other. So a third vector, $\vec{V}_3$, is set normal to $\vec{V}_1$ and $\vec{V}_2'$. Then $\vec{V}_2$ is set as the normal to $\vec{V}_1$ and $\vec{V}_3$.

$$\vec{V}_3 = (\vec{V}_1 \times \vec{V}_2')/\sin(x) \qquad (3.23)$$

where $\cos(x)=\vec{V}_1.\vec{V}_2$

Then: $\vec{V}_2 = \vec{V}_3 \times \vec{V}_1$ (3.24)

The transformation from displacements along the global x,y,z directions $[u,v,w]^T$ to displacements along the local directions (ξ η ζ in Figure 3.2) $[u',v',w']^T$ can be written as

$$[u,v,w]^T = [L].[u',v',w'] \qquad (3.25)$$

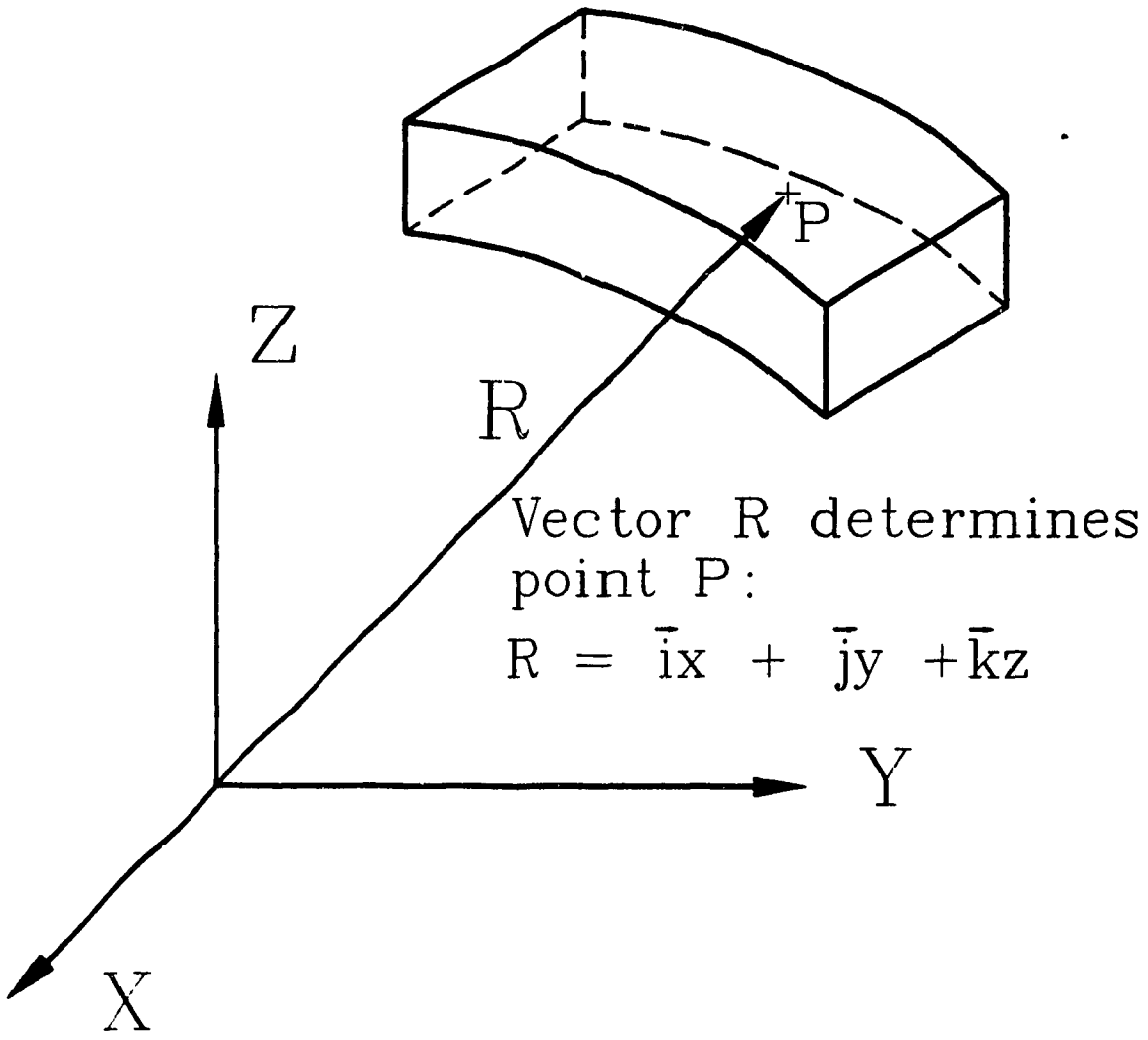


FIGURE 3.3

Location of a Point in x,y,z Coordinates

where

$$[L] = \begin{bmatrix} V_{1i} & V_{2i} & V_{3i} \\ V_{1j} & V_{2j} & V_{3j} \\ V_{1k} & V_{2k} & V_{3k} \end{bmatrix} \tag{3.26}$$

It can be shown that

$$\begin{bmatrix} \frac{\partial u'}{\partial x'} & \frac{\partial v'}{\partial x'} & \frac{\partial w'}{\partial x'} \\ \frac{\partial u'}{\partial y'} & \frac{\partial v'}{\partial y'} & \frac{\partial w'}{\partial y'} \\ \frac{\partial u'}{\partial z'} & \frac{\partial v'}{\partial z'} & \frac{\partial w'}{\partial z'} \end{bmatrix} = [L]^T \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} & \frac{\partial w}{\partial x} \\ \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} & \frac{\partial w}{\partial y} \\ \frac{\partial u}{\partial z} & \frac{\partial v}{\partial z} & \frac{\partial w}{\partial z} \end{bmatrix} [L] \tag{3.27}$$

Similar to Equation (3.19) we can express:

$$\{e'\} = [BF]\{\delta\} \tag{3.28}$$

where

$$\{e'\} = [e_{x'}, e_{y'}, e_{z'}, e_{x'y'}, e_{x'z'}, e_{y'z'}]^T \tag{3.29}$$

where

$$e_{x'} = \partial u'/\partial x' \qquad e_{x'y'} = \frac{1}{2}(\partial u'/\partial y' + \partial v'/\partial x')$$

$$e_{y'} = \partial v'/\partial y' \qquad e_{x'z'} = \frac{1}{2}(\partial w'/\partial x' + \partial u'/\partial z')$$

$$e_{z'} = \partial w'/\partial z' \qquad e_{y'z'} = \frac{1}{2}(\partial w'/\partial y' + \partial v'/\partial z') \tag{3.30}$$

Matrix [BF] is obtained by combining Equations (3.26), (3.27) and (3.30).

$$[BF] = \begin{bmatrix} P_1V_{1i} & P_1V_{1j} & P_1V_{1k} \\ P_2V_{2i} & P_2V_{2j} & P_2V_{2k} \\ P_3V_{3i} & P_3V_{3j} & P_3V_{3k} \\ \frac{P_2V_{1i}+P_1V_{2i}}{2} & \frac{P_2V_{1j}+P_1V_{2j}}{2} & \frac{P_2V_{1k}+P_1V_{2k}}{2} \\ \frac{P_3V_{1i}+P_1V_{3i}}{2} & \frac{P_3V_{1j}+P_1V_{3j}}{2} & \frac{P_3V_{1k}+P_1V_{3k}}{2} \\ \frac{P_3V_{2i}+P_2V_{3i}}{2} & \frac{P_3V_{2j}+P_2V_{3j}}{2} & \frac{P_3V_{2k}+P_2V_{3k}}{2} \end{bmatrix} \qquad (3.31)$$

where

$$P_m = V_{mi}.\partial[N]/\partial x + V_{mj}.\partial[N]/\partial y + V_{mk}.\partial[N]/\partial z \qquad (3.32)$$

$m=1,2,3$

This is a new concise formulation of the displacement to tensorial strain transformation. The on-axis stiffness matrix is defined by

$$\{\bar{\sigma}\} = [C].\{\bar{e}\} \qquad (3.33)$$

where

$$\{\bar{\sigma}\} = [\sigma_1, \sigma_2, \sigma_3, \sigma_{12}, \sigma_{13}, \sigma_{23}]$$

$$\{\bar{e}\} = [e_1, e_2, e_3, e_{12}, e_{13}, e_{23}] \quad \text{(tensorial strain)}$$

$$[C] = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{21} & C_{22} & C_{23} & 0 & 0 & 0 \\ C_{31} & C_{32} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & & C_{66} \end{bmatrix} \qquad (3.34)$$

where

$$C_{11} = (1-\nu_{23}\nu_{32})/(E_2E_3\Delta)$$

$$C_{12} = (\nu_{12}+\nu_{32}\nu_{13})/(E_1E_3\Delta)$$

$$C_{13} = (\nu_{13}+\nu_{12}\nu_{23})/(E_1E_2\Delta)$$

$$C_{22} = (1-\nu_{13}\nu_{31})/(E_1E_3\Delta)$$

$$C_{23} = (\nu_{23}+\nu_{21}\nu_{13})/(E_1E_2\Delta)$$

$$C_{33} = (1-\nu_{12}\nu_{21})/(E_1E_2\Delta)$$

$$C_{44} = 2G_{12}$$

$$C_{55} = 2G_{13}$$

$$C_{66} = 2G_{23} \qquad (3.35)$$

$$\Delta = (1-\nu_{12}\nu_{21}-\nu_{23}\nu_{32}-\nu_{31}\nu_{13}-2\nu_{21}\nu_{32}\nu_{13})/(E_1E_2E_3)$$

For the case of fiber reinforced materials there are six independent elastic constants:

$E_L = E_1$ = modulus along the fiber direction

$E_T = E_2 = E_3$ = modulus transverse to fiber direction

$G_{LT} = G_{12} = G_{13}$ = shear modulus

$G_{TT} = G_{23}$ = shear modulus

$\nu_{LT} = \nu_{12} = \nu_{13}$ = major Poison's ratio

$\nu_{TT} = \nu_{23}$ = minor Poison's ratio (3.36)

with the relation

$$\nu_{ij}/E_i = \nu_{ji}/E_j \qquad i,j=1,2,3 \qquad (3.37)$$

Equations (3.36) and (3.37) lead to

$$C_{33} = C_{22}$$

$$C_{13} = C_{12}$$

$$C_{55} = C_{44}$$

Now, the stress-strain Equation (3.33) has to be transformed to local coordinates

$$\{\sigma'\} = [TR]\{\bar{\sigma}\}$$

$$\{e'\} = [TR]\{\bar{e}\} \tag{3.38}$$

where $\{\sigma'\} = [\sigma_{x'}, \sigma_{y'}, \sigma_{z'}, \sigma_{x'y'}, \sigma_{x'z'}, \sigma_{y'z'},]^T$

$\{e'\} = [e_{x'}, e_{y'}, e_{z'}, e_{x'y'}, e_{x'z'}, e_{y'z'},]^T$

The transformation matrix (rotation about direction 3) is

$$[TR] = \begin{bmatrix} m^2 & n^2 & 0 & 2mn & 0 & 0 \\ n^2 & m^2 & 0 & -2mn & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -mn & mn & 0 & m^2-n^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & m & n \\ 0 & 0 & 0 & 0 & -n & m \end{bmatrix} \tag{3.39}$$

where: $m = \cos(\beta)$ and $n = \sin(\beta)$

β = angle between fiber direction (1) and local coordinate ξ (rotation about 3 direction)

Using Equations (3.33) and (3.38), the stress-strain relation can be expressed as

$$\{\sigma'\} = [EI']\{e'\} \tag{3.40}$$

where

$$[EI'] = [TR][C][TR]^T \tag{3.41}$$

Now the global stiffness matrix [K] can be obtained as in Equation (3.5) where $[K]=\int_{\Omega} [BG]^T[C][BG]dV$. The only difference here is

that a local coordinate system is used as [C] [Equation (3.2)] become [EI'] [Equation (3.40)] and [BG] becomes [BF] as shown in Equations (3.19) and (3.28). Also, the integration volume is normalized: $dV=|J|d\xi d\eta d\zeta$ so that

$$[K] = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} [BF]^T[EI'][BF]|J|d\xi d\eta d\zeta \qquad (3.42)$$

where $-1\leq(\xi,\eta,\zeta)\leq 1$

The stiffness matrix [K] is symmetrical, a feature that will save computing time.

3.4 Surface Traction [Zienkewicz (1977)]

For a traction $\bar{t}$ applied to a face of the element, this traction can be decomposed into nodal forces {f} such as:

$$[K]\{\delta\} = \{r\} + \{f\} \qquad (3.43)$$

where {r} is the applied nodes forces

$$\{r\} = [r_{x_1} \ldots r_{x_{20}}, r_{y_1} \ldots r_{y_{20}}, r_{z_1} \ldots r_{z_{20}}]^T \qquad (3.44)$$

{f} is the force on the nodes corresponding to a traction

$$\{f\} = \int_{-1}^{+1} [N]^T.\bar{t}.dA \qquad (3.45)$$

For a $\zeta=1$ face we have the following vector product

$$\{dA\} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \end{bmatrix}^T \wedge \begin{bmatrix} \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \end{bmatrix}^T d\xi d\eta \qquad (3.46)$$

3.5 Stress at Nodes [Burnett (1987)]

Instead of calculating the strains (or stresses) at the nodes directly from the displacement field, we evaluate it at the optimal sampling points (gauss points) which are the same points that we used for integrating the [K] stiffness matrix. Then the strains (or stresses) are extrapolated to the nodes from these optimal sampling points. This provides a higher convergence rate [Burnett (1987)]. For this element, 27 sampling points are used (n=27).

The objective here is to obtain a function that can give stress or strain at any point inside the element from a linear function $\tau_S(\xi,\eta,\zeta)$ as

$$\tau_S(\xi,\eta,\zeta) = c_1\psi_1(\xi,\eta,\zeta)+\ldots+c_m\psi_m(\xi,\eta,\zeta) \qquad (3.47)$$

here $m = 4$

$\psi_1 = 1$

$\psi_2 = \xi$

$\psi_3 = \eta$

$\psi_4 = \zeta$

c_1, c_2, c_3, c_4 = undetermined coefficients

{c} is obtained using a regression technique, knowing the stress or strain values of the optimal points τ_{opt}, we can write

$$[P]\{c\} = [Q]\{\tau\}_{opt} \qquad (3.48)$$

where

$$P_{jk} = \sum_{i=1}^{n} \psi_j(\xi_i, \eta_i, \zeta_i)\psi_k(\xi_i, \eta_i, \zeta_i) \qquad (3.49)$$

$1 \le j \le m \quad 1 \le k \le m$

$$\{c\} = [c_1, c_2, c_3, c_4]^T \qquad (3.50)$$

$$Q_{jk} = \psi_j(\xi_k, \eta_k, \zeta_k) \qquad (3.51)$$

$1 \le j \le m \quad 1 \le k \le n$

$$\{\tau\}_{opt} = [\tau_1, \ldots, \tau_n]^T \qquad (3.52)$$

The optimal sampling points for each direction are:

$$-a, 0, a \quad \text{where} \quad a = \sqrt{3/5} \qquad (3.53)$$

Equation (3.48) can now be written as:

$$\{c\} = [S]\{\tau\}_{opt} \qquad (3.54)$$

where

$$[S] = [P]^{-1}[Q] \qquad (3.55)$$

By substituting Equations (3.49) and (3.51) into (3.55) yields:

$$[S] = \frac{1}{18a}\begin{bmatrix} 2a/3 & 2a/3 & 2a/3 & \cdots & & & & & & & & & & & & \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & \\ -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & \end{bmatrix}$$

$$\begin{bmatrix} \cdots & 2a/3 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix} \qquad (3.56)$$

Using Equation (3.47), values of strain and stress at the nodes

$\{\tau\}_{nodes}$ can be expressed as

$$\{\tau\}_{nodes} = [E]\{c\} \tag{3.57}$$

where

$$E(i,1) = 1$$

$$E(i,2) = \xi_i$$

$$E(i,3) = \eta_i$$

$$E(i,4) = \zeta_i \qquad 1 \le i \le 20 \tag{3.58}$$

Combining Equations (3.54) and (3.57) yields

$$\{\tau\}_{nodes} = [E][S]\{\tau\}_{opt} \tag{3.59}$$

3.6 Summary

The basic element relations are developed and ready for implementation in a computer program:

- Stiffness matrix
- Surface traction boundary condition
- Stress and strain extrapolation to the nodes from the optimal sampling points

A timed execution profile for the evaluation of the stiffness matrix showed that most of the computing time (75%) is spent on a mathematically simple task: multiplying matrices for the stiffness matrix in Equation (3.42).

CHAPTER 4

DETERMINING MESH SIZE

4.1 Introduction

The formulation given in the previous chapter was implemented in an extensive finite element program (26000 lines or 600 pages) as shown in appendix B. Results obtained from the finite element model were compared with the closed form solution for the two dimensional case. Finite element results for the three dimensional cases were also compared to experimental values from tapered specimens.

After agreement was established with these other two methods, the three dimensional finite element model was used to study interlaminar stresses inside tapered laminates of different configurations. The effect of different parameter variations upon interlaminar stresses was examined.

4.2 Mesh Determination Based on Closed Form Solution

The presence of layer drop-offs in tapered laminates creates regions of high stress gradients. In order to obtain accurate results at these locations, it is necessary to have fine meshes.

Since the calculated interlaminar stresses are known to depend on the mesh size [Lucking, Hoa and Sankar (1984)], it is necessary to validate the mesh before use. Hence, successively finer meshes were examined and results were compared against values obtained from the closed form solution (Table 2.7) for the problem: P=1, α=0, $\varepsilon=\frac{1}{3}$, R=1, r=1, CE 9000 glass/epoxy, unidirectional (fibers along the x direction). The meshes are shown in Figures 4.1, 4.2 and 4.3. Table 4.1 presents some information on these meshes. The results for the normalized stresses σ_x^* , σ_y^* and σ_{xy}^* ($\sigma_{ij}^*=\sigma_{ij}+P$) are compared as shown in Figures 4.5, 4.6 and 4.7 while the stress plots are given in Figures 4.8 to 4.10. Note that in Figures 4.5 to 4.7, the abscissa is the normalized contour around the triangular hole ($180^\circ \geq \theta \geq 0^\circ$). Figure 4.4 shows the process of normalization. Point a on the triangle is mapped as coordinate -1, point c is mapped as coordinate 0 and g becomes coordinate 1. Linear scale is used between a and c and between c and g.

TABLE 4.1

Information on Meshes

	Degrees of Freedom	Nodes	Elements
Mesh 0	635	252	27
Mesh 1	1231	485	56
Mesh 2	1231	485	56

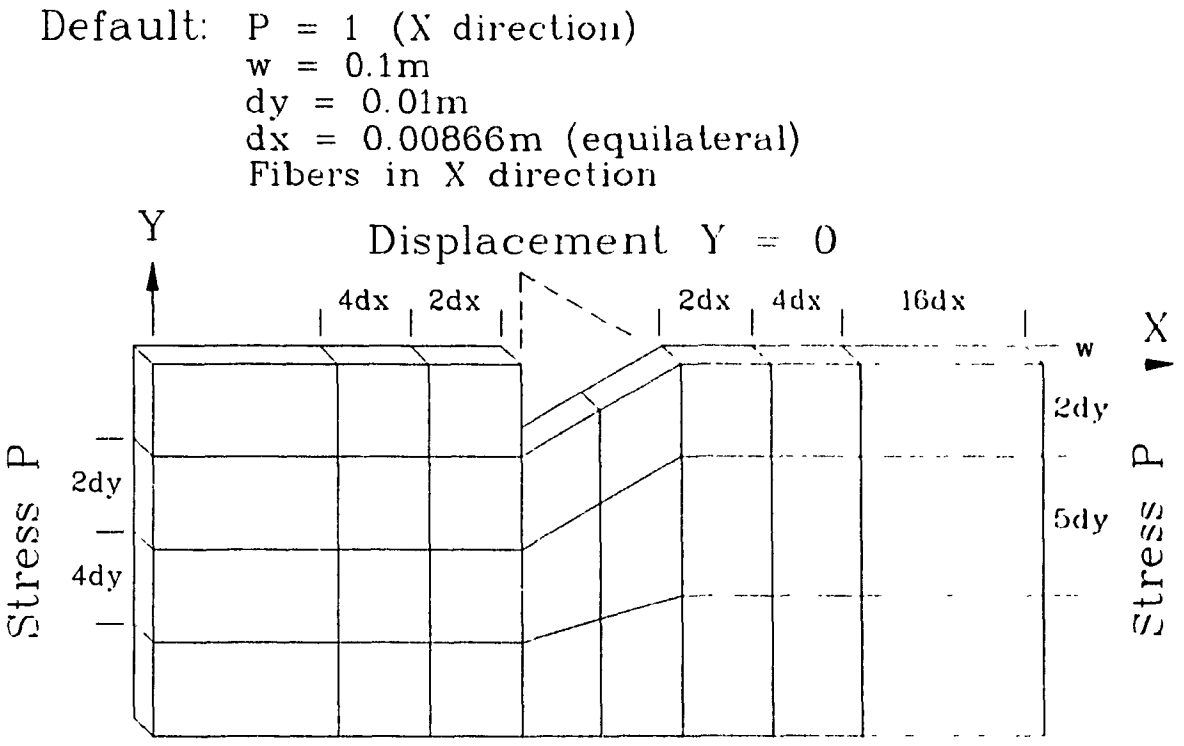


FIGURE 4.1

Equilateral triangular hole, mesh 0

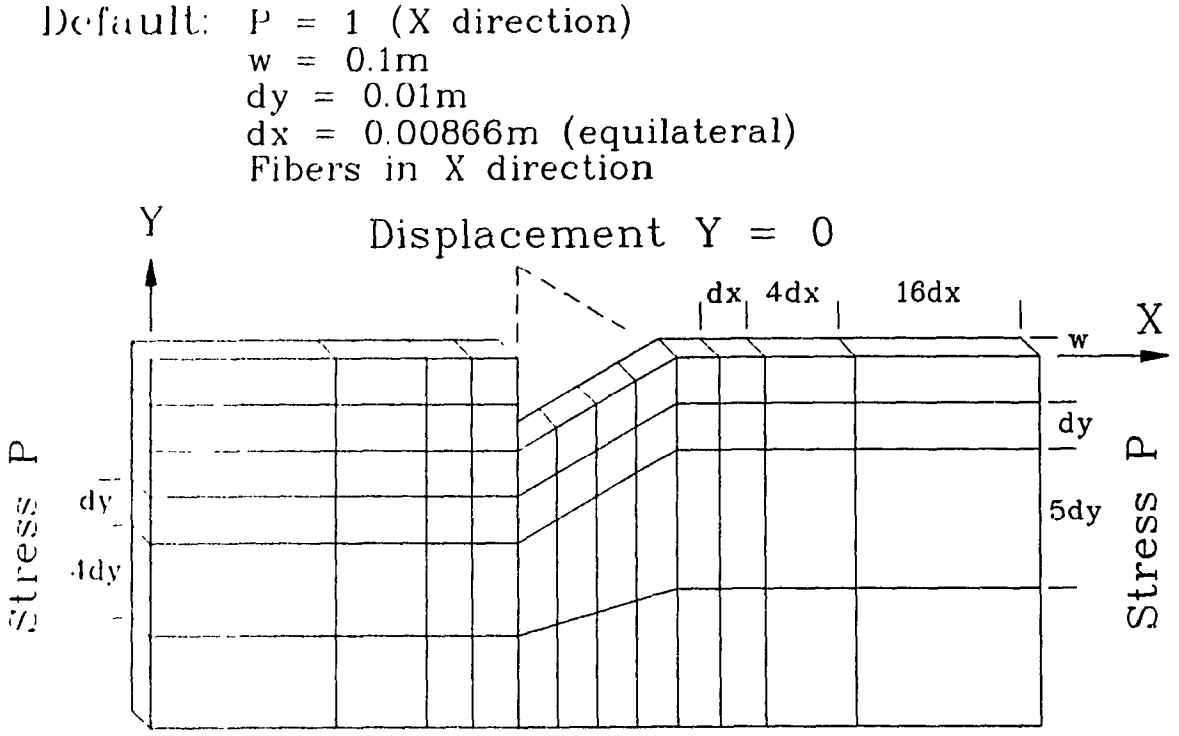


FIGURE 4.2

Equilateral triangular hole, mesh 1

Default: P = 1 (X direction)
w = 0.1m
dy = 0.01m
dx = 0.00866m (equilateral)
Fibers in X direction
f = 1/2
Displacement-Y = 0 on Y=0 face
Displacement-Z = 0 on Z=0 face

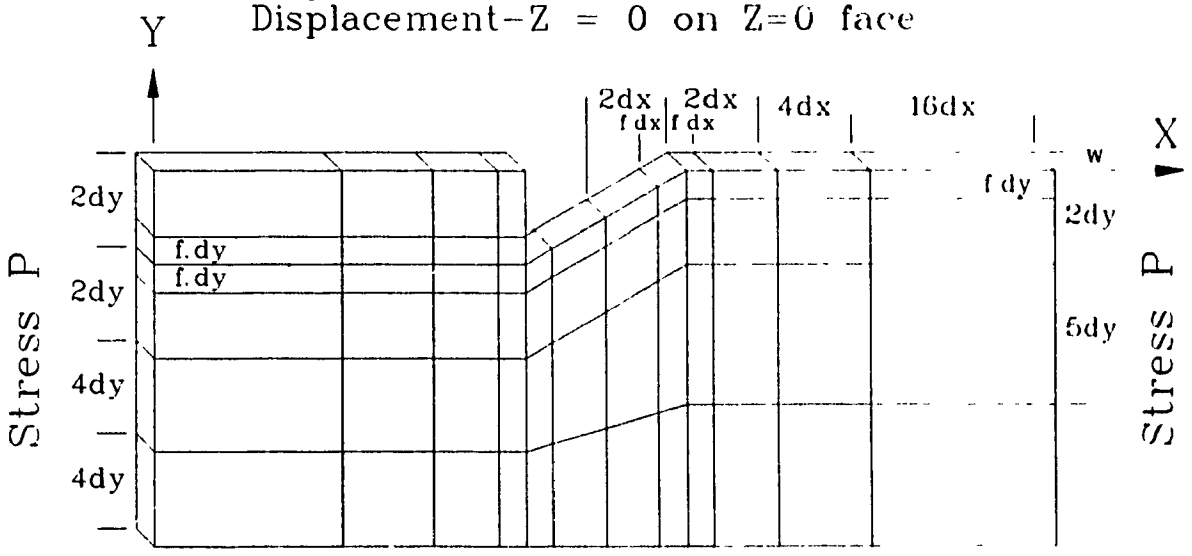


FIGURE 4.3

Equilateral triangular hole, mesh 2

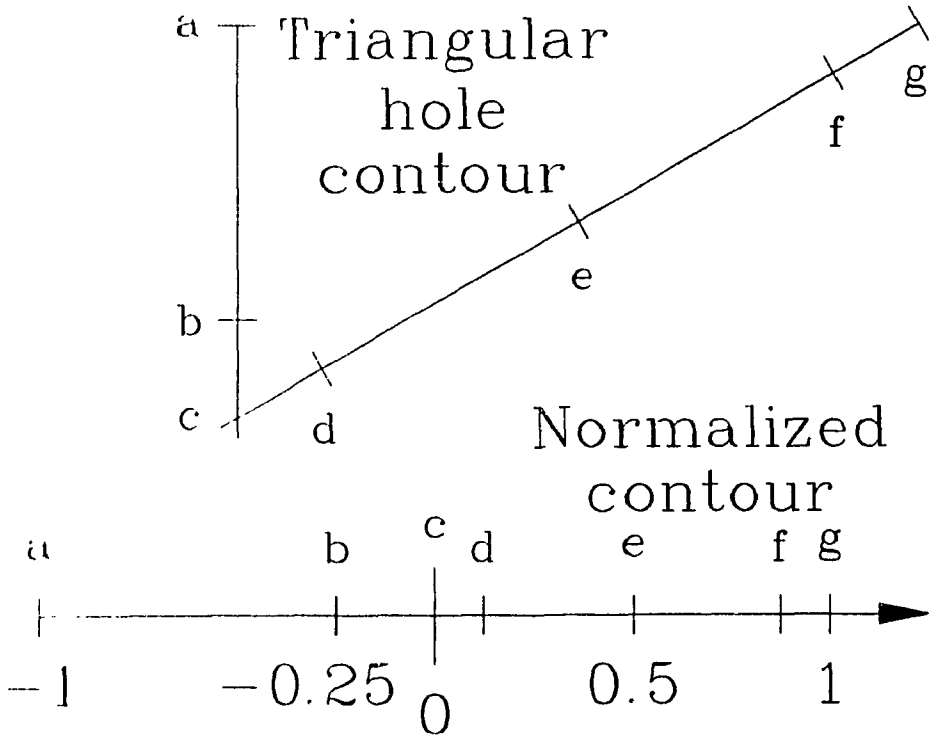


FIGURE 4.4

Triangle contour normalization

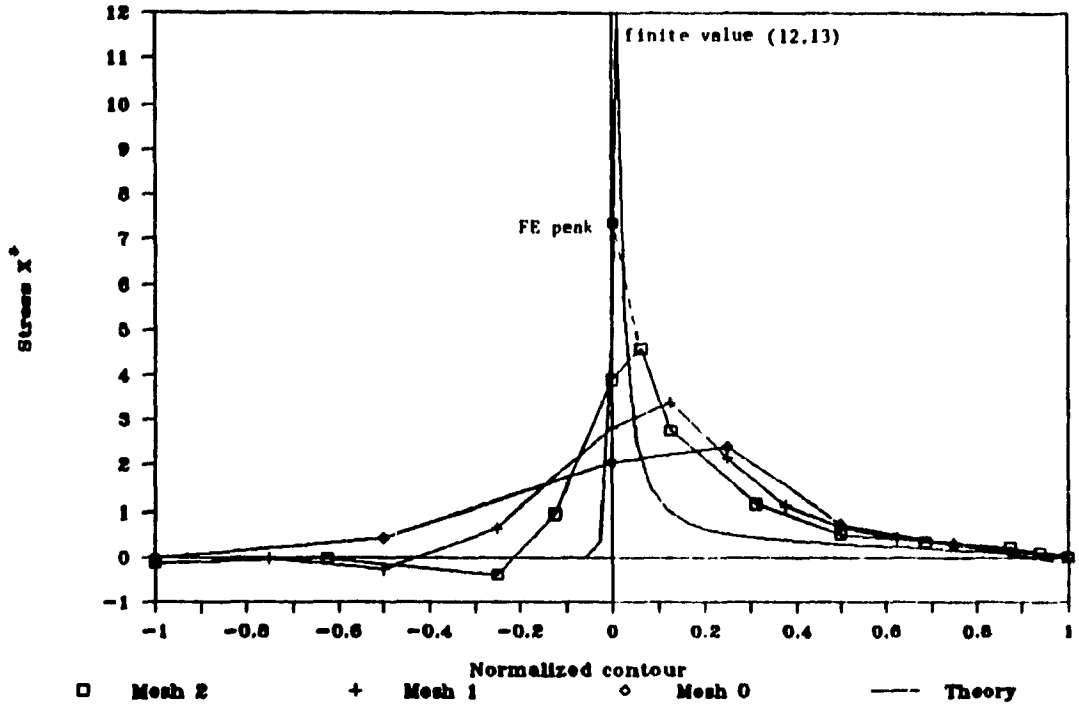


FIGURE 4.5

Theory and Finite Element Comparison, σ_x^*

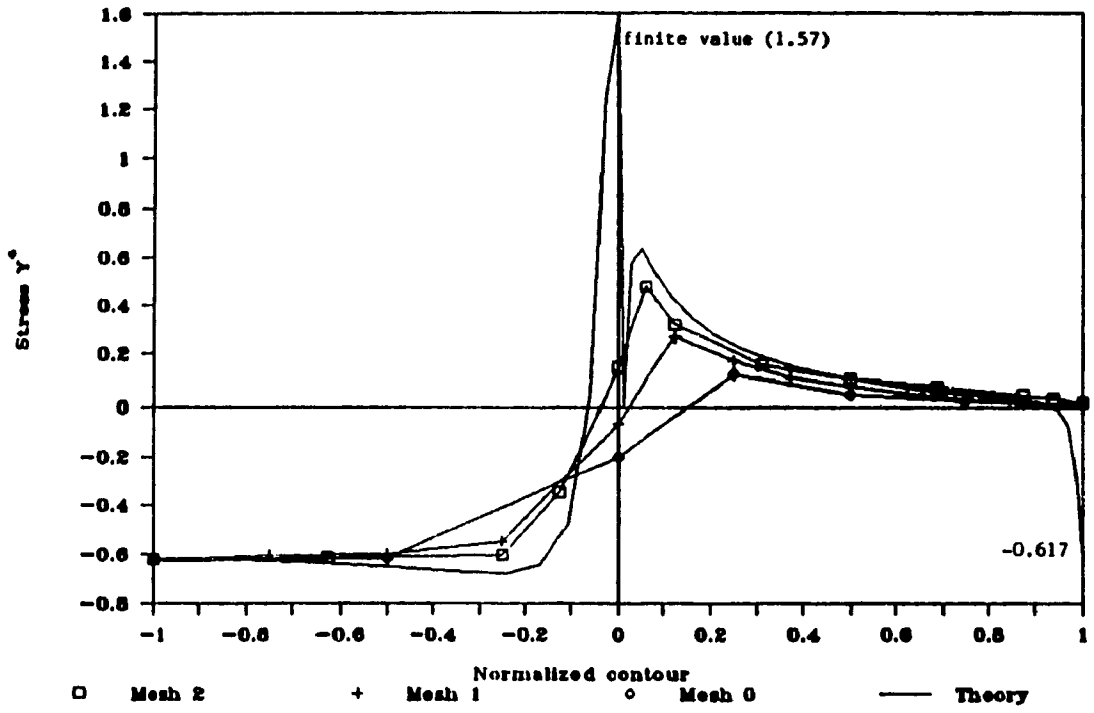


FIGURE 4.6

Theory and Finite Element Comparison, σ_y^*

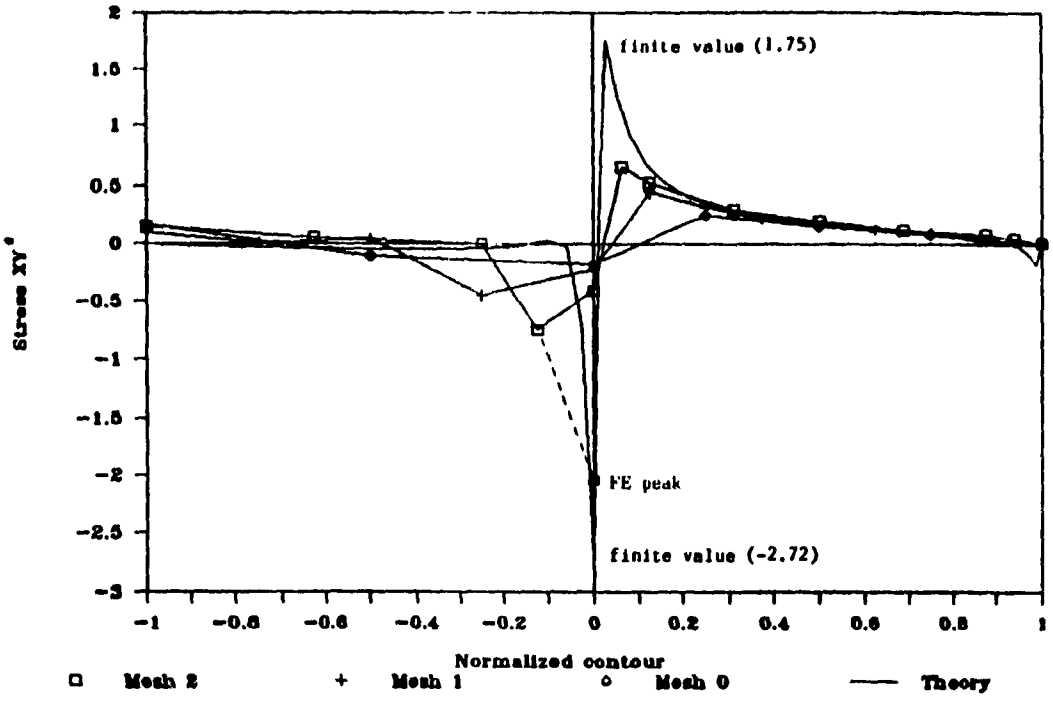
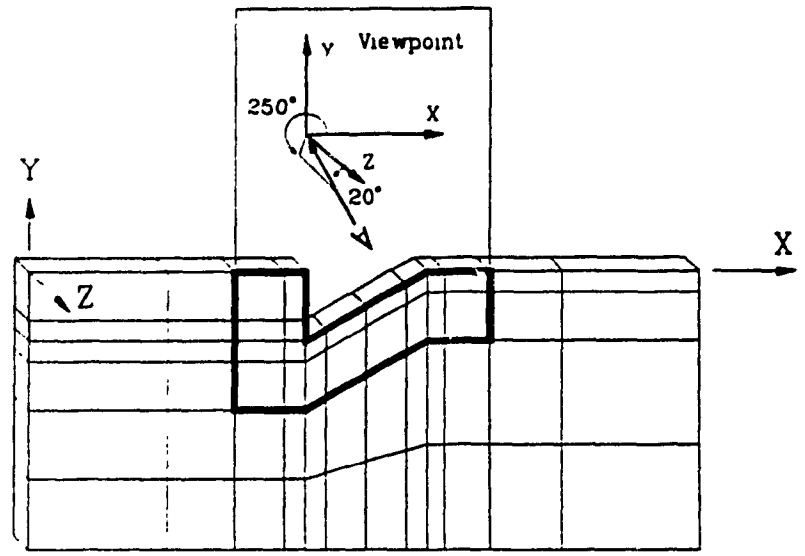


FIGURE 4.7

Theory and Finite Element Comparison, σ^*_{xy}



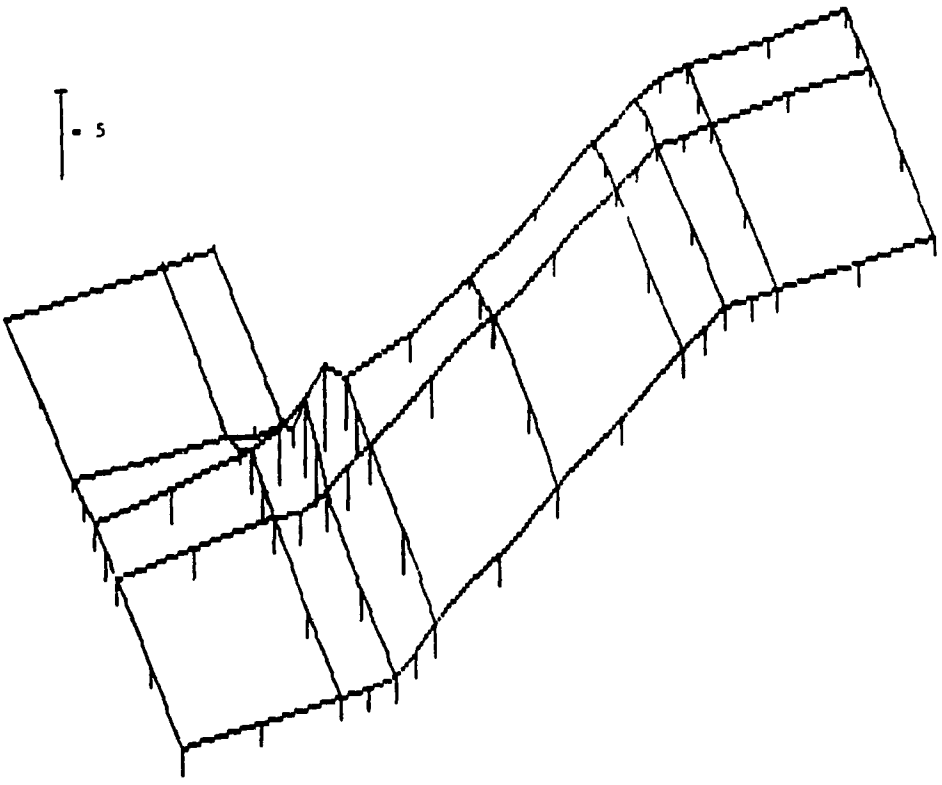
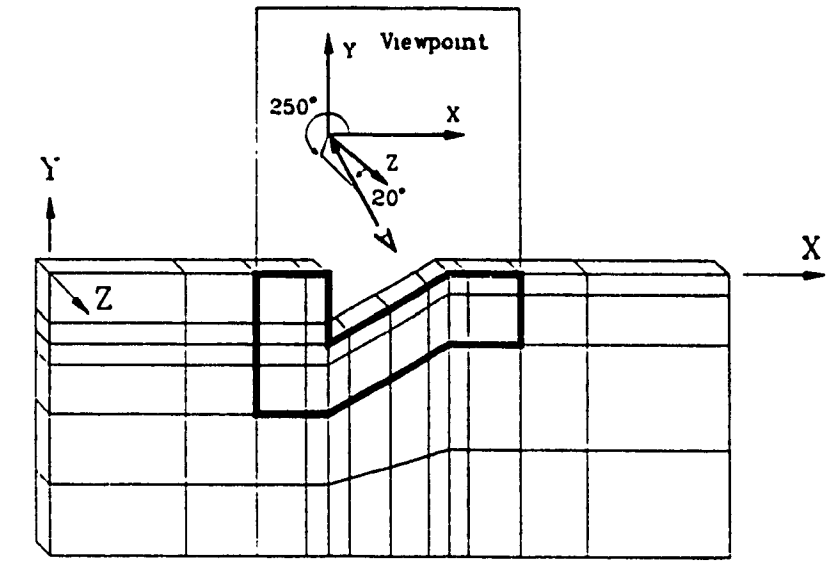


FIGURE 4.8

Finite Element, σ_x^* Distribution



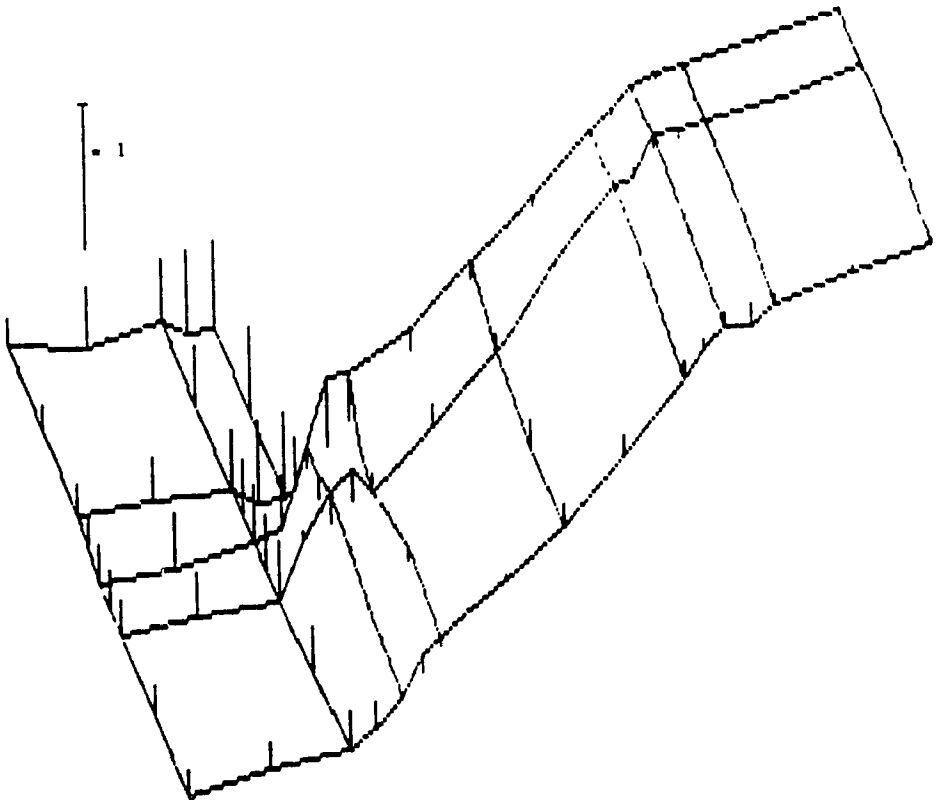
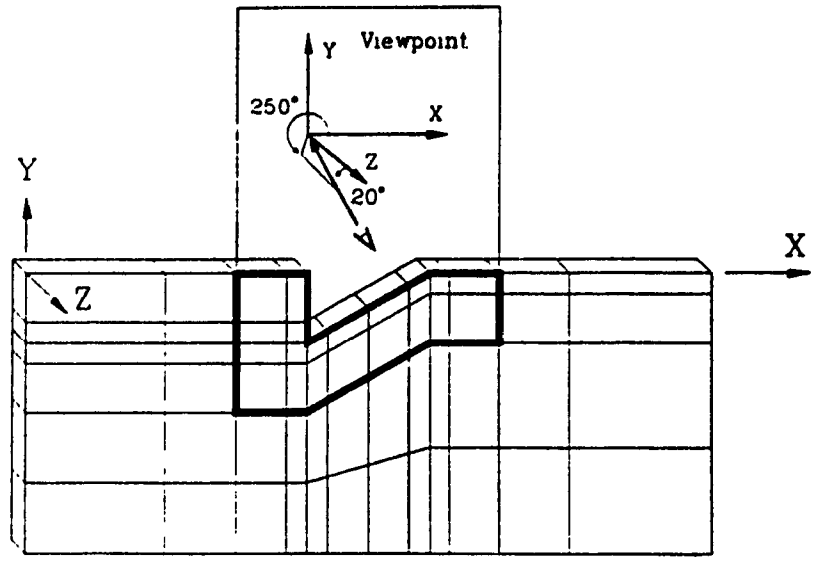


FIGURE 4.9

Finite Element, σ_y^* Distribution



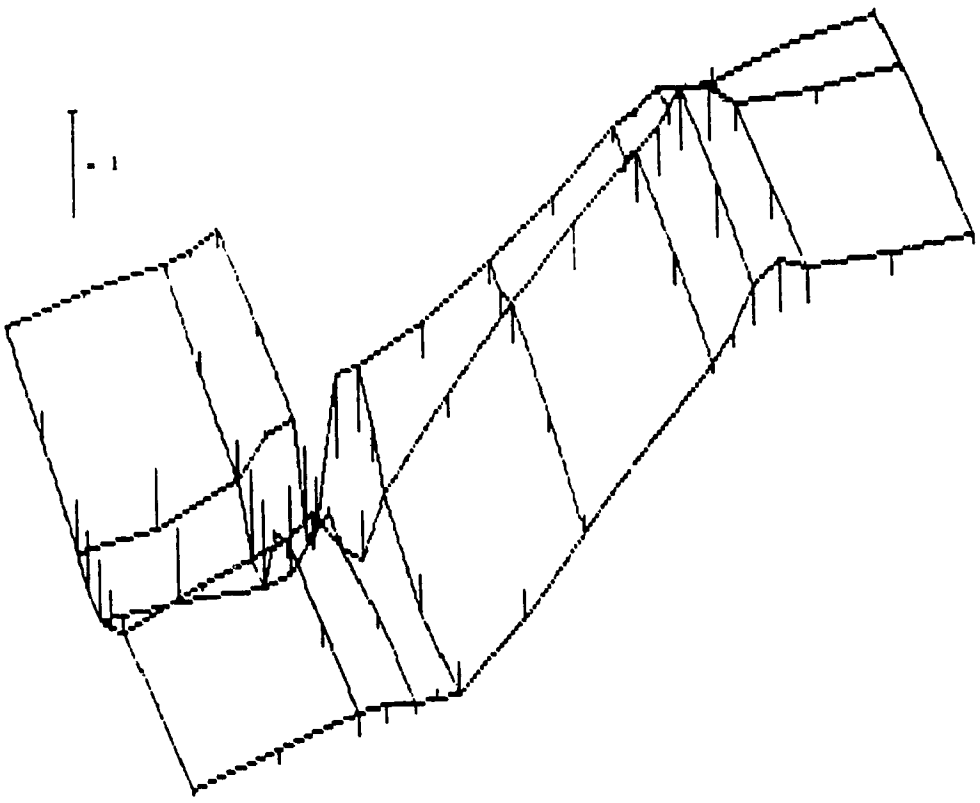


FIGURE 4.10

Finite Element, σ^{*}_{xy} Distribution

Mesh 2 showed the closest result with respect to the theory. As expected, σ_x^* is maximum at corner c (figure 4.4). refining the mesh improves the result but it does not give the value obtained from the closed form solution at location c. If the peak stress value is retained from the finite element (shown in Figures 4.5 and 4.7) solution, Table 4.2 shows better agreement for the peak stresses at $\vartheta=120^\circ$, σ_y^* from the closed form solution and from the finite element solution do not seem to agree. In the closed form solution, the value of σ_y^* at $\vartheta=0$ is always the same (for a given material) irrespective of the radius of curvature at point g (Figure 4.4). For an isotropic material, $\sigma_y^*=-1$ and for CE 9000 glass/epoxy $\sigma_y^*=-0.617$. Away from the location $\vartheta=0^\circ$, σ_y^* is a small positive value as can be seen by the agreement between the finite element and closed form solution. The disagreement only happens within a small region near $\vartheta=0^\circ$. As the radius of curvature decreases (sharper triangle), this region ($-\Delta\vartheta\leq\vartheta\leq\Delta\vartheta$), in which $\sigma_y^*<0$, diminishes. For an infinitely sharp triangle, one finds $\Delta\vartheta=0$. For the case of an infinitely sharp triangle the closed form solution would agree with the finite element solution of $\sigma_y^*\approx 0$ near $\vartheta=0^\circ$. The disagreement seen in Figure 4.6 is therefore due to the fact that the finite element is modelled as an infinitely sharp corner at point g in Figure 4.4 whereas the closed form solution requires a finite radius of curvature for its solution.

TABLE 4.2

Comparison of Critical Areas in Theory - Finite Element

	Stress (MPa)		
	Theory	FE	Difference
σ_x at $\theta=120^\circ$	12.1	7.32	39%
σ_{xy} at $\theta=120^\circ$	-2.72	-2.04	25%

4.3 Mesh Determination Based on Experiments

4.3.1 Experiments

As a further method for mesh validation, experiments on tapered specimens have been made. Failure stress was obtained from different meshes. A simple layup (Figure 4.11) was used for symmetric tapered specimen of 0° fiber on the outer plies. 0°, 45° and 90° fiber orientation was used between and at the drop-offs. This layup sequence is developed to examine the effect of fiber orientation at the drop-off on the strength of the laminate. Two drop-off locations were created to produce a sufficient taper.

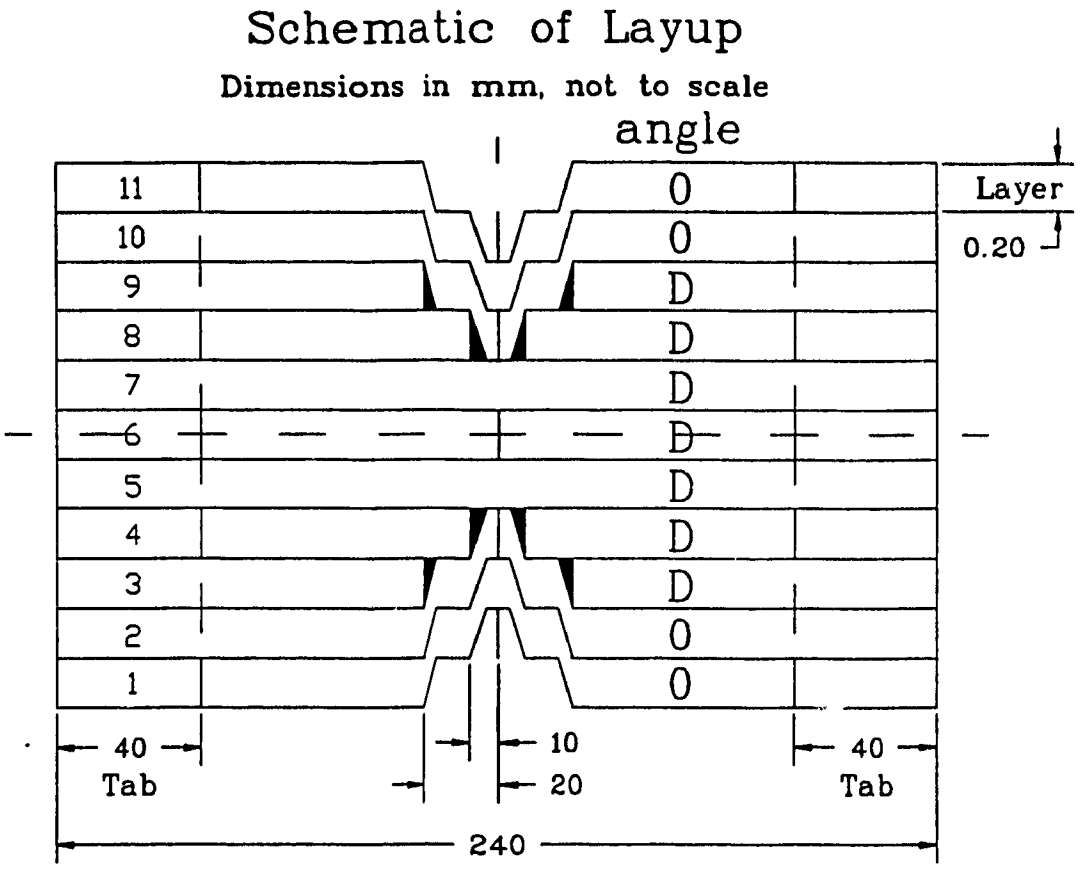


FIGURE 4.11

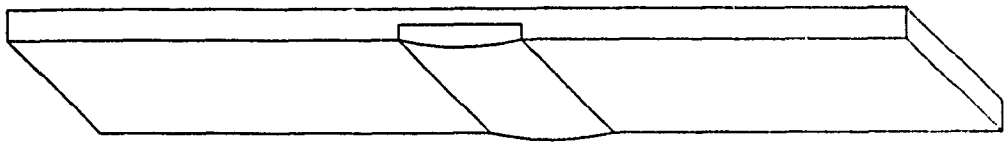
Layup Schematic

The material for the specimens is CE 9000 from Ferro corporation. It came in a "prepreg" (preimpregnated) form where the fibers were already set in the resin. Once cut and stacked, the laminate was sandwiched between resin-absorbing fabric and a tapered mold (Figure 4.12). Normal bagging procedure was used. The laminate was cured following the cure cycle shown in Figure 4.13. Specimens and their edges are shown in Figures 4.14 and 4.15. In these figures, the drop-off can be seen to have resin at the corner.

Using a MTS machine the samples were loaded in tension at a rate of 0.01 mm/s. The length of the specimens was 160 mm. Failure was defined to occur when a clear "plink" noise was heard from the sample. Damage was confirmed by the whitened area around the drop-off (Figure 4.16).

Table 4.3 gives the results obtained under tension for the following configurations:

$[0_2/D_{1.5}]_S$ with $D = 0^\circ$, 45° and 90°



Aluminum, not to scale, dimensions in mm

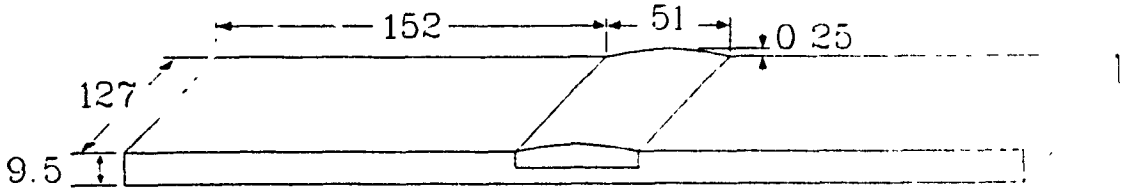


FIGURE 4.12

Mold

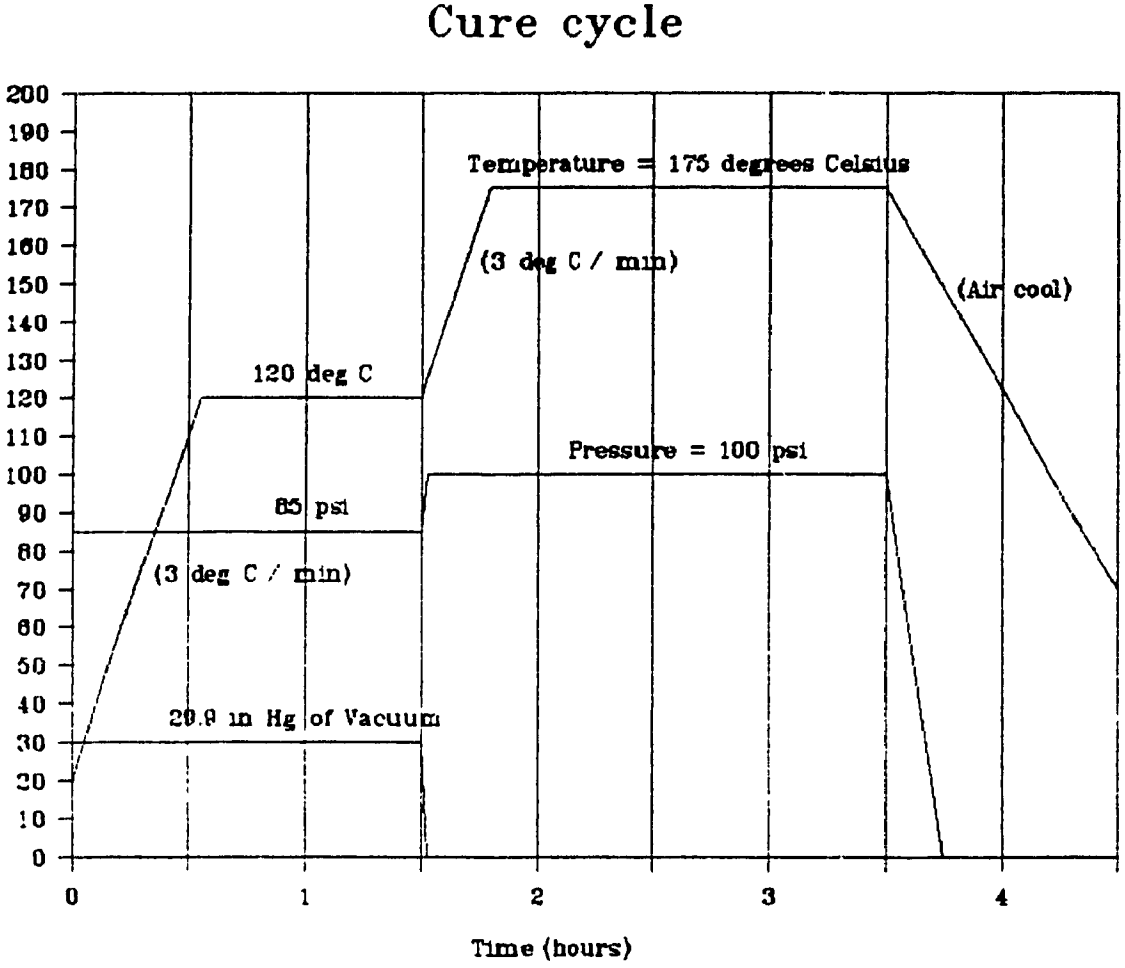


FIGURE 4.13

Cure Cycle

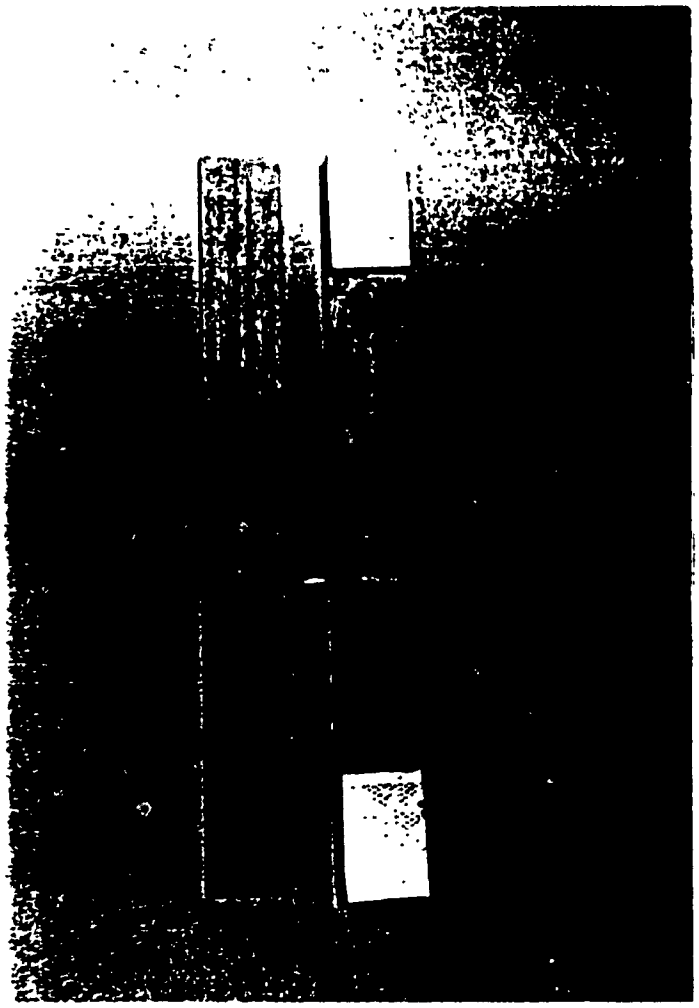


FIGURE 4.14

Tapered Specimens

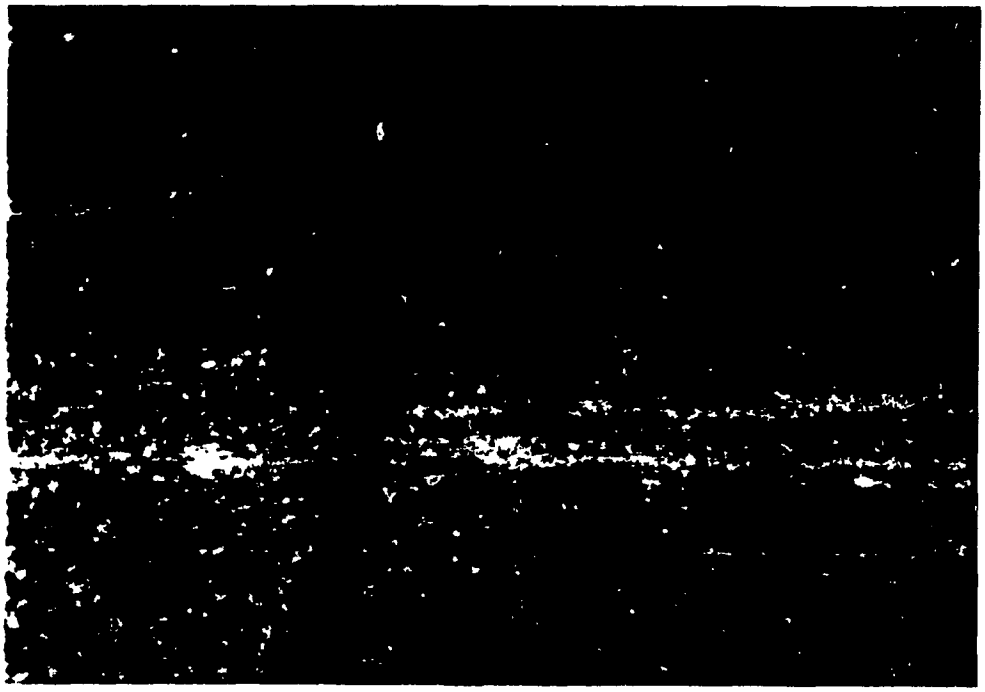


FIGURE 4.15

Specimen Edge

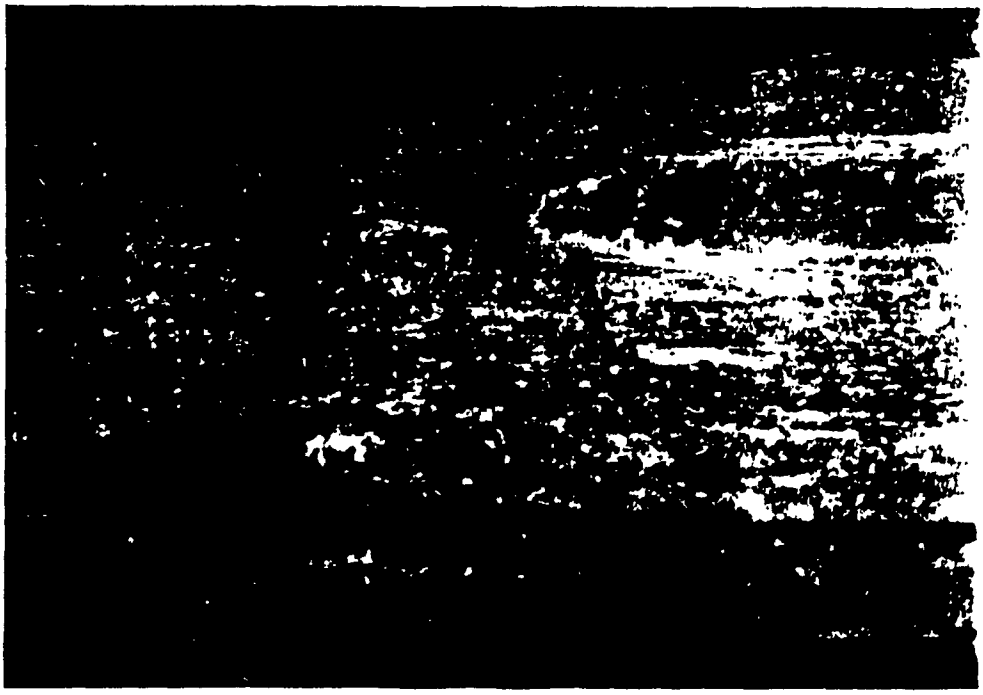


FIGURE 4.16

Failed Specimen Edge

TABLE 4.3

Experimental Results Under Tension

Angle D(°)	Specimen	Thickness (mm)	Width (mm)	Force (N)	Stress (MPa)	Average Stress (MPa)
0	1	1.31	25.4	4900	147	
	2	1.37	25.4	5250	151	
	3	1.34	25.4	4600	135	
	4	1.35	25.4	4770	139	
						143
45	1	1.48	10.60	1100	70.1	
	2	1.41	9.96	980	69.8	
	3	1.52	10.09	950	61.9	
	4	1.50	10.02	1000	66.5	
						67.1
90	1	1.64	9.55	510	32.6	
	2	1.62	9.45	510	33.3	
	3	1.61	9.35	540	35.9	
	4	1.63	9.73	540	34.0	
						33.9

A four point bending test was performed for determining the maximum bending stress at the outer fiber of the specimen. Table 4.4 presents results for a test of each configuration. Laminate theory was used to obtain σ_x at the extreme fiber location.

TABLE 4.4

Experimental Results Under Bending

	Width (mm)	Thickness (mm)	Moment (N.m)	$\frac{\text{Moment}}{\text{Width}}$ (N)	Bending Stress (MPa)
$[0_2/0_{1.5}]_S$	25.4	1.33	1.30	51.3	174
$[0_2/45_{1.5}]_S$	10.1	1.50	0.55	54.8	153
$[0_2/90_{1.5}]_S$	9.5	1.60	0.38	40.0	99

4.3.2 Meshes Considered

A three dimensional model of the specimen was made (Figure 4.17). D represents the angle of the fibers within the element coordinate system for the region shown. D = 0° indicates that the fibers are in the same direction as along the length of the laminate. A positive angle corresponds to the trigonometric rotation direction as seen from above the laminate.

A profile comparison is shown between a sample and the finite element model (Figure 4.17). For comparison purposes, the profile of the finite element model was scaled to match the average thickness of the specimen.

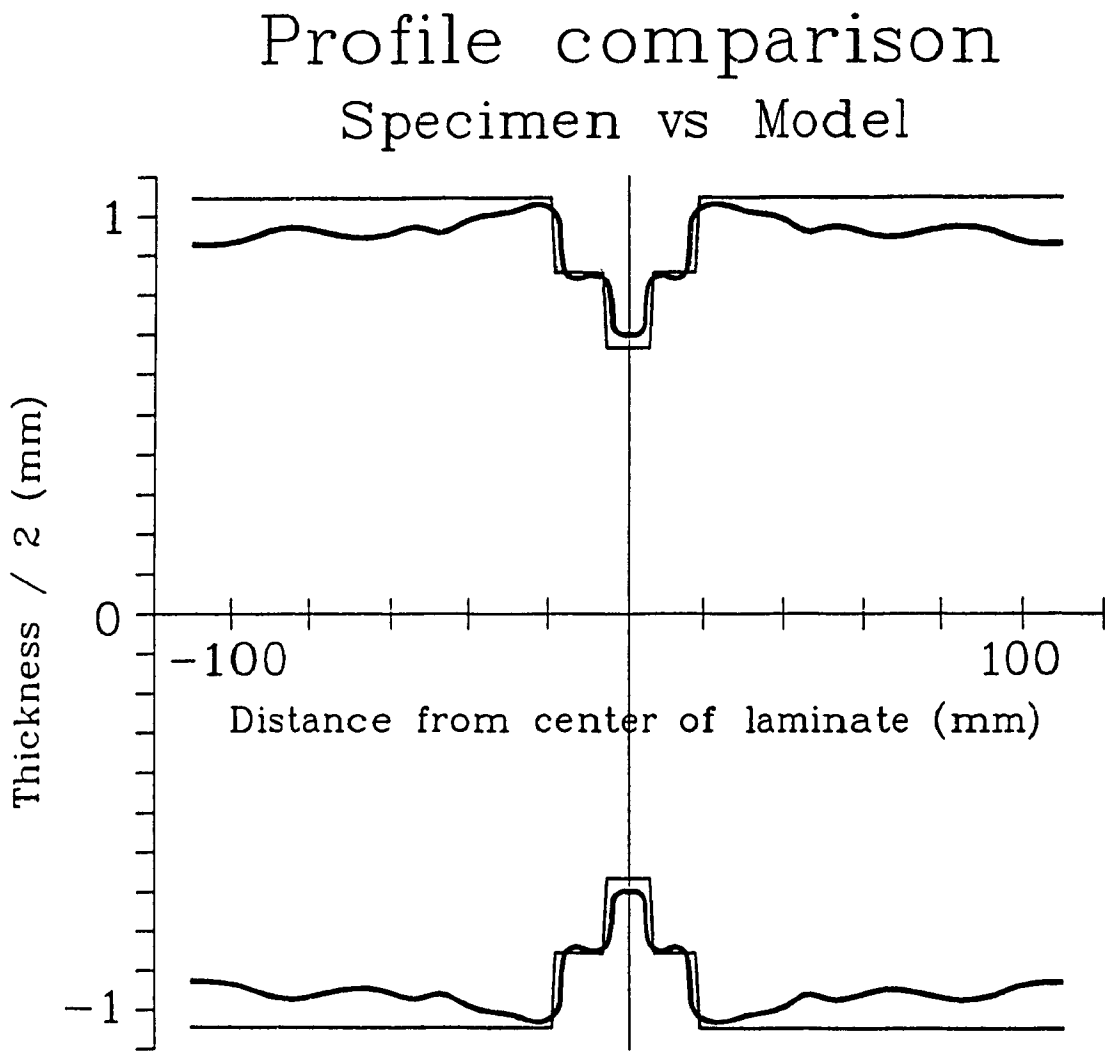


FIGURE 4.17

Profile Comparison

The first mesh (Figure 4.18) is known as a mesh level 0. Further mesh refinements are known as mesh 1, 2 and 3. Mesh level 3 was divided in two parts for greater calculation speed around the regions of interest.

Mesh level 0 and 1 were solved from the imposed stress boundary condition:

- The cross section normal to the laminate length direction in the smallest section was fixed.
- The x-z symmetry face (not shown) prevented any elevation displacements.
- The largest cross section which was normal to the laminate length direction had an imposed traction on every surface of the element which constituted the cross section.

Mesh 2 used the displacements obtained from mesh 1 along the thick contour line shown in Figure 4.19. The displacements from the nodes along the contour line in mesh 1 were imposed to mesh 2. The newly inserted nodes in mesh 2, which were not present in mesh 1, were interpolated using a 2^{nd} degree polynomial as to their coordinates and displacements. In a similar manner, mesh 3 used the results from mesh 2 (Figures 4.20 and 4.21).

A second model was also considered: the same geometry as the first model except the drop-off was considered filled with epoxy resin by including new elements in this region.

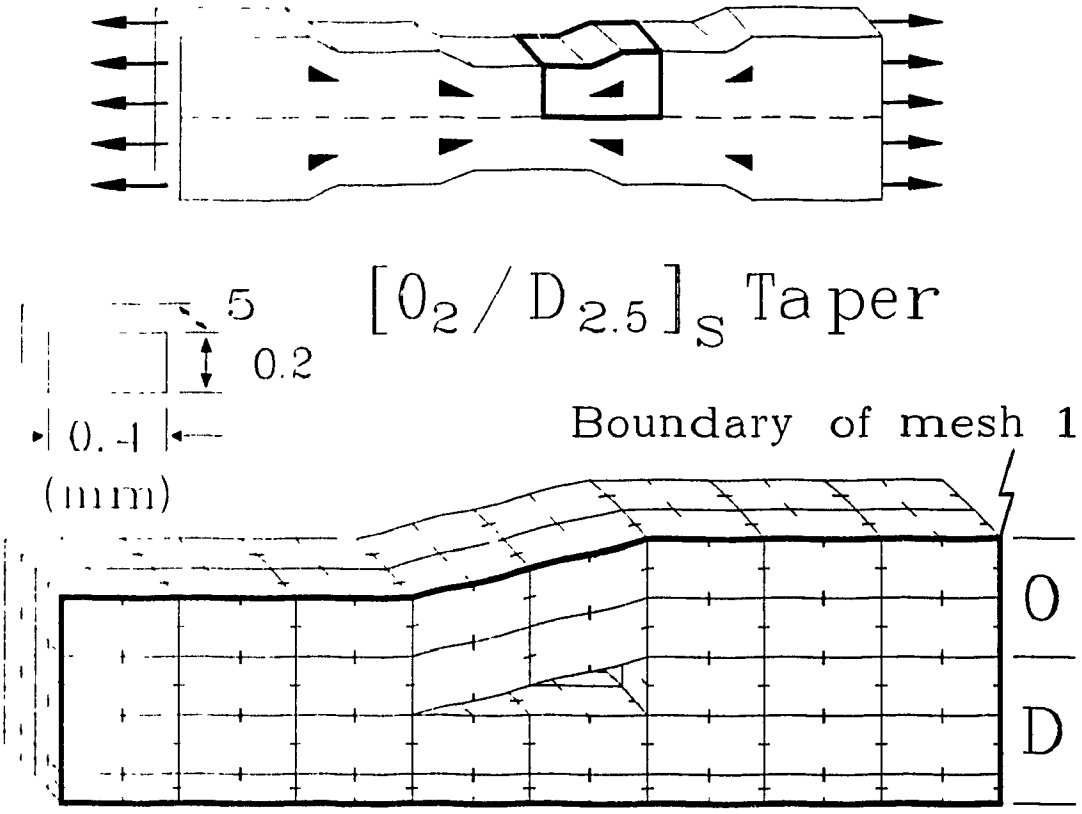


FIGURE 4.18

Mesh O

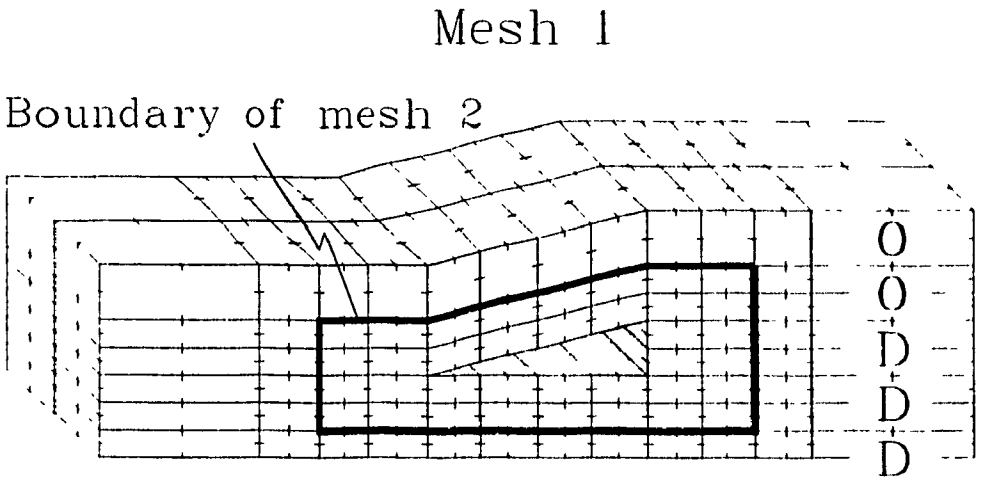


FIGURE 4.19

Mesh 1

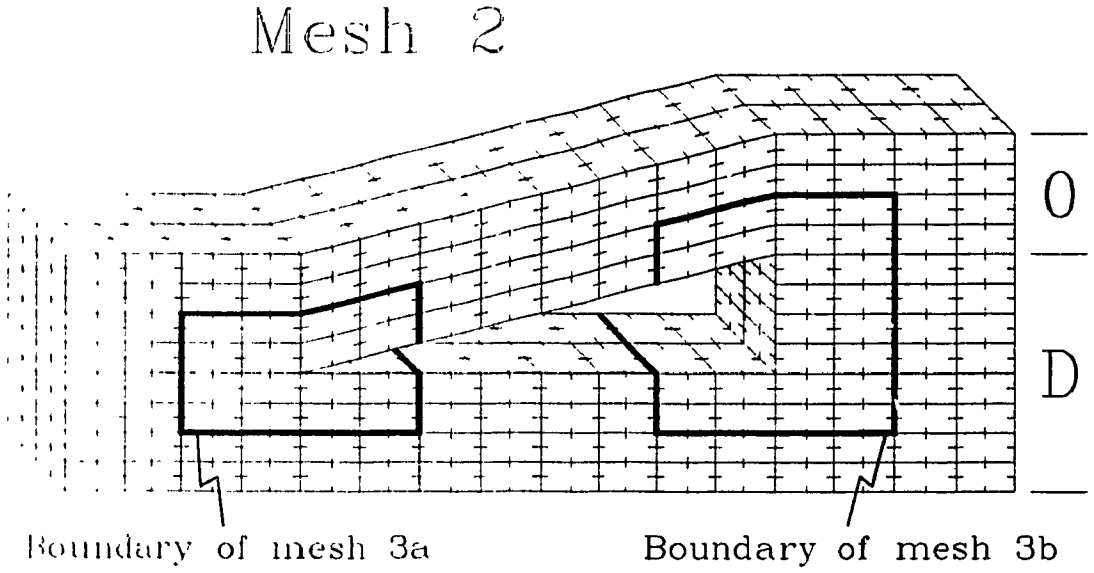


FIGURE 4.20

Mesh 2

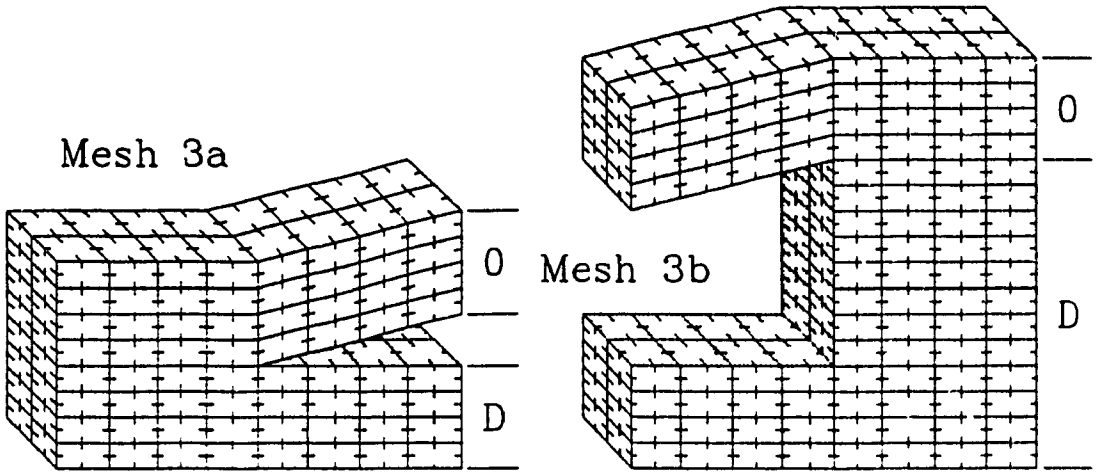


FIGURE 4.21

Mesh 3a and 3b

Another mesh, shown in Figure 4.22, was generated to allow for bending loading conditions. Applying forces on a face which correspond to bending loading condition was considerably simplified by extending the geometry to include a wall of "rigid" elements. The stiffness of these last elements was set to 1000 times that of the glass/epoxy properties found in the laminate. Using 16 digit computation accuracy, this did not induce any ill-conditioning of the stiffness matrix. This rigid wall of elements evenly distributed the load forces to the mating surface, while the bending moment was simply applied by two forces at the upper and lower extremities of the wall. Torsion loading conditions were applied in a similar manner.

As is shown in later figures plotting stress distributions (Figures 4.24 to 4.41), the local mesh refinement was justified by the distance to which the stress disturbance extends from the drop-off area.

Table 4.5 shows the time for solving the cases using the slowest IBM microcomputer available (IBM XT™ 4.77 MHz):

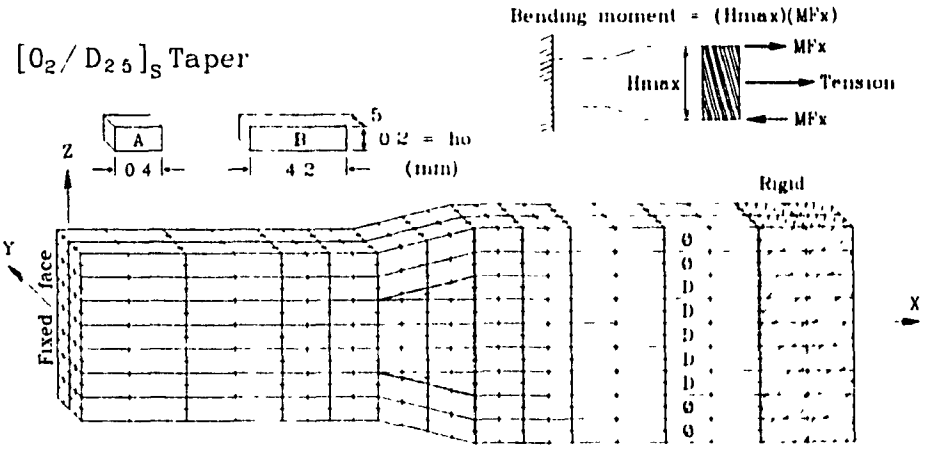


FIGURE 4.22

Bending Mesh

TABLE 4.5

Solving Time

	Time (Hour:Min)	Degrees of Freedom
Without Resin		
Mesh 0	2:55	1408
Mesh 1	9:50	3061
Mesh 2	22:10	4368
Mesh 3a	4:09	1836
Mesh 3b	6:17	2880
With Resin		
Mesh 0	3:05	1417
Mesh 1	11:45	3196
Bending Mesh	12:45	3267

Tests done on a leading 80386 based personal computer showed that these times drop by a factor of 10, bringing the maximum time in mesh 2 down to 2 hours from 22 hours.

4.3.3 Comparison of Results from Different Meshes

Tapered laminates were subjected to a uniaxial tensile stress of 1 MPa imposed at the largest section of mesh 0 and 1. Stress distribution plots show that the disturbances caused by the drop-off vanish before the fixed face (smaller cross section) of the model. The fixed face can be considered to have a normal

stress of $\frac{9}{7}$ = 1.286 MPa due to the thickness ratio of the large section (9 layers) and small section (7 layers).

Results will be presented by the allowable stress ratio:

$$R = \frac{\text{Stress}}{\text{Strength}} = \frac{1}{\text{Safety Factor}}$$

where strengths were obtained experimentally as shown in Table 2.2. The stress components that is used in this equation (σ_x, σ_y, σ_z, σ_{xy}, σ_{xz} or σ_{yz}) depends on which one that is closest to the corresponding strength. For example if σ_x is closest to X then $R=\sigma_x+X$. However if σ_y is closer to Y then $R=\sigma_y+Y$. Table 4.6 shows the stress ratios for a few fiber orientation D using the meshes shown in Figures 4.18 to 4.21. In table 4.6, a smaller number indicates a safer case. For example, the case where $D=0^\circ$ is safer than the case where $D=45^\circ$ or $D=90^\circ$. Also, the case where the drop-off is filled with resin is safer than the case without.

TABLE 4.6

Stress Ratio R of Meshes under tension

	D		
	0°	45°	90°
Empty Drop-off			
Mesh 0	0.0110	0.0253	0.0480
Mesh 1	0.0161	0.0293	0.0576
Mesh 2	0.0222	0.0352	0.0717
Mesh 3	0.0303	0.0437	0.0926
Filled Drop-off			
Mesh 0	0.0093	0.0211	0.0408
Mesh 1	0.0121	0.0228	0.0419

Figure 4.23 shows these results in a graphic manner. Two relations can be obtained from this figure. First, the safety factor decreases with the orientation D of the fiber as D increases from 0° to 90°. Second, the finer is the mesh, then the smaller is the safety factor. It is therefore necessary to be cautious in the selection of the mesh size: as the mesh is refined the percent increase of R value was calculated using the expression

$$\% = \frac{R_i - R_{i-1}}{R_{i-1}} \times 100$$

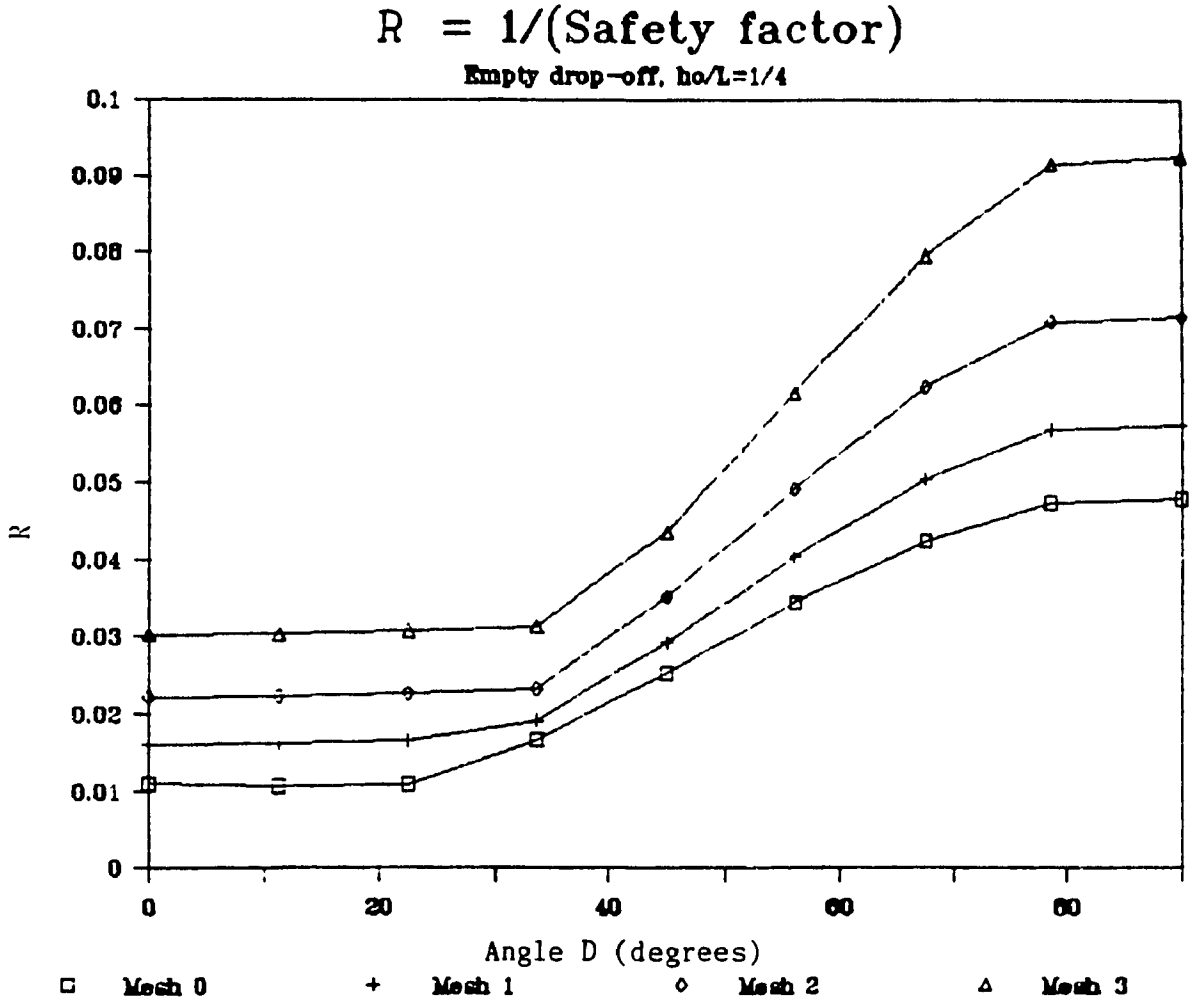


FIGURE 4.23

Effect of Mesh Refinement on R

where i indicates the mesh number. These values are shown in Table 4.7. Those numbers do not indicate any trend. It is possible of course to refine the mesh infinitely depending upon the computer resources available. However, if the size of the finite element is smaller than the size of the unit cell dimension (fiber plus some surrounding matrix) the effective modulus assumption breaks down and the finite element method based upon these effective modulus is no longer valid. Table 4.8 shows the size of the smallest element used with effective modulus. Comparing with a fiber diameter of 10 μm, it can be seen the smallest element in mesh 3 has already reached the size of the unit cell. It was then decided not to refine the mesh any further. Selection of a workable mesh was obtained by comparing with experiments.

TABLE 4.7

Stress Ratio R Increase with Mesh Size

	Fiber Orientation		
	0°	45°	90°
Empty Drop-off			
Mesh 0 → 1	46%	16%	20%
Mesh 1 → 2	38%	20%	25%
Mesh 2 → 3	36%	24%	29%
Filled Drop-off			
Mesh 0 → 1	30%	8%	3%

TABLE 4.8

Mesh Size

Mesh	Size (μm)
0	200
1	100
2	50
3	25

The diameter of a glass fiber is approximately 10 μm

4.3.4 Mesh Size Selection

Using the experimental values obtained in section 4.3.1, the maximum stress failure criterion and the result obtained for the three meshes, the finite element results can be compared with experimental results. This is shown in Table 4.9. To get the finite element stress, the values in table 4.6 are used. For example, in the case of D=0° with a drop-off filled with resin and mesh 0, 1/0.0093 is multiplied by the applied stress of 1.286 MPa to give a stress of 138 MPa. It can be seen from Table 4.9 that mesh 0 with a filled drop-off gives the closest failure stress as compared to experimental values.

TABLE 4.9

Experimental and Numerical Comparison for Tension

		Fiber Angle D		
		0°	45°	90°
Failure Stress (MPa)	Experimental	143	67	34
	Empty drop-off			
	F.E. Mesh 0	117	51	27
	F.E. Mesh 1	80	44	22
	F.E. Mesh 2	58	37	18
	F.E. Mesh 3	42	29	14
	Filled drop-off			
	F.E. Mesh 0	138	61	32
	F.E. Mesh 1	106	56	31

Mesh 0 with a filled drop-off was therefore selected for further generation of meshes. Using this mesh for the bending case also shows that mesh 0 with a filled drop-off gives good agreement (Table 4.10).

TABLE 4.10

Experimental and Numerical Comparison for Bending

		Fiber Angle		
		0°	45°	90°
Failure Stress (MPa)	Experimental	174	153	99
	F.E. Mesh 0, filled drop-off	168	146	94

4.4 Stress Distributions Presented for Mesh 0

Figures 4.24 - 4.29 respectively show the distribution of the normalized stresses along the element coordinate system, σ'_x, σ'_y, σ'_z, $\sigma'_{x'y}$, $\sigma'_{x'z}$, $\sigma'_{y'z}$, , for mesh 0 (Figure 4.22) with the full drop-off and all 0° fiber orientation. The tensile load applied produces an average σ_x stress of 55.6 MPa ($1x10^3$ N divided by an area of 1cm x 9 plies of 0.2 mm) which is normalized to become 1. Figures 4.30-4.35 and 4.36-4.41 show the same stresses for $[0_2/45_{1.5}]_S$ and $[0_2/90_{1.5}]_S$ respectively. From these figures it can be seen that σ'_x and σ'_y are being reduced in the region between the drop-offs for an increasing angle D from 0° to 90°. σ'_z remains similar for any D. σ'_{xy} σ'_{yz} are only of significance for $D=45^\circ$. As D increases from 0° to 90°, the disturbance for σ'_{xz} in the region between the drop-offs is reduced.

Figures 4.42 to 4.47 show the stresses in element coordinates (same as fiber direction) in the plane where x=Drop-off for all 0° fiber orientation. By looking at the figures the "W" shaped lines represent the stress at the drop-off. The shape and level of σ'_x, σ'_y, σ'_z, $\sigma'_{x'z}$, remain the same throughout the width of the laminate. $\sigma'_{x'y}$, $\sigma'_{y'z}$, changed but the distributions are anti-symmetric with respect to the Y-0 (mid-width) plane. Stresses σ'_y , σ'_{xy} and σ'_{yz} are negligible compared to other stresses.

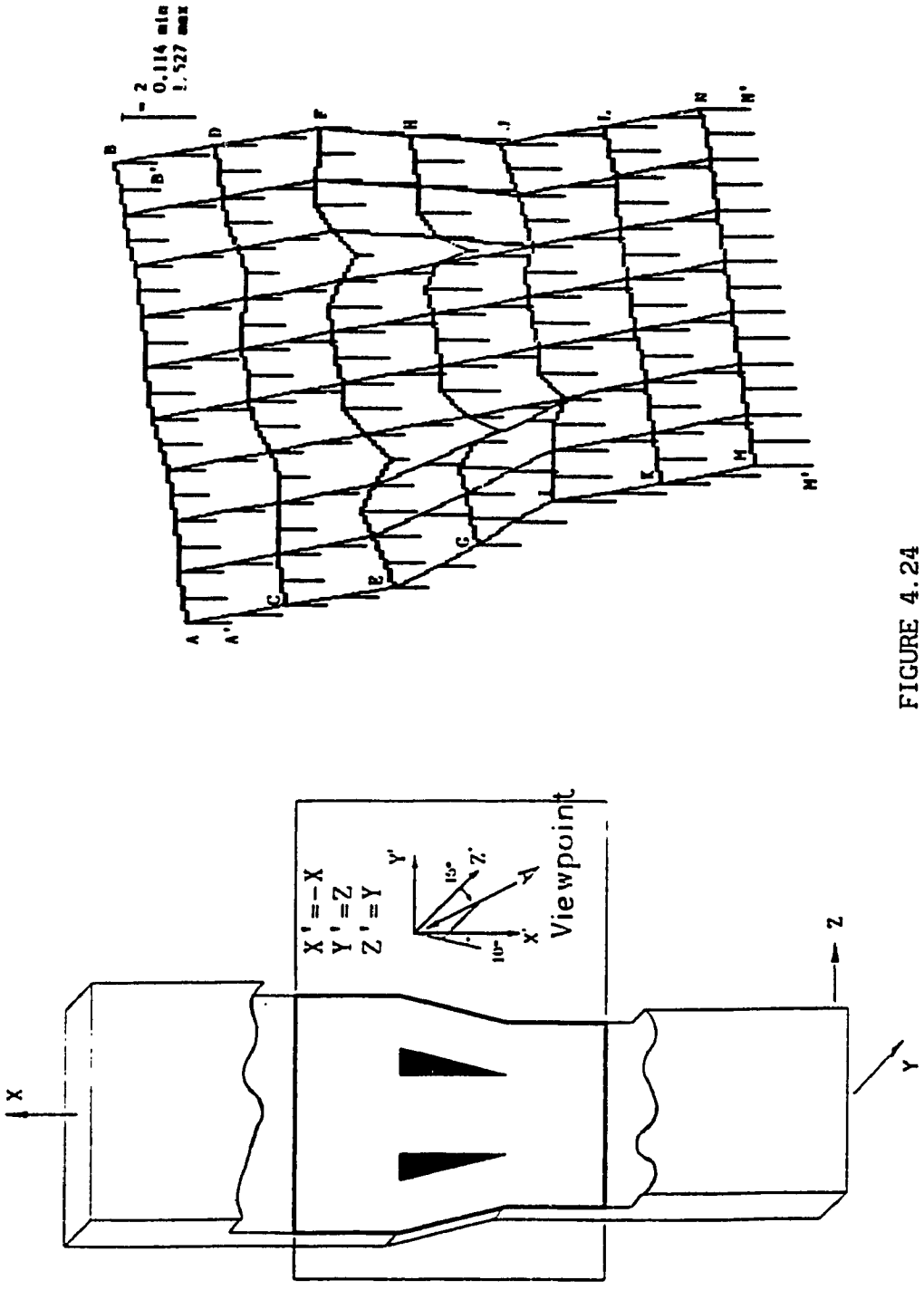


FIGURE 4.24

$\sigma^*_{x'}$, Tension Load, $[0_2/0_{2.5}]_S$, Y=0

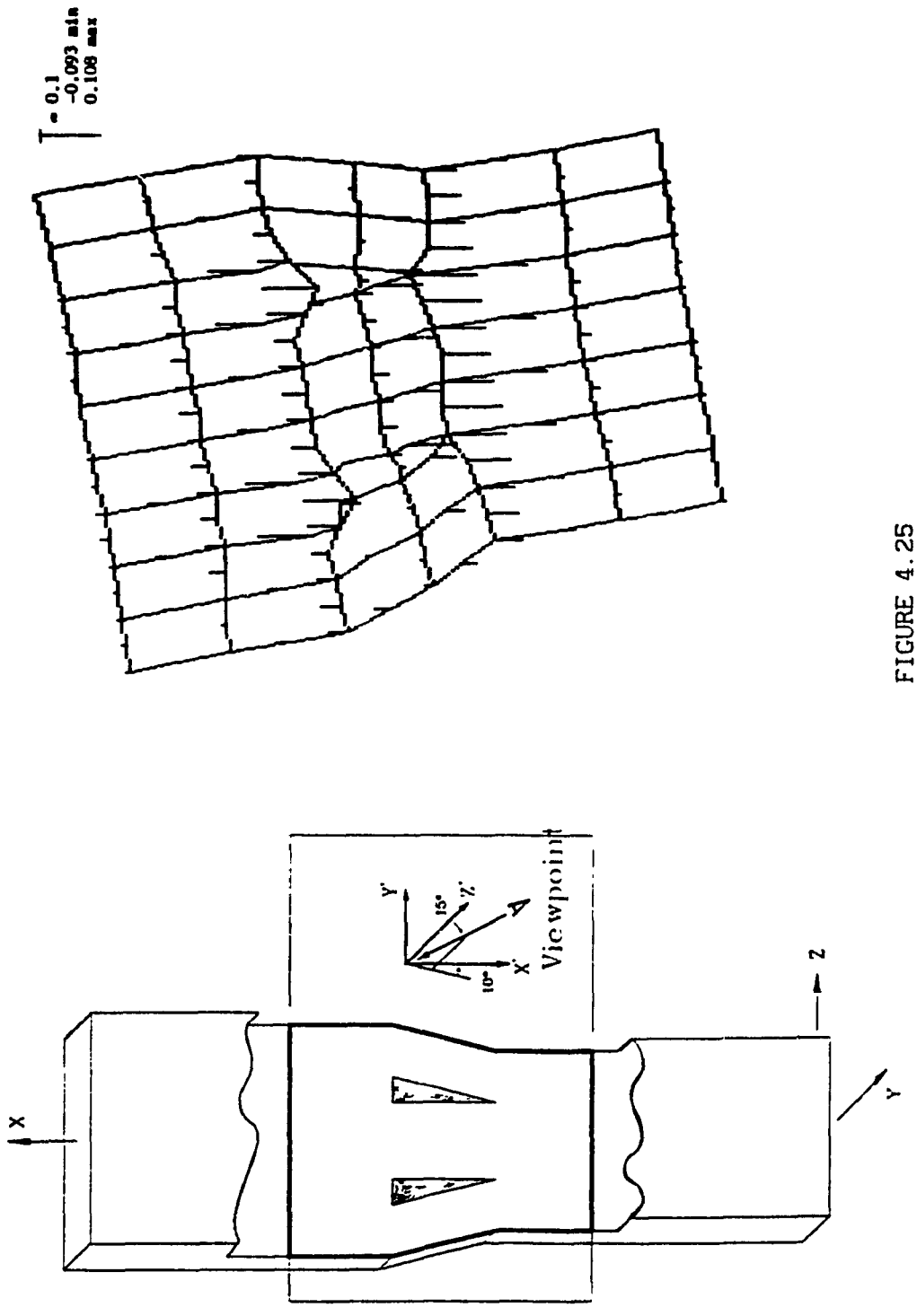


FIGURE 4.25

$\sigma_{y'}$, , Tension Load, $[0_2/0_{2.5}]_S$, Y=0

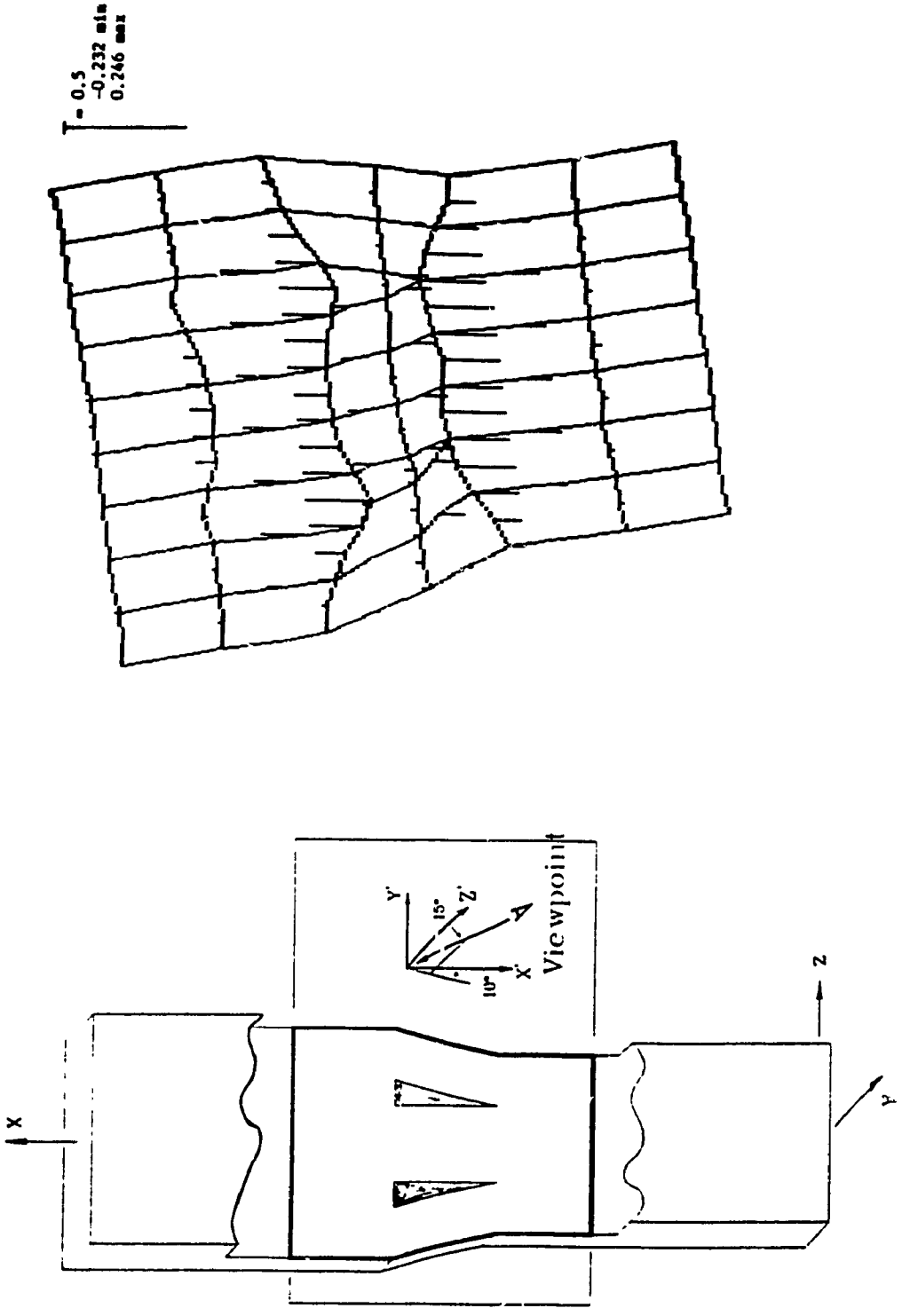


FIGURE 4.26

σ_z^*, , Tension Load, $[0_2/0_{2.5}]_S$, Y=0

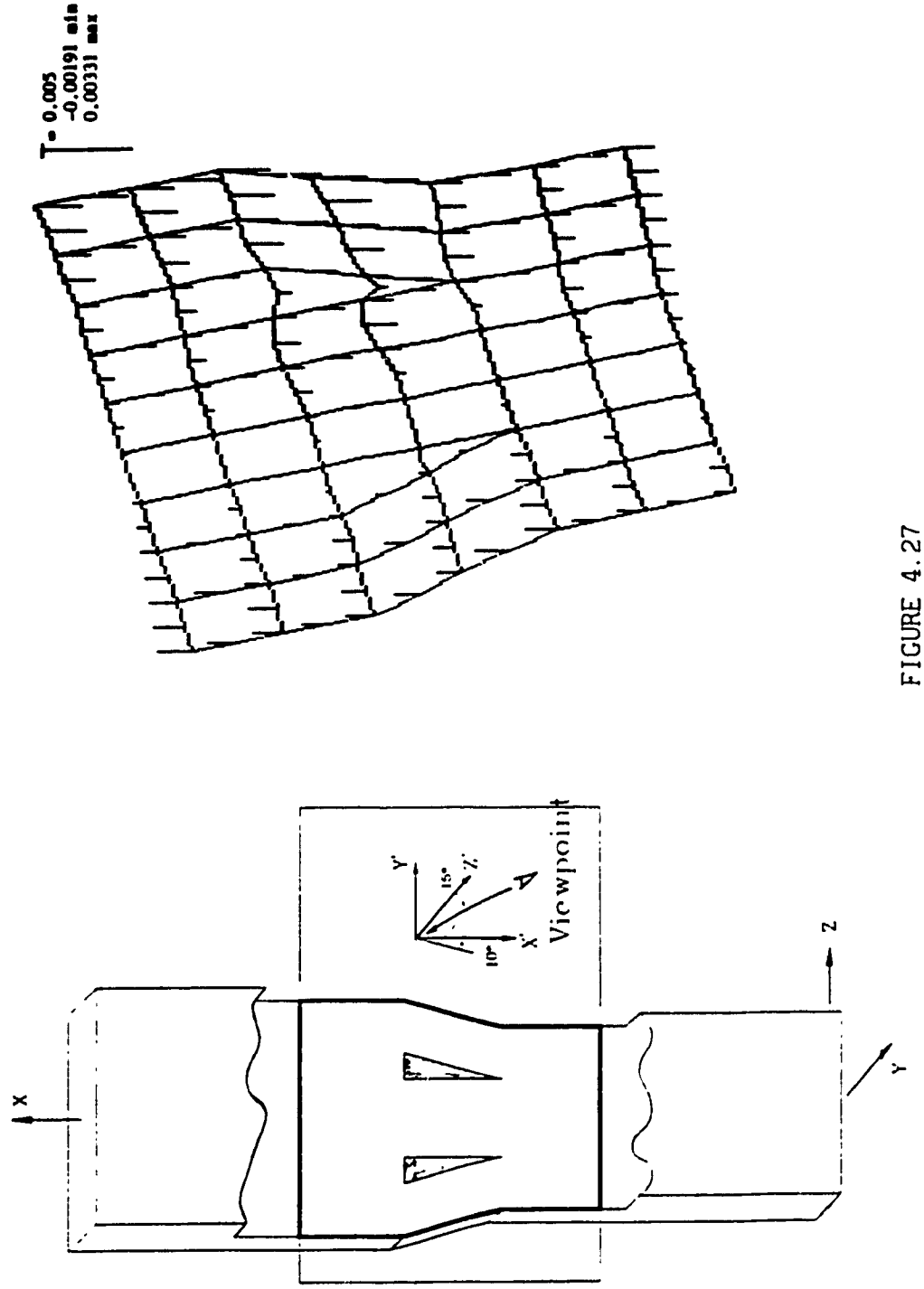


FIGURE 4.27

$\sigma_{x'y'}$, Tension Load, $[0_2/0_{2.5}]_S$, Y=0

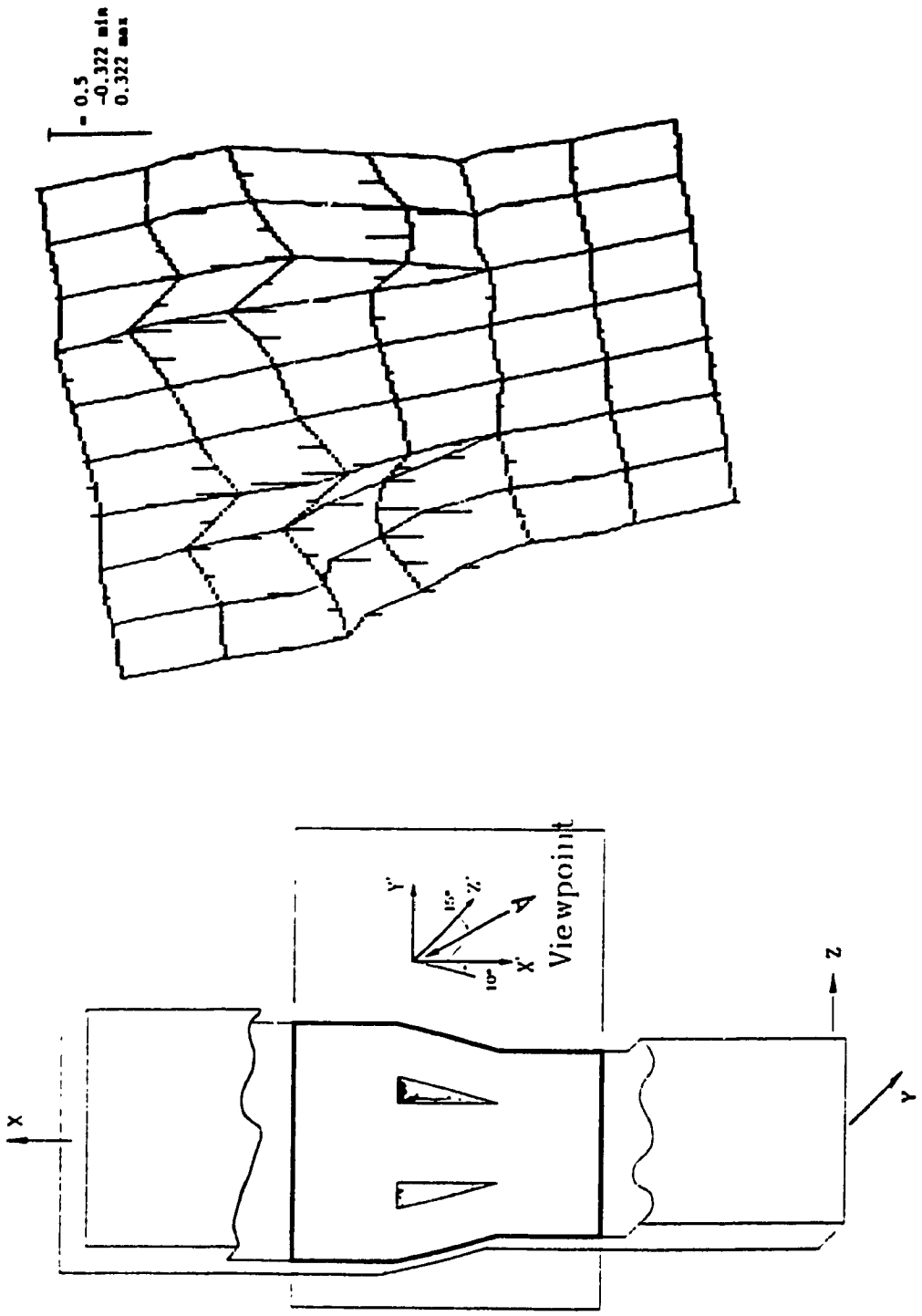


FIGURE 4.28

$\sigma^{*}_{x',z'}$, Tension Load, $[0_2/0_{2.5}]_S$, Y=0

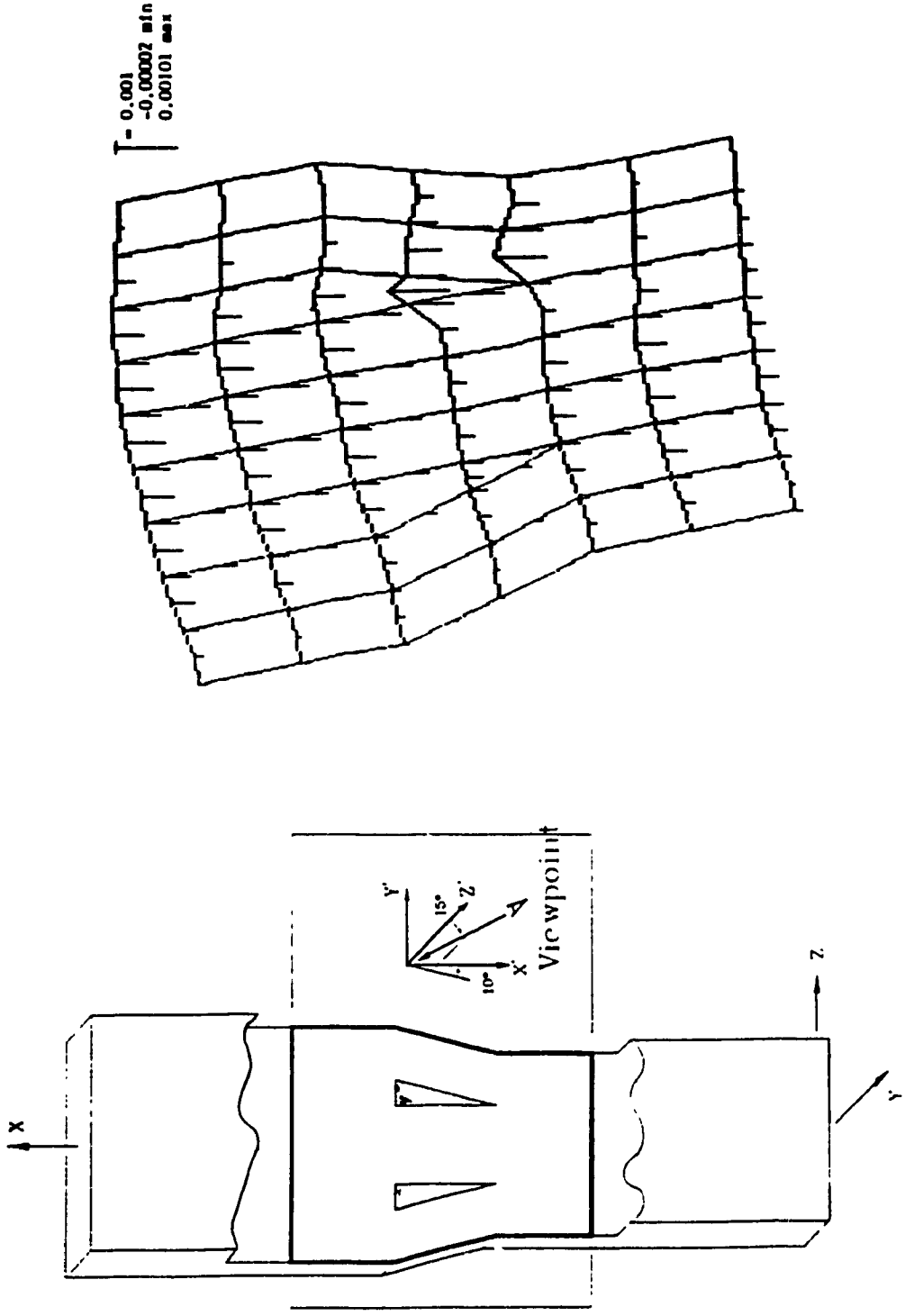


FIGURE 4.29

$\sigma_{y'z'}$, Tension Load, $[0_2/0_{2.5}]_S$, Y=0

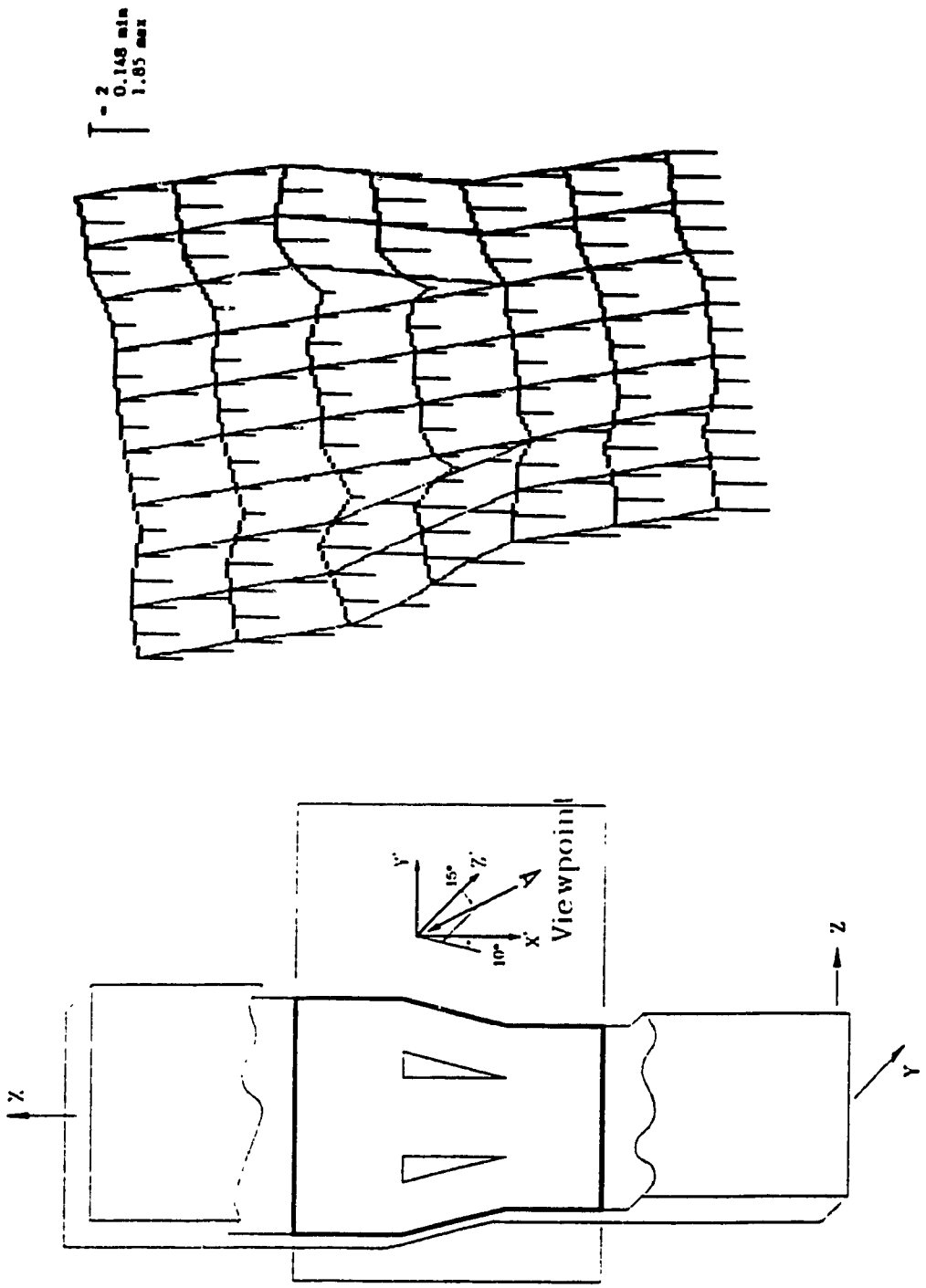


FIGURE 4.30

$\sigma^*_{x'}$, Tension Load, $[0_2/45_{2.5}]_S$, Y=0

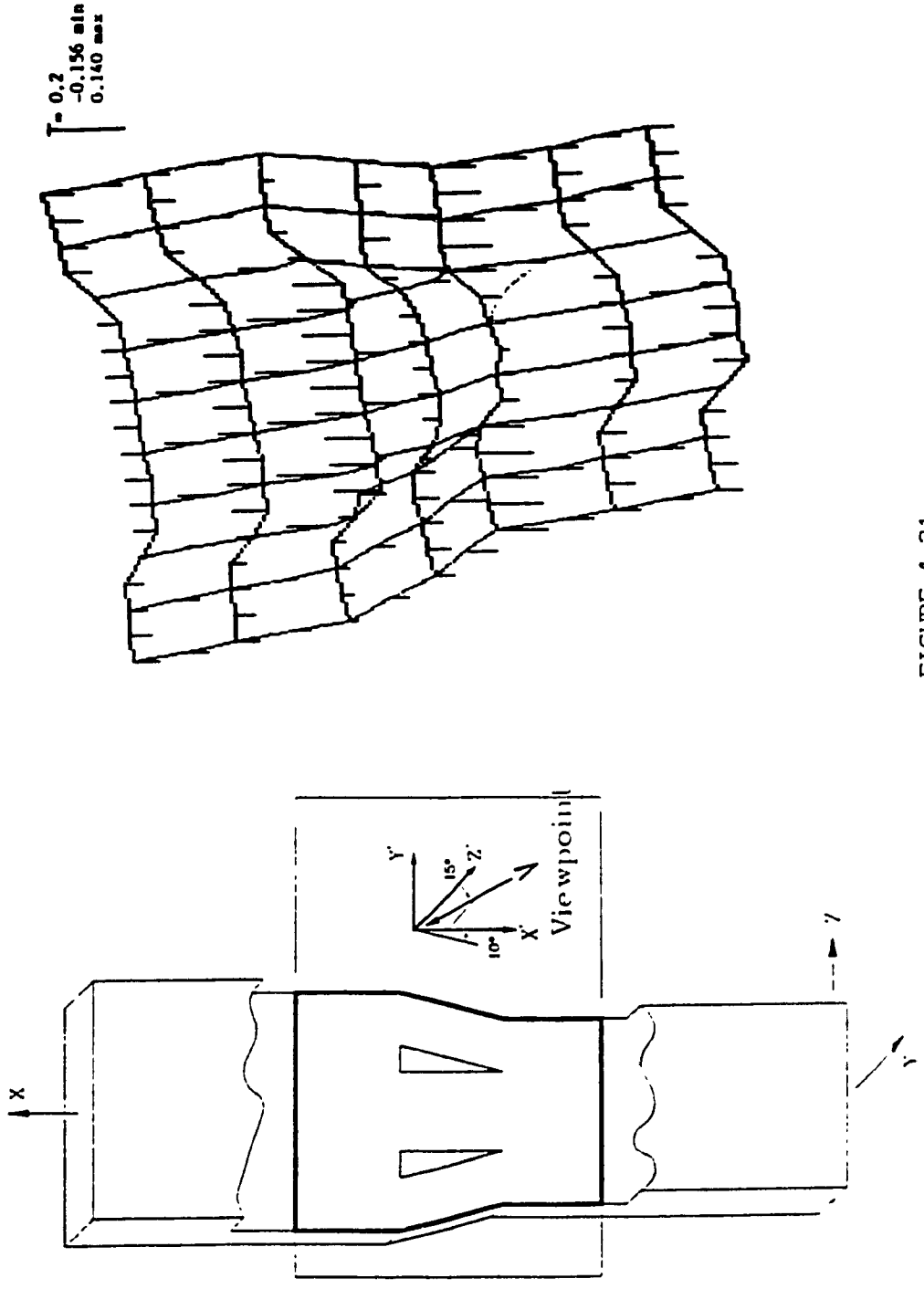


FIGURE 4.31

$\sigma_{y'}$, , Tension Load, $[0_2/45_{2.5}]_S$, Y=0

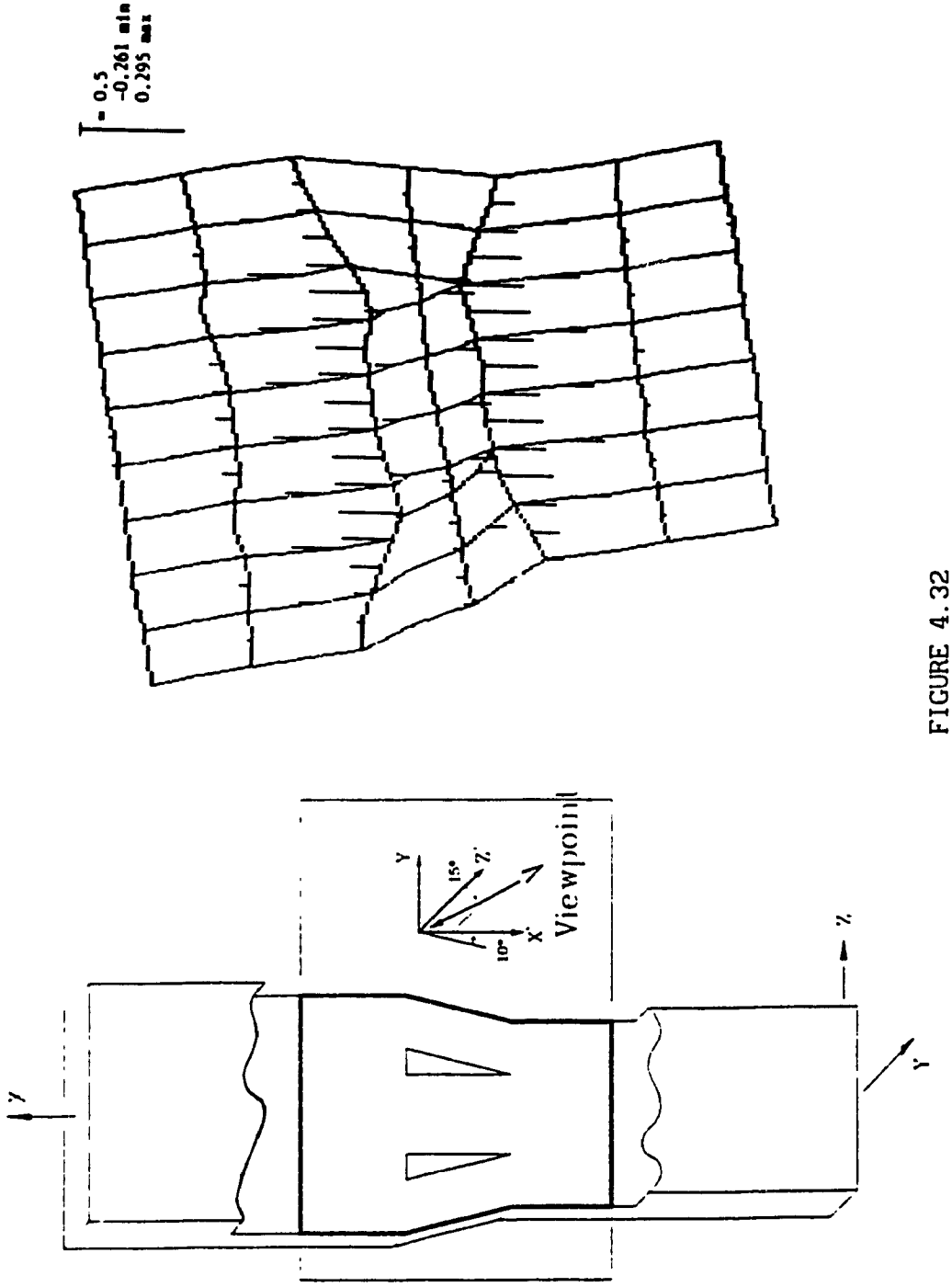


FIGURE 4.32

σ_z^*, , Tension Load, $[0_2/45_{2.5}]_S$, Y=0

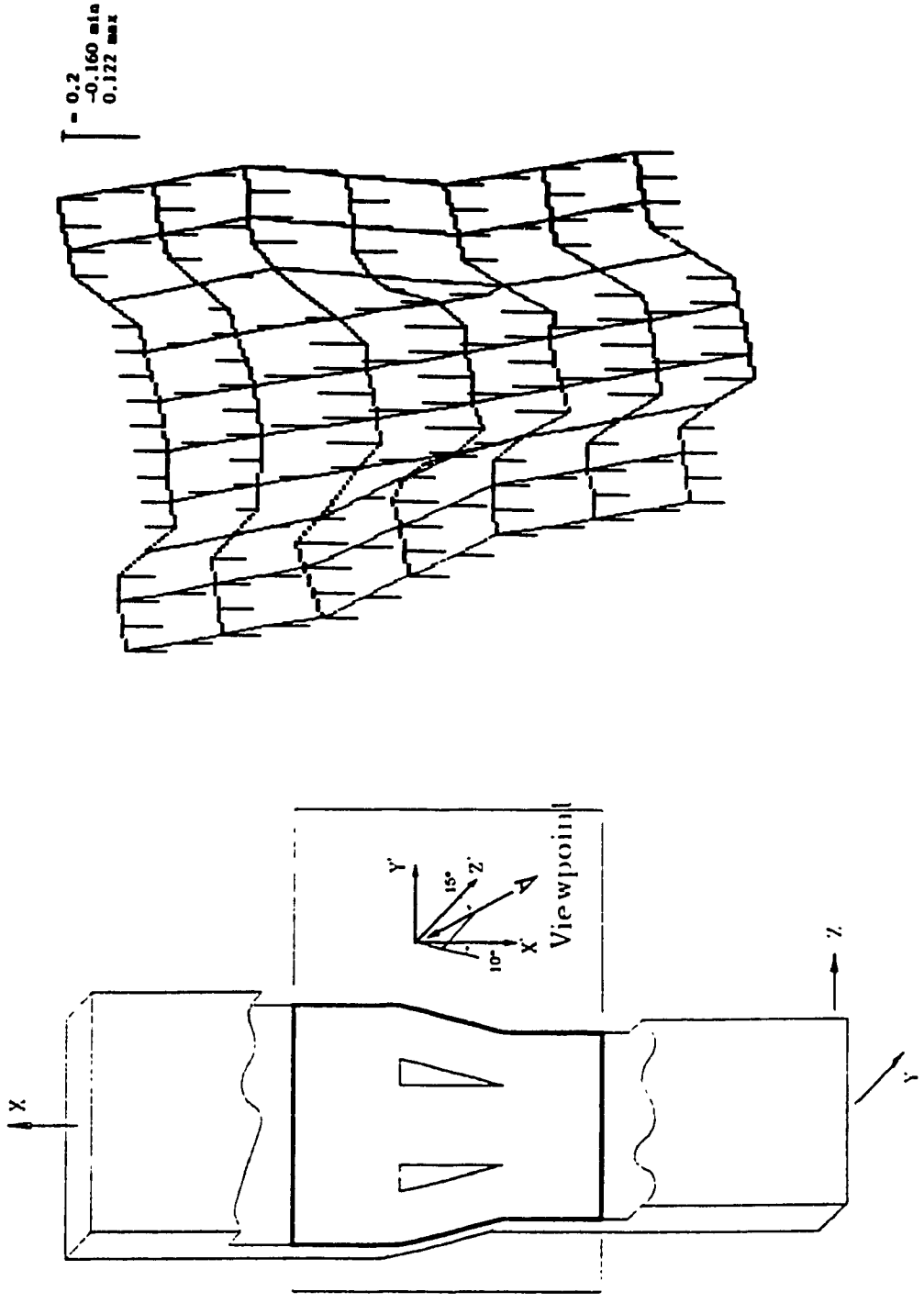


FIGURE 4.33

$\sigma_{x'y'}$, Tension Load, $[0_2/45_{2.5}]_S$, Y=0

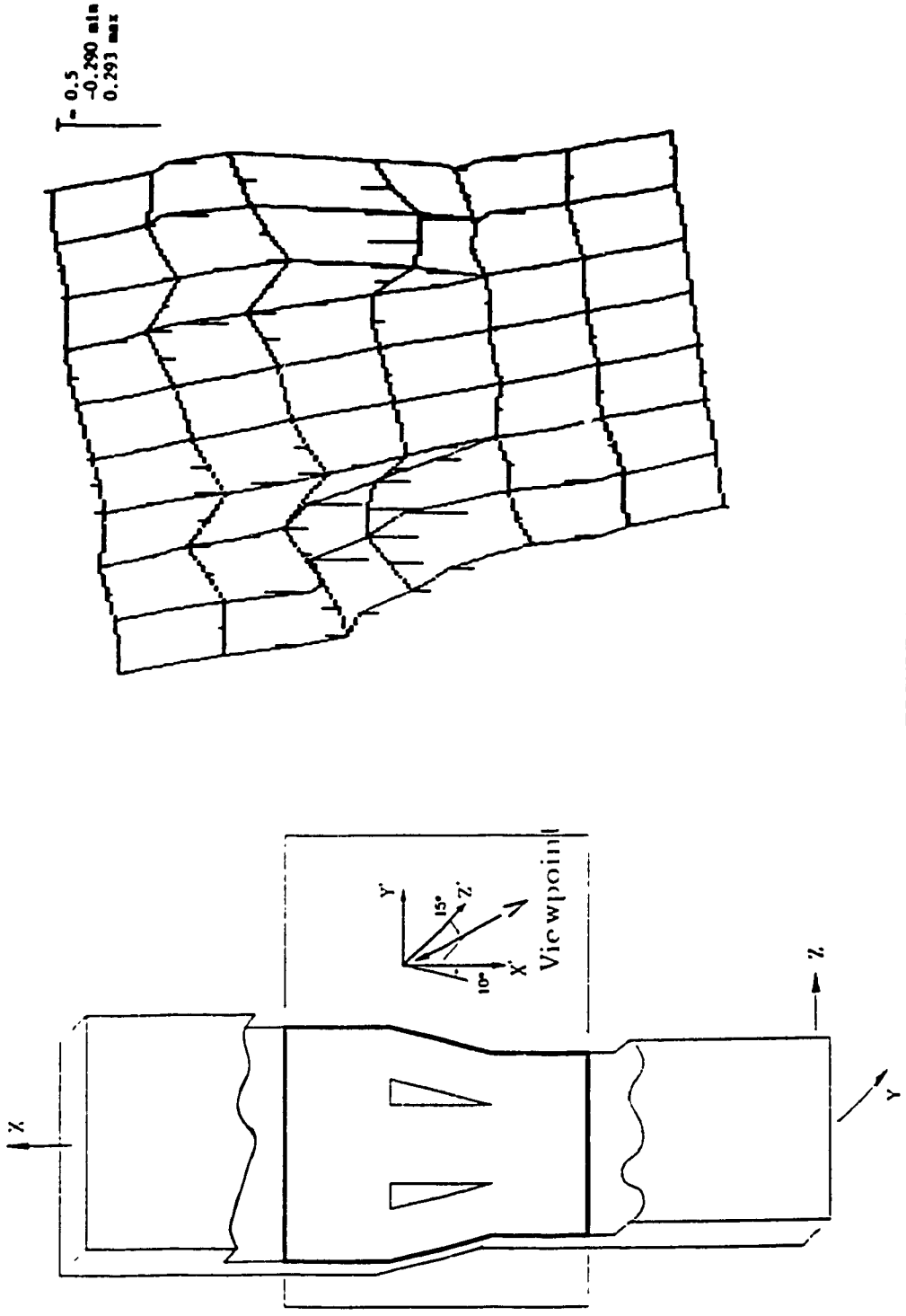


FIGURE 4.34

$\sigma^*_{x',z'}$, Tension Load, $[0_2/45_{2.5}]_S$, Y=0

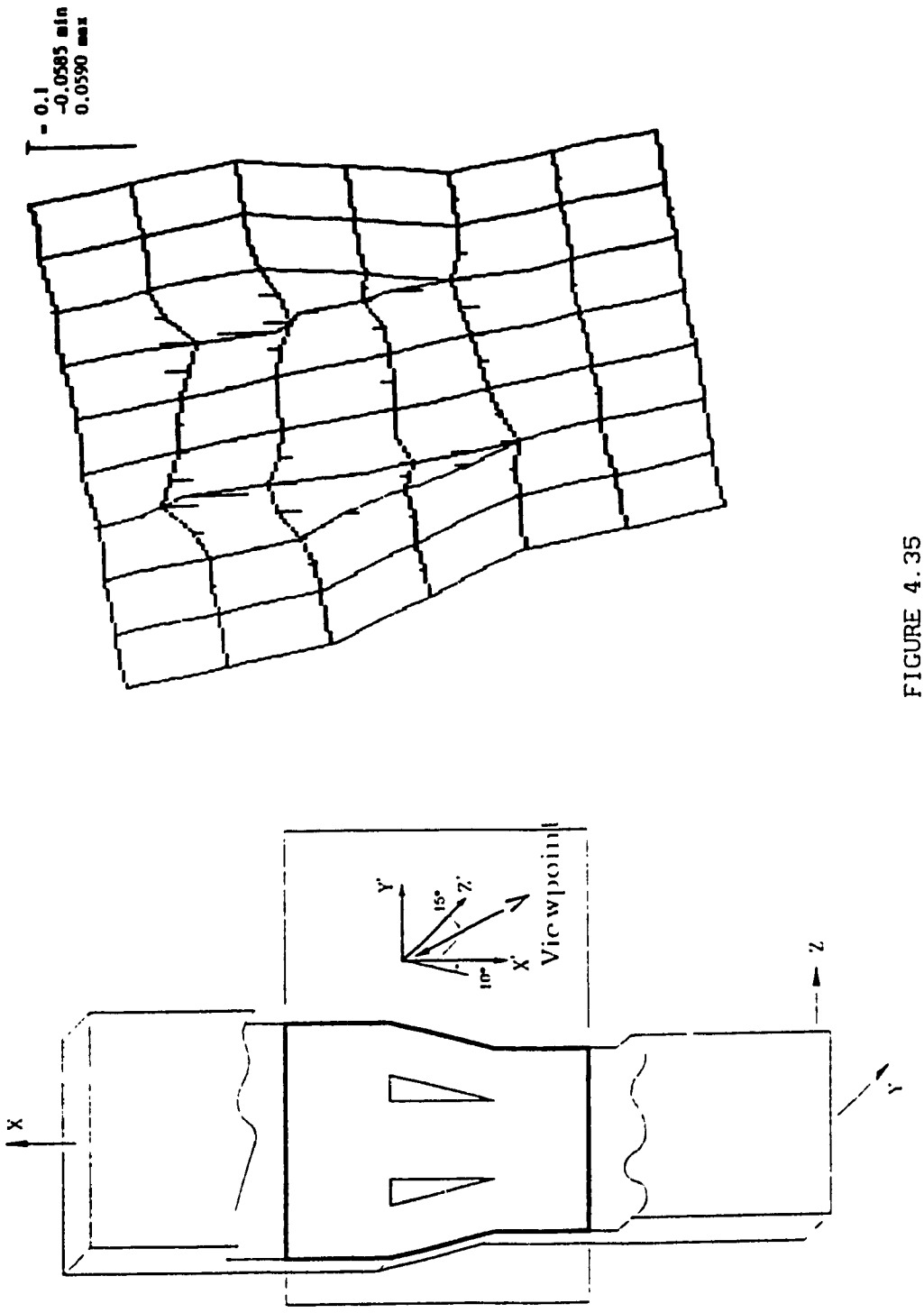


FIGURE 4.35

$\sigma^*_{y',z'}$, , Tension Load, $[0_2/45_{2.5}]_S$, Y=0

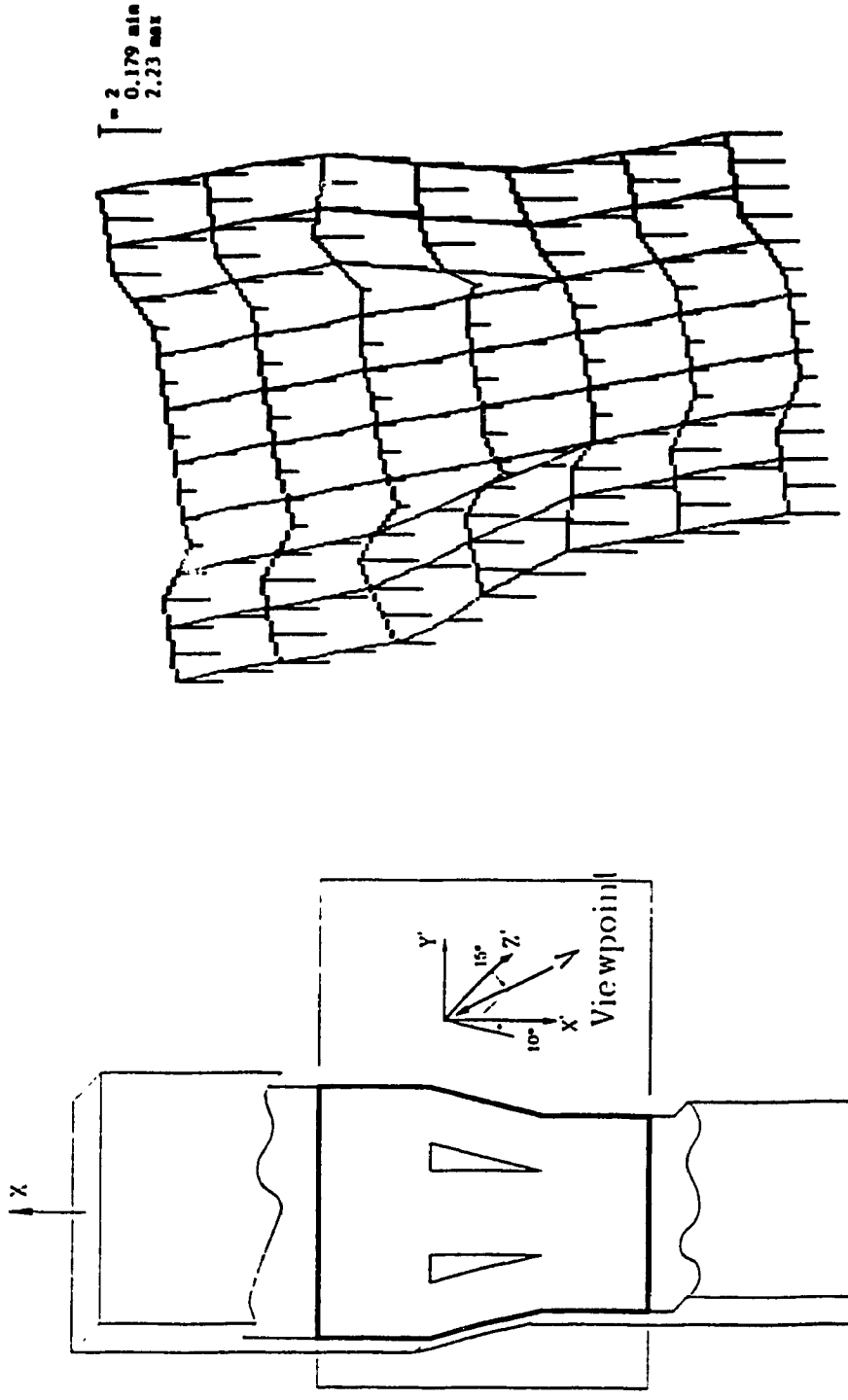
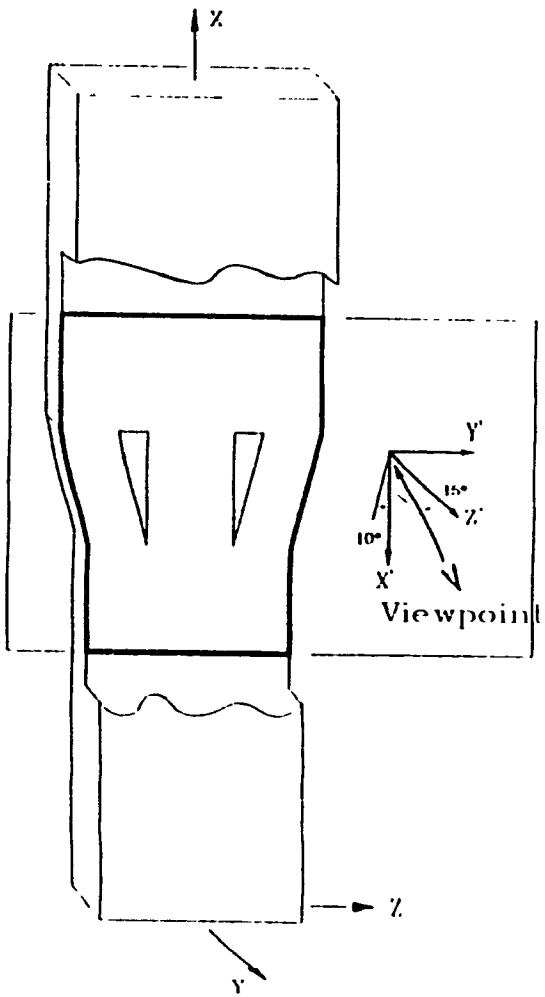


FIGURE 4.36

$\sigma^*_{x'}$, , Tension Load, $[0_2/90_{2.5}]_S$, Y=0



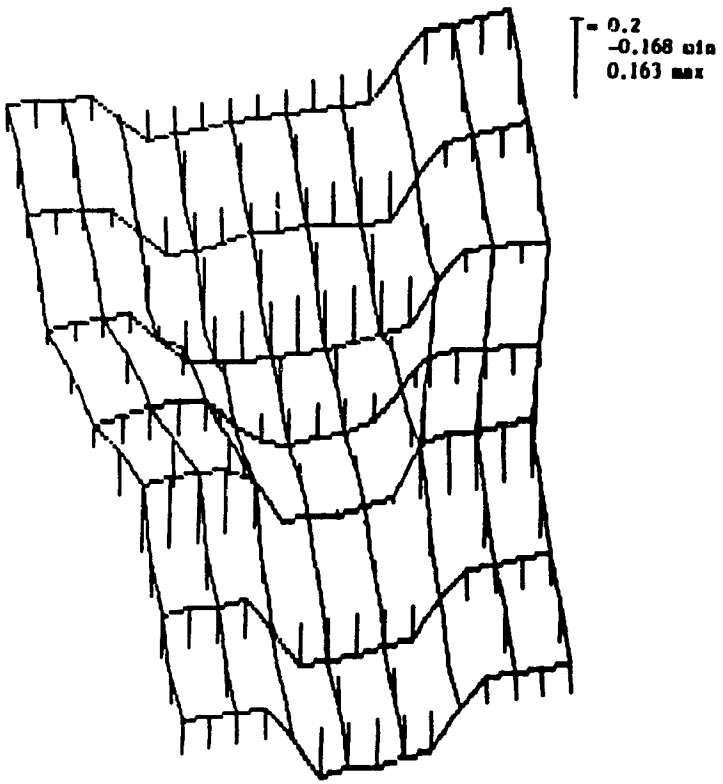


FIGURE 4.37

σ_y^*, , Tension Load, $[0_2/90_{2.5}]_S$, Y=0

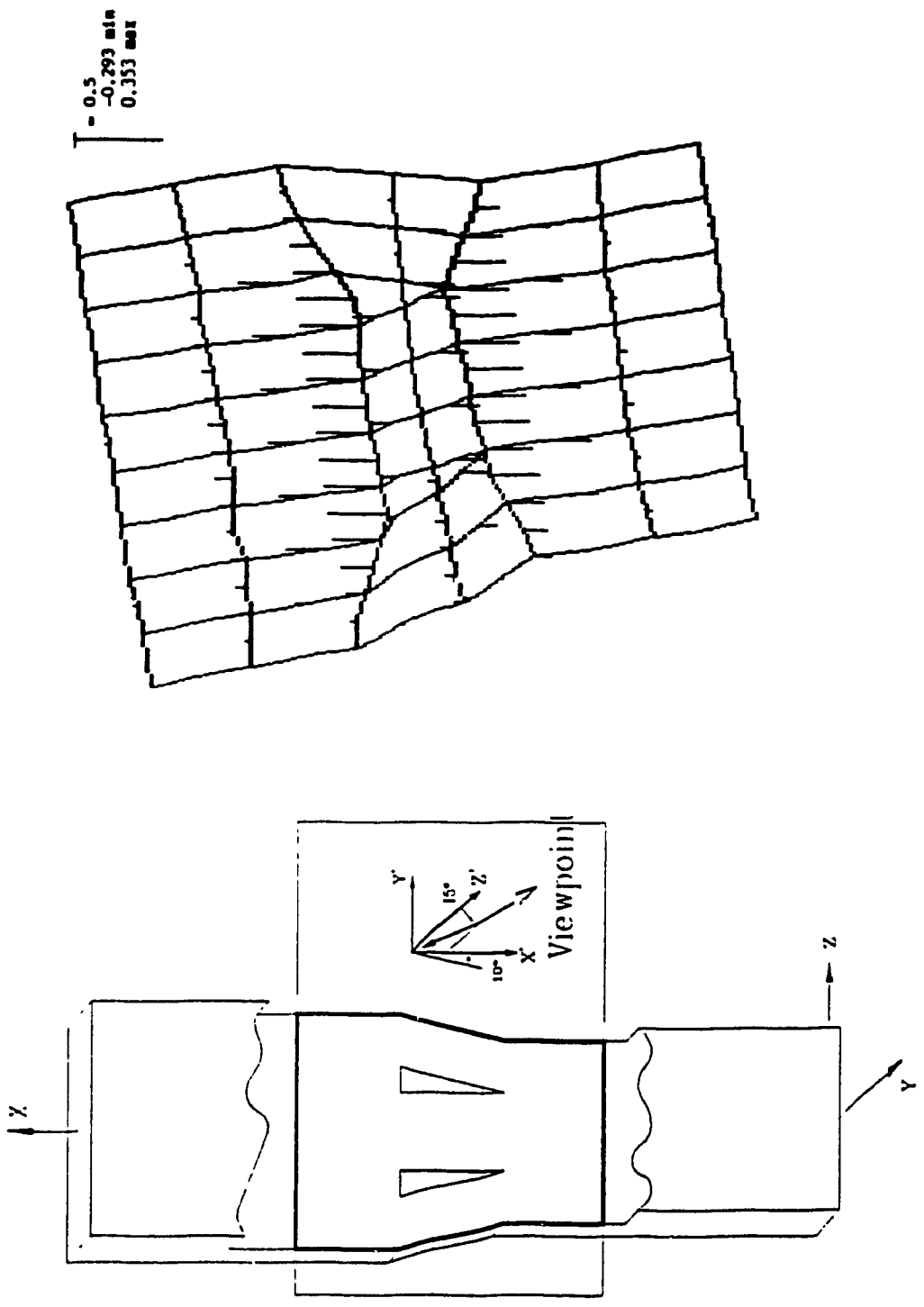


FIGURE 4.38

σ_z^*, , Tension Load, $[0_2/90_{2.5}]_S$, Y=0

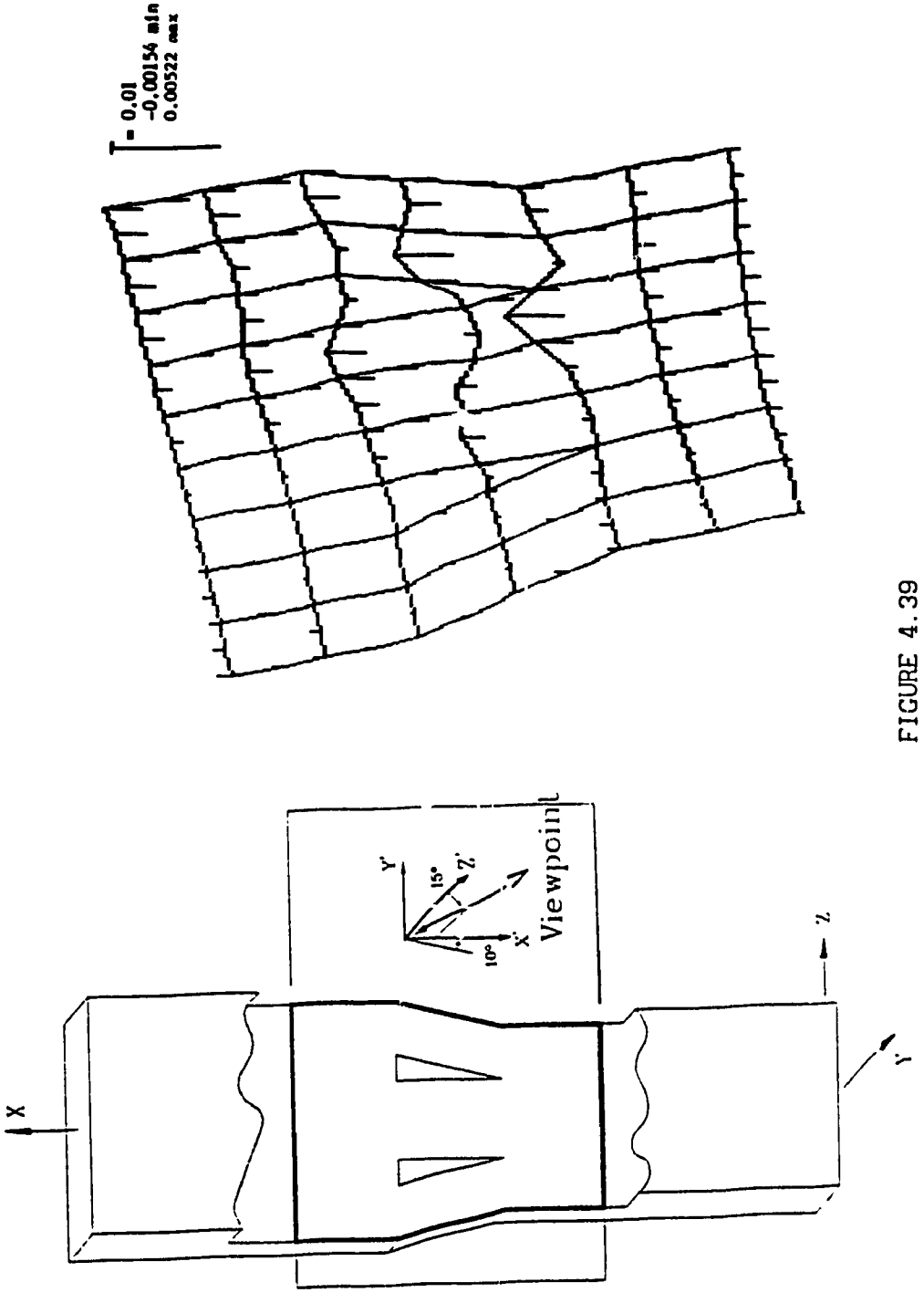


FIGURE 4.39

$\sigma_{x'y'}$, Tension Load, $[0_2/90_{2.5}]_S$, Y=0

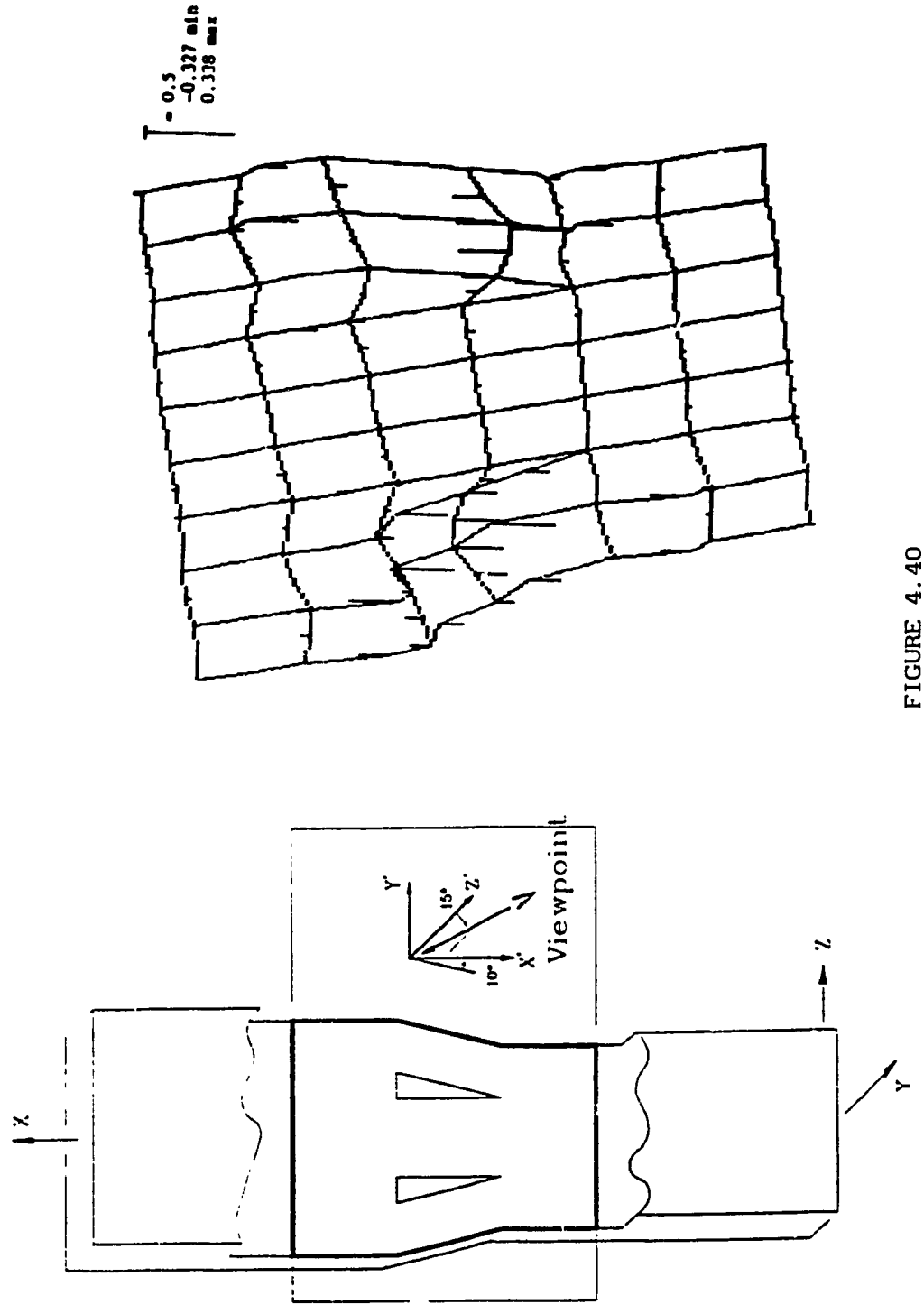


FIGURE 4.40

$\sigma^*_{x'z'}$, Tension Load, $[0_2/90_{2.5}]_S$, Y=0

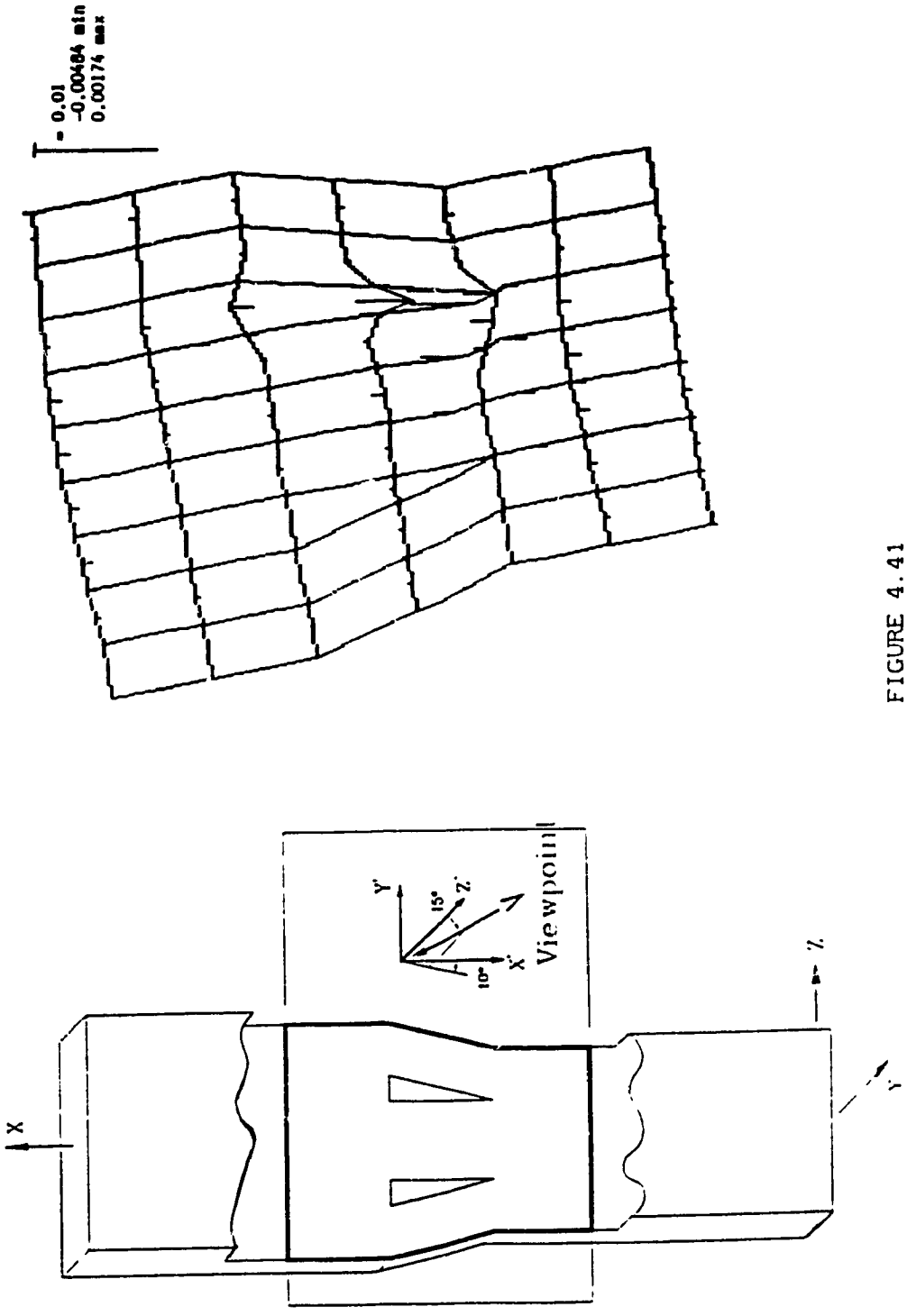
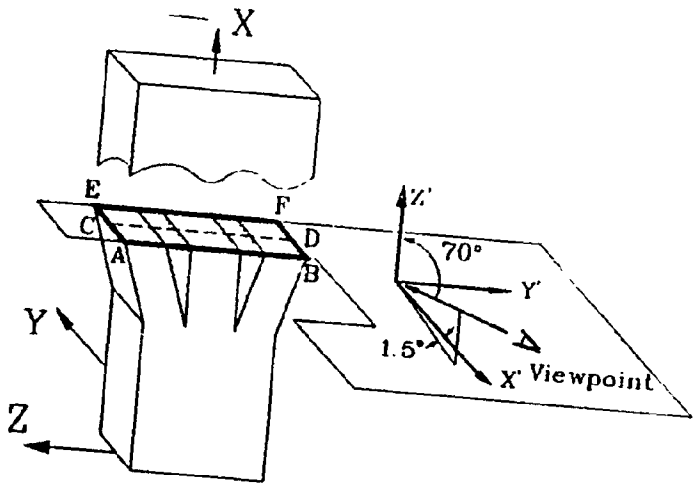


FIGURE 4.41

$\sigma_{y'z'}$, Tension Load, $[0_2/90_{2.5}]_S$, Y=0



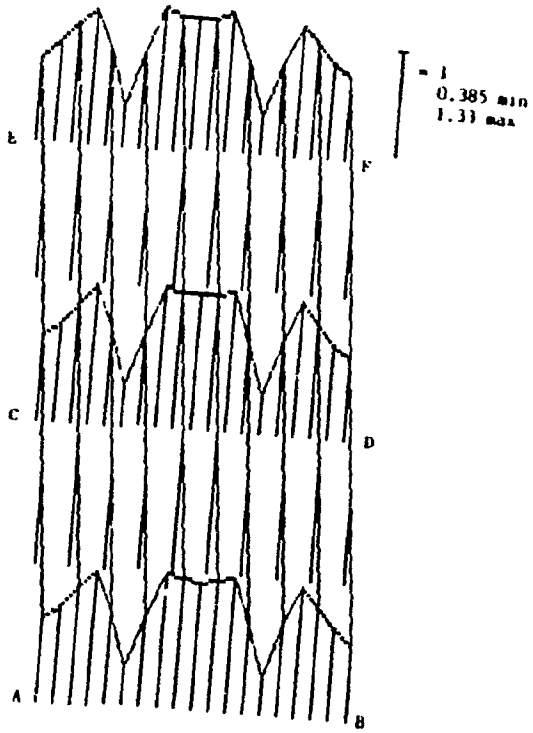
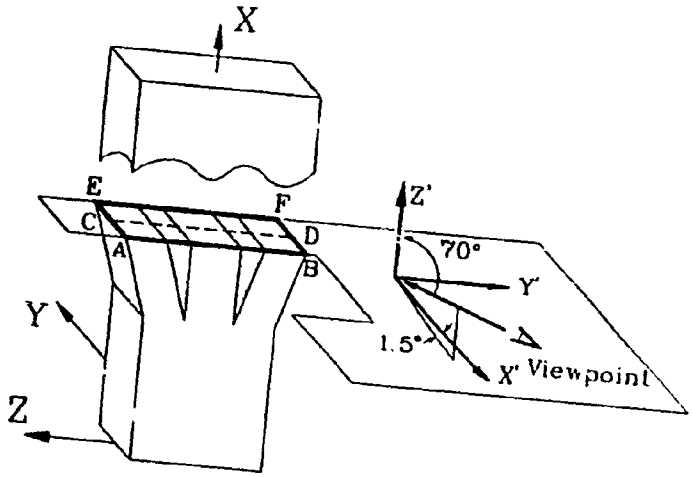


FIGURE 4.42

$\sigma^{*}_{x'}$, Tension Load, $[0_2/0_{2.5}]_S$, X=Drop-off



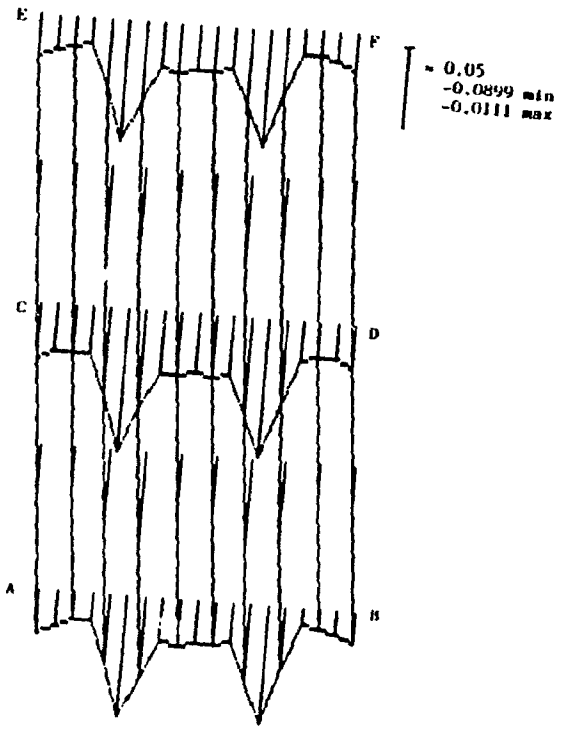
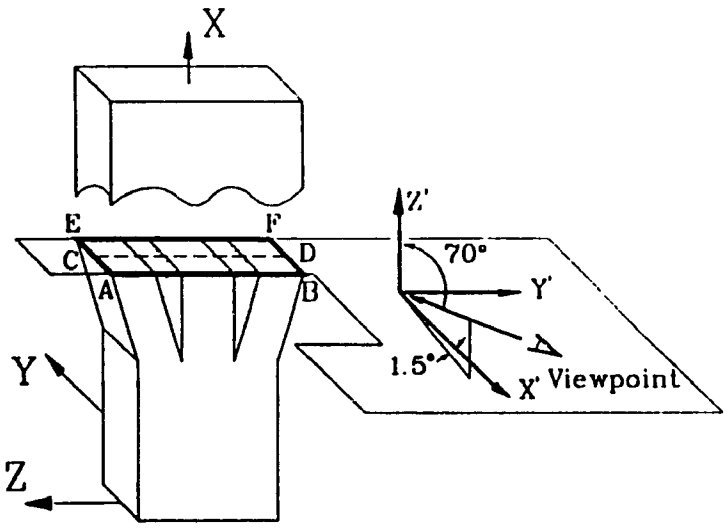


FIGURE 4.43

$\sigma_{y'}$, Tension Load, $[0_2/0_{2.5}]_S$, X=Drop-off



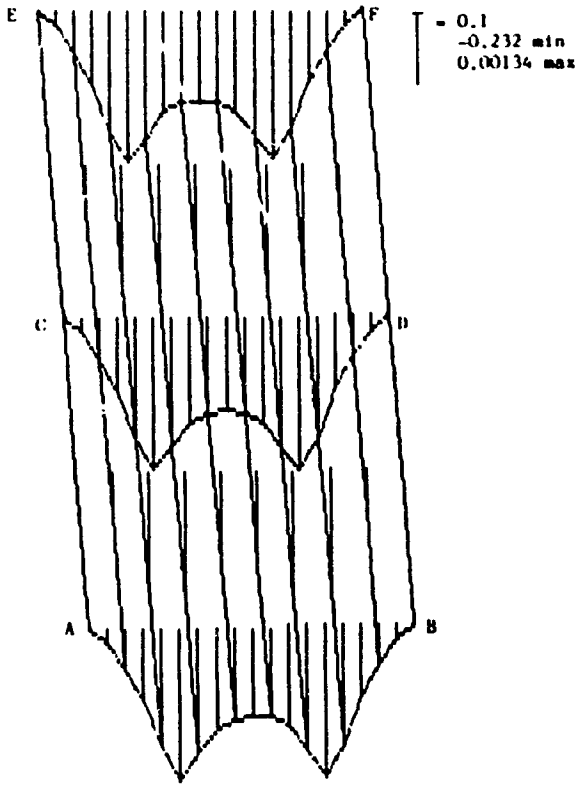
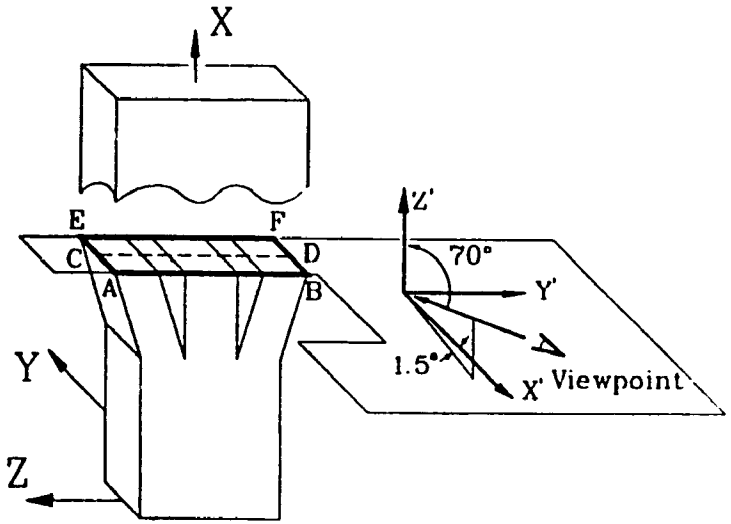


FIGURE 4.44

σ_z^*, , Tension Load, $[0_2/0_{2.5}]_S$, X=Drop-off



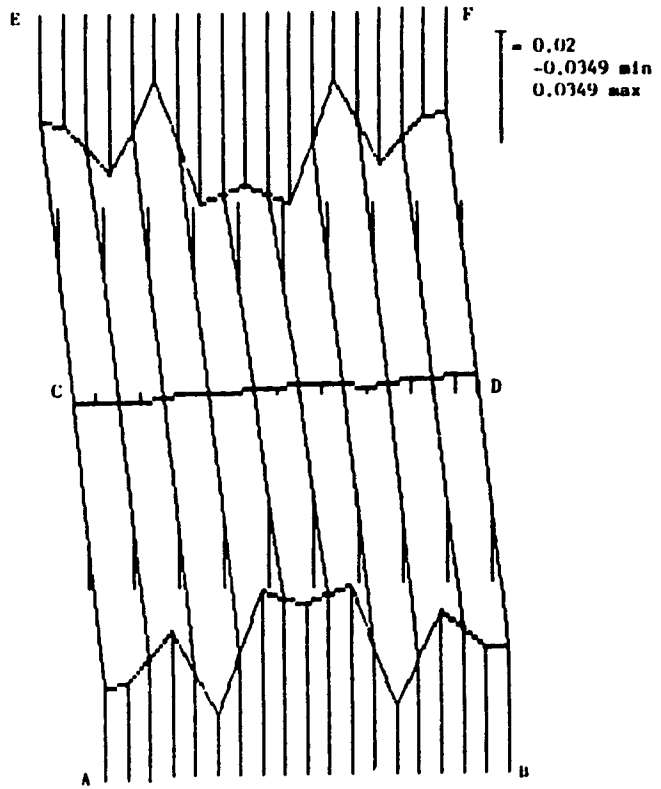
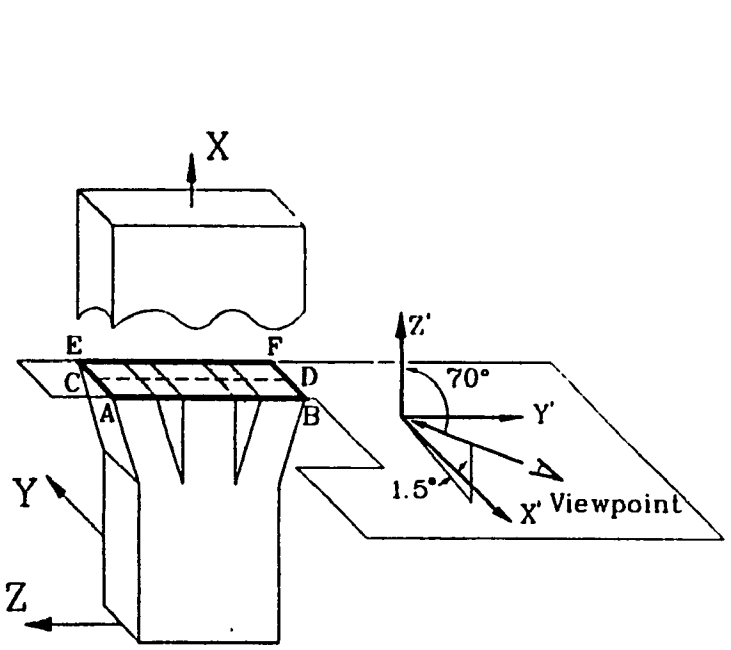


FIGURE 4.45

$\sigma^{*}_{x'y'}$, Tension Load, $[0_2/0_{2.5}]_S$, X=Drop-off



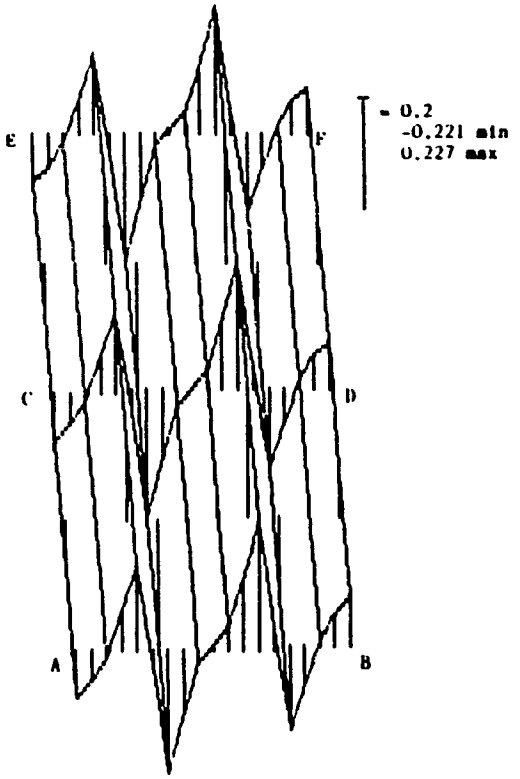
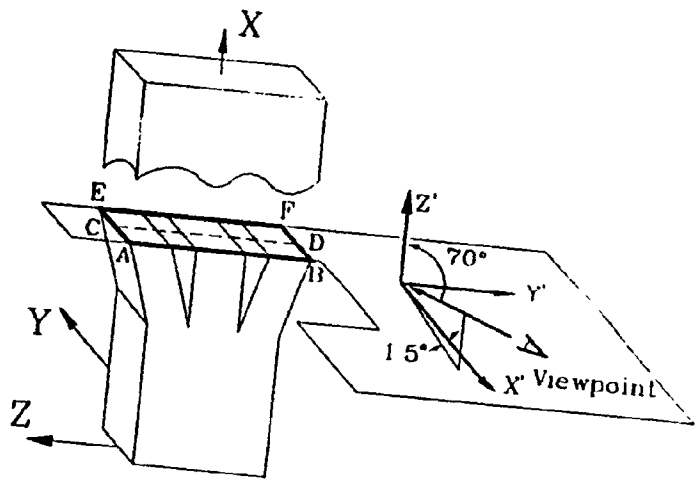


FIGURE 4.46

$\sigma^{*}_{x',z'}$, Tension Load, $[0_2/0_{2.5}]_S$, X=Drop-off



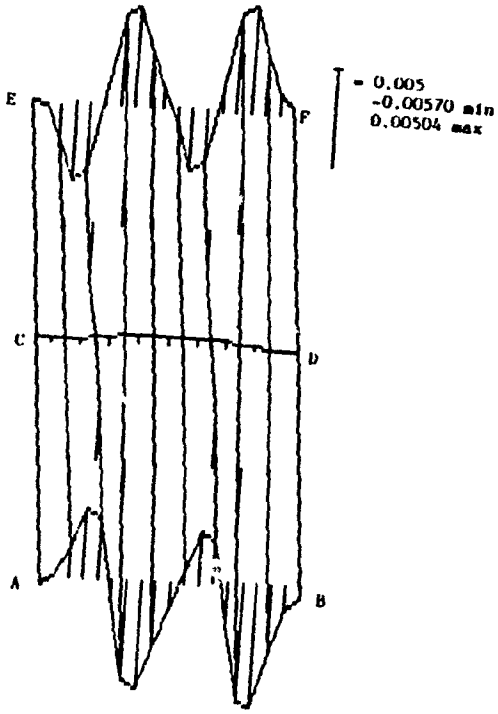


FIGURE 4.47

$\sigma^{*}_{y'z'}$, Tension Load, $[0_2/0_{2.5}]_S$, X=Drop-off

The maximum stress disturbance distance was found by observation to be approximatively equal to the length of the drop-off wedge in the laminate axis (Figures 4.25, 4.26, 4.28, 4.32, 4.34, 4.38, 4.40).

4.5 Summary

For the two dimensional case, general agreement was found between the closed form solution and the displacement formulation finite element method. This agreement gives confidence on the use of the finite element method for the study of interlaminar stresses around triangular holes in anisotropic material in three dimensional structures.

Comparing with experimental results, a workable mesh size has been obtained for tension and bending loading conditions in three dimensional cases. Best agreement was found for models with the drop-off filled with resin.

The stress disturbance caused by the drop-off is confined to approximatively 1 drop-off length of the drop-off in the laminate axis.

CHAPTER 5

FACTORS INFLUENCING INTERLAMINAR STRESSES

5.1 Introduction

Interlaminar stresses are responsible for delamination. A first step towards being able to control the level of interlaminar stresses is finding the parameters which have an effect and studying their influence. In order to study improvement obtained from a parameter variation, tapered laminates should have a common basis for comparison. There can be two different ways of comparing laminates. These are based on strength or efficiency. When results are compared in terms of laminate strength they strongly depend upon the laminate dimensions. Using efficiency as a means of comparison reduces the effect of dimensions and gives a handle upon the quality of tapered laminate designs.

In this chapter, tapered laminate efficiency is defined. It is followed by an analysis of parameters which can affect the level of interlaminar stresses. These parameters are:

- Drop-off location
- Drop-off wetness
- Drop-off shape
- Laminate width
- Fiber angle at drop-off

Failure mode and location under different loading conditions are investigated. Finally, crack growth is simulated by consecutive removal of failed elements. This simulation has not yet proven itself, and will be the subject of discussion.

5.2 Efficiency Definition

The efficiency of tapered laminates is defined here in a similar manner as a bolted connection. In bolted connections, efficiency is the ratio of the strength of the bolted sample by the strength of a continuous material. Efficiency is defined as

$$\text{Efficiency} = \frac{L_w}{L_{wo}}$$

where

L_w is the maximum load applied to a laminate with a drop-off

L_{wo} is the maximum load applied to a laminate without a drop-off. This is known as the reference laminate.

For tapered laminates, the reference laminate (not tapered) has the same layup as the smallest cross-section of the tapered laminate. Figures 5.1 and 5.2 respectively show the reference laminate and a tapered laminate.

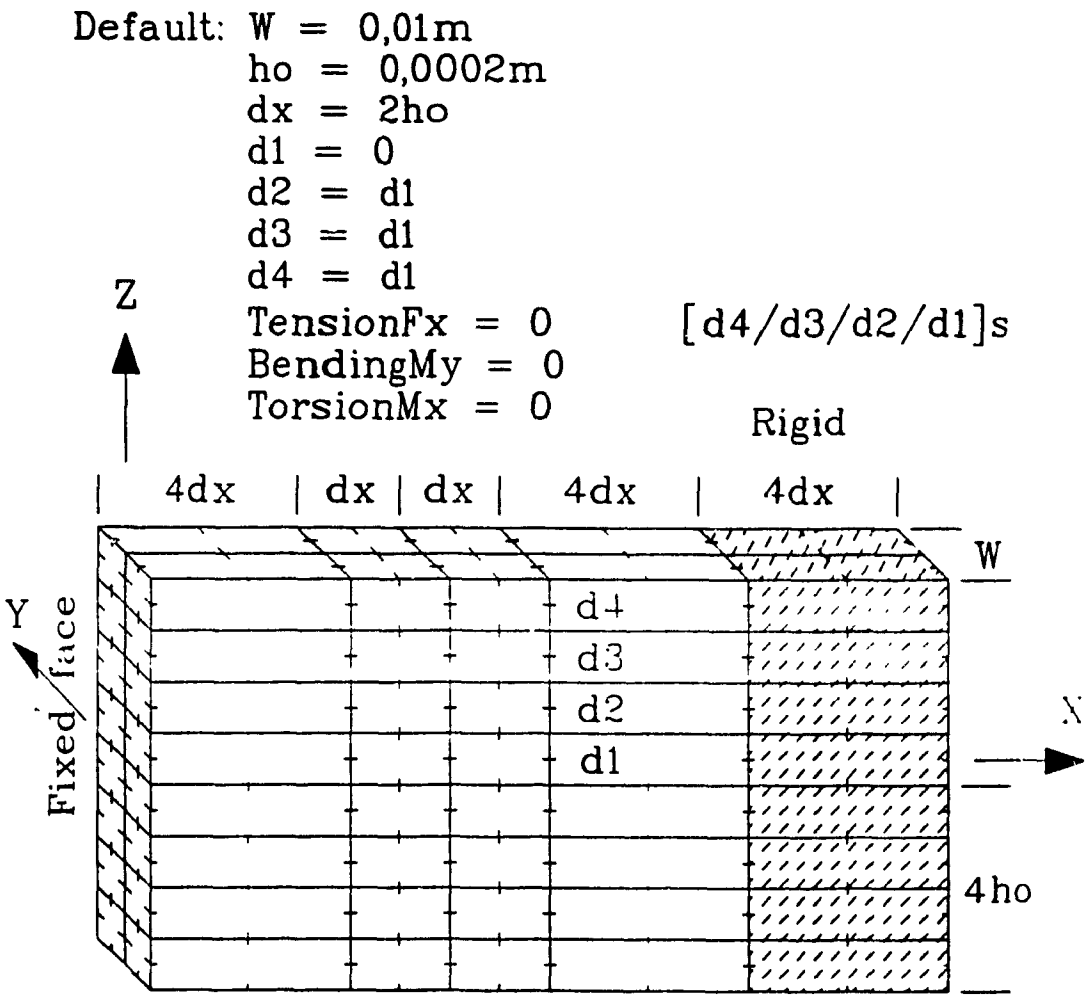


FIGURE 5.1

Reference Mesh

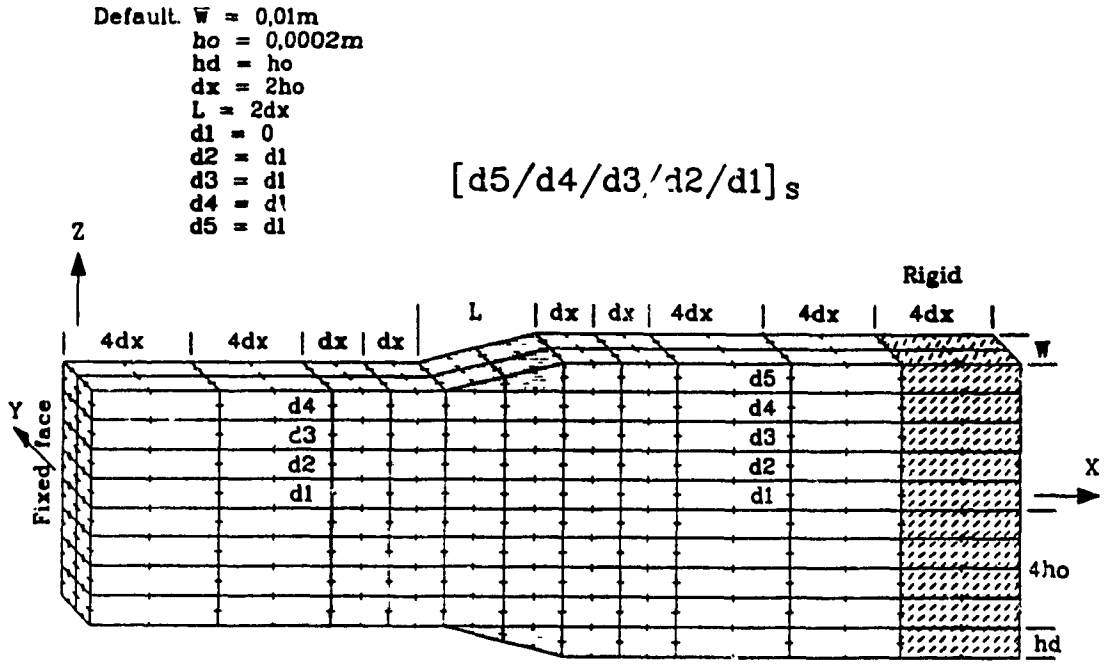


FIGURE 5.2

External Mesh

When using the finite element method to compare the maximum load of the reference laminate and that of the tapered laminate, similar mesh sizes should be used to insure the same stress level in both models.

5.3 Drop-off Location

In common practice, both internal and external drop-offs are used. It is important to know how the location of the drop-off in the thickness direction of a tapered laminate effect its efficiency. In this part, drop-offs of different locations will be considered.

5.3.1 Meshes and Loadings

The external drop-off mesh is shown in Figure 5.2, while an internal drop-off mesh with two layers above the drop-off is given in Figure 5.3 (this is known as internal2). Although not portrayed, two other internal drop-off locations were considered: with one (internal1) and three (internal3) layers above the drop-off. In all cases, the reference mesh is shown in Figure 5.1.

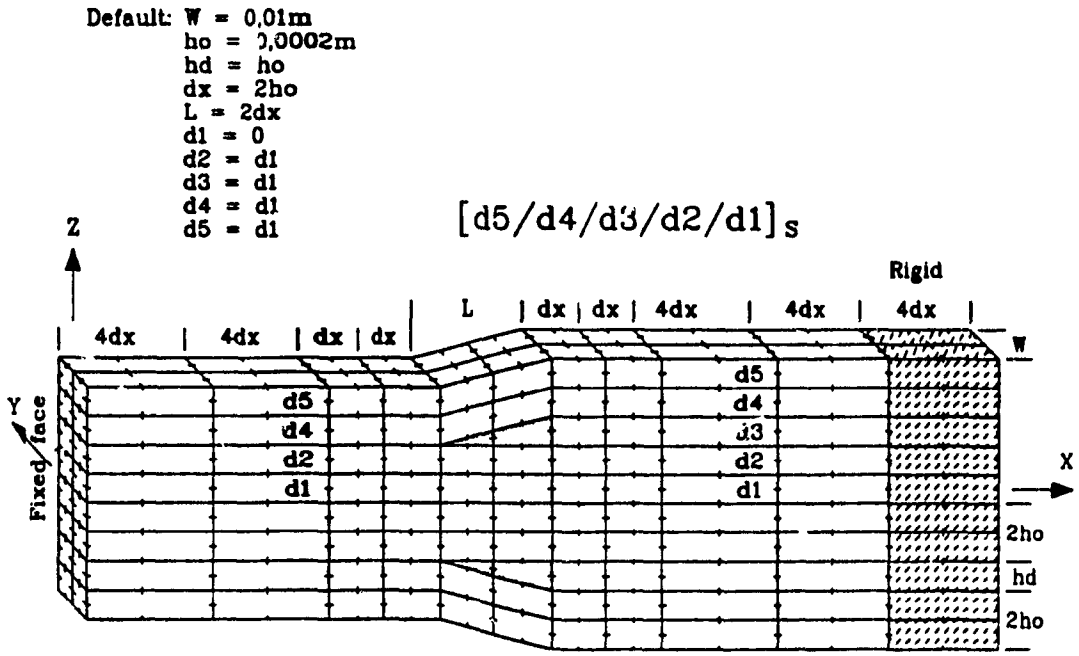


FIGURE 5.3

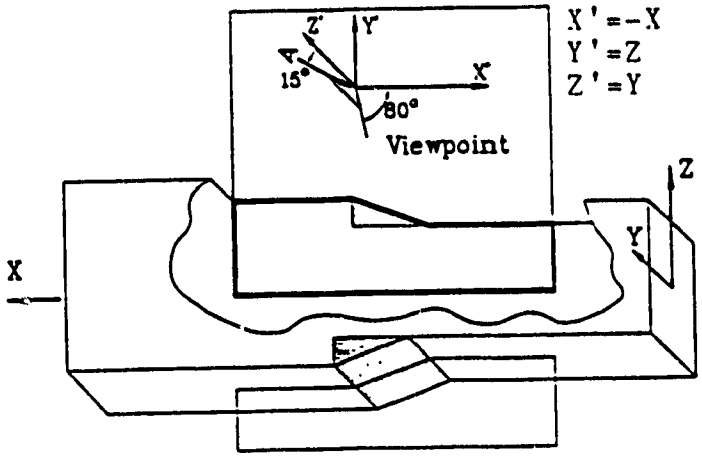
Internal 2 Mesh

Tension, bending and torsion were imposed to all the geometries. For tension, a force of 1000 N was applied to the center of the rigid wall of elements. A bending moment of 1000 Nm, in the direction of positive y, was applied by forces of opposite directions, centered at the upper and lower rigid wall limits. A positive x torsion moment of 1000 Nm was applied at the same nodes as the force for the bending moment.

In this case study, all fiber layers were oriented along the laminate axis (0° with respect to the elements). The same tension, bending and torsion moments were applied to the rigid wall of the reference case. All meshes considered had their drop-off filled with epoxy material elements.

5.3.2 Stress Distributions

Figures 5.4-5.9 and 5.10-5.15 show the stress distributions along the element directions $\sigma^{*}_{x'}$, $\sigma^{*}_{y'}$, $\sigma^{*}_{z'}$, $\sigma^{*}_{x'y'}$, $\sigma^{*}_{x'z'}$, $\sigma^{*}_{y'z'}$, for the external and internal2 models respectively. The tensile load applied produces an average σ_x stress of 50 MPa at the largest section (1×10^{3} N divided by an area of 1cm x 10 plies of 0.2 mm). Similar stress distribution shapes (but not stress level) are found in the region of the drop-off wedge for both external and internal drop-off configurations.



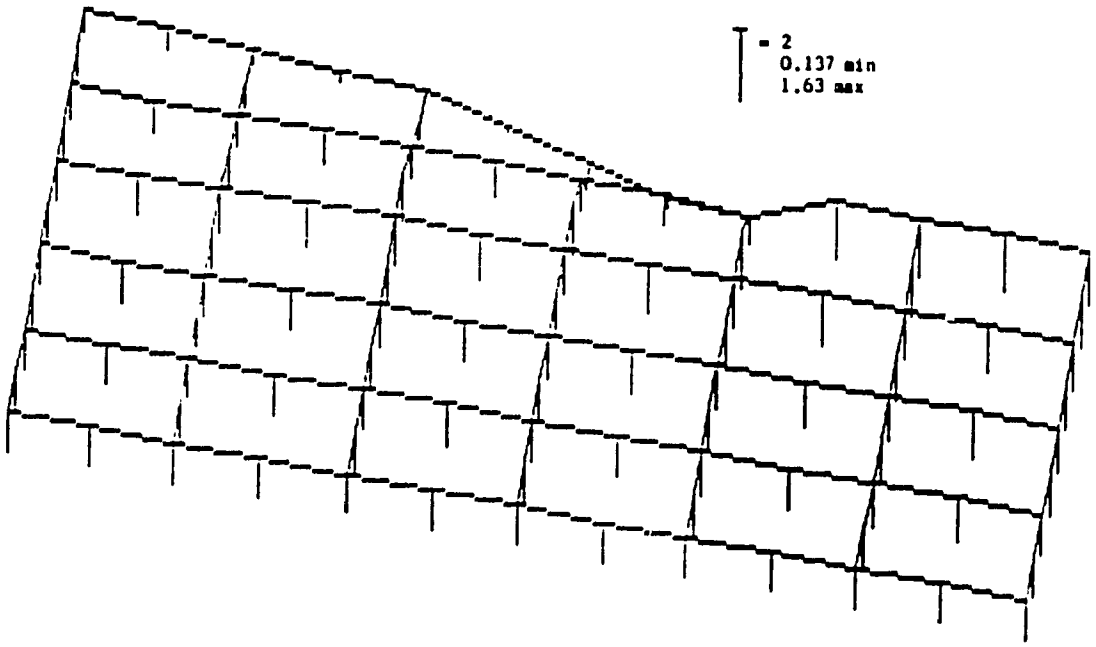


FIGURE 5.4

$\sigma^{*}_{x'}$, Tension Load, All 0°, External Drop-off, Y=0

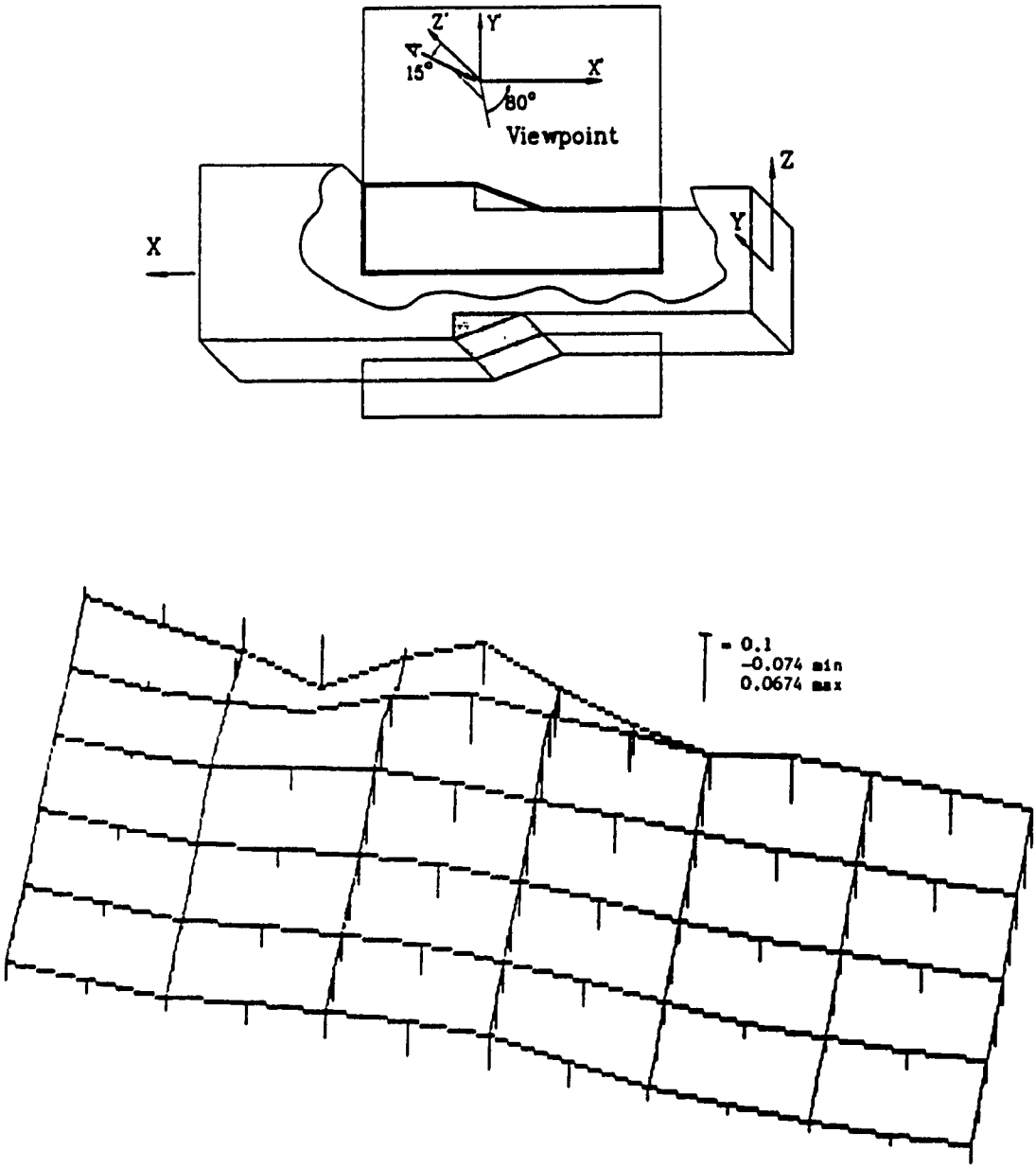
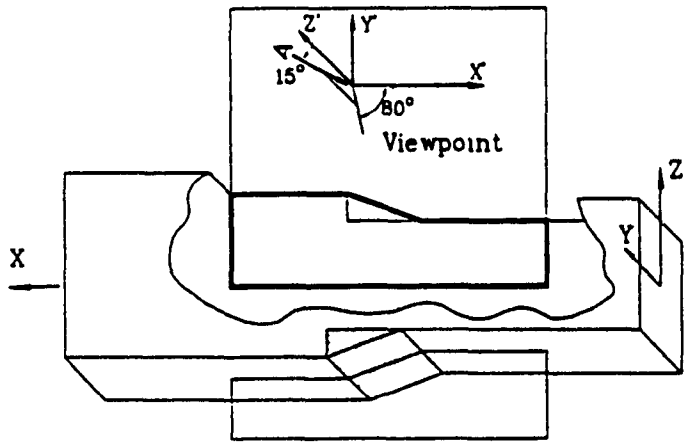


FIGURE 5.5

σ_y^*, , Tension Load, All 0°, External Drop-off, Y=0



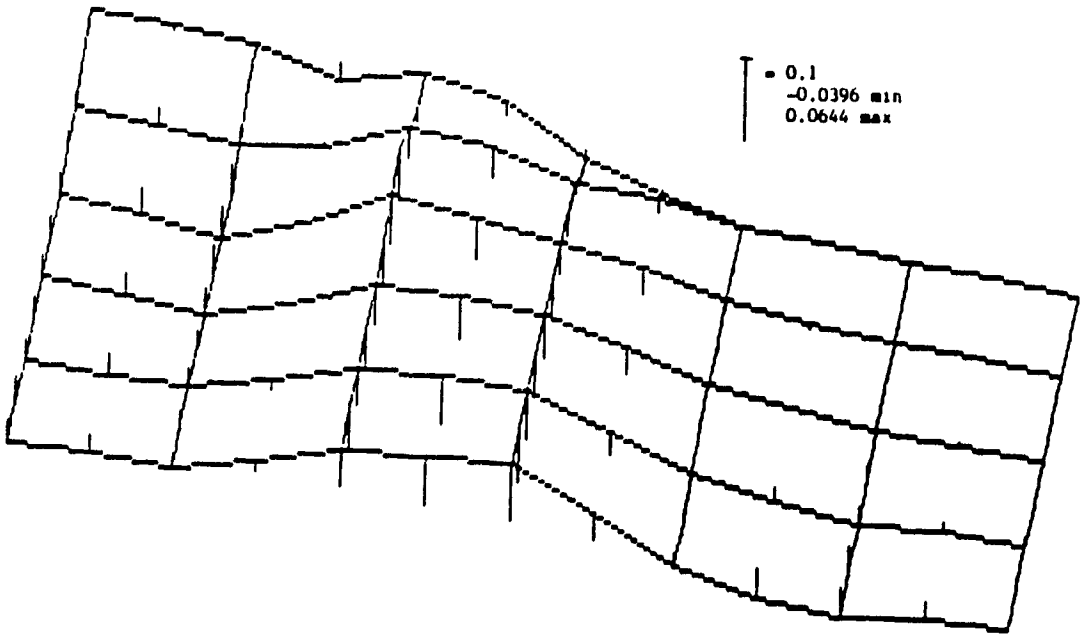
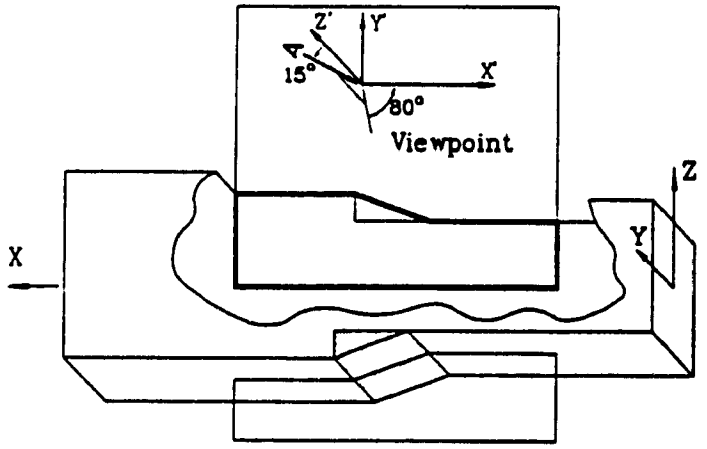


FIGURE 5.6

σ_z^*, , Tension Load, All 0°, External Drop-off, Y=0



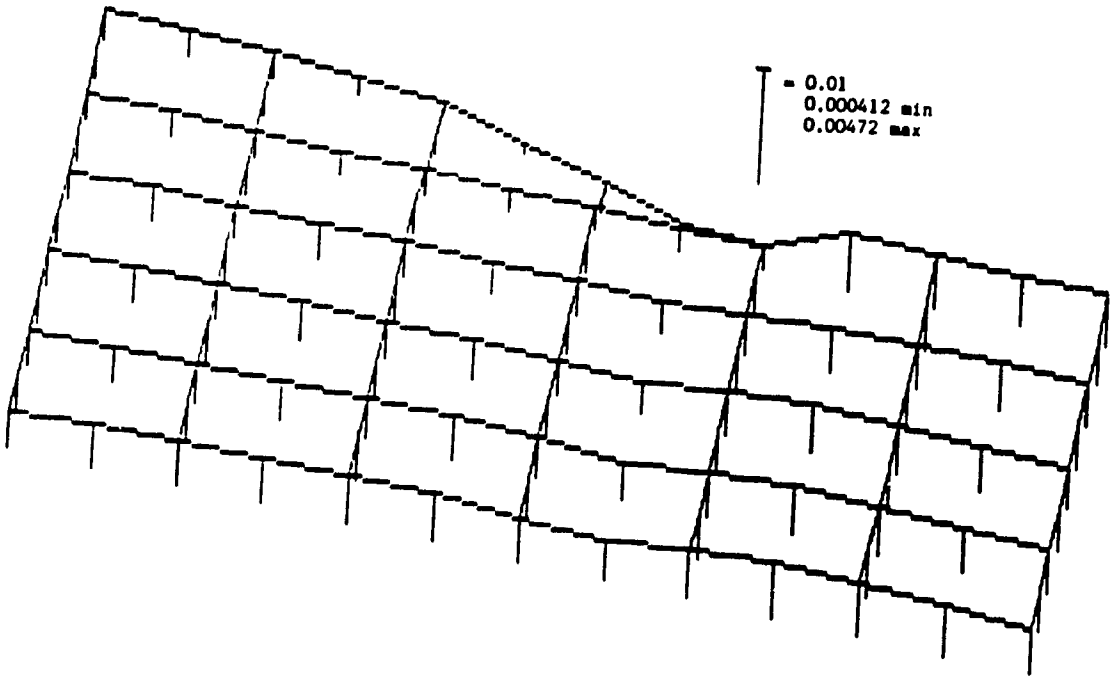
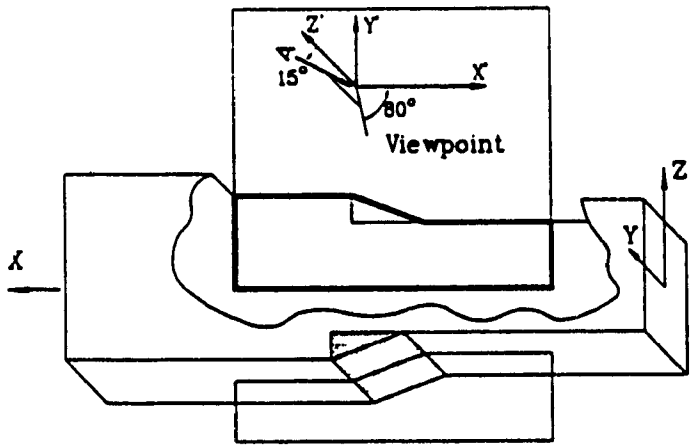


FIGURE 5.7

$\sigma^{*}_{x',y'}$, Tension Load, All 0°, External Drop-off, Y=0



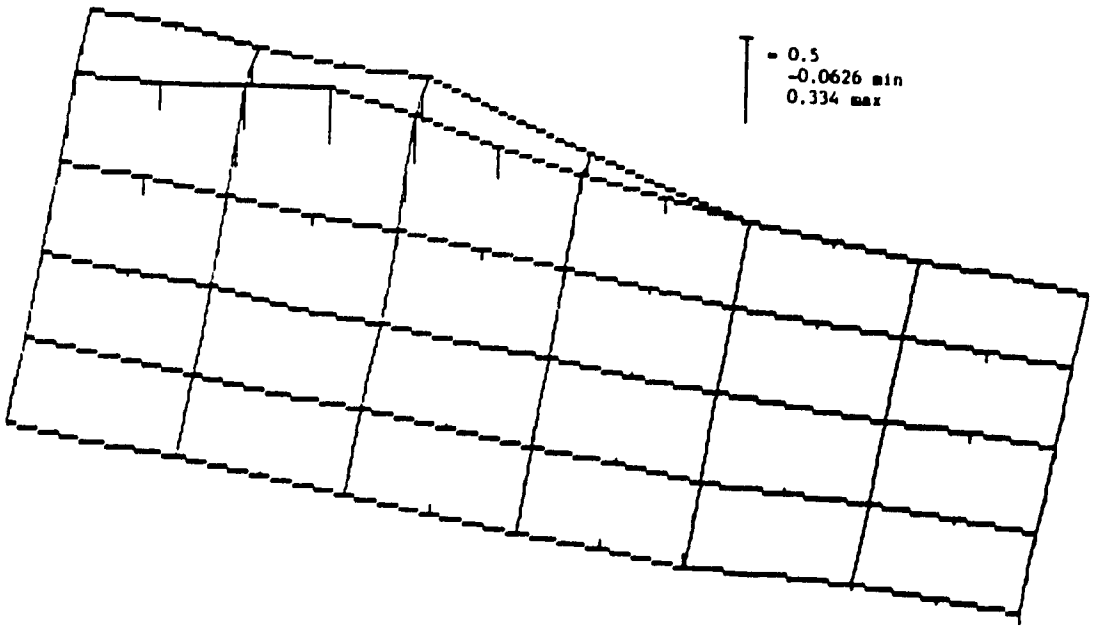
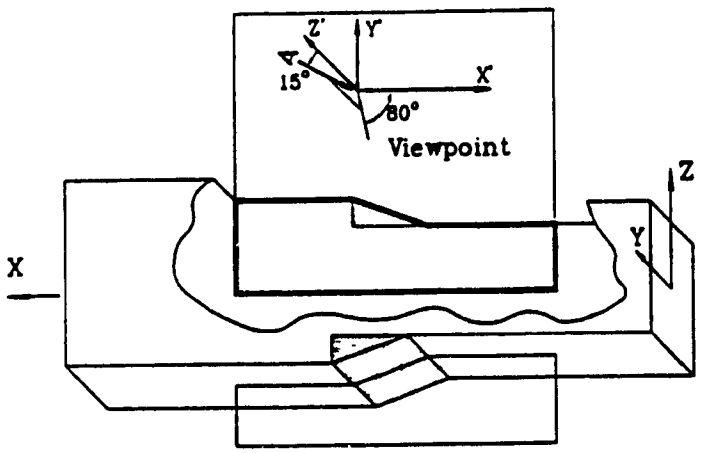


FIGURE 5.8

$\sigma^{*}_{x',z'}$, Tension Load, All 0°, External Drop-off, Y=0



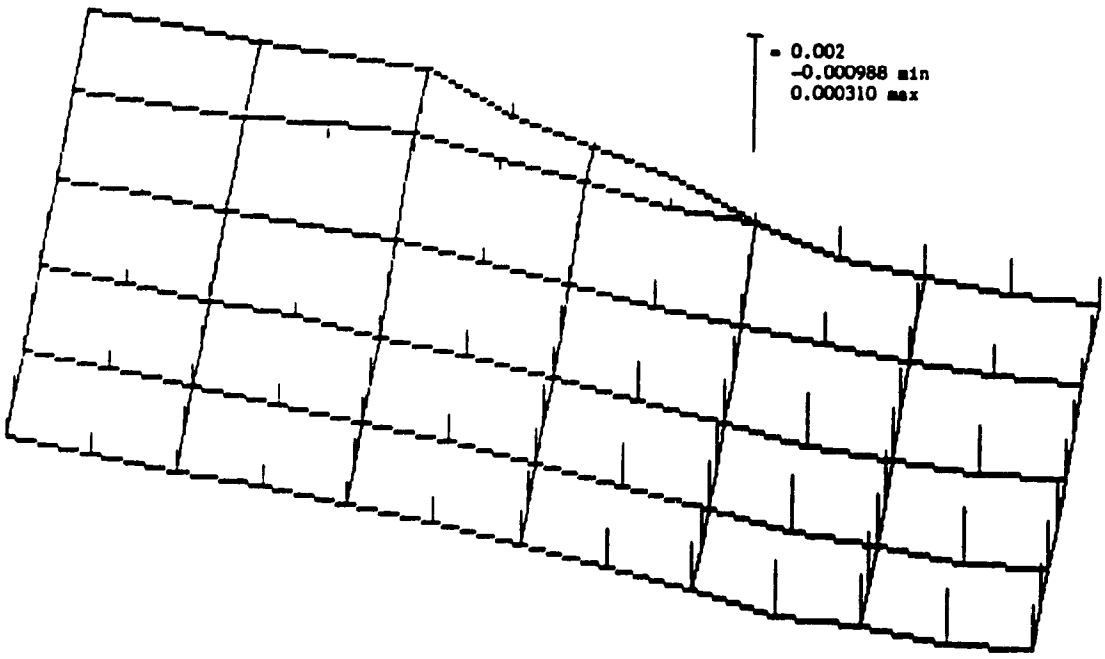
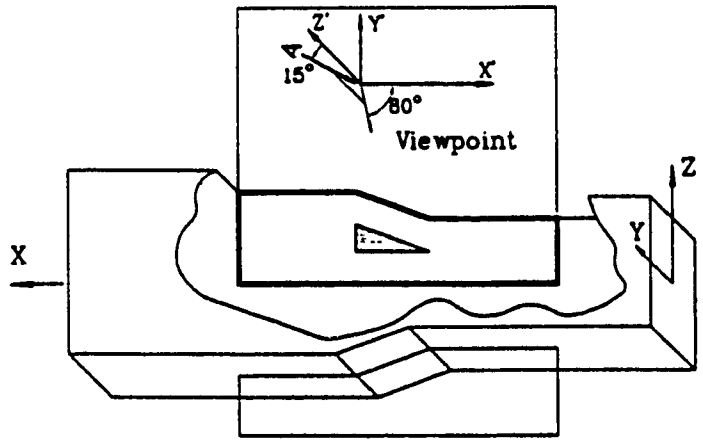


FIGURE 5.9

$\sigma^{*}_{y',z'}$, Tension Load, All 0°, External Drop-off, Y=0



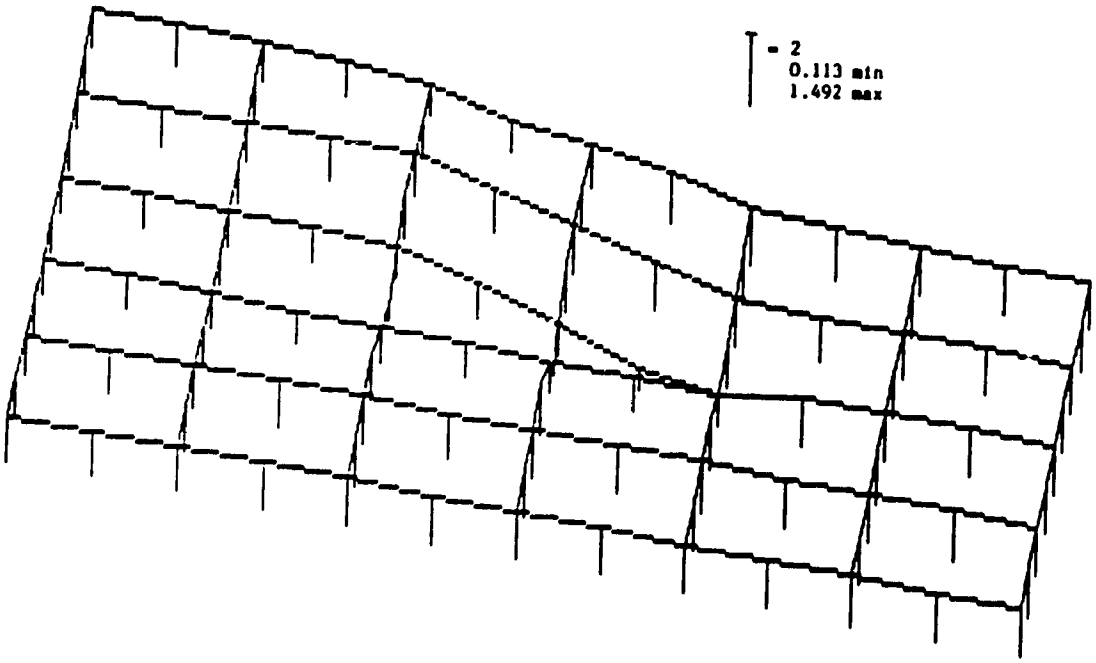
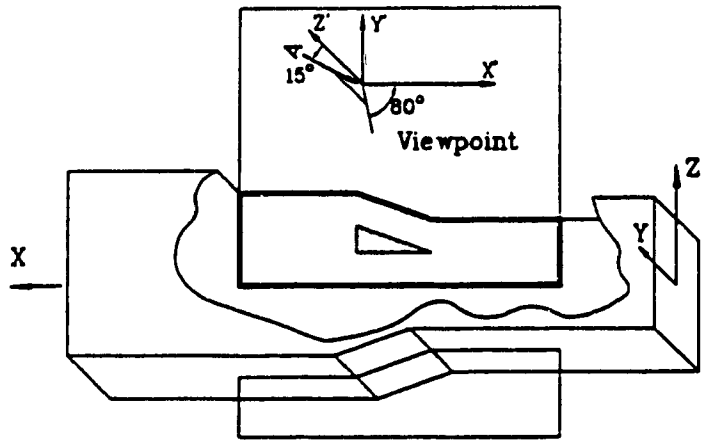


FIGURE 5.10

σ_x^*, , Tension Load, All 0°, Internal Drop-off, Y=0



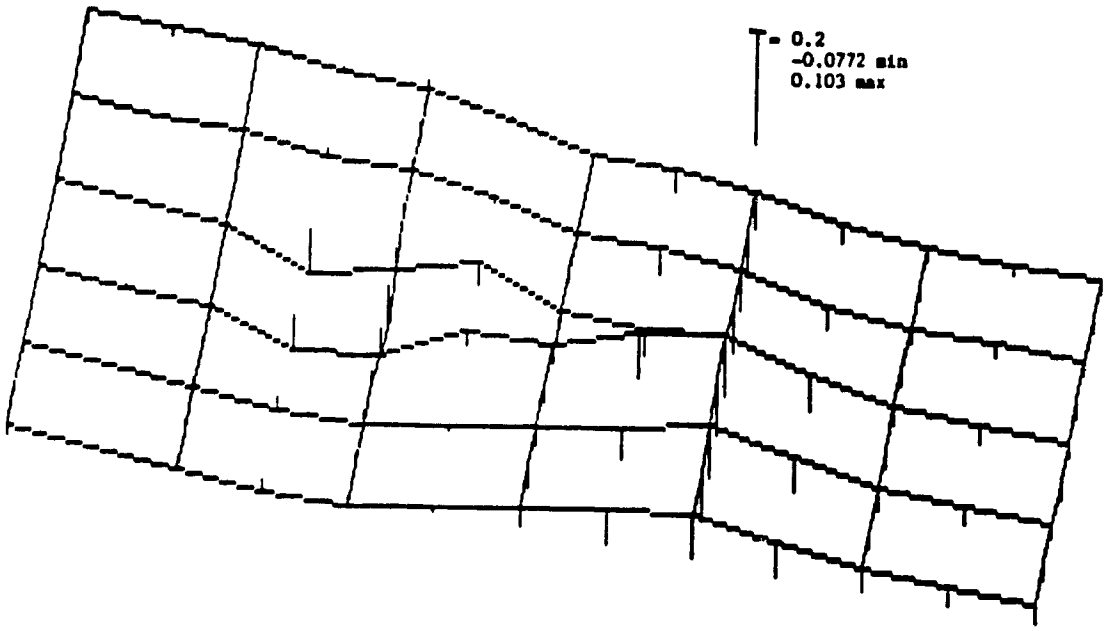
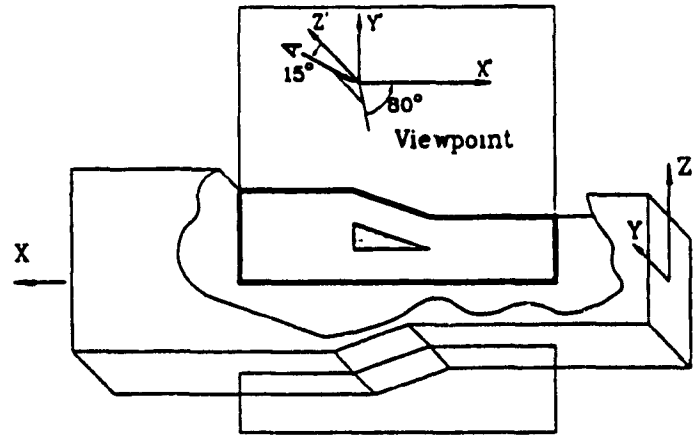


FIGURE 5.11

σ_y^*, , Tension Load, All 0°, Internal Drop-off, Y=0



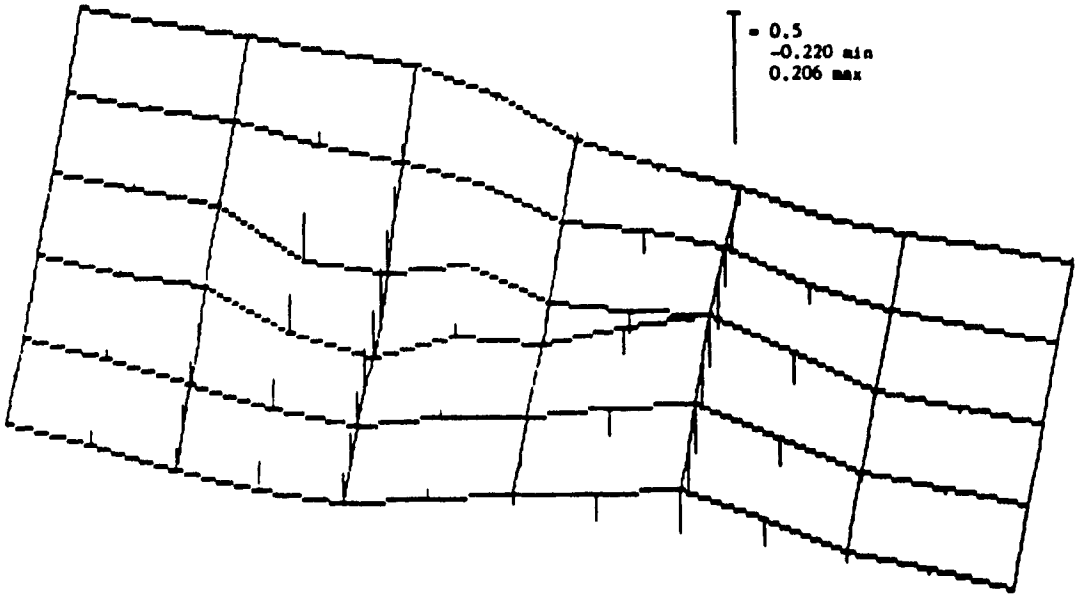
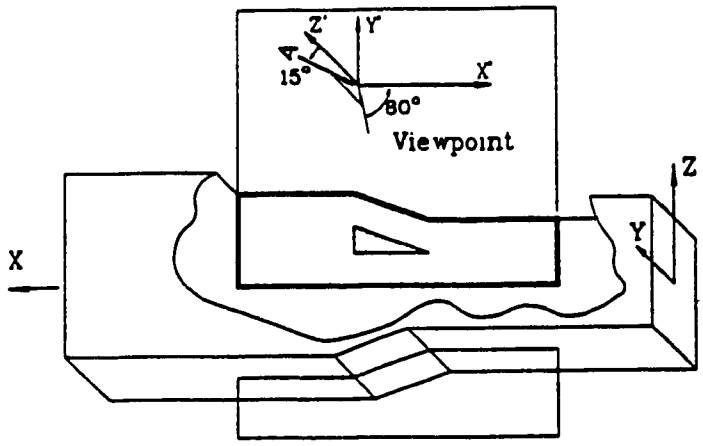


FIGURE 5.12

σ_z^*, , Tension Load, All 0°, Internal Drop-off, Y=0



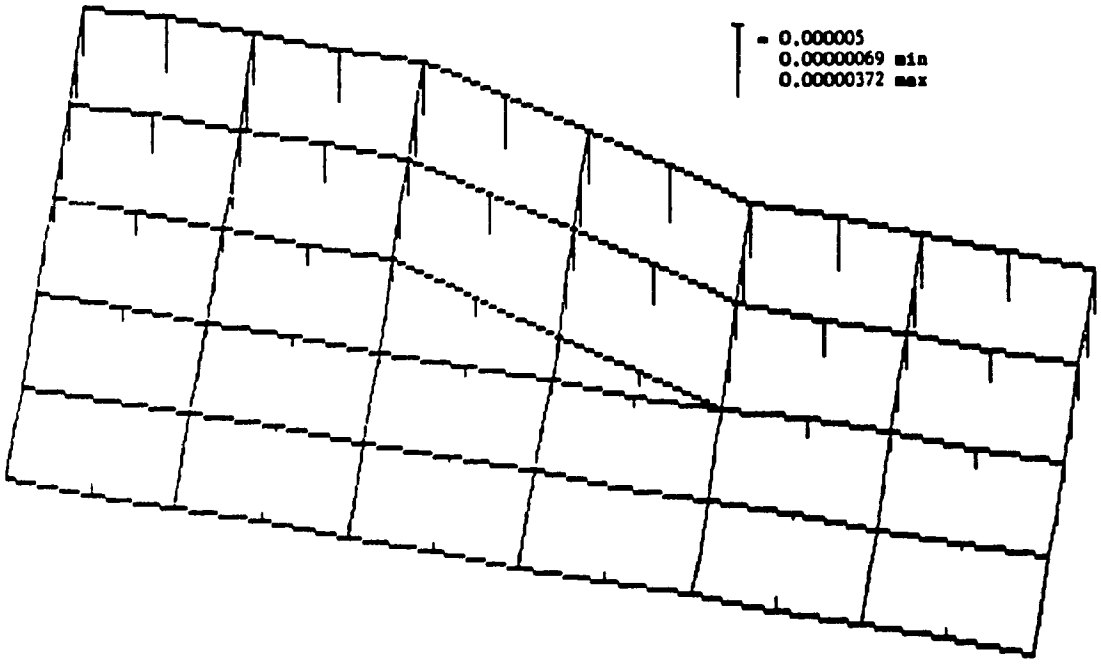


FIGURE 5.13

$\sigma^{*}_{x'y'}$, Tension Load, All 0°, Internal Drop-off, Y=0

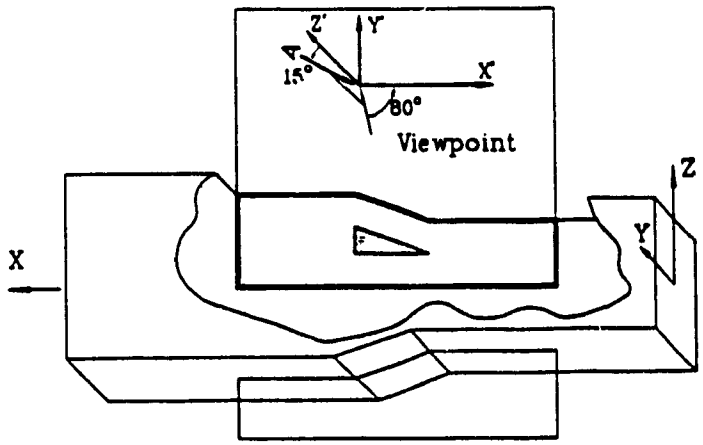
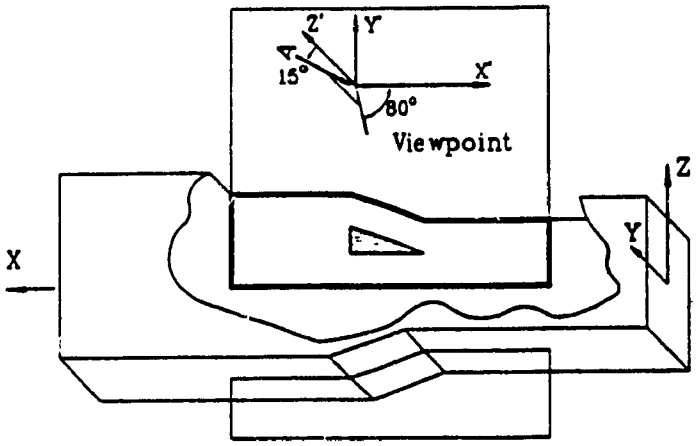


FIGURE 5.14

$\sigma^{*}_{X'Z'}$, Tension Load, All 0°, Internal Drop-off, Y=0



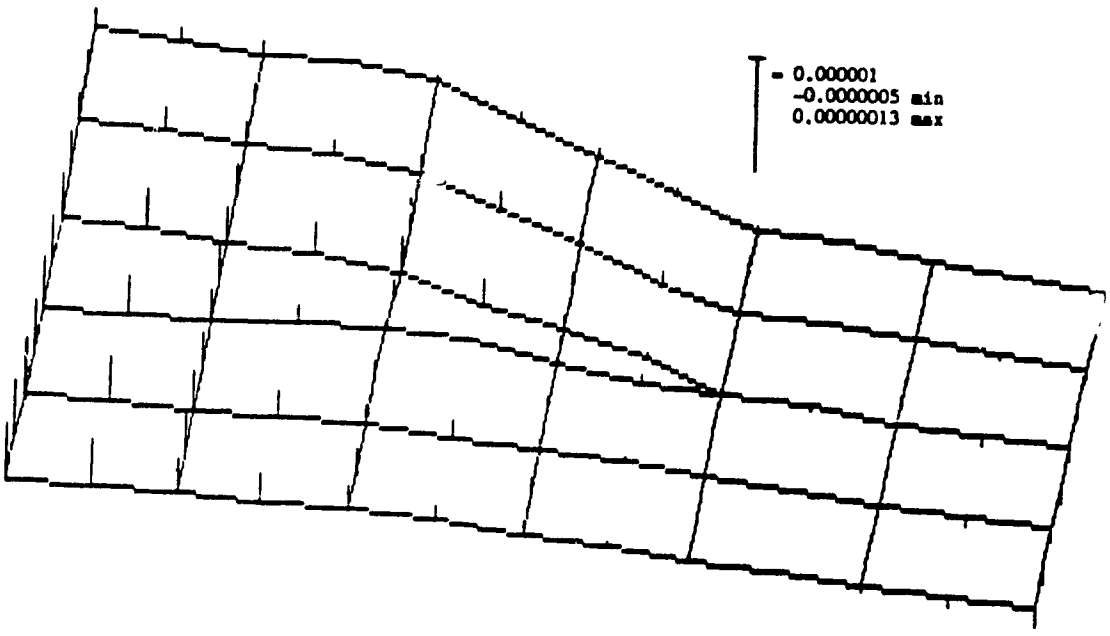


FIGURE 5.15

$\sigma^{*}_{y',z'}$, Tension Load, All 0°, Internal Drop-off, Y=0

Figures 5.16 to 5.21 compare σ^*_x, σ^*_y, σ^*_z, $\sigma^*_{x'y'}$, $\sigma^*_{x'z'}$, $\sigma^*_{y'z'}$ around the drop-off wedge contour for the external and internal2 models. The drop-off wedge contour is divided into three segments: the horizontal, vertical and oblique edges. The external model has a lower σ^*_x, at the vertical - oblique edge intersection due to the discontinuity of high modulus material above the horizontal edge. σ^*_y, is lower for the internal2 model except at the tip where $\sigma^*_{y'}$=0.12. σ^*_z, is low for the external drop-off model due to a free oblique edge. $\sigma^*_{x'y'}$, $\sigma^*_{y'z'}$, are neglectable. The distribution of $\sigma^*_{x'z'}$, has a similar shape for both models, but in the case of the internal2 model, the distribution is more centered along the $\sigma^*_{x'z'}$=0 axis. The external model reaches a maximum of $\sigma^*_{x'z'}$=0.28. The greater the imbalance of high modulus material over and under the drop-off, the more the average $\sigma^*_{x'z'}$, is offset from zero.

5.3.3 Comparison

Table 5.1 shows the safety factor (1/R as in section 4.3.3) for laminates containing drop-offs at different locations. All laminates including the reference laminate were modelled and solved with finite elements using similar meshes.

TABLE 5.1

Safety Factor for Laminates with Drop-offs at Different Locations

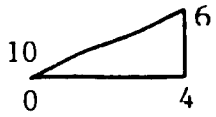
Case	Loading Condition		
	Tension	Bending	Torsion
Reference	3.2438	0.0008946	0.0004497
External	0.9771	0.0003148	0.0001411
Internal 1	2.1513	0.0005717	0.0004466
Internal 2	2.2441	0.0006057	0.0004456
Internal 3	1.8502	0.0006637	0.0004215

From this table, the efficiency as defined in section 5.2 was evaluated and is shown in the Table 5.2:

TABLE 5.2

Efficiency for Laminates with Drop-offs at Different Locations

Case	Loading Condition			Statistics	
	Tension	Bending	Torsion	Average	Lowest
External	30%	35%	31%	32%	30%
Internal 1	66%	64%	99%	77%	64%
Internal 2	69%	68%	99%	79%	68%
Internal 3	57%	74%	94%	75%	57%



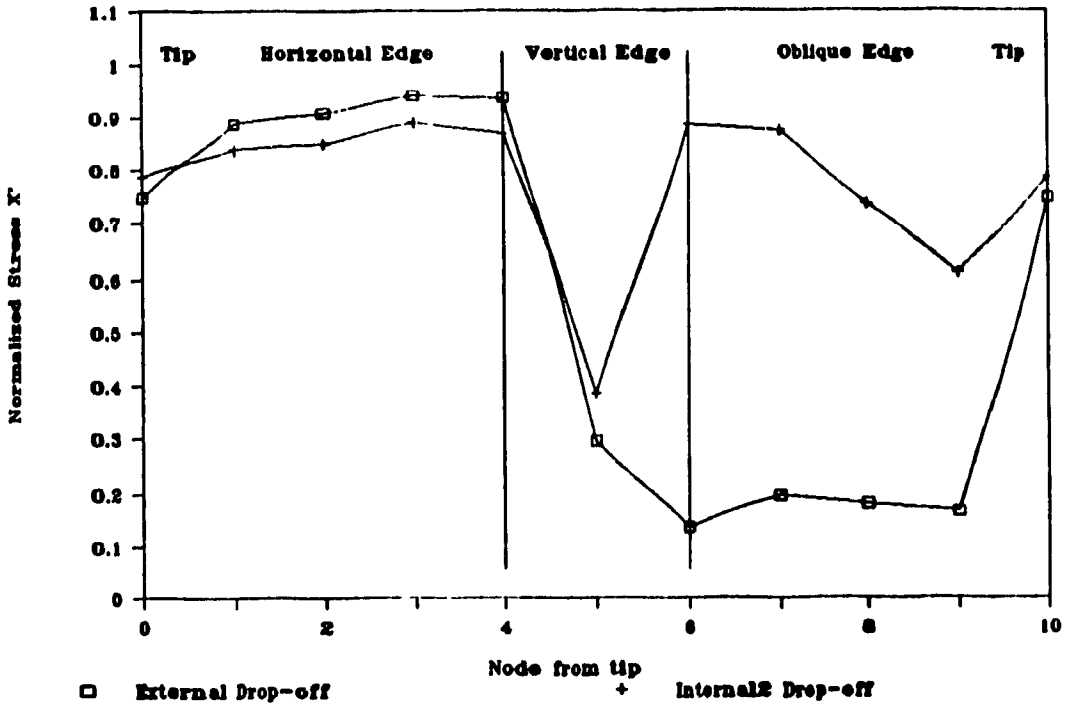


FIGURE 5.16

σ^*_x, Comparison at Drop-off Wedge Contour, Y=0

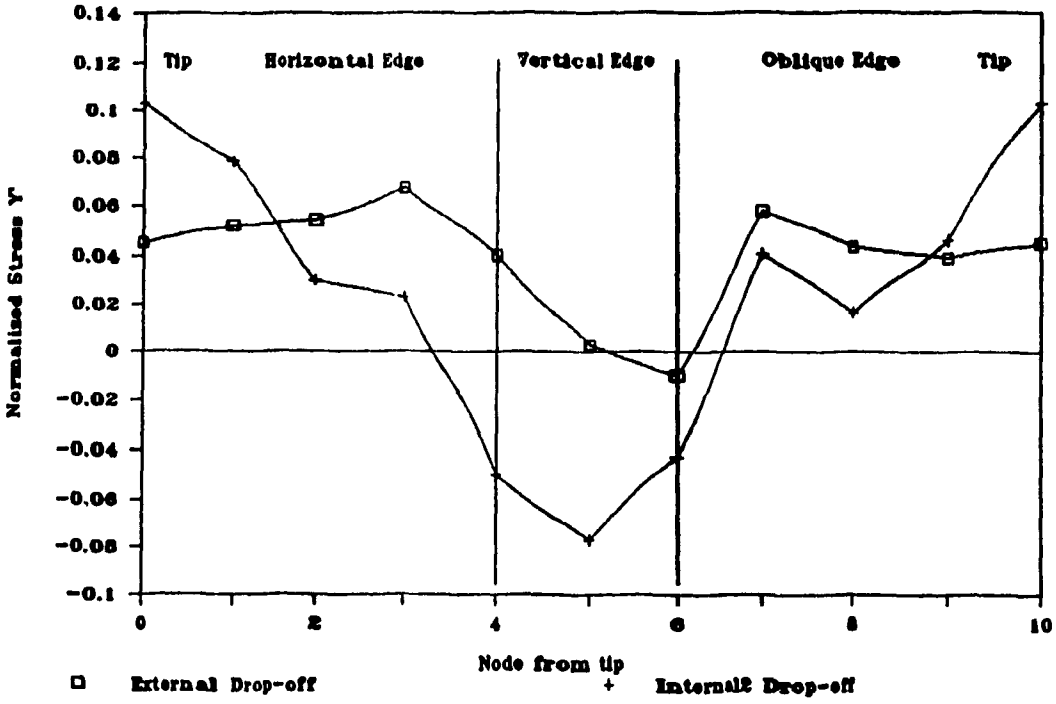


FIGURE 5.17

σ_y^*, Comparison at Drop-off Wedge Contour, Y=0

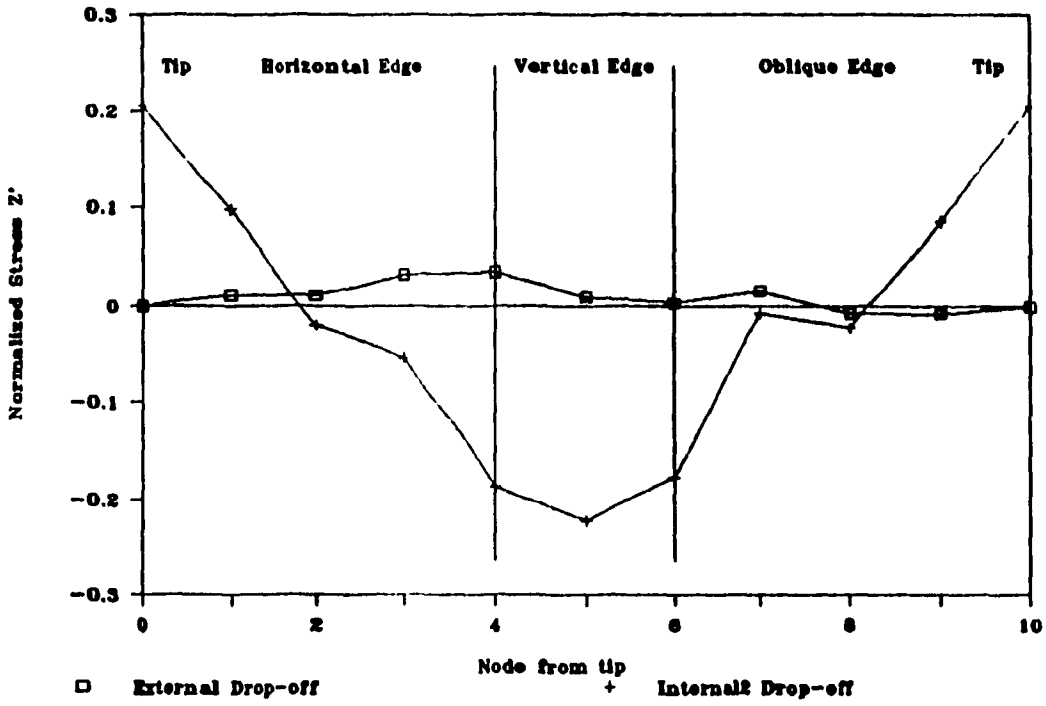


FIGURE 5.18

σ_z^*, Comparison at Drop-off Wedge Contour, Y=0

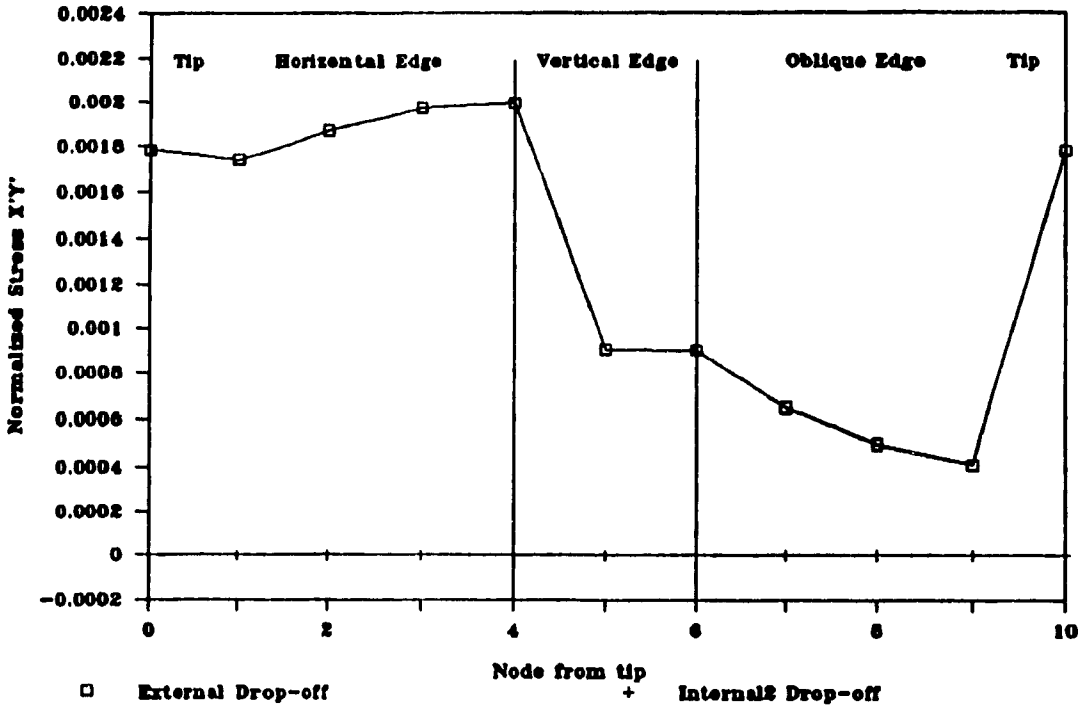


FIGURE 5.19

$\sigma^{*}_{x'y'}$ Comparison at Drop-off Wedge Contour, Y=0

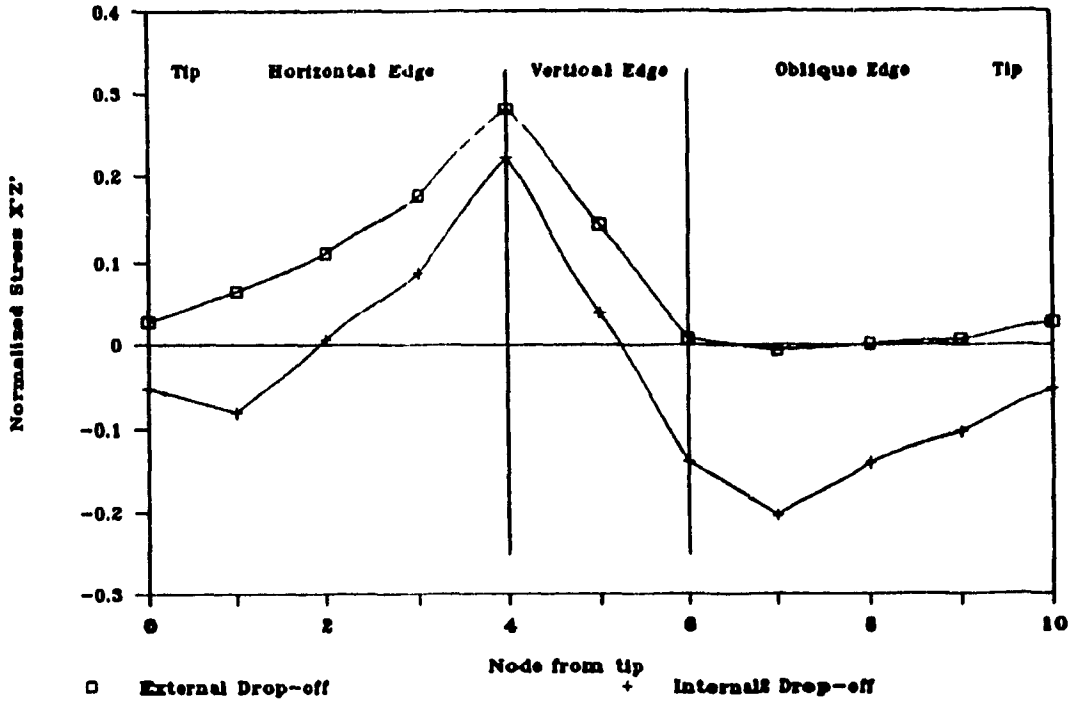


FIGURE 5.20

$\sigma^{*}_{x',z'}$, Comparison at Drop-off Wedge Contour, Y=0

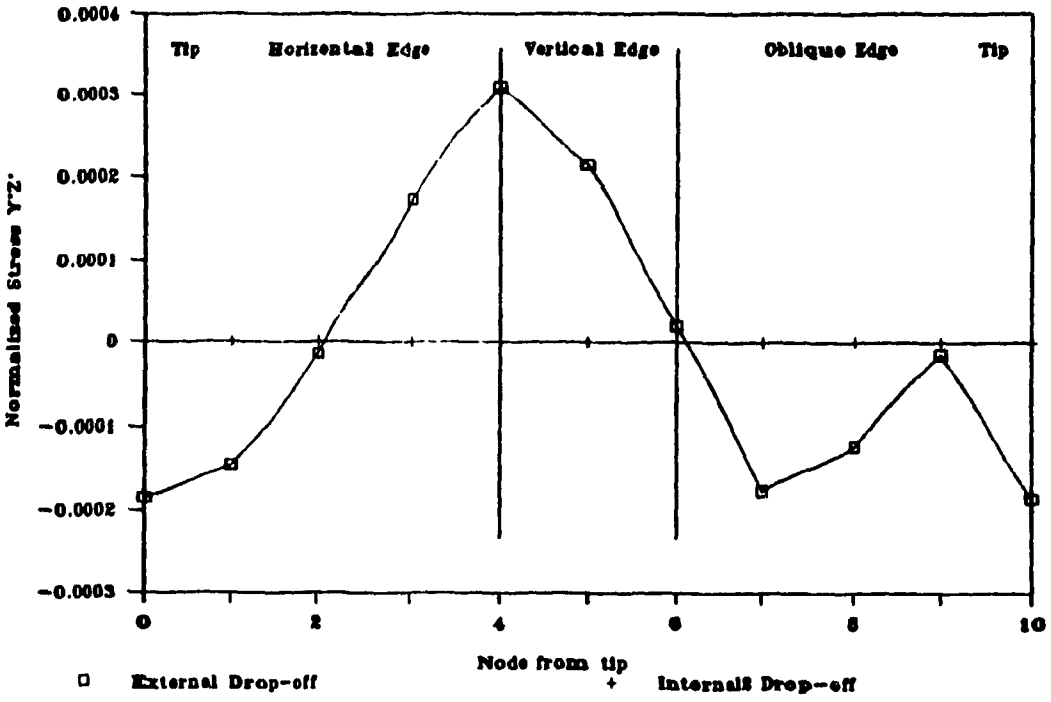


FIGURE 5.21

$\sigma^{*}_{y'z'}$, Comparison at Drop-off Wedge Contour, Y=0

Some conclusions can be derived from Table 5.2:

- Any internal drop-off configuration is more efficient than an external drop-off configuration under all loading conditions by a ratio of about 2 to 1 or more.
- Tapered laminates with internal drop-off have near to 100% efficiency when loaded under torsion.
- An internal drop-off location placed halfway between the mid-plane and the top or bottom of the laminate (internal2, 69% efficiency) is best suited for tension loading conditions.
- Placing the drop-off nearest to the mid-plane is optimal for bending loading conditions (internal3, 74% efficiency).
- Considering the lowest efficiency number under all loading conditions, a drop-off located halfway between mid-plane and top or bottom of the laminate provides the best performance (internal2, 68% efficiency).

Drop-offs nearest to the mid-plane were initially thought of as being the best for tensile loading conditions. However the thickness of the layers over the drop-off can contribute to peeling stresses at the tip of the drop-off which can be the controlling failure stress. At the same time, less layers above the drop-off can result in severe shear stress in the drop-off region. It was observed that for the external drop-off model, the largest value of R (section 4.3.3) occurs at the tip of the drop-off wedge and Y=0 in shear (σ_{xz}) mode. At the same time for the internal drop-off models, highest values of R occur at the tip

of the drop-off wedge and Y=0 in peeling mode (σ_z).

5.4 Drop-off Wetness

Wetness of the drop-off represents here the amount of resin which flowed to fill the drop-off area during the curing process. When using fresh material and curing with appropriate temperature and pressure cycles, resin usually flows well and fills any defect or drop-offs which should have been left empty of any air bubbles under the vacuum cycle.

This study will answer the following questions:

- How good is a laminate with dry drop-off areas?
- How much should a drop-off be filled to compare with a completely filled drop-off laminate?

For the purpose of studying drop-off wetness, a mesh similar to that of Figure 5.3 was done for three levels of drop-off wetness:

- 0% resin content in drop-off
- 50% resin content (as shown in Figure 5.22)
- 100% resin content

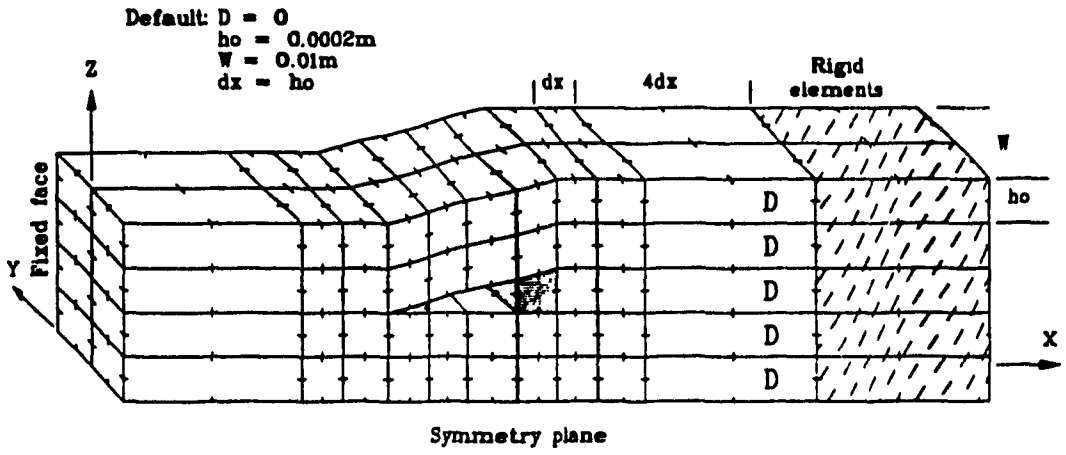


FIGURE 5.22

Wetting Mesh

The angle of the fibers is known as D in the mesh illustration. Two fiber angles were considered: 0° and 90°. The loading condition was limited to tension and symmetry features for the laminate was used, therefore only the upper half of the laminate was considered. The mesh shown in Figure 5.1 was used as a reference for determining efficiency levels.

A tensile force of 1000 N was applied to the center of the rigid wall. Due to fixed displacements in z direction at z=0, this model can only slide along the x direction. Table 5.3 presents the safety factor of the cases solved:

TABLE 5.3

Drop-off Wetness Safety Factor

Case	Fiber Angle 0°	90°
Reference	3.244	0.420
0% wet	1.476	0.275
50% wet	1.785	0.299
100% wet	1.821	0.356

From Table 5.3, the efficiency can then be determined as shown in Table 5.4:

TABLE 5.4

Drop-off Wetness Efficiency

Case	Fiber Angle 0°	90°
0% wet	46%	66%
50% wet	55%	71%
100% wet	56%	85%

From Table 5.4, the following observations can be made:

- The 90° laminate is affected most by the degree of wetness: from 66% to 85% efficiency, giving a 19% improvement compared to only 10% improvement for the 0° laminate.
- A half full drop-off region in the 0° laminate gives similar results as a completely filled drop-off.
- A 100% drop-off wetness in a 90° laminate makes a significant difference over half-filled conditions.

It was observed that for the dry 0° laminate (no resin wetness) the highest R value was found at the drop-off tip and it corresponds to σ_z whereas for the dry 90° laminate, the highest R value was found at the upper right corner of the drop-off and it corresponds to σ_x in tension. Adding a small amount of resin to the tip of the drop-off in the 0° laminate certainly can reduce the amount of discontinuity in the z direction at the tip. This therefore reduces the σ_z value. On the other hand for the 90° laminate the drop-off has to be filled to reduce the degree of

discontinuity in the x direction.

5.5 Drop-off Shape

The shape of the drop-offs can change according to the fabrication process of the laminate. The fiber-free region at the drop-off can be idealized as a wedge. The basic wedge shape is retained, but the ratio of the height, hd, (laminae thickness) over the length, Ld, (distance after which the plies over and under the drop-off meet) is varied to better understand its effect upon interlaminar stresses.

The hd/Ld ratio took the values of 0.1, 0.2 and 0.4. These numbers were based upon experimental observation. Two fiber orientations were considered for the different hd/Ld ratios: 0° and 90°.

The loading condition was tension, applied by a 1000 N force in the x direction. The same mesh generation file was able to generate the meshes for all 6 cases, using input variables hd and d1 through d5 (Figure 5.3). Tables 5.5 and 5.6 present the safety factor and efficiency for all cases:

TABLE 5.5

Safety Factor of Laminates with Drop-offs of Different Shapes

$\frac{hd}{Ld}$	Fiber Angle in Laminate	
	0°	30°
0 (Reference)	3.2438	0.4201
0.1	3.3975	0.3897
0.2	2.5724	0.3695
0.4	1.4659	0.3349

Using results in Table 5.5, the efficiency was calculated and is shown in Table 5.6.

TABLE 5.6

Efficiency of Laminates with Drop-offs of Different Shapes

$\frac{hd}{Ld}$	Fiber Angle in Laminate	
	0°	90°
0.1	105%	93%
0.2	79%	88%
0.4	45%	80%

The 105% efficiency in Table 5.6 is an artifact that can result from the increase in element dimensions as one moves from the reference laminate to the laminate that contains a drop-off which has hd/Ld of 0.1.

A more careful approach could have been done by changing the

hd/Ld ratio but keeping the element volume constant by correcting the drop-off height. Still, basic observations can be made:

- The 0° laminate is the most affected by the hd/Ld ratio.
- Reducing the hd/Ld ratio is beneficial.

Future work could include developing methods for reducing the hd/Ld ratio. Perhaps including a resin fillet at the tip of a cut laminae during laminate stacking before curing could build more pressure inside the normally empty wedge and extend the wedge length.

5.6 Laminate Width

Interlaminar stresses in regular composite laminate are known to be localized within a laminate thickness distance from the edges [Pipes and Pagano (1970)] [Pipes and Daniel (1971)]. Therefore, increasing the laminate width could lead to improving the percentage of the region without interlaminar stresses due to edge effect inside laminates. Varying the ratio of laminate width over the thinnest cross-section thickness, W/hmin, is analyzed for tapered laminates.

Usually the W/hmin ratio has a range of 1 to 10 in tapered plate laminate designs. The range was extended to include the following W/hmin ratios: 100, 10, 1 and 0.1 for tension and

bending. Higher numbers can be reached for circular designs like flywheels of tapered tubing that have no edge effects.

The model studied is shown in Figure 5.3. All fiber angles were consider to be 0°. A tension force, bending and torsion moments were applied to the large cross-section extremity (+x face), respectively having values of 1000 N, 1000 Nm and 1000 Nm. Table 5.7 presents results for safety factor and efficiency of all cases:

TABLE 5.7

Effect of Width on Efficiency

	$\frac{W}{hmin}$	Safety Factor		Efficiency
		Tapered	Reference	
Tension	100	18.8181	22.6920	83%
	10	3.5827	4.9037	73%
	1	0.3615	0.7639	47%
	0.1	0.0368	0.0786	47%
Bending	100	9.4580E-3	1.3911E-2	68%
	10	9.7002E-4	1.4212E-3	68%
	1	9.7870E-5	2.2761E-4	43%
	0.1	9.9950E-6	2.5743E-5	39%
Torsion	10	1.1020E-3	1.1210E-3	98%
	1	2.0537E-5	2.1193E-5	97%

Observations:

- A higher W/hmin ratio is beneficial for tapered laminates under

tension and bending.

- Efficiency of tapered laminates under torsion is near 100%.
- The efficiency under tension and bending is affected in a similar fashion: from 70% to 45% for respective W/hmin ratios of 10 and 1 which represent most design ranges.
- Efficiency outside the normal design range of W/hmin, 10 to 1, remains similar to the limit values of this range.

5.7 Failure Location

As seen in the section describing the effect of drop-off wetness, the failure location, failure mode and material continuity at the failure location in the direction of the stress initiating failure are related. Following the failure location for laminates under different loading conditions can be useful to explain results.

Basic 0° and 90° tapered laminates under tension, bending and torsion were studied by using the mesh shown in Figure 5.3. A tension force, bending and torsion moments were applied to the large cross-section extremity (+x face), respectively having values of 1000 N, 1000 Nm and 1000 Nm.

Results are presented by Table 5.8 and Figure 5.23 which graphically indicates the failure locations.

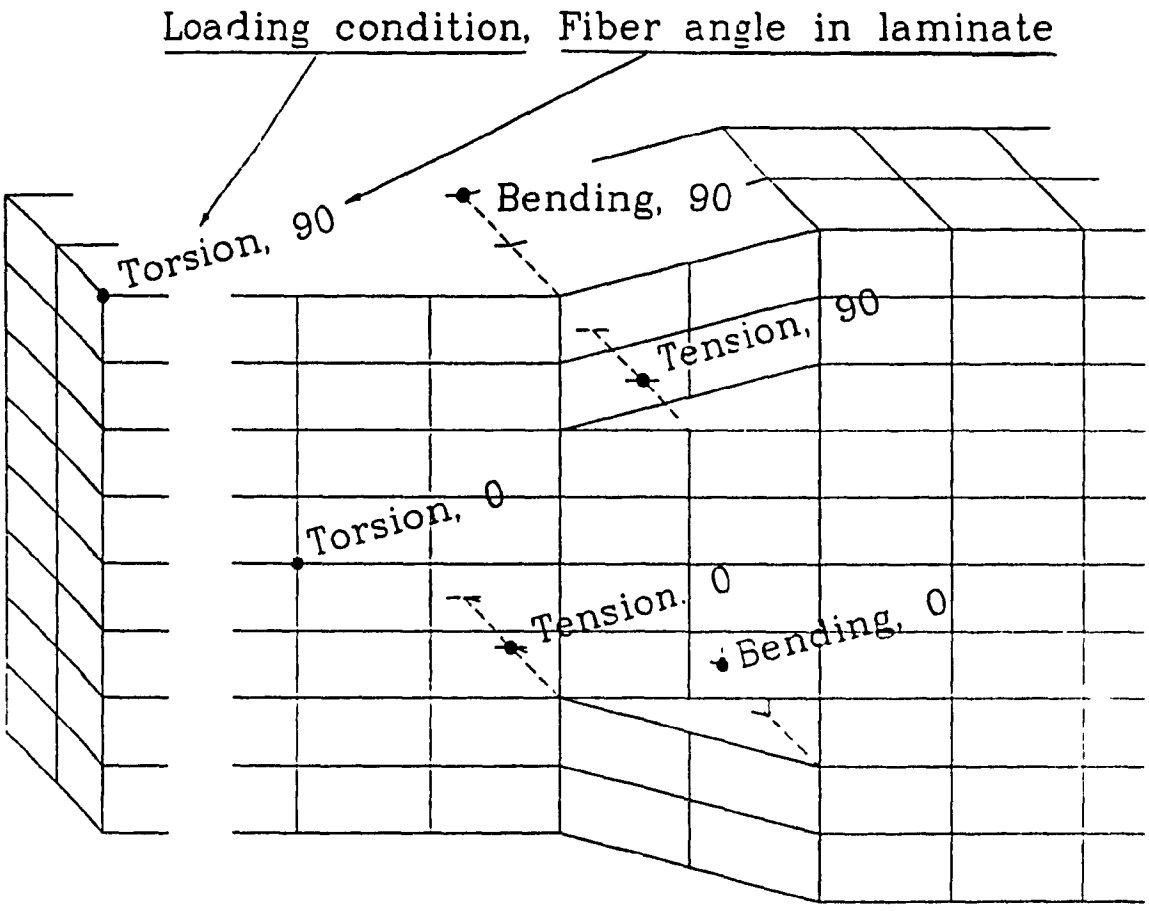


FIGURE 5.23

Failure Locations

TABLE 5.8

Failure Locations for Single Loading Conditions

Angle for the Whole Laminate	Loading Condition	Failure Location, Mode and Direction
0°	Tension	Tip of lower wedge, y=0, tension σ_3
90°	Tension	Mid bottom of upper wedge, y=0, tension σ_2
0°	Bending	Lower right corner of lower wedge, +y face, tension σ_3
90°	Bending	Maximum z over upper wedge tip, +y face, tension σ_2
0°	Torsion	Midplane at two elements from tip, -y face, shear σ_{13}
90°	Torsion	Fixed face at maximum z, -y face, tension σ_2

Before any interpretation, here are a few details:

- Direction 1 is that of the fibers.
- Direction 2 is transverse to the fibers, in the same plane as the ξ-η plane of the element in Figure 3.2.
- Direction 3 is normal to the ξ-η plane of the element.
- All failure locations occur inside the glass/epoxy material and not the epoxy elements inside the drop-off wedges.
- The method for finding the worst node considers the highest stress extrapolated from the optimum sampling points of an element. Stresses are not averaged at the nodes.

Observations:

- Under torsion, the failure location is near the fixed face of the model. The same failure location is found in regular laminates, explaining the efficiency of tapered laminates under torsion. This shows that torsional failure is not affected by the presence of the drop-off.
- Under tension and bending, only the 0° laminate fails due to interlaminar peeling stresses. The 90° laminate fails due to stress in a direction close to that of the x axis (transverse to fiber direction).
- There are many possible failure locations around the drop-off wedge.

More layup configurations were also considered, and more failure locations were found. No particular location, besides the general drop-off region, can be identified as the failure point.

5.8 Combined Loading Conditions

In this section, the contribution of individual loading components to the strength of a laminate under combined loading conditions is studied.

The strength of the laminates (0° and 90° layups) can be found from the safety factor of the cases studied in the previous

sections. The mesh is shown in Figure 5.3. Tension force, bending and torsion moments were applied to the large cross-section extremity (+x face), with respective values of 1000 N, 1000 Nm and 1000 Nm. Table 5.9 presents the results.

TABLE 5.9

Safety Factor of Single Loading Conditions

Angle D	Loading Condition	Safety Factor
0°	Tension	2.2441
90°	Tension	0.3606
0°	Bending	6.058E-4
90°	Bending	9.760E-5
0°	Torsion	4.457E-4
90°	Torsion	1.067E-4

From Table 5.9, the maximum single loading conditions can be found by multiplying the safety factor by the imposed loading conditions. The possible combinations obtained from 0%, 50% and 100% of each maximum loading conditions are considered. Theoretically, 3^3=27 possibilities exist (3 levels of 3 loading conditions). Of the original 27, some combined loading conditions are similar, like 50% of tension and bending compared to 100% of tension and bending. The reduced amount of combinations is 16.

With assumed linear elastic behavior, superposition was used and the combined loading cases were not solved through the process

of calculating and reducing the global stiffness matrix of the laminate. The displacements from the previous single loading conditions were independently multiplied by the appropriate ratios and summed to produce the solution of the combined loading conditions. This procedure is valid for elastic material properties and was conveniently implemented into a batch mode of *DIRECT*® (Appendix B). The safety factor for the combined loading cases are shown in Tables 5.10 and 5.11.

TABLE 5.10

Combined Loading Results for 0° Laminate

Fraction from Maximum			Safety
Tension	Bending	Torsion	Factor
1	0	0	1.0000
0	1	0	1.0000
0	0	1	1.0000
0	1	1	0.7135
0	0.5	1	0.9006
0	1	0.5	0.9057
1	0	1	0.5825
0.5	0	1	0.8130
1	1	0	0.5851
1	1	1	0.5335
0.5	0.5	1	0.7685
0.5	1	0	0.8271
0.5	1	0.5	0.7974
0.5	1	1	0.6977
1	0	0.5	0.7434
1	0.5	0	0.7380
1	0.5	0.5	0.7061
1	0.5	1	0.5593
1	1	0.5	0.5768

TABLE 5.11

Combined Loading Results for 90° Laminate

Fraction from Maximum			Safety
Tension	Bending	Torsion	Factor
1	0	0	1.0000
0	1	0	1.0000
0	0	1	1.0000
0	1	1	0.5420
0	0.5	1	0.7030
0	1	0.5	0.7432
1	0	1	0.5369
0.5	0	1	0.6988
1	1	0	0.5273
1	1	1	0.3693
0.5	0.5	1	0.5395
0.5	1	0	0.6906
0.5	1	0.5	0.5628
0.5	1	1	0.4393
1	0	0.5	0.7337
1	0.5	0	0.7159
1	0.5	0.5	0.5600
1	0.5	1	0.4376
1	1	0.5	0.4529

New failure locations are found in the combined loading conditions which were not found in single loading conditions. Summing the stresses of single loading conditions cases can be beneficial at some previous locations while other locations can become worse. The spectrum of failure locations makes it impossible to reliably sum results from only a few failure

locations obtained from single loading cases to predict failure due to combined loading cases. Summing has to be done over the whole model.

Using the results in Table 5.10 and 5.11, it was attempted to obtain an expression that could predict the safety factor of a model under combined loading conditions utilizing the safety factors from the single loading conditions and the state of the combined loads.

A regression program was developed for *DIRECT*® (REGRES.pas in Appendix B), to edit functions and return goodness of fit values when matched to data with many variables of any degree. Although complicated functions were found to have a better fit, retaining the best compromise of fit and function simplicity was a goal, a simpler function will indicate more clearly the effect of each variable. The functions are presented in Table 5.12:

TABLE 5.12

Safety Factor Functions*

0° Laminate $SF = 1.6524 - 0.4964 \times A - 0.6118 \times B - 0.5620 \times C + 0.2508 \times AB + 0.2427 \times AC + 0.3820 \times BC - 0.1909 \times A^2 - 0.0351 \times B^2 - 0.0873 \times C^2$ $\Sigma \left\|SF_i - Exact(SF)\right\|^2 = 0.0073$ $Max \left\|SF_i - Exact(SF)\right\| = 0.0402$
90° Laminate $SF = 1.7631 - 0.9485 \times A - 0.9273 \times B - 0.8359 \times C + 0.3054 \times AB + 0.3188 \times AC + 0.3311 \times BC + 0.1773 \times A^2 + 0.1527 \times B^2 + 0.0570 \times C^2$ $\Sigma \left\|SF_i - Exact(SF)\right\|^2 = 0.0032$ $Max \left\|SF_i - Exact(SF)\right\| = 0.0245$

*SF is the approximated Safety Factor

A: Tension/(Maximum Tension), $0 \le A \le 1$

B: Bending/(Maximum Bending), $0 \le B \le 1$

C: Torsion/(Maximum Torsion), $0 \le C \le 1$

Second degree relationships between the safety factor of combined loading conditions cases and individual loading components were established with a maximum error of 4% for a given laminate. Although the coefficients of the functions can depend of sampling quantity, the effect of each loading component can be seen.

The safety factor of a combined loading state is more proportional (linear) to individual loading components for the 90° laminate (higher coefficients for A, B, C and lower coefficients

for A^2 and C^2). The relation between loading components is more important for the 90° laminate (generally higher coefficient for AB, AC BC).

5.9 Fiber Angle at Drop-off

An often considered design parameter is the angle of the fibers at the drop-off. The mesh shown in Figure 5.3 was used for optimizing the angle of the layers in contact with the drop-off under tension and bending. The imposed tension force and bending moments were of 1000 N and 1000 Nm respectively. The material was CE 9000 glass/epoxy with a epoxy-filled drop-off wedge.

Angles d1 and d5 were kept at 0 degrees, while d2 = d3 = d4 were set to 0°, 22.5°, 45°, 67.5° and 90°. The results are presented in Tables 5.13 and 5.14:

TABLE 5.13

Efficiency for Laminates with Different Angles d2, d3 and d4 under Tension

Angle	Safety Factor		Efficiency
	Tapered	Reference	
0°	2.2441	3.2438	69%
22.5°	2.0628	2.1539	96%
45°	1.0850	1.1722	93%
67.5°	0.7350	0.8575	86%
90°	0.6564	0.8142	81%

TABLE 5.14

Efficiency for Laminates with Different Angles d2, d3 and d4 under Bending

Angle	Safety Factor		Efficiency
	Tapered	Reference	
0°	6.058E-4	8.946E-4	68%
22.5°	5.802E-4	8.676E-4	67%
45°	5.292E-4	5.315E-4	99%
67.5°	3.545E-4	3.573E-4	99%
90°	3.127E-4	3.172E-4	99%

Observations:

- Safety factor is highest with all 0° laminates under tension (SF=2.24) or bending (SF=0.00061)
- Highest efficiency (96%) for laminates under tension is reached with 22° fiber angle in contact with the drop-off
- Fiber angle in the range of 45° to 90° provides the best

efficiency under bending conditions (99%)

- 45° fiber orientation is the best compromise for efficiency under tension and bending providing at worst 93% efficiency
- Efficiency can vary by 30% when fiber orientation is changed from 0° to 90°.

5.10 Crack Growth Simulation

Presently, there is no proven crack growth simulation method. Removing "failed" elements from an initial geometry provides a crude method of simulation. This method is used to simulate crack growth under tension and bending of a laminate.

There are inherent problems with this method:

- Removing an element will generate new stress concentration locations.
- The failure mode can change between the elements removed.
- Simple pealing or delamination is not well represented.
- The crack propagation can stop and start elsewhere.

However due to the lack of better methods, this method is used here. Simple 0° and 90° laminates along with a more complex $[0_2/45/-45/90]_S$ laminate layup were considered under tension and bending. The last layup was intended for seeing crack growth through plies of different orientations. The mesh is shown in

Figure 5.24. The mesh has been locally refined around the drop-off area.

In order to determine if a crack growth simulation is a probable one, there must be a continuity in the crack direction, failure mode and safety factors.

5.10.1 Crack Growth under Tension

A tension force of 1000 N was imposed. The consecutive removal of elements is represented in Figures 5.25 to 5.27 for the three different layups. 8 elements were consecutively removed (each case was separately solved). Table 5.15 presents the safety factor of the layup for a given number of removed elements.

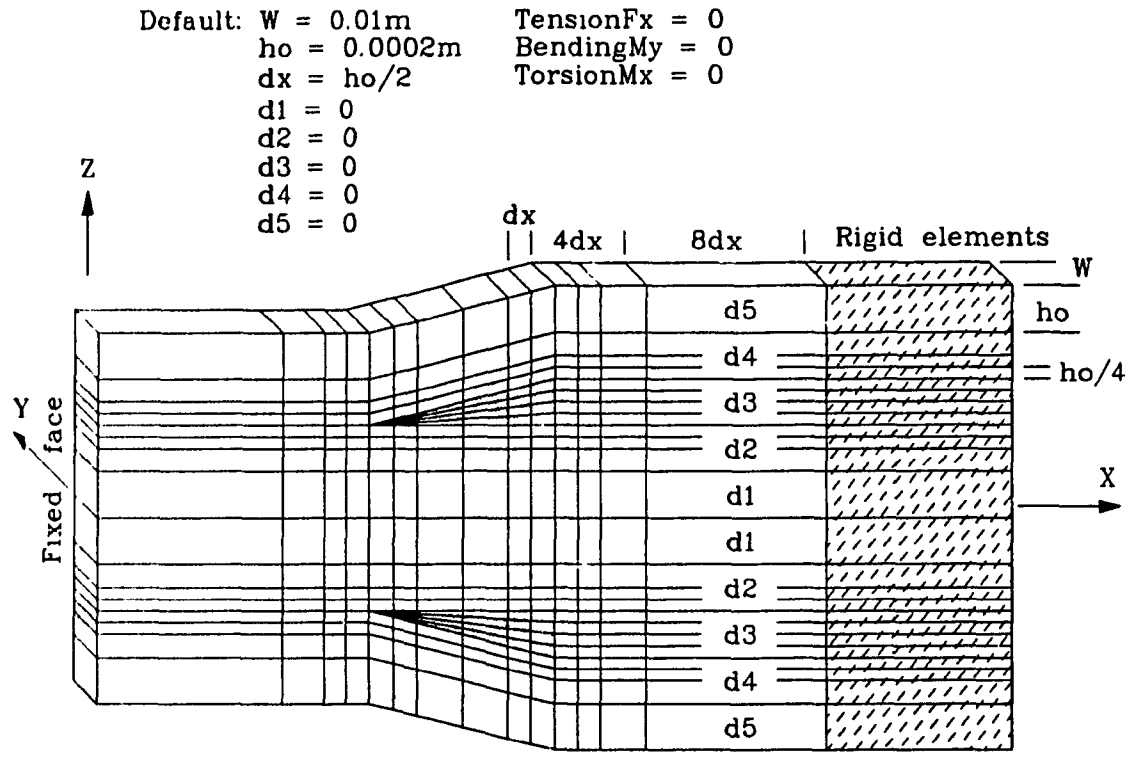


FIGURE 5.24

Crack Growth Mesh

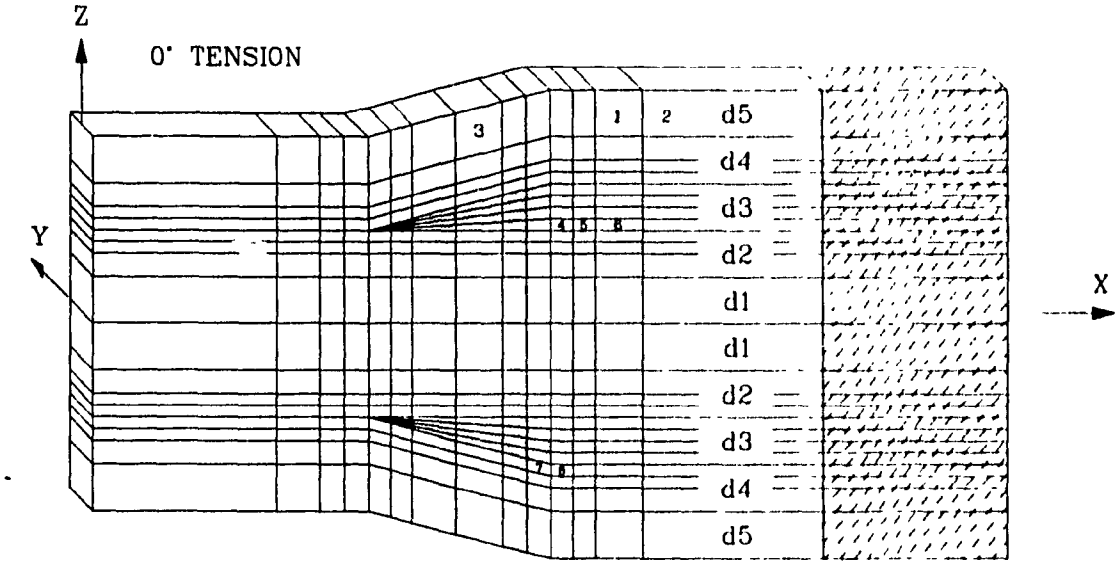


FIGURE 5.25

Crack Growth under Tension, all 0°

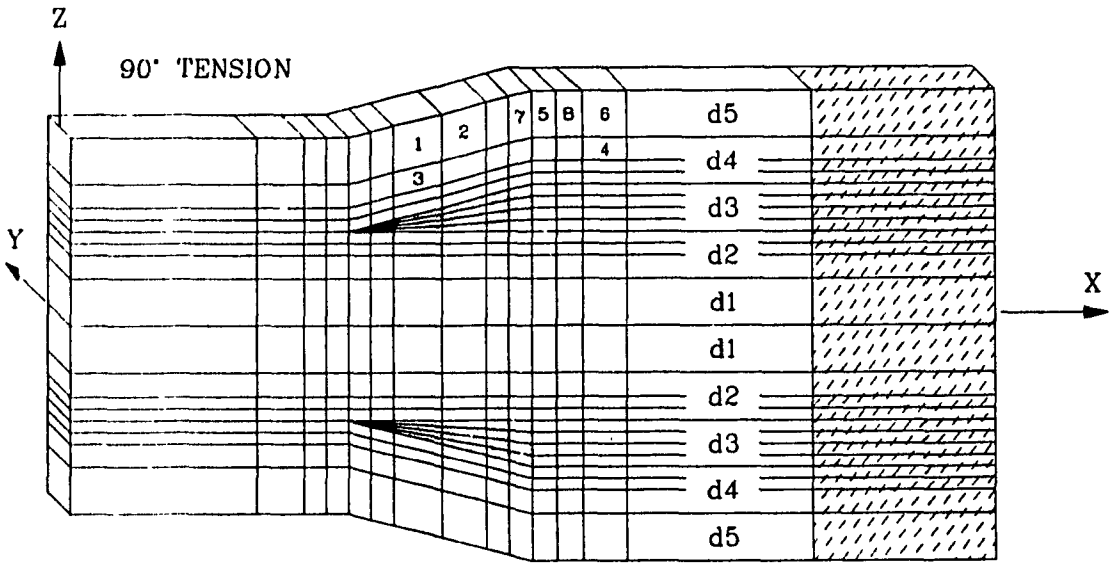


FIGURE 5.26

Crack Growth under Tension, all 90°

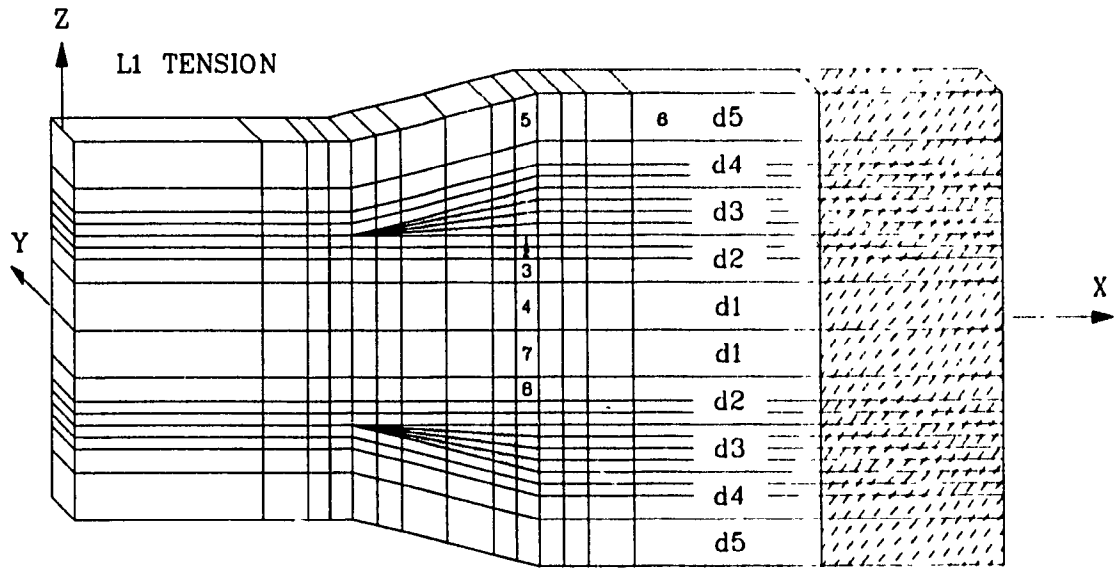


FIGURE 5.27

Crack Growth under Tension, $[0_2/45/-45/90]_S$
(d5=d4=0°, d3=45°, d2=-45°, d1=90°)

TABLE 5.15

Safety Factor and Failure Mode
Corresponding to Elements Removed Under Tension

Element Removed	Laminate Layup		
	[0]	[90]	$[0_2/45/-45/90]_S$
1	0.787 tension 2	0.082 tension 2	0.583 shear 1-2
2	0.100 shear 1-2	0.060 shear 1-3	0.418 tension 2
3	0.777 shear 1-3	0.055 tension 2	0.418 tension 2
4	0.923 shear 1-3	0.009 tension 3	0.260 tension 2
5	0.725 shear 1-3	0.061 tension 3	0.088 tension 3
6	0.751 shear 1-3	0.029 shear 1-2	0.167 shear 1-2
7	1.053 shear 1-3	93E-9 tension 2	0.246 tension 2
8	0.954 shear 1-3	0.017 tension 3	0.267 tension 2

The most interesting portion of this simulation lies in the 0° layup for elements 4 to 6. These removed elements show a continuity in the crack direction, failure mode and safety factor. Such a continuity indicates a possible crack growth.

Figures 5.28 to 5.30 show the evolution of the safety factor for the removed elements in the three layups. The safety factor of these elements was expected to decrease as elements were removed from the geometry. But these figures show that the safety factor can increase as more elements are removed. This behavior is also found in composite structures where the strength can increase after a crack has been observed. Therefore, cracks in a composite structure do not mean that the structure is weakened.

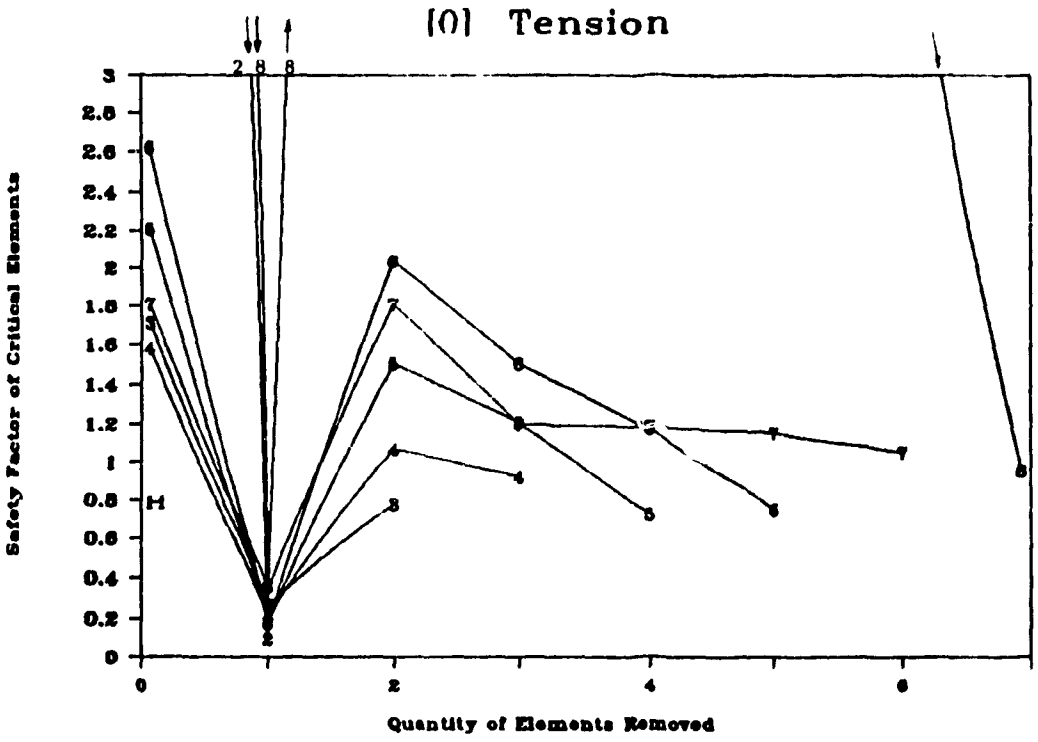


FIGURE 5.28

Safety Factor Variation in Elements to be Removed for all 0° Laminate under Tension

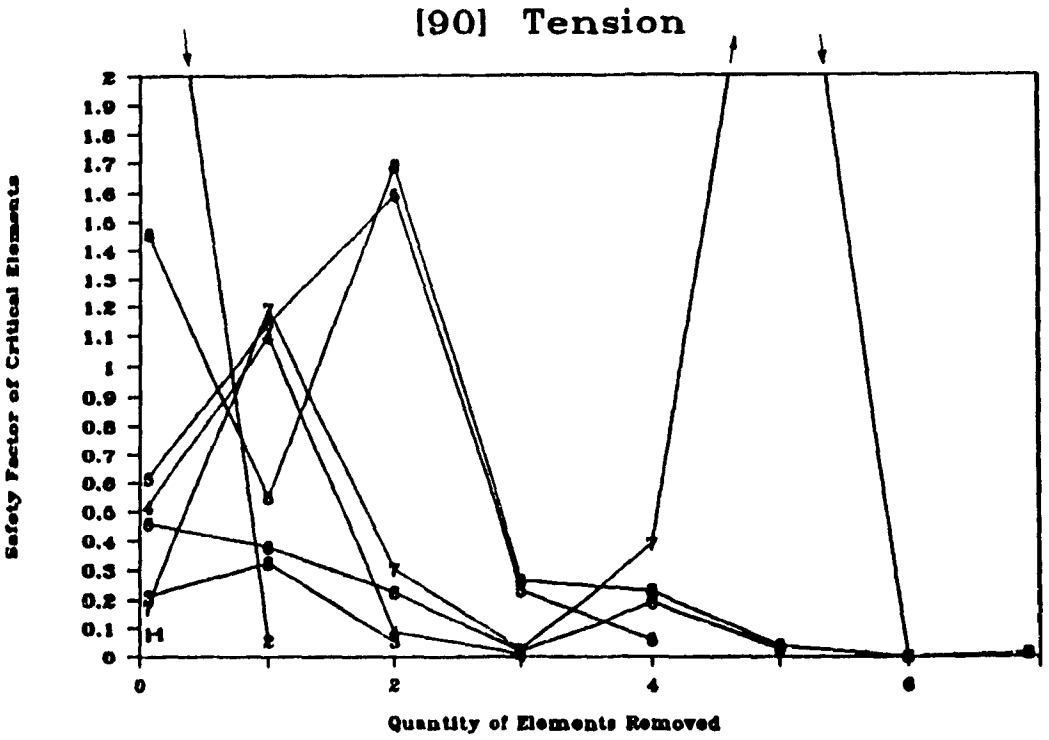


FIGURE 5.29

Safety Factor Variation in Elements to be Removed for all 90° Laminate under Tension

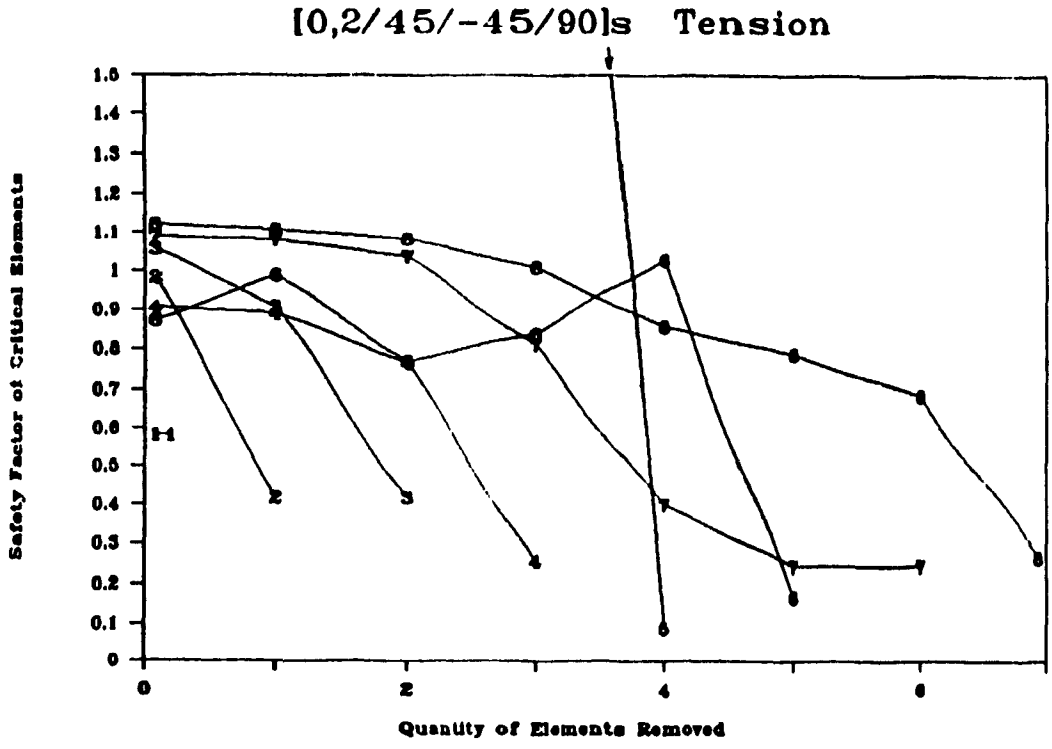


FIGURE 5.30

Safety Factor Variation in Elements to be Removed for $[0_2/45/-45/90]_S$ Laminate under Tension

5.10.2 Crack Growth under Bending

A bending moment (positive y) of 1000 Nm is imposed. The consecutive removal of elements is represented in Figures 5.31 to 5.33 for the three different layups. Table 5.16 presents the safety factor of the layup for a given number of removed elements.

TABLE 5.16
Safety Factor and Failure Mode
Corresponding to Elements Removed Under Bending

Element Removed	Laminate Layup		
	[0]	[90]	$[0_2/45/-45/90]_S$
1	25E-6 tension 2	89E-6 tension 2	39E-5 tension 2
2	21E-6 shear 1-2	45E-6 tension 2	32E-5 shear 1-2
3	11E-5 tension 3	33E-6 tension 2	58E-6 tension 3
4	17E-6 tension 3	27E-6 tension 2	37E-7 tension 3
5	30E-6 tension 2	25E-9 tension 3	20E-5 shear 1-2
6	15E-5 shear 1-2	54E-6 tension 2	10E-5 shear 1-3
7	20E-6 tension 2	49E-6 tension 2	49E-6 shear 1-3
8	15E-5 shear 1-2	52E-6 tension 2	93E-8 tension 3

Elements 1 to 4 and 6 to 8 of the 90° layup have continuity of crack direction, failure mode and safety factor. This sequence is a probable crack growth.

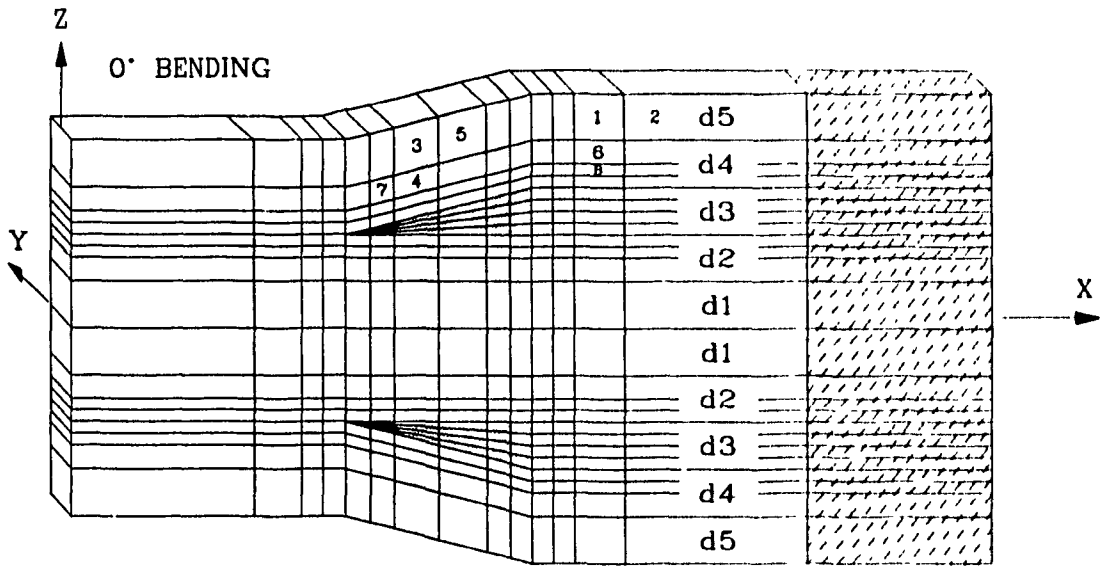


FIGURE 5.31

Crack Growth under Bending, all 0°

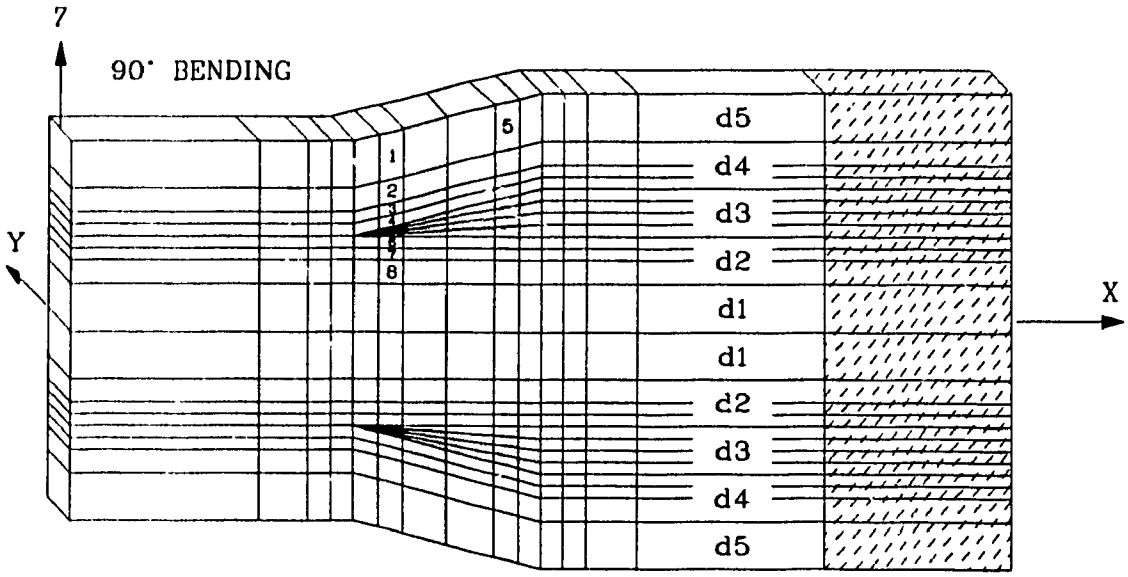


FIGURE 5.32

Crack Growth under Bending, all 90°

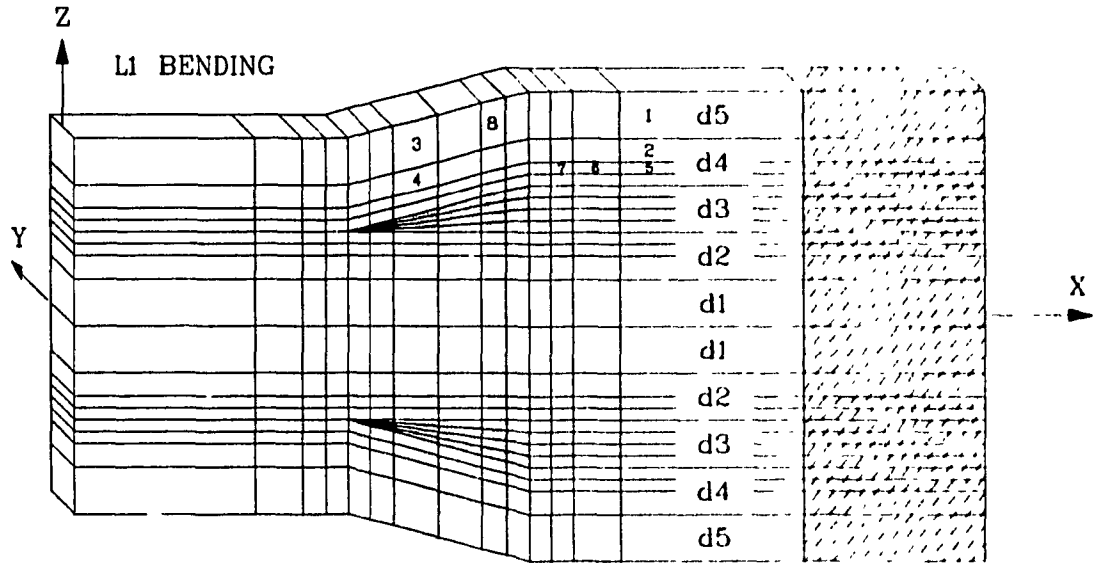


FIGURE 5.33

Crack Growth under Bending, $[0_2/45/-45/90]_S$
(d5=d4=0°, d3=45°, d2=-45°, d1=90°)

Although no crack growth was performed experimentally, the likeliness of a crack growth increases as there is a continuity in crack direction, failure mode and safety factor. Such continuity is often found in real crack propagations.

Figures 5.34 to 5.36 show the evolution of the safety factor for the removed elements in the three layups. As was found under a tension load, these figures also show that the safety factor can increase as more elements are removed under a bending load.

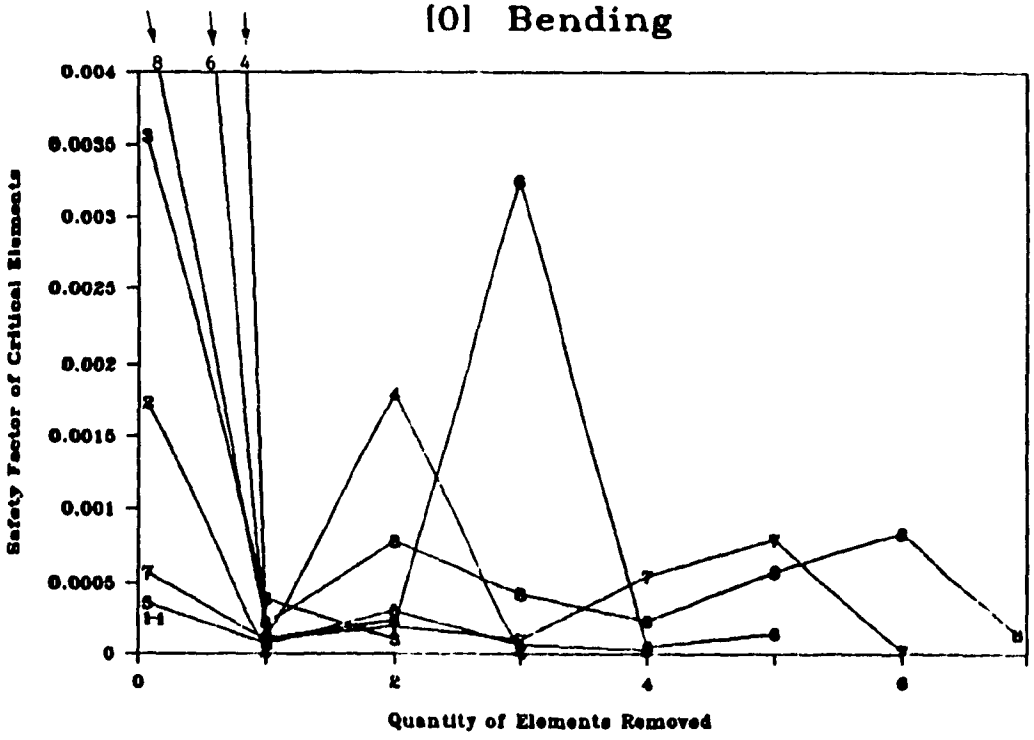


FIGURE 5.34

Safety Factor Variation in Elements to be Removed for all 0° Laminate under Bending

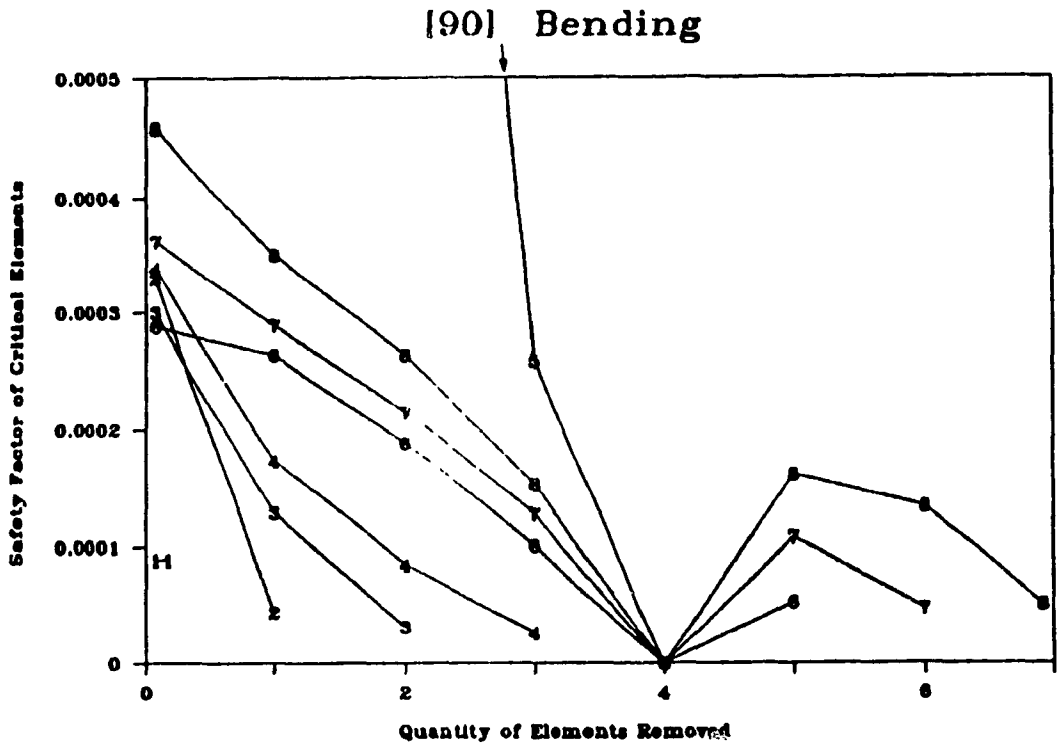


FIGURE 5.35

Safety Factor Variation in Elements to be Removed for all 90° Laminate under Bending

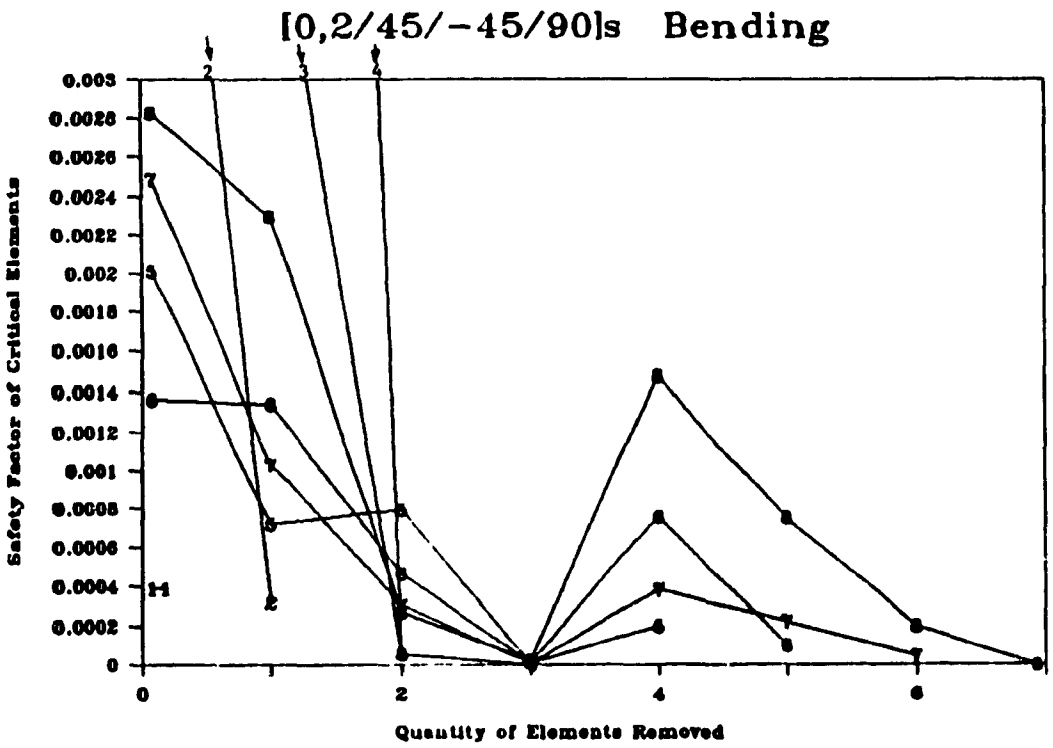


FIGURE 5.36

Safety Factor Variation in Elements to be Removed for $[0_2/45/-45/90]_S$ Laminate under Bending

5.11 Summary

From the results presented in this chapter, the following can be concluded:

- Any internal drop-off configuration is more efficient than an external drop-off configuration under all loading conditions by a ratio of about 2 to 1 or more.
- The presence of a drop-off does not affect torsional strength.
- Considering the lowest efficiency number under all loading conditions, a drop-off located halfway between mid-plane and top or bottom of the laminate provides the best performance (68% efficiency).
- When drop-off wetness is imperfect, material continuity in the direction of the critical stress at the failure location is the key factor.
- Extending the length of the fiber-free wedge at the drop-off reduces interlaminar stresses.
- A higher W/hmin ratio is beneficial for tapered laminates under tension and bending.
- No particular location, besides the general drop-off region, can be identified as the failure point.
- 45° fiber orientation is the best compromise for efficiency under tension and bending providing at worst 93% efficiency
- Crack growth simulation by consecutive removal of failed elements provides a graphical method of showing crack growth. A

simulation should be verified by a continuity of crack direction, failure mode and safety factor. These simulations show that laminate strength can increase as cracks occur.

CHAPTER 6

PROGRAM FEATURES

6.1 Introduction

Due to the amount of work involved in the finite element program (26000 lines) and for the purpose of presenting this program to potential users, this chapter describes its features.

Until recently, microcomputers have not been thought as having the necessary processing power for finite element programs. When used, they provided the user with a limited geometry size and number of degrees of freedom.

Despite inherent disadvantages (low speed and memory size), the benefits (working environment, portability) of implementing a finite element program on a microcomputer prevail. Many companies who can benefit from this program do not have to have access to large computers. Large computers can have considerable amounts of wait time inside a program execution which can produce an overall speed which is slower than that of a microcomputer. The disadvantages vanish as faster microcomputers are released. Current 80386 based computers compare in speed with most minicomputers available in universities. Intel™, major manufacturer of computer boards, announced the completion of 80486 based processors (released in summer of 1989) and currently works

on 80586 processor. Each processor is 3 times faster than the previous, indicating that current minicomputers will be surpassed by microcomputers within a couple of years. The choice gets even clearer when comparing available compilers and languages on microcomputers with those found on mainframe computers.

Having chosen the microcomputer option, the *PASCAL* language was chosen over *BASIC*, *FORTRAN* and *C* for its clarity, structure and available compiler features. *Turbo Pascal*™ version 5.0 was the selected compiler for its execution speed, unlimited code size and spectrum of available functions.

One example of the use of a microcomputer in finite elements was given by Chen and Yang (1985) who developed an anisotropic symmetrical laminated beam finite element including the effect of shear. These shell type elements have 2 nodes with 6 degrees of freedom each. These elements can not be stacked to form a real three dimensional geometry. Solution is obtained by using a banded matrix solver. The program was written in the BASIC language.

The present computer program, named *DIRECT*® (Appendix B), will go beyond these previous capabilities, due to a new submatrix manipulation technique and keeping the most active part of the equations in computer memory allowing for a flexible limit of the wavefront. This program will also feature a designed preprocessor language (Appendix C) capable of handling variables.

Emphasis was put on user features and simplicity. Many programs can claim a few features, but the integration level of all the features in this program is a strong point.

6.2 Variable Handleability

One of the goals of this thesis was to study the effect of some parameters on interlaminar stresses. Over 300 finite element cases were generated, solved and analyzed in order to study the effect of different meshes, materials or fiber angles. Generating a new mesh for each case was impractical. In order to study the effect of a parameter it should be defined as a variable in the geometry definition. This could have been solved by making a specific program to generate a mesh depending on a set of chosen variables. However a new program would have to be written for each different geometry when the variables can not account for the new geometry requirement.

The appropriate solution was to develop a preprocessing language similar to others like *ANSYS*® (v4.3 1987) with a capability of handling variables like a program would (Appendix C). Mathematical operations can be done on variables to declare new ones. Variables can also be passed to the file where the geometry code is written. This last feature allows the complete process of generating a geometry, solving and finding the failure

location to act as a loop for optimizing a variable.

Creating a preprocessing language required extensive work in:

- building procedures to interpret commands
- incorporating complete error detection messages
- calling material properties from a database
- compacting the element and node numbers while remembering original numbering for solution output
- working in rectangular, cylindrical and spherical local coordinate systems
- managing variables declarations

There is no practical limit to the number of nodes or element thanks to self-expanding file procedures which simply expand file size on demand for additional information storage. The preprocessing language is shown in Appendix C and the interpreter is part of PREP.pas (Appendix B).

6.3 Submatrix Handling Technique

Limited memory size is a problem which leads to others. Current microcomputers have 640 kB to 1MB of RAM (random access memory) available minus memory already used for loading the program or other utilities. The program will use all available memory left. Very often, this is insufficient. The user could purchase more memory or use the commonly found disk storage space

on a hard disk capable of handling much more memory storage (20 MB or more). However, access, storage and retrieval of information on a hard disk is slow compared to RAM memory.

One obvious solution is to keep the most often used part of the system of equations in RAM and keep the rest on disk storage. At the same time, it would be ideal to minimize the amount of information exchange between the disk and RAM storage. Here is where the choice of an adequate solving technique occurs.

The front solver technique [Irons and Ahmad (1980)] is suitable for the task by minimizing the size of the matrix considered for solving. The front solver technique also takes care or reusing rows and columns of the matrix which have been freed, keeping matrix size to a minimum and keeping to most active part of the matrix as the left upper corner, hence minimizing information exchange between RAM and disk storage.

The front solver technique is summarized in the following manner: As element stiffness matrices are added to the global stiffness matrix, some nodes will not be used any more by elements stiffness matrices to be added. A partial solving is done for such a node in order to extract its related equations from the global stiffness matrix. Those equations can be stored for later use in back-substitution. The rows and columns in contact with the removed node location can be freed and reset to zero so that

an other node can use the same space once occupied.

We now have a technique which minimizes the amount of memory needed and keeps the most often used part of the global stiffness matrix in RAM. Still, the amount of memory is often insufficient and the disk storage media must still be used for that extra memory needed. The goal is to minimize the number of times which is needed to read and write to such a device, thus reducing the amount of slow accesses.

To reduce the amount of disk access, a submatrix manipulation technique has been used. A submatrix size is defined by the number of dimensions of the problem at hand. Considering a three dimensional problem and three degrees of freedom per node, submatrices are 3 by 3 matrices. This technique uses such submatrices as single pieces of information, allowing to solve equations in any dimension with great ease. The program was then generalized to solve a problem of any number of dimensions (1,2,3 or more).

Example:

A simple example is given, representing a three dimensional problem of 2 nodes. The following would constitute the augmented matrix (matrix of equation coefficients + equation constants) [Burden and Faires (1985)] which we must prepare for back

substitution:

$$\left\{ \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \right. \qquad (6.1)$$
$$\left. \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \right\}$$

The augmented matrix is treated as if it contains only single elements of [a], [b], [c], [d], {e} and {f}:

$$\left\{ \begin{matrix} [a] & [b] & \{e\} \\ [c] & [d] & \{f\} \end{matrix} \right. \qquad (6.2)$$

The first step is to upper triangularize the first submatrix [a]. The multipliers needed to do this are calculated from submatrix [a] like a normal gaussian elimination. It is important to remember the multipliers used for later use. Submatrix [a] is a pivot submatrix.

To reduce the 2^{nd} row of submatrix [a], multiply the 1^{st} row of [a] by $\frac{a_{21}}{a_{11}}$ and subtract the result from the 2^{nd} row. The result of the subtraction is stored in the 2^{nd} row. To reduce the 3^{rd} row of [a], perform the similar operation. The multipliers $\begin{bmatrix} \frac{a_{21}}{a_{11}} & \frac{a_{31}}{a_{11}} \end{bmatrix}$ are kept in the 1^{st} column of a matrix [m]. The modified submatrix [a] now is

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} - \frac{a_{12}a_{21}}{a_{11}} & a_{23} - \frac{a_{13}a_{21}}{a_{11}} \\ 0 & a_{32} - \frac{a_{12}a_{31}}{a_{11}} & a_{33} - \frac{a_{13}a_{31}}{a_{11}} \end{bmatrix}$$

To reduce the 3^{rd} row of the modified [a] submatrix, multiply the 2^{nd} row by

$$\frac{a_{32} - \frac{a_{12}a_{31}}{a_{11}}}{a_{22} - \frac{a_{12}a_{21}}{a_{11}}}$$

and subtract the result from the 3^{rd} row. The new multipliers are stored in position (3,2) of a matrix [m]:

$$[m] = \begin{bmatrix} 0 & 0 & 0 \\ a_{21}/a_{11} & 0 & 0 \\ a_{31}/a_{11} & \frac{a_{32}-a_{12}m_{31}}{a_{22}-a_{12}m_{21}} & 0 \end{bmatrix} \qquad (6.3)$$

where

$$m_{31} = a_{31}/a_{11}$$

$$m_{21} = a_{21}/a_{11}$$

The matrix [m] therefore contains the multipliers needed to upper triangularize [a]. Submatrix [a] now is upper triangularized as:

$$[a]' = \begin{bmatrix} a'_{11} & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a'_{33} \end{bmatrix} \qquad (6.4)$$

The same multiplier matrix [m] can be used on submatrix [b] and subvector {e}. By using [a]' and [c], a similar multiplier submatrix is calculated for the second submatrix row (rows 4,5 and 6) of Equation (6.1):

$$[m]' = \begin{bmatrix} \frac{c_{11}}{a'_{11}} & \frac{c_{12}-m'_{11}a'_{12}}{a'_{22}} & \frac{c_{13}-m'_{11}a'_{13}-m'_{12}a'_{23}}{a'_{33}} \\ \frac{c_{21}}{a'_{11}} & \frac{c_{22}-m'_{21}a'_{12}}{a'_{22}} & \frac{c_{23}-m'_{21}a'_{13}-m'_{22}a'_{23}}{a'_{33}} \\ \frac{c_{31}}{a'_{11}} & \frac{c_{32}-m'_{31}a'_{12}}{a'_{22}} & \frac{c_{33}-m'_{31}a'_{13}-m'_{32}a'_{23}}{a'_{33}} \end{bmatrix} \qquad (6.5)$$

This multiplier matrix [m]' along with modified submatrix [b]' is then used to modify submatrix [d] into [d]'. A similar operation is done to subvector {f}. At this point we have:

$$\begin{cases} \begin{bmatrix} a'_{11} & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a'_{33} \end{bmatrix} \begin{bmatrix} b'_{11} & b'_{12} & b'_{13} \\ b'_{21} & b'_{22} & b'_{23} \\ b'_{31} & b'_{32} & b'_{33} \end{bmatrix} \begin{bmatrix} e'_1 \\ e'_2 \\ e'_3 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d'_{11} & d'_{12} & d'_{13} \\ d'_{21} & d'_{22} & d'_{23} \\ d'_{31} & d'_{32} & d'_{33} \end{bmatrix} \begin{bmatrix} f'_1 \\ f'_2 \\ f'_3 \end{bmatrix} \end{cases} \qquad (6.6)$$

On a submatrix level we have:

$$\begin{cases} [a]' & [b]' & \{e\}' \\ 0 & [d]' & \{f\}' \end{cases} \qquad (6.7)$$

To complete this example we have to find the multipliers for pivot submatrix [d]' in a similar way as was done for pivot submatrix [a] at the start and then reduce subvector {f}'.

By defining pivot submatrices, multiplier submatrices and subvectors we achieve similar operation at this level as if the submatrices were single numbers Because the submatrices have to be read and stored in less times than the case where all individual elements of the submatrices are read and stored for each internal row reduction, we can minimize the time spent reading and storing from slow access and transfer rate media.

Assuming 400 kB of RAM, double precision real numbers, the number of submatrices kept in RAM is:

$$\text{Submatrix quantity} = 400(1024 \text{ bytes})/[3\text{x}3(8 \text{ bytes})]$$
$$= 5688 \qquad (6.8)$$

Due to a symmetric global stiffness matrix and in order for the submatrices to be kept in RAM closest to the most active corner of the global stiffness matrix, the submatrices are numbered in the following manner:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 7 \\ & 6 & 8 \\ & & 9 \end{bmatrix} \begin{array}{l} \leftarrow \text{row 0 (used as storage buffer)} \\ \leftarrow \text{row 1} \\ \leftarrow \text{row 2} \\ \leftarrow \text{row 3} \end{array} \qquad (6.9)$$

A small procedure can determine the submatrix number from a given row and column:

If Row=0 then SubmatrixNumber=Column
If Row>0 then SubmatrixNumber=MaximumFrontSize + Row

$$+ \frac{(\text{Column}-1)}{2} \text{Column} \qquad (6.10)$$

Submatrix numbers over the allowable quantity are simply kept on disk media. Using 30 MB of combined RAM and disk storage allows for approximately 2800 degrees of freedom in the wavefront alone. This is implemented in SOLVE.pas (Appendix B).

6.4 Node and Element Number Compacting

DIRECT® will perform node and element renumbering for the following reasons:

- Reduces the size of the files generated by the node and element information. Unused nodes and elements are deleted to avoid retaining unnecessary storage space.
- Simplifies programming for the solving process.
- The user of the program is not obliged to provide a continuous numbering of nodes or elements.
- A node or an element can be deleted.

The time spent renumbering can take a few minutes on a slow computer. In order to determine if a node is used, a counter for each node is incremented each time it is used by an element. This will also serve in later stress averaging routines which need to know the number of times a node will serve in different elements. The nodes which were not flagged are deleted. All information

related to the nodes is adjusted. Although node and element numbering is compacted, an index is established so that the original numbers are returned with any results, eliminating confusion. This process is implemented in PREP.pas (Appendix B).

6.5 Removal of Failed Elements

Element removal is used for crack propagation simulation. Once the displacement solution is obtained for all the nodes of a given problem, the stress along the fiber direction can be evaluated at each node of each element. By comparing these stresses with the maximum stress allowed (uniaxial strength criterion) as given in the database for the material of the element, the highest ratio of stress/strength is found throughout the geometry. The program returns the worst element number, material, worst node, mode of failure (stress direction) and safety factor which is the inverse of the previous ratio.

When the worst element is found, it is removed from the initial geometry given to be solved. Because the preprocessor allows for discontinuous element and node numbering, the removal of the element becomes an added command line in the file which created the initial geometry. The program appends this new command to the existing file and can repeat the whole process over as long as desired. This is done by the EDEL preprocessing

command (Appendix C) and implemented in PREP.pas (Appendix B).

6.6 Mesh Refinement Automation

In order to accurately calculate stress at high stress gradient locations, it is necessary to take a segment out of the structure and refine its mesh in that segment.

A mesh refinement illustration is given in Figures 4.18 and 4.19. The bold contour line in the first mesh became the outer limits of the second. The displacements obtained from the contour in the first mesh were then transposed to the matching nodes in the second. The displacements for the nodes left on the external boundary of the second mesh were interpolated from the known displacement nodes.

The process of transferring node coordinates and displacement solution of the nodes from the coarse mesh is simple but painstaking, let alone the human error factor. This process was automated by two preprocessing commands: OLD and INTER (Appendix C).

6.7 Stress Landscapes

One of the postprocessing feature is used to plot the strain or stress in material, element or global coordinates, force or displacement within a region of the geometry. Plotting process: A flat plane R is fitted by regression to all the chosen nodes. All selected nodes are projected on plane R. A new coordinate system with X' and Y' in plane R is established in such a manner that Y' is in the plane formed by the normal N of plane R and Z. The positive Y' axis is selected such that the angle between Z and Y' is less than 90°. The direction of X' is determined from the right hand coordinate X' Y' N system.

The results are viewed as a landscape elevation in X' Y' Z' (Z'=N) (Figure 4.24). X' and Y' represent the selected plane while Z' coordinate gives the values of the selected output. Continuous lines like $\overline{AB}$, $\overline{CD}$, $\overline{EF}$, $\overline{ACEGIKM}$ and $\overline{BDFHJLN}$ show the connections between values at the nodes. The connections are made in a similar manner as the node connections to form the elements. Any particular viewpoint can be specified for visual inspection of the landscape. A hard copy can be performed. Geometries are viewed from an infinitely distant point to eliminate perspective and leave an undistorted shape. All the selection data is stored for next view time. All plots are automatically scaled to fit the screen without axis distortion.

Positive stresses are seen in Figure 4.24 as lines rise from an "invisible" plane where the stress is zero. Plane A'B'N'M' in

Figure 4.24 is such a plane. A positive stress is characterized by a line rising upward like in Figure 4.24. The reverse is true for negative stresses, like the left side of Figure 4.27. This process was implemented in RESULT.pas (Appendix B).

6.8 Solution Superposition

Because the element formulation is done under linear elastic material behavior, solutions of individual loading conditions can be summed to give the same result as if the loading conditions were imposed together and the case was solved afterwards.

Proportionally summing the results from different loading conditions while using the same geometry is a feature incorporated in this program. The solution of a separate loading condition can be proportionally modified before being added to other solutions. Summing previous results is done within a few minutes on the microcomputer instead of hours. This can be done by the batch command SUM which is the executable version of SUM.pas (Appendix B).

6.9 Other Features

The program developed is not a simple summation of different features which do not interact. Efforts were put into creating an integrated environment. Here are some other features which have

not been mentioned before:

- Batch or interactive mode selection.
- Any word processor is called from within the program to edit the files defining the geometry.
- Memory buffers are used to reduce disk access time.
- Results can be sent to: printer, Lotus 1-2-3™, screen.
- Extremes in results are found (maximum, minimum) along with average and standard deviation.
- A complete help and tutorial is available from the program which describes the preprocessing language with examples and figures.
- After preprocessing, information is available upon the geometry generated: number of nodes used, exact number of degrees of freedom, coordinate limits...
- Viewing the geometry is offered.
- An exaggerated deformed body shape can be viewed from any angle.
- Finding the safety factor, mode of failure, failure location can be performed.
- A case manager can store a case under study to start a new one and later return to the previous case where it was left.
- Before the solving process starts, disk and memory size are compared to the required space needed for solving.
- The element library can be expanded.
- Different element types can be used within the same geometry.
- A material property database is incorporated. Material properties can be changed. Material can be added or removed from the database. The database simplifies the information

transfer of the material properties and permits material optimization.

- Whenever possible, displacements are kept in RAM.
- A regression program allows the edition of formulas to fit data. The formula can have 127 terms of any power.
- Many configurations are allowed for directories and RAM drives. An execution profile timer is given at the end of the solving process to help find the best configuration.
- Stress and strain along material, element and global direction can be calculated simultaneously or separately to save disk storage space.
- 16 digits accuracy is used in all calculations.
- Any math coprocessor chip can be used: 8087, 80287, 80387.
- Any graphics card is acceptable: CGA, MCGA, EGA, VGA, Hercules, AT&T 6300 and IBM 3270 PC.
- Alongside the main menu choices, a status of completion is given. The sequence of analysis is foolproof by checking process dependency.
- Many cases can be derived from a single preprocessing file by passing new variables and renaming the case name.

6.10 Limitations and Requirements

Although the program has no inherent limitations, the size of the problem can be limited by disk storage space. For example, a three dimensional geometry with 600 degrees of freedom in the

wave front and 10000 in total, can require a maximum of 50 MB.

Minimal requirements:

- Any IBM PS/2™, IBM® or compatible
- Any math coprocessor: 8087/80287/80387
- 1 hard disk drive, 1 floppy disk drive
- *Turbo Pascal*™ 5.0 for its graphic drivers (*.BGI)
- Any version of DOS
- Any graphic card
- 256 kB of RAM memory
- Any word processor or editor

Suggested additions:

- Enough memory for a RAM drive or disk caching software
- Printer
- High speed 40 MB hard disk or larger

6.11 Summary

Medium to large size problems can be analyzed with this program: 1000 elements with 60 degrees of freedom per element. The program provides a good base for developing new elements as the element library can be expanded. Human manipulation of data is minimized. A submatrix handling technique allows to easily work in any dimension and reduced storage media access time.

Variables can be used within a preprocessing language allowing for the study of the effect of any parameter: dimensions, fiber angle and materia.. Integration of features, graphing capabilities and portability of the program make it a useful tool for analyzing interlaminar stresses in composite laminates.

CHAPTER 7

CONCLUSIONS, CONTRIBUTIONS AND RECOMMENDATIONS FOR FUTURE WORK

7.1 Conclusions

Closed Form Solution

Tapered laminates with internal drop-offs were studied for the first time. Analytical functions have been developed for two dimensional stress distributions around a triangular hole of a given Length/Height ratio in a plate made of anisotropic material subjected to a loading at any angle (Figure 2.5). The radius of curvature at the summits of the triangle can be varied. Reasonable agreement was obtained between a finite element analysis and closed form solution results. Only the finite element method can handle the case where the fibers follow the contour of the drop-off.

Program Developed

An extensive finite element program (26000 lines), named *DIRECT*®, has been developed on a microcomputer in PASCAL. The displacement formulation was used. The major difficulties lay in creating a program capable of handling a large quantity of elements of 60 degrees of freedom per element with the limitations

of a microcomputer's speed and memory. The program provides accessibility and portability and constitutes one important tool for future work in composite structures.

A complete preprocessing language was developed to handle variables (appendix C) for studying the effect of parameters. The complete process of generating a mesh, solving and finding the failure location with possible element removal, can be set as a loop.

Using a uniaxial strength criterion on the finite element solutions, tapered laminates were found to fail due to interlaminar stresses arising from the drop-offs. The stress disturbance caused by the drop-off is confined to approximatively 1 drop-off length of the drop-off in the laminate axis.

Laminate Behavior

Many parameters were studied for their effect upon the efficiency (strength of a tapered laminate divided by the strength of a reference untapered laminate) of tapered laminates. The following presents the main results:

- Any internal drop-off configuration is more efficient than an external drop-off configuration under all loading conditions by a ratio of about 2 to 1 or more.

- The presence of a drop-off does not affect torsional strength.
- Considering the lowest efficiency number under all loading conditions, a drop-off located halfway between mid-plane and top or bottom of the laminate provides the best performance (68% efficiency).
- When drop-off wetness is imperfect, material continuity in the direction of the critical stress at the failure location is the key factor.
- Extending the length of the fiber-free wedge at the drop-off reduces interlaminar stresses.
- A higher W/hmin ratio is beneficial for tapered laminates under tension and bending.
- No particular location, besides the general drop-off region, can be identified as the failure point.
- 45° fiber orientation is the best compromise for efficiency under tension and bending providing at worst 93% efficiency

Crack growth simulation by consecutive removal of failed elements provided a graphical method for showing crack growth, but the method has not been verified with experiments. In the simulations performed, crack growth direction and failure mode often changed. Continuity of those two parameters are an indication of the validity of this simulation. These simulations show that laminate strength can increase as cracks occur.

7.2 Contributions

Closed Form Solution

Previous two-dimensional closed form solution for stress around a triangular hole has been extended to include non-equilateral triangular holes. Features of previous solutions have been combined (see Table 2.8):

- Tension load in any direction
- Triangular shape varying from a circle to a sharp triangle
- Stress solution beyond boundary of the hole
- Anisotropic material

A general method of solution for higher order triangular shape functions using a numerical technique was developed through the control of the roots of a polynomial with complex coefficients.

Program Developed

Although most of the element's formulation was previously done, new transformations were developed: stress extrapolation matrix and compactness of displacement to strain transformation matrix.

A preprocessing language capable of handling variables has been created to offer parameter optimization through the finite element method. Designing the finite element program for microcomputers has been a significant improvement in accessibility and portability particularly for small companies. The level of integration of the features of the program is a new step in ease of use. Using the submatrix handling technique, the program also breaks new ground in the number of degrees of freedom for a problem using a microcomputer.

Laminate Behavior

New knowledge has been generated for understanding the state of stress in tapered laminates. A crack growth simulation has been developed for composites.

7.3 Recommendations for future work

Extending the closed form solution to three dimensional stress distributions would be an improvement.

Obviously, not every parameter which can affect the strength of tapered laminates through interlaminar stresses has been

studied. The program developed could be enhanced with elements providing higher convergence rates in order to reduce mesh size dependency of the results. A 32 node element could easily be added to the element library.

Crack growth simulation techniques can be enhanced from the simple removal of failed elements, to edge or face separation depending upon failure mode. These simulations should be compared to experimental crack growth patterns.

REFERENCES

1. Turbo PASCAL version 5.0, Borland International, California, 1988.

2. BAR-YOSEPH, P., AVRASHI, J., "Interlaminar Stress Analysis for Laminated Plates Containing a Curvilinear Hole", Composite Structures, Vol. 21, No. 5, 1985, pp. 917-932.

3. BAR-YOSEPH, P., SITON, G., "Effect of Material Non-linearity on the Interlaminar Stress Field in Composite Laminates", Composite Structures, Vol. 21, No. 6, 1985, pp. 1105-1118.

4. BARKER, R.M., MacLAUGHLIN, T.F., "Stress Concentrations Near a Discontinuity in Fibrous Composites", Journal of composite Materials, Vol. 5, Oct., 1971, pp. 492-503.

5. BURDEN, R.L., FAIRES, J.D., "Numerical Analysis", Third Edition, Prindle, Weber & Schmidt Publication, Youngstown State University, 1985.

6. BURNETT, D.S., "Finite Element Analysis, From Concepts to Applications", Addison-Wesley, 1987.

7. CARLSON, L., "Interlaminar Stresses at a Hole in a Composite Member Subjected to In-plane Loading", Journal of Composite Materials, Vol. 17, May, 1983, pp. 238-249.

8. CARSWELL, W.S., "Fatigue of Notched Composites", Failure Modes in Composites IV, Vol. 4, 1979, pp. 92-104.

9. CHEN, A.T., YANG, T.Y., "Static and Dynamic Formulation of a Symmetrically Laminated Beam Finite Element for a Microcomputer", Journal of Composite Materials, Vol. 19,

Sept., 1985, pp. 459-475.

10. CHEN, J.K., SUN, C.T., "Nonlinear Analysis of Interlaminar Stresses in Graphite/Epoxy Laminates With and Without Delamination Cracks", Composite Structures, Vol. 8, No. 4, 1987, pp. 271-285.

11. CHURCHILL, R.V., "Complex Variables and Applications", McGraw-Hill, 1960.

12. CORTEN, H.T., "Fracture Mechanics of Composites", Fracture, an Advanced Treatise, Vol. 7, 1972, pp. 675-769.

13. CURISKIS, J.I., "On the Determination of the Non-linear Longitudinal Mechanical Behavior of Short-fiber Composites by a Three-dimensional Finite Element Procedure", Advances in Composite Materials, Proceedings of the Third International Conference on Composite Materials, Paris, France, August 26-29 1980, Published by Permagon Press, Oxford, England, Vol. 1, 26 August, 1980, pp. 796-811.

14. DAOUST, J., HOA, S.V., "HYBRID Version 5c", Concordia University, 1988.

15. DESALVO, G.J., GORMAN, R.W., "ANSYS PC Version 4.3", Swanson Analysis and Systems, Huston, 1987.

16. DONG, S.B., PISTER, K.S., TAYLOR, R.L., "On the Theory of Laminated Anisotropic Shells and Plates", Journal of Aerospace Sciences, Vol. 29, August, 1962, pp. 969-975.

17. DREGER, D.R., "Design Guidelines for Joining Advanced Composites", Fabrication of Composite Materials Source Book, 1985, pp. 9-13.

18. ENGBLOM, J.J., OCHOA, O.O., "Finite Element Formulation

Including Interlaminar Stress Calculations", Composite Structures, Vol. 23, No. 2, 1986, pp. 241-249.

19. GREGORY, M.A., HERAKOVICH, C.T., "Predicting Crack Growth Direction in Unidirectional Composites", Journal of Composite Materials, Vol. 20, Jan., 1986, pp. 67-85.

20. HAHN, H.T., TSAI, S.W., "On the Behavior of Composite Laminates After Initial Failures", Journal of Composite Materials, Vol. 8, July, 1974, pp. 288-305.

21. HERTZBERG, R.W., "Deformation and Fracture Mechanics of Engineering Materials", John Whiley & sons, New-York, Santa Barbara, London, Sydney, Toronto, 1976.

22. HOA, S.V., DAOUST, J., DU, B.L., VU-KHANH, T., "Interlaminar Stresses in Tapered Laminates", Polymer Composites, Vol. 9, No. 5, 1988.

23. HOA, S.V., YU, C.W., SANKAR, T.S., "Analysis of Filament Wound Vessel Using Finite Elements", Composite Structures, Vol. 3, No. 1, 1985, pp. 1-18.

24. IRONS, B., AHMAD, S., "Techniques of Finite Elements", Halsted Press, New York, 1980.

25. ISAKSON, G., LEVY, A., "Finite-element Analysis of Interlaminar Shear in Fibrous Composites", Journal of Composite Materials, Vol. 5, April, 1971, pp. 273-276.

26. JONES, R.M., "Mechanics of Composite Materials", McGraw-Hill, New-York, 1975.

27. KASSAPOGLOU, C., LAGACE, P.A., "Efficient Method for the Calculation of Interlaminar Stresses in Composite Materials", Journal of Applied Mechanics (Trans. ASME),

Vol. 53, No. 4, Dec., 1986, pp. 744-750.

28. KASSAPOGLOU, C., LAGACE, P.A., "Closed Form Solutions for the Interlaminar Stress Field in Angle-ply and Cross-ply Laminates", Journal of Composite Materials, Vol. 21, No. 4, April, 1987, pp. 292-308.

29. KRISHNA MURTY, A.V., VELLAICHAMY, S., "On Higher Order Shear Deformation Theory of Laminated Composite Panels", Composite Structures, Vol. 8, No. 4, 1987, pp. 247-270.

30. KULKARNI, S.V., STONE, R.G., TOLAND, R.H., "Prototype Development of an Optimized, Tapered - Thickness, Graphite / Epoxy Composite Flywheel", Lawrence Livermore Laboratory, University of California, Nov., 1978, pp. 1-37.

31. LAGACE, P.A., BREWER, J., KASSAPOGLOU, C., "Effect of Thickness on Interlaminar Stresses and Delamination in Straight-edged Laminates", Journal of Composite Technology, Vol. 9, No. 3, 1987, pp. 81-87.

32. LEKHNITSKII, S.G., "Anisotropic Plates", Gordon and Breach Science Publishers, New-York, 1968.

33. LUBIN, G., "Handbook of Composites", Van Nostrand Rheinhold, 1982.

34. LUCKING, W.M., HOA, S.V., SANKAR, T.S., "Effects of Geometry on Interlaminar Stresses of [0/90]s Composite Laminates with Circular Holes", Journal of Composite Materials, Vol. 18, No. 2, March, 1984, pp. 188-198.

35. MARKHAM, M.F., DAWSON, D., "Interlaminar Shear Strength of Fibre - Reinforced Composites", Composites, Vol. 6, No. 4,

July, 1975, pp. 173-176.

36. MUSKHELISHVILI, N.I., "Some basic problems of the mathematical theory of elasticity", P. Noordhoff Ltd, Netherlands, 1963.

37. MYERS, R.E., "Microcomputer Graphics for the IBM PC", Addison-Wesley, Ontario, Pensylvania State University, 1984.

38. PAGANO, N.J., "Stress Gradients in Laminated Composite Cylinders", Journal of Composite Materials, Vol. 5, April, 1971, pp. 260-265.

39. PAGANO, N.J., "On the Calculation of Interlaminar Normal Stress in Composite Laminate", Journal of Composite Materials, Vol. 8, Jan., 1974, pp. 65-81.

40. PETERSON, R.E., "Stress Concentration Factors", John Wiley & Sons, New-York, 1974.

41. PIAN, T.H.H., "Derivation of Element Stiffness by Assumed Stress Distributions", A.I.A.A. Jl., 1964, 2, 1333-6.

42. PIPES, R.B., DANIEL, I.M., "Moiré Analysis of the Interlaminar Shear Edge Effect in Laminated Composites", Journal of Composite Materials, Vol. 5, April, 1971, pp. 255-259.

43. PIPES, R.B., PAGANO, N.J., "Interlaminar Stresses in Composite Laminates Under Uniform Axial Extension", Journal of Composite Materials, Vol. 4, Oct., 1970, pp. 538-548.

44. PIPES, R.B., PAGANO, N.J., "Interlaminar Stresses in Composite Laminates - an Approximate Elasticity Solution", Journal of Applied Mechanics ASME, Vol. 41 ser E, No. 3,

Sept., 1974, pp. 668-672.

45. PONCIA, E.C., KRAUSS, T.A., STAAB, G.H., "New Tapered Composite Spar Design", Final Report, June, 1975, pp. 1-119.

46. PRYOR, C.W. Jr., BARKER, R.M., "Finite Element Analysis of Bending - Extensional Coupling in Laminated Composites", Journal of Composite Materials, Vol. 4, Oct., 1970, pp. 549-552.

47. PUPPO, A.H., EVENSEN, H.A., "Interlaminar Shear in Laminated Composites Under Generalized Plane Stress", Journal of Composite Materials, Vol. 4, April, 1970, pp. 204-220.

48. RIZZI, S.A., LEEWOOD, A.R., DOYLE, J.F., SUN, C.T., "Elastic-Plastic Analysis of Boron/Aluminum Composite Under Constrained Plasticity Conditions", Journal of Composite Materials, Vol. 21, No. 8, Aug., 1987, pp. 734-749.

49. ROSEN, B.W., DOW, N.F., "Mechanics of Failure of Fibrous Composites", Fracture, an Advanced Treatise, Vol. 7, 1972, pp. 611-674.

50. SAVIN, G.N., "Stress Concentration Around Holes", Pergamon Press, New York, 1961.

51. SENDECKYJ, G.P., "Surface Notches in Composites", Fracture of Composite Materials, 1982, pp. 115-127.

52. SIERAKOWSKI, R.L., EBCIOGLU, I.K., "On Interlaminar Shear Stresses in Composites", Journal of Composite Materials, Vol. 4, April, 1970, pp. 144-149.

53. SPIEGEL, M.R., "Formules et Tables de Mathématiques", Série Schaum, McGraw-Hill, New-York, 1981.

54. SPILKER, R.L., CHOU, S.C., ORRINGER, O., "Alternate Hybrid-Stress Elements for Analysis of Multilayer Composite Plates", Journal of Composite Materials, Vol. 11, Jan., 1977, pp. 51-70.

55. TANG S., "Interlaminar Stresses of Uniformly Loaded Rectangular Composite Plates", Journal of Composite Materials, Vol. 10, No. 1, Jan., 1976, pp. 69-78.

56. TIMOSHENKO, S.P., GOODIER, J.N., "Theory of Elasticity", Third Edition, McGraw-Hill, New-York, Engineering Societies Monographs, 1970.

57. TSAI, S.W., HAHN, H.T., "Introduction to Composite Materials", Technomics, Pennsylvania, 1980.

58. UENG, C.E., ZHANG, K.-D., "Simplified Approach for Interlaminar Stresses in Orthotropic Laminated Strips", Journal of Reinforced Plastics and Composites, Vol. 4, No. 3. July, 1985, pp. 273-286.

59. VONG, T.S., "New Variational Approach to the Delamination Problem", Composite Structures, Vol. 5, No. 4, 1986, pp. 245-250.

60. WHITCOMB, J.D., RAJU, I.S., "Superposition Method for Analysis of Free-Edge Stresses", Journal of Composite Materials, Vol. 17, Nov., 1983, pp. 492-507.

61. WHITNEY, J.M., "On the Use of Shell Theory for Determining Stresses in Composite Cylinders", Journal of Composite Materials, Vol. 5, April, 1971, pp. 340-353.

62. WHITNEY, J.M., HALPIN, J.C., "Analysis of Laminated Anisotropic Tubes Under Combined Loading", Journal of Composite Materials, Vol. 2, No. 3, July, 1968, pp. 360-367.

63. WOODBERRY, R.F.H., "Failure and Rupture Characteristics of Laid-up Composites", International Conference on Analytical and Testing Methodologies for Design with Advanced Materials (ATMAM 1987), session 2, paper 9, 26 August 1987.

64. WU, C.M.L., "Non-linear Analysis of Tapered Laminated Plate Under Uniform Inplane Load", Composite Structures, Vol. 7, No. 3, 1987, pp. 205-223.

65. WU, C.M.L., WEBBER, J.P.H., "Analysis of Tapered (in Steps) Laminated Plates Under Uniform Inplane Load", Composite Structures, Vol. 5, No. 2, 1986, pp. 87-100.

66. YEH, J.R., TADJBAKHSH, I.G., "Stress Singularity in Composite Laminates by Finite Element Method", Journal of Composite Materials, Vol. 20, July, 1986, pp. 347-364.

67. ZAGUSKIN, V.L., "Handbook of Numerical Methods for the Solution of Algebraic and Transcendental Equations", Pergamon Press, New-York, 1961.

68. ZIENKIEWICZ, O.C., "The Finite Element Method", Third Edition, McGraw-Hill, New-York, 1977.

69. ZWIERS, R.I., TING, T.C.T., "Singularity of Contact-Edge Stress in Laminated Composites Under Uniform Extension", Journal of Composite Materials, Vol. 17, Jan., 1982, pp. 49-63.

70. Discussion with Huang Q., Ph.D. student at Concordia University. Thesis title: "Hybrid Finite Element Stress Formulation for Composite Structures".

APPENDIX A

LISTING OF CLOSED FORM SOLUTION PROGRAM

The following *PASCAL* code is used for calculating the stresses around a triangular hole as presented in chapter 2. The logical sequence of events is to use ROOTS.PAS to obtain s_1 and s_2 for given material properties and then use STRESS.PAS to plot σ_x σ_y σ_{xy} σ_ρ σ_ϑ and $\sigma_{\rho\vartheta}$, send results to a printer or LOTUS 123®.

ANALYT.PAS

```
{
   Program to plot stress distribution around triangular hole
   with anisotropic material
   ANALYTICAL METHOD
      only valid at    theta=  0   180 degrees, rho=1
                with  zeta1:  -    -
                        m1:  -    +
                        n1:  +    -
}
{-------------------------------------------------------------------------}
{$N+}
Uses
   Crt,Graph,WaitKey,Declare,Printer,J_Graph,WhatKey,Math,CompRoot,Nombre;
Const
   Npoint_max = 500;
   Ncon_max = 1000;
Type
   MatrixR_PointxD = array[1..Npoint_max,1..3] of Rtype;
   VectI_Ncon = array[1..Ncon_max] of Integer;
   VectS2_Point = array[1..Npoint_max] of String[2];
   VectC_4 = array[1..4] of Ctype;
{-------------------------------------------------------------------------}
function equation(    e,theta0:Rtype):Rtype;
{
   Here is where we give the equation
}
var
   f: Rtype;
begin
   f:= cos(theta0)-2*e*cos(2*theta0);
   equation:=f;
end;
{-------------------------------------------------------------------------}
procedure Bisect(    a,b,TOL,e: Rtype;
                     Nmax: integer;
                var p: Rtype;
                var Root_ok: Boolean);
{
   Input:
      a,b: minimum and maximum values for root
      Tol: Tolerance on p error
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Nmax: maximum number of iterations
   Output:
      p: root
      Root_ok: True if root is ok
}
var
   i: integer;
   f_a,f_p: Rtype;
```

```
begin
   i:=1;
   Root_ok:=False;
   While i<=Nmax do
      begin
      p := (a+b)/2;
      f_a:=Equation(e,a);
      f_p:=Equation(e,p);
      If ((f_a=0) or ((b-a)/2<TOL)) then
         begin
         i:=Nmax;
         Root_ok:=True;
         end;
      i:=i+1;
      If f_a*f_p>0 then
         a:=p
      else
         b:=p;
      end;
end;
{-------------------------------------------------------------------}
Procedure Get_Theta0(    e: Rtype;
                     var Theta0: Rtype);
{
   Input:
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
   Output:
      Theta0: value of theta so that the boundary of the hole
              is paralel to x axis (4 TERM equation of hole)
}
var
   Root_ok: Boolean;
begin
   Theta0:=0;
   If (1/6<e) and (e<1/3)  then
      begin
      Bisect(105/180*Pi,125/180*Pi,1E-4,e,10000,Theta0,Root_ok);
      If Root_ok=False then
         begin
         writeln('Theta 0 not found');
         halt;
         end;
      end;
end;
{-------------------------------------------------------------------}
Procedure Two(    Title: String80;
             var XYZ: MatrixR_PointxD;        {var to save memory only}
                 Npoint: Integer;
             var Dep,Fin: VectI_Ncon;
             var Name: VectS2_Point;
                 Ntrait: integer;
                 Rho,D,Theta,Phi: Rtype;
```

```
                var Action: String6);
{
   Input:
      Title: Title;
      XYZ[i]: Z Y and Z Coordinates of point i
      Npoints: number of points
      Dep[i]: starting point # for line i
      Fin[i]: ending point # for line i
      Name[i]: Label for node i
      Ntrait: number of line
      Rho: The distance from the viewer to the origin
      D: Distance between the viewer and intersection plane
      Theta: Angle with X axis of view point in Degrees
      Phi: Angle with Z axis of view point in Degrees
   Output:
      Draw the preceding with Labeling of nodes
      Action: Action taken by user:
              'LEFT','RIGHT','UP','DOWN','ESC'
}
var
   XY: array[1..Npoint_Max,1..2] of Rtype;
   Margin,Ratio,amplitude,s1,c1,s2,c2,xe,ye,ze: Rtype;
   iD,i,GrDriver,GrMode: integer;
   Xasp,Yasp: word;
   a,Max,Min: array[1..2] of Rtype;
   Smax,Smin,Sign: array[1..2] of integer;
   C: Char;
   Mot: String[8];
begin
{
                Transform from degrees to radians
}
   Theta:=Theta*Pi/180;
   Phi:=Phi*Pi/180;
{
                Get Graphic Setup
}
   GrDriver:=Detect;
   InitGraph(grDriver,grMode,'');
   Smax[1]:=GetMaxX;
   Smin[1]:=0;
   Smax[2]:=0;
   Smin[2]:=GetMaxY;
   GetAspectRatio(Xasp,Yasp);
   Ratio:=Yasp/Xasp;
   S1:=Sin(Theta) ; C1:=Cos(Theta);
   S2:=Sin(Phi) ; C2:=Cos(Phi);
{
                Initialize Max Min
}
   For iD:=1 to 2 do
      begin
```

```
      Max[iD]:=-1E30;
      Min[iD]:=1E30;
      end;
{
                  Perspective projection
}
   for i:=1 to Npoint do
      begin
      Xe:=-xyz[i,1]*S1+xyz[i,2]*C1;
      Ye:=-xyz[i,1]*C1*C2-xyz[i,2]*S1*C2+xyz[i,3]*S2;
      Ze:=-xyz[i,1]*S2*C1-xyz[i,2]*S1*S2-xyz[i,3]*C2+Rho;
      XY[i,1]:=(D*Xe/Ze)*Ratio;
      XY[i,2]:=D*Ye/Ze;
{
                  Find Max min
}
      For iD:=1 to 2 do
         begin
         If XY[i,iD]<Min[iD] then Min[iD]:=XY[i,iD];
         If Max[iD]<XY[i,iD] then Max[iD]:=XY[i,iD];
         end;
      end;
{
                  Leave a margin for Max Min
}
   Margin:=0.025*(max[1]-Min[1]);   { X margin } {0.025 for = 2/80 = 2 col.}
   max[1]:=max[1]+Margin;
   min[1]:=min[1]-Margin;
   Margin:=0.03*(max[2]-Min[2]);   { Y margin } {0.03 for > 1/25/2=half a line}
   If Title<>'' then max[2]:=max[2]+Margin+(max[2]-min[2])/20
               else max[2]:=max[2]+Margin;
   min[2]:=min[2]-Margin;
{
                  Find appropriate amplitude (lowest ratio)
                  in order to keep proportion in geometry
                  Take Sign in account
}
   For iD:=1 to 2 do
      a[iD]:=(Smax[iD]-Smin[iD])/(max[iD]-min[iD]);
   If abs(a[1])<=abs(a[2]) then Amplitude:=abs(a[1])
                           else Amplitude:=abs(a[2]);
   For iD:=1 to 2 do
      Sign[iD]:=Round((Smax[iD]-Smin[iD])/abs(Smax[iD]-Smin[iD]));
{
                  Scale and center coordinates on screen
}
   For i:=1 TO Npoint do
      For iD:=1 to 2 do
      XY[i,iD]:=(XY[i,iD]-(max[iD]+min[iD])/2)*Sign[iD]*Amplitude
               +(Smax[iD]+Smin[iD])/2;
{
                  Draw lines
```

```
}
   For i:=1 to Ntrait do
      Line(Round(XY[Dep[i],1]),Round(XY[Dep[i],2]),
           Round(XY[Fin[i],1]),Round(XY[Fin[i],2]));
{
                  Get node Labels
}
   For i:=1 to Npoint do
      If Name[i]<>'' then
         Put_Mot(Name[i],XY[i,1],XY[i,2]);
{
                  Write Title
}
   If Title<>'' then
      OutTextXY(Round(GetMaxX/2-TextWidth(Title)/2),0,Title);
   Get_Key(C,Action);
   CloseGraph;    {Free heap memory used for graphics}
end;
{-------------------------------------------------------------------}
Procedure CalcZeta(    s1,z1: Ctype;
                       e,Ratio,Radius: Rtype;
                   var Zeta1: Ctype);
{
   Input:
      s1: Root for material (mu1 and mu2 in Lekhnitskii)
      z1: coordinate in conjugate plane
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
   Output:
      Zeta1: curvilinear coordinate
}
var
   t,t1,t2,t3,t4,a1,b1,c1,d1: Ctype;
   c1_2,c1_3,c1_6,d1_2,d1_3,d1_4: Ctype;
   z1_2,z1_3,z1_4: Ctype;
   y1,m1,n1: Ctype;
   V11,V21,V31,V41,V51,V61,V71,V81: Ctype;
begin
{
                  Get a1,b1,c1,d1
}
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CAdd(t1,t,a1);
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CSub(t1,t,b1);
   CDiv(a1,b1,c1);
   t[1]:=-2/e/Radius;   t[2]:=0;
```

```
   CDiv(t,b1,d1);
{
                Get powers of c1 and d1
}
   Cmult(c1,c1,c1_2);
   Cmult(c1,c1_2,c1_3);
   Cmult(c1_3,c1_3,c1_6);
   Cmult(d1,d1,d1_2);
   Cmult(d1,d1_2,d1_3);
   Cmult(d1_2,d1_2,d1_4);
{
                Get V11 ... V81
}
   t[1]:=1;    t[2]:=0;
   Cadd(t,c1_3,t);
   t2[1]:=1/2/e/e;    t2[2]:=0;
   Cmult(t2,t,V11);
   t[1]:=-(8+1/e/e)/6;  t[2]:=0;
   Cmult(t,c1,t);
   Cmult(t,d1,V21);
   t[1]:=1/27;    t[2]:=0;
   Cmult(t,d1_3,V31);
   t1[1]:=(27/e/e/e/e)/108;  t1[2]:=0;
   t2[1]:=(4/e/e/e/e/e/e+6/e/e/e/e+192/e/e-256)/108;    t2[2]:=0;
   t3[1]:=(27/e/e/e/e)/108;    t3[2]:=0;
   Cmult(t1,c1_6,t1);
   Cmult(t2,c1_3,t2);
   Cadd(t1,t2,t);
   Cadd(t,t3,V41);
   t[1]:=-(1/6/e/e/e/e+4/3/e/e);    t[2]:=0;
   Cmult(t,c1,t);
   Cmult(t,d1,t2);
   t[1]:=1;  t[2]:=0;
   Cadd(t,c1_3,t);
   Cmult(t2,t,V51);
   t[1]:=(128-1/e/e/e/e+80/e/e)/108;  t[2]:=0;
   Cmult(t,c1_2,t);
   Cmult(t,d1_2,V61);
   t[1]:=1/27/e/e;  t[2]:=0;
   Cmult(t,d1_3,t2);
   t[1]:=1;  t[2]:=0;
   Cadd(t,c1_3,t);
   Cmult(t2,t,V71);
   t[1]:=-4/27;  t[2]:=0;
   Cmult(t,c1,t);
   Cmult(t,d1_4,V81);
{
                get powers of z1
}
   Cmult(z1,z1,z1_2);
   Cmult(z1,z1_2,z1_3);
   Cmult(z1_2,z1_2,z1_4);
```

```
{
                get y1
}
   Cmult(V21,z1,t);
   Cadd(V11,t,t1);
   Cmult(V31,z1_3,t);
   Cadd(t1,t,t1);
   Cmult(V51,z1,t);
   Cadd(V41,t,t2);
   Cmult(V61,z1_2,t);
   Cadd(t2,t,t2);
   Cmult(V71,z1_3,t);
   Cadd(t2,t,t2);
   Cmult(V81,z1_4,t);
   Cadd(t2,t,t2);
   Croot(2,t2,t2);
   Cadd(t1,t2,t3);
   Csub(t1,t2,t4);
   Croot(3,t3,t3);
   Croot(3,t4,t4);
   Cadd(t3,t4,t1);
   t[1]:=1/3;  t[2]:=0;
   Cmult(t,d1,t);
   Cmult(t,z1,t2);
   Cadd(t1,t2,y1);
{
                Check for imaginary y1
}
   If abs(y1[2])>0.0001 then
      begin
      writeln('y1 is not real ');
      writeln('z1 = ',z1[1]:8,'+i',z1[2]:8);
      halt;
      end;
{
                get m1, n1 and zeta1
}
   t1[1]:=1/e/e;  t1[2]:=0;
   t2[1]:=-4;     t2[2]:=0;
   t3[1]:=4;      t3[2]:=0;
   Cmult(t1,c1_2,t1);
   Cmult(t2,d1,t2);
   Cmult(t2,z1,t2);
   Cmult(t3,y1,t3);
   Cadd(t1,t2,t2);
   Cadd(t2,t3,t2);
   Croot(2,t2,t2);
   t[1]:=1/e;  t[2]:=0;
   Cmult(t,c1,t1);
   Csub(t1,t2,m1);          ( + or - sign for m1 )
   Cmult(y1,y1,t1);
   t[1]:=-4;   t[2]:=0;
```

```
   Cmult(t,c1,t2);
   Cadd(t1,t2,t2);
   Croot(2,t2,t2);
   Cadd(y1,t2,n1);          { inverse of that for m1 }
   Cmult(m1,m1,t1);
   t[1]:=-8;  t[2]:=0;
   Cmult(t,n1,t2);
   Cadd(t1,t2,t2);
   Croot(2,t2,t2);
   t[1]:=0; t[2]:=0;
   Csub(t,m1,t1);
   Csub(t1,t2,t1);          { + or - sign for zeta1 }
   t[1]:=1/4;  t[2]:=0;
   Cmult(t,t1,Zeta1);
end;
{-------------------------------------------------------------------------}
Procedure CalcAllZeta(    s,z: Ctype;
                          e,Ratio,Radius: Rtype;
                      var Zeta,dZeta: Ctype);
{
   Input:
      s: Root of (1.59) for material
      z: coordinate in conjugate plane
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
   Output:
      Zeta: curvilinear coordinate
      dZeta: derivate of Zeta1
}
var
   z_2: Ctype;
   Zeta_2: Ctype;
   dreal: Ctype;
begin
{
                  Calculate Zeta1 and
                  Numerically differentiate Zeta1
}
   CalcZeta(s,z,e,Ratio,Radius,Zeta);
   dreal[1]:=1E-8;   dreal[2]:=0;
   CAdd(dreal,z,z_2);
   CalcZeta(s,z_2,e,Ratio,Radius,Zeta_2);
   CSub(Zeta_2,Zeta,dZeta);
   CDiv(dZeta,dreal,dZeta);
end;
{-------------------------------------------------------------------------}
Procedure CalcStress(    X,Y: Rtype;
                         s1,s2: Ctype;
                     var P,e,alpha,Ratio,Radius,lambda,Rho: Rtype; {var for 8087}
                     var StressX,StressY,StressXY,
                         StressR,StressT,StressRT: Rtype);
```

```
{
   Input:
      X,Y: coordinates of point of interest
      s1,s2: material property roots (mu1 and mu2 in Lekhnitskii's book)
      P:  Stress load
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Alpha: angle for P with x axis
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
      lambda: angle(radians) of rho axis to x axis
      Rho: second coordinate of curvilinear coordinate system
           expressing radius in a unit circle
           used to know if the last root seed must be used when rho<>1
   Output:
      StressX,StressY,StressXY: Stress in X-Y plane
      StressR,StressT,StressRT: Stress in Rho-Theta plane
}
var
   t,ta,tb,t1,t2: Ctype;   {temporary}
   z1,z2: Ctype;
   a1,b1,a2,b2: Ctype;
   cg,sg,ca,sa,Bs,Bps,Cps,al1,al2,be1,be2: Rtype;
   Zeta1,dZeta1,Zeta2,dZeta2: Ctype;
   k1,k1c,k2,k2c,k5,k5c,k6,k6c: Ctype;
   dx: Ctype;
   dPhi1,dPsi2: Ctype;
   Sx,Sy,Sxy,Sr,St,Srt: Rtype;
begin
{
                  Get z1,z2
}
   t[1]:=y;      t[2]:=0;
   CMult(s1,t,z1);
   z1[1]:=z1[1]+X;
   t[1]:=y;      t[2]:=0;
   CMult(s2,t,z2);
   z2[1]:=z2[1]+X;
{
                  Get a1,b1,a2,b2
}
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CAdd(t1,t,a1);
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CSub(t1,t,b1);
   t[1]:=0;    t[2]:=1;
   CMult(t,s2,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CAdd(t1,t,a2);
```

```
   t[1]:=0;    t[2]:=1;
   CMult(t,s2,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CSub(t1,t,b2);
{
                  Get Zeta1,dZeta1,Zeta2,dZeta2 for each 4 roots
}
   CalcAllZeta(s1,z1,e,Ratio,Radius,Zeta1,dZeta1);
   CalcAllZeta(s2,z2,e,Ratio,Radius,Zeta2,dZeta2);
{
                  Get k1,k2,k5,k6 and conjugates
}
   ca:=cos(alpha);
   sa:=sin(alpha);
   al1:=s1[1];
   be1:=s1[2];
   al2:=s2[1];
   be2:=s2[2];
   Bs:=P*(ca*ca+(al2*al2+be2*be2)*sa*sa+al2*sin(2*alpha))
        /2/(sqr(al2-al1)+(be2*be2-be1*be1));
   Bps:=P*((al1*al1-be1*be1-2*al1*al2)*sa*sa-ca*ca-al2*sin(2*alpha))
        /2/(sqr(al2-al1)+(be2*be2-be1*be1));
   Cps:=P*((al1-al2)*ca*ca+(al2*(al1*al1-be1*be1)
           -al1*(al2*al2-be2*be2))*sa*sa
           +((al1*al1-be1*be1)-(al2*al2-be2*be2))*sa*ca)
        /2/be2/(sqr(al2-al1)+(be2*be2-be1*be1));
   t[1]:=Bs;   t[2]:=0;
   CMult(t,a1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,a2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k1);
   t[1]:=Bs;   t[2]:=0;
   CMult(t,b1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,b2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k2);
   t[1]:=Bs;   t[2]:=0;
   CMult(t,s1,t1);
   CMult(t1,a1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,s2,t2);
   CMult(t2,a2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k5);
   t[1]:=Bs;   t[2]:=0;
   CMult(t,s1,t1);
   CMult(t1,b1,t1);
```

```
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,s2,t2);
   CMult(t2,b2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k6);
   CConj(k1,k1c);
   CConj(k2,k2c);
   CConj(k5,k5c);
   CConj(k6,k6c);
{
                Get derivate of Phi, Psi
}
   CAdd(k1,k2c,t1);
   CMult(s2,t1,t1);
   CAdd(k5,k6c,t2);
   CSub(t1,t2,t1);
   CSub(s1,s2,t2);
   CDiv(dZeta1,t2,ta);
   CMult(ta,t1,ta);
   CAdd(k2,k1c,t1);
   CMult(s2,t1,t1);
   CAdd(k6,k5c,t2);
   CSub(t1,t2,t1);
   CSub(s1,s2,t2);
   t[1]:=2.0*e;   t[2]:=0;
   CMult(t,Zeta1,tb);
   CMult(tb,dZeta1,tb);
   CDiv(tb,t2,tb);
   CMult(tb,t1,tb);
   CAdd(ta,tb,dPhi1);
   CAdd(k1,k2c,t1);
   CMult(s1,t1,t1);
   CAdd(k5,k6c,t2);
   CSub(t2,t1,t1);
   CSub(s1,s2,t2);
   CDiv(dZeta2,t2,ta);
   CMult(ta,t1,ta);
   CAdd(k2,k1c,t1);
   CMult(s1,t1,t1);
   CAdd(k6,k5c,t2);
   CSub(t2,t1,t1);
   CSub(s1,s2,t2);
   t[1]:=2.0*e;   t[2]:=0;
   CMult(t,Zeta2,tb);
   CMult(tb,dZeta2,tb);
   CDiv(tb,t2,tb);
   CMult(tb,t1,tb);
   CAdd(ta,tb,dPsi2);
{
                Get stresses
                When rho=1 look for the roots which give (stress)rho=0
```

```
)
   CMult(s1,s1,t1);
   CMult(t1,dPhi1,t1);
   CMult(s2,s2,t2);
   CMult(t2,dPsi2,t2);
   CAdd(t1,t2,t1);
   Sx := P*sqr(cos(alpha))+2*t1[1];
   CAdd(dPhi1,dPsi2,t1);
   Sy := P*sqr(sin(alpha))+2*t1[1];
   CMult(s1,dPhi1,t1);
   CMult(s2,dPsi2,t2);
   CAdd(t1,t2,t1);
   Sxy := P*sin(alpha)*cos(alpha)-2*t1[1];
{
                  Transform stress into curvilinear coordinate system
}
   cg:=cos(lambda);
   sg:=sin(lambda);
   Sr:=cg*cg*Sx+sg*sg*Sy+2*cg*sg*Sxy;
   St:=sg*sg*Sx+cg*cg*Sy-2*cg*sg*Sxy;
   Srt:=-cg*sg*Sx+cg*sg*Sy+(cg*sg-sg*sg)*Sxy;
   StressX  := Sx;
   StressY  := Sy;
   StressXY := Sxy;
   StressR  := Sr;
   StressT  := St;
   StressRT := Srt;
end;
{-------------------------------------------------------------------------}
Procedure GetXY(    Rho,Theta,e,Ratio,Radius: Rtype;
                var X,Y,lambda: Rtype);
{
   4 TERM FORMULA
   Input:
      Rho,Theta: curvilinear coordinates
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
   Output:
      X,Y: Plane coordinates
      lambda: angle(radians) of rho axis to x axis
}
var
   a,b,rho2,x2,y2: Rtype;
begin
   a:=ratio+1;
   b:=Ratio-1;
   x:=Radius/2*((a/rho+b*rho)*cos(theta)+e*(a*rho*rho+b/rho/rho)*cos(2*theta));
   y:=Radius/2*((-a/rho+b*rho)*sin(theta)+e*(a*rho*rho-b/rho/rho)*sin(2*theta));
{
                  get angle of theta axis
}
```

```
   rho2:=rho+0.00001;
   x2:=Radius/2*((a/rho2+b*rho2)*cos(theta)+e*(a*rho2*rho2+b/rho2/rho2)*cos(2*theta));
   y2:=Radius/2*((-a/rho2+b*rho2)*sin(theta)+e*(a*rho2*rho2-b/rho2/rho2)*sin(2*theta));
   lambda:=ArcTan4(x2-x,y2-y);
   If Pi<lambda then lambda:=lambda-2*Pi;
end;
{-------------------------------------------------------------------------}
Procedure DrawStress(    Material: String80;
                         s1,s2: Ctype;
                         P,e,alpha,Ratio,Radius: Rtype;
                         StepDegree: Rtype;
                         UseOutput: String80;
                         WhichStress: Byte;
                         LotusDirectory: String80);
{
   Input:
      Material: Name of Material
      s1,s2: material property roots
      P: Stress load
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Alpha: (radians) angle for P with x axis
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
      StepDegree: step for angle (degrees)
      UseOutput: 'PRINTER' if output results along rho=1 to printer
                 'GRAPH' if plot is desired
                 'LOTUS' if Lotus 123 output is desired
      WhichStress: 1 for StressX
                   2     StressY
                   3     StressXY
                   4     StressTheta
                   5     StressRho
                   6     StressThetaRho
      LotusDirectory: directory for lotus (with last \)
   Output:
      Printer Listing of stresses
      or
      Stress Plot
}
var
   Title: String80;
   X,Y: Rtype;
   LastDegree,Degree,Rho,Theta,Theta0,lambda: Rtype;
   StressX,StressY,StressXY: Rtype;
   StressR,StressT,StressRT: Rtype;
   j,iPoint,ToContour,Contour: integer;
   R: array[1..10] of Rtype;
   Stress: array[1..6] of Rtype;
   F: text;
   Fname: String[80];
{
                    Graph variables
```

```
}
   XYZ: MatrixR_PointxD;
   iStress,NpointBefore,Npoint: Integer;
   Dep,Fin: VectI_Ncon;
   Name: VectS2_Point;
   Ntrait: integer;
   Action: String6;
   StoreZ: array[1..6,1..250] of Rtype;
   Scale,MaxZ,MinZ: Rtype;
begin
   ClrScr;
   If UseOutput='LOTUS' then
     begin
     Repeat
        ClrScr;
        gotoXY(1,10);
        writeln('Previous file with same name will be erased');
        write('Lotus file name (no extension .PRN): ',LotusDirectory);
        readln(Fname);
        until Pos('.',Fname)=0;
     Assign(F,LotusDirectory+Fname+'.PRN');
     Rewrite(F);
     end;
   Get_ThetaO(e,ThetaO);
{
                Display initial data, theta=0
}
   R[1]:=1;
   R[2]:=0.95;
   R[3]:=0.9;
   R[4]:=0.85;
   R[5]:=0.8;
   R[6]:=0.75;
   R[7]:=0.7;
   For iPoint:=1 to Npoint_max do Name[iPoint]:='';
   NpointBefore:=0;
   Npoint:=0;
   Ntrait:=0;
{
               Axis
}
   XYZ[1][1]:=0;    XYZ[1][2]:=0;     XYZ[1][3]:=0;    Name[1]:='O';
   XYZ[2][1]:=2.3;  XYZ[2][2]:=0;     XYZ[2][3]:=0;    Name[2]:='+X';
   XYZ[3][1]:=0;    XYZ[3][2]:=-2;    XYZ[3][3]:=0;    Name[3]:='-Y';
   XYZ[4][1]:=0;    XYZ[4][2]:=0;     XYZ[4][3]:=2;    Name[4]:='+S';
   Npoint:=4;
   NpointBefore:=4;
   Dep[1]:=1;  Fin[1]:=2;
   Dep[2]:=1;  Fin[2]:=3;
   Dep[3]:=1;  Fin[3]:=4;
   Ntrait:=3;
{
```

```
                Calculate stresses
                so that (stress)rho=0
}
  If UseOutput='PRINTER' then ToContour:=1;
  If UseOutput='GRAPH'   then ToContour:=5;
  If UseOutput='LOTUS'   then ToContour:=1;
  If UseOutput='PRINTER' then
     begin
     writeln(Lst,'Material: ',Material);
     writeln(Lst,'s1 = ',s1[1]:7:5,'+(',s1[2]:7:5,')i');
     writeln(Lst,'s2 = ',s2[1]:7:5,'+(',s2[2]:7:5,')i');
     writeln(Lst,'P (Pa) = ',P:5:2);
     writeln(Lst,'epsilon = ',e:6:4);
     writeln(Lst,'alpha (degrees)= ',alpha*180/Pi:8:3);
     writeln(Lst,'L/h = ',Ratio*Sqrt(3)/2:7:4,'  r = ',Ratio:7:4);
     writeln(Lst,'Radius of unit circle= ',Radius:7:4);
     writeln(Lst);
     write(Lst,chr(27),'M',chr(13));
     writeln(Lst,'  Theta      X        Y       lambda      Sx       Sy       Sxy        Sr
St       Srt');
     writeln(Lst,'(degree)    (m)      (m)      (deg)      (Pa)     (Pa)      (Pa)      (Pa)       (
Pa)      (Pa)');
     end;
  If UseOutput='LOTUS' then
     begin
     writeln(F,'"Material: ',Material);
     writeln(F,'"s1 = ',Rfield(s1[1],7),'+(',Rfield(s1[2],7),')i');
     writeln(F,'"s2 = ',Rfield(s2[1],7),'+(',Rfield(s2[2],7),')i');
     writeln(F,'"P (Pa) = ',Rfield(P,5));
     writeln(F,'"epsilon = ',Rfield(e,6));
     writeln(F,'"alpha (degrees)= ',Rfield(alpha*180/Pi,8));
     writeln(F,'"L/h = ',Rfield(Ratio*Sqrt(3)/2,7),'  r = ',Rfield(Ratio,7));
     writeln(F,'"Radius of unit circle= ',Rfield(Radius,7));
     writeln(F);
     writeln(F,'"  Theta      X        Y       lambda      Sx       Sy       Sxy      Sr       St
    Srt');
     writeln(F,'"(degree)    (m)      (m)      (deg)      (Pa)     (Pa)      (Pa)     (Pa)     (Pa)
    (Pa)');
     end;
  Degree:=0;
  While Degree<=180.001 do
     begin
     Theta:=Degree/180*Pi;
     For Contour:=1 to ToContour do
        begin
        gotoxy(1,12);
        write('Angle = ',Degree:5:3,'    Contour: ',Contour,'    ');
        Rho:=R[Contour];
        GetXY(Rho,Theta,e,Ratio,Radius,X,Y,lambda);
        CalcStress(X,Y,s1,s2,P,e,alpha,Ratio,Radius,lambda,Rho,
                   StressX,StressY,StressXY,StressR,StressT,StressRT);
        Stress[1]:=StressX;
```

```
        Stress[2]:=StressY;
        Stress[3]:=StressXY;
        Stress[4]:=StressR;
        Stress[5]:=StressT;
        Stress[6]:=StressRT;
{
                Store all elevations
}
        For iStress:=1 to 6 do
          StoreZ[iStress][Round((Npoint+2-4)/2)]:=Stress[iStress];
        If UseOutput='PRINTER' then
          begin
{
                Highlight result for Theta0
}
          If 0.001<Abs(Degree-Round(Degree/StepDegree)*StepDegree) then
            begin
            Write(Lst,Theta*180/Pi:8:4,Chr(13));
            Write(Lst,Theta*180/Pi:8:4,Chr(13));
            write(Lst,Theta*180/Pi:8:4,' ',
                      X:7:4,' ',
                      Y:7:4,' ',
                      lambda*180/Pi:8:3,' ',
                      Stress[1]:9:5,' ',
                      Stress[2]:9:5,' ',
                      Stress[3]:9:5,' ',
                      Stress[4]:9:5,' ',
                      Stress[5]:9:5,' ',
                      Stress[6]:9:5,Chr(13));
            end;
          writeln(Lst,Theta*180/Pi:8:4,' ',
                    X:7:4,' ',
                    Y:7:4,' ',
                    lambda*180/Pi:8:3,' ',
                    Stress[1]:9:5,' ',
                    Stress[2]:9:5,' ',
                    Stress[3]:9:5,' ',
                    Stress[4]:9:5,' ',
                    Stress[5]:9:5,' ',
                    Stress[6]:9:5);
          end;
        If UseOutput='LOTUS' then
          begin
          writeln(F,Rfield(Theta*180/Pi,10),' ',
                    Rfield(X,10),' ',
                    Rfield(Y,10),' ',
                    Rfield(lambda*180/Pi,10),' ',
                    Rfield(Stress[1],11),' ',
                    Rfield(Stress[2],11),' ',
                    Rfield(Stress[3],11),' ',
                    Rfield(Stress[4],11),' ',
                    Rfield(Stress[5],11),' ',
```

```
                    Rfield(Stress[6],11));
{  For Thesis
          writeln(F,Theta*180/Pi:3:0,' ',
                    Stress[1]:6:3,' ',
                    Stress[2]:6:3,' ',
                    Stress[3]:6:3,' ',
                    Stress[4]:6:3,' ',
                    Stress[5]:6:3,' ',
                    Stress[6]:6:3);
}
          end;
        If UseOutput='GRAPH' then
          begin
          If NpointBefore<Npoint then
             begin
             Dep[Ntrait+1]:=Npoint;   (last top)
             Fin[Ntrait+1]:=Npoint+2; (this top)
             Inc(Ntrait);
             end;
          Dep[Ntrait+1]:=Npoint+1;    (this bottom)
          Fin[Ntrait+1]:=Npoint+2;    (this top)
          Inc(Ntrait);
          Inc(Npoint);
          XYZ[Npoint][1]:=x;    {this bottom)
          XYZ[Npoint][2]:=y;
          XYZ[Npoint][3]:=0;
          Inc(Npoint);
          XYZ[Npoint][1]:=x;    (this top)
          XYZ[Npoint][2]:=y;
          XYZ[Npoint][3]:=Stress[WhichStress];
          end;
        end;
{
                  Draw lines with previous contour
}
      If UseOutput='GRAPH' then
        begin
        If 0<Degree then
          begin
          For j:=0 to Round((Npoint-NpointBefore)/2)-1 do
             begin
             Inc(Ntrait);
             Dep[Ntrait]:=NpointBefore-j*2;
             Fin[Ntrait]:=Npoint-j*2;
             end;
          end;
        NpointBefore:=Npoint;
        end;
      LastDegree:=Degree;
      If ((UseOutput='PRINTER') or (UseOutput='LOTUS'))
           and (Degree<Theta0*180/Pi)
           and (Theta0*180/Pi<=Degree+StepDegree) then
```

```
        Degree:=Theta0*180/Pi
      else
        Degree:=Trunc(Degree/StepDegree)*StepDegree+StepDegree;
      If abs(Degree-LastDegree)<0.0001 then Degree:=Degree+StepDegree;
      end;
    If UseOutput='PRINTER' then
      begin
      write(Lst,Chr(12),Chr(13));
      write(Lst,chr(27),'P',chr(13));
      end;
    If UseOutput='LOTUS' then
      Close(F);
    If UseOutput='GRAPH' then
      begin
      ClrScr;
      gotoXY(1,10);
      writeln('Use <Up> <Down> <Home> <End> to change from stress plot:');
      writeln('    -> Stress: X  Y  XY  Rho  Theta  RhoTheta');
      writeln('Press <Esc> to end after first plot');
      wait;
      If Npoint_max<Npoint then Npoint:=Npoint_max;
      If Ncon_max<Ntrait then Ntrait:=Ncon_max;
{
                Scale Z coordinate
}
      iStress:=WhichStress;
      Repeat
        For j:=1 to Round((Npoint-4)/2) do
          XYZ[4+2*j][3]:=StoreZ[iStress][j];
        MaxZ:=-1E30;
        MinZ:=+1E30;
        For j:=5 to Npoint do
          begin
          If maxZ<XYZ[j][3] then maxZ:=XYZ[j][3];
          If XYZ[j][3]<minZ then minZ:=XYZ[j][3];
          end;
        Scale:=abs(XYZ[4][3])/(maxZ-minZ);   {use z axis to scale}
        For j:=5 to Npoint do
          XYZ[j][3]:=XYZ[j][3]*Scale;
        Case iStress of
          1: Title:='Stress X';
          2: Title:='Stress Y';
          3: Title:='Stress XY';
          4: Title:='Stress Rho';
          5: Title:='Stress Theta';
          6: Title:='Stress RhoTheta';
          end;
        Two(Title,XYZ,Npoint,Dep,Fin,Name,Ntrait,1000,1000,-115,30,Action);
        If Action='UP' then iStress:=iStress-1;
        If Action='DOWN' then iStress:=iStress+1;
        If Action='HOME' then iStress:=1;
        If Action='END' then iStress:=6;
```

```
        If iStress=-1 then iStress:=6;
        If iStress=7 then iStress:=1;
        until Action='ESC';
      end;
end;
{-------------------------------------------------------------------}
var
   s1,s2: Ctype;
   StepDegree,e,L,h,P,alpha,Ratio,Radius: Rtype;
   WhichStress: integer;
   LotusDirectory,UseOutput,Material: String80;
   C: Char;
   Action: String6;
begin
{
   Material:='Isotropic, plane STRESS or STRAIN';
   s1[1]:=0;
   s1[2]:=1.01;
   s2[1]:=0;
   s2[2]:=1;
   Material:='Plywood, plane STRAIN';
   s1[1]:=0;
   s1[2]:=4.1032;
   s2[1]:=0;
   s2[2]:=0.3293;
   Material:='T300/5208 Graphite/Epoxy, plane STRAIN';
   s1[1]:=0;
   s1[2]:=4.8940;
   s2[1]:=0;
   s2[2]:=0.8042;
   Material:='CE 9000 Glass/Epoxy, plane STRAIN';
   s1[1]:=0;
   s1[2]:=2.3992;
   s2[1]:=0;
   s2[2]:=0.6757;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -)
   Material:='CE 9000 Glass/Epoxy, plane STRAIN';
   s1[1]:=0;
   s1[2]:=2.3992;
   s2[1]:=0;
   s2[2]:=0.6757;
   P:=1;
   e:=1/3;
   alpha:=0*Pi/180;  (radians)
{          0.8660254 for equilateral triangle -> r=1)
   L:=0.8660254;
   h:=1;
   Ratio:=2/sqrt(3)*L/h;
   Radius:=1;
{                  7.5 degrees -> 112.5, 180)
{                  7.2 degrees -> 115.2, 180)
   StepDegree:=0.1;  (positive only)
```

```
   WhichStress:=1;
   ClrScr;
   writeln('Material: ',Material);
   writeln('s1 = ',s1[1]:7:5,'+(',s1[2]:7:5,')i');
   writeln('s2 = ',s2[1]:7:5,'+(',s2[2]:7:5,')i');
   writeln('P (Pa) = ',P:5:2);
   writeln('epsilon = ',e:6:4);
   writeln('alpha (degrees)= ',alpha*180/Pi:8:3);
   writeln('L/h = ',Ratio*Sqrt(3)/2:7:4,'  r = ',Ratio:7:4);
   writeln('Radius of unit circle= ',Radius:7:4);
   writeln;
   writeln('Which Output?');
   writeln('   Graph');
   writeln('   Printer');
   writeln('   Lotus');
   Get_Key(C,Action);
   C:=upCase(C);
   Case C of
      'G': UseOutput:='GRAPH';
      'P': UseOutput:='PRINTER';
      'L': UseOutput:='LOTUS';
      end;
   LotusDirectory:='C:\Lotus\Files\';
   If C in ['G','P','L'] then
      DrawStress(Material,s1,s2,P,e,alpha,Ratio,Radius,
                 StepDegree,UseOutput,WhichStress,LotusDirectory);
end.
```

BISECT.PAS

```
{$N+}  {8087 math coprocessor}
Uses
   Crt;
Type
   Rtype=Double;
{-------------------------------------------------------------------------}
function equation(    x:Rtype):Rtype;
{
   Here is where we give the equation
}
var
   E,Xc,L,p1,p2,p3,p4: Rtype;
begin
   E:=0;
   p1:=-9/16*(3*E*E-2*E-1/9);
   p2:= 27/16*(3*E*E-2/3*E-1);
   p3:=-27/16*(3*E*E+2/3*E-1);
   p4:= 9/16*(3*E*E+2*E-1/9);
   L:=1;
   Xc:=L/2;
   equation := p1*(Xc-L/2)+p2*(Xc-x)+p3*(Xc+x)+p4*(Xc+L/2);
end;
{-------------------------------------------------------------------------}
procedure Bisect(    a,b,TOL: Rtype;
                     Nmax: integer;
                var p: Rtype;
                var Root_ok: Boolean);
{
   Bisection algorithm
   Burden & Faires, Numerical Analysis, 3rd edition, Page 29
                                   Written by Jerome Daoust in Fall '87
}
var
   i: integer;
   f_a,f_p: Rtype;
begin
   i:=1;
   Root_ok:=False;
   While i<=Nmax do
      begin
      p := (a+b)/2;
      f_a:=Equation(a);
      f_p:=Equation(p);
      If ((f_a=0) or ((b-a)/2<TOL)) then
         begin
         i:=Nmax;
         Root_ok:=True;
         end;
      i:=i+1;
      If f_a*f_p>0 then
```

```
        a:=p
      else
        b:=p;
      end;
end;
{-------------------------------------------------------------------------}
var
   a,b,TOL,p: Rtype;
   Nmax: integer;
   root_ok: boolean;
begin
   clrscr;
   write('Borne inferieure: ');readln(a);
   write('Borne superieure: ');readln(b);
   write('Tolerance       : ');readln(TOL);
   Nmax:=1000;
   Bisect(a,b,TOL,Nmax,p,Root_ok);
   If Root_ok then
      writeln('La racine est ',p:20:15)
   else
      writeln('Not found after ',Nmax,' iterations');
end.
```

COMPROOT.PAS

```
{$N+}
Unit CompRoot;
Interface
  Uses Crt,Declare,Math;
Procedure CheckOneRoot(    A: VectC_D;
                           N: integer;
                           RootOk: Boolean;
                           Criterion: Rtype;
                           x: Ctype;
                       var Value: Ctype;
                       var ZeroOk: Boolean);
Procedure ComplexRoots(    A: VectC_D;
                           N: integer;
                           HowMany: String;
                           MaxCount: integer;
                           RS_Precision: Rtype;
                       var r,s: VectC_D;
                       var x: VectC_D;
                       var Iter: VectI_D;
                       var RootOk: VectBoolean_D);
{-------------------------------------------------------------------------}
Implementation
Procedure CheckOneRoot;
{
Procedure CheckOneRoot(    A: VectC_D;
                           N: integer;
                           RootOk: Boolean;
                           Criterion: Rtype;
                           x: Ctype;
                       var Value: Ctype;
                       var ZeroOk: Boolean);
}
{
  Input:
    A[0..N]: complex coeficients of A[0]z^N+A[1]z^(N-1)+...A[N]=0
    N: degree of equation
    RootOk: If the value is thought to be a root
    Criterion: accepted margin upon real and imaginary part of
               polynomial with this root
    x: alleged root of A
  Output:
    Value: Value of the polynomial
    ZeroOk: True if polynomial is zero
}
var
  xp: VectC_D;
  i: integer;
  t: Ctype;
begin
  ZeroOk:=False;
```

```
   If RootOk=True then
      begin
      xp[0][1]:=1;          xp[0][2]:=0;
      Value[1]:=a[N][1];    Value[2]:=a[N][2];
      For i:=N-1 downto 0 do
         begin
         Cmult(xp[N-i-1],x,xp[N-i]);
         Cmult(xp[N-i],a[i],t);
         Cadd(Value,t,Value);
         end;
      If (Criterion>abs(Value[1])+abs(Value[2])) then ZeroOk:=True;
      end;
end;
{-------------------------------------------------------------------}
Procedure ComplexBairstow(var A: VectC_D;
                          var N: integer;
                              MaxCount: Integer;
                              RS_Precision: Rtype;
                          var RS_Ok: Boolean;
                          var r,s: Ctype;
                          var IterationCount: integer;
                          var x1,x2: Ctype);
{
   Input:
      A[0..N]: complex coeficients of A[0]z^N+A[1]z^(N-1)+...A[N]=0
      N: degree of equation
      MaxCount: maximum number of iteration
      RS_Precision: precision required upon r,s
      r,s: initial seed guess
   Output
      A[0..N]: deflated polynomial complex coeficients
               A[0]z^N+A[1]z^(N-1)+...A[N]=0
      N: degree of deflated polynomial:  (original degree)-2
      RS_Ok: True if we have desired r,s precision
             after MaxCount iterations
      r,s: corrected guess
      IterationCount: number of iterations done
      x1,x2: roots, function of r,s
}
var
   b,c: array[-2..20] of Ctype;
   r0,s0,r1,s1: Ctype;
   i,k: integer;
   t,J,delta: Ctype;
begin
   r1:=r;
   s1:=s;
   IterationCount:=0;
   Repeat
      s0:=s1;
      r0:=r1;
      b[-2][1]:=0;   b[-2][2]:=0;
```

```
    b[-1][1]:=0;   b[-1][2]:=0;
    c[-2][1]:=0;   c[-2][2]:=0;
    c[-1][1]:=0;   c[-1][2]:=0;
    For k:=0 to N do
      begin
      Cmult(r0,b[k-1],t);
      Cadd(a[k],t,b[k]);
      Cmult(s0,b[k-2],t);
      Cadd(b[k],t,b[k]);    { b[k] }
      Cmult(r0,c[k-1],t);
      Cadd(b[k],t,c[k]);
      Cmult(s0,c[k-2],t);
      Cadd(c[k],t,c[k]);    { c[k] }
      end;
    Cmult(c[n-2],c[n-2],J);
    Cmult(c[n-1],c[n-3],t);
    CSub(J,t,J);
    Cmult(b[n-1],c[n-2],r1);
    Cmult(b[n],c[n-3],t);
    Csub(r1,t,r1);
    Cdiv(r1,J,r1);
    Csub(r0,r1,r1);
    Cmult(b[n-1],c[n-1],s1);
    Cmult(b[n],c[n-2],t);
    Csub(s1,t,s1);
    Cdiv(s1,J,s1);
    Cadd(s0,s1,s1);
    Inc(IterationCount);
    until (abs(r1[1]-r0[1])+abs(r1[2]-r0[2])
          +abs(s1[1]-s0[1])+abs(s1[2]-s0[2])<RS_Pracision)
          or (MaxCount<=IterationCount);
  If MaxCount<IterationCount then
    RS_Ok:=False
  else
    begin
    RS_Ok:=True;
    r:=r1;
    s:=s1;
{
                get roots x1,x2
}
    Cmult(r,r,delta);
    t[1]:=4;    t[2]:=0;
    Cmult(t,s,t);
    Cadd(delta,t,delta);
    Croot(2,delta,delta);
    t[1]:=2;    t[2]:=0;
    Cdiv(Delta,t,Delta);
    Cdiv(r,t,t);
    Cadd(t,delta,x1);
    Csub(t,delta,x2);
{
```

```
                Deflate Polynomial
)
      For i:=0 to N-2 do
         A[i]:=B[i];
      N:=N-2;
      end;
end;
(-------------------------------------------------------------------)
Procedure ComplexRoots;
(
Procedure ComplexRoots(     A: VectC_D;
                            N: integer;
                            HowMany: String;
                            MaxCount: integer;
                            RS_Precision: Rtype;
                        var r,s: VectC_D;
                        var x: VectC_D;
                        var Iter: VectI_D;
                        var RootOk: VectBoolean_D);
)
(
   Input:
      A[0..N]: complex coeficients of A[0]z^N+A[1]z^(N-1)+...A[N]=0
      N: degree of equation
      HowMany: if 'TwoFirstOnly' then only get the two first roots
               Only 1 if N=1 and HowMany='TwoFirstOnly'
      MaxCount: maximum number of iteration
      RS_Precision: precision required upon r,s
      r[i],s[i]: initial seed guess of each pair of roots [i=Trunc(N/2)]
   Output
      r[i],s[i]: corrected guess for each pair of roots [i=Trunc(N/2)]
      x[1..N]: roots, x[1] and x[2] are function r,s
      Iter[1..N]: number of iterations done for a root
      RootOk[1..N]: True if this root converged within
                    MaxCount and RS_Precision conditions
)
var
   t: Ctype;
   i,k,Iteration2: integer;
   iPair,iRoot,N_Original,N_Previous,N_Deflated: integer;
   Original,Previous,Deflated: VectC_D;
   RS_Ok: Boolean;
   b: array[-2..20] of Ctype;
   Xscreen,Yscreen: Byte;
begin
   Xscreen:=WhereX;
   Yscreen:=WhereY;
   GotoXY(Xscreen,Yscreen);
   write(N,' ');
   If DegreeMax<N then
      begin
      writeln('An equation of degree ',N,' is too high, (',DegreeMax,' max)');
```

```
    Halt;
    end;
{
                Initialize
}
  For i:=1 to N do
    begin
    x[i][1]:=0;   x[i][2]:=0;
    RootOk[i]:=False;
    Iter[i]:=0;
    end;
{
                Set original polynomial as deflated one
}
  N_Deflated:=N;
  For i:=0 to N do
    Deflated[i]:=A[i];
{
                Find first two roots, r, s: x[1],x[2],r,s
}
  iPair:=1;
  If 2<=N_Deflated then
    begin
    ComplexBairstow(Deflated,N_Deflated,MaxCount,RS_Precision,
                    RS_Ok,r[iPair],s[iPair],Iter[1],x[1],x[2]);
    Iter[2]:=0;
    RootOk[1]:=RS_Ok;
    RootOk[2]:=RS_Ok;
    end;
{
                Find other roots
}
  While ((2<=N_Deflated) and (HowMany<>'TwoFirstOnly')) do
    begin
    GotoXY(Xscreen,Yscreen);
    write(N_Deflated,' ');
{
                Keep previous deflated polynomial
}
    For i:=0 to N_Deflated do
      Previous[i]:=Deflated[i];
    N_Previous:=N_Deflated;
{
                Get roots
}
    iRoot:=N-N_Deflated+1;
    Inc(iPair);
    ComplexBairstow(Deflated,N_Deflated,MaxCount,RS_Precision,
                    RS_Ok,r[iPair],s[iPair],Iter[iRoot],
                    x[iRoot],x[iRoot+1]);
    Iter[iRoot+1]:=0;
    RootOk[iRoot]:=RS_Ok;
```

```
      RootOk[iRoot+1]:=RS_Ok;
{
                  Improve precision by using original polynomial
                  with same r,s
                  useful for polynomial of degree 6 and higher
}
      If RS_Ok=True then
         begin
         For i:=0 to N do
            Original[i]:=a[i];
         N_Original:=N;
         ComplexBairstow(Original,N_Original,MaxCount,RS_Precision,
                        RS_Ok,r[iPair],s[iPair],Iteration2,
                        x[iRoot],x[iRoot+1]);
         Iter[iRoot]:=Iter[iRoot] + Iteration2;
         Iter[iRoot+1]:=0;
         RootOk[iRoot]:=RS_Ok;
         RootOk[iRoot+1]:=RS_Ok;
{
                  Improve deflated polynomial with r,s
                  from Previous deflated polynomial
                  do not alter roots
}
         b[-2][1]:=0;   b[-2][2]:=0;
         b[-1][1]:=0;   b[-1][2]:=0;
         For k:=0 to N_Previous do
            begin
            Cmult(r[iPair],b[k-1],t);
            Cadd(Previous[k],t,b[k]);
            Cmult(s[iPair],b[k-2],t);
            Cadd(b[k],t,b[k]);      { b[k] }
            end;
         For i:=0 to N_Previous-2 do    { Deflate }
            Deflated[i]:=B[i];
         N_Deflated:=N_Previous-2;
         end;
      end;
{
                  Solve last odd root
                  D[0]z+D[1]=0
}
   If ((1=N_Deflated)and(HowMany<>'TwoFirstOnly'))
      or ((1=N)and(HowMany='TwoFirstOnly')) then
      begin
      GotoXY(Xscreen,Yscreen);
      write(N_Deflated,' ');
      iRoot:=N-N_Deflated+1;
      t[1]:=0;    t[2]:=0;
      Csub(t,Deflated[1],t);
      Cdiv(t,Deflated[0],x[iRoot]);
      RootOk[iRoot]:=True;
      Iter[iRoot]:=0;
```

```
      end;
   GotoXY(Xscreen,Yscreen);
   write(' ');
   GotoXY(Xscreen,Yscreen);
end;
{-------------------------------------------------------------------}
end.
```

DECLARE.PAS

```
{$N+}
Unit Declare;
Interface
Const
   DegreeMax = 20;
Type
   Rtype = Double;
   Ctype = array[1..2] of Rtype;
   Itype = LongInt;
   VectC_D = array[0..DegreeMax] of Ctype;
   VectI_D = array[0..DegreeMax] of integer;
   VectBoolean_D = array[0..DegreeMax] of Boolean;
   String6 = String[6];
   String80 = String[80];
   String255 = String[255];
{-------------------------------------------------------------------}
Implementation
end.
```

J_GRAPH.PAS

```
{$N+}
Unit J_Graph;
Interface
Uses
   Graph,Declare;
Procedure Put_Mot(    Mot: String6;
                      X,Y: Rtype);
{-------------------------------------------------------------------------}
Implementation
Procedure Put_Mot;
{
Procedure Put_Mot(    Mot: String6;
                      X,Y: Rtype);
}
{
   Input:
      Mot: word to be displayed
      X,Y: screen coordinate to center the number
   Output:
      Number is centered at screen coordinates (X,Y)
      for the closest row and column
}
begin
   OutTextXY(Round(X-TextWidth(Mot)/2.0),
             Round(Y-TextHeight(Mot)/2.0),Mot);
end;
{-------------------------------------------------------------------------}
end.
```

LEKHNIT.PAS

```
{
      Using Lekhnitskii's formula (54.8)
      and Plywood
}
var
   stressA,stressC,stressD,
      theta0,                { theta0 is for point D }
      mu1,mu2,
      sint,sin2t,cost,cos2t,
      d,e,g,h,k,l,n,
      Ac,Bc,Cc,Dc,Lc: Real;     { second letter for capital }
   Material: String[255];
begin
{
   Material:='Isotropic, plane STRAIN or STRESS';
   mu1:=1.0000;
   mu2:=1.0000;
   e:=1/4;
   theta0:=111.47/180*Pi;
   Material:='Isotropic, plane STRAIN or STRESS';
   mu1:=1.0000;
   mu2:=1.0000;
   e:=1/3;
   theta0:=115.17/180*Pi;
   Material:='Isotropic, plane STRAIN or STRESS, circle';
   mu1:=1.0000;
   mu2:=1.0000;
   e:=0;
   theta0:=90/180*Pi;
   Material:='Plywood, plane STRAIN';
   mu1:=4.1032;
   mu2:=0.3293;
   e:=1/4;
   theta0:=111.47/180*Pi;
   Material:='Plywood, plane STRESS';
   mu1:=4.1019;
   mu2:=0.3449;
   e:=1/4;
   theta0:=111.47/180*Pi;
   Material:='Plywood, plane STRAIN';
   mu1:=4.1032;
   mu2:=0.3293;
   e:=1/3;
   theta0:=115.17/180*Pi;
   Material:='Plywood, plane STRESS';
   mu1:=4.1019;
   mu2:=0.3449;
   e:=1/3;
   theta0:=115.17/180*Pi;
   Material:='CE 9000, plane STRAIN';
```

```
   mu1:=2.3992;
   mu2:=0.6757;
   e:=1/4;
   theta0:=111.47/180*Pi;
   Material:='CE 9000, plane STRAIN';
   mu1:=2.3992;
   mu2:=0.6757;
   e:=1/3;
   theta0:=115.17/180*Pi;
}
   Material:='CE 9000, plane STRAIN';
   mu1:=2.3992;
   mu2:=0.6757;
   e:=1/3;
   theta0:=115.17/180*Pi;
   k:=mu1*mu2;
   n:=mu1+mu2;
   g:=8*(1-k)/sqr(1+k+n);
   h:=2*(sqr(1-n)+k*(k+2*n-6))/sqr(1+k+n);
   d:=4/(1+k+n);
   l:=2*(1-k-n)/(1+k+n);
   sint:=sin(theta0);
   sin2t:=sin(2*theta0);
   cost:=cos(theta0);
   cos2t:=cos(2*theta0);
   Ac:=cost-2*e*cos2t;
   Bc:=sint+2*e*sin2t;
   Cc:=sqrt(Ac*Ac+Bc*Bc);
   Dc:=-Ac*Ac*Ac*Ac*k+Ac*Ac*Bc*Bc*(1-2*k-k*k)+Bc*Bc*Bc*Bc*(2+k-n*n);
   Lc:=(Bc*Bc+sqr(mu1)*Ac*Ac)*(Bc*Bc+sqr(mu2)*Ac*Ac);
{
                  ortho, point A, D, C
}
   writeln(Material);
   writeln('e=',e:8:5);
   writeln('theta0=',theta0*180/Pi:8:3,' degrees');
   writeln('mu1=',mu1:8:5,'i');
   writeln('mu2=',mu2:8:5,'i');
   stressA:=1/k*(-1+e*e*g*n/(1-2*e)-e*e*e*n*(d*h+g*l+d*g*n)/(1-2*e));
   writeln('Point A, stress theta = ',stressA:6:3);
   stressD:=1+n/Bc*(sint-2*e*sin2t+e*e*h*sint+e*e*e*(d*g*k-h*l)*sin2t);
   writeln('Point D, stress theta = ',stressD:6:3);
   stressC:=1/k*(-1+e*e*g*n/(1+2*e)+e*e*e*n*(d*h+g*l+d*g*n)/(1+2*e));
   writeln('Point C, stress theta = ',stressC:6:3);
{
                  Isotropic (ok)
   stress:=1/Cc/Cc*(1-2*cos(2*theta0)+4*e*cos(theta0)-4*e*e);
   writeln(stress);
}
end.
```

MATH.PAS

```
{$N+}
Unit Math;
Interface
   Uses Declare;
Function ArcCos(    X: Rtype): Rtype;
Function ArcTan4(    x,y: Rtype):Rtype;
Function PowerR(    base,puis:Rtype) :Rtype;
Procedure Cadd(    n1,n2: Ctype;
                var t: Ctype);
Procedure Csub(    n1,n2: Ctype;
                var t: Ctype);
Procedure Cmult(    n1,n2: Ctype;
                 var t: Ctype);
Procedure Cdiv(    n1,n2: Ctype;
                var t: Ctype);
Procedure Cconj(    n1: Ctype;
                 var t: Ctype);
Procedure Croot(    n: integer;
                    c: Ctype;
                 var t: Ctype);
{-------------------------------------------------------------------------}
Implementation
Function ArcCos;
{
Function ArcCos(    X: Rtype): Rtype;
}
{
   Neuton-Raphson algorithm used to get ArcCos(X)
   Burden & Faires, Numerical Analysis, 3rd edition, Page 42
}
var
   Root_ok: Boolean;
   TOL: Rtype;
   Nmax,i: integer;
   p,p0: Rtype;
begin
   TOL:=1E-10;
   Nmax:=1000;
   p0:=0.5;
   i:=1;
   Root_ok:=False;
   While i<=Nmax do
      begin
      p := p0 - (Cos(p0)-X)/(-Sin(p0));  {x=x0-f/f'}
      If (abs(p-p0)<TOL) then
         begin
         i:=Nmax;
         Root_ok:=True;
         end;
      i:=i+1;
```

```
      p0:=p;
      end;
   If Root_ok=False then
      begin
      writeln('Unable to get ArcCos(',X,')');
      Halt;
      end;
   If p0<0 then ArcCos:=-p0    { keep result in 0<=angle<=Pi }
          else ArcCos:= p0;    { knowing that cos(angle)=cos(-angle) }
end;
{-------------------------------------------------------------------------}
Function ArcTan4;
{
Function ArcTan4(    x,y: Rtype):Rtype;
}
{
   Input:
      x,y: 2-D coordinates
   Output:
      4 quadrant ArcTangent angle in radians
}
var
   Angle: Rtype;
begin
   if x=0 then
      begin
      If 0<=y then
         Angle:=Pi/2
      else
         Angle:=-Pi/2;
      end
   else
      Angle:=ArcTan(y/x);
   If x<0 then Angle:=Angle+Pi;
   ArcTan4:=Angle;
end;
{-------------------------------------------------------------------------}
function PowerR;
{
function PowerR(    base,puis:Rtype) :Rtype;
}
{
            Returns   (base)^(puis)
                                   Written by Jerome Daoust in Fall '87
}
var
   Crit: Rtype;
begin
   Crit:=1E-5;
   If 0<base then
      PowerR := exp(puis*ln(base))          {base>0}
   else
```

```
      begin
      If base=0 then
         PowerR := 0.0                          {base=0}
      else
         begin
         if Abs(Puis-round(Puis))<Crit then
            if Abs(Puis/2-Round(Puis/2))<Crit then
               PowerR := exp(puis*ln(-base))    {base<0 Puis=Pair}
            else
               PowerR := -exp(puis*ln(-base))   {base<0 Puis=Impair}
         else
            if Abs(1/Puis-round(1/Puis))<Crit then
               if Abs(1/Puis/2-Round(1/Puis/2))<Crit then
                  PowerR := exp(puis*ln(-base))    {base<0 1/Puis=Pair}
               else
                  PowerR := -exp(puis*ln(-base))   {base<0 1/Puis=Impair}
            else
               Writeln('(ERROR: Base=',base:8,' Power=',Puis:8,')');
         end;
      end;
end;
{-------------------------------------------------------------------------}
Procedure Cadd;
{
Procedure Cadd(    n1,n2: Ctype;
                var t: Ctype);
}
{
   Input:
     n1,n2: complex numbers  n1=a+i.b  n2=c+i.d
   Output:
     t=n1+n2
}
var
   a,b,c,d: Rtype;
begin
   a:=n1[1];
   b:=n1[2];
   c:=n2[1];
   d:=n2[2];
   t[1]:=a+c;
   t[2]:=b+d;
end;
{-------------------------------------------------------------------------}
Procedure Csub;
{
Procedure Csub(    n1,n2: Ctype;
                var t: Ctype);
}
{
   Input:
     n1,n2: complex numbers  n1=a+i.b  n2=c+i.d
```

```
   Output:
      t=n1-n2
}
var
   a,b,c,d: Rtype;
begin
   a:=n1[1];
   b:=n1[2];
   c:=n2[1];
   d:=n2[2];
   t[1]:=a-c;
   t[2]:=b-d;
end;
{-------------------------------------------------------------------------}
Procedure Cmult;
{
Procedure Cmult(    n1,n2: Ctype;
                var t: Ctype);
}
{
   Input:
      n1,n2: complex numbers  n1=a+i.b  n2=c+i.d
   Output:
      t=n1*n2
}
var
   a,b,c,d: Rtype;
begin
   a:=n1[1];
   b:=n1[2];
   c:=n2[1];
   d:=n2[2];
   t[1]:=a*c-b*d;
   t[2]:=a*d+b*c;
end;
{-------------------------------------------------------------------------}
Procedure Cdiv;
{
Procedure Cdiv(    n1,n2: Ctype;
               var t: Ctype);
}
{
   Input:
      n1,n2: complex numbers  n1=a+i.b  n2=c+i.d
   Output:
      t=n1/n2
}
var
   a,b,c,d: Rtype;
begin
   a:=n1[1];
   b:=n1[2];
```

```
   c:=n2[1];
   d:=n2[2];
   t[1]:=(a*c+b*d)/(sqr(c)+sqr(d));
   t[2]:=(b*c-a*d)/(sqr(c)+sqr(d));
end;
{-------------------------------------------------------------------}
Procedure Cconj;
{
Procedure Cconj(    n1: Ctype;
                var t: Ctype);
}
{
   Input:
      n1: complex number  n1=a+i.b
   Output:
      t = a-i.b
}
begin
   t[1]:= n1[1];
   t[2]:=-n1[2];
end;
{-------------------------------------------------------------------}
Procedure Croot;
{
Procedure Croot(    n: integer;
                    c: Ctype;
                var t: Ctype);
}
{
   Input:
      c: complex number  c1=a+i.b
   Output:
      t = c^(1/n)    first root
}
var
   rp,x,y,r,theta: Rtype;
begin
   x:=c[1];
   y:=c[2];
{
                    Convert x,y to polar coordinates r,theta
                    x+i.y = r[cos(theta)+i.sin(theta)]
}
   r:=sqrt(sqr(x)+sqr(y));
   Theta:=ArcTan4(x,y);
{
                    Get root with 6.10 of Schaum
}
   rp:=PowerR(r,1/n);
   t[1]:=rp*cos(theta/n);
   t[2]:=rp*sin(theta/n);
end;
```

```
{------------------------------------------------------------------------}
end.
```

NOMBRE.PAS

```
{$N+}
Unit Nombre;
Interface
Uses
   Declare;
procedure Separate(    Number: Rtype;
                   var Prefix: Rtype;
                   var Power: Integer);
Function RField(    NumberR: Rtype;
                    Field: Integer): String255;
Function RWord(    NumberR: Rtype;
                   Field: Integer): String255;
Procedure StoX(    S. String;    {do not use Var (modified)}
               var Ni: Itype;
               var Nr: Rtype;
               var IRtype: Byte);
{-------------------------------------------------------------------}
Implementation
procedure Separate;
{
procedure Separate(    Number: Rtype;
                   var Prefix: Rtype;
                   var Power: Integer);
}
{
          Separate Prefix from Power in a real number
}
var
   Ln10,P: Rtype;
begin
   Ln10:=ln(10);
   If number=0.0 then
      begin
      Power:=0;
      Prefix:=0.0;
      end
   else
      begin
      P:=ln(abs(number))/ln10;
      If abs(Round(P)-P)<1E-8 then
         Power:=Round(P)
      else
         Power:=Trunc(P);
      If P<0 then Dec(Power);
      Prefix:=number/exp(Power*ln10);
      end;
end;
{-------------------------------------------------------------------}
Function RField;
{
```

```
Function RField(    NumberR: Rtype;
                    Field: Integer): String255;
}
{
   Input:
      NumberR: Real number
      Field: Length of field for display
   Output:
      Real number is put into a word of length FIELD
}
var
   Prefix: Rtype;
   PowerOfPrefix,After,PowerMin,PowerMax,Power: Integer;
   Mot,PreW,PowW: String255;
begin
{
                  Find maximum and minimum power to avoid exponent
}
   If 0<=NumberR then PowerMax:=Field-1
               else PowerMax:=Field-2;
   If 6<PowerMax then PowerMax:=6;
   If 0<=NumberR then PowerMin:=-Trunc(Field/3)
               else PowerMin:=-Trunc(Field/3)+1;
   If PowerMin<-3 then PowerMin:=-3;
{
                  Separate number:  -3.45E-0056 ->  -3.45  -56
}
   Separate(NumberR,Prefix,Power);
   If (PowerMin<=Power) and (Power<=PowerMax) then
      begin
      If 0<=Power then
         After:=Field-Power-2  {for a positive Power}
      else
         After:=Field-2;       {for a negative Power}
      If NumberR<0 then
         Dec(After);
      str(NumberR:Field:After,Mot)
      end  .
   else
      begin
{
                  Put in Engineering exponents: 3 6 9 -3 -6 -9 if possible
}
      PowerOfPrefix:=0;
      Case Power of
         -17,-14,-11,-8,-5,-2,1,4,7,10,13,16: begin
            Prefix:=Prefix*10.0;
            Power:=Power-1;
            PowerOfPrefix:=1;
            end;
         -16,-13,-10,-7,-4,-1,2,5,8,11,14,17: begin
            Prefix:=Prefix*100.0;
```

```
            Power:=Power-2;
            PowerOfPrefix:=2;
            end;
         end;
      str(Power,PowW);
      If Power>=0 then
         PowW:='+'+PowW;
      PowW:='E'+PowW;
{
                  erase 0 just after 'E+' or 'E-'
                  E+04 ->  E+4
}
      While PowW[Pos('E',PowW)+2]='0' do
         Delete(PowW,Pos('E',PowW)+2,1);
{
                  Erase '+' just after 'E'
}
      If PowW[Pos('E',PowW)+1]='+' then
         Delete(PowW,Pos('E',PowW)+1,1);
{
                  Get Prefix and back fill with 0's to match Field
                  524.3:4:0  -> ' 524'
}
      If Prefix>=0 then
         str(Prefix:Field-Length(PowW)
                  :Field-(Length(PowW)+2+PowerOfPrefix),PreW)
      else
         str(Prefix:Field-Length(PowW)
                  :Field-(Length(PowW)+3+PowerOfPrefix),PreW);
{
                  Truncate 100.0000E+12 to
                           100.000E+12      when number is 99.999999E+12
}
      While Field<Length(PreW)+Length(PowW) do
         Delete(PreW,Length(PreW),1);
      Mot:=PreW+PowW;
      end;
   RField:=Mot;
end;
{-------------------------------------------------------------------}
Function RWord;
{
Function RWord(    NumberR: Rtype;
                   Field: Integer): String255;
}
{
   Input:
      NumberR: Real number
      Field: Maximum Length of field for display
   Output:
      Real number is put into a word of length FIELD
         unneccessary 0's are taken out
```

```
}
var
   Mot: String255;
begin
   Mot:=RField(NumberR,Field);
{
                  Delete Blanks
                  524.3:4:0 -> ' 524' -> '524'
}
   While Mot[1]=' ' do
      Delete(Mot,1,1);
{
           erase '0' or '.' just before 'E' only if '.' is present
           3.45000E+04 ->  3.45E+04
           500E6 -> 500E6
}
   If 0<Pos('E',mot) then
      While (0<Pos('.',Mot)) and (Mot[Pos('E',mot)-1] in ['0','.']) do
         Delete(Mot,Pos('E',mot)-1,1);
{
           erase 0 at the end if no 'E' is found and a '.' is found
           3.45000E+04 ->  3.45E+04
}
   If Pos('E',Mot)<=0 then
      If 0<Pos('.',Mot) then
         begin
         While Mot[Length(Mot)] in ['0'] do
            Delete(Mot,Length(Mot),1);
         If Mot[Length(Mot)]='.' then
            Delete(Mot,Length(Mot),1);
         end;
   RWord:=Mot;
end;
{-------------------------------------------------------------------}
Procedure StoX;
{
Procedure StoX(    S: String;   do not use Var (modified)
              var Ni: Itype;
              var Nr: Rtype;
              var IRtype: Byte);
}
{
   Input:
      S: string
   Output:
      Ni: integer number
      Nr: real number
      IRtype: 0 if no conversion can be done (string only)
              1 if integer conversion done
              2 if real conversion done
}
Var
```

```
   Result: integer;
begin
   IRtype:=0;
   If S<>'' then
      begin
{
                  so +X is considered as number X
                  If a number is given as   .3  ->  0.3
                                           -.3 -> -0.3
                                            3.  ->  3.0
}
      If (copy(S,1,2)='-.') then
         Insert('0',S,2)
      else
         begin
         If S[1]='+' then Delete(S,1,1);
         If (S[1]='.') then S:='0'+S;
         end;
      If (S[Length(S)]='.') then S:=S+'0';
{
                  Store IType or Real number in appropriate place
                  if possible
}
      Val(S,Ni,Result);
      If Result=0 then
         IRtype:=1
      else
         begin
         val(S,Nr,Result);
         If Result=0 then IRtype:=2
         end;
      end;
end;
{-------------------------------------------------------------------}
end.
```

ROOTS.PAS

```
{
  Program used to obtain coefficient of equation 1.59 of Savin (1961)
}
{-------------------------------------------------------------------}
{$N+}
Uses
   Crt,CompRoot,Declare;
Type
   MatrixR_3x3 = array[1..3,1..3] of Rtype;
   MatrixR_6x6 = array[1..6,1..6] of Rtype;
   MatrixR_20x20 = array[1..20,1..20] of Rtype;
   MatrixR_20x5 = array[1..20,1..5] of Rtype;
{-------------------------------------------------------------------}
procedure Separate(    Number: Rtype;
                   var Prefix: Rtype;
                   var Power: Integer);
{
          Separate Prefix from Power in a real number
}
var
   Ln10,P: Rtype;
begin
   Ln10:=ln(10);
   If number=0 then
      begin
      Power:=0;
      Prefix:=0;
      end
   else
      begin
      P:=ln(abs(number))/ln10;
      If abs(Round(P)-P)<1E-8 then
         Power:=Round(P)
      else
         Power:=Trunc(P);
      If P<0 then Power:=Power-1;
      Prefix:=number/exp(Power*ln10);
      end;
end;
{--------- -------------------------------------------------------------}
procedure multiply(var prefix1: Rtype;
                   var power1: integer;
                   var number2:Rtype);
{
        (prefix1).10^(power1)=(prefix1).10^(power1).number2
        by seperating prefixes and exponants
        and avoiding an exponant overflow
}
var
   prefix1_power,power2: integer;
```

```
   prefix2: Rtype;
begin
   Separate(Number2,Prefix2,Power2);
   power1:=power1+power2;
   Separate(prefix1*prefix2,Prefix1,Prefix1_Power);
   power1:=power1+prefix1_power;
end;
{-------------------------------------------------------------------------}
procedure Matx_IDS(    N:integer;
                   var A: MatrixR_20x20;
                       M: integer;
                   var B: MatrixR_20x5;
                   var Determinant: Rtype;
                   var Error_Type: integer);
{
Aim
   This Subroutine is for MATriX Inversion, Determinant
   and simultaneous linear equation Solving (MATX_IDS).
Method
   Matrix inversion with complete pivoting to avoid
   nul pivots and minimize rounding errors.
   Saves memory by storing results in the original matrices.
   This subroutine was originaly written in FORTRAN in
   chapter 8 of "Computer Applications of Numerical Methods"
   by S.S. Kuo, Addison-Wosley.
   Re-written PASCAL and modified by Jerome Daoust in Fall of '87.
Definition of the variables
   N           : Number of columns an rows in matrix A
   A[N,N]      : Matrix to be inversed.
                 Inversed matrix is stored here also.
   M           : Number of columns in matrix B.
   B[N,M]      : coefficient matrix, each column constitutes
                 with matrix A a set of simultaneous linear
                 equations. The result of each system will
                 be stored in the same matrix.
   Determinant : determinant of matrix A.
   Error_Type  : Error code
                 =0 No problem
                 =1 Matrix can't be inverted
                 =2 Determinant out of bounds
                    (-1E201...-1E-201  and  1E-201...1E201)
}
var
   T,Eps,Det_Prefix: Rtype;
   i,j,k,h,irow,icol,li,Det_Power: integer;
   Finish: boolean;
   Ipvot,Pivot: array[1..30] of Rtype;
   Index: array[1..30,1..2] of integer;
begin
{
                   Set smallest dividor for accuracy in division
}
```

```
   Eps:= 1E-15;
{
                  Initialisation
}
   Finish:=false;
   Det_Prefix:=1;
   Det_Power:=0;
   For j:=1 to N do Ipvot[j]:=0;
   For i:=1 to N do if Finish=False then
{
                  Search of pivot element
}
      begin
      T:=0;
      For j:=1 to N do
         If Ipvot[j]<>1 then
            begin
            For k:=1 to N do
               begin
               If Ipvot[k]-1<0 then
                  If ABS(T)<ABS(A[j,k]) then
                  begin
                  irow:=j;
                  icol:=k;
                  T:=A[j,k];
                  end;
               If Ipvot[k]-1>0 then Finish:=true
               end;
            end;
      if Finish=false then
{
                  Testing for non-inverting matrix
}
         begin
         If ABS(T)>Eps then
            begin
            Error_Type:=0;
            Pivot[i]:=T;
            Ipvot[icol]:=Ipvot[icol]+1;
{
                  Put Pivot element on diagonal
}
            If irow<>icol then
               begin
               Det_Prefix:=-Det_Prefix;
               for h:=1 to N do
                  begin
                  T:=A[irow,h];
                  A[irow,h]:=A[icol,h];
                  A[icol,h]:=T;
                  end;
               If M<=0 then
```

```
                for h:=1 to M do
                   begin
                   T:=B[irow,h];
                   B[irow,h]:=B[icol,h];
                   B[icol,h]:=T;
                   end;
                end;
             Index[i,1]:=irow;
             Index[i,2]:=icol;
{
                Use a multiplication routine for huge or tiny
                intermediate values of the determinant
}
             Multiply(Det_Prefix,Det_Power,Pivot[i]);
{
                Divide Pivot row by Pivot element
}
             A[icol,icol]:=1;
             for h:=1 to N do A[icol,h]:=A[icol,h]/Pivot[i];
             If M>0 then for h:=1 to M do B[icol,h]:=B[icol,h]/Pivot[i];
{
                Reduce non-pivot rows
}
             for li:=1 to N do
                begin
                If li<>icol then
                   begin
                   T:=A[li,icol];
                   A[li,icol]:=0;
                   for h:=1 to N do A[li,h]:=A[li,h]-A[icol,h]*T;
                   If M>0 then for h:=1 to M do B[li,h]:=B[li,h]-B[icol,h]*T;
                   end;
                end
             end
          else
             begin
             Error_Type:=1;
             Finish:=true;
             end;
          end;
       end;
{
                Interchange columns
}
   For i:=1 to N do if Finish=False then
      begin
      h:=N-i+1;
      If Index[h,1]<>Index[h,2] then
         begin
         irow:=Index[h,1];
         icol:=Index[h,2];
         For k:=1 to N do
```

```
            begin
            T:=A[k,irow];
            A[k,irow]:=A[k,icol];
            A[k,icol]:=T;
            end;
         end;
      end;
{
                  Restore determinant as a Rtype
}
   If ((-200<=Det_Power)and(Det_Power<=200)) then
      begin
      Determinant := Det_Prefix;
      If 0<=Det_Power then
         For i:=1 to Det_Power do
            Determinant:=Determinant*10
      else
         For i:=1 to Det_Power do
            Determinant:=Determinant/10;
      end
   else
      Error_Type:=2;
end;
{-------------------------------------------------------------------}
procedure Invert6(var INP,OUT: MatrixR_6x6);
{
                  INP is the input matrix,
                  OUT is the inverted matrix
}
var
   i,j,N:integer;
   A: MatrixR_20x20;
   B: MatrixR_20x5;
   Determinant: Rtype;
   Error_Type: integer;
begin
   N:=6;
   For i:=1 to N do
      For j:=1 to N do
         A[i,j]:=INP[i,j];
   For i:=1 to N do            {initialize coefficient matrix}
      For j:=1 to 5 do
         B[i,j]:=0;
   Matx_IDS(N,A,0,B,Determinant,Error_Type);
   If Error_Type<>0 then
      begin
      ClrScr;
      writeln('I can not invert this matrix:');
      For i:=1 to N do
         begin
         For j:=1 to N-1 do
            write(INP[i,j]:9,' ');
```

```
        writeln(INP[i,N]:9);
        end;
     writeln('You should use a smaller EPS in inversion subroutine');
     Repeat until Keypressed;
     end
  else
     For i:=1 to N do
        For j:=1 to N do
           OUT[i,j]:=A[i,j];
end;
{-------------------------------------------------------------------------}
Procedure Get_c_PlaneStrain(    El,Et,Nlt,Ntt,Glt,Gtt: Rtype;
                            var c: MatrixR_6x6);
{
   Input:
     El: Longitudinal Young's modulus
     Et: Transverse Young's modulus
     Nlt: Longitudinal-Transverse Poisson Ratio
     Ntt: Transverse-Transverse Poisson Ratio
     Glt: Longitudinal-Transverse Shear Stiffness (Tensorial)
     Gtt: Transverse-Transverse Shear Stiffness (Tensorial)
   Output:
     c[1..6,1..6]: On-Axis stiffness matrix
        [e1,e2,e3,g23,g13,g12]^T = [c].[s1,s2,s3,s23,s13,s12]^T
        strains are engineering strains
}
Var
   i,j: byte;
   A: MatrixR_6x6;
   Delta,E1,E2,E3,G12,G13,G23,n12,n13,n23,n21,n31,n32: Rtype;
begin
{
                Initialize stiffness matrix
}
   For i:=1 to 6 do
     For j:=1 to 6 do
        A[i,j]:=0.0;
{
                Get On-axis stiffness matrix
                [s1,s2,s3,s23,s13,s12]^T = [A].[e1,e2,e3,g23,g13,g12]^T
                strains are engineering strains
}
   E1:=El;
   E2:=Et;
   E3:=E2;
   G12:=Glt;
   G13:=G12;
   G23:=Gtt;
   n12:=Nlt;
   n13:=n12;
   n23:=Ntt;
   n21:=n12/E1*E2;
```

```
   n31:=n13/E1*E3;
   n32:=n23/E2*E3;
   Delta:=(1-n12*n21-n23*n32-n31*n13-2*n21*n32*n13)/(E1*E2*E3);
{
                  [C] is symmetric so get upper traingular then copy
}
   A[1,1]:=(1-n23*n32)/(E2*E3*Delta);
   A[1,2]:=(n12+n32*n13)/(E1*E3*Delta);
   A[1,3]:=A[1,2];
   A[2,2]:=(1-n13*n31)/(E1*E3*Delta);
   A[2,3]:=(n23+n21*n13)/(E1*E2*Delta);
   A[3,3]:=A[2,2];
   A[6,6]:=G12;    (not doubled)
   A[5,5]:=G12;    (not doubled)
   A[4,4]:=G23;    (not doubled)
{
                   [A] is symmetric so get upper traingular then copy
}
   For i:=1 to 6 do
      For j:=i to 6 do
         A[j,i]:=A[i,j];
{
                   Invert [A] to get [c]
}
   Invert6(A,c);
end;
{-------------------------------------------------------------------}
Procedure Get_S_PlaneStress(    Ex,Ey,Nx,Es: Rtype;
                            var S: MatrixR_3x3);
{
   Intended for orthotropic material along x axis
   Input:
      Ex: Young's modulus along x
      Ey: Young's modulus along y
      Nx: Major Poisson Ratio
      Es: Shear Stiffness (engineering)
   Output:
      S[1..3,1..3]: stiffness matrix
         [ex,ey,gxy]^T = [S].[Sx,Sy,Sxy]^T
         strains are engineering strains (g12=2*e12)
}
var
   Ny,m: Rtype;
begin
   ny:=nx/Ex*Ey;
   m:=1/(1-nx*ny);
   S[1,1]:=1/Ex;
   S[1,2]:=-Ny/Ey;
   S[1,3]:=0;
   S[2,1]:=-Nx/Ex;
   S[2,2]:=1/Ey;
   S[2,3]:=0;
```

```
   S[3,1]:=0;
   S[3,2]:=0;
   S[3,3]:=1/Es;
end;
{-------------------------------------------------------------------}
Procedure MaterialRoots(    El,Et,Nlt,Ntt,Glt,Gtt: Rtype;
                            Material,PlaneStrainStress: String255);
{
   Input:
      El: Longitudinal Young's modulus
      Et: Transverse Young's modulus
      Nlt: Longitudinal-Transverse Poisson Ratio
      Ntt: Transverse-Transverse Poisson Ratio
      Glt: Longitudinal-Transverse Shear Stiffness (Tensorial)
      Gtt: Transverse-Transverse Shear Stiffness (Tensorial)
      Material: Material name
      PlaneStrainStress: ='STRAIN' if plane strain case is desired
                         ='STRESS' if plane stress case is desired
   Output:
      fourth degree equation coefficients to obtain s1, s2
}
var
   S: MatrixR_3x3;
   c: MatrixR_6x6;
   a11,a12,a16,a22,a26,a66: Rtype;
{       root variables  }
   A: VectC_D;
   i,N: integer;
   HowMany: String;
   MaxCount: integer;
   RS_Precision: Rtype;
   rf,sf,x: VectC_D;
   Iter: VectI_D;
   RootOk: VectBoolean_D;
   Value: Ctype;
   ZeroOk: Boolean;
begin
   If (PlaneStrainStress<>'STRAIN') and (PlaneStrainStress<>'STRESS') then
      begin
      writeln('Must be Plane STRAIN or Plane STRESS');
      Halt;
      end;
   If PlaneStrainStress='STRESS' then
      begin
      Get_S_PlaneStress(El,Et,Nlt,Glt,S);
      a11:= S[1,1];
      a12:= S[1,2];
      a16:= S[1,3];
      a22:= S[2,2];
      a26:= S[2,3];
      a66:= S[3,3];
      end;
```

```
  If PlaneStrainStress='STRAIN' then
     begin
     Get_c_PlaneStrain(El,Et,Nlt,Ntt,Glt,Gtt,c);
     a11:= (c[1,1]*c[3,3]-c[1,3]*c[1,3])/c[3,3];
     a12:= (c[1,2]*c[3,3]-c[1,3]*c[2,3])/c[3,3];
     a16:= (c[1,6]*c[3,3]-c[1,3]*c[3,6])/c[3,3];
     a22:= (c[2,2]*c[3,3]-c[2,3]*c[2,3])/c[3,3];
     a26:= (c[2,6]*c[3,3]-c[2,3]*c[3,6])/c[3,3];
     a66:= (c[6,6]*c[3,3]-c[3,6]*c[3,6])/c[3,3];
     end;
  writeln('Plane ',PlaneStrainStress,' case for ',Material);
  writeln('a11 = ',a11);
  writeln('a12 = ',a12);
  writeln('a16 = ',a16);
  writeln('a22 = ',a22);
  writeln('a26 = ',a26);
  writeln('a66 = ',a66);
  writeln;
  writeln(a11,'s^4 +');
  writeln(-2*a16,'s^3 +');
  writeln(2*a12+a66,'s^2 +');
  writeln(-2*a26,'s +');
  writeln(a22,' = 0');
  A[0][1]:=a11;          A[0][2]:=0;
  A[1][1]:=-2*a16;       A[1][2]:=0;
  A[2][1]:=2*a12+a66;    A[2][2]:=0;
  A[3][1]:=-2*a26;       A[3][2]:=0;
  A[4][1]:=a22;          A[4][2]:=0;
  N:=4;
  HowMany:='All';
  MaxCount:=100;
  RS_Precision:=1E-8;
  For i:=1 to Trunc(N/2) do
     begin
     rf[i][1]:=1;   rf[i][2]:=0;
     sf[i][1]:=1;   sf[i][2]:=0;
     end;
  ComplexRoots(A,N,HowMany,MaxCount,RS_Precision,rf,sf,x,Iter,RootOk);
  For i:=1 to N do
     begin
     CheckOneRoot(A,N,RootOk[i],1E-6,x[i],Value,ZeroOk);
     If ZeroOk=False then
        begin
        writeln('Root ',i,' is no good');
        Halt;
        end;
     end;
  writeln('Roots:');
  For i:=1 to 4 do
     writeln('s',i,' = ',x[i][1]:10:7,' + (',x[i][2]:10:7,')i');
end;
{-------------------------------------------------------------}
```

```
begin
   (El,Et,Nlt,Ntt,Glt,Gtt)
(
   MaterialRoots(47.4E9,16.2E9,0.26,0.35,7E9,1.28E9,'CE 9000','STRAIN');
   MaterialRoots(181E9,10.3E9,0.28,0.35,7.17E9,1.28E9,'T300/5208 Graphite/Epoxy','STRAIN');
   MaterialRoots(11.79E9,5.89E9,0.071,0.3,0.69E9,0.5E9,'Plywood','STRAIN');
   MaterialRoots(1E9,1E9,0.3,0.3,1E9/2/(1+0.3),1E9/2/(1+0.3),'Isotropic','STRAIN');
)
   MaterialRoots(1E9,1E9,0.3,0.3,1E9/2/(1+0.3),1E9/2/(1+0.3),'Isotropic','STRESS');
end.
```

STRESS.PAS

```
{
   Program to plot stress distribution around triangular hole
   with anisotropic material
}
{-------------------------------------------------------------------------}
{$N+}
Uses
   Crt,Graph,WaitKey,Declare,Printer,J_Graph,WhatKey,Math,CompRoot,Nombre;
Const
   Npoint_max = 500;
   Ncon_max = 1000;
Type
   MatrixR_PointxD = array[1..Npoint_max,1..3] of Rtype;
   VectI_Ncon = array[1..Ncon_max] of Integer;
   VectS2_Point = array[1..Npoint_max] of String[2];
   VectC_4 = array[1..4] of Ctype;
{-------------------------------------------------------------------------}
function equation(    e,theta0:Rtype):Rtype;
{
   Here is where we give the equation
}
var
   f: Rtype;
begin
   f:= cos(theta0)-2*e*cos(2*theta0);
   equation:=f;
end;
{-------------------------------------------------------------------------}
procedure Bisect(    a,b,TOL,e: Rtype;
                     Nmax: integer;
                var p: Rtype;
                var Root_ok: Boolean);
{
   Input:
      a,b: minimum and maximum values for root
      Tol: Tolerance on p error
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Nmax: maximum number of iterations
   Output:
      p: root
      Root_ok: True if root is ok
}
var
   i: integer;
   f_a,f_p: Rtype;
begin
   i:=1;
   Root_ok:=False;
   While i<=Nmax do
      begin
```

```
      p := (a+b)/2;
      f_a:=Equation(e,a);
      f_p:=Equation(e,p);
      If ((f_a=0) or ((b-a)/2<TOL)) then
         begin
         i:=Nmax;
         Root_ok:=True;
         end;
      i:=i+1;
      If f_a*f_p>0 then
         a:=p
      else
         b:=p;
      end;
end;
{-------------------------------------------------------------------}
Procedure Get_Theta0(    e: Rtype;
                     var Theta0: Rtype);
{
   Input:
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
   Output:
      Theta0: value of theta so that the boundary of the hole
              is paralel to x axis (4 TERM equation of hole)
}
var
   Root_ok: Boolean;
begin
   Theta0:=0;
   If (1/6<e) and (e<1/3)  then
      begin
      Bisect(105/180*Pi,125/180*Pi,1E-4,e,10000,Theta0,Root_ok);
      If Root_ok=False then
         begin
         writeln('Theta 0 not found');
         halt;
         end;
      end;
end;
{-------------------------------------------------------------------}
Procedure Two(    Title: String80;
              var XYZ: MatrixR_PointxD;        (var to save memory only)
                  Npoint: Integer;
              var Dep,Fin: Vect1_Ncon;
              var Name: VectS2_Point;
                  Ntrait: integer;
                  Rho,D,Theta,Phi: Rtype;
              var Action: String6);
{
   Input:
      Title: Title;
      XYZ[i]: Z Y and Z Coordinates of point i
```

```
      Npoints: number of points
      Dep[i]: starting point # for line i
      Fin[i]: ending point # for line i
      Name[i]: Label for node i
      Ntrait: number of line
      Rho: The distance from the viewer to the origin
      D: Distance between the viewer and intersection plane
      Theta: Angle with X axis of view point in Degrees
      Phi: Angle with Z axis of view point in Degrees
   Output:
      Draw the preceding with Labeling of nodes
      Action: Action taken by user:
              'LEFT','RIGHT','UP','DOWN','ESC'
}
var
   XY: array[1..Npoint_Max,1..2] of Rtype;
   Margin,Ratio,amplitude,s1,c1,s2,c2,xe,ye,ze: Rtype;
   iD,i,GrDriver,GrMode: integer;
   Xasp,Yasp: word;
   a,Max,Min: array[1..2] of Rtype;
   Smax,Smin,Sign: array[1..2] of integer;
   C: Char;
   Mot: String[8];
begin
{
                  Transform from degrees to radians
}
   Theta:=Theta*Pi/180;
   Phi:=Phi*Pi/180;
{
                  Get Graphic Setup
}
   GrDriver:=Detect;
   InitGraph(grDriver,grMode,'');
   Smax[1]:=GetMaxX;
   Smin[1]:=0;
   Smax[2]:=0;
   Smin[2]:=GetMaxY;
   GetAspectRatio(Xasp,Yasp);
   Ratio:=Yasp/Xasp;
   S1:=Sin(Theta) ; C1:=Cos(Theta);
   S2:=Sin(Phi) ; C2:=Cos(Phi);
{
                  Initialize Max Min
}
   For iD:=1 to 2 do
      begin
      Max[iD]:=-1E30;
      Min[iD]:=1E30;
      end;
{
                  Perspective projection
```

```
}
   for i:=1 to Npoint do
     begin
     Xe:=-xyz[i,1]*S1+xyz[i,2]*C1;
     Ye:=-xyz[i,1]*C1*C2-xyz[i,2]*S1*C2+xyz[i,3]*S2;
     Ze:=-xyz[i,1]*S2*C1-xyz[i,2]*S1*S2-xyz[i,3]*C2+Rho;
     XY[i,1]:=(D*Xe/Ze)*Ratio;
     XY[i,2]:=D*Ye/Ze;
{
                 Find Max min
}
     For iD:=1 to 2 do
       begin
       If XY[i,iD]<Min[iD] then Min[iD]:=XY[i,iD];
       If Max[iD]<XY[i,iD] then Max[iD]:=XY[i,iD];
       end;
     end;
{
                 Leave a margin for Max Min
}
   Margin:=0.025*(max[1]-Min[1]);   { X margin } {0.025 for = 2/80 = 2 col.}
   max[1]:=max[1]+Margin;
   min[1]:=min[1]-Margin;
   Margin:=0.03*(max[2]-Min[2]);   { Y margin } {0.03 for > 1/25/2=half a line}
   If Title<>'' then max[2]:=max[2]+Margin+(max[2]-min[2])/20
               else max[2]:=max[2]+Margin;
   min[2]:=min[2]-Margin;
{
                 Find appropriate amplitude (lowest ratio)
                 in order to keep proportion in geometry
                 Take Sign in account
}
   For iD:=1 to 2 do
     a[iD]:=(Smax[iD]-Smin[iD])/(max[iD]-min[iD]);
   If abs(a[1])<=abs(a[2]) then Amplitude:=abs(a[1])
                           else Amplitude:=abs(a[2]);
   For iD:=1 to 2 do
     Sign[iD]:=Round((Smax[iD]-Smin[iD])/abs(Smax[iD]-Smin[iD]));
{
                 Scale and center coordinates on screen
}
   For i:=1 TO Npoint do
     For iD:=1 to 2 do
     XY[i,iD]:=(XY[i,iD]-(max[iD]+min[iD])/2)*Sign[iD]*Amplitude
              +(Smax[iD]+Smin[iD])/2;
{
                 Draw lines
}
   For i:=1 to Ntrait do
     Line(Round(XY[Dep[i],1]),Round(XY[Dep[i],2]),
          Round(XY[Fin[i],1]),Round(XY[Fin[i],2]));
{
```

```
                Get node Labels
}
   For i:=1 to Npoint do
      If Name[i]<>'' then
         Put_Mot(Name[i],XY[i,1],XY[i,2]);
{
                Write Title
}
   If Title<>'' then
      OutTextXY(Round(GetMaxX/2-TextWidth(Title)/2),0,Title);
   Get_Key(C,Action);
   CloseGraph;    (Free heap memory used for graphics)
end;
{-------------------------------------------------------------------}
Procedure CalcZeta(    s1,z1: Ctype;
                       e,Ratio,Radius: Rtype;
                   var r,s: VectC_D;
                   var Zeta1: VectC_4);
{
   Input:
      s1: Root for material (mu1 and mu2 in Lekhnitskii)
      z1: coordinate in conjugate plane
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
      r[i],s[i]: seed for pairs of roots  [i_max=Trunc(Roots/2)]
   Output:
      r[i],s[i]: corrected seed for pairs of roots  [i_max=Trunc(Roots/2)]
      Zeta1[i]: curvilinear coordinate of ith root
}
var
   t,t1,a1,b1,c1,d1: Ctype;
{                  for numerical Zeta roots        }
   RS_Precision,ZeroPrecision: Rtype;
   MaxCount: integer;
   A,x,Value: VectC_D;
   RootOk,ZeroOk: VectBoolean_D;
   Iter: VectI_D;
   i,N: integer;
begin
{
                   Get a1,b1,c1,d1
}
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CAdd(t1,t,a1);
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CSub(t1,t,b1);
   CDiv(a1,b1,c1);
```

```
   t[1]:=-2/e/Radius;    t[2]:=0;
   CDiv(t,b1,d1);
{
                  Solve for Zeta1
}
   a[0][1]:=1;              a[0][2]:=0;              { 1.zeta^4      }
   a[1][1]:=c1[1]/e;        a[1][2]:=c1[2]/e;        { c1/e.zeta^3   }
   Cmult(d1,z1,t);
   a[2][1]:=t[1];           a[2][2]:=t[2];           { d1.z1.zeta^2  }
   a[3][1]:=1/e;            a[3][2]:=0;              { 1/e.zeta      }
   a[4][1]:=c1[1];          a[4][2]:=c1[2];          { c1            }
   N:=4;
{
                  GET ROOTS
}
   MaxCount:=100;
   RS_Precision:=1E-10;
   ComplexRoots(A,N,'AllRoots',MaxCount,RS_Precision,r,s,x,Iter,RootOk);
   For i:=1 to N do
      Zeta1[i]:=x[i];
{
                  VERIFY All ROOTS
}
   ZeroPrecision:=1E-6;
   For i:=1 to N do
      begin
      CheckOneRoot(A,N,RootOk[i],ZeroPrecision,Zeta1[i],Value[i],ZeroOk[i]);
      If ZeroOk[i]=False then
         begin
         writeln('Root ',i,' does not produce zero polynomial');
         halt;
         end;
      end;
end;
{-------------------------------------------------------------------------}
Procedure CalcAllZeta(    s,z: Ctype;
                          e,Ratio,Radius: Rtype;
                      var Zeta_r,Zeta_s: VectC_D;
                      var Zeta,dZeta: VectC_4);
{
   Input:
      s: Root of (1.59) for material
      z: coordinate in conjugate plane
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
      Zeta_r[i],Zeta_s[i]: seed for pairs of roots of Zeta
                           [i_max=Trunc(Roots/2)]
   Output:
      Zeta_r[i],Zeta_s[i]: corrected seed for pairs of roots of Zeta
                           [i_max=Trunc(Roots/2)]
      Zeta[i]: curvilinear coordinate of ith root
```

```
      dZeta[i]: derivate of Zeta1 at z1 of ith root
}
var
   z_2: Ctype;
   Zeta_2: VectC_4;
   Zeta_r_2,Zeta_s_2: VectC_D;
   dreal: Ctype;
   i,Nroot: integer;
begin
{
                  Calculate Zeta1 and
                  Numerically differentiate Zeta1
}
   Nroot:=4;
   CalcZeta(s,z,e,Ratio,Radius,Zeta_r,Zeta_s,Zeta);
   dreal[1]:=1E-8;   dreal[2]:=0;
   CAdd(dreal,z,z_2);
   Zeta_r_2:=Zeta_r;  {speeds up root finding)
   Zeta_s_2:=Zeta_s;
   CalcZeta(s,z_2,e,Ratio,Radius,Zeta_r_2,Zeta_s_2,Zeta_2);
   For i:=1 to Nroot do
      begin
      CSub(Zeta_2[i],Zeta[i],dZeta[i]);
      CDiv(dZeta[i],dreal,dZeta[i]);
      end;
end;
{-------------------------------------------------------------------}
Procedure CalcStress(    X,Y: Rtype;
                         s1,s2: Ctype;
                     var P,e,alpha,Ratio,Radius,lambda,Rho: Rtype; {var for 8087)
                     var Zeta1_r,Zeta1_s,Zeta2_r,Zeta2_s: VectC_D;
                     var Zeta1rootNumber,Zeta2rootNumber: integer;
                     var StressX,StressY,StressXY,
                         StressR,StressT,StressRT: Rtype);
{
   Input:
      X,Y: coordinates of point of interest
      s1,s2: material property roots (mu1 and mu2 in Lekhnitskii's book)
      P:  Stress load
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Alpha: angle for P with x axis
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
      lambda: angle(radians) of rho axis to x axis
      Rho: second coordinate of curvilinear coordinate system
           expressing radius in a unit circle
           used to know if the last root seed must be used when rho<>1
      Zeta1_r[i],Zeta1_s[i]: seed for pairs of roots of Zeta1
                             [i_max=Trunc(Roots/2)]
      Zeta2_r[i],Zeta2_s[i]: seed for pairs of roots of Zeta2
                             [i_max=Trunc(Roots/2)]
      Zeta1rootNumber,Zeta2rootNumber: Last selected root number for
```

```
        Zeta1 and Zeta2. Used when rho<>0
    Output:
      Zeta1_r[i],Zeta1_s[i]: corrected seed for pairs of roots of Zeta1
                             [i_max=Trunc(Roots/2)]
                             seed of selected pair for stress is set as first
      Zeta2_r[i],Zeta2_s[i]: corrected seed for pairs of roots of Zeta2
                             [i_max=Trunc(Roots/2)]
                             seed of selected pair for stress is set as first
      Zeta1rootNumber,Zeta2rootNumber: Last selected root number for
        Zeta1 and Zeta2
      StressX,StressY,StressXY: Stress in X-Y plane
      StressR,StressT,StressRT: Stress in Rho-Theta plane
    Note: in order to calculate at Rho<>1, stresses at Rho=1 must be calculated
          first.  After ofthet values of Rho can be incremented slowly to keep
          the seed for the root from varying sharply.
}
var
   t,ta,tb,t1,t2: Ctype;   {temporary}
   z1,z2: Ctype;
   a1,b1,a2,b2: Ctype;
   cg,sg,ca,sa,Bs,Bps,Cps,al1,al2,be1,be2: Rtype;
   Zeta1,dZeta1,Zeta2,dZeta2: VectC_4;
   k1,k1c,k2,k2c,k5,k5c,k6,k6c: Ctype;
   dx: Ctype;
   dPhi1,dPsi2: array[1..4] of Ctype;
   Sx,Sy,Sxy,Sr,St,Srt: array[1..4,1..4] of Rtype;
   iRoot,i1,i2,j1,j2: integer;
   Start_i1,End_i1,Start_i2,End_i2: integer;
begin
{
                  Get z1,z2
}
   t[1]:=y;      t[2]:=0;
   CMult(s1,t,z1);
   z1[1]:=z1[1]+X;
   t[1]:=y;      t[2]:=0;
   CMult(s2,t,z2);
   z2[1]:=z2[1]+X;
{
                  Get a1,b1,a2,b2
}
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CAdd(t1,t,a1);
   t[1]:=0;    t[2]:=1;
   CMult(t,s1,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CSub(t1,t,b1);
   t[1]:=0;    t[2]:=1;
   CMult(t,s2,t);
   t1[1]:=Ratio;   t1[2]:=0;
```

```
   CAdd(r',t,a2);
   t[1]:=0;    t[2]:=1;
   CMult(t,s2,t);
   t1[1]:=Ratio;   t1[2]:=0;
   CSub(t1,t,b2);
{
                Get Zeta1,dZeta1,Zeta2,dZeta2 for each 4 roots
}
   CalcAllZeta(s1,z1,e,Ratio,Radius,Zeta1_r,Zeta1_s,Zeta1,dZeta1);
   CalcAllZeta(s2,z2,e,Ratio,Radius,Zeta2_r,Zeta2_s,Zeta2,dZeta2);
{
                Get k1,k2,k5,k6 and conjugates
}
   ca:=cos(alpha);
   sa:=sin(alpha);
   al1:=s1[1];
   be1:=s1[2];
   al2:=s2[1];
   be2:=s2[2];
   Bs:=P*(ca*ca+(al2*al2+be2*be2)*sa*sa+al2*sin(2*alpha))
        /2/(sqr(al2-al1)+(be2*be2-be1*be1));
   Bps:=P*((al1*al1-be1*be1-2*al1*al2)*sa*sa-ca*ca-al2*sin(2*alpha))
        /2/(sqr(al2-al1)+(be2*be2-be1*be1));
   Cps:=P*((al1-al2)*ca*ca+(al2*(al1*al1-be1*be1)
           -al1*(al2*al2-be2*be2))*sa*sa
           +((al1*al1-be1*be1)-(al2*al2-be2*be2))*sa*ca)
        /2/be2/(sqr(al2-al1)+(be2*be2-be1*be1));
   t[1]:=Bs;   t[2]:=0;
   CMult(t,a1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,a2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k1);
   t[1]:=Bs;   t[2]:=0;
   CMult(t,b1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,b2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k2);
   t[1]:=Bs;   t[2]:=0;
   CMult(t,s1,t1);
   CMult(t1,a1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,s2,t2);
   CMult(t2,a2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k5);
   t[1]:=Bs;   t[2]:=0;
   CMult(t,s1,t1);
```

```
   CMult(t1,b1,t1);
   t[1]:=Bps;   t[2]:=Cps;
   CMult(t,s2,t2);
   CMult(t2,b2,t2);
   CAdd(t1,t2,t1);
   t[1]:=Radius/2;   t[2]:=0;
   CMult(t,t1,k6);
   CConj(k1,k1c);
   CConj(k2,k2c);
   CConj(k5,k5c);
   CConj(k6,k6c);
{
                  Get derivate of Phi, Psi
}
   For iRoot:=1 to 4 do
      begin
      CAdd(k1,k2c,t1);
      CMult(s2,t1,t1);
      CAdd(k5,k6c,t2);
      CSub(t1,t2,t1);
      CSub(s1,s2,t2);
      CDiv(dZeta1[iRoot],t2,ta);
      CMult(ta,t1,ta);
      CAdd(k2,k1c,t1);
      CMult(s2,t1,t1);
      CAdd(k6,k5c,t2);
      CSub(t1,t2,t1);
      CSub(s1,s2,t2);
      t[1]:=2.0*e;   t[2]:=0;
      CMult(t,Zeta1[iRoot],tb);
      CMult(tb,dZeta1[iRoot],tb);
      CDiv(tb,t2,tb);
      CMult(tb,t1,tb);
      CAdd(ta,tb,dPhi1[iRoot]);
      CAdd(k1,k2c,t1);
      CMult(s1,t1,t1);
      CAdd(k5,k6c,t2);
      CSub(t2,t1,t1);
      CSub(s1,s2,t2);
      CDiv(dZeta2[iRoot],t2,ta);
      CMult(ta,t1,ta);
      CAdd(k2,k1c,t1);
      CMult(s1,t1,t1);
      CAdd(k6,k5c,t2);
      CSub(t2,t1,t1);
      CSub(s1,s2,t2);
      t[1]:=2.0*e;   t[2]:=0;
      CMult(t,Zeta2[iRoot],tb);
      CMult(tb,dZeta2[iRoot],tb);
      CDiv(tb,t2,tb);
      CMult(tb,t1,tb);
      CAdd(ta,tb,dPsi2[iRoot]);
```

```
    end;
{
                Get stresses
                Use last root numbers when rho<>1
                -> Do not select stress as when rho=1
                When rho=1 look for the roots which give (stress)rho=0
                Save root numbers as if for the first pair of roots
                Save seed for roots as first seed
}
  If abs(Rho-1)<0.0001 then
    begin
    Start_i1:=1;   End_i1:=4;
    Start_i2:=1;   End_i2:=4;
    end
  else
    begin
    Start_i1:=Zeta1rootNumber;   End_i1:=Start_i1;
    Start_i2:=Zeta2rootNumber;   End_i2:=Start_i2;
    end;
  StressR:=1E30;
  For i1:=Start_i1 to End_i1 do
    begin
    For i2:=Start_i2 to End_i2 do
      begin
      CMult(s1,s1,t1);
      CMult(t1,dPhi1[i1],t1);
      CMult(s2,s2,t2);
      CMult(t2,dPsi2[i2],t2);
      CAdd(t1,t2,t1);
      Sx[i1,i2] := P*sqr(cos(alpha))+2*t1[1];
      CAdd(dPhi1[i1],dPsi2[i2],t1);
      Sy[i1,i2] := P*sqr(sin(alpha))+2*t1[1];
      CMult(s1,dPhi1[i1],t1);
      CMult(s2,dPsi2[i2],t2);
      CAdd(t1,t2,t1);
      Sxy[i1,i2] := P*sin(alpha)*cos(alpha)-2*t1[1];
{
                Transform stress into curvilinear coordinate system
}
      cg:=cos(lambda);
      sg:=sin(lambda);
      Sr[i1,i2]:=cg*cg*Sx[i1,i2]+sg*sg*Sy[i1,i2]+2*cg*sg*Sxy[i1,i2];
      St[i1,i2]:=sg*sg*Sx[i1,i2]+cg*cg*Sy[i1,i2]-2*cg*sg*Sxy[i1,i2];
      Srt[i1,i2]:=-cg*sg*Sx[i1,i2]+cg*sg*Sy[i1,i2]+(cg*sg-sg*sg)*Sxy[i1,i2];
{
                Retain lowest Stress in direction Rho
}
      If Abs(Sr[i1,i2])<Abs(StressR) then
        begin
        StressR:=Sr[i1,i2];
        j1:=i1;
        j2:=i2;
```

```
            end;
          end;
        end;
    StressX  := Sx[j1,j2];
    StressY  := Sy[j1,j2];
    StressXY := Sxy[j1,j2];
    StressR  := Sr[j1,j2];
    StressT  := St[j1,j2];
    StressRT := Srt[j1,j2];
{
                Save seed for appropriate roots as first seed
                even when rho<>1 to permit gradual variation
}
    Zeta1_r[1]:=Zeta1_r[Trunc((j1-1)/2)+1];
    Zeta1_s[1]:=Zeta1_s[Trunc((j1-1)/2)+1];
    Zeta2_r[1]:=Zeta2_r[Trunc((j2-1)/2)+1];
    Zeta2_s[1]:=Zeta2_s[Trunc((j2-1)/2)+1];
{
                root number to use for 1<rho with zeta1 and zeta2
}
    Zeta1rootNumber:=j1-Trunc((j1-1)/2)*2;
    Zeta2rootNumber:=j2-Trunc((j2-1)/2)*2;
{
    Use this to see the selected seed for the roots, root,
    and root number for selected seed as if it was the first pair of root
    writeln(Lst,'r1=',Zeta1_r[1][1]:10:7,'+',Zeta1_r[1][2]:10:7,'i ',
                's1=',Zeta1_s[1][1]:10:7,'+',Zeta1_s[1][2]:10:7,'i ',
                'zeta1=',Zeta1[j1][1]:10:7,'+',Zeta1[j1][2]:10:7,'i ',
                'root#=',Zeta1rootNumber);
    writeln(Lst,'r2=',Zeta2_r[1][1]:10:7,'+',Zeta2_r[1][2]:10:7,'i ',
                's2=',Zeta2_s[1][1]:10:7,'+',Zeta2_s[1][2]:10:7,'i ',
                'zeta2=',Zeta2[j2][1]:10:7,'+',Zeta2[j2][2]:10:7,'i ',
                'root#=',Zeta2rootNumber);
}
end;
{-------------------------------------------------------------------------}
Procedure GetXY(    Rho,Theta,e,Ratio,Radius: Rtype;
                var X,Y,lambda: Rtype);
{
    4 TERM FORMULA
    Input:
      Rho,Theta: curvilinear coordinates
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
    Output:
      X,Y: Plane coordinates
      lambda: angle(radians) of rho axis to x axis
}
var
    a,b,rho2,x2,y2: Rtype;
begin
```

```
   a:=ratio+1;
   b:=Ratio-1;
   x:=Radius/2*((a/rho+b*rho)*cos(theta)+e*(a*rho*rho+b/rho/rho)*cos(2*theta));
   y:=Radius/2*((-a/rho+b*rho)*sin(theta)+e*(a*rho*rho-b/rho/rho)*sin(2*theta));
{
                get angle of theta axis
}
   rho2:=rho+0.00001;
   x2:=Radius/2*((a/rho2+b*rho2)*cos(theta)+e*(a*rho2*rho2+b/rho2/rho2)*cos(2*theta));
   y2:=Radius/2*((-a/rho2+b*rho2)*sin(theta)+e*(a*rho2*rho2-b/rho2/rho2)*sin(2*theta));
   lambda:=ArcTan4(x2-x,y2-y);
   If Pi<lambda then lambda:=lambda-2*Pi;
end;
{-------------------------------------------------------------------}
Procedure DrawStress(    Material: String80;
                         s1,s2: Ctype;
                         P,e,alpha,Ratio,Radius: Rtype;
                         StepDegree: Rtype;
                         UseOutput: String80;
                         WhichStress: Byte;
                         LotusDirectory: String80);
{
   Input:
      Material: Name of Material
      s1,s2: material property roots
      P: Stress load
      e: blunt factor: 0 for a circle, 1/3 (max) for sharp triangle
      Alpha: (radians) angle for P with x axis
      Ratio: length/height*2/sqrt(3) of triangle
      Radius: Radius of unit circle
      StepDegree: step for angle (degrees)
      UseOutput: 'PRINTER' if output results along rho=1 to printer
                 'GRAPH' if plot is desired
                 'LOTUS' if Lotus 123 output is desired
      WhichStress: 1 for StressX
                   2     StressY
                   3     StressXY
                   4     StressTheta
                   5     StressRho
                   6     StressThetaRho
      LotusDirectory: directory for lotus (with last \)
   Output:
      Printer Listing of stresses
      or
      Stress Plot
}
var
   Title: String80;
   X,Y: Rtype;
   LastDegree,Degree,Rho,Theta,Theta0,lambda: Rtype;
   StressX,StressY,StressXY: Rtype;
   StressR,StressT,StressRT: Rtype;
```

```
   iPair,j,iPoint,ToContour,Contour: integer;
   R: array[1..10] of Rtype;
   Stress: array[1..6] of Rtype;
   Zeta1_r,Zeta1_s,Zeta2_r,Zeta2_s: VectC_D;
   Zeta1rootNumber,Zeta2rootNumber: integer;
   F: text;
   Fname: String[80];
{
                  Graph variables
}
   XYZ: MatrixR_PointxD;
   iStress,NpointBefore,Npoint: Integer;
   Dep,Fin: VectI_Ncon;
   Name: VectS2_Point;
   Ntrait: integer;
   Action: String6;
   StoreZ: array[1..6,1..250] of Rtype;
   Scale,MaxZ,MinZ: Rtype;
begin
   ClrScr;
   If UseOutput='LOTUS' then
      begin
      Repeat
         ClrScr;
         gotoXY(1,10);
         writeln('Previous file with same name will be erased');
         write('Lotus file name (no extension .PRN): ',LotusDirectory);
         readln(Fname);
         until Pos('.',Fname)=0;
      Assign(F,LotusDirectory+Fname+'.PRN');
      Rewrite(F);
      end;
   Get_Theta0(e,Theta0);
{
                  Display initial data, theta=0
}
   R[1]:=1;
   R[2]:=0.95;
   R[3]:=0.9;
   R[4]:=0.85;
   R[5]:=0.8;
   R[6]:=0.75;
   R[7]:=0.7;
   For iPoint:=1 to Npoint_max do Name[iPoint]:='';
   NpointBefore:=0;
   Npoint:=0;
   Ntrait:=0;
{
                  Axis
}
   XYZ[1][1]:=0;    XYZ[1][2]:=0;    XYZ[1][3]:=0;   Name[1]:='O';
   XYZ[2][1]:=2.3;  XYZ[2][2]:=0;    XYZ[2][3]:=0;   Name[2]:='+X';
```

```
   XYZ[3][1]:=0;    XYZ[3][2]:=-2;    XYZ[3][3]:=0;    Name[3]:='-Y';
   XYZ[4][1]:=0;    XYZ[4][2]:=0;     XYZ[4][3]:=2;    Name[4]:='+S';
   Npoint:=4;
   NpointBefore:=4;
   Dep[1]:=1;  Fin[1]:=2;
   Dep[2]:=1;  Fin[2]:=3;
   Dep[3]:=1;  Fin[3]:=4;
   Ntrait:=3;
{
                Calculate stresses
                so that (stress)rho=0
}
   If UseOutput='PRINTER' then ToContour:=1;
   If UseOutput='GRAPH'   then ToContour:=5;
   If UseOutput='LOTUS'   then ToContour:=1;
   If UseOutput='PRINTER' then
      begin
      writeln(Lst,'Material: ',Material);
      writeln(Lst,'s1 = ',s1[1]:7:5,'+(',s1[2]:7:5,')i');
      writeln(Lst,'s2 = ',s2[1]:7:5,'+(',s2[2]:7:5,')i');
      writeln(Lst,'P (Pa) = ',P:5:2);
      writeln(Lst,'epsilon = ',e:6:4);
      writeln(Lst,'alpha (degrees)= ',alpha*180/Pi:8:3);
      writeln(Lst,'L/h = ',Ratio*Sqrt(3)/2:7:4,'  r = ',Ratio:7:4);
      writeln(Lst,'Radius of unit circle= ',Radius:7:4);
      writeln(Lst);
      write(Lst,chr(27),'M',chr(13));
      writeln(Lst,'  Theta      X        Y       lambda      Sx        Sy        Sxy        Sr
St       Srt');
      writeln(Lst,'(degree)    (m)      (m)      (deg)      (Pa)      (Pa)      (Pa)       (Pa)       (
Pa)      (Pa)');
      end;
   If UseOutput='LOTUS' then
      begin
      writeln(F,'"Material: ',Material);
      writeln(F,'"s1 = ',Rfield(s1[1],7),'+(',Rfield(s1[2],7),')i');
      writeln(F,'"s2 = ',Rfield(s2[1],7),'+(',Rfield(s2[2],7),')i');
      writeln(F,'"P (Pa) = ',Rfield(P,5));
      writeln(F,'"epsilon = ',Rfield(e,6));
      writeln(F,'"alpha (degrees)= ',Rfield(alpha*180/Pi,8));
      writeln(F,'"L/h = ',Rfield(Ratio*Sqrt(3)/2,7),'  r = ',Rfield(Ratio,7));
      writeln(F,'"Radius of unit circle= ',Rfield(Radius,7));
      writeln(F);
      writeln(F,'"  Theta      X        Y       lambda     Sx        Sy        Sxy       Sr        St
     Srt');
      writeln(F,'"(degree)    (m)      (m)      (deg)      (Pa)      (Pa)      (Pa)      (Pa)      (Pa)
     (Pa)');
      end;
{
                Initialise seed for roots
}
   For iPair:=1 to 4 do
```

```
      begin
      Zeta1_r[iPair][1]:=1;   Zeta1_r[iPair][2]:=0;
      Zeta1_s[iPair][1]:=1;   Zeta1_s[iPair][2]:=0;
      Zeta2_r[iPair][1]:=1;   Zeta2_r[iPair][2]:=0;
      Zeta2_s[iPair][1]:=1;   Zeta2_s[iPair][2]:=0;
      end;
   Zeta1RootNumber:=1;
   Zeta2RootNumber:=1;
   Degree:=0;
   While Degree<=180.001 do
      begin
      Theta:=Degree/180*Pi;
      For Contour:=1 to ToContour do
         begin
         gotoxy(1,12);
         write('Angle = ',Degree:5:3,'    Contour: ',Contour,'    ');
         Rho:=R[Contour];
         GetXY(Rho,Theta,e,Ratio,Radius,X,Y,lambda);
         CalcStress(X,Y,s1,s2,P,e,alpha,Ratio,Radius,lambda,Rho,
                    Zeta1_r,Zeta1_s,Zeta2_r,Zeta2_s,
                    Zeta1rootNumber,Zeta2rootNumber,
                    StressX,StressY,StressXY,StressR,Stress ,StressRT);
         Stress[1]:=StressX;
         Stress[2]:=StressY;
         Stress[3]:=StressXY;
         Stress[4]:=StressR;
         Stress[5]:=StressT;
         Stress[6]:=StressRT;
{
                  Store all elevations
}
         For iStress:=1 to 6 do
            StoreZ[iStress][Round((Npoint+2-4)/2)]:=Stress[iStress];
         If UseOutput='PRINTER' then
            begin
{
                  Highlight result for Theta0
}
            If 0.001<Abs(Degree-Round(Degree/StepDegree)*StepDegree) then
               begin
               Write(Lst,Theta*180/Pi:8:4,Chr(13));
               Write(Lst,Theta*180/Pi:8:4,Chr(13));
               write(Lst,Theta*180/Pi:8:4,' ',
                         X:7:4,' ',
                         Y:7:4,' ',
                         lambda*180/Pi:8:3,' ',
                         Stress[1]:9:5,' ',
                         Stress[2]:9:5,' ',
                         Stress[3]:9:5,' ',
                         Stress[4]:9:5,' ',
                         Stress[5]:9:5,' ',
                         Stress[6]:9:5,Chr(13));
```

```
            end;
          writeln(Lst,Theta*180/Pi:8:4,' ',
                      X:7:4,' ',
                      Y:7:4,' ',
                      lambda*180/Pi:8:3,' ',
                      Stress[1]:9:5,' ',
                      Stress[2]:9:5,' ',
                      Stress[3]:9:5,' ',
                      Stress[4]:9:5,' ',
                      Stress[5]:9:5,' ',
                      Stress[6]:9:5);
          end;
        If UseOutput='LOTUS' then
          begin
          writeln(F,Rfield(Theta*180/Pi,10),' ',
                    Rfield(X,10),' ',
                    Rfield(Y,10),' ',
                    Rfield(lambda*180/Pi,10),' ',
                    Rfield(Stress[1],11),' ',
                    Rfield(Stress[2],11),' ',
                    Rfield(Stress[3],11),' ',
                    Rfield(Stress[4],11),' ',
                    Rfield(Stress[5],11),' ',
                    Rfield(Stress[6],11));
{ For Thesis
          writeln(F,Theta*180/Pi:3:0,' ',
                    Stress[1]:6:3,' ',
                    Stress[2]:6:3,' ',
                    Stress[3]:6:3,' ',
                    Stress[4]:6:3,' ',
                    Stress[5]:6:3,' ',
                    Stress[6]:6:3);
}
          end;
        If UseOutput='GRAPH' then
          begin
          If NpointBefore<Npoint then
            begin
            Dep[Ntrait+1]:=Npoint;   {last top}
            Fin[Ntrait+1]:=Npoint+2; {this top}
            Inc(Ntrait);
            end;
          Dep[Ntrait+1]:=Npoint+1;    {this bottom}
          Fin[Ntrait+1]:=Npoint+2;    {this top}
          Inc(Ntrait);
          Inc(Npoint);
          XYZ[Npoint][1]:=x;    {this bottom}
          XYZ[Npoint][2]:=y;
          XYZ[Npoint][3]:=0;
          Inc(Npoint);
          XYZ[Npoint][1]:=x;    {this top}
          XYZ[Npoint][2]:=y;
```

```
            XYZ[Npoint][3]:=Stress[WhichStress];
            end;
         end;
{
                  Draw lines with previous contour
}
      If UseOutput='GRAPH' then
         begin
         If 0<Degree then
            begin
            For j:=0 to Round((Npoint-NpointBefore)/2)-1 do
               begin
               Inc(Ntrait);
               Dep[Ntrait]:=NpointBefore-j*2;
               Fin[Ntrait]:=Npoint-j*2;
               end;
            end;
         NpointBefore:=Npoint;
         end;
      LastDegree:=Degree;
      If ((UseOutput='PRINTER') or (UseOutput='LOTUS'))
            and (Degree<Theta0*180/Pi)
            and (Theta0*180/Pi<=Degree+StepDegree) then
         Degree:=Theta0*180/Pi
      else
         Degree:=Trunc(Degree/StepDegree)*StepDegree+StepDegree;
      If abs(Degree-LastDegree)<0.0001 then Degree:=Degree+StepDegree;
      end;
   If UseOutput='PRINTER' then
      begin
      write(Lst,Chr(12),Chr(13));
      write(Lst,chr(27),'P',chr(13));
      end;
   If UseOutput='LOTUS' then
      Close(F);
   If UseOutput='GRAPH' then
      begin
      ClrScr;
      gotoXY(1,10);
      writeln('Use <Up> <Down> <Home> <End> to change from stress plot:');
      writeln('   -> Stress: X  Y  XY  Rho  Theta  RhoTheta');
      writeln('Press <Esc> to end after first plot');
      wait;
      If Npoint_max<Npoint then Npoint:=Npoint_max;
      If Ncon_max<Ntrait then Ntrait:=Ncon_max;
{
                  Scale Z coordinate
}
      iStress:=WhichStress;
      Repeat
         For j:=1 to Round((Npoint-4)/2) do
            XYZ[4+2*j][3]:=StoreZ[iStress][j];
```

```
        MaxZ:=-1E30;
        MinZ:=+1E30;
        For j:=5 to Npoint do
           begin
           If maxZ<XYZ[j][3] then maxZ:=XYZ[j][3];
           If XYZ[j][3]<minZ then minZ:=XYZ[j][3];
           end;
        Scale:=abs(XYZ[4][3])/(maxZ-minZ);   {use z axis to scale}
        For j:=5 to Npoint do
           XYZ[j][3]:=XYZ[j][3]*Scale;
        Case iStress of
           1: Title:='Stress X';
           2: Title:='Stress Y';
           3: Title:='Stress XY';
           4: Title:='Stress Rho';
           5: Title:='Stress Theta';
           6: Title:='Stress RhoTheta';
           end;
        Two(Title,XYZ,Npoint,Dep,Fin,Name,Ntrait,1000,1000,-115,30,Action);
        If Action='UP' then iStress:=iStress-1;
        If Action='DOWN' then iStress:=iStress+1;
        If Action='HOME' then iStress:=1;
        If Action='END' then iStress:=6;
        If iStress=-1 then iStress:=6;
        If iStress=7 then iStress:=1;
        until Action='ESC';
      end;
end;
{-------------------------------------------------------------------------}
var
   s1,s2: Ctype;
   StepDegree,e,L,h,P,alpha,Ratio,Radius: Rtype;
   WhichStress: integer;
   LotusDirectory,UseOutput,Material: String80;
   C: Char;
   Action: String6;
begin
{
   Material:='Isotropic, plane STRESS or STRAIN';
   s1[1]:=0;
   s1[2]:=1.01;
   s2[1]:=0;
   s2[2]:=1;
   Material:='Plywood, plane STRAIN';
   s1[1]:=0;
   s1[2]:=4.1032;
   s2[1]:=0;
   s2[2]:=0.3293;
   Material:='T300/5208 Graphite/Epoxy, plane STRAIN';
   s1[1]:=0;
   s1[2]:=4.8940;
   s2[1]:=0;
```

```
   s2[2]:=0.8042;
   Material:='CE 9000 Glass/Epoxy, plane STRAIN';
   s1[1]:=0;
   s1[2]:=2.3992;
   s2[1]:=0;
   s2[2]:=0.6757;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
   Material:='CE 9000 Glass/Epoxy, plane STRAIN';
   s1[1]:=0;
   s1[2]:=2.3992;
   s2[1]:=0;
   s2[2]:=0.6757;
   P:=1;
   e:=1/3;
   alpha:=0*Pi/180;  (radians)
(          0.8660254 for equilateral triangle -> r=1)
   L:=0.8660254;
   h:=1;
   Ratio:=2/sqrt(3)*L/h;
   Radius:=1;
(                  7.5 degrees -> 112.5, 180)
(                  7.2 degrees -> 115.2, 180)
   StepDegree:=7.2;      ( positive only )
   WhichStress:=1;
   ClrScr;
   writeln('Material: ',Material);
   writeln('s1 = ',s1[1]:7:5,'+(',s1[2]:7:5,')i');
   writeln('s2 = ',s2[1]:7:5,'+(',s2[2]:7:5,')i');
   writeln('P (Pa) = ',P:5:2);
   writeln('epsilon = ',e:6:4);
   writeln('alpha (degrees)= ',alpha*180/Pi:8:3);
   writeln('L/h = ',Ratio*Sqrt(3)/2:7:4,'  r = ',Ratio:7:4);
   writeln('Radius of unit circle= ',Radius:7:4);
   writeln;
   writeln('Which Output?');
   writeln('   Graph');
   writeln('   Printer');
   writeln('   Lotus');
   Get_Key(C,Action);
   C:=upCase(C);
   Case C of
      'G': UseOutput:='GRAPH';
      'P': UseOutput:='PRINTER';
      'L': UseOutput:='LOTUS';
      end;
   LotusDirectory:='C:\Lotus\Files\';
   If C in ['G','P','L'] then
      DrawStress(Material,s1,s2,P,e,alpha,Ratio,Radius,
                 StepDegree,UseOutput,WhichStress,LotusDirectory);
end.
```

TRIANGLE.PAS

```
{$N+}
Uses
   Declare,Graph,Crt,J_Graph,Math,WhatKey,WaitKey;
Const
   Npoint_max = 300;
   Ncon_max = 300;
Type
   MatrixR_PointxD = array[1..Npoint_max,1..3] of Rtype;
   VectI_Ncon = array[1..Ncon_max] of Integer;
   VectS5_Point = array[1..Npoint_max] of String[5];
{-------------------------------------------------------------------}
Procedure P3_Equat(    x1,x2,x3,y1,y2,y3: Rtype;
                   var A,B,C: Rtype);
{
  Finds the equation (y=Axx+Bx+C) passing through (x1,y1) (x2,y2) (x3,y3)
  Input:
     x1,x2,x3,y1,y2,y3: coordinates of three points
  Output:
     A,B,C: coefficients of  y=Axx+Bx+C
                                    Written by Jerome Daoust in Fall '87
}
begin
{
       check that x1<>x2<>x3 before getting A,B,C
}
   If ((x1<>x2) and (x2<>x3)) then
      begin
      B:= ((y1-y2)/(x1*x1-x2*x2)-(y1-y3)/(x1*x1-x3*x3))
         / ((x1-x2)/(x1*x1-x2*x2)-(x1-x3)/(x1*x1-x3*x3));
      A:= ((y1-y2)-B*(x1-x2)) / (x1*x1-x2*x2);
      C:= y1-A*x1*x1-B*x1;
      end
   else
      begin
      A:=0;
      B:=0;
      C:=0;.
      end;
end;
{-------------------------------------------------------------------}
Procedure Two(    Title: String80;
              var XYZ: MatrixR_PointxD;        (var to save memory only)
                  Npoint: Integer;
              var Dep,Fin: VectI_Ncon;
              var Name: VectS5_Point;
                  Ntrait: integer;
                  Rho,D,Theta,Phi: Rtype;
              var Action: String6);
{
   Input:
```

```
      Title: Title;
      XYZ[i]: Z Y and Z Coordinates of point i
      Npoints: number of points
      Dep[i]: starting point # for line i
      Fin[i]: ending point # for line i
      Name[i]: Label for node i
      Ntrait: number of line
      Rho: The distance from the viewer to the origin
      D: Distance between the viewer and intersection plane
      Theta: Angle with X axis of view point in Degrees
      Phi: Angle with Z axis of view point in Degrees
   Output:
      Draw the preceding with Labeling of nodes
      Action: Action taken by user:
              'LEFT','RIGHT','UP','DOWN','ESC'
}
var
   XY: array[1..Npoint_Max,1..2] of Rtype;
   Margin,Ratio,amplitude,s1,c1,s2,c2,xe,ye,ze: Rtype;
   iD,i,GrDriver,GrMode: integer;
   Xasp,Yasp: word;
   a,Max,Min: array[1..2] of Rtype;
   Smax,Smin,Sign: array[1..2] of integer;
   C: Char;
   Mot: String[8];
begin
{
                  Transform from degrees to radians
}
   Theta:=Theta*Pi/180;
   Phi:=Phi*Pi/180;
{
                  Get Graphic Setup
}
   GrDriver:=Detect;
   InitGraph(grDriver,grMode,'');
   Smax[1]:=GetMaxX;
   Smin[1]:=0;
   Smax[2]:=0;
   Smin[2]:=GetMaxY;
   GetAspectRatio(Xasp,Yasp);
   Ratio:=Yasp/Xasp;
   S1:=Sin(Theta) ; C1:=Cos(Theta);
   S2:=Sin(Phi) ; C2:=Cos(Phi);
{
                  Initialize Max Min
}
   For iD:=1 to 2 do
      begin
      Max[iD]:=-1E30;
      Min[iD]:=1E30;
      end;
```

```
{
                Perspective projection
}
   for i:=1 to Npoint do
      begin
      Xe:=-xyz[i,1]*S1+xyz[i,2]*C1;
      Ye:=-xyz[i,1]*C1*C2-xyz[i,2]*S1*C2+xyz[i,3]*S2;
      Ze:=-xyz[i,1]*S2*C1-xyz[i,2]*S1*S2-xyz[i,3]*C2+Rho;
      XY[i,1]:=(D*Xe/Ze)*Ratio;
      XY[i,2]:=D*Ye/Ze;
{
                Find Max min
}
      For iD:=1 to 2 do
         begin
         If XY[i,iD]<Min[iD] then Min[iD]:=XY[i,iD];
         If Max[iD]<XY[i,iD] then Max[iD]:=XY[i,iD];
         end;
      end;
{
                Leave a margin for Max Min
}
   Margin:=0.025*(max[1]-Min[1]);   { X margin } (0.025 for = 2/80 = 2 col.)
   max[1]:=max[1]+Margin;
   min[1]:=min[1]-Margin;
   Margin:=0.03*(max[2]-Min[2]);   { Y margin } (0.03 for > 1/25/2=half a line)
   If Title<>'' then max[2]:=max[2]+Margin+(max[2]-min[2])/20
               else max[2]:=max[2]+Margin;
   min[2]:=min[2]-Margin;
{
                Find appropriate amplitude (lowest ratio)
                in order to keep proportion in geometry
                Take Sign in account
}
   For iD:=1 to 2 do
      a[iD]:=(Smax[iD]-Smin[iD])/(max[iD]-min[iD]);
   If abs(a[1])<=abs(a[2]) then Amplitude:=abs(a[1])
                           else Amplitude:=abs(a[2]);
   For iD:=1 to 2 do
      Sign[iD]:=Round((Smax[iD]-Smin[iD])/abs(Smax[iD]-Smin[iD]));
{
                Scale and center coordinates on screen
}
   For i:=1 TO Npoint do
      For iD:=1 to 2 do
      XY[i,iD]:=(XY[i,iD]-(max[iD]+min[iD])/2)*Sign[iD]*Amplitude
              +(Smax[iD]+Smin[iD])/2;
{
                Draw lines
}
   For i:=1 to Ntrait do
      Line(Round(XY[Dep[i],1]),Round(XY[Dep[i],2]),
```

```
          Round(XY[Fin[i],1]),Round(XY[Fin[i],2]));
{
                  Get node Labels
}
   For i:=1 to Npoint do
      If Name[i]<>'' then
         Put_Mot(Name[i],XY[i,1],XY[i,2]);
{
                  Write Title
}
   If Title<>'' then
      OutTextXY(Round(GetMaxX/2-TextWidth(Title)/2),0,Title);
   Get_Key(C,Action);
   CloseGraph;    {Free heap memory used for graphics}
end;
{-------------------------------------------------------------}
Procedure ExactTriangle;
var
   i,Npoint,Ntrait,Inter: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
   x,y,Xc,Yc: Rtype;
   e,maxX,maxY,minX,minY,Ratio,theta,L,h: Rtype;
begin
   ClrScr;
   e:=1/3;
   L:=1;
   h:=1;
   Ratio:=2/sqrt(3)*L/h;
   MinX:= 1E30;
   MaxX:=-1E30;
   MinY:= 1E30;
   MaxY:=-1E30;
   For i:=0 to 60 do
      begin
      theta:=i*2*Pi/60;
      x:=Ratio*(cos(theta)+e*(cos(2*theta)+1/15*cos(5*theta)+1/54*cos(8*theta)+7/891*cos(11*theta
)));
      y:=      -sin(theta)+e*(sin(2*theta)+1/15*sin(5*theta)+1/54*sin(8*theta)+7/891*sin(11*theta
));
      xyz[i+1,1]:=x;
      xyz[i+1,2]:=y;
      xyz[i+1,3]:=0;
      If x<MinX then MinX:=x;
      If MaxX<x then MaxX:=x;
      If y<Miny then Miny:=y;
      If Maxy<y then Maxy:=y;
      end;
   writeln('ratio = ',(maxX-MinX)/(maxY-MinY));
   wait;
```

```
   Npoint:=61;
   For i:=1 to 60 do
      begin
      Dep[i]:=i;
      Fin[i]:=i+1;
      end;
   Ntrait:=60;
   For i:=1 to 61 do
      name[i]:='';
   Two('Triangle',XYZ,Npoint,Dep,Fin,Name,Ntrait,
       1000,1000,-90,0,Action);
end;
{-------------------------------------------------------------------------}
var
   x,y,L,h,Ratio,dt,t,x1,y1,x2,y2,x3,y3,a,b,c: Rtype;
   i: integer;
begin
{
   dt:=Pi*2/10000;
clrscr;
For i:=1 to 23 do
   begin
   L:=1;
   h:=L*i/2/10;
   e:=1/3;
   Ratio:=2/sqrt(3)*L/h;
   t:=0;
   x1:=Ratio*(cos(t)+e(*cos(2*t)+1/15*cos(5*t)+1/54*cos(8*t)+7/891*cos(11*t)));
   y1:=      -sin(t)+e(*sin(2*t)+1/15*sin(5*t)+1/54*sin(8*t)+7/891*sin(11*t));
   t:=0-dt;
   x2:=Ratio*(cos(t)+e(*cos(2*t)+1/15*cos(5*t)+1/54*cos(8*t)+7/891*cos(11*t)));
   y2:=      -sin(t)+e(*sin(2*t)+1/15*sin(5*t)+1/54*sin(8*t)+7/891*sin(11*t));
   t:=0+dt;
   x3:=Ratio*(cos(t)+e(*cos(2*t)+1/15*cos(5*t)+1/54*cos(8*t)+7/891*cos(11*t)));
   y3:=      -sin(t)+e(*sin(2*t)+1/15*sin(5*t)+1/54*sin(8*t)+7/891*sin(11*t));
   P3_Equat(y1,y2,y3,x1,x2,x3,A,B,C);
   writeln('h/L = ',h/L:8:4,'  radius at tip = (1/',-2*A:8:2,')R');
   end;
halt;
}
   ExactTriangle;
end.
```

WAITKEY.PAS

```
{$N+}
Unit WaitKey;
Interface
Uses
   Crt;
procedure Wait;
procedure WaitIfNo;
Implementation
{-------------------------------------------------------------------}
procedure Wait;
{
                  waits until a key is pressed
                  a Key previously pressed is ignored
}
var
   Ch: char;
begin
   While KeyPressed do   { Empty keyboard buffer }
      Ch:=ReadKey;
   Ch:=ReadKey;          { keyboard buffer is left empty }
end;
{-------------------------------------------------------------------}
procedure WaitIfNo;
{
                  waits until a key is pressed
                  if none were previously pressed
}
var
   Ch: char;
begin
   Ch:=ReadKey;
   While KeyPressed do   { Empty keyboard buffer }
      Ch:=ReadKey;
end;
{-------------------------------------------------------------------}
end.
```

WHATKEY.PAS

```
{$N+}
Unit WhatKey;
Interface
Uses
   Crt,Declare;
procedure get_key(var C: Char;
                  var action: String6);
Implementation
{-------------------------------------------------------------------
                   Taken from \TP\INC\GET_KEY.INC
-------------------------------------------------------------------}
procedure get_key;
{
procedure get_key(var C: Char;
                  var action: String6);
}
{
   output:
      C<>chr(0) or Action<>''
      C: chr(0) if no character pressed
         else: [' '..'~','c','a','a','e','e','e','e',
               'i','i','o','u','u','u']  that include A..Z a..z 0..9
      Action: '' if no action key was pressed
              else: (UP,DOWN,RIGHT,LEFT,PGDN,PGUP,HOME,END,INS,DEL)
                    (CHOME,CEND,CPGUP,CPGDN,CLEFT,CRIGHT)
                    (ENTER,BS,ESC)
                    (F1...F10, SF1..SF10, CF1...CF10, AF1...AF10)
}
var
   Ch1,Ch2,Ch3,Char_0,Char_Esc: char;
begin
   Char_0:=Chr(0);
   Char_Esc:=Chr(27);
   Repeat
      C:=Char_0;
      Action:='';
      Ch1:=Char_0;
      Ch2:=Char_0;
      Ch3:=Char_0;
{
                    Empty keyboard buffer
}
      While Keypressed do Ch1:=ReadKey;
      Ch1:=ReadKey;
{
                    get action
}
      if Ch1=Char_0 then
         begin
         Ch2:=readKey;
```

```
          case Ch2 of
             'H': action:='UP';
             'P': action:='DOWN';
             'M': action:='RIGHT';
             'K': action:='LEFT';
             'G': action:='HOME';
             'O': action:='END';
             'Q': action:='PGDN';
             'I': action:='PGUP';
             'R': action:='INS';
             'S': action:='DEL';
             't': action:='CRIGHT';
             's': action:='CLEFT';
             'w': action:='CHOME';
             'u': action:='CEND';
             'a': action:='CPGUP';
             'v': action:='CPGDN';
             end;
{
                   F1..F10
}
          If ord(Ch2) in [59..68] then
             begin
             str(ord(Ch2)-58,action);
             action:='F'+action;
             end;
{
                   SF1..SF10
}
          If ord(Ch2) in [84..93] then
             begin
             str(ord(Ch2)-83,action);
             action:='SF'+action;
             end;
{
                   CF1..CF10
}
          If ord(Ch2) in [94..103] then
             begin
             str(ord(Ch2)-93,action);
             action:='CF'+action;
             end;
{
                   AF1..AF10
}
          If ord(Ch2) in [104..113] then
             begin
             str(ord(Ch2)-103,action);
             action:='AF'+action;
             end;
          end;
{
```

```
                Get single character actions
}
      If Ch2=Char_0 then
        case ord(Ch1) of
           8: action:='BS';
           13: action:='ENTER';
           27: action:='ESC';
           end;
      if (Ch1 in [' '..'~','c','a','a','e','e','e','e','i','i','o','u','u','u'])
        and (Ch2=Char_0) then C:=Ch1;
      until ((C<>Char_0) or (Action<>''));
end;
{-------------------------------------------------------------}
end.
```

APPENDIX B

LISTING OF FINITE ELEMENT PROGRAM

The finite element program, named *DIRECT*®, consists of many sub-unit programs. Units are listed alphabetically. As you will notice, the element formulation only represents 5% of the programming work.

ADDTOEND.PAS

```
{-------------------------------------------------------------------
   Program to append a command to the end of a text file
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 1024,0,0}
Uses
   Crt,Declare,Where,Sonore,WaitKey,File_RW;
{-------------------------------------------------------------------}
var
   Mot,FileName,AppendString,WorstElementString: String[255];
   FileToAppend: Text;
   i,iPar: Word;
   Bad,RemoveWorstElement: Boolean;
   WorstElementNew,WorstElementOld: Itype;
   F_ElemOld,F_BadElem: FileI;
begin
   Bad:=False;
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
{
                  Get file name in upper case
}
   FileName := G_Dir+ParamStr(1)+'.GEN';
   For i:=1 to Length(FileName) do
      FileName[i]:=UpCase(FileName[i]);
{
                  open file for appending
}
   Assign(FileToAppend,FileName);
   {$I-}
   Append(FileToAppend);
   {$I+}
   If IoResult<>0 then
      begin
      Writeln('   File ',FileName,' does not exist');
      Bad_Beep;
      Wait;
      end
   else
      begin
{
                  Look for REMOVEWORSTELEMENT parameter
                  -> add Edel (worst element number); to given file name
}
      RemoveWorstElement:=False;
      iPar:=2;
      While iPar<=ParamCount do
         begin
         i:=1;
         RemoveWorstElement:=True;
```

```
        Mot:=ParamStr(iPar);
        While i<=Length(ParamStr(iPar)) do
           begin
           If UpCase(Mot[i])<>Copy('REMOVEWORSTELEMENT',i,1) then
              begin
              RemoveWorstElement:=False;
              i:=Length(ParamStr(iPar));  {exit loop}
              end;
           Inc(i);
           end;
        If RemoveWorstElement=True then
           iPar:=ParamCount; {exit loop}
        Inc(iPar);
        end;
{
                Get line to append
}
     If RemoveWorstElement = True then
        begin
        Assign(F_ElemOld,C_Dir+'OldElem.P');
        Assign(F_BadElem,C_Dir+'BadElem.P');
{
                Check that worst stress location is already found
                Found if C_Dir+'BadElem.P' exists
}
        {$I-}
        Reset(F_BadElem);
        {$I+}
        If IoResult<>0 then
           begin
           writeln('Worst stress location was not found. Use WORST before.');
           Bad:=True;
           Bad_Beep;
           Wait;
           end
        else
           begin
           Read(F_BadElem,WorstElementNew);
           Close(F_BadElem);
{
                 Get original element number
}
           Reset(F_ElemOld);
           I_Read(F_ElemOld,WorstElementNew,WorstElementOld);
           str(WorstElementOld,WorstElementString);
           AppendString:='Edel '+WorstElementString+';';
           end;
        end
     else
        begin
        AppendString:='';
        For iPar:=2 to ParamCount-1 do
```

```
            AppendString:=AppendString+ParamStr(iPar)+' ';
          AppendString:=AppendString+ParamStr(ParamCount);
          end;
{
                    Append AppendString to file
}
       If Bad=False then
          begin
          writeln(FileToAppend,'                    ',
                               chr(123),
                               'The following was added with AddToEnd ',
                               chr(125));
          writeln(FileToAppend,AppendString);
{
                    Give screen message
}
          writeln('File ',FileName,' was appended with:');
          writeln(AppendString);
          Close(FileToAppend);
          end;
       end;
end.
```

BACKSUB.PAS

```
{$N+}
Unit BackSub;
Interface
Uses
   Declare,Crt,Displace,File_RW,Nne_Dim;
Procedure Back(var F_Displ,F_Force: FileR;
                   Dim,Nnt,NneMax: IType;
               var F_Fd: FileByte;
               var F_Elem,F_Rev: FileI;
               var F_Eq: FileR);
{-------------------------------------------------------------------}
Implementation
Procedure Back;
{
Procedure Back(var F_Displ,F_Force: FileR;
                   Dim,Nnt,NneMax: IType;
               var F_Fd: FileByte;
               var F_Elem,F_Rev: FileI;
               var F_Eq: FileR);
}
{
   Input:
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
         each node
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
         each node
      Dim: Dimension of problem
      Nnt: Maximum node number
      NneMax: Maximum number of nodes in an element
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Node are negative in Element matrix when they are use for
              the last time according to the numbering of the elements
      F_Rev: File containing order of back substitution
        pos 0¦  node # to be solved
            1¦     node used
             ¦     ...
             ¦     node used
             ¦     # of nodes used (reduced node included) (POSITIVE)
          —> current file position
        or
        pos 0¦  Element number (NEGATIVE)
          —> current file position
      F_Eq: File containing equation constants
             ¦  Constants for node to be solved
             ¦     Constant for node used
             ¦     ...
```

```
            ¦     Constant for node used
            ¦  Constants of equations
            or
            ¦   rows needed in stiffness matrix
            ¦   ...
            ¦   rows needed in stiffness matrix
   Output:
      F_Displ: Missing displacement are calculated
      F_Force: Missing Forces are calculated
)
var
   iRev,iEq,iE: IType;
   Nne,Count,i,j,iNnt,jNode,QttStored,Node_Solve,Node_Used: IType;
   iNne,iD,jNne,jD,RowCount: integer;
   X_R,Kij,Cte: Rtype;
   C: VectR_D;
   K,Ko: MatrixR_DxD;
   Force,Displ: VectR_D;
   Fd: VectByte_D;
   Xs,Ys: Integer;
   Elem: VectI_Nne2;
   Fd_Elem: MatrixByte_NnexD;
   Displ_Elem: array[1..Nne_max,1..D_max] of Rtype;
begin
   write('Back substituting ');
   Xs:=WhereX;
   Ys:=WhereY;
   Count:=0;
   While 0<FilePos(F_Rev) do
      begin
{
                  Read the number of nodes used in equation
}
      Seek(F_Rev,FilePos(F_Rev)-1);
      Read(F_Rev,QttStored);
      If 0<=QttStored then
         begin
{
                  Get Missing Displacements - - - - - - - - - - - -
                  Get iRev: Position in F_Rev of node # to be solved
                  Get iEq: Position in F_Rev of First Contant in equations
}
         iRev := FilePos(F_Rev)-1-QttStored;
         iEq := FilePos(F_Eq)-Dim-QttStored*sqr(Dim);
{
                  Set file pointers to iRev and iEq
}
         Seek(F_Rev,iRev);
         Seek(F_Eq,iEq);
{
                  as stored:        (3 Dimension)
                     iD
```

```
                    1  ku1 kv1 kw1 ku2 kv2 kw2 ... kwN   Cu
                    2   0  kv1 kw1 ku2 kv2 kw2 ... kwN   Cv
                    3   0   0  kw1 ku2 kv2 kw2 ... kwN   Cw
                  stored as row being the fastest loop
}
        Read(F_Rev,Node_Solve);
{
                  Show countdown
}
        D_Byte_Read(F_Fd,Node_Solve,Dim,Fd);
        For iD:=1 to Dim do
           begin
           If Fd[iD]=0 then   (Displacement unknown)
              begin
              GotoXY(Xs,Ys);
              write('D-',Dim*Nnt-Count,' ');
              Count:=Count+1;
              end;
           end;
        For j:=1 to Dim do
           begin
           C[j]:=0;    { initialize }
           For i:=1 to Dim do
              read(F_Eq,K[i,j]);
           end;
        For jNode:=2 to QttStored do
           begin
           Read(F_Rev,Node_Used),
           Displ_R_Read(F_Displ,Node_Used,Dim,Displ);
           Case Dim of
              1..2: begin
                 For j:=1 to Dim do
                    For i:=1 to Dim do
                       begin
                       Read(F_Eq,X_R);
                       C[i]:=C[i]-X_R*Displ[j]
                       end;
                 end;
{
                  Here Ko stands for:  [K] other
}
              3: begin
                 Read(F_Eq,Ko[1,1]);
                 Read(F_Eq,Ko[2,1]);
                 Read(F_Eq,Ko[3,1]);
                 Read(F_Eq,Ko[1,2]);
                 Read(F_Eq,Ko[2,2]);
                 Read(F_Eq,Ko[3,2]);
                 Read(F_Eq,Ko[1,3]);
                 Read(F_Eq,Ko[2,3]);
                 Read(F_Eq,Ko[3,3]);
                 C[1]:=C[1]-Ko[1,1]*Displ[1]-Ko[1,2]*Displ[2]-Ko[1,3]*Displ[3];
```

```
                C[2]:=C[2]-Ko[2,1]*Displ[1]-Ko[2,2]*Displ[2]-Ko[2,3]*Displ[3];
                C[3]:=C[3]-Ko[3,1]*Displ[1]-Ko[3,2]*Displ[2]-Ko[3,3]*Displ[3];
                end;
             end;
          end;
       Case Dim of
          1..2: begin
             For i:=1 to Dim do
                begin
                Read(F_Eq,X_R);
                C[i]:=C[i]+X_R;
                end;
             end;
          3: begin
             Read(F_Eq,X_R);
             C[1]:=C[1]+X_R;
             Read(F_Eq,X_R);
             C[2]:=C[2]+X_R;
             Read(F_Eq,X_R);
             C[3]:=C[3]+X_R;
             end;
          end;
{
                Solve unknowns
}
       Displ_R_Read(F_Displ,Node_Solve,Dim,Displ);
       Case Dim of
          1..2: begin
             For iD:=Dim downto 1 do
                begin
                Cte:=C[iD];
                For i:=Dim downto iD+1 do
                   Cte:=Cte-K[iD,i]*Displ[i];
                Displ[iD]:=Cte/K[iD,iD];
                end;
             end;
          3: begin
             Displ[3]:=C[3]/K[3,3];
             Displ[2]:=(C[2]-K[2,3]*Displ[3])/K[2,2];
             Displ[1]:=(C[1]-K[1,3]*Displ[3]-K[1,2]*Displ[2])/K[1,1];
             end;
          end;
       Displ_R_Write(F_Displ,Node_Solve,Dim,Displ);
       end
    else
{
                Get Missing Forces - - - - - - - - - - - - -
}
       begin
       iE:=abs(OttStored);     {get element number}
       Nne2_I_Read(F_Elem,iE,NneMax,Elem);
       get_Fd_Elem(Dim,Elem,F_Fd,Fd_Elem);
```

```
        Nne:=Get_Nne(Elem[2]);
        RowCount:=0;
{
                Count then number of unknown forces in any direction
                for nodes used for the last time in this element
}
        For iNne:=1 to Nne do
           For iD:=1 to Dim do
              If (Fd_Elem[iNne,iD]=1) and (Elem[2+iNne]<0) then
                 Inc(RowCount);
{
                 Set file pointers to iRev and iEq
}
        iRev := FilePos(F_Rev)-1;
        iEq := FilePos(F_Eq)-RowCount*Nne*Dim;
        Seek(F_Eq,iEq);
{
                 Get all Displacements in Element
}
        For iNne:=1 to Nne do
           begin
           Displ_R_Read(F_Displ,abs(Elem[2+iNne]),Dim,Displ);
           For iD:=1 to Dim do
              Displ_Elem[iNne,iD]:=Displ[iD];
           end;
{
                 Get missing forces
}
        For iNne:=1 to Nne do
           begin
           For iD:=1 to Dim do
              If (Fd_Elem[iNne,iD]=1) and (Elem[2+iNne]<0) then
                 begin
                 GotoXY(Xs,Ys);
                 write('F-',Dim*Nnt-Count,' ');
                 Count:=Count+1;
                 D_R_Read(F_Force,abs(Elem[2+iNne]),Dim,Force);
                 Force[iD]:=0;
                 For jNne:=1 to Nne do
                    For jD:=1 to Dim do
                       begin
                       Read(F_Eq,Kij);
                       Force[iD]:=Force[iD]+Kij*Displ_Elem[jNne,jD];
                       end;
                 D_R_Write(F_Force,abs(Elem[2+iNne]),Dim,Force);
                 end;
           end;
        end;
{
                 Set file pointers to iRev and iEq
}
      Seek(F_Rev,iRev);
```

```
      Seek(F_Eq,iEq);
      end;
   GotoXY(Xs,Ys);
   writeln('   '); {3 blanks}
   Close(F_Rev);
   Erase(F_Rev);
   Close(F_Eq);
   Erase(F_Eq);
end;
{-------------------------------------------------------------------}
end.
```

CONFIG.PAS

```
{-------------------------------------------------------------------
   Configuration program
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}
{$M 12000,0,0}
Uses
   Crt,WaitKey,WhatKey,Declare,Where,Param,Sonore;
Const
   N_Ask_Max = 10;
Type
   VectS255_N = array[1..N_ask_Max] of String255;
   MatrixI_Nx2 = array[1..N_Ask_Max,1..2] of integer;
{-------------------------------------------------------------------}
Procedure Display_Dir(    Ask,Answer: VectS255_N;
                          N_Ask: Integer;
                      var Place: MatrixI_Nx2);
{
   Input:
      Ask: Phrase for Questions
      Answer: Current Answers
      N_Ask: Quantity of Questions
   Output:
      Prints the Menu
      Place[Position,j]: Line (Place[Position,2]) and Column (Place[Position,1]) for Answer
                  output
}
var
   RowStart,Position: integer;
begin
   ClrScr;
   RowStart:=5;
   For Position:=1 to N_Ask do
      begin
      Place[Position,1]:=Length(Ask[Position])+1;
      Place[Position,2]:=RowStart+Position-1;
      GotoXY(Place[Position,1]-Length(Ask[Position]),Place[Position,2]);
      TextColor(LightGreen);
      write(Ask[Position]);
      GotoXY(Place[Position,1],Place[Position,2]);
      TextColor(LightRed);
      write(Answer[Position]);
      TextColor(LightGreen);
      end;
   GotoXY(1,1);
   write('<F1> for Help, <Esc> to end.');
end;
{-------------------------------------------------------------------}
Procedure Display_Answer(    Place: MatrixI_Nx2;
                             Position: integer;
```

```
                        Answer: VectS255_N);
{
   Input:
      Answer: Current Answers
      Position: item # to print
      Place[Position,j]: Line (Place[Position,2]) and Column (Place[Position,1]) for Answer
                  output
   Output:
      Print an item of Answer
}
var
   I: integer;
begin
   GotoXY(Place[Position,1],Place[Position,2]);
   ClrEOL;
   GotoXY(Place[Position,1],Place[Position,2]);
   write(Answer[Position]);
end;
{-------------------------------------------------------------------------}
Procedure Help;
{
                  Gives a Help Screen
}
begin
   ClrScr;
   writeln('Permanent files (*.P): those used for the geometry, solution and set-up.');
   writeln('   *.C are permanent file used for Coarse case to be refined.');
   writeln('For the current case they are kept in the "Current case" directory.');
   writeln('The "Save root" directory is the root directory where all sub-directories');
   writeln('   keep the *.P files pertaining to each case.');
   writeln('This allows to keep many cases at different status.');
   writeln;
   writeln('Generation files (*.GEN): those you give to generate geometry.');
   writeln;
   writeln('Temporary file (*.T): those used during the solution process or node selection.');
   writeln('   They are automaticaly erase after the child process is over.');
   writeln;
   writeln('You can choose any word processor to edit you geometry generation files.');
   writeln('   So specify the word processor''s name (with extension) and directory.');
   writeln('   If your word processor is a single file, you can use this directory');
   writeln('   as the directory of the generation files.');
   writeln;
   writeln('To erase an entry, enter a space.');
   writeln('Use the <arrow> <Home> <End: keys to move.');
   writeln;
   writeln('Press any key to return');
   wait;
end;
{-------------------------------------------------------------------------}
Procedure Configure(var C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255);
{
   Input:
```

```
      C_Dir: Where current case is stored (*.P)
      S_Dir: Where cases are permanently stored (root directory) (*.P)
      G_Dir: Where Generation files are (*.gen)
      T_Dir: Where Temporary files are (*.T)
      W_Dir: Where Word Processor is
      WPname: Word Processor name
   Output:
      C_Dir: Where current case is stored (*.P)
      S_Dir: Where cases are permanently stored (root directory) (*.P)
      G_Dir: Where Generation files are (*.gen)
      T_Dir: Where Temporary files are (*.T)
      W_Dir: Where Word Processor is
      WPname: Word Processor name
}
var
   N_Ask,i,AnswerMax,Position,PreviousPosition: integer;
   C: Char;
   Question,Reponse: String255;
   Ask,Answer: VectS255_N;
   Place: MatrixI_Nx2;
   Action: String6;
begin
   N_Ask:=6;
   If N_Ask_Max<N_Ask then Exit;
   Ask[1]:='Directories:¦      Current case (*.P): ';
   Ask[2]:='            ¦         Save root (*.P): ';
   Ask[3]:='            ¦Generation files (*.GEN): ';
   Ask[4]:='            ¦   Temporary files (*.T): ';
   Ask[5]:='            ¦          Word processor: ';
   Ask[6]:='Word processor name: ';
{
                  Put information into a vector
}
   Answer[1]:=C_Dir;
   Answer[2]:=S_Dir;
   Answer[3]:=G_Dir;
   Answer[4]:=T_Dir;
   Answer[5]:=W_Dir;
   Answer[6]:=WPname;
   Display_Dir(Ask,Answer,N_Ask,Place);
   Position:=1;
   TextColor(White);
   Display_Answer(Place,Position,Answer);
   TextColor(White+Blink);
   write(' <-');
   TextColor(LightGreen);
   AnswerMax:=80-Place[1,1]-3;
   Position:=1;
   Action:='NIL';
   While Action<>'ESC' do
      begin
      Get_Key(C,Action);
```

```
      PreviousPosition:=Position;
{
                  Accept Movements
}
      If Action='UP' then
         If 1<Position then Position:=Position-1;
      If Action='HOME' then
         Position:=1;
      If Action='DOWN' then
         If Position<N_Ask then Position:=Position+1;
      If Action='END' then
         Position:=N_Ask;
{
                  Capture an answer
}
      If C<>chr(0) then
         begin
         Case Position of
            1: Question:='Current case directory: ';
            2: Question:='Save root directory: ';
            3: Question:='Generation file directory: ';
            4: Question:='Temporary file directory: ';
            5: Question:='Word processor directory: ';
            6: Question:='Word processor name: ';
            end;
         TextColor(LightRed);
         Reponse:=UpCase(C);
         GotoXY(1,Place[6,2]+2);
         write(Question,Reponse,' ');
         GotoXY(WhereX-1,WhereY);
         While (C<>chr(13)) and (Length(Reponse)<AnswerMax) do
            begin
            Repeat until keypressed;
            C:=ReadKey;
            C:=UpCase(C);
            If C in ['0'..'9','A'..'Z','\',':','.',','] then  { no blanks }
               Reponse:=Reponse+C;
            If C=chr(8) then
               Delete(Reponse,Length(Reponse),1);
            GotoXY(1,Place[6,2]+2);
            write(Question,Reponse,' ');
            GotoXY(WhereX-1,WhereY);
            end;
         TextColor(LightGreen);
{
                  Erase Building zone
}
         GotoXY(1,Place[6,2]+2);
         ClrEOL;
         GotoXY(1,Place[6,2]+3);
         ClrEOL;
{
```

```
                Check for a complete directory name
}
        If Position<N_Ask then
           If Copy(Reponse,Length(Reponse),1)<>'\' then
              Reponse:=Reponse+'\';
        If Copy(Reponse,1,1)=' ' then Reponse:='';
        Answer[Position]:=Reponse
        end;
{
                Help Screen
}
     If Action='F1' then
        begin
        Help;
        Display_Dir(Ask,Answer,N_Ask,Place);
        end;
{
                Highlight current position
}
     TextColor(LightRed);
     If Position<>PreviousPosition then
        Display_Answer(Place,PreviousPosition,Answer);
     TextColor(White);
     Display_Answer(Place,Position,Answer);
     TextColor(White+Blink);
     write(' <-');
     TextColor(LightGreen);
     end;
{
                Return information from a vector
}
   C_Dir:=Answer[1];
   S_Dir:=Answer[2];
   G_Dir:=Answer[3];
   T_Dir:=Answer[4];
   W_Dir:=Answer[5];
   WPname:=Answer[6];
   ClrScr;
end;
{-------------------------------------------------------------------}
Procedure ParamConfigure(var C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255);
{
   Input:
     ParameterStrin[1..ParamCount]: Parameter strings (upper case)
     C_Dir: Where current case is stored (*.P)
     S_Dir: Where cases are permanently stored (root directory) (*.P)
     G_Dir: Where Generation files are (*.gen)
     T_Dir: Where Temporary files are (*.T)
     W_Dir: Where Word Processor is
     WPname: Word Processor name
   Output:
     C_Dir: Where current case is stored (*.P)
```

```
      S_Dir: Where cases are permanently stored (root directory) (*.P)
      G_Dir: Where Generation files are (*.gen)
      T_Dir: Where Temporary files are (*.T)
      W_Dir: Where Word Processor is
      WPname: Word Processor name
}
Var
   Position: Byte;
   ReplaceString: String255;
begin
{.
      CONFIG ChangedItem [NewString]
         ChangedItem: "CurrentCaseDir"
                   or "SaveRootDir"
                   or "GenerationDir"
                   or "TemporaryDir"
                   or "WpDir"
                   or "WpName".
         NewString: New information corresponding to "ChangedItem".
            If NewString is omitted, "ChangedItem" information is erased.
      Ex:
         CONFIG GenerationDir C:\Direct\Gen\
         --> The generation directory becomes C:\Direct\Gen\
      For more information see the Help screen in the CONFIGURE process.
}
   Position:=0;
   If ParameterString[1]='CURRENTCASEDIR' then Position:=1;
   If ParameterString[1]='SAVEROOTDIR'    then Position:=2;
   If ParameterString[1]='GENERATIONDIR'  then Position:=3;
   If ParameterString[1]='TEMPORARYDIR'   then Position:=4;
   If ParameterString[1]='WPDIR'          then Position:=5;
   If ParameterString[1]='WPNAME'         then Position:=6;
   If ParamCount=2 then ReplaceString:=ParameterString[2];
   If ParamCount=1 then ReplaceString:='     ';
{
                  Check for a complete directory name
}
   If Position in [1..5] then
      begin
      If Copy(ReplaceString,Length(ReplaceString),1)<>'\' then
         ReplaceString:=ReplaceString+'\';
      end;
   If Position=0 then
      begin
      writeln('Invalid first parameter');
      Bad_Beep;
      Wait;
      Halt;
      end;
   If Copy(ReplaceString,1,1)=' ' then ReplaceString:='';
{
                  Replace information
```

```
}
   Case Position of
      1: C_Dir:=ReplaceString;
      2: S_Dir:=ReplaceString;
      3: G_Dir:=ReplaceString;
      4: T_Dir:=ReplaceString;
      5: W_Dir:=ReplaceString;
      6: WPname:=ReplaceString;
      end;
{
                  Give video feedback of transaction
}
   If ReplaceString<>'' then
      begin
      Case Position of
         1: writeln('Current case directory replaced with:');
         2: writeln('Save root directory replaced with:');
         3: writeln('Generation file directory replaced with:');
         4: writeln('Temporary file directory replaced with:');
         5: writeln('Word processor directory replaced with:');
         6: writeln('Word processor name replaced with:');
         end;
      TextColor(LightRed);
      write('   ');
      Case Position of
         1: writeln(C_Dir);
         2: writeln(S_Dir);
         3: writeln(G_Dir);
         4: writeln(T_Dir);
         5: writeln(W_Dir);
         6: writeln(WPname);
         end;
      TextColor(LightGreen);
      end;
   If ReplaceString='' then
      begin
      Case Position of
         1: writeln('Current case directory is erased');
         2: writeln('Save root directory is erased');
         3: writeln('Generation file directory is erased');
         4: writeln('Temporary file directory is erased');
         5: writeln('Word processor directory is erased');
         6: writeln('Word processor name is erased');
         end;
      end;
end;
{-------------------------------------------------------------------}
Var
   C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255;
begin
   Parameters;
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
```

```
   If ParamCount=0 then Configure(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   If 0<ParamCount then ParamConfigure(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   WriteDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
end.
```

CONNECT.PAS

```
Unit Connect;
Interface
   Uses Declare,Linear;
Procedure InitConnectHeapRAM;
Procedure DisposeConnectHeapRAM;
Procedure Con_I_Read(var F: FileI;
                         row: IType;
                     var V: VectI_Con);
Procedure Con_I_Write(var F: FileI;
                          row: IType;
                          V: VectI_Con);
Procedure Con_I_O(var F: FileI;
                      row: IType);
{-------------------------------------------------------------------}
Implementation
Const
   SelectNodeMaxRAM = 2047;
Type
{
                  65504 bytes for 8 connections per node
                  = SelectNodeMaxRAM*(8)*4
}
   ConnectVectRAM = array[1..SelectNodeMaxRAM] of VectI_Con;
   ConnectVectRAMptr = ^ConnectVectRAM;
var
   Connect_RAM: ConnectVectRAMptr;
{-------------------------------------------------------------------}
Procedure InitConnectHeapRAM;
begin
   New(Connect_RAM);
end;
{-------------------------------------------------------------------}
Procedure DisposeConnectHeapRAM;
begin
   Dispose(Connect_RAM);
end;
{-------------------------------------------------------------------}
Procedure Con_I_Read;
{
Procedure Con_I_Read(var F: FileI;
                         row: IType;
                     var V: VectI_Con);
}
{
   Input:
      F: File containing sub-Vectors
      row: position in Global Vector
   Output:
      V: Sub-Vector Read from File F
}
```

```
var
   i_sub: IType;
   i: Byte;
begin
   If row<=SelectNodeMaxRAM then
      V:=Connect_RAM^[row]
   else
      begin
{
                  0 for the first sub-matrix
                  sustract nodes connections kept in RAM
}
      i_sub:=ConnectMax*(row-1-SelectNodeMaxRAM);
      Seek(F,i_sub);
      i:=0;
      Repeat
         Inc(i);
         Read(F,V[i]);
         until ((V[i]=0)or(i=ConnectMax));
      end;
end;
{-------------------------------------------------------------------}
Procedure Con_I_Write;
{
Procedure Con_I_Write(var F: FileI;
                          row: IType;
                          V: VectI_Con);
}
{
   Input:
      F: File containing sub-Vectors
      row: position in Global Vector
   Output:
      V: Sub-Vector written from File F
}
var
   i_sub: IType;
   i: Byte;
begin
   If row<=SelectNodeMaxRAM then
      Connect_RAM^[row]:=V
   else
      begin
{
                  0 for the first sub-matrix
                  sustract nodes connections kept in RAM
}
      i_sub:=ConnectMax*(row-1-SelectNodeMaxRAM);
      Seek(F,i_sub);
      i:=0;
      Repeat
         Inc(i);
```

```
        Write(F,V[i]);
        until ((V[i]=0)or(i=ConnectMax));
      end;
end;
{-------------------------------------------------------------------}
Procedure Con_I_0;
{
Procedure Con_I_0(var F: FileI;
                      row: IType);
}
{
   Input:
      F: File containing sub-Vectors
      row: position in Global Vector
   Output:
      Initializes V: Sub-Vector written to File F
}
var
   i_sub: IType;
   i: Byte;
   X_I,X_zero: IType;
begin
   If row<=SelectNodeMaxRAM then
      begin
      Connect_RAM^[row][1]:=-1;    {-1 for non-selected nodes}
      Connect_RAM^[row][2]:=0;
      end
   else
      begin
{
                  0 for the first sub-matrix
                  sustract nodes connections kept in RAM
}
      i_sub:=ConnectMax*(row-1-SelectNodeMaxRAM);
      Seek(F,i_sub);
      X_I:=-1;
      Write(F,X_I);    {-1 for non-selected nodes}
      X_zero:=0;
      For i:=2 to ConnectMax do
         Write(F,X_zero);
      end;
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
end.
```

COPYGEN.PAS

```
{-------------------------------------------------------------------
   Program to copy a Generation file to another
   Initially written in June '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 2000,0,0}
Uses
   Crt,Param,Sonore,Where,Declare,Dos;
var
   Bad: Boolean;
   FromCaseName,ToCaseName: String[255];
   F: Text;
begin
   Bad:=False;
{
      COPYGEN FromCaseName ToCaseName
         FromCaseName: Name of case without ".GEN" extension
                       FROM which we copy.
         ToCaseName:   Name of case without ".GEN" extension
                       TO which we copy.
      Note: this procedure reduces directory confusion for the used
}
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Parameters;
   If ParamCount<>2 then
      begin
      Bad:=True;
      writeln('   There must be two parameters');
      end
   else
      begin
      FromCaseName:=ParameterString[1];
      ToCaseName:=ParameterString[2];
      end;
   If Bad=False then
      If 0<Pos('.',FromCaseName) then
         begin
         Bad:=True;
         writeln('   There must be no period character in FromCaseName');
         end;
   If Bad=False then
      If 0<Pos('.',ToCaseName) then
         begin
         Bad:=True;
         writeln('   There must be no period character in ToCaseName');
         end;
   If Bad=False then
      If FromCaseName=ToCaseName then
         begin
         Bad:=True;
```

```
        writeln('   FromCaseName and ToCaseName should be different');
        end;
    If Bad=False then
        begin
        Assign(F,G_Dir+FromCaseName+'.gen');
        {$I-}
        Reset(F);
        {$I+}
        If IoResult=0 then
            Close(F)
        else
            begin
            Bad:=True;
            writeln('   Case ',FromCaseName,' is not found in ',
                    G_Dir,' directory');
            end;
        end;
    If Bad=True then Alert;
    If Bad=False then
        begin
        Writeln('Case ',FromCaseName,' is copied to ',ToCaseName);
        FromCaseName:=FromCaseName+'.gen';
        ToCaseName:=ToCaseName+'.gen';
        Exec('\COMMAND.COM','/C '+'COPY '+G_Dir+FromCaseName
                                    +' '+G_Dir+ToCaseName);
        end;
end.
```

DECLARE.PAS

```
{$N+}
Unit Declare;
Interface
Const
   D_max = 3;              { Dimension: =2 for 2D, =3 for 3D }
   DD_max = 9;             { D_max*D_max                     }
   Dim2 = 2;
   Dim3 = 3;
   Nne_max = 20;           { Number of nodes per element     }
   Nne2_max = 22;          { Nne_max + 2                     }
   NneD_max = 60;          { Nne_max.D_max                   }
   Nstress_max = 6;        { Number of stress directions     }
   NstressNne_max = 120,   { Nstress_max x Nne_max           }
   Nmat_max = 50;          { Number of materials             }
   Span_max = 1024;        { used as a window width in Front.pas, here 1 kB }
   Ngq_max = 27;           { Number of Gauss-Quadrature points }
   EtypeMax = 2;       { Number of Element Types         }
   Nc_max = 30;        { Number of data in a command     }
   N_region_max = 3;   { Number of regions for cut       }
   N_group_max = 10;   { Number of sets for cut          }
   ConnectMax = 8;     { Number of connections for a node}
   CrtLine_max = 25;   { Number of lines on the screen   }
   PrtLine_max = 66;   { Number of lines on the printer  }
   Col_max = 80;       { Number of columns on the screen }
   Var_max = 200;      { Number of variables in PREP }
{
                  Real number definition:
                  Double: 8 bytes, 16 digits, max=1.7E308, fast
                  Single: 4 bytes, 8  digits, max=3.4E38, fast
                  Real:   6 bytes, 12 digits, max=1.7E38, slow
}
Type
   Rtype = Double;
Const
   BigRmax = 1.7E308;{ Biggest Real according to RType }
   BigR = 1.6E308;   { A bit smaller than BigRmax      }
{
                  Integer number definition:
                  If you use Integer: limited to 60=sqrt(32767/Dim^2)
                                      nodes on wave front
}
Type
   Itype = LongInt;
Const
   BigImax = 214743647;  { Biggest Integer according to Itype }
   BigI =    214743600;  { A bit smaller than BigImax        }
Type
   MatrixR_NneDxNneD = array[1..NneD_max,1..NneD_max] of Rtype;
   MatrixR_DxD = array[1..D_max,1..D_max] of Rtype;
   MatrixR_Nrx6 = array[1..n_region_max,1..6] of Rtype;
```

```
  MatrixR_Nmatx6 = array[1..Nmat_max,1..6] of Rtype;
  MatrixR_Nmatx9 = array[1..Nmat_max,1..9] of Rtype;
  MatrixR_Nnex12 = array[1..Nne_max,1..12] of Rtype;
  MatrixR_Nnex3 = array[1..Nne_max,1..3] of Rtype;
  MatrixR_3xNne = array[1..3,1..Nne_max] of Rtype;
  MatrixR_6xNneD = array[1..6,1..NneD_max] of Rtype;
  MatrixR_3x3 = array[1..3,1..3] of Rtype;
  MatrixR_6x6 = array[1..6,1..6] of Rtype;
  MatrixR_NnexNgq = array[1..Nne_max,1..Ngq_max] of Rtype;
  MatrixR_NgqxD = array[1..Ngq_max,1..D_max] of Rtype;
  MatrixI_NnexD = array[1..Nne_max,1..D_max] of Integer; {no need for LongInt}
  MatrixI_Nsx4 = array[1..n_group_max,1..4] of IType;
  MatrixI_21x6 = array[1..21,1..6] of IType;
  MatrixByte_NnexD = array[1..Nne_max,1..D_max] of Byte;
  MatrixByte_3x10x2 = array[1..3,1..10,1..2] of Byte;
  MatrixS30_3x10 = Array[1..3,1..10] of String[30];
  VectR_DD = array[1..DD_max] of RType;
  VectR_NstressNne = array[1..NstressNne_max] of Rtype;
  VectR_NneD = array[1..NneD_max] of Rtype;
  VectR_D = array[1..D_max] of Rtype;
  VectR_3 = array[1..3] of Rtype;
  VectR_Nne = array[1..Nne_max] of Rtype;
  VectR_Nc = array[1..Nc_max] of Rtype;
  VectR_Ngq = array[1..Ngq_max] of Rtype;
  VectI_D = array[1..D_max] of IType;
  VectI_Nc = array[1..Nc_max] of IType;
  VectI_Nne = array[1..Nne_max] of IType;
  VectI_Con = array[1..ConnectMax] of IType;
  VectI_NneD = array[1..NneD_max] of IType;
  VectI_Nne2 = array[1..Nne2_max] of IType;
  VectI_21 = array[1..21] of IType;
  VectI_3 = array[1..3] of IType;
  VectC_Screen = array[1..2000] of Char;
  VectS80_Nmat = array[1..Nmat_max] of string[80];
  VectS80_Nc = array[1..Nc_max] of string[80];
  VectS80_Var = array[1..Var_max] of string[80];
  VectByte_D = array[1..D_max] of Byte;
  VectByte_Nmat = array[1..Nmat_max] of Byte;
  VectByte_25 = array[1..25] of Byte;
  VectB_Nne = array[1..Nne_max] of Boolean;
  String6 = String[6];
  String12 = String[12];
  String80 = String[80];
  String255 = String[255];
  FileI = File of IType;
  FileR = File of RType;
  FileByte = File of Byte;
  FileC = File of Char;
{-------------------------------------------------------------}
{
                Directory variables
}
```

```
var
  C_Dir,S_Dir,G_Dir,W_Dir,T_Dir,WPname: String255;
{-------------------------------------------------------------------------}
{
                  Parameter variables
}
Const
  Param_max = 50;     { Number of parameters passed to a program }
  Param_Long = 80;    { Length of parameters }
Var
  ParameterString: array[1..Param_max] of String[Param_Long];
  iParam: Byte;
  OnPrinter,NoWait: Boolean;
{-------------------------------------------------------------------------}
Implementation
end.
```

DEFORM.PAS

```
{-------------------------------------------------------------------------
  Program to see deformed geometry geometry
  Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 10000,0,76004}    {max memory used for drawing elements}
                      {10500 bytes in Heap needed for InitGraph}
                      {65504=2047*8*4 bytes in Heap needed for Connect.pas}
Uses
  Crt,Graph,Dos,Declare,Draw,Where,Nombre,Linear,Nne_Dim,File_RW,Get,Connect;
{-------------------------------------------------------------------------}
Procedure AddDispl(    Nnt: IType;
                   var F_xyz3D,F_Displ: FileR;
                       Dim: IType;
                   var Title: String80);
{
  Input:
    Nnt: Number of nodes
    F_xyz3D: File giving xyz in 3-d
    Dim: Original Dimension of geometry
    F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
        each node
  Output:
    F_xyz3D: File giving xyz in 3-d + added displacement (exagerated)
    Title: Title for graph with exageration ratio
}
var
  iD,iNnt,iCol: IType;
  ConvertResult: integer;
  MaxXYZ,MinXYZ,xyz,Displ: VectR_D;
  UserRatio,HiVarXYZ,HiD,DisplLong,Fraction,ExagScale: Rtype;
  UserRatioString: String255;
  Xs,Ys: Byte;
begin
  write('Finding extremes in coordinates ');
  Xs:=WhereX;
  Ys:=WhereY;
{
                  Find max min of coordinates and displacements
}
  For iD:=1 to 3 do
    begin
    MaxXYZ[iD]:=-1E30;
    MinXYZ[iD]:=+1E30;
    end;
  HiD:=-1E30;
  For iNnt:=1 to Nnt do
    begin
    gotoXY(Xs,Ys);
    write(Nnt-iNnt+1,' ');
```

```
      D_R_Read(F_xyz3D,iNnt,3,xyz);
      D_R_Read(F_Displ,iNnt,Dim,Displ);
      If Dim=2 then Displ[3]:=0;
{
                  Get vector length of displacement
}
      DisplLong:=0;
      For iD:=1 to Dim do
         DisplLong:=DisplLong+sqr(Displ[iD]);
      DisplLong:=Sqrt(DisplLong);
      If HiD<DisplLong then HiD:=DisplLong;
      For iD:=1 to 3 do
         begin
         If MaxXYZ[iD]<xyz[iD] then MaxXYZ[iD]:=xyz[iD];
         If xyz[iD]<MinXYZ[iD] then MinXYZ[iD]:=xyz[iD];
         end;
      end;
   gotoXY(Xs,Ys);
   writeln(' ');
{
                  Get Highest variation in X Y Z
                  and Highest absolute displacement
}
   HiVarXYZ:=-1E30;
   For iD:=1 to D_max do
      If HiVarXYZ<MaxXYZ[iD]-MinXYZ[iD] then HiVarXYZ:=MaxXYZ[iD]-MinXYZ[iD];
{
                  Find Scaling factor for displacements:
                  Fraction of maximum geometry variation
}
   Fraction:=1/5;
   ExagScale:=HiVarXYZ/HiD*Fraction;
   writeln('Displacements will be exagerated to 1/',Round(1/Fraction),
           ' of the largest coordinate variation.');
   writeln('--> Exaggeration = ',Rword(ExagScale,8));
   write('Fine ratio tuning (default is 1): ');
   Xs:=WhereX;
   Ys:=WhereY;
   TextColor(LightRed);
   Repeat
      gotoXY(Xs,Ys);
      For iCol:=Xs to 80 do write(' ');
      gotoXY(Xs,Ys);
      UserRatio:=0;
      readln(UserRatioString);
      If Length(UserRatioString)=0 then
         begin
         UserRatio:=1;
         UserRatioString:='1';
         end
      else
         begin
```

```
        If UserRatioString[1]='.' then UserRatioString:='0'+UserRatioString;
        If not (UserRatioString[1] in ['+','-']) then
          begin
          Val(UserRatioString,UserRatio,ConvertResult);
          If ConvertResult<>0 then UserRatio:=0;
          end;
        end;
    until 0<UserRatio;
  GotoXY(Xs,Ys);
  write(UserRatioString,' ');
  TextColor(LightGreen);
  writeln('Ok!');
  ExagScale:=ExagScale*UserRatio;
{
                Add exagerated displacements to coordinates
}
  write('Adding exagerated displacements to coordinates ');
  Xs:=WhereX;
  Ys:=WhereY;
  For iNnt:=1 to Nnt do
    begin
    gotoXY(Xs,Ys);
    write(Nnt-iNnt+1,' ');
    D_R_Read(F_xyz3D,iNnt,3,xyz);
    D_R_Read(F_Displ,iNnt,Dim,Displ);
    If Dim=2 then Displ[3]:=0;
    For iD:=1 to 3 do
       xyz[iD]:=xyz[iD]+Displ[iD]*ExagScale;
    D_R_write(F_xyz3D,iNnt,3,xyz);
    end;
{
                Create Title
}
  Title:='Deformed geometry (exaggeration='+Rword(ExagScale,8)+')';
  gotoXY(Xs,Ys);
  writeln('  ');
end;
{-------------------------------------------------------------------------}
var
  NneMax,i,iD,Net,Nnt,Dim,Draw_Axis,Draw_Elev,Numbering,N_node: IType;
  Phi,Theta: Rtype;
  Modified: Boolean;
  Title: String80;
  F_xyz,F_xyz3D,F_Displ: FileR;
  F_Connect,F_NewOld,F_Elem: File1;
begin
  ClrScr;
  TextColor(LightGreen);
  Writeln('Loading geometry');
  ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
  Get_I('Net.P',Net);
  Get_I('Nnt.P',Nnt);
```

```
   Get_I('Dim.P',Dim);
   Get_I('NneMax.P',NneMax);
{
                Get XYZ coordinates
}
   Assign(F_xyz,C_Dir+'xyz.P');
   Assign(F_xyz3D,T_Dir+'xyz_3d.T');
   Assign(F_Elem,C_Dir+'Elem.P');
   Assign(F_NewOld,C_Dir+'NewToOld.P');
   Assign(F_Connect,T_Dir+'Connect.T');   {Don't call it 'CON.dat'}
   Assign(F_Displ,C_Dir+'Displ.P');
   Reset(F_xyz);
   Rewrite(F_xyz3D);
   Reset(F_Elem);
   Reset(F_NewOld);
   Rewrite(F_Connect);
   Reset(F_Displ);
   Modified:=True;
   xyz_3d(Nnt,F_xyz,F_xyz3D,Dim,Modified);
   AddDispl(Nnt,F_xyz3D,F_Displ,Dim,Title);
   InitConnectHeapRAM;
   Select('DEFORM',Net,Nnt,NneMax,F_xyz3D,F_Elem,F_NewOld,F_connect,
         Phi,Theta,Numbering,Draw_Axis,Draw_Elev);
   Dessine(Title,Nnt,Phi,Theta,
          Numbering,Draw_Axis,Draw_Elev,F_xyz3D,1.0,F_NewOld,F_Connect);
   DisposeConnectHeapRAM;
   Close(F_xyz);
   Close(F_xyz3D);      Erase(F_xyz3D);
   Close(F_Elem);
   Close(F_NewOld);
   Close(F_Connect);    Erase(F_Connect);
   Close(F_Displ);
   ClrScr;
end.
```

DELSTRES.PAS

```
{-------------------------------------------------------------------------
   Program to remove stress files previously calculated and correct status
   Initially written in May '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 3000,0,0}
Uses
   Crt,Dos,Declare,Where,Param,Status;
{-------------------------------------------------------------------------}
var
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   DelFiberStrain,DelFiberStress,
      DelElemStrain,DelElemStress,
      DelGlobalStrain,DelGlobalStress: Boolean;
begin
{
      DELSTRESS [FiberStrain] [FiberStress] [ElemStrain] [ElemStress]
                [GlobalStrain] [GlobalStress]
         FiberStrain:  For deleting strain in the material direction.
         FiberStress:  For deleting stress in the material direction.
         ElemStrain:   For deleting strain in the element  direction.
         ElemStress:   For deleting stress in the element  direction.
         GlobalStrain: For deleting strain in the global   direction.
         GlobalStress: For deleting stress in the global   direction.
         Parameters can be simultaneous and interchanged.
}
   Parameters;
{
                  Read Directories and Status
}
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
{
                   Initialize stress to remove
}
   DelFiberStrain:=False;
   DelFiberStress:=False;
   DelElemStrain:=False;
   DelElemStress:=False;
   DelGlobalStrain:=False;
   DelGlobalStress:=False;
   For iParam:=1 to ParamCount do
```

```
      begin
      If ParameterString[iParam]='FIBERSTRAIN' then DelFiberStrain:=True;
      If ParameterString[iParam]='FIBERSTRESS' then DelFiberStress:=True;
      If ParameterString[iParam]='ELEMSTRAIN' then DelElemStrain:=True;
      If ParameterString[iParam]='ELEMSTRESS' then DelElemStress:=True;
      If ParameterString[iParam]='GLOBALSTRAIN' then DelGlobalStrain:=True;
      If ParameterString[iParam]='GLOBALSTRESS' then DelGlobalStress:=True;
      end;
{
                  Avoid deleting an unexistant stress file
}
   If StatusFiberStrain <>'Done' then DelFiberStrain :=False;
   If StatusFiberStress <>'Done' then DelFiberStress :=False;
   If StatusElemStrain  <>'Done' then DelElemStrain  :=False;
   If StatusElemStress  <>'Done' then DelElemStress  :=False;
   If StatusGlobalStrain<>'Done' then DelGlobalStrain:=False;
   If StatusGlobalStress<>'Done' then DelGlobalStress:=False;
   If DelFiberStrain=True then
      begin
      StatusFiberStrain:='';
      Exec('\COMMAND.COM','/C Del '+C_Dir+'Fstrain.P ');
      GotoXY(1,WhereY);
      ClrEOL;
      writeln('Strain in material direction deleted');
      end;
   If DelFiberStress=True then
      begin
      StatusFiberStress:='';
      Exec('\COMMAND.COM','/C Del '+C_Dir+'Fstress.P ');
      GotoXY(1,WhereY);
      ClrEOL;
      writeln('Stress in material direction deleted');
      end;
   If DelElemStrain=True then
      begin
      StatusElemStrain:='';
      Exec('\COMMAND.COM','/C Del '+C_Dir+'Estrain.P ');
      GotoXY(1,WhereY);
      ClrEOL;
      writeln('Strain in element direction deleted');
      end;
   If DelElemStress=True then
      begin
      StatusElemStress:='';
      Exec('\COMMAND.COM','/C Del '+C_Dir+'Estress.P ');
      GotoXY(1,WhereY);
      ClrEOL;
      writeln('Stress in element direction deleted');
      end;
   If DelGlobalStrain=True then
      begin
      StatusGlobalStrain:='':
```

```
      Exec('\COMMAND.COM','/C Del '+C_Dir+'Gstrain.P ');
      GotoXY(1,WhereY);
      ClrEOL;
      writeln('Strain in global direction deleted');
      end;
   If DelGlobalStress=True then
      begin
      StatusGlobalStress:='';
      Exec('\COMMAND.COM','/C Del '+C_Dir+'Gstress.P ');
      GotoXY(1,WhereY);
      ClrEOL;
      writeln('Stress in global direction deleted');
      end;
{
                  Correct status
}
   Write_Status(CaseName,CoarseCaseName,
                StatusPrep,StatusForce,StatusDispl,StatusWorst,
                StatusFiberStrain,StatusFiberStress,
                StatusElemStrain,StatusElemStress,
                StatusGlobalStrain,StatusGlobalStress);
end.
```

DETAIL.PAS

```
AddToEnd.pas -> executable
   Add a command to the end of a *.gen file
BackSub .pas -> unit
   Backsubstitution
Config  .pas -> executable
   sets directory configuration
Connect .pas -> unit
   nodal connections
CopyGen .pas -> executable
   copy a *.gen file
Declare .pas -> unit
   declares types, matrices...
Deform  .pas -> executable
   show deformed body
DelStres.pas -> executable
   delete a calculated stress
Direct  .pas -> executable
   manages all executable programs
Displace.pas -> unit
   access displacements from RAM
Draw    .pas -> unit
   node selection, drawing
Edit    .pas -> unit
   edit a data
Elem1   .pas -> unit
   element 1 formulation: stiffness, stress, traction
Elem2   .pas -> unit
   element 2 formulation: stiffness, stress, traction
Exyz    .pas -> unit
   gets the element nodal coordinates
File_RW .pas -> unit
   read and write to file
Fill    .pas -> executable
   diskette filling
Fixed   .pas -> unit
   transforms stiffnes matrices for known displacements
Front   .pas -> unit
   front solver
Get     .pas -> unit
   special file reads
Help    .pas -> executable
   help
H_Batch .pas -> unit
   help on batch mode
H_Common.pas -> unit
   common routines to help units
H_Etype .pas -> unit
   help on element definition
H_Featur.pas -> unit
   help on features
```

```
H_Lang  .pas -> unit
   help on language
H_Main  .pas -> unit
   help on main menu
J_Graph .pas -> unit
   personal graphing routines
K_Common.pas -> unit
   common routines to preprocessing commands
K_Elem  .pas -> unit
   element definition related preprocessing commands
K_ForDis.pas -> unit
   force and displacement related preprocessing commands
K_Inter .pas -> unit
   interpolation preprocessing commands
K_Local .pas -> unit
   local coordinates transformation routines for preprocessing commands
K_Mat   .pas -> unit
   material definition preprocessing commands
K_Node  .pas -> unit
   node definition related preprocessing commands
K_Old   .pas -> unit
   mesh refinement related preprocessing commands
K_Oper  .pas -> unit
   variable operation preprocessing command
K_Trac  .pas -> unit
   traction preprocessing commands
K_Var   .pas -> unit
   variable preprocessing command
Linear  .pas -> unit
   matrix index linearization
Lines   .pas -> executable
   line counter
Listing .pas -> executable
   produces listing
LocGlo  .pas -> unit
   local - global coordinate transformation
Manager .pas -> executable
   stores, retreives cases
Material.pas -> executable
   material property definition, edition
Math    .pas -> unit
   mathematical routines
NewName .pas -> executable
   gives the case a new name
Nne_Dim .pas -> unit
   dimension and element type functions
Nombre  .pas -> unit
   number printing
Numbers .pas -> executable
   gives information upon case and computer
Param   .pas -> unit
   handles program parameters
```

```
Pickfile.pas -> unit
   file selection
Prep    .pas -> executable
   preprocessor
Put     .pas -> unit
   special file writes
Regres  .pas -> executable
   data regression
Result  .pas -> executable
   presents results
Shuffle .pas -> unit
   stiffness matrix index permutation
Solve   .pas -> executable
   solve for displacement and forces
Sonore  .pas -> unit
   sound functions
Status  .pas -> unit
   read and writes the case status
Stress  .pas -> executable
   calculates strain and stress
Sub     .pas -> unit
   store submatrices in RAM
Sum     .pas -> unit
   creates batch file to sum cases
SumToNew.pas -> executable
   sum case solutions
Timer   .pas -> unit
   stopwatch
Translat.pas -> unit
   preprocessing command line interpreter
VectMat3.pas -> unit
   operations on vectors and matrices of order 3
View    .pas -> executable
   view the geometry
WaitKey .pas -> unit
   program interuption
WhatKey .pas -> unit
   returns the name of a pressed key
Where   .pas -> unit
   read and write configuration directories
Worst   .pas -> executable
   finds the failure location
```

DIRECT.PAS

```
{-------------------------------------------------------------------------
   Parent program
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}
{$M 2400,0,0} {Minimum stack required, but no heap required or reserved}
Uses
   Crt,Dos,WaitKey,Where,Declare,Sonore,Status,Timer;
Type
   MatrixByte_25x2 = array[1..25,1..2] of Byte;
   VectS20_25 = array[1..25] of String[20];
{-------------------------------------------------------------------------}
Procedure Message;
var
   Vary,Dir,X,Xl,i: Integer;
begin
   clrscr;
   TextColor(LightGreen);
   writeln;
   GotoXY(26,WhereY); Writeln('-------------------------');
   GotoXY(26,WhereY); Writeln('-------------------------');
   GotoXY(26,WhereY); Writeln('-------          -------');
   GotoXY(26,WhereY); Writeln('-------  D I R E C T  -------');
   GotoXY(26,WhereY); Writeln('-------          -------');
   GotoXY(26,WhereY); Writeln('-------------------------');
   GotoXY(26,WhereY); Writeln('-------------------------');
   GotoXY(35,5);
   TextColor(White);
   write('D I R E C T');
   GotoXY(1,11);
   TextColor(LightCyan);
   Xl:=19;
   GotoXY(Xl-1,WhereY); Writeln('+----+-----------------------------------------+');
   GotoXY(Xl-1,WhereY); Writeln('| xx | J. Daoust Ph.D. and S.V. Hoa Ph.D.      |');
   GotoXY(Xl-1,WhereY); Writeln('+----+ Composite Material Laboratory          |');
   GotoXY(Xl-1,WhereY); Writeln('|   Department of Mechanical Engineering     |');
   GotoXY(Xl-1,WhereY); Writeln('|          Concordia University              |');
   GotoXY(Xl-1,WhereY); Writeln('|        1455 de Maisonneuve West            |');
   GotoXY(Xl-1,WhereY); Writeln('|     Montreal, Quebec, Canada, H3G 1M8      |');
   GotoXY(Xl-1,WhereY); Writeln('|              (514) 848-3139                |');
   GotoXY(Xl-1,WhereY); Writeln('|              (514) 848-8796                |');
   GotoXY(Xl-1,WhereY); Writeln('|  Copyright (C) 1988 Concordia University   |');
   GotoXY(Xl-1,WhereY); Writeln('+-------------------------------------------+');
   TextColor(LightRed);
   GotoXY(Xl+1,12);       write('8n');
   GotoXY(Xl+5,12);          write(' J. Daoust Ph.D. and S.V. Hoa Ph.D.   ');
   GotoXY(Xl+5,WhereY+1);    write('  Composite Material Laboratory       ');
   GotoXY(Xl,WhereY+1); write('   Department of Mechanical Engineering    ');
   GotoXY(Xl,WhereY+1); write('          Concordia University             ');
   GotoXY(Xl,WhereY+1); write('        1455 de Maisonneuve West           ');
```

```
    GotoXY(Xl,WhereY+1); write('     Montreal, Quebec, Canada, H3G 1M8     ');
    GotoXY(Xl,WhereY+1); write('              (514) 848-3139               ');
    GotoXY(Xl,WhereY+1); write('              (514) 848-8796               ');
    GotoXY(Xl,WhereY+1); write('  Copyright (C) 1988 Concordia University ');
    GotoXY(79,25);
    TextColor(LightGreen);
    Vary:=13;
    X:=-Vary;
    Dir:=1;
    Repeat
      If Dir=1 then
         begin
         For i:=1 to 7 do
            begin
            GotoXY(54+Vary+X,1+i);
            write('-',Chr(16));
            end;
         For i:=1 to 7 do
            begin
            GotoXY(26-Vary+X,1+i);
            write(' ');
            end;
         Inc(X);
         end
      else
         begin
         For i:=1 to 7 do
            begin
            GotoXY(55+Vary+X,1+i);
            write(' ');
            end;
         For i:=1 to 7 do
            begin
            GotoXY(26-Vary+X,1+i);
            write(Chr(17),'-');
            end;
         Dec(X);
         end;
      GotoXY(79,25);
      Delay(20);
      If (X=Vary) and (Dir=1) then
         begin
         Dir:=-1;
         Dec(X);
         end;
      If (X=-Vary) and (Dir=-1) then
         begin
         Dir:=1;
         Inc(X);
         end;
      until KeyPressed;
end;
```

```
{-------------------------------------------------------------------------}
procedure ShowMenu(var PlaceXY:MatrixByte_25x2;
                   var Menu: VectS20_25;
                   var Choice_qtt: integer;
                   var TimeX,TimeY: Byte;
                   var CaseName,CoarseCaseName: String255;
                   var StatusPrep,StatusForce,StatusDispl,StatusWorst,
                       StatusFiberStrain,StatusFiberStress,
                       StatusElemStrain,StatusElemStress,
                       StatusGlobalStrain,StatusGlobalStress: String12);
{
   Input:
      CaseName: File name being considered (Without extension)
      CoarseCaseName: Solved case used as a coarse mesh for refinement
      StatusPrep,StatusForce,StatusDispl,StatusWorst,
         StatusFiberStrain,StatusFiberStress,
         StatusElemStrain,StatusElemStress,
         StatusGlobalStrain,StatusGlobalStress: "Done", " ", "Error"
   Output:
      Displays input data
      PlaceXY[i,1]: X screen position of menu item
                2 : Y screen position of menu item
      Menu[i]: menu item number i
      Choice_qtt: number of menu items
      TimeX,TimeY: screen coordinates for time display
}
var
   StatusX,First,i: integer;
begin
   ClrScr;
   GotoXY(1,1);
   write('Case considered: ');
   TextColor(LightRed);
   writeln(CaseName);
   TextColor(LightGreen);
   write('Case used as a coarse mesh solution: ');
   TextColor(LightRed);
   writeln(CoarseCaseName);
   TextColor(LightGreen);
   First:=5;
   GotoXY(1,First);
   Menu[1]:= 'Help';
   Menu[2]:= 'Configure';
   Menu[3]:= 'Material';
   Menu[4]:= 'Edit';
   Menu[5]:= 'Preprocess';
   Menu[6]:= 'Numbers';
   Menu[7]:= 'View';
   Menu[8]:= 'Solve';
   Menu[9]:= 'Stress';
   Menu[10]:='Worst';
   Menu[11]:='Deform';
```

```
   Menu[12]:='Result';
   Menu[13]:='Manager';
   Menu[14]:='Exit';
   Choice_qtt:=14;
{
                Other programs:
                AddToEnd CopyGen DelStres Fill NewName Regres SumToNew
}
     writeln('+-----------+-----------+');
      write('¦           ¦           ¦');
     TimeX:=WhereX-13; TimeY:=WhereY;  writeln;
     writeln('¦           +-----------+');
   For i:=1 to Choice_qtt-2 do
     writeln('¦           ¦');
     writeln('+-----------+');
   StatusX:=Length('+-----------+')+2;
   TextColor(LightGreen);
   For i:=1 to Choice_qtt do
     begin
     PlaceXY[i,1]:=3;
     PlaceXY[i,2]:=First+i;
     GotoXY(PlaceXY[i,1],PlaceXY[i,2]),
     write(Menu[i]);
     If Menu[i]='Preprocess' then
        begin
        GotoXY(StatusX,WhereY);
        write(StatusPrep);
        end;
     If Menu[i]='Solve' then
        begin
        GotoXY(StatusX,WhereY);
        If ((StatusForce='Done') or
            (StatusDispl='Done')) then
           begin
           write('Done:');
           If StatusDispl<>'' then
              write(' Displacement');
           If StatusForce<>'' then
              write(' Force');
           end;
        end;
     If Menu[i]='Worst' then
        begin
        GotoXY(StatusX,WhereY);
        write(StatusWorst);
        end;
     If Menu[i]='Stress' then
        begin
        GotoXY(StatusX,WhereY);
        If ((StatusFiberStrain<>'') or
            (StatusFiberStress<>'') or
            (StatusElemStrain<>'') or
```

```
            (StatusElemStress<>'') or
            (StatusGlobalStrain<>'') or
            (StatusGlobalStress<>'')) then
           begin
           write('Done:');
           If StatusFiberStrain<>'' then
              write(' Fiber-e');
           If StatusFiberStress<>'' then
              write(' Fiber-S');
           If StatusElemStrain<>'' then
              write(' Elem-e');
           If StatusElemStress<>'' then
              write(' Elem-S');
           If StatusGlobalStrain<>'' then
              write(' Global-e');
           If StatusGlobalStress<>'' then
              write(' Global-S');
           end;
        end;
     end;
  TextColor(White);
  For i:=1 to Choice_qtt do
     begin
     GotoXY(PlaceXY[i,1],PlaceXY[i,2]);
     write(Menu[i][1]);
     end;
  TextColor(LightGreen);
end;
{-------------------------------------------------------------------------}
Var
  CaseName,CoarseCaseName,CurrentDir,Prefix: String255;
  StatusPrep,StatusForce,StatusDispl,StatusWorst,
     StatusFiberStrain,StatusFiberStress,
     StatusElemStrain,StatusElemStress,
     StatusGlobalStrain,StatusGlobalStress: String12;
  i,ChoixOld,Choix,Choice_qtt: Integer;
  ActionLetter,letter:Char;
  Menu: Vect520_25;
  PlaceXY: MatrixByte_25x2;
  ErrorX,ErrorY,TimeX,TimeY: Byte;
  DoMessage,Found: Boolean;
Begin
  ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
  ErrorX:=20;
  ErrorY:=22;
  DoMessage:=True;
  Repeat
     Read_Status(CaseName,CoarseCaseName,
                 StatusPrep,StatusForce,StatusDispl,StatusWorst,
                 StatusFiberStrain,StatusFiberStress,
                 StatusElemStrain,StatusElemStress,
                 StatusGlobalStrain,StatusGlobalStress);
```

```
      If DoMessage=True then
         begin
         DoMessage:=False;
         Message;
         end;
      ShowMENU(PlaceXY,Menu,Choice_qtt,TimeX,TimeY,
               CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
      TextColor(LightRed);
      ChoixOld:=0;
      Choix:=1;
      Repeat
{
                  Display Old and New Choice
}
         If (0<ChoixOld) and (ChoixOld<>Choix) then
            begin
            GotoXY(PlaceXY[ChoixOld,1],PlaceXY[ChoixOld,2]);
            write(Menu[ChoixOld]);
            TextColor(White);
            GotoXY(PlaceXY[ChoixOld,1],PlaceXY[ChoixOld,2]);
            write(Menu[ChoixOld][1]);
            TextColor(LightGreen);
            Beep;
            end;
         If ChoixOld<>Choix then
            begin
            TextColor(Black);
            TextBackground(LightRed);
            GotoXY(PlaceXY[Choix,1],PlaceXY[Choix,2]);
            write(Menu[Choix]);
            TextColor(LightGreen);
            TextBackground(Black);
            end;
         ChoixOld:=Choix;
{
                  Read Key
}
         While Keypressed do Letter:=ReadKey;  { empty keyboard buffer }
         While not Keypressed do Show_Time(TimeX,TimeY);
         Letter:=Readkey;
         Letter:=UpCase(letter);
         ActionLetter:=chr(0);
         If Letter=chr(0) then ActionLetter:=ReadKey;
{
                  Use letter to find next process starting with that letter
}
         Found:=False;
         i:=ChoixOld+1;
```

```
        While (i<=Choice_Qtt) and (Found=False) do
           begin
           If UpCase(Menu[i][1])=Letter then
              begin
              Choix:=i;
              Found:=True;
              end;
           Inc(i);
           end;
        i:=1;
        While (i<ChoixOld) and (Found=False) do
           begin
           If UpCase(Menu[i][1])=Letter then
              begin
              Choix:=i;
              Found:=True;
              end;
           Inc(i);
           end;
{
                Displace selection bar
}
        If ActionLetter='H' then  { Up }
           If 1<Choix then
              Choix:=Choix-1;
        If ActionLetter='P' then  { Down }
           If Choix<Choice_Qtt then
              Choix:=Choix+1;
        If ActionLetter='G' then  { Home }
           Choix:=1;
        If (ActionLetter='O') or (Letter=chr(27)) then  { End Esc }
           Choix:=Choice_Qtt;
        Until letter=chr(13);     { Enter }
{
                Show selected child process
}
     Beep;
     GotoXY(PlaceXY[Choix,1],PlaceXY[Choix,2]);
     write(Menu[Choix]);
     TextColor(White+Blink);
     GotoXY(PlaceXY[Choix,1],PlaceXY[Choix,2]);
     write(Menu[Choix][1]);
     TextColor(LightGreen);
{
                Erase Time Display
}
     GotoXY(TimeX,TimeY);
     write('Loading ...');
     GotoXY(80,25);
     GetDir(0,CurrentDir);
     If Menu[Choix]='Edit' then ClrScr;
{
```

```
                Execute child process
}
    If Menu[Choix]='Help' then
       Exec('HELP.EXE','');
    If Menu[Choix]='Configure' then
       begin
       Exec('CONFIG.EXE','');
       ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
       end;
    If Menu[Choix]='Material' then
       Exec('MATERIAL.EXE','');
    If Menu[Choix]='Edit' then
       begin
       Delete(W_Dir,Length(W_Dir),1);   { take out last '\' }
       ChDir(W_Dir);
       W_Dir:=W_Dir+'\';   { restore last '\' }
       Exec(WPname,'');
       ChDir(CurrentDir);
       end;
    If Menu[Choix]='Preprocess' then
       Exec('PREP.EXE','');
    If Menu[Choix]='Numbers' then
       If (StatusPrep='Done') then
          Exec('NUMBERS.EXE','')
       else
          begin
          GotoXY(ErrorX,ErrorY);
          write('You must preprocess before!');
          wait;
          end;
    If Menu[Choix]='View' then
       If (StatusPrep='Done') then
          Exec('VIEW.EXE','')
       else
          begin
          GotoXY(ErrorX,ErrorY);
          write('You must preprocess before!');
          wait;
          end;
    If Menu[Choix]='Solve' then
       If (StatusPrep='Done') and
          ((StatusDispl<>'Done') or (StatusDispl<>'Done')) then
          Exec('SOLVE.EXE','')
       else
          begin
          GotoXY(ErrorX,ErrorY);
          If StatusPrep<>'Done' then
             write('You must preprocess before!');
          If not ((StatusDispl<>'Done') or (StatusDispl<>'Done')) then
             write('The case is already solved!');
          wait;
          end;
```

```
      If Menu[Choix]='Stress' then
         If (StatusDispl='Done') then
            Exec('STRESS.EXE','')
         else
            begin
            GotoXY(ErrorX,ErrorY);
            write('You must solve for displacements first!');
            wait;
            end;
      If Menu[Choix]='Worst' then
         If (StatusDispl='Done') then
            Exec('WORST.EXE','')
         else
            begin
            GotoXY(ErrorX,ErrorY);
            write('You must solve for displacements first!');
            wait;
            end;
      If Menu[Choix]='Deform' then
         If (StatusDispl='Done') then
            Exec('DEFORM.EXE','')
         else
            begin
            GotoXY(ErrorX,ErrorY);
            write('You must solve for displacements first!');
            wait;
            end;
      If Menu[Choix]='Result' then
         If (StatusDispl='Done') then
            Exec('RESULT.EXE','')
         else
            begin
            GotoXY(ErrorX,ErrorY);
            write('You must solve for displacements first!');
            wait;
            end;
      If Menu[Choix]='Manager' then
         Exec('MANAGER.EXE','');
      Until Menu[Choix]='Exit';
   ClrScr;
End.
```

DISPLACE.PAS

```
{$N+}
Unit Displace;
Interface
Uses
   Declare,Crt,File_RW;
Procedure Displ_R_Read(var F: FileR;
                           DisplNumber,Dim: IType;
                       var V: VectR_D);
Procedure Displ_R_Write(var F: FileR;
                            DisplNumber,Dim: IType;
                        var V: VectR_D);                    (var for speed)
Procedure InitDisplHeapRAM(    Dim,Nnt: Itype;
                           var F: FileR);
Procedure RamDisplToDisk(    Dim: Itype;
                         var F: FileR);
Procedure DisposeDisplHeapRAM;
{-------------------------------------------------------------------------}
const
   MaxDisplVectHeapRAM = 800;  {3.2 k max for pointers array)
                           {MaxDisplVectHeapRAM*[4=SizeOf(DisplVectHeapPtr)]}
{
                  In order to optimize speed of
                  Dim=1, Dim=2 and Dim=3   cases
                  set DisplVectHeapLong as a product of 1*2*3*N
                  So a SubVector is all contained inside
                     a same heap vector
}
   DisplVectHeapLong = 102; { (1*2*3)*17 }
                           {652800 Bytes max for heap storage of Rtype)
type
   DisplVectHeap = array[1..DisplVectHeapLong] of Rtype;
   DisplVectHeapPtr = ^DisplVectHeap;
var
   DisplHeapRAM: array[1..MaxDisplVectHeapRAM] of DisplVectHeapPtr;
   DisplNumberMaxRAM,DisplNumberMax,LastDisplVectHeapRAM: Itype;
{-------------------------------------------------------------------------}
Implementation
Procedure PosInRAM(    Dim,DisplNumber: Itype;
                   var StartVect,PosInVect: Itype);
{
   Input:
      Dim: Dimension of problem
      DisplNumber: SubVector number:
         1...N
      DisplVectHeapLong (Constant): Length of vectors kept in Heap RAM
   Output:
      StartVect: Vector in which first element of SubVector is found:
         1...N
      PosInVect: position inside StartVect vector of first element:
         1...DisplVectHeapLong
```

```
  Check:   1.2.3.4.5.6.   DisplNelem=2
           1..2..3..4..   DisplVectHeapLong=3
           1->1,1
           2->1,3
           3->2,2
           4->3,1
)
var
   DisplNElem,PosMinus1: Itype;
begin
   DisplNelem:=Dim;
   PosMinus1 := (DisplNumber-1)*DisplNelem;
   StartVect := Trunc(PosMinus1/DisplVectHeapLong)+1;
   PosInVect := PosMinus1+1-(StartVect-1)*DisplVectHeapLong;
end;
{-------------------------------------------------------------------}
Procedure DisplPlace(    DisplNumber,Dim: IType;
                     var FilePoint: Itype;
                     var InRAM: Boolean);
{
   Input:
      DisplNumber: SubVector number
      Dim: Dimension of problem
   Output:
      FilePoint: File position of first element of SubVector
      InRAM: True if SubVector is kept in RAM
}
begin
   If DisplNumber <= DisplNumberMaxRAM then
      InRAM:=True
   else
      begin
      InRAM:=False;
{
                  0 based
                  Reduced file length for portion in RAM
}
      FilePoint:=(DisplNumber-1-DisplNumberMaxRAM)*Dim;
      end;
end;
{-------------------------------------------------------------------}
Procedure Displ_R_Read;
{
Procedure Displ_R_Read(var F: FileR;
                           DisplNumber,Dim: IType;
                       var V: VectR_D);
}
{
   Input:
      F: File containing SubVectors
      Displ_Number: displacment vector number
      Dim: Dimension of problem
```

```
   Output:
      V: SubVector
         Reads the S SubVector from a file
}
Var
   FilePoint,VectNumber,PosInVect: IType;
   InRAM: Boolean;
begin
   DisplPlace(DisplNumber,Dim,FilePoint,InRAM);
   If InRAM=True then
      begin
      PosInRAM(Dim,DisplNumber,VectNumber,PosInVect);
{
                  The SubVector vector is all contained in one
                     heap vector thanks to the heap vector length being a
                     multiple of (1*2*3)=6.
                  See DisplVectHeapLong = (1*2*3)*N in constant declaration.
}
      Case Dim of
         1: begin
            V[1]:=DisplHeapRAM[VectNumber]^[PosInVect];
            end;
         2: begin
            V[1]:=DisplHeapRAM[VectNumber]^[PosInVect];
            V[2]:=DisplHeapRAM[VectNumber]^[PosInVect+1];
            end;
         3: begin
            V[1]:=DisplHeapRAM[VectNumber]^[PosInVect];
            V[2]:=DisplHeapRAM[VectNumber]^[PosInVect+1];
            V[3]:=DisplHeapRAM[VectNumber]^[PosInVect+2];
            end;
         end;
      end
   else
      begin
      Seek(F,FilePoint);
      Case Dim of
         1: begin
            Read(F,V[1]);
            end;
         2: begin
            Read(F,V[1]);
            Read(F,V[2]);
            end;
         3: begin
            Read(F,V[1]);
            Read(F,V[2]);
            Read(F,V[3]);
            end;
         end;
      end;
end;
```

```
{-------------------------------------------------------------------}
Procedure Displ_R_Write;
{
Procedure Displ_R_Write(var F: FileR;
                            DisplNumber,Dim: IType;
                        var V: VectR_D);                  var for speed
}
{
   Input:
      F: File containing SubVectors
      Displ_Number: displacment vector number
      Dim: Dimension of problem
      V: SubVector
   Output:
      Store the S SubVector into a file
}
var
   FilePoint,VectNumber,PosInVect: IType;
   X_Rtype: Rtype;
   InRAM: Boolean;
begin
   DisplPlace(DisplNumber,Dim,FilePoint,InRAM);
   If InRAM=True then
      begin
      PosInRAM(Dim,DisplNumber,VectNumber,PosInVect);
{
                The SubVector vector is all contained in one
                   heap vector thanks to the heap vector length being a
                   multiple of (1*2*3)=6.
                See DisplVectHeapLong = (1*2*3)*N in constant declaration.
}
      Case Dim of
         1: begin
            DisplHeapRAM[VectNumber]^[PosInVect] := V[1];
            end;
         2: begin
            DisplHeapRAM[VectNumber]^[PosInVect] := V[1];
            DisplHeapRAM[VectNumber]^[PosInVect+1] := V[2];
            end;
         3: begin
            DisplHeapRAM[VectNumber]^[PosInVect] := V[1];
            DisplHeapRAM[VectNumber]^[PosInVect+1] := V[2];
            DisplHeapRAM[VectNumber]^[PosInVect+2] := V[3];
            end;
         end;
      end
   else
      begin
      Seek(F,FilePoint);
      Case Dim of
         1: write(F,V[1]);
         2: begin
```

```
            write(F,V[1]);
            write(F,V[2]);
            end;
         3: begin
            write(F,V[1]);
            write(F,V[2]);
            write(F,V[3]);
            end;
         end;
      end;
end;
{-------------------------------------------------------------------}
Procedure InitDisplHeapRAM;
{
Procedure InitDisplHeapRAM(    Dim,Nnt: Itype;
                           var F: FileR);
}
{
   Input:
      Dim: Dimension of problem
      Nnt: Total number of nodes
      F: File containing SubVectors
      DisplVectHeapLong (Constant): Length of vectors kept in Hcap RAM
      MaxDisplVectHeapRAM (Constant): Number of vectors of length (DisplVectHeapLong)
         kept in Heap RAM
   Output:
      LastDisplVectHeapRAM: Number of vectors of length (DisplVectHeapLong) kept in Heap RAM
      DisplHeapRAM: Pointers for DisplVectHeap kept in heap RAM
      DisplNumberMaxRAM: number of SubVectors stored in RAM
}
var
   i,FilePoint: Itype;
   Xs,Ys: Byte;
   InRAM: Boolean;
   Displ: VectR_D;
begin
   LastDisplVectHeapRAM:=Trunc(MaxAvail/DisplVectHeapLong/SizeOf(Rtype));
   If MaxDisplVectHeapRAM<LastDisplVectHeapRAM then
      LastDisplVectHeapRAM:=MaxDisplVectHeapRAM;
{
                  Get Maximum number of SubVectors: DisplNumberMax
}
   DisplNumberMax:=Nnt;
{
                  Correct last heap vector number used with DisplNumberMax
}
   If Trunc(DisplNumberMax*Dim/DisplVectHeapLong)+1 < LastDisplVectHeapRAM then
      LastDisplVectHeapRAM := Trunc(DisplNumberMax*Dim/DisplVectHeapLong)+1;
{
      DisplNumberMaxRAM: maximum number of SubVectors which can be stored in RAM
}
   DisplNumberMaxRAM:=Trunc(LastDisplVectHeapRAM*DisplVectHeapLong/Dim);
```

```
{
                Correct maximum number of SubVectors kept in RAM
                with maximum SubVector number if inferior
}
   If DisplNumberMax<DisplNumberMaxRAM then DisplNumberMaxRAM := DisplNumberMax;
{
                Show Heap Status
}
   TextColor(LightRed);
   write(Trunc(LastDisplVectHeapRAM*DisplVectHeapLong*SizeOf(Rtype)/1000)+1);
   TextColor(LightGreen);
   writeln(' kB used for heap RAM storage');
   TextColor(LightRed);
   write((DisplNumberMaxRAM/DisplNumberMax)*100:5:1);
   TextColor(LightGreen);
   writeln('% of displacements can be kept in RAM');
{
                Initialize Heap Pointers
                Dispose to Check that Pointers can be disposed
                  (instead of bugging at the complete end)
                  And it only takes 0.16s on a XT at 4.77 MHz
                Initialize Heap Pointers
}
   For i:=1 to LastDisplVectHeapRAM do
      New(DisplHeapRAM[i]);
   For i:=LastDisplVectHeapRAM downto 1 do
      Dispose(DisplHeapRAM[i]);
   For i:=1 to LastDisplVectHeapRAM do
      New(DisplHeapRAM[i]);
{
                Put displacements into RAM
}
   Write('Reading displacements into RAM ');
   Xs:=WhereX;
   Ys:=WhereY;
   For i:=1 to DisplNumberMaxRAM do
      begin
      gotoXY(Xs,Ys);
      write(DisplNumberMaxRAM-i+1,' ');
      D_R_Read(F,i,Dim,Displ);
      Displ_R_Write(F,i,Dim,Displ);
      end;
   gotoXY(Xs,Ys);
   writeln(' ');
end;
{-------------------------------------------------------------------}
Procedure RamDisplToDisk;
{
Procedure RamDisplToDisk(    Dim: Itype;
                         var F: FileR);
}
{
```

```
   Dispose of Heap Pointers
   Input:
      Dim: Dimension of problem
      F: File containing SubVectors
      DisplHeapRAM: Pointers for DisplVectHeap kept in heap RAM
      DisplNumberMaxRAM: number of SubVectors stored in RAM
}
var
   i: Integer;
   Xs,Ys: Byte;
   Displ: VectR_D;
begin
{
                  Put displacements back on DISK
}
   Write('Writing displacements from RAM to disk ');
   Xs:=WhereX;
   Ys:=WhereY;
   For i:=1 to DisplNumberMaxRAM do
      begin
      gotoXY(Xs,Ys);
      write(DisplNumberMaxRAM-i+1,' ');
      Displ_R_Read(F,i,Dim,Displ);
      D_R_Write(F,i,Dim,Displ);
      end;
   gotoXY(Xs,Ys);
   writeln(' ');
end;
{-------------------------------------------------------------------}
Procedure DisposeDisplHeapRAM;
{
Procedure DisposeDisplHeapRAM;
}
{
   Dispose of Heap Pointers
   Input:
      DisplHeapRAM: Pointers for DisplVectHeap kept in heap RAM
      LastDisplVectHeapRAM: Number of vectors of length (DisplVectHeapLong) kept in Heap RAM
}
var
   i: Integer;
begin
{
                  Use RamDisplToDisk to save displacements
                  before disposing of memory for displacements
}
   For i:=LastDisplVectHeapRAM downto 1 do  {reverse release}
      dispose(DisplHeapRAM[i]);
end;
{-------------------------------------------------------------------}
end.
```

DRAW.PAS

```
{$N+}
Unit Draw;
Interface
Uses
   Graph,Dos,Crt,Declare,Nombre,connect,Timer,WaitKey,WhatKey,Linear,Nne_Dim,
   File_RW,Sonore,J_Graph;
Procedure xyz_3d(    Nnt: IType;
                 var F_xyz,F_xyz3D: FileR;
                     Dim: IType;
                 var Modified: Boolean);
Procedure Select(    File_Name: String6;
                     Net,Nnt,NneMax: IType;
                 var F_XYZ: FileR;
                 var F_Elem,F_NewOld,F_Connect: FileI;
                 var Phi,Theta: Rtype;
                 var Numbering,Draw_axis,Draw_Elev: IType);
Procedure Dessine(   Title: String80;
                     Nnt: IType;
                     Phi,Theta: Rtype;
                     Numbering,Draw_axis,Draw_Elev: IType;
                  var F_XYZ: FileR;
                     Zscale: Rtype;
                  var F_NewOld,F_Connect: FileI);
{-------------------------------------------------------------------}
Implementation
Procedure xyz_3d;
{
Procedure xyz_3d(    Nnt: IType;
                 var F_xyz,F_xyz3D: FileR;
                     Dim: IType;
                 var Modified: Boolean);
}
{
   Input:
      Nnt: Number of nodes
      F_xyz: Original file with coordinates of nodes
      Dim: Dimension of geometry
   Output:
      F_xyz3D:  File giving xyz in 3-d from 2-d or 3-d
}
var
   i: IType;
   xyz: VectR_D;
   Xs,Ys: Byte;
begin
   If Modified=True then
      begin
      write('Generating 3 dimensional coordinates ');
      Xs:=WhereX;
      Ys:=WhereY;
```

```
{
                Get original Dimension of problem
}
      For i:=1 to Nnt do
        begin
        gotoXY(Xs,Ys);
        write(Nnt-i+1,' ');
        D_R_Read(F_xyz,i,Dim,xyz);
        If Dim=2 then xyz[3]:=0;
        D_R_write(F_xyz3D,i,3,xyz);
        end;
      gotoXY(Xs,Ys);
      writeln('  ');
      end;
   Modified:=False;
end;
{-------------------------------------------------------------------------}
Procedure Display(var row,col,All: IType;
                  var N_region: IType;
                  var Region: MatrixR_Nrx6;
                  var N_Group: IType;
                  var Group: MatrixI_Nsx4;
                  var Numbering,Draw_axis,Draw_Elev: IType;
                  var Phi,Theta: Rtype;
                  var PlaceX,Field: IType);
{
   Input:
      row,col: position in information matrix (screen display)
      All: =1 if if all the nodes are considered
           =0 otherwise
      N_region: Number of regions
      Region[1..N_region,i]: for 1<=i<=6: [ ]<=X<=[ ] [ ]<=Y<=[ ] [ ]<=Z<=[ ]
      N_Group: number of Groups
      Group[1..N_Group,i]: node [ ] to [] step [ ]
      Numbering: =1 if we want the # displayed with 3-D image at nodes
                 =0 otherwise
      Draw_axis: =1 for drawing axis, =0 otherwise
      Draw_Elev: =1 for drawing lines of elevation, =0 otherwise
      Phi: angle with Z
      Theta: angle with X
      PlaceX: X screen position of in formation in row,col
      Field: Length of field for display
      Couleur: Color for information displayed
   Output:
      Screen updated for information in row,col
}
var
   i: integer;
begin
{
                Erase old information
}
```

```
   Gotoxy(PlaceX,row);
   For i:=1 to Field do
      write(' ');
   Gotoxy(PlaceX,row);
{
}
   Case row of
      1:If All=1 then write('Yes')
                 else write('No');
      2: write(N_region:Field);
      3..5:write(RField(1.0*Region[row-2,col],Field));
      6: write(N_Group:Field);
      7..16: write(Group[row-6,col]:Field);
      17:If Numbering=1 then write('Yes')
                        else write('No');
      18:If Draw_Axis=1 then write('Yes')
                        else write('No');
      19:If Draw_Elev=1 then write('Yes')
                        else write('No');
      20: write(Theta:Field:3);
      21: write(Phi:Field:3);
      end;
end;
{-------------------------------------------------------------------}
Procedure Check_Con(    iNne1,iNne2: IType;
                    var Elem: VectI_Nne2;        { var for speed }
                    var F_Connect: FileI);
{
   Input:
      iNne1,iNne2: node index in element to be connected
      Elem[1..Net+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...) (can be negative)
      F_Connect: File Giving Connection for each node
                 to the nodes Con[1..Con], =0 for last connection
   Output:
      F_Connect: File Giving Connection for each node
                 to the nodes Con[1..Con], =0 for last connection
         Correct Connect to add this connection if Node2 is selected
}
var
   iCon,Pos_free,keep,Node1,Node2: IType;
   Connect_ok,Found_con: Boolean;
   Connect: VectI_Con;
begin
   Node1:=abs(Elem[2+iNne1]);
   Node2:=abs(Elem[2+iNne2]);
   If Node2<Node1 then
      begin
      keep:=Node1;    { exchange node number }
      Node1:=Node2;
      Node2:=keep;
```

```
    end;
{
                When Connect vectors where initialized, Connect[1]=-1
                But considered node is only selected when Connect[1]<>-1
}
  Connect_ok:=True;
  Con_I_Read(F_Connect,Node2,Connect);     (look if node 2 is selected)
  If Connect[1]=-1 then Connect_ok:=False;
  If Connect_ok=True then
    begin
    Con_I_Read(F_Connect,Node1,Connect);     (look if node 1 is selected)
    If Connect[1]=-1 then Connect_ok:=False;
    end;
  If Connect_ok=True then
    begin
{
                Look if connection is already established
}
    Found_con:=False;
    iCon:=1;
    While (iCon<=ConnectMax) and (Connect[iCon]<>0) do
      begin
      If Connect[iCon]=Node2 then Found_con:=True;
      Inc(iCon);
      end;
{
                Establish new connection
}
    If Found_con=False then
      begin
      Pos_free:=0;
      iCon:=1;
      While (iCon<=ConnectMax) and (Pos_free=0) do
        begin
        If Connect[iCon]=0 then Pos_free:=iCon;
        Inc(iCon);
        end;
      If 0<Pos_free then
        begin
        Connect[pos_free]:=Node2;
        If Pos_Free<ConnectMax then Connect[Pos_Free+1]:=0;
        Con_I_Write(F_Connect,Node1,Connect);
        end;
      end;
    end;
end;
{-------------------------------------------------------------------}
Procedure Select;
{
Procedure Select(    File_Name: String6;
                     Net,Nnt,NneMax: IType;
                var F_XYZ: FileR;
```

```
                var F_Elem,F_NewOld,F_Connect: FileI;
                var Phi,Theta: Rtype;
                var Numbering,Draw_axis,Draw_Elev: IType);
}
{
   Input:
      File_Name: Name of File where previous information was stored
      Net: Number of elements
      Nnt: Number of nodes
      NneMax: Maximum number of nodes in an element
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1]= Biggy for undefined position
          initialized as expansion needs it
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Node are negative in Element matrix when they are use for
              the last time according to the numbering of the elements
      F_NewOld: original node number given by user for each "compacted" node #
   Output:
      F_Connect: File Giving Connection for each node
                 Connected to nodes Connect[1..Con], =0 for last connection
                 Connect[1]=-1 if node not selected
      Phi: angle with Z
      Theta: angle with X
      Numbering: =1 if we want the # displayed with 3-D image at nodes
                 =0 otherwise
      Draw_axis: =1 for drawing axis, =0 otherwise
      Draw_Elev: =1 for drawing lines of elevation, =0 otherwise
   Example:
      Relations:
         Nnt: 13
         Nodes selected: 1 3 8 13
         Connections: 3-7 3-8 8-3 10-13
      Ends up as:
            i          1   2   3   4   5   6   7   8   9  10  11  12  13
         Connect[1] =  0  -1   8  -1  -1  -1  -1   0  -1  -1  -1  -1   0
         Connect[2] =  0   0   0   0   0   0   0   0   0   0   0   0   0
         Connect[3] =  0   0   0   0   0   0   0   0   0   0   0   0   0
         ...
         Connect[ConnectMax] = 0 ... 0
}
var
   DatI: FileI;
   DatR: FileR;
   i,N_region,N_Group,row,col,New_row,New_col,
      All,Nne,NumberI,iStep,iRegion,iGroup,iNode,iD,iE,
      iNne,J,Node_Old,Add,Node,Etype: IType;
   Xs,Ys,Error_Type: Integer;
   NumberR: Rtype;
   Region: MatrixR_Nrx6;
   Group: MatrixI_Nsx4;
```

```
   Name: String[255];
   NodeSelected,Inside: Boolean;
   PlaceX: MatrixI_21x6;
   Many,Field: VectI_21;
   Action: String6;
   C: Char;
   Mot: String[255];
   xyz: VectR_D;
   Elem: VectI_Nne2;
   Connect: VectI_Con;
   IRtype: Byte;
begin
   ResetTimer(1);
   StartTimer(1);
{
                Assign names  to files
}
   Name:=C_Dir+File_Name+'_I.P';
   Assign(DatI,Name);
   Name:=C_Dir+File_Name+'_R.P';
   Assign(DatR,Name);
{
                Get All         (ITypes)
                    Numbering
                    Draw_Axis
                    Draw_Elev
                    N_region
                    N_Group
                    Group[1..N_Group,1..3]
}
   {$I-}
   Reset(DatI);
   {$I+}
   If IOresult=0 then
      begin
      Read(DatI,All);
      Read(DatI,Numbering);
      Read(DatI,Draw_Axis);
      Read(DatI,Draw_Elev);
      Read(DatI,N_region);
      Read(DatI,N_Group);
      For i:=1 to N_Group_max do
         For j:=1 to 4 do
            read(DatI,Group[i,j]);
      Close(DatI);
      end
   else
      begin
      All:=1;        { yes }
      Numbering:=0;  { no }
      Draw_Axis:=1;  { yes }
      Draw_Elev:=0;  { no }
```

```
      N_region:=0;
      N_Group:=0;
      For i:=1 to N_Group_max do
         For j:=1 to 4 do
            Group[i,j]:=0;
      end;
{
                  Get Region[1..N_region,1..6]  Rtypes
}
   {$I-}
   Reset(DatR);
   {$I+}
   If IOresult=0 then
      begin
      read(DatR,Phi);
      read(DatR,Theta);
      For i:=1 to N_region_max do
         For j:=1 to 6 do
            read(DatR,Region[i,j]);
      Close(DatR);
      end
   else
      begin
      Theta:=45;
      Phi:=45;
      For i:=1 to N_region_max do
         For j:=1 to 6 do
            Region[i,j]:=0;
      end;
{
                  Write Display
}
   StopTimer(1);
   If 60<ReadTimer(1) then Alert;
   ClrScr;
   TextColor(LightGreen);
   Writeln('1- All the nodes: __');
   Writeln('2- Number of regions: _');
   Writeln('   1- _______<X<_______, _______<Y<_______, _______<Z<_______');
   Writeln('   2- _______<X<_______, _______<Y<_______, _______<Z<_______');
   Writeln('   3- _______<X<_______, _______<Y<_______, _______<Z<_______');
   Writeln('3- Number of Groups: _');
   Writeln('   1- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   2- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   3- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   4- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   5- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   6- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   7- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   8- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('   9- node _____ to _____ , repeat _____ times by step of _____ ');
   Writeln('  10- node _____ to _____ , repeat _____ times by step of _____ ');
```

```
   Writeln('4- Node numbering: __');
   Writeln('5- Draw the axis: __');
   Writeln('6- Draw elevation: __');
   Writeln('7- Theta (angle with X): _______ (degrees)');
   Writeln('8- Phi (angle with Z): _______ (degrees)');
   writeln;
   write('Press <');
   TextColor(LightRed);
   write('F2');
   TextColor(LightGreen);
   write('> to change from Yes to No. Press <');
   TextColor(LightRed);
   write('F10');
   TextColor(LightGreen);
   writeln('> when finished.');
   write('Use <');
   TextColor(LightRed);
   write('arrows');
   TextColor(LightGreen);
   write('>, <');
   TextColor(LightRed);
   write('Home');
   TextColor(LightGreen);
   write('> and <');
   TextColor(LightRed);
   write('End');
   TextColor(LightGreen);
   writeln('> to move.');
   write('Nodes considered are the sum of items 1, 2 and 3');
{
                  Get Place to write numbers
}
   PlaceX[1,1]:= 19;
   PlaceX[2,1]:= 23;
   For i:=3 to 5 do
      begin
      PlaceX[i,1]:=7;
      PlaceX[i,2]:=19;
      PlaceX[i,3]:=30;
      PlaceX[i,4]:=42;
      PlaceX[i,5]:=53;
      PlaceX[i,6]:=65;
      end;
   PlaceX[6,1]:= 22;
   For i:=7 to 16 do
      begin
      PlaceX[i,1]:=12;
      PlaceX[i,2]:=21;
      PlaceX[i,3]:=36;
      PlaceX[i,4]:=59;
      end;
   PlaceX[17,1]:= 20;
```

```
   PlaceX[18,1]:= 19;
   PlaceX[19,1]:= 20;
   PlaceX[20,1]:= 26;
   PlaceX[21,1]:= 24;
{
                Get number of informations to give per line
}
   Many[1]:=1;
   Many[2]:=1;
   For i:=3 to 5 do
      Many[i]:=6;
   Many[6]:=1;
   For i:=7 to 16 do
      Many[i]:=4;
   For i:=17 to 21 do
      Many[i]:= 1;
{
                Get length of field for each line
}
   Field[1]:=3;
   Field[2]:=1;
   For i:=3 to 5 do
      Field[i]:=9;
   Field[6]:=2;
   For i:=7 to 16 do
      Field[i]:=5;
   Field[17]:=3;
   Field[18]:=3;
   Field[19]:=3;
   Field[20]:=8;
   Field[21]:=8;
{
                Fill Display
}
   TextColor(LightRed);
   For row:=1 to 21 do
      For col:=1 to Many[row] do
         Display(row,col,All,N_region,Region,N_Group,Group,Numbering,
                 Draw_Axis,Draw_Elev,Phi,Theta,PlaceX[row,col],Field[row]);
{
                 Set at first position
}
   Row:=1;
   Col:=1;
   TextColor(LightRed+Blink);
   TextBackground(LightBlue);
   Display(row,col,All,N_region,Region,N_Group,Group,Numbering,
           Draw_Axis,Draw_Elev,Phi,Theta,PlaceX[row,col],Field[row]);
{
                Modify Values
}
   Action:='NUL';
```

```
  While action<>'F10' do
     begin
     New_row:=row;
     New_col:=Col;
     Get_Key(C,Action);
{
                  Act upon Arrow movement
}
     If Action='HOME' then
        begin
        New_Row:=1;
        New_Col:=1;
        end;
     If Action='END' then
        begin
        New_Row:=21;
        New_Col:=1;
        end;
     If Action='UP' then
        Case row of
           3,7:
              begin
              New_Row:=Row-1;
              New_Col:=1;
              end;
           2,4..6,8..21:
              begin
              New_Row:=Row-1;
              New_Col:=Col;
              end;
           end;
     If Action='DOWN' then
        Case row of
           5,16:
              begin
              New_Row:=Row+1;
              New_Col:=1;
              end;
           1..4,6..15,17..20:
              begin
              New_Row:=Row+1;
              New_Col:=Col;
              end;
           end;
     If Action='LEFT' then
        If 1<Col then New_col:=Col-1;
     If Action='RIGHT' then
        If Col<Many[row] then New_col:=Col+1;
{
                  Change from Yes to No
}
     If Action='F2' then
```

```
            Case Row of
               1: begin
                  If All=1 then
                     All:=0
                  else
                     All:=1;
                  end;
               17:begin
                  If Numbering=1 then
                     Numbering:=0
                  else
                     Numbering:=1;
                  end;
               18:begin
                  If Draw_axis=1 then
                     Draw_axis:=0
                  else
                     Draw_axis:=1;
                  end;
               19:begin
                  If Draw_Elev=1 then
                     Draw_Elev:=0
                  else
                     Draw_Elev:=1;
                  end;
               end;
{
                    Capture Number
}
         If (C<>chr(0)) and (row in [2..16,20..21]) then
            begin
            TextColor(Yellow);
            TextBackground(Black);
            Mot:=C;
            GotoXY(60,1);
            write(Mot,' ');
            GotoXY(60+Length(Mot),1);
            While (C<>chr(13)) and (Length(Mot)<20) do
               begin
               Repeat until keypressed;
               C:=ReadKey;
               If C in ['0'..'9','.','e','E','+','-'] then
                  Mot:=Mot+C;
               If C=chr(8) then
                  Delete(Mot,Length(Mot),1);
               GotoXY(60,1);
               write(Mot,' ');
               GotoXY(60+Length(Mot),1);
               end;
            gotoxy(60,1);
            write('                    ');
{
```

```
                Store Number
}
        StoX(Mot,NumberI,NumberR,IRtype);
        If IRtype=1 then              { IType }
           Case row of
              2: If NumberI in [0..3] then N_region:=NumberI;
              6: If NumberI in [0..10] then N_Group:=NumberI;
              7..16: Group[row-6,col]:=NumberI;
              3..5: Region[row-2,col]:=NumberI;  { accept IType for Rtype }
              20: Theta:=NumberI;
              21: Phi:=NumberI;
              end;
        If IRtype=2 then              { Rtype }
           Case row of
              3..5: Region[row-2,col]:=NumberR;
              20: Theta:=NumberR;
              21: Phi:=NumberR;
              end;
        end;
{
                Update screen
}
      TextColor(LightRed);
      TextBackground(Black);
      Display(row,col,All,N_region,Region,N_Group,Group,Numbering,
              Draw_Axis,Draw_Elev,Phi,Theta,PlaceX[row,col],Field[row]);
      Col:=New_Col;
      Row:=New_Row;
      TextColor(LightRed+Blink);
      TextBackground(LightBlue);
      Display(row,col,All,N_region,Region,N_Group,Group,Numbering,
              Draw_Axis,Draw_Elev,Phi,Theta,PlaceX[row,col],Field[row]);
      end;
{
                Display Message
}
   TextColor(LightGreen);
   TextBackground(Black);
   ClrScr;
   ResetTimer(1);
   StartTimer(1);
   Writeln('Saving Selection');
{
                Save All        (ITypes)
                     Numbering
                     Draw_Axis
                     Draw_Elev
                     N_region
                     N_Group
                     Group[1..N_Group,1..3]
}
   Rewrite(DatI);
```

```
   write(DatI,All);
   write(DatI,Numbering);
   write(DatI,Draw_Axis);
   write(DatI,Draw_Elev);
   Write(DatI,N_region);
   Write(DatI,N_Group);
   For i:=1 to N_Group_max do
     For j:=1 to 4 do
        Write(DatI,Group[i,j]);
   Close(DatI);
{
                Save Phi
                     Theta
                     Region[1..N_region,1..6]  Rtypes
}
   Rewrite(DatR);
   write(DatR,Phi);
   write(DatR,Theta);
   For i:=1 to N_region_max do
     For j:=1 to 6 do
        Write(DatR,Region[i,j]);
   Close(DatR);
{- - - - - - Nodes - - - - - - - - - - - - - - - - - - -}
{
                Initialize connection vector
}
   write('Initializing ');
   Xs:=WhereX;
   Ys:=WhereY;
   For iNode:=1 to Nnt+6 do   {+6 for axis}
     begin
     gotoXY(Xs,Ys);
     write(Nnt+6-iNode+1,' ');
     Con_I_O(F_connect,iNode)
     end;
   gotoXY(Xs,Ys);
   writeln(' ');
{
                Fill Node vector
}
   NodeSelected:=False;
   Write('Finding valid node numbers ');
   Xs:=WhereX;
   Ys:=WhereY;
   If All=1 then
     begin
     NodeSelected:=True;
     For iNode:=1 to Nnt do
        begin
        gotoXY(Xs,Ys);
        write(Nnt-iNode+1,' ');
        Connect[1]:=0;          {node is selected, erase connect[1]=-1}
```

```
        Con_I_write(F_Connect,iNode,Connect);
        end;
      end
    else
      begin
{
                Consider all Regions for Regions
}
      If 0<N_region then
        begin
        For iNode:=1 to Nnt do
          begin
          gotoXY(Xs,Ys);
          write(Nnt-iNode+1,' ');
          D_R_Read(F_xyz,iNode,dim3,xyz);
          Inside:=False;
          iRegion:=1;
          While (iRegion<=N_region) and (Inside=False) do
            begin
            Inside:=True;
            iD:=1;
            While iD<=dim3 do
              begin
              If not ((Region[iRegion,Lin(iD,1,2)]<=XYZ[iD]) and
                      (XYZ[iD]<=Region[iRegion,Lin(iD,2,2)])) then
                Inside:=False;
              Inc(iD);
              end;
            Inc(iRegion);
            end;
          If Inside=True then
            begin
            NodeSelected:=True;
            Connect[1]:=0;          {node is selected, erase connect[1]=-1}
            Con_I_write(F_Connect,iNode,Connect);
            end;
          end;
        end;
{
                Consider all Nodes for node groups
}
      If 0<N_Group then
        begin
        For iNode:=1 to Nnt do
          begin
          gotoXY(Xs,Ys);
          write(Nnt-iNode+1,' ');
          I_Read(F_NewOld,iNode,Node_old);
          iGroup:=1;
          While iGroup<=N_Group do
            begin
            iStep:=0;
```

```
                While iStep<=Group[iGroup,3] do
                   begin
                   Add:=iStep*Group[iGroup,4];
                   If (Group[iGroup,1]+Add<=Node_old) and
                       (Node_Old<=Group[iGroup,2]+add) then
                      begin
                      NodeSelected:=True;
                      Connect[1]:=0;        (node is selected, erase connect[1]=-1)
                      Con_I_write(F_Connect,iNode,Connect);
                      iStep:=Group[iGroup,3];          { exit loop }
                      iGroup:=N_Group;
                      end;
                   iStep:=iStep+1;
                   end;
                iGroup:=iGroup-1;
                end;
             end;
          end;
       end;
    gotoXY(Xs,Ys);
    writeln('  ');
(- - - - - - Connections - - - - - - - - - - - - - - - - - -)
{
                Initialize Connect vector
}
    Write('Establishing connections between nodes ');
    Xs:=WhereX;
    Ys:=WhereY;
{
                Establish connections between chosen nodes
}
    For iE:=1 to Net do
       begin
       GotoXY(Xs,Ys);
       write(Net-iE+1,' ');
       Nne2_I_Read(F_Elem,iE,NneMax,Elem);
       Etype:=Elem[2];
       Nne:=Get_Nne(Etype);
       Case Etype of
{
                Relations: 1 - 2,3   For Element type 1
                           2 - 3
}
          1: begin
             Check_Con(1,2,Elem,F_Connect);
             Check_Con(1,3,Elem,F_Connect);
             Check_Con(2,3,Elem,F_Connect);
             end;
{
                Relations: 1 - 2,8,9   For Element type 2
                           2 - 3
                           3 - 4,10
```

```
                                4 - 5
                                5 - 6,11
                                6 - 7
                                7 - 8,12
                                9 - 13    no 8
                                10 - 15
                                11 - 17
                                12 - 19
                                13 - 14,20
                                14..19 - i+1
}
        2: Begin
           Check_Con(1,2,Elem,F_Connect);
           Check_Con(1,8,Elem,F_Connect);
           Check_Con(1,9,Elem,F_Connect);
           Check_Con(2,3,Elem,F_Connect);
           Check_Con(3,4,Elem,F_Connect);
           Check_Con(3,10,Elem,F_Connect);
           Check_Con(4,5,Elem,F_Connect);
           Check_Con(5,6,Elem,F_Connect);
           Check_Con(5,11,Elem,F_Connect);
           Check_Con(6,7,Elem,F_Connect);
           Check_Con(7,8,Elem,F_Connect);
           Check_Con(7,12,Elem,F_Connect);
           Check_Con(9,13,Elem,F_Connect);
           Check_Con(10,15,Elem,F_Connect);
           Check_Con(11,17,Elem,F_Connect);
           Check_Con(12,19,Elem,F_Connect);
           Check_Con(13,14,Elem,F_Connect);
           Check_Con(13,20,Elem,F_Connect);
           Check_Con(14,15,Elem,F_Connect);
           Check_Con(15,16,Elem,F_Connect);
           Check_Con(16,17,Elem,F_Connect);
           Check_Con(17,18,Elem,F_Connect);
           Check_Con(18,19,Elem,F_Connect);
           Check_Con(19,20,Elem,F_Connect);
           end;
        end;
     end;
{
                Get Phi and Theta in radians
}
   Phi:=Phi*Pi/180;
   Theta:=Theta*Pi/180;
{
                Verify that a node has been selected
}
   If NodeSelected=False then
     begin
     ClrScr;
     writeln('Error: no coordinates selected for drawing');
     writeln('Possible cure: Review region limits along with geometry dimensions');
```

```
      writeln('              Use NUMBERS process for geometry dimensions');
      wait;
      Halt;
      end;
{
                  Call the user
}
   StopTimer(1);
   If 60<ReadTimer(1) then Alert;
end;
{-------------------------------------------------------------------------}
Procedure Dessine;
{
Procedure Dessine(    Title: String80;
                      Nnt: IType;
                      Phi,Theta: Rtype;
                      Numbering,Draw_axis,Draw_Elev: IType;
                  var F_XYZ: FileR;
                      Zscale: Rtype;
                  var F_NewOld,F_Connect: FileI);
}
{
   Input:
      Title: Title of graph
      Nnt: Number of nodes
      Phi: angle with X
      Theta: angle with Z
      Numbering: =1 if we want the # displayed with 3-D image at nodes
                 =0 otherwise
      Draw_axis: =1 for drawing axis, =0 otherwise
      Draw_Elev: =1 for drawing lines of elevation, =0 otherwise
      F_XYZ[1..3*Nnt]: File with X Y Z of nodes
      Zscale: Scale factor for third dimension (=1 to keep it the same)
      F_NewOld: original node number given by user for each "compacted" node #
      F_Connect: File Giving Connection for each node
                 Connected to nodes Connect[1..Con], =0 for last connection
                 Connect[1]=-1 if node not selected
   Output:
      Draw 3-D graph of geometry
}
var
   Node,Old,SXmax,SXmin,SYmax,SYmin,NodesAdded,i,Node_max,iD,iNode: IType;
   BarLongX,SpaceBarX,dY,BarMin,BarMax,MaxScaleY,MinScaleY: Itype;
   Ratio,a1,a2,Amplitude,Margin,S1,S2,C1,C2,X,Y: Rtype;
   Max,Min: array[1..3] of Rtype;
   Sign: array[1..2] of Integer;
   Name: array[1..3] of string6;
   Mot: String6;
   Message: String255;
   grDriver, grMode: Integer;
   Xasp,Yasp: word;
   xyz,xy0,xyz2: VectR_D;
```

```
   Connect: VectI_Con;
   F_xyO: FileR;
{    Clock related variables }
   DisplayInc: Integer;
   Angle,Radius: Rtype;
   Xcenter,Ycenter: word;
begin
{
                  Get Graphic Setup
}
   GrDriver:=Detect;
   InitGraph(grDriver,grMode,'');
   SXmax:=GetMaxX;
   SXmin:=0;
   SYmax:=0;
   SYmin:=GetMaxY;
{
                  Get sign of screen axis
                  and ratio of X/Y for a square
}
   Sign[1]:=Round((SXmax-SXmin)/abs(SXmax-SXmin));
   Sign[2]:=Round((SYmax-SYmin)/abs(SYmax-SYmin));
   GetAspectRatio(Xasp,Yasp);
   Ratio:=Yasp/Xasp;
   Message:='Scaling graph':
   OutTextXY(0,0,Message);
{
                  Initialize Max Min in all Directions
}
   For iD:=1 to 3 do
      begin
      max[iD]:=-1E30;
      min[iD]:=1E30;
      end;
{
                  Find maximum minimum of selected nodes
                  Do not use Zscale here
}
   For iNode:=1 to Nnt do
      begin
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then
         begin
         D_R_Read(F_xyz,iNode,3,xyz);
         For iD:=1 to 3 do
            begin
            If XYZ[iD]<Min[iD] then Min[iD]:=XYZ[iD];
            If Max[iD]<XYZ[iD] then Max[iD]:=XYZ[iD];
            end;
         end;
      end;
{
```

```
                Add maximum elevation value to title
}
   If Draw_Elev=1 then
      Title:=Title+' ('+Rword(Min[3],9)+' to '+Rword(Max[3],9)+')';
{
                If 0<MinZ then set MinZ=0
                If MaxZ<0 then set MaxZ=0
}
   If Draw_Elev=1 then
      begin
      If 0.0<Min[3] then Min[3]:=0.0;
      If Max[3]<0.0 then Max[3]:=0.0;
      end;
{
                If There is no variation in Z, Give MaxZ as a portion
                of the smallest variation in the other two dimensions
}
   If Max[3]-Min[3]=0 then
      If (Max[1]-Min[1])<=(Max[2]-Min[2]) then
         Max[3]:=Max[3]+0.25*(Max[1]-Min[1])
      else
         Max[3]:=Max[3]+0.25*(Max[2]-Min[2]);
{
                add axis to geometry XYZ
                Endpoints of axis are XYZ[Node[Nnt+2 +4 +6],1..3]
                With nodes Nnt+1 and Nnt+2 connected
                           Nnt+3     Nnt+4
                           Nnt+5     Nnt+6
}
   If Draw_axis=1 then
      begin
      For iD:=1 to 3 do XYZ[iD]:=0;
      For i:=1 to 6 do
         D_R_Write(F_xyz,Nnt+i,3,xyz);
{
                Define origin positions
}
      For iD:=1 to 3 do XYZ[iD]:=0;
      XYZ[1]:=Min[1]-(Max[1]-Min[1])*0.05;
      D_R_Write(F_xyz,Nnt+1,3,xyz);
      For iD:=1 to 3 do XYZ[iD]:=0;
      XYZ[2]:=Min[2]-(Max[2]-Min[2])*0.05;
      D_R_Write(F_xyz,Nnt+3,3,xyz);
      For iD:=1 to 3 do XYZ[iD]:=0;
      XYZ[3]:=Min[3]-(Max[3]-Min[3])*0.05;
      D_R_Write(F_xyz,Nnt+5,3,xyz);
{
                Define axis extremity
}
      For iD:=1 to 3 do XYZ[iD]:=0;
      XYZ[1]:=Max[1]+(Max[1]-Min[1])*0.1;
      D_R_Write(F_xyz,Nnt+2,3,xyz);
```

```
      For iD:=1 to 3 do XYZ[iD]:=0;
      XYZ[2]:=Max[2]+(Max[2]-Min[2])*0.1;
      D_R_Write(F_xyz,Nnt+4,3,xyz);
      For iD:=1 to 3 do XYZ[iD]:=0;
      XYZ[3]:=Max[3]+(Max[3]-Min[3])*0.1;
      D_R_Write(F_xyz,Nnt+6,3,xyz);
      Name[1]:='X';
      Name[2]:='Y';
      Name[3]:='Z';
{
                  Y and Z translation when 0<MinX or MaxX<0
}
      If 0<Min[1] then
         begin
         Name[2]:='Y'+chr(39);
         Name[3]:='Z'+chr(39);
         For i:=3 to 6 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[1]:=Min[1];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         end;
      If Max[1]<0 then
         begin
         Name[2]:='Y'+chr(39);
         Name[3]:='Z'+chr(39);
         For i:=3 to 6 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[1]:=Max[1];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         end;
{
                  X and Z translation when 0<MinY or MaxY<0
}
      If 0<Min[2] then
         begin
         Name[1]:='X'+chr(39);
         Name[3]:='Z'+chr(39);
         For i:=1 to 2 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[2]:=Min[2];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         For i:=5 to 6 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[2]:=Min[2];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
```

```
            end;
         end;
      If Max[2]<0 then
         begin
         Name[1]:='X'+chr(39);
         Name[3]:='Z'+chr(39);
         For i:=1 to 2 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[2]:=Max[2];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         For i:=5 to 6 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[2]:=Max[2];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         end;
{
                  X and Y translation when 0<MinZ or MaxZ<0
}
      If 0<Min[3] then
         begin
         Name[1]:='X'+chr(39);
         Name[2]:='Y'+chr(39);
         For i:=1 to 4 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[3]:=Min[3];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         end;
      If Max[3]<0 then
         begin
         Name[1]:='X'+chr(39);
         Name[2]:='Y'+chr(39);
         For i:=1 to 4 do
            begin
            D_R_Read(F_xyz,Nnt+i,3,xyz);
            XYZ[3]:=Max[3];
            D_R_Write(F_xyz,Nnt+i,3,xyz);
            end;
         end;
{
                  Connect axis extremities
}
      Connect[1]:=Nnt+2;
      Connect[2]:=0;
      Con_I_Write(F_Connect,Nnt+1,Connect);
      Connect[1]:=0;
      Con_I_Write(F_Connect,Nnt+2,Connect);   (node selected)
```

```
      Connect[1]:=Nnt+4;
      Connect[2]:=0;
      Con_I_Write(F_Connect,Nnt+3,Connect);
      Connect[1]:=0;
      Con_I_Write(F_Connect,Nnt+4,Connect);   (node selected)
      Connect[1]:=Nnt+6;
      Connect[2]:=0;
      Con_I_Write(F_Connect,Nnt+5,Connect);
      Connect[1]:=0;
      Con_I_Write(F_Connect,Nnt+6,Connect);   (node selected)
      end;
{
                  Get number of added nodes when we add axis
}
   If Draw_axis=1 then
      NodesAdded:=6
   else
      NodesAdded:=0;
{
                  Convert to 2D, no perspective (D=rho=infinity)
}
   S1:=sin(theta);
   C1:=cos(theta);
   S2:=sin(phi);
   C2:=cos(phi);
{
                  Assign (X Y 0) coordinates to a file
}
   If Draw_Elev=1 then
      begin
      Assign(F_xy0,T_Dir+'xy0.T');
      Rewrite(F_xy0);
{
                  Initialize 0 elevation coordinates
                  (not all nodes used or calculated)
}
      xy0[1]:=0;
      xy0[2]:=0;
      xy0[3]:=0;
      For iNode:=1 to Nnt+NodesAdded do
         D_R_write(F_xy0,iNode,2,xy0);
      end;
{
                  Initialize 2-D Max min
}
   For iD:=1 to 2 do
      begin
      max[iD]:=-1E30;
      min[iD]:=1E30;
      end;
{
                  Find position of projection and Max Min
```

```
)
   Xcenter:=Round((SXMin+SXmax)/2);
   Ycenter:=Round((SYMin+SYmax)/2);
   If abs(Xcenter)<abs(Ycenter) then
      Radius:=abs(Xcenter/3)
   else
      Radius:=abs(Ycenter/3);
   DisplayInc:=Trunc((Nnt+NodesAdded)/60);
   If DisplayInc<1 then DisplayInc:=1;
   For iNode:=1 to Nnt+NodesAdded do
      begin
{
                  Draw countdown clock
}
      If abs((iNode/DisplayInc)-Round(iNode/DisplayInc))<1E-4 then
         begin
         Angle:=Pi/2-2*Pi*(Nnt+NodesAdded-iNode+1)/(Nnt+NodesAdded);
         X:=Xcenter+sign[1]*Ratio*Radius*Cos(Angle);
         Y:=Ycenter+sign[2]*Radius*Sin(Angle);
         Line(Xcenter,Ycenter,Round(X),Round(Y));
         end;
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then   {nodes selected}
         begin
         D_R_Read(F_xyz,iNode,3,xyz);
{
                  Correct 3rd direction coordinates
}
         xyz[3]:=xyz[3]*Zscale;
         If (Draw_Elev=1) and (iNode<=Nnt) then
            begin
{
                  Find position For (X Y 0) of node
}
            X:=-XYZ[1]*S1+XYZ[2]*C1;
            Y:=-XYZ[1]*C1*C2-XYZ[2]*S1*C2;   { Z=0 }
            xy0[1]:=X*ratio;
            xy0[2]:=Y;
            xy0[3]:=0;
            D_R_write(F_xy0,iNode,2,xy0);
{
                  Get Max Min
}
            If xy0[1]<Min[1] then Min[1]:=xy0[1];
            If Max[1]<xy0[1] then Max[1]:=xy0[1];
            If xy0[2]<Min[2] then Min[2]:=xy0[2];
            If Max[2]<xy0[2] then Max[2]:=xy0[2];
            end;
{
                  Find position For (X Y Z) of node
}
         X:=-XYZ[1]*S1+XYZ[2]*C1;
```

```
        Y:=-XYZ[1]*C1*C2-XYZ[2]*S1*C2+XYZ[3]*S2;
        XYZ[1]:=X*ratio;
        XYZ[2]:=Y;
        XYZ[3]:=0;
        D_R_write(F_xyz,iNode,3,xyz);
{
                Find Max min
}
        For iD:=1 to 2 do
          begin
          If XYZ[iD]<Min[iD] then Min[iD]:=XYZ[iD];
          If Max[iD]<XYZ[iD] then Max[iD]:=XYZ[iD];
          end;
        end;
      end;
{
                Leave a margin for Max Min
}
   Margin:=0.025*(max[1]-Min[1]);   { X margin } {0.025 for = 2/80 = 2 col.}
   max[1]:=max[1]+Margin;
   min[1]:=min[1]-Margin;
   Margin:=0.03*(max[2]-Min[2]);   { Y margin } {0.03 for > 1/25/2=half a line}
   If Title<>'' then max[2]:=max[2]+Margin+(max[2]-min[2])/20
               else max[2]:=max[2]+Margin;
   min[2]:=min[2]-Margin;
{
                Find appropriate amplitude (lowest ratio)
                in order to keep proportion in geometry
                Take Sign in account
                Check for a 0<MAX-MIN in each direction
}
   If abs(max[1]-min[1])<1/BigR then
      begin
      CloseGraph;    {Free heap memory used for graphics}
      DisposeConnectHeapRAM;
      writeln('Choose another viewpoint because the is no thickness in the');
      writeln('X direction for the selected geometry');
      wait;
      Halt;
      end;
   If abs(max[2]-min[2])<1/BigR then
      begin
      CloseGraph;    {Free heap memory used for graphics}
      DisposeConnectHeapRAM;
      writeln('Choose another viewpoint because the is no thickness in the');
      writeln('Y direction for the selected geometry');
      wait;
      Halt;
      end;
   a1:=(SXmax-SXmin)/(max[1]-min[1]);
   a2:=(SYmax-SYmin)/(max[2]-min[2]);
   If abs(a1)<=abs(a2) then Amplitude:=abs(a1)
```

```
                    else Amplitude:=abs(a2);
{
                Draw lines
}
   MaxScaleY:=-BigImax;
   MinScaleY:= BigImax;
   ClearDevice;
   For iNode:=1 to Nnt+NodesAdded do
      begin
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then  (for selected nodes)
         begin
         D_R_Read(F_xyz,iNode,3,xyz);
{
                Scale and center coordinates on screen
}
         XYZ[1]:=(XYZ[1]-(max[1]+min[1])/2)*Sign[1]*Amplitude+(SXmax+SXmin)/2;
         XYZ[2]:=(XYZ[2]-(max[2]+min[2])/2)*Sign[2]*Amplitude+(SYmax+SYmin)/2;
{
                Draw lines according to Connect
}
         i:=1;
         While (Connect[i]<>0) and (i<=ConnectMax) do
            begin
            D_R_Read(F_xyz,Connect[i],3,xyz2);
            xyz2[1]:=(xyz2[1]-(max[1]+min[1])/2)
               *Sign[1]*Amplitude+(SXmax+SXmin)/2;
            xyz2[2]:=(xyz2[2]-(max[2]+min[2])/2)
               *Sign[2]*Amplitude+(SYmax+SYmin)/2;
            Line(Round(XYZ[1]),Round(XYZ[2]),Round(xyz2[1]),Round(xyz2[2]));
            i:=i+1;
            end;
         If iNode<=Nnt then
            begin
{
                Draw Elevation
}
            If Draw_Elev=1 then
               begin
               D_R_Read(F_xy0,iNode,2,xy0);
               xy0[1]:=(xy0[1]-(max[1]+min[1])/2)
                      *Sign[1]*Amplitude+(SXmax+SXmin)/2;
               xy0[2]:=(xy0[2]-(max[2]+min[2])/2)
                      *Sign[2]*Amplitude+(SYmax+SYmin)/2;
               Line(Round(XYZ[1]),Round(XYZ[2]),Round(xy0[1]),Round(xy0[2]));
{
                   Retain maximum and minimum of elevation in screen coordinate
}
               dY:=Sign[2]*(Round(XYZ[2])-Round(xy0[2]));
               If MaxScaleY < dY then MaxScaleY:=dY;
               If dY < MinScaleY then MinScaleY:=dY;
               end;
```

```
{
                Place node numbers
}
          If Numbering=1 then
             begin
             I_Read(F_NewOld,iNode,Old);
             str(Old,Mot);
             Put_Mot(Mot,XYZ[1],XYZ[2]);
             end;
          end;
{
                Put axis letters
}
        If Draw_axis=1 then
          begin
          If iNode=Nnt+2 then Put_Mot(Name[1],XYZ[1],XYZ[2]);
          If iNode=Nnt+4 then Put_Mot(Name[2],XYZ[1],XYZ[2]);
          If iNode=Nnt+6 then Put_Mot(Name[3],XYZ[1],XYZ[2]);
          end;
        end;
    end;
{
                Write Title
}
  OutTextXY(Round(GetMaxX/2-TextWidth(Title)/2),0,Title);
{
                Draw elevation marker: maximum and minimum
}
  If Draw_Elev=1 then
    begin
    If 0<=MaxScaleY then BarMax:=SYmax
                    else BarMax:=SYmax-Sign[2]*(-MaxScaleY);
    If 0<=MinScaleY then BarMin:=SYmax
                    else BarMin:=SYmax-Sign[2]*(-MinScaleY);
    BarLongX:=4;       { length of bar }
    SpaceBarX:=2;      { space between bars }
    Line(SXmax-Sign[1]*(2*BarLongX+SpaceBarX),BarMin,
         SXmax-Sign[1]*(1*BarLongX+SpaceBarX),BarMin);     {Bar for Min}
    Line(SXmax-Sign[1]*(1*BarLongX),BarMax,
         SXmax-Sign[1]*(0*BarLongX),BarMax);               {Bar for Max}
    Line(SXmax-Sign[1]*Round(1.5*BarLongX+SpaceBarX),SYmax,
         SXmax-Sign[1]*Round(1.5*BarLongX+SpaceBarX),
                        SYmax-Sign[2]*abs(MinScaleY));     {Min height}
    Line(SXmax-Sign[1]*Round(0.5*BarLongX),SYmax,
         SXmax-Sign[1]*Round(0.5*BarLongX),
                        SYmax-Sign[2]*abs(MaxScaleY));     {Max height}
    end;
{
                Empty keyboard buffer, wait for keystroke
}
  If Draw_Elev=1 then
    begin
```

```
      Close(F_xy0);
      Erase(F_xy0);
      end;
   wait;
   CloseGraph;    {Free heap memory used for graphics}
end;
{-------------------------------------------------------------------------}
end.
```

EDIT.PAS

```
{$N+}
Unit Edit;
Interface
Uses
   Crt,Graph,Declare,Nombre,WhatKey,Sonore,WaitKey;
procedure get_Paragraph(    Long,Top,Xleft,Xright: integer;
                            Color2: Byte;
                            Locked: Boolean;
                         var Paragraph: VectC_Screen;
                         var L: integer);
procedure get_action(    Long,Top,Xleft,Xright: integer;
                      var LookAction,LookPara,LookNumber: Boolean;
                      var action: string6;
                      var Paragraph: VectC_Screen;
                      var L: integer;
                      var numberS: String255);
{-------------------------------------------------------------------}
Implementation
Var
   EcranCGA: Array[1..4000] of byte Absolute $B800:$0000;
   EcranHERC: Array[1..4000] of byte Absolute $B000:$0000;
   GrDriver, GrMode: Integer;
{-------------------------------------------------------------------}
Procedure GetEcran(    i: integer;
                   var C: Byte);
{
   Input:
      GrDriver: (Byte) 1 for CGA
                       2     MCGA
                       3     EGA
                       4     EGA64
                       5     EGAmono
                       6     RESERVED
                       7     HercMono
                       8     ATT400
                       9     VGA
                       10    PC3270
      EcranCGA[1..4000]: sreen memory for (character,color) of CGA,EGA,VGA
      EcranHERC[1..4000]: sreen memory for (character,color) of HercMono
      i: screen position (1..4000) 1 is for first character 2 for its color
                                   3 is for second ...      4 for its color
   Output:
      C: character or color
}
begin
   Case GrDriver of
      1..6,8..10: C:=EcranCGA[i];
      7: C:=EcranHERC[i];
      end;
end;
```

```
{-------------------------------------------------------------------}
Procedure PutEcran(    i: integer;
                       C: Byte);
{
   Input:
      GrDriver: (Byte) 1 for CGA
                       2     MCGA
                       3     EGA
                       4     EGA64
                       5     EGAmono
                       6     RESERVED
                       7     HercMono
                       8     ATT400
                       9     VGA
                       10    PC3270
      EcranCGA[1..4000]: sreen memory for (character,color) of CGA,EGA,VGA
      EcranHERC[1..4000]: sreen memory for (character,color) of HercMono
      i: screen position (1..4000) 1 is for first character 2 for its color
                                   3 is for second ...      4 for its color
      C: character or color
}
begin
   Case GrDriver of
      1..6,8..10: EcranCGA[i]:=C;
      7: EcranHERC[i]:=C;
      end;
end;
{-------------------------------------------------------------------}
procedure get_Paragraph;
{
procedure get_Paragraph(    Long,Top,Xleft,Xright: integer;
                            Color2: Byte;
                            Locked: Boolean;
                        var Paragraph: VectC_Screen;
                        var L: integer);
}
{
       ¦ Input:
       +----> Long: maximum of characters for Paragraph
            ¦ Top: Top line of border for intermediate state
            ¦              (word or number)
            ¦ XLeft,Xright: Margins of border for intermediate state
            ¦ Color2: Characters
            ¦           0..15 are Black..White
            ¦           For background color add (Background color)*16
            ¦              0..7 are Black..LightGrey
            ¦           For Blink add (8*16) = 128
            ¦ Locked: =True if we can only look at a message
            ¦          It will set the border to the same color as text
            ¦         =False if we can edit
       ¦ Output:
       +----> Paragraph: Vector of Characters
```

```
          + L: Number of characters in Paragraph
       example:
          +(Xleft,Top)     (Xright,Top)
          +---+Ins-------------+
              ¦Jerome Daoust  ¦     L=13   Long=25
              ¦        +------+     Par=Jerome Daoust est le meil
              +---------+
}
var
   Par: array[1..2000] of Byte;
   Ecran_save: Array[1..4000] of byte;
   PlaceE: Array[1..2000] of integer;
   PlaceXY:Array[1..2000,1..2] of byte;
   Del_From,Del_To,i_Line,Xstart,Ystart,Ecran_Start,Ecran_end,
      Color,L_save,i,iL,X_before,Y_before,
      Help_L,Columns,Lines,count,HorizLine: Integer;
   Letter,Character: char;
   S1: String[1];
   Action: String6;
   Procedure_ok,On_Space,Ins,Found: boolean;
   Help,Insert_Mot:string[255];
   Help_Para: VectC_Screen;
   Color1,Color3: Byte;
   C,c1,c2: Byte;
begin
{
                  Detect Graphic card
}
   GrDriver:=Detect;
   DetectGraph(GrDriver,GrMode);
   Color1:=LightGreen;       {Borders + Help}
   Color3:=LightRed+8*16;    {Insert message}
   If Locked=True then Color1:=Color2;
   Xstart:=Xleft+1;
   Ystart:=Top+1;
   Columns:=Xright-Xleft-1;
{
          Check margins
}
   Procedure_ok:=True;
   If (Columns<1) or (Top<1) or (23<Top) then Procedure_ok:=False;
   If Procedure_ok then
      If not ((Xleft in [1..79]) and (Xright in [2..80])) then
         Procedure_ok:=False;
{
          Check Long
}
   If Procedure_ok then
      If Long<=0 then
         begin
         Procedure_ok:=False;
         Paragraph[1]:=' ';
```

```
        L:=0;
        end;
{
          Check if we can write (Lines+2) lines
}
   If Procedure_ok then
      begin
      Lines:=(Long div Columns);
      If (Long mod Columns)>0 then Lines:=Lines+1;
      If 25<Top+Lines+1 then procedure_ok:=False;
      end;
   If Procedure_ok then
      begin
{
            Save original paragraph and position
}
      if L>Long then L:=Long;
      if L<0 then L:=0;
      For i:=1 to L do Par[i]:=ord(Paragraph[i]);
      L_save:=L;
      X_before:=WhereX;
      Y_before:=WhereY;
{
             Initialize vectors
}
      For i:=1 to Long do
         begin
         PlaceXY[i,1]:=(i-1) MOD Columns  + Xstart;
         PlaceXY[i,2]:=(i-1) DIV Columns  + Ystart;
         PlaceE[i]:=((PlaceXY[i,1]*2)-1)+(PlaceXY[i,2]-1)*160;
         end;
      PlaceXY[Long+1,1]:=PlaceXY[Long,1]+1;
      PlaceXY[Long+1,2]:=PlaceXY[Long,2];
      PlaceE[Long+1]:=PlaceE[Long]+2;
      Ecran_start:=PlaceE[1]-160-2;
      Ecran_end:=PlaceE[Long]+160+2;
{
         Save screen and clear it
}
      For i:=Ecran_start to Ecran_end+1 do
         GetEcran(i,Ecran_save[i]);
      For i:=1 to Long do
         begin
         PutEcran(PlaceE[i],32);            (spaces)
         PutEcran(PlaceE[i]+1,Color1);      (We only need to define the color once)
         end;
{
      +------+
}
      PutEcran(Ecran_Start,ord('+'));
      PutEcran(Ecran_Start+1,Color1);
      If Long<Columns then Count:=Long
```

```
                else Count:=Columns;
      For i:=1 to count do
         begin
         PutEcran(PlaceE[i]-160,ord('-'));
         PutEcran(PlaceE[i]-160+1,Color1);
         end;
      PutEcran(PlaceE[count]-160+2,ord('+'));
      PutEcran(PlaceE[count]-160+2+1,Color1),
{
          Help message
}
      If Locked=False then
         begin
         Help:='<F1> help';
         If Length('Insert')+Length(Help)<=count then
            For i:=1 to Length(Help) do
               begin
               S1:=Copy(Help,i,1);
               PutEcran(PlaceE[Count-Length(Help)+i]-160,ord(S1[1]));
               PutEcran(PlaceE[Count-Length(Help)+i]-160+1,LightRed);
               end;
         end;
{
      ¦   ¦
      ¦ +-+
}
      For i_Line:=1 to Lines do
         begin
         PutEcran(PlaceE[1+Columns*(i_Line-1)]-2,ord('¦'));
         PutEcran(PlaceE[1+Columns*(i_Line-1)]-2+1,Color1);
         If (Long<Columns) then
            begin
            PutEcran(PlaceE[Long]+2,ord('¦'));
            PutEcran(PlaceE[Long]+2+1,Color1);
            end
         else
            begin
            If Long<Columns*i_Line then
               begin
               PutEcran(PlaceE[Long]+2,ord('+'));
               PutEcran(PlaceE[Long]+2+1,Color1);
               HorizLine:=0;
               For i:=1 to (Columns*i_Line-(Long+2)+1) do
                  begin
                  PutEcran(PlaceE[Long]+2+2*i,ord('-'));
                  PutEcran(PlaceE[Long]+2+2*i+1,Color1);
                  Inc(HorizLine);
                  end;
               PutEcran(PlaceE[Long]+4+2*HorizLine,ord('+'));
               PutEcran(PlaceE[Long]+4+2*HorizLine+1,Color1);
               end
            else
```

```
            begin
            PutEcran(PlaceE[Columns*i_Line]+2,ord('¦'));
            PutEcran(PlaceE[Columns*i_Line]+2+1,Color1);
            end;
         end;
      end;
{
   +------+
}
   PutEcran(PlaceE[(Lines-1)*Columns+1]+160-2,ord('+'));
   PutEcran(PlaceE[(Lines-1)*Columns+1]+160-2+1,Color1);
   For i:=(Lines-1)*Columns+1 to Long do
      begin
      PutEcran(PlaceE[i]+160,ord('-'));
      PutEcran(PlaceE[i]+160+1,Color1);
      end;
   PutEcran(Ecran_End,ord('+'));
   PutEcran(Ecran_End+1,Color1);
{
        Write paragraph
}
   For i:=1 to L do
      begin
      PutEcran(PlaceE[i],Par[i]);
      PutEcran(PlaceE[i]+1,Color2);
      end;
   gotoxy(PlaceXY[L+1,1],PlaceXY[L+1,2]);
{
            Get new Paragraph
}
   iL:=L+1;
   Ins:=False;
   repeat
      Get_Key(Character,Action);
{
                  Escape
}
      If Action='ESC' then
         begin
         For i:=L_save+1 to L do
            begin
            PutEcran(PlaceE[i],32);
            PutEcran(PlaceE[i]+1,Color1);
            end;
         L:=L_save;
         iL:=L+1;
         For i:=1 to L do
            begin
            PutEcran(PlaceE[i],Par[i]);
            PutEcran(PlaceE[i]+1,Color2);
            end;
         end;
```

```
{
                Home
}
        if Action='HOME' then
           iL:=1;
{
                End
}
        if Action='END' then
           iL:=L+1;
{
                Ins
}
        if Action='INS' then
           begin
           If Ins=True then
              begin
              Ins:=False;
              Insert_Mot:='------';
              Color:=Color1;
              end
           else
              begin
              Ins:=True;
              Insert_Mot:='Insert';
              Color:=Color3;
              end;
           Insert_mot:=Copy(Insert_mot,1,Long);
           For i:=1 to length(Insert_mot) do
              begin
              S1:=copy(Insert_mot,i,1);
              PutEcran(Ecran_start+i*2,ord(S1[1]));
              PutEcran(Ecran_start+i*2+1,Color);
              end;
           end;
{
                Delete
}
        if (Action='DEL') and (L-(iL-1)>0) then
           begin
           For i:=iL to L-1 do
              begin
              GetEcran(PlaceE[i+1],C);
              PutEcran(PlaceE[i],C);
              end;
           PutEcran(PlaceE[L],32);
           PutEcran(PlaceE[L]+1,Color1);
           L:=L-1;
           end;
{
                Up
}
```

```
        if (Action='UP') and (iL-1)=Columns) then
           iL:=iL-Columns;
{
                    Down
}
        if (Action='DOWN') and (L+1-iL)=Columns) then
           iL:=iL+Columns;
{
                    Left
}
        if (Action='LEFT') and (iL>1) then
           iL:=iL-1,
{
                    Right
}
        if (Action='RIGHT') and (iL<L+1) then
           iL:=iL+1;
{
                    Ctrl+Left
}
        if (Action='CLEFT') and (iL>1) then
           begin
           iL:=iL-1;
           Found:=False;
           While ((1<iL) and (Found=False)) do
              begin
              GetEcran(PlaceE[iL-1],c1);
              GetEcran(PlaceE[iL],c2);
              If (chr(c1)=' ')and(chr(c2)<>' ') then Found:=True
                                                else iL:=iL-1;
              end;
           end;
{
                    Ctrl+Right
}
        if (Action='CRIGHT') and (iL<L+1) then
           begin
           iL:=iL+1;
           Found:=False;
           While ((iL<L+1) and (Found=False)) do
              begin
              GetEcran(PlaceE[iL-1],c1);
              GetEcran(PlaceE[iL],c2);
              If (chr(c1)=' ')and(chr(c2)<>' ') then Found:=True
                                                else iL:=iL+1;
              end;
           end;
{
                       Delete a word
}
        if (Action='F4') and (0<L) then
           begin
```

```
            GetEcran(PlaceE[iL],C);
            if chr(C)=' ' then On_Space:=True
                         else On_Space:=False;
            Del_From:=iL;
            Del_To:=0;
{
               Erase current character
}
            If (1<=iL) and (iL<=L) then
               begin
               Del_To:=iL;
               Found:=False;
               end
            else
               Found:=True;
{
                  erase all current character wich are nor A..Z,a..z,c..u
}
            While (1<=iL) and (Del_To+1<=L) and (Found=False) do
               begin
               GetEcran(PlaceE[Del_To+1],C);
               Letter:=chr(C);
               If Letter in ['A'..'Z','a'..'z','c','a','a','e','e','e','e',
                             'i','i','o','u','u','u'] then
                  Del_To:=Del_To+1
               else
                  Found:=True;
               end;
{
                  erase character before if it is a space and BackStep
}
            GetEcran(PlaceE[Del_From-1],C);
            If (Del_From>1) and (chr(C)=' ') then Del_From:=Del_From-1;
{
                  erase this character if it is a space and iL=1
}
            GetEcran(PlaceE[Del_To+1],C);
            If (Del_From=1) and (chr(C)=' ') and (Del_To+1<=L) then
                  Del_To:=Del_To+1;
{
                    erase from Del_From to Del_To included
}
            If Del_From<=Del_To then
               begin
               For i:=Del_From to L-(Del_To-Del_From+1) do
                  begin
                  GetEcran(PlaceE[i+(Del_To-Del_From+1)],C);
                  PutEcran(PlaceE[i],C);
                  end;
               For i:=L-(Del_To-Del_From+1)+1 to L do
                  begin
                  PutEcran(PlaceE[i],32);
```

```
                PutEcran(PlaceE[i]+1,Color1);
                end;
             iL:=Del_From;
             L:=L-(Del_To-Del_From+1);
             GetEcran(PlaceE[iL],C);
             if (On_Space=False) and (chr(C)<>' ') and (iL<=L) then
                iL:=iL+1;
             end;
          end;
(
               Add or modify a character
)
       If Character<>Chr(0) then
          If Ins=false then                    (Not Insert mode)
             begin
             If iL<=Long then
                begin
                PutEcran(PlaceE[iL],ord(Character));
                If L<iL then
                   begin
                   L:=L+1;
                   PutEcran(PlaceE[iL]+1,Color2);
                   end;
                iL:=iL+1;
                end;
             end
          else                                 (Insert mode)
             begin
             If L<Long then
                begin
                For i:=L downto iL do
                   begin
                   GetEcran(PlaceE[i],C);
                   PutEcran(PlaceE[i+1],C);
                   end;
                PutEcran(PlaceE[iL],ord(Character));
                iL:=iL+1;
                L:=L+1;
                PutEcran(PlaceE[L]+1,Color2);
                end;
             end;
(
                Delete left character
)
       if (Action='BS') and (1<iL) then
          begin
          For i:=iL-1 to L-1 do
             begin
             GetEcran(PlaceE[i+1],C);
             PutEcran(PlaceE[i],C);
             end;
          PutEcran(PlaceE[L],32);
```

```
            PutEcran(PlaceE[L]+1,Color1);
            iL:=iL-1;
            L:=L-1;
            end;
{
                  Help Screen
}
         if (Action='F1') and (Locked=False) then
            begin
            Help_L:=0;
            Help:=
                 '<Esc> gives the original content'+
                 '<Ins> insert / overwrite        '+
                 '<Del> <F4> delete letter - word ';
            For i:=1 to Length(Help) do
               begin
               S1:=Copy(Help,i,1);
               Help_Para[Help_L+i]:=S1[1];
               end;
            Help_L:=Help_L+Length(Help);
            Help:=
                 '<BackStep> delete left character'+
                 '<Home> <End> goes to Home - End '+
                 '<Arrow> change line - letter    ';
            For i:=1 to Length(Help) do
               begin
               S1:=Copy(Help,i,1);
               Help_Para[Help_L+i]:=S1[1];
               end;
            Help_L:=Help_L+Length(Help);
            Help:=
                 '<Ctrl+Left,Right> change word   ';
            For i:=1 to Length(Help) do
               begin
               S1:=Copy(Help,i,1);
               Help_Para[Help_L+i]:=S1[1];
               end;
            Help_L:=Help_L+Length(Help);
            get_Paragraph(Help_L,4,22,55,LightRed+Blue*16,True,
                        Help_Para,Help_L);
            end;
         gotoxy(PlaceXY[iL,1],PlaceXY[iL,2]);
         until ((Locked=True) or (Action='ENTER') or (L>Long));   {wait for ENTER}
{
      transfer Ecran to paragraph
}
      For i:=1 to L do
         begin
         GetEcran(PlaceE[i],C);
         Paragraph[i]:=chr(C);
         end;
{
```

```
            Restore initial screen
}
      For i:=Ecran_start to Ecran_end+1 do
         PutEcran(i,Ecran_save[i]);
      gotoXY(X_before,Y_before);
{
               Empty keyboard buffer
}
      While keypressed do Letter:=ReadKey;
      end;
end;
{-------------------------------------------------------------------}
procedure get_action;
{
procedure get_action(    Long,Top,Xleft,Xright: integer;
                    var LookAction,LookPara,LookNumber: Boolean;
                    var action: string6;
                    var Paragraph: VectC_Screen;
                    var L: integer;
                    var numberS: String255);
}
{
       ¦ Input:
       +----+ Long: maximum of characters for Paragraph or number
            ¦ Top: Top line of border for intermediate state
            ¦              (word or number)
            ¦ XLeft,Xright: Margins of border for intermediate state
            ¦ LookAction: True if we accept an action
            ¦ LookPara: True if we accept a Paragraph
            + LookNumber: True if we accept a number (any kind)
       ¦ Output:
       +----+ cas: type of answer
            ¦ LookAction: True if an action was captured
            ¦ LookPara: True if a Paragraph was captured
            ¦ LookNumber: True if a number was captured
            ¦ Action: (UP,DOWN,RIGHT,LEFT,PGDN,PGUP,HOME,END,INS,DEL)
            ¦         (CHOME,CEND,CPGUP,CPGDN,CLEFT,CRIGHT)
            ¦         (ENTER,BS,ESC)
            ¦         (F1...F10, SF1..SF10, CF1...CF10, AF1...AF10)
            ¦ Paragraph: Vector of Characters
            ¦ L: Number of characters in Paragraph
            + NumberS: number in a string
}
var
   C: char;
   i,ErrorType,cas: Integer;
   Possible: array[1..3] of byte;
   Number: real;
begin
{
                  Avoid an Imposible solution
}
```

```
   If ((LookAction=False)and(LookPara=False)and(LookNumber=False)) then
      LookAction:=True;
   Possible[1]:=100;  (Cas will be in 0..3)
   Possible[2]:=100;
   Possible[3]:=100;
   If LookAction=True then Possible[1]:=1;
   If LookPara=True then Possible[2]:=2;
   If LookNumber=True then Possible[3]:=3;
   cas:=0;
   Action:='';
   Paragraph[1]:=' ';
   L:=0;
   numberS:='';
   repeat
      get_Key(C,Action);
{
                  Look if we have an Action
                  cas=1
}
      if (Possible[1]=1) and (Action<>'') then cas:=1;
{
                  get Paragraph or number
                  cas=2             cas=3
}
      If (C<>Chr(0)) and ((Possible[2]=2) or (Possible[3]=3)) then
         begin
         cas:=2;              (Paragraph by default)
         Paragraph[1]:=C;
         L:=1;
         get_Paragraph(Long,Top,Xleft,Xright,LightRed+Blue*16,
                     False,Paragraph,L);
         If Possible[3]=3 then
            begin
            NumberS:='';
            If L<=255 then
               For i:=1 to L do
                  NumberS:=NumberS+Paragraph[i];
            if '+'=COPY(NumberS,1,1) then    {so +X is considered as number X}
               begin
               val(COPY(NumberS,2,Length(NumberS)-1),number,ErrorType);
               if ErrorType=0 then Delete(NumberS,1,1);
               end;
{
                  if no error --> number, error --> Paragraph
}
            val(NumberS,number,ErrorType);
            if (ErrorType=0)and(NumberS<>'') then
               begin
               cas:=3;
               L:=L;   ( Keep Paragraph, even if a number is found )
               end;
            end;
```

```
        if (cas=2)and(L=0) then
          cas:=0;
        if cas=2 then numberS:='';
        end;
     until cas in [Possible[1],Possible[2],Possible[3]];
  Beep;
{
                Return case
}
  LookAction:=False;
  LookPara:=false;
  LookNumber:=False;
  Case cas of
     1: LookAction:=True;
     2: LookPara:=True;
     3: begin
        LookPara:=True;
        LookNumber:=True;
        end;
     end;
  If Possible[2]=100 then
     begin
     LookPara:=False;
     L:=0;
     end;
end;
{-------------------------------------------------------------------}
end.
```

ELEM1.PAS

```
{$N+}
Unit Elem1;
Interface
   Uses Declare,VectMat3,File_RW,Linear,Nne_Dim,WaitKey,Math,Displace;
Procedure K_elem1(    x1,y1,x2,y2,x3,y3: Rtype;
                      E,nu: Rtype;
                  var Ke: MatrixR_NneDxNneD);
Procedure SS1(var F_Displ: FileR;
              var Exyz: MatrixR_Nnex3;
              var Elem: VectI_Nne2;
              var Mat: MatrixR_Nmatx6;
              var FiberStrain,FiberStress,ElemStrain,ElemStress,
                    GlobalStrain,GlobalStress: VectR_NstressNne);
procedure Traction1(    Face: Byte;
                        Traction: Rtype;
                    var Exyz: MatrixR_Nnex3;  {var for speed}
                    var Tindex: VectI_Nne;
                    var Nt: Byte;
                    var Ft: MatrixR_Nnex3);
{-------------------------------------------------------------------}
Implementation
Procedure Get_A6(    x1,y1,x2,y2,x3,y3: Rtype;
                 var A6: MatrixR_6x6);
var
   i,j: IType;
   A,iA: MatrixR_3x3;
begin
{
                   Fill A matrix
}
   A[1,1]:=1;   A[1,2]:=x1;   A[1,3]:=y1;
   A[2,1]:=1;   A[2,2]:=x2;   A[2,3]:=y2;
   A[3,1]:=1;   A[3,2]:=x3;   A[3,3]:=y3;
{
                   iA = inverse of A
}
   Invert3(A,iA);
{
                   A6 = + iA  0 +
                        + 0  iA +
}
   For i:=1 to 3 do
      For j:=1 to 3 do
         begin
         A6[i,j]:=iA[i,j];
         A6[i+3,j+3]:=iA[i,j];
         A6[i+3,j]:=0;
         A6[i,j+3]:=0;
         end;
end;
```

```
{-------------------------------------------------------------------------}
Procedure K_elem1;
{
Procedure K_elem1(    x1,y1,x2,y2,x3,y3: Rtype;
                      E,nu: Rtype;
                  var Ke: MatrixR_NneDxNneD);
}
{
   Triangular 2-dimensional finite element, isotropic materials
   Input:
      x1..x3,y1 .y3: X and Y value of tringle's corners given in
                     a trigonometric rotation order
      N: Total number of nodes
      E: Young's modulus (Pa)
      nu: Poisson ratio
   Output:
      Ke: Element Stiffness matrix:  |Fx1|          |u1 |
                                     |Fy1|          |v1 |
                                     |...|  =  [Ke]. |...|
                                     |FxN|          |uN |
                                     |FyN|          |vN |
}
var
   Nne,N,i,j,k: IType;
   D,Multiplier: Rtype;
   X,Y: array[1..3] of Rtype;
   P1_6,P2_6,A6,BtXB: MatrixR_6x6;
begin
   Nne:=3;
   X[1]:=x1;
   X[2]:=x2;
   X[3]:=x3;
   Y[1]:=y1;
   Y[2]:=y2;
   Y[3]:=y3;
   Get_A6(X[1],Y[1],X[2],Y[2],X[3],Y[3],A6);
{
                  Get Multiplier
}
   D:=X[1]*(Y[2]-Y[3])+X[2]*(Y[3]-Y[1])+X[3]*(Y[1]-Y[2]);
   Multiplier:=abs(D)*E/2/(1-sqr(nu));
{
                  P1 = Multiplier x (Transpose of A6)
}
   For i:=1 to Dim2*Nne do
      For j:=1 to Dim2*Nne do
         P1_6[i,j]:=Multiplier*A6[j,i];
{
                  Get BtXB
}
   For i:=1 to Dim2*Nne do
      For j:=1 to Dim2*Nne do
```

```
        BtXB[i,j]:=0.0;
   BtXB[2,2]:=1;
   BtXB[6,2]:=nu;
   BtXB[3,3]:=(1-nu)/2;
   BtXB[5,3]:=(1-nu)/2;
   BtXB[3,5]:=(1-nu)/2;
   BtXB[5,5]:=(1-nu)/2;
   BtXB[2,6]:=nu;
   BtXB[6,6]:=1;
{
                Ke = Multiplier x (Transpose of A6) x BtXB x A6
}
   N:=6;
   for i:=1 to N do
      for j:=1 to N do
         begin
         P2_6[i,j]:=0;
         for k:=1 to N do
            P2_6[i,j]:=P2_6[i,j]+P1_6[i,k]*BtXB[k,j];
         end;
{
                Ke matrix is symmetric, so
                Get upper triangular of Ke then copy to lower (symmetrical)
}
   for i:=1 to N do
      for j:=1 to i do
         begin
         Ke[i,j]:=0;
         for k:=1 to N do
            Ke[i,j]:=Ke[i,j]+P2_6[i,k]*A6[k,j];
         end;
   for i:=1 to N do
      for j:=i+1 to N do
         Ke[i,j]:=Ke[j,i];
end;
{-------------------------------------------------------------}
Procedure SS1;
{
Procedure SS1(var F_Displ: FileR;
              var Exyz: MatrixR_Nnex3;
              var Elem: VectI_Nne2;
              var Mat: MatrixR_Nmatx6;
              var FiberStrain,FiberStress,ElemStrain,ElemStress,
                    GlobalStrain,GlobalStress: VectR_NstressNne);
}
{
   Input:
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      Exyz: Coordinates for the element
      Elem[1..Nne+2]: defines Material # (i=1),
                              Element Type (i=2),
```

```
                              nodes for element (i=3...)
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
   Output:
      FiberStrain[1..N_Stress(Dim).Nne]: Material direction e1 e2 e12
                                         for each node of element
      FiberStress[1..N_Stress(Dim).Nne]: Material direction s1 s2 s12
                                         for each node of element
      ElemStrain[1..N_Stress(Dim).Nne]: Element direction e1 e2 e12
                                         for each node of element
      ElemStress[1..N_Stress(Dim).Nne]: Element direction s1 s2 s12
                                         for each node of element
      GlobalStrain[1..N_Stress(Dim).Nne]: Global direction e1 e2 e12
                                          for each node of element
      GlobalStress[1..N_Stress(Dim).Nne]: Global direction s1 s2 s12
                                          for each node of element
)
var
   row,i,j,k,Etype,iNne: IType;
   E,nu: Rtype;
   B,P: array[1..3,1..6] of Rtype;
   strain,Stress: array[1..3] of Rtype;
   A: MatrixR_6x6;
   d: array[1..6] of Rtype;
   Displ: VectR_D;
begin
   Etype:=Elem[2];
   Get_A6(Exyz[1,1],Exyz[1,2],Exyz[2,1],Exyz[2,2],Exyz[3,1],Exyz[3,2],A);
{
                  d[1..6] are the u1 u2 u3 v1 v2 v3
                          of the nodes for the element
}
   Displ_R_Read(F_Displ,abs(Elem[2+1]),Dim2,Displ);
   d[1]:=Displ[1];
   d[4]:=Displ[2];
   Displ_R_Read(F_Displ,abs(Elem[2+2]),Dim2,Displ);
   d[2]:=Displ[1];
   d[5]:=Displ[2];
   Displ_R_Read(F_Displ,abs(Elem[2+3]),Dim2,Displ);
   d[3]:=Displ[1];
   d[6]:=Displ[2];
{
                  B is the derivate of [M]=[1 x y]
}
   For i:=1 to 3 do
      For j:=1 to 6 do
         B[i,j]:=0;
   B[1,2]:=1;
   B[3,3]:=1;
   B[3,5]:=1;
   B[2,6]:=1;
{
                  Calculate strains
```

```
                e1   b11 b12 b13 b14 b15 b16  a11 a12 a13 a14 a15 a16  d1
                e2 = b21 b22 b23 b24 b25 b26  a21 a22 a23 a24 a25 a26  d2
                e12  b31 b32 b33 b34 b35 b36  a31 a32 a33 a34 a35 a36  d3
                                              a41 a42 a43 a44 a45 a46  d4
                                              a51 a52 a53 a54 a55 a56  d5
                                              a61 a62 a63 a64 a65 a66  d6
}
  For i:=1 to 3 do
    For j:=1 to 6 do
      begin
      P[i,j]:=0;
      For k:=1 to 6 do
        P[i,j]:=P[i,j]+B[i,k]*A[k,j];
      end;
  For i:=1 to 3 do
    begin
    Strain[i]:=0;
    For j:=1 to 6 do
      Strain[i]:=Strain[i]+P[i,j]*d[j];
    end;
{
                Calculate stresses
                S1     E      + 1  nu 0      +   +e1 +
                S2 = ----     ¦ nu 1  0      ¦   ¦e2 ¦
                S12  1-nu^2   + 0  0 (1-nu)/2+   +e12+
}
  E:=Mat[Elem[1],1];
  nu:=Mat[Elem[1],5];
  Stress[1]:=Strain[1]+nu*Strain[2];
  Stress[2]:=nu*Strain[1]+Strain[2];
  Stress[3]:=(1-nu)/2*Strain[3];
  For i:=1 to 3 do
    Stress[i]:=E/(1-sqr(nu))*Stress[i];
{
                Place strain and stress at appropriate node
                On-axis, Element and Global strain-stress are the same
                  because we have an isotropic material
}
  For iNne:=1 to 3 do
    For row:=1 to 3 do
      begin
      j:=Lin(iNne,row,3);
      FiberStrain[j]:=Strain[row];
      FiberStress[j]:=Stress[row];
      ElemStrain[j]:=Strain[row];
      ElemStress[j]:=Stress[row];
      GlobalStrain[j]:=Strain[row];
      GlobalStress[j]:=Stress[row];
      end;
end;
{-------------------------------------------------------------}
procedure Traction1;
```

```
{
procedure Traction1(    Face: Byte;
                        Traction: Rtype;
                    var Exyz: MatrixR_Nnex3;      var for speed
                    var Tindex: VectI_Nne;
                    var Nt: Byte;
                    var Ft: MatrixR_Nnex3);
}
{
   Input:
      Face: which face where a traction is applied
            1: line 1-2
            2: line 2-3
            3: line 3-1
      Traction: Traction value on Face (inverse of pressure)
      Exyz: coordinates of nodes in element
   Output:
      Tindex: index of nodes in element in contact with the face
      Nt: number of nodes in contact with the face
      Ft[iNne,Direction]: Forces on element nodes due to traction on face
         Add these to node forces
}
var
   Nne,iNne,iD: integer;
   x1,x2,y1,y2: Rtype;
   L,l1,m1,l2,m2,alpha,beta: Rtype;
begin
{
                  Set Tindex (here order is used for normal direction)
}
   Nne:=Get_Nne(1);
   For iNne:=1 to Nne do
      begin
      Tindex[iNne]:=0;
      For iD:=1 to Dim2 do
         Ft[iNne,iD]:=0;
      end;
   Nt:=2;
   case Face of
      1: begin
         Tindex[1]:=1;
         Tindex[2]:=2;
         end;
      2: begin
         Tindex[1]:=2;
         Tindex[2]:=3;
         end;
      3: begin
         Tindex[1]:=3;
         Tindex[2]:=1;
         end;
      end;
```

```
{
                Get segment orientation
}
   x1:=Exyz[Tindex[1],1];
   x2:=Exyz[Tindex[2],1];
   y1:=Exyz[Tindex[1],2];
   y2:=Exyz[Tindex[2],2];
   L:=sqrt(sqr(x2-x1)+sqr(y2-y1));
   if L=0 then
      begin
      writeln('   Segment has no length');
      wait;
      halt;
      end;
   l1:=(x2-x1)/L;   {cos alpha}  {normalize}
   m1:=(y2-y1)/L;   {sin alpha}
   alpha:=ArcTan4(l1,m1);
{
                Orientation of normal is 90 degrees less
                than orientation of segment
}
   Beta:=alpha-Pi/2;
{
                Set Traction forces
}
   Ft[Tindex[1],1]:=Traction/2*cos(Beta)*L;
   Ft[Tindex[1],2]:=Traction/2*sin(Beta)*L;
   Ft[Tindex[2],1]:=Ft[Tindex[1],1];
   Ft[Tindex[2],2]:=Ft[Tindex[1],2];
end;
{-------------------------------------------------------------}
end.
```

ELEM2.PAS

```
{$N+}
Unit Elem2;
Interface
   Uses Declare,VectMat3,Crt,File_RW,Linear,Nne_Dim,Displace,Sonore,WaitKey;
Procedure Get_enb(var enb: MatrixR_NgqxD);                    {For ELEM in PREP}
Procedure Get_BF(    e,n,b: Rtype;                            {For ELEM in PREP}
                 var Exyz: MatrixR_Nnex3;  { var for speed }
                 var BF: MatrixR_6xNneD;
                 var J: MatrixR_3x3);
Procedure K_elem2(    iMat: integer;
                  var Exyz: MatrixR_Nnex3;    {var for speed}
                  var Mat: MatrixR_Nmatx6;    {var for speed}
                      Angle: Rtype;
                  var Ke: MatrixR_NneDxNneD);
Procedure SS2(var F_Displ: FileR;
              var Exyz: MatrixR_Nnex3;   { var for speed }
              var Elem: VectI_Nne2;      { var for speed }
              var Mat: MatrixR_Nmatx6;   { var for speed }
                  Angle: Rtype;
              var FiberStrain,FiberStress,ElemStrain,ElemStress,
                    GlobalStrain,GlobalStress: VectR_NstressNne);
procedure Traction2(    Face: Byte;
                        Traction: Rtype;
                    var Exyz: MatrixR_Nnex3;  {var for speed}
                    var Tindex: VectI_Nne;
                    var Nt: Byte;
                    var Ft: MatrixR_Nnex3);
{-------------------------------------------------------------------}
Implementation
Procedure Get_enb;
{
Procedure Get_enb(var enb: MatrixR_NgqxD);
}
{
   Output:
      enb[iGQ,iD]: local coordinates enb[iGQ,1], enb[iGQ,2], enb[iGQ,3]
                   for gauss point number iGQ
}
var
   a: Rtype;
begin
   a:=sqrt(3/5);  {gauss-quadrature locations: -a, 0, +a}
   enb[ 1,1]:=-a;   enb[ 1,2]:=-a;   enb[ 1,3]:=-a;
   enb[ 2,1]:=-a;   enb[ 2,2]:=-a;   enb[ 2,3]:= 0.0;
   enb[ 3,1]:=-a;   enb[ 3,2]:=-a;   enb[ 3,3]:= a;
   enb[ 4,1]:=-a;   enb[ 4,2]:= 0.0; enb[ 4,3]:=-a;
   enb[ 5,1]:=-a;   enb[ 5,2]:= 0.0; enb[ 5,3]:= 0.0;
   enb[ 6,1]:=-a;   enb[ 6,2]:= 0.0; enb[ 6,3]:= a;
   enb[ 7,1]:=-a;   enb[ 7,2]:= a;   enb[ 7,3]:=-a;
   enb[ 8,1]:=-a;   enb[ 8,2]:= a;   enb[ 8,3]:= 0.0;
```

```
   enb[ 9,1]:=-a;   enb[ 9,2]:= a;   enb[ 9,3]:= a;
   enb[10,1]:= 0.0; enb[10,2]:=-a;   enb[10,3]:=-a;
   enb[11,1]:= 0.0; enb[11,2]:=-a;   enb[11,3]:= 0.0;
   enb[12,1]:= 0.0; enb[12,2]:=-a;   enb[12,3]:= a;
   enb[13,1]:= 0.0; enb[13,2]:= 0.0; enb[13,3]:=-a;
   enb[14,1]:= 0.0; enb[14,2]:= 0.0; enb[14,3]:= 0.0;
   enb[15,1]:= 0.0; enb[15,2]:= 0.0; enb[15,3]:= a;
   enb[16,1]:= 0.0; enb[16,2]:= a;   enb[16,3]:=-a;
   enb[17,1]:= 0.0; enb[17,2]:= a;   enb[17,3]:= 0.0;
   enb[18,1]:= 0.0; enb[18,2]:= a;   enb[18,3]:= a;
   enb[19,1]:= a;   enb[19,2]:=-a;   enb[19,3]:=-a;
   enb[20,1]:= a;   enb[20,2]:=-a;   enb[20,3]:= 0.0;
   enb[21,1]:= a;   enb[21,2]:=-a;   enb[21,3]:= a;
   enb[22,1]:= a;   enb[22,2]:= 0.0; enb[22,3]:=-a;
   enb[23,1]:= a;   enb[23,2]:= 0.0; enb[23,3]:= 0.0;
   enb[24,1]:= a;   enb[24,2]:= 0.0; enb[24,3]:= a;
   enb[25,1]:= a;   enb[25,2]:= a;   enb[25,3]:=-a;
   enb[26,1]:= a;   enb[26,2]:= a;   enb[26,3]:= 0.0;
   enb[27,1]:= a;   enb[27,2]:= a;   enb[27,3]:= a;
end;
{-------------------------------------------------------------------}
Procedure Get_Multiplier(var Multiplier: VectR_Ngq);
{
   Output:
      Gauss-Quadrature multplier for each gauss points in GET_ENB()
      procedure
}
var
   a3,a2,a1,a0: Rtype;
begin
   a0:=8/9*8/9*8/9;     {sqrt(3/5) not used   }
   a1:=5/9*8/9*8/9;     {          used 1 time}
   a2:=5/9*5/9*8/9;     {               2     }
   a3:=5/9*5/9*5/9;     {               3     }
   Multiplier[1]:=a3;
   Multiplier[2]:=a2;
   Multiplier[3]:=a3;
   Multiplier[4]:=a2;
   Multiplier[5]:=a1;
   Multiplier[6]:=a2;
   Multiplier[7]:=a3;
   Multiplier[8]:=a2;
   Multiplier[9]:=a3;
   Multiplier[10]:=a2;
   Multiplier[11]:=a1;
   Multiplier[12]:=a2;
   Multiplier[13]:=a1;
   Multiplier[14]:=a0;
   Multiplier[15]:=a1;
   Multiplier[16]:=a2;
   Multiplier[17]:=a1;
   Multiplier[18]:=a2;
```

```
   Multiplier[19]:=a3;
   Multiplier[20]:=a2;
   Multiplier[21]:=a3;
   Multiplier[22]:=a2;
   Multiplier[23]:=a1;
   Multiplier[24]:=a2;
   Multiplier[25]:=a3;
   Multiplier[26]:=a2;
   Multiplier[27]:=a3;
end;
{-------------------------------------------------------------------}
Procedure Get_N(    e,n,b: Rtype;
                 var Ne: VectR_Nne);
{
   Input:
      e,n,b: Xi, Eta, Zeta: local coordinates in the element
   Output:
      Ne: Shape function for 20 node isoparametric element
}
var
   me,pe,mn,pn,mb,pb,mee,mnn,mbb: Rtype;
begin
   me:=1-e;
   pe:=1+e;
   mn:=1-n;
   pn:=1+n;
   mb:=1-b;
   pb:=1+b;
   mee:=0.25*(1-sqr(e));
   mnn:=0.25*(1-sqr(n));
   mbb:=0.25*(1-sqr(b));
{
                  Corner Nodes
}
   Ne[1]:= 0.125*me*mn*mb*(-e-n-b-2);
   Ne[3]:= 0.125*pe*mn*mb*( e-n-b-2);
   Ne[5]:= 0.125*pe*pn*mb*( e+n-b-2);
   Ne[7]:= 0.125*me*pn*mb*(-e+n-b-2);
   Ne[13]:=0.125*me*mn*pb*(-e-n+b-2);
   Ne[15]:=0.125*pe*mn*pb*( e-n+b-2);
   Ne[17]:=0.125*pe*pn*pb*( e+n+b-2);
   Ne[19]:=0.125*me*pn*pb*(-e+n+b-2);
{
                  Mid-Edge Nodes (e=0)
}
   Ne[2]:= mee*mn*mb;
   Ne[6]:= mee*pn*mb;
   Ne[14]:=mee*mn*pb;
   Ne[18]:=mee*pn*pb;
{
                  Mid-Edge Nodes (n=0)
}
```

```
   Ne[4]:= mnn*pe*mb;
   Ne[8]:= mnn*me*mb;
   Ne[16]:=mnn*pe*pb;
   Ne[20]:=mnn*me*pb;
{
                Mid-Edge Nodes (b=0)
}
   Ne[9]:= mbb*me*mn;
   Ne[10]:=mbb*pe*mn;
   Ne[11]:=mbb*pe*pn;
   Ne[12]:=mbb*me*pn;
end;
{-------------------------------------------------------------------}
Procedure Get_dNlocal(    e,n,b: Rtype;
                      var dNl: MatrixR_3xNne);
{
   Input:
     e,n,b: Xi, Eta, Zeta: local coordinates in the element
   Output:
     dNl: Derivate of Shape function for 20 node isoparametric element
          dNl[1,1..20]: derivate with Xi
          dNl[2,1..20]: derivate with Eta
          dNl[3,1..20]: derivate with Zeta
}
var
   me,pe,mn,pn,mb,pb: Rtype;
   mee,mnn,mbb,mee4,mnn4,mbb4: Rtype;
   me8,mn8,mb8,pe8,pn8,pb8: Rtype;
   me4,mn4,mb4,pe4,pn4,pb4: Rtype;
   e2,n2,b2: Rtype;
begin
   me:=1-e;
   pe:=1+e;
   mn:=1-n;
   pn:=1+n;
   mb:=1-b;
   pb:=1+b;
   mee:=1.0-sqr(e);
   mnn:=1.0-sqr(n);
   mbb:=1.0-sqr(b);
   mee4:=mee/4.0;
   mnn4:=mnn/4.0;
   mbb4:=mbb/4.0;
   me8:=me/8.0;
   mn8:=mn/8.0;
   mb8:=mb/8.0;
   pe8:=pe/8.0;
   pn8:=pn/8.0;
   pb8:=pb/8.0;
   me4:=me/4.0;
   mn4:=mn/4.0;
   mb4:=mb/4.0;
```

```
   pe4:=pe/4.0;
   pn4:=pn/4.0;
   pb4:=pb/4.0;
   e2:=2.0*e;
   n2:=2.0*n;
   b2:=2.0*b;
(
                Derivate with respect to Xi
)
   dNl[1,1] := mn8*mb*(e2+n+b+1);
   dNl[1,2] :=-mn4*mb*e2;
   dNl[1,3] := mn8*mb*(e2-n-b-1);
   dNl[1,4] := mnn4*mb;
   dNl[1,5] := pn8*mb*(e2+n-b-1);
   dNl[1,6] :=-pn4*mb*e2;
   dNl[1,7] := pn8*mb*(e2-n+b+1);
   dNl[1,8] :=   -dNl[1,4];
   dNl[1,9] :=-mn4*mbb;
   dNl[1,10]:=   -dNl[1,9];
   dNl[1,11]:= pn4*mbb;
   dNl[1,12]:=   -dNl[1,11];
   dNl[1,13]:= mn8*pb*(e2+n-b+1);
   dNl[1,14]:=-mn4*pb*e2;
   dNl[1,15]:= mn8*pb*(e2-n+b-1);
   dNl[1,16]:= mnn4*pb;
   dNl[1,17]:= pn8*pb*(e2+n+b-1);
   dNl[1,18]:=-pn4*pb*e2;
   dNl[1,19]:= pn8*pb*(e2-n-b+1);
   dNl[1,20]:=   -dNl[1,16];
(
                Derivate with respect to Eta
)
   dNl[2,1] := me8*mb*(n2+e+b+1);
   dNl[2,2] :=-mee4*mb;
   dNl[2,3] := pe8*mb*(n2-e+b+1);
   dNl[2,4] :=-pe4*mb*n2;
   dNl[2,5] := pe8*mb*(n2+e-b-1);
   dNl[2,6] :=   -dNl[2,2];
   dNl[2,7] := me8*mb*(n2-e-b-1);
   dNl[2,8] :=-me4*mb*n2;
   dNl[2,9] :=-me4*mbb;
   dNl[2,10]:=-pe4*mbb;
   dNl[2,11]:=   -dNl[2,10];
   dNl[2,12]:=   -dNl[2,9];
   dNl[2,13]:= me8*pb*(n2+e-b+1);
   dNl[2,14]:=-mee4*pb;
   dNl[2,15]:= pe8*pb*(n2-e-b+1);
   dNl[2,16]:=-pe4*pb*n2;
   dNl[2,17]:= pe8*pb*(n2+e+b-1);
   dNl[2,18]:=   -dNl[2,14];
   dNl[2,19]:= me8*pb*(n2-e+b-1);
   dNl[2,20]:=-me4*pb*n2;
```

```
{
                Derivate with respect to Zeta
}
   dNl[3,1] := me8*mn*(b2+e+n+1);
   dNl[3,2] :=-mee4*mn;
   dNl[3,3] := pe8*mn*(b2-e+n+1);
   dNl[3,4] :=-pe4*mnn;
   dNl[3,5] := pe8*pn*(b2-e-n+1);
   dNl[3,6] :=-mee4*pn;
   dNl[3,7] := me8*pn*(b2+e-n+1);
   dNl[3,8] :=-me4*mnn;
   dNl[3,9] :=-me4*mn*b2;
   dNl[3,10]:=-pe4*mn*b2;
   dNl[3,11]:=-pe4*pn*b2;
   dNl[3,12]:=-me4*pn*b2;
   dNl[3,13]:= me8*mn*(b2-e-n-1);
   dNl[3,14]:=   -dNl[3,2];
   dNl[3,15]:= pe8*mn*(b2+e-n-1);
   dNl[3,16]:=   -dNl[3,4];
   dNl[3,17]:= pe8*pn*(b2+e+n-1);
   dNl[3,18]:=   -dNl[3,6];
   dNl[3,19]:= me8*pn*(b2-e+n-1);
   dNl[3,20]:=   -dNl[3,8];
end;
{-------------------------------------------------------------------}
Procedure Get_J(    e,n,b: Rtype;
                var dNl: MatrixR_3xNne;  { var for speed }
                var Exyz: MatrixR_Nnex3; { var for speed }
                var J: MatrixR_3x3);
{
   Input:
      e,n,b: Xi, Eta, Zeta: local coordinates in the element
      dNl: Derivate of Shape function for 20 node isoparametric element
           dNl[1,1..20]: derivate with Xi
           dNl[2,1..20]: derivate with Eta
           dNl[3,1..20]: derivate with Zeta
      Exyz: Coordinates for the element
   Output:
      J: |dN[1..20]/dx|        |dN[1..20]/dXi  |
         |dN[1..20]/dy| =  [J] |dN[1..20]/dEta |
         |dN[1..20]/dy|        |dN[1..20]/dZeta|
         [J] is a 3x3 matrix
         [J] = | dx/de dy/de dz/de |
               | dx/dn dy/dn dz/dn |
               | dx/db dy/db dz/db |
}
var
   row,col: Byte;
begin
   For row:=1 to Dim3 do
      For col:=1 to Dim3 do
         begin
```

```
            J[row,col]:=dNl[row,1]*Exyz[1,col]
                       +dNl[row,2]*Exyz[2,col]
                       +dNl[row,3]*Exyz[3,col]
                       +dNl[row,4]*Exyz[4,col]
                       +dNl[row,5]*Exyz[5,col];
            J[row,col]:=J[row,col]
                       +dNl[row,6]*Exyz[6,col]
                       +dNl[row,7]*Exyz[7,col]
                       +dNl[row,8]*Exyz[8,col]
                       +dNl[row,9]*Exyz[9,col]
                       +dNl[row,10]*Exyz[10,col];
            J[row,col]:=J[row,col]
                       +dNl[row,11]*Exyz[11,col]
                       +dNl[row,12]*Exyz[12,col]
                       +dNl[row,13]*Exyz[13,col]
                       +dNl[row,14]*Exyz[14,col]
                       +dNl[row,15]*Exyz[15,col];
            J[row,col]:=J[row,col]
                       +dNl[row,16]*Exyz[16,col]
                       +dNl[row,17]*Exyz[17,col]
                       +dNl[row,18]*Exyz[18,col]
                       +dNl[row,19]*Exyz[19,col]
                       +dNl[row,20]*Exyz[20,col];
            end;
end;
{-------------------------------------------------------------------}
Procedure Get_dNglobal(    e,n,b: Rtype;
                       var dNl: MatrixR_3xNne;  { var for speed }
                       var Exyz: MatrixR_Nnex3; { var for speed }
                       var J: MatrixR_3x3;      { var for speed }
                       var dNg: MatrixR_3xNne);
{
   Input:
     dNl: Derivate of Shape function for 20 node isoparametric element
          dNl[1,1..20]: derivate with Xi
          dNl[2,1..20]: derivate with Eta
          dNl[3,1..20]: derivate with Zeta
     Exyz: Coordinates for the element
     J: |dN[1..20]/dx|                |dN[1..20]/dXi  |
        |dN[1..20]/dy| =  [J]inverse  |dN[1..20]/dEta |
        |dN[1..20]/dy|                |dN[1..20]/dZeta|
        [J] is a 3x3 matrix
   Output:
     dNg: Derivate of Shape function for 20 node isoparametric element
          dNl[1,1..20]: derivate with X
          dNl[2,1..20]: derivate with Y
          dNl[3,1..20]: derivate with Z
}
var
   row,col,Nne: Byte;
   iJ: MatrixR_3x3;
begin
```

```
   invert3(J,iJ);
   Nne:=20;
   For row:=1 to Dim3 do
      For col:=1 to Nne do
         dNg[row,col]:=iJ[row,1]*dNl[1,col]
                      +iJ[row,2]*dNl[2,col]
                      +iJ[row,3]*dNl[3,col];
end;
{-------------------------------------------------------------------}
Procedure Get_V(var J: MatrixR_3x3;   { var for speed }
                var V: MatrixR_3x3);
{
   Input:
      J: |dN[1..20]/dx|         |dN[1..20]/dXi  |
         |dN[1..20]/dy| =  [J] |dN[1..20]/dEta |
         |dN[1..20]/dy|         |dN[1..20]/dZeta|
         [J] = | dx/de dy/de dz/de |
               | dx/dn dy/dn dz/dn |
               | dx/db dy/db dz/db |
   Output:
      V: V[i,j] is the directive cosine of local axis i with global axis j
         |l1 m1 n1|
         |l2 m2 n2|
         |l3 m3 n3|
}
var
   Nne,iNnc,row,col: Byte;
   Long: Rtype;
begin
   Nne:=20;
{
                  [J] -> first two rows of [V]
}
   For row:=1 to 2 do
      For col:=1 to Dim3 do
         V[row,col]:=J[row,col];
{
                  Get V3 vector
                  V3 = V1 x V2
}
   VectorP3(V[1,1],V[1,2],V[1,3],V[2,1],V[2,2],V[2,3],
            V[3,1],V[3,2],V[3,3]);
{
                  Get V2 vector
                  V2 = V3 x V1   (V3 First!)
}
   VectorP3(V[3,1],V[3,2],V[3,3],V[1,1],V[1,2],V[1,3],
            V[2,1],V[2,2],V[2,3]);
{
                  Normalize Vectors
}
   For row:=1 to 3 do
```

```
      NormV3(V[row,1],V[row,2],V[row,3]);
end;
{-------------------------------------------------------------------}
Procedure Get_BF;
{
Procedure Get_BF(    e,n,b: Rtype;
                 var Exyz: MatrixR_Nnex3;   var for speed
                 var BF: MatrixR_6xNneD;
                 var J: MatrixR_3x3);
}
{
   Input:
      e,n,b: Xi, Eta, Zeta: local coordinates in the element
      Exyz: Coordinates for the element
   Output:
      BF: [element tensorial strain] = [BF] x [u1..uN v1..vN w1..wN]element
      J: |dN[1..20]/dx|        |dN[1..20]/dXi  |
         |dN[1..20]/dy| =  [J] |dN[1..20]/dEta |
         |dN[1..20]/dy|        |dN[1..20]/dZeta|
         [J] is a 3x3 matrix
}
var
   Nne,iD,row,col: integer;
   Vrow,dNl,dNg: MatrixR_3xNne;
   V: MatrixR_3x3;
   BFindex: array[1..3,1..20] of integer;
begin
{
                  Calculate dNl: local derivate of shape function
                           J: [dNg] = [J] [dNl]
                           dNg. global derivate of shape function
                           V: |l1 m1 n1|  direction of local coordinate
                              |l2 m2 n2|  system in global coord. syst.
                              |l3 m3 n3|
}
   Get_dNlocal(e,n,b,dNl);
   Get_J(e,n,b,dNl,Exyz,J);
   Get_dNglobal(e,n,b,dNl,Exyz,J,dNg);
   Get_V(J,V);
{
                  Get Vrow
}
   Nne:=20;
   For Row:=1 to Dim3 do
      For col:=1 to Nne do
         Vrow[row,col]:=V[row,1]*dNg[1,col]
                       +V[row,2]*dNg[2,col]
                       +V[row,3]*dNg[3,col];
{
                  Fill [BF] matrix
}
   For col:=1 to Nne do
```

```
      For iD:=1 to Dim3 do
         BFindex[iD,col]:=(iD-1)*Nne+col;
   For col:=1 to Nne do
      begin
      For row:=1 to Dim3 do
         For iD:=1 to Dim3 do
            BF[row,BFindex[iD,col]]:=Vrow[row,col]*V[row,iD];
      For iD:=1 to Dim3 do
         begin
         BF[4,BFindex[iD,col]]:=0.5*(Vrow[2,col]*V[1,iD]+Vrow[1,col]*V[2,iD]);
         BF[5,BFindex[iD,col]]:=0.5*(Vrow[3,col]*V[1,iD]+Vrow[1,col]*V[3,iD]);
         BF[6,BFindex[iD,col]]:=0.5*(Vrow[3,col]*V[2,iD]+Vrow[2,col]*V[3,iD]);
         end;
      end;
end;
{-------------------------------------------------------------------}
Procedure Get_C(    iMat: integer;
                var Mat: MatrixR_Nmatx6; { var for speed }
                var C: MatrixR_6x6);
{
   Input:
      iMat: material number
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
   Output:
      C[1..6,1..6]: On-Axis stiffness matrix
}
Var
   Delta,El,Et,Glt,Gtt,Nlt,Ntt: Rtype;
   E1,E2,E3,G12,G13,G23,n12,n13,n23,n21,n31,n32: Rtype;
begin
{
                  Get On-axis stiffness matrix
}
   El:=Mat[iMat,1];
   Et:=Mat[iMat,2];
   Glt:=Mat[iMat,3];
   Gtt:=Mat[iMat,4];
   Nlt:=Mat[iMat,5];
   Ntt:=Mat[iMat,6];
   E1:=El;
   E2:=Et;
   E3:=E2;
   G12:=Glt;
   G13:=G12;
   G23:=Gtt;
   n12:=Nlt;
   n13:=n12;
   n23:=Ntt;
   n21:=n12/E1*E2;
   n31:=n13/E1*E3;
   n32:=n23/E2*E3;
   Delta:=(1-n12*n21-n23*n32-n31*n13-2*n21*n32*n13)/(E1*E2*E3);
```

```
{
                [C] is symmetric so get upper traingular then copy
}
   C[1,1]:=(1-n23*n32)/(E2*E3*Delta);
   C[1,2]:=(n12+n32*n13)/(E1*E3*Delta);
   C[1,3]:=C[1,2];
   C[1,4]:=0.0;
   C[1,5]:=0.0;
   C[1,6]:=0.0;
   C[2,2]:=(1-n13*n31)/(E1*E3*Delta);
   C[2,3]:=(n23+n21*n13)/(E1*E2*Delta);
   C[2,4]:=0.0;
   C[2,5]:=0.0;
   C[2,6]:=0.0;
   C[3,3]:=C[2,2];
   C[3,4]:=0.0;
   C[3,5]:=0.0;
   C[3,6]:=0.0;
   C[4,4]:=2*G12;
   C[4,5]:=0.0;
   C[4,6]:=0.0;
   C[5,5]:=C[4,4];
   C[5,6]:=0.0;
   C[6,6]:=2*G23;
   C[2,1]:=C[1,2];
   C[3,1]:=C[1,3];
   C[3,2]:=C[2,3];
   C[4,1]:=C[1,4];
   C[4,2]:=C[2,4];
   C[4,3]:=C[3,4];
   C[5,1]:=C[1,5];
   C[5,2]:=C[2,5];
   C[5,3]:=C[3,5];
   C[5,4]:=C[4,5];
   C[6,1]:=C[1,6];
   C[6,2]:=C[2,6];
   C[6,3]:=C[3,6];
   C[6,4]:=C[4,6];
   C[6,5]:=C[5,6];
end;
{-------------------------------------------------------------------}
Procedure Get_TR(    Angle: Rtype;
                 var TR: MatrixR_6x6);
var
   m,n,mm,nn,mn: Rtype;
begin
{
                  Get Transformation matrix (not symmetric)
      [Sx',Sy',Sz',Sx'y',Sx'z',Sy'z']trans = [TR].[S1,S2,S3,S12,S13,S23]trans.
      note: [TR]^-1 = [TR]transpose
      [TR] = ¦ mm  nn  0  2mn   0  0 ¦
             ¦ nn  mm  0 -2mn   0  0 ¦
```

```
            |  0   0  1   0    0  0 |
            | -mn  mn  0 mm-nn  0  0 |
            |  0   0  0   0    m  n |
            |  0   0  0   0   -n  m |
      note: [TR]^-1 = [TR] transposed
}
   m:=cos(Angle);
   n:=sin(Angle);
   mm:=m*m;
   nn:=n*n;
   mn:=m*n;
   TR[1,1]:=mm;
   TR[1,2]:=nn;
   TR[1,3]:=0.0;
   TR[1,4]:=2*mn;
   TR[1,5]:=0.0;
   TR[1,6]:=0.0;
   TR[2,1]:=nn;
   TR[2,2]:=mm;
   TR[2,3]:=0.0;
   TR[2,4]:=-2*mn;
   TR[2,5]:=0.0;
   TR[2,6]:=0.0;
   TR[3,1]:=0.0;
   TR[3,2]:=0.0;
   TR[3,3]:=1.0;
   TR[3,4]:=0.0;
   TR[3,5]:=0.0;
   TR[3,6]:=0.0;
   TR[4,1]:=-mn;
   TR[4,2]:= mn;
   TR[4,3]:=0.0;
   TR[4,4]:=mm-nn;
   TR[4,5]:=0.0;
   TR[4,6]:=0.0;
   TR[5,1]:=0.0;
   TR[5,2]:=0.0;
   TR[5,3]:=0.0;
   TR[5,4]:=0.0;
   TR[5,5]:=m;
   TR[5,6]:=n;
   TR[6,1]:=0.0;
   TR[6,2]:=0.0;
   TR[6,3]:=0.0;
   TR[6,4]:=0.0;
   TR[6,5]:=-n;
   TR[6,6]:= m;
end;
{-------------------------------------------------------------------}
Procedure Get_EI(    iMat: integer;
                 var Mat: MatrixR_Nmatx6; { var for speed }
                     Angle: Rtype;
```

```
                var EI: MatrixR_6x6);
{
   Input:
      iMat: material number
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      Angle: angle for the material in the element
   Output:
      EI: Stiffness matrix in element coordinate system
          [Tensorial Stress] = [EI] [Tensorial Strain]
}
var
   row,col: integer;
   P,TR,C: MatrixR_6x6;
begin
{
                  Get Transformation matrix (not symmetric)
}
   Get_TR(Angle,TR);
{
                  [C]: On-axis stiffness matrix
}
   Get_C(iMat,Mat,C);
{
                  [EI] = [TR] x [C] x [TRt]
}
   for row:=1 to 6 do
      for col:=1 to 6 do
         P[row,col]:=TR[row,1]*C[1,col]
                    +TR[row,2]*C[2,col]
                    +TR[row,3]*C[3,col]
                    +TR[row,4]*C[4,col]
                    +TR[row,5]*C[5,col]
                    +TR[row,6]*C[6,col];
{
                  [EI] is symmetric so get lower triangular part and then copy
}
   for row:=1 to 6 do
      for col:=1 to row do
         EI[row,col]:=P[row,1]*TR[col,1]   ( <-- transpose of [TR] )
                     +P[row,2]*TR[col,2]
                     +P[row,3]*TR[col,3]
                     +P[row,4]*TR[col,4]
                     +P[row,5]*TR[col,5]
                     +P[row,6]*TR[col,6];
   for row:=1 to 6 do
      for col:=row+1 to 6 do
         EI[row,col]:=EI[col,row];
end;
{-------------------------------------------------------------------}
Procedure K_elem2;
{
Procedure K_elem2(    iMat: integer;
```

```
                var Exyz: MatrixR_Nnex3;    var for speed
                var Mat: MatrixR_Nmatx6;    var for speed
                    Angle: Rtype;
                var Ke: MatrixR_NneDxNneD);
)
(
   Orthotropic 20 node 3-dimensional element, quadratic displacement.
   Principal material direction at an angle from local coordinates in the
   plane of the first two directions.
   Input:
      iMat: Material #
      Exyz: Coordinates for the element
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      Angle: angle for the material in the element
   Output:
      Ke: Element Stiffness matrix:  |Fx1|          |u1 |
                                     |Fy1|          |v1 |
                                     |...|  =  [Ke]. |...|
                                     |FxN|          |uN |
                                     |FyN|          |vN |
)
var
   irow,row,col: Byte;  { speed is increased by using a smaller integer type }
   Ngq,iGQ,Xs,Ys: Byte;
   M,J_det: Rtype;
   EI: MatrixR_6x6;
   BF: MatrixR_6xNneD;
   P: array[1..60,1..6] of Rtype;
   J: MatrixR_3x3;
   enb: MatrixR_NgqxD;
   Multiplier: VectR_Ngq;
begin
   write('   Getting stiffness ');
   Xs:=WhereX;
   Ys:=WhereY;
(
                Set Gauss-Quadrature coefficients
)
   Ngq:=27;
   Get_enb(enb);
   Get_Multiplier(Multiplier);
(
                Get EI matrix
)
   Get_EI(iMat,Mat,Angle,EI);  { takes 0.11s on PC XT 4.77 MHz }
(
                Initialize Ke
)
   For row:=1 to 60 do
      For col:=1 to 60 do
         Ke[row,col]:=0.0;
(
```

```
                Intergrate by summing the Product of matrices
}
  For iGQ:=1 to Ngq do
    begin
    GotoXY(Xs,Ys);
    write(Ngq-iGQ+1,' ');
    Get_BF(enb[iGQ,1],enb[iGQ,2],enb[iGQ,3],Exyz,BF,J);
{
                P = [BF]transposed x [EI]
}
    For row:=1 to 60 do
      For col:=1 to 6 do
        P[row,col]:=BF[1,row]*EI[1,col]
                   +BF[2,row]*EI[2,col]
                   +BF[3,row]*EI[3,col]
                   +BF[4,row]*EI[4,col]
                   +BF[5,row]*EI[5,col]
                   +BF[6,row]*EI[6,col];
{
                J_det = Determinant of [J]
                Verify that determinant is positive
}
    Determinant3(J,J_Det);
    If J_Det<0 then
      begin
      writeln;
      writeln('Error: This element is too distorted!');
      writeln('Possible causes:');
      writeln('- Improper element node numbering in your ELEM commands.');
      writeln('  The numbering direction is important for an element.');
      writeln('- Element is curved too much.');
      Bad_Beep;
      wait;
      Halt;
      end;
{
                [Ke] = Sum of: HXi.HEta.HZeta x [P] x [BF] x det[J]
                This is the slowest part of this procedure
                  due to the large loops
                Ke is symmetric so only the lower triangular section
                  is calculated, then we copy for the upper at end only
                Speed: all steps in integration loop prior to this point
                       takes 0.38s on 4.77 MHz XT
                       Following multiplication loop: 1.21s
}
    M:=Multiplier[iGQ]*J_det;
    For row:=1 to 60 do
      For col:=1 to row do
        Ke[row,col]:=Ke[row,col]+M*(P[row,1]*BF[1,col]
                                   +P[row,2]*BF[2,col]
                                   +P[row,3]*BF[3,col]
                                   +P[row,4]*BF[4,col]
```

```
                                        +P[row,5]*BF[5,col]
                                        +P[row,6]*BF[6,col]);
      end;
{
                  Copy Lower triangular part to upper (symmetric)
}
   For row:=1 to 60 do
      For col:=row+1 to 60 do
         Ke[row,col]:=Ke[col,row];
   GotoXY(Xs,Ys);
   writeln('  ');
end;
{-------------------------------------------------------------------}
Procedure Get_OptimalToNode(    Ngq,Nne: Byte;
                            var enb: MatrixR_NgqxD;
                            var TR2: MatrixR_NnexNgq);
{
   Input:
      Ngq: number of optimal sampling points
      Nne: number of nodes in element
      enb[iGQ,iD]: local coordinates enb[iGQ,1], enb[iGQ,2], enb[iGQ,3]
                   for gauss point number iGQ
   Output:
      TR2: [strain-stress]nodes = [TR2].[strain-stress]optimal
}
var
   iGQ,iNne: Byte;
   a,a1,a2: Rtype;
   S: array[1..4,1..27] of Rtype;
   E: array[1..20,1..4] of Rtype;
begin
   a:=sqrt(3/5);  {gauss-quadrature locations: -a, 0, +a}
{
   Evaluate [TR2] matrix in:
      [strain-stress]nodes = [TR2].[strain-stress]optimal
   [TR2] = [E].[S]
}
   E[1,1]:=1.0;  E[1,2]:=-1.0;  E[1,3]:=-1.0;  E[1,4]:=-1.0;
   E[2,1]:=1.0;  E[2,2]:= 0.0;  E[2,3]:=-1.0;  E[2,4]:=-1.0;
   E[3,1]:=1.0;  E[3,2]:= 1.0;  E[3,3]:=-1.0;  E[3,4]:=-1.0;
   E[4,1]:=1.0;  E[4,2]:= 1.0;  E[4,3]:= 0.0;  E[4,4]:=-1.0;
   E[5,1]:=1.0;  E[5,2]:= 1.0;  E[5,3]:= 1.0;  E[5,4]:=-1.0;
   E[6,1]:=1.0;  E[6,2]:= 0.0;  E[6,3]:= 1.0;  E[6,4]:=-1.0;
   E[7,1]:=1.0;  E[7,2]:=-1.0;  E[7,3]:= 1.0;  E[7,4]:=-1.0;
   E[8,1]:=1.0;  E[8,2]:=-1.0;  E[8,3]:= 0.0;  E[8,4]:=-1.0;
   E[9,1]:=1.0;  E[9,2]:=-1.0;  E[9,3]:=-1.0;  E[9,4]:= 0.0;
   E[10,1]:=1.0; E[10,2]:= 1.0; E[10,3]:=-1.0; E[10,4]:= 0.0;
   E[11,1]:=1.0; E[11,2]:= 1.0; E[11,3]:= 1.0; E[11,4]:= 0.0;
   E[12,1]:=1.0; E[12,2]:=-1.0; E[12,3]:= 1.0; E[12,4]:= 0.0;
   E[13,1]:=1.0; E[13,2]:=-1.0; E[13,3]:=-1.0; E[13,4]:= 1.0;
   E[14,1]:=1.0; E[14,2]:= 0.0; E[14,3]:=-1.0; E[14,4]:= 1.0;
   E[15,1]:=1.0; E[15,2]:= 1.0; E[15,3]:=-1.0; E[15,4]:= 1.0;
```

```
   E[16,1]:=1.0; E[16,2]:= 1.0; E[16,3]:= 0.0; E[16,4]:= 1.0;
   E[17,1]:=1.0; E[17,2]:= 1.0; E[17,3]:= 1.0; E[17,4]:= 1.0;
   E[18,1]:=1.0; E[18,2]:= 0.0; E[18,3]:= 1.0; E[18,4]:= 1.0;
   E[19,1]:=1.0; E[19,2]:=-1.0; E[19,3]:= 1.0; E[19,4]:= 1.0;
   E[20,1]:=1.0; E[20,2]:=-1.0; E[20,3]:= 0.0; E[20,4]:= 1.0;
   a1:=1/27;
   a2:=1/18/a/a;
   For iGQ:=1 to Ngq do
      begin
      S[1,iGQ]:=a1;
      S[2,iGQ]:=a2*enb[iGQ,1];
      S[3,iGQ]:=a2*enb[iGQ,2];
      S[4,iGQ]:=a2*enb[iGQ,3];
      end;
   For iNne:=1 to Nne do
      For iGQ:=1 to Ngq do
         TR2[iNne,iGQ]:=E[iNne,1]*S[1,iGQ]
                      + E[iNne,2]*S[2,iGQ]
                      + E[iNne,3]*S[3,iGQ]
                      + E[iNne,4]*S[4,iGQ];
end;
{-------------------------------------------------------------------}
Procedure SS2;
{
Procedure SS2(var F_Displ: FileR;
              var Exyz: MatrixR_Nnex3;     var for speed
              var Elem: VectI_Nne2;        var for speed
              var Mat: MatrixR_Nmatx6;     var for speed
                  Angle: Rtype;
              var FiberStrain,FiberStress,ElemStrain,ElemStress,
                    GlobalStrain,GlobalStress: VectR_NstressNne);
}
{
   Input:
      N: number of nodes;
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      Exyz: Coordinates for the element
      Ne: Number of elements;
      Elem[1..Nne+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      Angle: angle for the material in the element
   Output:
      FiberStrain[1..N_Stress(Dim).Nne]: Material direction e1 e2 e3 ...e23
                                         for each node of element
      FiberStress[1..N_Stress(Dim).Nne]: Material direction s1 s2 s3 ...s23
                                         for each node of element
      ElemStrain[1..N_Stress(Dim).Nne]: Element direction e1 e2 e3 ...e23
                                         for each node of element
      ElemStress[1..N_Stress(Dim).Nne]: Element direction s1 s2 s3 ...s23
```

```
                                        for each node of element
      GlobalStrain[1..N_Stress(Dim).Nne]: Global direction e1 e2 e3 ...e23
                                        for each node of element
      GlobalStress[1..N_Stress(Dim).Nne]: Global direction s1 s2 s3 ...s23
                                        for each node of element
)
var
   jPos: Integer;
   row,col,iD,iNne,Nne,iGQ,Ngq: Byte;
   iTR,TR,TRf,EI: MatrixR_6x6;
   dNl: MatrixR_3xNne;
   d: array[1..60] of Rtype;
   BF: MatrixR_6xNneD;
   J,V: MatrixR_3x3;
   Displ: VectR_D;
   iRow: array[1..27,1..6] of integer;
   NodePos,GQpos: integer;
   Xs,Ys: integer;
   enb: MatrixR_NgqxD;
   TR2: MatrixR_NnexNgq;
   FiberStrainGQ,FiberStressGQ,ElemStrainGQ,ElemStressGQ,
     GlobalStrainGQ,GlobalStressGQ: array[1..162] of Rtype; {162=27*6}
begin
   Nne:=20;       {20 node element}
   Ngq:=27;       {27 sampling points using Gauss-Quadrature locations}
   Get_enb(enb);
   Xs:=WhereX;
   Ys:=WhereY;
{
                  Get Displacements of nodes in the the format
                  [d] = u1..uN v1..vN w1..wN
}
   For iNne:=1 to Nne do
      begin
      Displ_R_Read(F_Displ,abs(Elem[2+iNne]),Dim3,Displ);
      For iD:=1 to Dim3 do
         d[Lin(iD,iNne,Nne)]:=Displ[iD];
      end;
{
                  Get [EI]
                  EI: Stiffness matrix in element coordinate system
                      [Element Tensorial Stress] = [EI] [Tensorial Strain]
}
   Get_EI(Elem[1],Mat,Angle,EI);
{
                  Calculate only once iRow[iGQ,row]:=Lin(iGQ,row,6)
}
   For iGQ:=1 to Ngq do
      For row:=1 to 6 do
         iRow[iGQ,row]:=Lin(iGQ,row,6);
{
           Get [TRf]
```

```
          [On-axis stress] = [TRf]trans.[Element stress]
}
   Get_TR(Angle,TRf);
   For iGQ:=1 to Ngq do
      begin
      gotoXY(Xs,Ys);
      write(Ngq-iGQ+7,' ');
{
                Get [BF]
                BF: [element tensorial strain] = [BF]
                    x [u1..uN v1..vN w1..wN]element
}
      Get_BF(enb[iGQ,1],enb[iGQ,2],enb[iGQ,3],Exyz,BF,J);
{
                Get element direction strain
                [ElemStrainGQ] = [BF].[d]
}
      For row:=1 to 6 do
         begin
         jPos:=iRow[iGQ,row];
         ElemStrainGQ[jPos]:=0.0;
         For col:=1 to 60 do
             ElemStrainGQ[jPos] := ElemStrainGQ[jPos] + BF[row,col]*d[col];
         end;
{
                Get element direction stress
                [ElemStressGQ] = [EI].[ElemStrainGQ]
}
      For row:=1 to 6 do
         begin
         jPos:=iRow[iGQ,row];
         ElemStressGQ[jPos]:=0.0;
         For col:=1 to 6 do
            ElemStressGQ[jPos] := ElemStressGQ[jPos]
                                + EI[row,col]*ElemStrainGQ[iRow[iGQ,col]];
         end;
{
                Get on-axis direction strain at optimum sampling points
                [FiberStrainGQ] = [TRf]trans.[ElemStrainGQ]
}
      For row:=1 to 6 do
         begin
         jPos:=iRow[iGQ,row];
         FiberStrainGQ[jPos] := TRf[1,row]*ElemStrainGQ[iRow[iGQ,1]]
                              + TRf[2,row]*ElemStrainGQ[iRow[iGQ,2]]
                              + TRf[3,row]*ElemStrainGQ[iRow[iGQ,3]]
                              + TRf[4,row]*ElemStrainGQ[iRow[iGQ,4]]
                              + TRf[5,row]*ElemStrainGQ[iRow[iGQ,5]]
                              + TRf[6,row]*ElemStrainGQ[iRow[iGQ,6]];
         FiberStressGQ[jPos] := TRf[1,row]*ElemStressGQ[iRow[iGQ,1]]
                              + TRf[2,row]*ElemStressGQ[iRow[iGQ,2]]
                              + TRf[3,row]*ElemStressGQ[iRow[iGQ,3]]
```

```
                        + TRf[4,row]*ElemStressGQ[iRow[iGQ,4]]
                        + TRf[5,row]*ElemStressGQ[iRow[iGQ,5]]
                        + TRf[6,row]*ElemStressGQ[iRow[iGQ,6]];
      end;
{
              Get V: ¦l1 m1 n1¦
                     ¦l2 m2 n2¦
                     ¦l3 m3 n3¦
}
   Get_dNlocal(enb[iGQ,1],enb[iGQ,2],enb[iGQ,3],dNl);
   Get_V(J,V);
{
              Get transformation matrix: [Global strain or stress GQ] =
                 [TR]inversed x [Local strain or stress GQ]
  [TR] = ¦ l1.l1 m1.m1 n1.n1   2l1.m1      2n1.l1      2m1.n1    ¦
         ¦ l2.l2 m2.m2 n2.n2   2l2.m2      2n2.l2      2m2.n2    ¦
         ¦ l3.l3 m3.m3 n3.n3   2l3.m3      2n3.l3      2m3.n3    ¦
         ¦ l1.l2 m1.m2 n1.n2 l1.m2+l2.m1 l1.n2+l2.n1 m1.n2+m2.n1 ¦
         ¦ l1.l3 m1.m3 n1.n3 l1.m3+l3.m1 l1.n3+l3.n1 m1.n3+m3.n1 ¦
         ¦ l2.l3 m2.m3 n2.n3 l2.m3+l3.m2 l2.n3+l3.n2 m2.n3+m3.n2 ¦
}
   For row:=1 to 3 do
      begin
      For col:=1 to 3 do
         begin
         TR[row,col]:=sqr(V[row,col]);
         end;
      TR[row,4]:=2*V[row,1]*V[row,2];
      TR[row,5]:=2*V[row,3]*V[row,1];
      TR[row,6]:=2*V[row,2]*V[row,3];
      end;
   For col:=1 to 3 do
      begin
      TR[4,col]:=V[1,col]*V[2,col];
      TR[5,col]:=V[1,col]*V[3,col];
      TR[6,col]:=V[2,col]*V[3,col];
      end;
   TR[4,4]:=V[1,1]*V[2,2]+V[2,1]*V[1,2];
   TR[4,5]:=V[1,1]*V[2,3]+V[2,1]*V[1,3];
   TR[4,6]:=V[1,2]*V[2,3]+V[2,2]*V[1,3];
   TR[5,4]:=V[1,1]*V[3,2]+V[3,1]*V[1,2];
   TR[5,5]:=V[1,1]*V[3,3]+V[3,1]*V[1,3];
   TR[5,6]:=V[1,2]*V[3,3]+V[3,2]*V[1,3];
   TR[6,4]:=V[2,1]*V[3,2]+V[3,1]*V[2,2];
   TR[6,5]:=V[2,1]*V[3,3]+V[3,1]*V[2,3];
   TR[6,6]:=V[2,2]*V[3,3]+V[3,2]*V[2,3];
{
   [TR]inverse = Where [TR] has been transposed,
                 then divided into for parts for which we do:
                 ¦ same  double ¦
                 ¦ Half   same  ¦
}
```

```
      For row:=1 to 6 do
         For col:=1 to 6 do
            iTR[row,col]:=TR[col,row];
      For row:=1 to 3 do
         For col:=1 to 3 do
            begin
            iTR[row,col+3]:=2*iTR[row,col+3];
            iTR[row+3,col]:=0.5*iTR[row+3,col];
            end;
{
                  Get global strain and stress at optimum sampling points
}
      For row:=1 to 6 do
         begin
         jPos:=iRow[iGQ,row];
         GlobalStrainGQ[jPos] := iTR[row,1]*ElemStrainGQ[iRow[iGQ,1]]
                               + iTR[row,2]*ElemStrainGQ[iRow[iGQ,2]]
                               + iTR[row,3]*ElemStrainGQ[iRow[iGQ,3]]
                               + iTR[row,4]*ElemStrainGQ[iRow[iGQ,4]]
                               + iTR[row,5]*ElemStrainGQ[iRow[iGQ,5]]
                               + iTR[row,6]*ElemStrainGQ[iRow[iGQ,6]];
         GlobalStressGQ[jPos] := iTR[row,1]*ElemStressGQ[iRow[iGQ,1]]
                               + iTR[row,2]*ElemStressGQ[iRow[iGQ,2]]
                               + iTR[row,3]*ElemStressGQ[iRow[iGQ,3]]
                               + iTR[row,4]*ElemStressGQ[iRow[iGQ,4]]
                               + iTR[row,5]*ElemStressGQ[iRow[iGQ,5]]
                               + iTR[row,6]*ElemStressGQ[iRow[iGQ,6]];
         end;
      end;
{
   Evaluate [TR2] matrix in:
      [strain-stress]nodes = [TR2].[strain-stress]optimal
   [TR2] = [E].[S]
}
   Get_OptimalToNode(Ngq,Nne,enb,TR2);
{
                  Get strain-stress in Local-Global at nodes
                  using values at optimum sampling points (gauss points)
}
   For row:=1 to 6 do
      begin
      gotoXY(Xs,Ys);
      write(6-row+1,' ');
      For iNne:=1 to Nne do
         begin
         NodePos:=iRow[iNne,row];
         FiberStrain[NodePos]:=0.0;
         FiberStress[NodePos]:=0.0;
         ElemStrain[NodePos]:=0.0;
         ElemStress[NodePos]:=0.0;
         GlobalStrain[NodePos]:=0.0;
         GlobalStress[NodePos]:=0.0;
```

```
            For iGQ:=1 to Ngq do
               begin
               GQpos:=iRow[iGQ,row];
               FiberStrain[NodePos]:=FiberStrain[NodePos]
                                   + TR2[iNne,iGQ]*FiberStrainGQ[GQpos];
               FiberStress[NodePos]:=FiberStress[NodePos]
                                   + TR2[iNne,iGQ]*FiberStressGQ[GQpos];
               ElemStrain[NodePos]:=ElemStrain[NodePos]
                                   + TR2[iNne,iGQ]*ElemStrainGQ[GQpos];
               ElemStress[NodePos]:=ElemStress[NodePos]
                                   + TR2[iNne,iGQ]*ElemStressGQ[GQpos];
               GlobalStrain[NodePos]:=GlobalStrain[NodePos]
                                    + TR2[iNne,iGQ]*GlobalStrainGQ[GQpos];
               GlobalStress[NodePos]:=GlobalStress[NodePos]
                                    + TR2[iNne,iGQ]*GlobalStressGQ[GQpos];
               end;
            end;
         end;
      gotoXY(Xs,Ys);
      write('  ');
      gotoXY(Xs,Ys);
   end;
   {-------------------------------------------------------------------}
   Procedure Get_dA(    e,n,b: Rtype;
                        Face: Byte;
                    var Exyz: MatrixR_Nnex3;  (var for speed)
                    var dA: VectR_D);
   {
      Input:
        e,n,b: Xi, Eta, Zeta: local coordinates in the element
        Face: which face where a traction is applied
              1: b=-1 (bottom)
              2: n=-1 (lateral face with nodes 1-2-3)
              3: e=1  (other faces in trigonometric order looking in -b dir.)
              4: n=1
              5: e=-1
              6: b=1 (top)
        Exyz: coordinates of nodes in element
      Output:
        dA: Direction of normal to face at given e,b,n
   }
   var
      iV,iD: integer;
      Vindex: array[1..2] of integer;
      Vface: array[1..2,1..3] of Rtype;
      J: MatrixR_3x3;
      dNl: MatrixR_3xNne;
   begin
   {
                    Get index of first and second vector for
                    vector multiplication in order to have the face
                    direction
```

```
)
  Case Face of
    1: begin   { vector product of (n)x(e) is in -b direction }
       Vindex[1]:=2;
       Vindex[2]:=1;
       end;
    2: begin   { vector product of (e)x(b) is in -n direction }
       Vindex[1]:=1;
       Vindex[2]:=3;
       end;
    3: begin   { vector product of (n)x(b) is in +e direction }
       Vindex[1]:=2;
       Vindex[2]:=3;
       end;
    4: begin   { vector product of (b)x(e) is in +n direction }
       Vindex[1]:=3;
       Vindex[2]:=1;
       end;
    5: begin   { vector product of (b)x(n) is in -e direction }
       Vindex[1]:=3;
       Vindex[2]:=2;
       end;
    6: begin   { vector product of (e)x(n) is in +b direction }
       Vindex[1]:=1;
       Vindex[2]:=2;
       end;
    end;
{
                Get J matrix
                [J] = ¦ dx/de dy/de dz/de ¦
                      ¦ dx/dn dy/dn dz/dn ¦
                      ¦ dx/db dy/db dz/db ¦
}
  Get_dNlocal(e,n,b,dNl);
  Get_J(e,n,b,dNl,Exyz,J);
{
                Fill the 2 vector
}
  For iV:=1 to 2 do
    For iD:=1 to Dim3 do
      Vface[iV,iD]:=J[Vindex[iV],iD];
{
                Get Normal to Face vector
                dA = Vface[1] x Vface[2]
                   note: V1 x V2 = ¦  i   j   k  ¦
                                   ¦ l1  m1  n1  ¦ V1
                                   ¦ l2  m2  n2  ¦ V2
                                 =  (m1.n2-m2.n1)i+
                                   -(l1.n2-l2.n1)j+
                                    (l1.m2-l2.m1)k
}
  dA[1]:= (Vface[1,2]*Vface[2,3]-Vface[2,2]*Vface[1,3]);
```

```
   dA[2]:=-(Vface[1,1]*Vface[2,3]-Vface[2,1]*Vface[1,3]);
   dA[3]:= (Vface[1,1]*Vface[2,2]-Vface[2,1]*Vface[1,2]);
{
                DO NOT Normalize V3
}
end;
{-------------------------------------------------------------------}
procedure Traction2;
{
procedure Traction2(    Face: Byte;
                        Traction: Rtype;
                    var Exyz: MatrixR_Nnex3;  var for spaed
                    var Tindex: VectI_Nne;
                    var Nt: Byte;
                    var Ft: MatrixR_Nnex3);
}
{
   Input:
      Face: which face where a traction is applied
            1: b=-1 (bottom)
            2: n=-1 (lateral face with nodes 1-2-3)
            3: e=1  (other faces in trigonometric order looking in -b dir.)
            4: n=1
            5: e=-1
            6: b=1 (top)
      Traction: Traction value on Face (inverse of pressure)
      Exyz: coordinates of nodes in element
   Output:
      Tindex: index of nodes in element in contact with the face
      Nt: number of nodes in contact with the face
      Ft[iNne,Direction]: Forces on element nodes due to traction on face
         Add these to node forces
}
var
   i,iD,Xs,Ys,id1,id2: integer;
   count,NgqDirection,iGQ,iNne,Nne: Byte;
   d1,d2,Hd1,Hd2: Array[1..3] of Rtype;
   Multiplier,e,n,b: Rtype;
   Ne: VectR_Nne;
   dA: VectR_D;
begin
   write('  Setting forces to simulate pressure ');
   Xs:=WhereX;
   Ys:=WhereY;
   Nne:=Get_Nne(2);
   For iNne:=1 to Nne do
      begin
      Tindex[iNne]:=0;
      For iD:=1 to Dim3 do
         Ft[iNne,iD]:=0;
      end;
{
```

```
                Index local direction so that
                last index indicates fixed direction
}
{
                Set Gauss-Quadrature coefficients
}
   NgqDirection:=3;
   d1[1]:=sqrt(3/5);
   d1[2]:=-d1[1];
   d1[3]:=0.0;
   Hd1[1]:=5/9;
   Hd1[2]:=5/9;
   Hd1[3]:=8/9;
   For iGQ:=1 to NgqDirection do
      begin
      d2[iGQ]:=d1[iGQ];
      Hd2[iGQ]:=Hd1[iGQ];
      end;
{
                Intergrate by summing the Product of matrices
                Ft = -[ sum of ([N]transposed.Traction.[dA]) ]
}
   Count:=0;
   For id1 := 1 to NgqDirection do
      For id2 := 1 to NgqDirection do
         begin
         GotoXY(Xs,Ys);
         write(Ngq*Direction*sqr(NgqDirection)-count,' ');
         Inc(Count);
{
                Get integrating positions
}
         Case Face of
            1: begin
               e:=d1[iD1];
               n:=d2[iD2];
               b:=-1.0;
               end;
            2: begin
               e:=d1[iD1];
               n:=d2[iD2];
               b:=+1.0;
               end;
            2: begin
               e:=d1[iD1];
               b:=d2[iD2];
               n:=-1.0;
               end;
            4: begin
               e:=d1[iD1];
               b:=d2[iD2];
               n:=+1.0;
```

```
                end;
            3: begin
                n:=d1[iD1];
                b:=d2[iD2];
                e:=+1.0;
                end;
            5: begin
                n:=d1[iD1];
                b:=d2[iD2];
                e:=-1.0;
                end;
            end;
{
                Get dA
}
        Get_dA(e,n,b,Face,Exyz,dA);
{
                Sum product of matrices
                Multiplier is positive because Ft will be added
                   to node forces
}
        Multiplier:=Traction*Hd1[iD1]*Hd2[id2];
        Get_N(e,n,b,Ne);
        For iNne:=1 to Nne do
           For iD:=1 to Dim3 do
              Ft[iNne,iD]:=Ft[iNne,iD]+Multiplier*Ne[iNne]*dA[iD];
        end;
{
                Set Tindex
}
   Nt:=8;
   Case Face of
      1: begin
         Tindex[1]:=1;
         Tindex[2]:=2;
         Tindex[3]:=3;
         Tindex[4]:=4;
         Tindex[5]:=5;
         Tindex[6]:=6;
         Tindex[7]:=7;
         Tindex[8]:=8;
         end;
      2: begin
         Tindex[1]:=1;
         Tindex[2]:=2;
         Tindex[3]:=3;
         Tindex[4]:=9;
         Tindex[5]:=10;
         Tindex[6]:=13;
         Tindex[7]:=14;
         Tindex[8]:=15;
         end;
```

```
      3: begin
         Tindex[1]:=3;
         Tindex[2]:=4;
         Tindex[3]:=5;
         Tindex[4]:=10;
         Tindex[5]:=11;
         Tindex[6]:=15;
         Tindex[7]:=16;
         Tindex[8]:=17;
         end;
      4: begin
         Tindex[1]:=5;
         Tindex[2]:=6;
         Tindex[3]:=7;
         Tindex[4]:=11;
         Tindex[5]:=12;
         Tindex[6]:=17;
         Tindex[7]:=18;
         Tindex[8]:=19;
         end;
      5: begin
         Tindex[1]:=7;
         Tindex[2]:=8;
         Tindex[3]:=1;
         Tindex[4]:=12;
         Tindex[5]:=9;
         Tindex[6]:=19;
         Tindex[7]:=20;
         Tindex[8]:=13;
         end;
      6: begin
         Tindex[1]:=13;
         Tindex[2]:=14;
         Tindex[3]:=15;
         Tindex[4]:=16;
         Tindex[5]:=17;
         Tindex[6]:=18;
         Tindex[7]:=19;
         Tindex[8]:=20;
         end;
      end;
{
                Erase line
}
   GotoXY(1,Ys);
   For i:=1 to Xs+1 do write(' ');
   GotoXY(1,Ys);
end;
{-------------------------------------------------------------}
end.
```

EXYZ.PAS

```
{$N+}
Unit Exyz;
Interface
   Uses Declare,Nne_Dim,File_RW;
Procedure Get_Exyz(    Elem: VectI_Nne2;
                   var F_xyz: FileR;
                   var Exyz: MatrixR_Nnex3);
{-------------------------------------------------------------------}
Implementation
Procedure Get_Exyz(    Elem: VectI_Nne2;
                   var F_xyz: FileR;
                   var Exyz: MatrixR_Nnex3);
{
Procedure Get_Exyz(    Elem: VectI_Nne2;
                   var F_xyz: FileR;
                   var Exyz: MatrixR_Nnex3);
}
{
   Input:
      Elem[1..Nne+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
   Output:
      Exyz: coordinates for the element nodes
}
var
   Etype,Dim,iD,iNne: Itype;
   NodeXYZ: VectR_D;
begin
   Etype:=Elem[2];
   Dim:=Get_D(Etype);
   For iNne:=1 to Get_Nne(Etype) do
      begin
      D_R_Read(F_xyz,abs(Elem[2+iNne]),Dim,NodeXYZ);
      For iD:=1 to Dim do
         Exyz[iNne,iD]:=NodeXYZ[iD];
      end;
end;
{-------------------------------------------------------------------}
end.
```

FILE_RW.PAS

```
{$N+}
Unit File_RW;
Interface
Uses
   Declare,Linear,Nne_Dim;
Procedure D_R_Read(var F: FileR;
                       row,Dim: IType;
                   var V: VectR_D);
Procedure D_R_Write(var F: FileR;
                        row,Dim: IType;
                        V: VectR_D);
Procedure D_R_O(var F: FileR;
                    row,Dim: IType);
Procedure Nne2_I_Read(var F: FileI;
                          row,NneMax: IType;
                      var V: VectI_Nne2);
Procedure Nne2_I_Write(var F: FileI;
                           row,NneMax: IType;
                           V: VectI_Nne2);
Procedure Nne2_I_O(var F: FileI;
                       row,NneMax: IType);
Procedure D_Byte_Read(var F: FileByte;
                          row,Dim: IType;
                      var V: VectByte_D);
Procedure D_Byte_Write(var F: FileByte;
                           row,Dim: IType;
                           V: VectByte_D);
Procedure D_Byte_O(var F: FileByte;
                       row,Dim: IType);
Procedure R_Read(var F: FileR;
                     row: IType;
                 var numberR: Rtype);
Procedure R_Write(var F: FileR;
                      row: IType;
                      NumberR: Rtype);
Procedure R_O(var F: FileR;
                  row: IType);
Procedure I_Read(var F: FileI;
                     row: IType;
                 var numberI: IType);
Procedure I_Write(var F: FileI;
                      row: IType;
                      numberI: IType);
Procedure I_O(var F: FileI;
                  row: IType);
Procedure Byte_Read(var F: FileByte;
                        row: IType;
                    var number: Byte);
Procedure Byte_Write(var F: FileByte;
                         row: IType;
```

```
                        Number: Byte);
Procedure Byte_O(var F:  FileByte;
                     row: IType);
Procedure get_Fd_Elem(     Dim: Itype;
                       var Elem: VectI_Nne2;   { var for speed }
                       var F_Fd: FileByte;
                       var Fd_Elem: MatrixByte_NnexD);
{-------------------------------------------------------------------}
Implementation
Procedure D_R_Read;
{
Procedure D_R_Read(var  F: FileR;
                        row,Dim: IType;
                    var V: VectR_D);
}
{
   Input:
      F: File containing sub-Vectors
      row: position of node in Vector
      Dim: Dimension of problem
   Output:
      V: Sub-Vector
         Reads the V Sub-Vector from a file
}
var
   i_sub: IType;
   i: IType;
begin
   i_sub:=Lin(row,1,Dim)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   For i:=1 to Dim do
      Read(F,V[i]);
end;
{-------------------------------------------------------------------}
Procedure D_R_Write;
{
Procedure D_R_Write(var  F: FileR;
                         row,Dim: IType;
                         V: VectR_D);
}
{
   Input:
      F: File containing sub-Vectors
      row: position of node in Vector
      Dim: Dimension of problem
      V: Sub-Vector
   Output:
      Writes the V Sub-Vector to a file
}
var
   i_sub: IType;
   i: IType;
```

```
begin
   i_sub:=Lin(row,1,Dim)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   For i:=1 to Dim do
      Write(F,V[i]);
end;
{-------------------------------------------------------------------}
Procedure D_R_0;
{
Procedure D_R_0(var F: FileR;
                    row,Dim: IType);
}
{
   Input:
      F: File containing sub-Vectors
      row: position of node in Vector
      Dim: Dimension of problem
   Output:
      Initializes the V Sub-Vector as 0 to a file
}
var
   i_sub: IType;
   i: IType;
   X_Rtype: Rtype;
begin
   i_sub:=Lin(row,1,Dim)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   X_Rtype:=0.0;
   For i:=1 to Dim do
      Write(F,X_Rtype);
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
Procedure Nne2_I_Read;
{
Procedure Nne2_I_Read(var F: FileI;
                          row,NneMax: IType;
                      var V: VectI_Nne2);
}
{
   Input:
      F: File containing sub-Vectors
      row: position in Global Vector
      NneMax: Maximum Number of nodes
   Output:
      V: Sub-Vector Read from File F
}
var
   i_sub: IType;
   i: IType;
begin
   i_sub:=Lin(row,1,NneMax+2)-1;     { 0 for the first sub-matrix }
```

```
   Seek(F,i_sub);
   For i:=1 to 2 do
      Read(F,V[i]);
   For i:=2+1 to 2+Get_Nne(V[2]) do
      Read(F,V[i]);
end;
{-------------------------------------------------------------------}
Procedure Nne2_I_Write;
{
Procedure Nne2_I_Write(var F: FileI;
                           row,NneMax: IType;
                           V: VectI_Nne2);
}
{
   Input:
      F: File containing sub-Vectors
      row: position in Global Vector
      Nne: Number of nodes in this element
      NneMax: Maximum Number of nodes
   Output:
      V: Sub-Vector written to File F
}
var
   i_sub: IType;
   i: IType;
begin
   i_sub:=Lin(row,1,NneMax+2)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   For i:=1 to 2+Get_Nne(V[2]) do
     Write(F,V[i]);
end;
{-------------------------------------------------------------------}
Procedure Nne2_I_0;
{
Procedure Nne2_I_0(var F: FileI;
                       row,NneMax: IType);
}
{
   Input:
      F: File containing sub-Vectors
      row: position in Global Vector
      NneMax: Maximum Number of nodes
   Output:
      V: Sub-Vector written to File F
}
var
   i_sub,X_I: IType;
   i: IType;
begin
   i_sub:=Lin(row,1,NneMax+2)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   X_I:=0;
```

```
   For i:=1 to 2+NneMax do
      Write(F,X_I);
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
Procedure D_Byte_Read;
{
Procedure D_Byte_Read(var F: FileByte;
                          row,Dim: IType;
                      var V: VectByte_D);
}
{
   Input:
      F: File containing sub-Vectors
      row: position of node in Vector
      Dim: Dimension of problem
   Output:
      V: Sub-Vector
         Reads the V Sub-Vector from a file
}
var
   i_sub: IType;
   i: IType;
begin
   i_sub:=Lin(row,1,Dim)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   For i:=1 to Dim do
      Read(F,V[i]);
end;
{-------------------------------------------------------------------}
Procedure D_Byte_Write;
{
Procedure D_Byte_Write(var F: FileByte;
                           row,Dim: IType;
                           V: VectByte_D);
}
{
   Input:
      F: File containing sub-Vectors
      row: position of node in Vector
      Dim: Dimension of problem
      V: Sub-Vector
   Output:
      Writes the V Sub-Vector to a file
}
var
   i_sub: IType;
   i: IType;
begin
   i_sub:=Lin(row,1,Dim)-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   For i:=1 to Dim do
```

```
      Write(F,V[i]);
end;
{-------------------------------------------------------------------}
Procedure D_Byte_O;
{
Procedure D_Byte_O(var F: FileByte;
                       row,Dim: IType);
}
{
   Input:
      F: File containing sub-Vectors
      row: position of node in Vector
      Dim: Dimension of problem
   Output:
      Initializes the V Sub-Vector to a file
}
var
   i_sub: IType;
   i: IType;
   X_Byte: Byte;
begin
   i_sub:=Lin(row,1,Dim)-1;      { 0 for the first sub-matrix }
   Seek(F,i_sub);
   X_Byte:=0;
   For i:=1 to Dim do
      Write(F,X_Byte);
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
Procedure R_Read;
{
Procedure R_Read(var F: FileR;
                     row: IType;
                 var numberR: Rtype);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      numberR: Number
}
var
   i_sub: IType;
begin
   i_sub:=row-1;      { 0 for the first sub-matrix }
   Seek(F,i_sub);
   Read(F,NumberR);
end;
{-------------------------------------------------------------------}
Procedure R_Write;
{
```

```
Procedure R_Write(var F: FileR;
                      row: IType;
                      NumberR: Rtype);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
      NumberR: Number
   Output:
      Number is written to file F
}
var
   i_sub: IType;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   write(F,NumberR);
end;
{-------------------------------------------------------------------------}
Procedure R_0;
{
Procedure R_0(var F: FileR;
                  row: IType);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      Number is initialized to file F
}
var
   i_sub: IType;
   X_Rtype: Rtype;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   X_Rtype:=0.0;
   write(F,X_Rtype);
end;
{-------------------------------------------------------------------------}
{-------------------------------------------------------------------------}
Procedure I_Read;
{
Procedure I_Read(var F: FileI;
                      row: IType;
                  var numberI: IType);
}
{
   Input:
      F: File containing numbers
```

```
      row: position of node in Vector
   Output:
      numberI: Number
}
var
   i_sub: IType;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   Read(F,numberI);
end;
{-------------------------------------------------------------------------}
Procedure I_Write;
{
Procedure I_Write(var F: FileI;
                      row: IType;
                      numberI: IType);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
      numberI: Number
   Output:
      Number is written to file F
}
var
   i_sub: IType;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   write(F,numberI);
end;
{-------------------------------------------------------------------------}
Procedure I_0;
{
Procedure I_0(var F: FileI;
                  row: IType);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      Number is initialized to file F
}
var
   i_sub,X_I: IType;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   X_I:=0;
```

```
   write(F,X_I);
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
Procedure Byte_Read;
{
Procedure Byte_Read(var F: FileByte;
                        row: IType;
                    var number: Byte);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      number: Number
}
var
   i_sub: IType;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   Read(F,number);
end;
{-------------------------------------------------------------------}
Procedure Byte_Write;
{
Procedure Byte_Write(var F: FileByte;
                         row: IType;
                         Number: Byte);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
      Number: Number
   Output:
      Number is written to file F
}
var
   i_sub: IType;
begin
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   write(F,Number);
end;
{-------------------------------------------------------------------}
Procedure Byte_0;
{
Procedure Byte_0(var F: FileByte;
                     row: IType);
}
```

```
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      Number is initialized to file F
}
var
   i_sub: IType;
   X_Byte: Byte;
begin
   X_Byte:=0;
   i_sub:=row-1;     { 0 for the first sub-matrix }
   Seek(F,i_sub);
   write(F,X_Byte);
end;
{-------------------------------------------------------------------}
Procedure get_Fd_Elem;
{
Procedure get_Fd_Elem(    Dim: Itype;
                      var Elem: VectI_Nne2;   var for speed
                      var F_Fd: FileByte;
                      var Fd_Elem: MatrixByte_NnexD);
}
{
   Input:
      Dim: Dimension of problem  =2 for 2-D, =3 for 3-D
      Elem[1..i+2]: defines Material # (i=1),
                            Element Type (i=2),
                            nodes for element (i=3...)
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
   Output:
      Fd_Elem[iNne,iD]: =0 if a force is given
                        =1 ...  displacement
{
                  Look if a force has not been calculated
}
var
   iD,iNne,Nne: Itype;
   Fd: VectByte_D;
begin
   Nne:=Get_Nne(Elem[2]);
   For iNne:=1 to Nne do
      begin
      D_Byte_Read(F_Fd,abs(Elem[2+iNne]),Dim,Fd);
      For iD:=1 to Dim do
         Fd_Elem[iNne,iD]:=Fd[iD];
      end;
end;
{-------------------------------------------------------------------}
end.
```

FILL.PAS

```
{$M 65520,0,0}
Uses
   Dos,Crt,WaitKey;
Type
   String255 = string[255];
{-------------------------------------------------------------------------}
Procedure Help;
begin
   writeln('Disquette filler program              Version 1b');
   writeln('                                      by Jerome Daoust');
   writeln('Format:');
   writeln;
   writeln('FILL [FromDirectory]Filter ToDirectory');
   writeln('   FromDirectory: Directory from which files are copied');
   writeln('   ToDirectory: Directory to which files are copied');
   writeln('      Must start with A: or B:');
   writeln('      You must not copy to and from the same drive.');
   writeln('   Filter: Any DOS filter: *.*  ABC??.pas  *.exe');
   writeln;
   writeln('The program looks if sufficient space is found in ToDirectory');
   writeln('for the next file found in Filter pattern, then copies it.');
   writeln('A message is given when ToDirectory is filled.');
   writeln('Diskettes must be previously formatted.');
   writeln('No files are erased. Copying is done in alphabetical order');
   writeln;
   writeln('example:  C>Fill C:\TP\FE\*.pas A: <Enter>');
   writeln;
end;
{-------------------------------------------------------------------------}
Procedure Get_Parameters(var FromDir,ToDir,Filter: String255);
{
   Input:
      Command line parameters
   Output:
      FromDir: Directory FROM which files are to be copied
      ToDir:  Directory TO which files are to be copied
      Filter: filter for files to be copied
}
var
   i,iPar,ToEnd : Byte;
   CurrentDir: String255;
begin
{
                  Give information when number of parameters <> 2
}
   If ParamCount<>2 then
      begin
      writeln('2 parameters needed:');
      ClrScr;
      Help;
```

```
      Halt;
      end;
{
                  Get Filter and ToDir
}
   FromDir:='';
   Filter:=ParamStr(1);
   ToDir:=ParamStr(2);
{
                  Separate FromDir from Filter
}
   ToEnd:=0;
   For i:=1 to Length(Filter) do
      If Filter[i] in [':','\'] then
         ToEnd:=i;
{
                  Set FromDir to Current Directory if ommited
}
   If 0<ToEnd then
      begin
      FromDir:=Copy(Filter,1,ToEnd);
      Delete(Filter,1,ToEnd);
      end
   else
      begin
      GetDir(0,FromDir);
      If FromDir[Length(FromDir)]<>'\' then
         FromDir:=FromDir+'\';
      end;
{
                  Add drive letter if missing to FromDir
}
   If FromDir[2]<>':' then
      begin
      GetDir(0,CurrentDir);
      FromDir:=Copy(CurrentDir,1,2)+FromDir;
      end;
end;
{-------------------------------------------------------------------------------}
Procedure Fill_Dir(    FromDir,ToDir,Filter: String255);
{
   Input:
      FromDir: Directory FROM which files are to be copied
      ToDir:  Directory TO which files are to be copied
      Filter: filter for files to be copied
   Output:
      Copies files
}
Const
   MaxFiles = 3000;
var
   i,j,iFile,KeepI,FileQtt,FreeSpaceOnDisk: LongInt;
```

```
   FileName: String[255];
   DirInfo: SearchRec;
   ToDriveNumber,AttemptToReplace,AttemptToReplaceMax: Byte;
   Letter: Char;
   NameOfFile: array[1..MaxFiles] of String[12];
   SizeOfFile: array[1..MaxFiles] of LongInt;
begin
{
                  Get drive number to which we will copy
                     A -> 1
                     B -> 2   Must be A: or B:
                  Must not copy to and from same drive
}
   ToDriveNumber:=0;
   If 0<Length(ToDir) then
      If UpCase(ToDir[1]) in ['A'..'B'] then
         ToDriveNumber:=Ord(UpCase(ToDir[1]))-64;
   If ((ToDriveNumber=0)or(UpCase(ToDir[1])=UpCase(FromDir[1]))) then
      begin
      Help;
      Halt;
      end;
{
                  Find files matching filter
}
   FileQtt:=0;
   FindFirst(FromDir+Filter,Archive,DirInfo);
   While DosError=0 do
      begin
      Inc(FileQtt);
      If MaxFiles<FileQtt then
         begin
         writeln('Too many files: over ',MaxFiles);
         Halt;
         end;
      NameOfFile[FileQtt]:=DirInfo.Name;
      SizeOfFile[FileQtt]:=DirInfo.Size;
      FindNext(DirInfo);
      end;
{
                  Sort files by bubble sort
}
   For i:=1 to FileQtt do
      begin
      gotoXY(1,WhereY);
      write(FileQtt-i+1,' ');
      For j:=FileQtt-1 downto 1 do
         If NameOfFile[j]>NameOfFile[j+1] then
            begin
            FileName:=NameOfFile[j];
            NameOfFile[j]:=NameOfFile[j+1];
            NameOfFile[j+1]:=FileName;
```

```
            KeepI:=SizeOfFile[j];
            SizeOfFile[j]:=SizeOfFile[j+1];
            SizeOfFile[j+1]:=KeepI;
            end;
      end;
   AttemptToReplaceMax:=3;
   gotoXY(1,WhereY);
   write('Free space on disk ',chr(64+ToDriveNumber));
   gotoXY(25,WhereY);
   write('File');
   gotoXY(40,WhereY);
   write('    Size');
   gotoXY(50,WhereY);
   writeln;
   For iFile:=1 to FileQtt do
      begin
{
                  Look if sufficient space is available
}
      AttemptToReplace:=0;
      FreeSpaceOnDisk:=DiskFree(ToDriveNumber);
      write(FreeSpaceOnDisk:8);
      gotoXY(25,WhereY);
      write(NameOfFile[iFile]);
      gotoXY(40,WhereY);
      write(SizeOfFile[iFile]:8);
      gotoXY(50,WhereY);
      While (FreeSpaceOnDisk<SizeOfFile[iFile]) and
            (AttemptToReplace<AttemptToReplaceMax) do
         begin
         writeln;
         write('Replace disk ',chr(64+ToDriveNumber),' and press any key');
         If 0<AttemptToReplace then write(' <',AttemptToReplace+1,'>');
         wait;
{
                  Check new free disk space
}
         FreeSpaceOnDisk:=DiskFree(ToDriveNumber);
         Inc(AttemptToReplace);
{
                  Erase Replace message
                  Display new free space
}
         gotoXY(1,WhereY);
         write('                                          ');
         gotoXY(1,WhereY-1);
         write(FreeSpaceOnDisk:8);
         gotoXY(50,WhereY);
         end;
{
                  Stop after too many tries
}
```

```
      If AttemptToReplace=AttemptToReplaceMax then
         begin
         writeln;
         writeln('Still not enough space after ',AttemptToReplaceMax,' tries. GoobBye!');
         Halt;
         end;
(
                  Get File name and copy file
)
      Exec('\Command.com','/c COPY '+FromDir+NameOfF'ˇs[1File]+' '+ToDir);
      end
end;
(=============================================================================)
var
   FromDir,ToDir,Filter: String255;
begin
   Get_Parameters(FromDir,ToDir,Filter);
   Fill_Dir(FromDir,ToDir,Filter);
end.
```

FIXED.PAS

```
{$N+}
Unit Fixed;
Interface
Uses
   Declare,File_RW;
Procedure FixedNode(var F_Fd: FileByte;
                    var Elem: VectI_Nne2;  {var for speed}
                        Nne,Dim: Itype;
                    var Order: VectI_Nne;
                    var Fixed: VectB_Nne);
{-------------------------------------------------------------}
Implementation
Procedure FixedNode;
{
Procedure FixedNode(var F_Fd: FileByte;
                    var Elem: VectI_Nne2;
                        Nne,Dim: Itype;
                    var Order: VectI_Nne;
                    var Fixed: VectB_Nne);
}
{
   Input:
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      Elem[1..Net+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      Nne: number of nodes in element
      Dim: Dimensions for geometry
   Output:
      Order[iNne]: gives the node index to be used in reducing
         When a node is to be reduced within this element and is completely
            fixed in all direction, it is reduced first because it reduces
            quickly.
         The front matrix will be smaller for the following nodes to reduce
           and consequently go faster.
      Fixed[iNne]: True if a node is fixed in all directions
   Note: It is possible for an element to use a node more than once.
      This doesn't cause a problem, even if that node is to be reduced,
      because it set for reduction (negative node number) only once in
      the reduction process. So it will be reduced and not used again
      for reduction, so the process need not be altered.
}
var
   ID,iNne,jNne,node: Itype;
   FixedTemp,Free: array[1..Nne_max] of boolean;
   Fd: VectByte_D;
begin
{
                  Find out which nodes are completely fixed (if any)
```

```
}
   For iNne:=1 to Nne do
      begin
      D_Byte_Read(F_Fd,abs(Elem[2+iNne]),Dim,Fd);
      Fixed[iNne]:=True;
      For iD:=1 to Dim do
         If Fd[iD]=0 then Fixed[iNne]:=False;
      end;
{
                  Copy Fixed values into FixedTemp because they
                     will be changed
                  Get the state of free nodes
}
   For iNne:=1 to Nne do
      begin
      FixedTemp[iNne]:=Fixed[iNne];
      If Fixed[iNne]=True then
         Free[iNne]:=False
      else
         Free[iNne]:=True;
      end;
{
                  Establish order of solving within nodes
                  Put fixed node at the beginning
}
   For iNne:=1 to Nne do
      begin
      Order[iNne]:=0;
      jNne:=1;
      While jNne<=Nne do
         begin
         if FixedTemp[jNne]=True then
            begin
            Order[iNne]:=jNne;
            FixedTemp[jNne]:=False;
            jNne:=Nne;  { exit loop }
            end;
         jNne:=jNne+1;
         end;
      end;
{
                  Complete order with other nodes
}
   For iNne:=1 to Nne do
      begin
      If Order[iNne]=0 then
         begin
         jNne:=1;
         While (jNne<=Nne) do
            begin
            If (Free[jNne]=True) then
               begin
```

```
                Free[jNne]:=False;
                Order[iNne]:=jNne;
                jNne:=Nne;  { exit loop }
                end;
             jNne:=jNne+1;
             end;
          end;
       end;
end;
{-------------------------------------------------------------------}
end.
```

FRONT.PAS

```
{$N+}
Unit Front;
Interface
Uses
   Crt,Declare,Nne_Dim,Linear,File_RW,Sub,Fixed,Dos,WaitKey;
Procedure Front_node_I_Read(var F: FileI;
                                row: IType;
                            var numberI: IType);
Procedure Front_node_I_Write(var F: FileI;
                                 row: IType;
                                 numberI: IType);
Procedure Front_Pos_I_Read(var F: FileI;
                               row: IType;
                           var numberI: IType);
Procedure Front_Pos_I_Write(var F: FileI;
                                row: IType;
                                numberI: IType);
Procedure Front_Place(var Elem: VectI_Nne2;  { var for speed }
                      var N_Front: IType;
                      var F_Front_pos,F_Front_node: FileI;
                      var EFpos,EFindex: VectI_Nne);
Procedure Correct_Front(    iNne,Nne: IType;
                        var N_Front: IType;
                        var F_Front_pos,F_Front_node: FileI;
                        var EFindex: VectI_Nne);
Procedure Check_Size(    Net,Nnt,Dim,NneMax: Itype;
                     var F_Elem: FileI;
                     var F_Fd: FileByte;
                     var FrontNodeSize: IType);
{-------------------------------------------------------------------------}
Implementation
Const
   RAMnodeMax = 1000;
var
   Front_Node_RAM,Front_Pos_RAM: array[1..RAMnodeMax] of Itype; {Itype necessary?}
{-------------------------------------------------------------------------}
Procedure Front_node_I_Read;
{
Procedure Front_node_I_Read(var F: FileI;
                                row: IType;
                            var numberI: IType);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      numberI: Number
}
begin
```

```
{
                First positions are in RAM
}
   If row<=RAMnodeMax then
      NumberI:=Front_node_RAM[row]
   else
      begin
{
                - 1 so that the first item is # 0 in file
                - RAMnodeMax kept in RAM
}
      Seek(F,row-1-RAMnodeMax);
      Read(F,numberI);
      end;
end;
{-------------------------------------------------------------------}
Procedure Front_node_I_Write;
{
Procedure Front_node_I_Write(var F: FileI;
                             row: IType;
                             numberI: IType);
}
{
   Input:
      F: File containing numbers
      row: position of node in Vector
      numberI: Number
   Output:
      NumberI is written to file F
}
begin
{
                First positions are in RAM
}
   If row<=RAMnodeMax then
      Front_node_RAM[row]:=NumberI
   else
      begin
{
                - 1 so that the first item is # 0 in file
                - RAMnodeMax kept in RAM
}
      Seek(F,row-1-RAMnodeMax);
      write(F,numberI);
      end;
end;
{-------------------------------------------------------------------}
Procedure Front_Pos_I_Read;
{
Procedure Front_Pos_I_Read(var F: FileI;
                           row: IType;
                       var numberI: IType);
```

```
)
{
   Input:
      F: File containing numbers
      row: position of node in Vector
   Output:
      numberI: Number
)
begin
{
                  First positions are in RAM
)
   If row<=RAMnodeMax then
      NumberI:=Front_Pos_RAM[row]
   else
      begin
{
                  - 1 so that the first item is # 0 in file
                  - RAMnodeMax kept in RAM
)
      Seek(F,row-1-RAMnodeMax);
      Read(F,numberI);
      end;
end;
{-------------------------------------------------------------------)
Procedure Front_Pos_I_Write;
{
Procedure Front_Pos_I_Write(var F: FileI;
                                row: IType;
                                numberI: IType);
)
{
   Input:
      F: File containing numbers
      row: position of node in Vector
      numberI: Number
   Output:
      NumberI is written to file F
)
begin
{
                  First positions are in RAM
)
   If row<=RAMnodeMax then
      Front_Pos_RAM[row]:=NumberI
   else
      begin
{
                  - 1 so that the first item is # 0 in file
                  - RAMnodeMax kept in RAM
)
      Seek(F,row-1-RAMnodeMax);
```

```
      write(F,numberI);
      end;
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
Function Max_pos(    N_Front: IType;
                  var F_Front_pos: FileI): IType;
{
   Input:
      N_Front: number of active node in front
      F_Front_pos: File with position of working nodes in front
   Output:
      Max_pos: Maximum position used for a node in Front matrix
}
var
   Fpos,i,Max: IType;
begin
   Max:=0;
   For i:=1 to N_Front do
      begin
      Front_pos_I_Read(F_Front_pos,i,Fpos);
      If Max<Fpos then Max:=Fpos;
      end;
   Max_pos:=Max;
end;
{-------------------------------------------------------------------}
Procedure Front_Place;
{
Procedure Front_Place(var Elem: VectI_Nne2;
                      var N_Front: ITypo;
                      var F_Front_pos,F_Front_node: FileI;
                      var EFpos,EFindex: VectI_Nne);
}
{
   Input:
      Elem[1..Net+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      N_Front: number of active node in front
      F_Front_pos: File with position of nodes in front
      F_Front_node: File with node number at Front_pos,
                    negative for nodes used for the last time
   Output:
      N_Front: New number of active node in front
      F_Front_pos: File with position of nodes in Front (more),
      F_Front_node: modified so that new nodes are placed in Front,
                    negative value for nodes used for the last time
      EFpos: Element to Front position (in actual matrix)
             Negative if it is a new position
      EFindex: Element to Front index
   Example:
         i        1  2  3  4  5  6  7  8  9  10 ...
```

```
      Occupied?    y       y   y       y   y
      N_Front=5
      iFront:      1   2   3   4   5   6   7   8 ...
      Front_pos:   1   3   4   6   7
      Front_node:  23  12  45  8   17
      If we add nodes:  8(a place exists)  -6(to be eliminated) and 34
      N_Front=7
      iFront       1   2   3   4   5   6   7   8 ...
      Front_pos    1   3   4   6   7   2   5
      Front_node   23  12  45  8   17 -6   34
}
var
   iNne,Nne,StartPosSpan,MaxPosSpan,iSpan,iFront,iPos,Fpos,Fnode,MaxPos: Itype;
   Used: Boolean;
   Empty: array[1..Span_max] of Boolean;
   AbsNode: VectI_Nne;
begin
{
                  Initialize EFpos  EFindex
}
   Nne:=Get_Nne(Elem[2]);
   For iNne:=1 to Nne do
      begin
      EFpos[iNne]:=0;
      EFindex[iNne]:=0;
      AbsNode[iNne]:=abs(Elem[2+iNne]);
      end;
{
                  Find out if there is already a place with that node #
                  in the Front
}
   For iFront:=1 to N_Front do    ( a WHILE doesn't speed up here )
      begin
      Front_node_I_Read(F_Front_Node,iFront,Fnode);
      For iNne:=1 to Nne do
         begin
         If abs(Fnode)=AbsNode[iNne] then
            begin
            If Elem[2+iNne]<0 then
               Front_node_I_Write(F_Front_Node,iFront,Elem[2+iNne]);
            Front_pos_I_Read(F_Front_Pos,iFront,fpos);
            EFpos[iNne]:=Fpos;                ( remember position )
            EFindex[iNne]:=iFront;            ( remember Front index )
            end;
         end;
      end;
{
                  Try to place node in a free place in the front
                  This method used a window (Span) for storing Positions of
                     the front matrix, then checks if a position is available
}
   MaxPos:=Max_pos(N_Front,F_Front_pos);
```

```
   iSpan:=1;
   While (Lin(iSpan,1,Span_max)<=MaxPos) do
      begin
      StartPosSpan:=Lin(iSpan,1,Span_max);
{
                  Find out necessary span within maximum span
}
      If MaxPos<StartPosSpan+Span_max then
         MaxPosSpan:=MaxPos-StartPosSpan+1
      else
         MaxPosSpan:=Span_max;
{
                  Initialize Empty vector
                  Then fill it up when position is occupied
}
      For iPos:=1 to MaxPosSpan do
         Empty[iPos]:=True;
      For iFront:=1 to N_Front do
         begin
         Front_pos_I_Read(F_Front_pos,iFront,Fpos);
         If (StartPosSpan<=Fpos) and (Fpos<StartPosSpan+Span_max) then
            Empty[Fpos-StartPosSpan+1]:=False;
         end;
{
                  Look for empty positions
}
      For iNne:=1 to Nne do
         begin
         iPos:=1;
         While (iPos<=MaxPosSpan) and (EFpos[iNne]=0) do
            begin
            If Empty[iPos]=True then
               begin
               Empty[iPos]:=False;                { state position as filled }
               Front_pos_I_Write(F_Front_pos,N_Front+1,iPos+StartPosSpan-1);
               Front_node_I_Write(F_Front_node,N_Front+1,Elem[2+iNne]);
               EFpos[iNne]:=-(iPos+StartPosSpan-1); { remember position }
                                                  {   negative for new one }
               EFindex[iNne]:=N_Front+1;          { remember Front index }
               Inc(N_Front);
               end;
            Inc(iPos);
            end;
         end;
      Inc(iSpan);
      end;
{
                  If not placed yet and no free space in between
                  --> place at the end
}
   For iNne:=1 to Nne do
      If EFpos[iNne]=0 then
```

```
        begin
        Front_pos_I_Write(F_Front_pos,N_Front+1,MaxPos+1);
        Front_node_I_Write(F_Front_node,N_Front+1,Elem[2+iNne]);
        EFpos[iNne]:=-(MaxPos+1);       { remember position }
                                        {   negative for new one }
        EFindex[iNne]:=N_Front+1;   { remember Front index }
        Inc(MaxPos);
        Inc(N_Front);
        end;
end;
{-------------------------------------------------------------------}
Procedure Correct_Front;
{
Procedure Correct_Front(    iNne,Nne: IType;
                        var N_Front: IType;
                        var F_Front_pos,F_Front_node: FileI;
                        var EFindex: VectI_Nne);
}
{
   Input:
      iNne: Element index, where Front_pos[EFindex[iNne]
            and Front_node[EFindex[iNne]] are to be removed
      Nne: number of nodes in the element
      N_Front: number of active node in front
      F_Front_pos: File with position of working nodes in front
      F_Front_node: File with node number at Front_pos,
                    negative for nodes used for the last time
      EFindex: Element to Front index
   Ouput:
      N_Front,F_Front_pos,F_front_node: modified for removal of a node
      EFindex: Element to Front index, corrected
}
var
   jNne,RiFront,Fpos,Fnode,i: IType;
begin
   RiFront:=EFindex[iNne];
{
                  Correct N_Front, Front_pos, Front_node
                  Take last element of Front_pos and Front_node
                     and put it in position to be removed
}
   Front_pos_I_Read(F_Front_pos,N_Front,Fpos);
   Front_pos_I_Write(F_Front_pos,RiFront,Fpos);
   Front_node_I_Read(F_Front_node,N_Front,Fnode);
   Front_node_I_Write(F_Front_node,RiFront,Fnode);
{
                  Correct EFindex
}
   For jNne:=1 to Nne do
      If EFindex[jNne]=N_Front then
         EFindex[jNne]:=RiFront;
   Dec(N_Front);
```

```
end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
Procedure Front_Size(    Net,NneMax,Dim: IType;
                     var F_Elem: FileI;
                     var F_Fd: FileByte;
                     var FrontNodeSize,ReverseFileSize,EquationFileSize: IType);
{
   Input:
      Net: Number of elements;
      NneMax: Maximum number of nodes in an element (found)
      Dim: Dimension of problem
      F_Elem: Element[1..Net,i]: defines Material # (i=1),
                                 Element Type (i=2),
                                 nodes for each element (i=3...)
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
   Output:
      FrontNodeSize: Maximum Front size (in nodes) that the front Matrix will reach
      ReverseFileSize: Size in bytes of file containing reverse node numbers
         used in back-substitution
         (If all submatrices used are non-zero -> maximum)
      EquationFileSize: Size in bytes of file containing equations
         used in back-substitution
         (If all submatrices used are non-zero -> maximum)
}
var
   RiNne,iNne,iD,Rnode,Nne,N_Front,iE: IType;
   F_Front_pos,F_Front_node: FileI;
   Elem: VectI_Nne2;
   Order,EFindex,EFpos: VectI_Nne;
   Xs,Ys: Byte;  (screen positions)
   X_Rtype: Rtype;
   Isize,Rsize: Byte;
   SubVectorSize,SubMatrixSize: word;
   AllFixed,StoreElem: Boolean;
   Fd_Elem: MatrixByte_NnexD;
   Fixed: VectB_Nne;
begin
{
   What is stored:
      Equations: When a node is to be reduced, and in one of the directions
                 we have a unknown displacement, we store all submatrices
                 in the node front matrix (except null submatrices, but not
                 taken in account here) and constants.
      Reverse: we store all node numbers used in the previous
               plus quantity of nodes used
      Equations: when we have an unknown force we store the pertaining row of
                 stiffness matrix
      Reverse: when we have an unknown force, the element's number is stored
               as a negative value
}
```

```
   Write('Predicting Front Matrix Size ');
   Xs:=WhereX;
   Ys:=WhereY;
   N_Front:=0;
   FrontNodeSize:=0;
   ReverseFileSize:=0;
   EquationFileSize:=0;
   Isize:=SizeOf(FrontNodeSize);   {size of integers used in reverse file}
   Rsize:=SizeOf(X_Rtype);           {size of real used for storing}
   SubMatrixSize:=Dim*Dim*Rsize;  {size in bytes of a submatrix}
   SubVectorSize:=Dim*Rsize;      {size of a subvector for constants
                                     in equations of Dim nodes}
   Assign(F_Front_Pos,T_Dir+'Fpos.T');
   Assign(F_Front_node,T_Dir+'Fnode.T');
   Rewrite(F_Front_Pos);
   Rewrite(F_Front_node);
   For iE:=1 to Net do
      begin
      gotoXY(Xs,Ys);
      write(Net-iE+1,' ');
      N:=2_I_Read(F_Elem,iE,NneMax,Elem);
      Nne:=Get_Nne(Elem[2]);
{
                update the node # used in the front (Front_pos,
                Front_node,N_Front), see the subroutine's explanations
                Estabish correspondance between order of the nodes in
                the element and where this node is placed in the Front
}
      Front_Place(Elem,N_Front,F_Front_pos,F_Front_node,EFpos,EFindex);
{
                Retain maximum size of Front Matrix
}
      If FrontNodeSize<N_Front then FrontNodeSize:=N_Front;
{
                Look for fixed nodes
                Store needed row in the element's stiffness matrix
                Only for nodes used for last time (negative)
}
      StoreElem:=False;
      get_Fd_Elem(Dim,Elem,F_Fd,Fd_Elem);
      For iNne:=1 to Nne do
         begin
         For iD:=1 to Dim do
            If (Fd_Elem[iNne,iD]=1) and (Elem[2+iNne]<0) then
               begin
               EquationFileSize := EquationFileSize
                                   + Nne*Dim*Rsize;
               StoreElem:=True;
               end;
         end;
      If StoreElem=True then
         ReverseFileSize := ReverseFileSize + Isize;
```

```
{
                Correct Front nodes and Position
                Increase EquationFileSize
}
      FixedNode(F_Fd,Elem,Nne,Dim,Order,Fixed);
      For iNne:=1 to Nne do
         begin
         RiNne:=Order[iNne];
         Rnode:=Elem[2+RiNne];
{
                Look if a node is to be reduced and not all fixed
}
         If Rnode<0 then
            begin
            If Fixed[RiNne]=False then   {only save nodes not completely fixed}
               begin
               ReverseFileSize := ReverseFileSize
                               + (N_Front+1)*Isize;
               EquationFileSize := EquationFileSize
                                + N_Front*SubMatrixSize
                                + SubVectorSize;
               end;
            Correct_Front(RiNne,Nne,N_Front,F_Front_pos,F_Front_node,EFindex);
            end;
         end;
      end;
   gotoXY(Xs,Ys);
   writeln('(',FrontNodeSize,' node wave front)');
   Close(F_Front_Pos);
   Close(F_Front_node);
   Erase(F_Front_Pos);
   Erase(F_Front_node);
end;
{-------------------------------------------------------------------}
Procedure Check_Size;
{
Procedure Check_Size(    Net,Nnt,Dim,NneMax: Itype;
                     var F_Elem: FileI;
                     var F_Fd: FileByte;
                     var FrontNodeSize: IType);
}
{
   Input:
      Net: total number of elements
      Nnt: total number of nodes
      Dim: dimension of problem
      NneMax: Maximum number of nodes in an element (found)
      F_Elem: Element[1..Net,i]: defines Material # (i=1),
                                 Element Type (i=2),
                                 nodes for each element (i=3...)
   Output:
      FrontNodeSize: Maximum Front size (in nodes) that the front Matrix will reach
```

```
      Message is displayed concerning available disk space
}
var
   CurrentDir: String255;
   Connect,Nstress: Itype;
   T,F,N: LongInt;
   Isize,Rsize,Bsize: Byte;
   X_Rtype: Rtype;
   ReverseFileSize,EquationFileSize: Itype;
begin
   Front_Size(Net,NneMax,Dim,F_Elem,F_Fd,
              FrontNodeSize,ReverseFileSize,EquationFileSize);
{
                  Initialize Heap memory for submatrices
}
   InitSubHeapRAM(Dim,FrontNodeSize);
   Connect:=ConnectMax;
   Nstress:=N_Stress(Dim);
{
                  Get byte size of:  Integers Reals Byte
}
   Isize:=SizeOf(Itype);
   Rsize:=SizeOf(Rtype);
   Bsize:=SizeOf(Byte);
{
                  Taken from \Lotus\Files\Kbyte.wk1
                  here Nnf = FrontNodeSize (Number of Nodes in Front)
                       Nne = NneMax
T          Front R,Dim^2.(3/2.Nnf+Nnf^2/2)  (Exact)
T             CF R,Nnf.Dim                  (Exact)
T      Rev (max) I,Nnt.Nnf                  (Max, can be much less)
T       Eq (max) R,Nnt.Nnf.D                (Max, can be much less)
T       FrontPos I,Nnf-RAMnodeMax           (Exact)
T      FrontNode I,Nnf-RAMnodeMax           (Exact)
            Elem R,(Nne+2)*N                (Exact)
           Force R,Nnt*Dim                  (Exact)
           Displ R,Nnt*Dim                  (Exact)
           Angle R,Net                      (Exact)
             XYZ R,(Nnt*Dim)                (Exact)
N         LStrain R,Nnt*Nstre               (Exact)
N         LStress R,Nnt*Nstre               (Exact)
N         GStrain R,Nnt*Nstre               (Exact)
N         GStress R,Nnt*Nstre               (Exact)
           NewOld I,Nnt                     (Exact)
           OldNew I,NntOld                  (Exact)
               Fd B,Nnt*Dim                 (Exact)
            Used B,Nnt                      (Exact)
            Node I,Nnt+6                    (Exact)
         Connect I,(Nnt+6)*C                (Exact)
                  Here: T for temporary in SOLVE process
                        N for New
                          Sum of T's is 100 times sum of N
```

```
                        and occur separately --> not considered when
                        these files are saved in same place as T files
                      F for free space on disk used
}
  T:=0;
{
                Use exact number of submatrices not kept in RAM
                calculated in InitSubHeapRAM procedure
}
  T:=T+(SubNumberMax-SubNumberMaxRAM)*Rsize*Dim*Dim;
  T:=T+Rsize*(FrontNodeSize*Dim);
  T:=T+ReverseFileSize;
  T:=T+EquationFileSize;
  If RAMnodeMax<FrontNodeSize then
    T:=T+2*Isize*(FrontNodeSize-RAMnodeMax);
  N:=4*Rsize*(Nnt*Nstress);
  If C_Dir=T_Dir then
    begin
    write('*.T files: ');
    TextColor(LightRed);
    write(T/sqr(1024.0):9:4);
    TextColor(LightGreen);
    write(' MB neaded (max), ');
    GetDir(0,CurrentDir);
    ChDir(T_Dir);
    F:=DiskFree(0);
    ChDir(CurrentDir);
    writeln(F/sqr(1024.0):7:2,' MB available');
    If F<T then
      begin
      writeln('Error!');
      wait;
      Halt;
      end;
    end
  else
    begin
    write('*.T files: ');
    TextColor(LightRed);
    write(T/sqr(1024.0):9:4);
    TextColor(LightGreen);
    write(' MB neaded (max), ');
    GetDir(0,CurrentDir);
    ChDir(T_Dir);
    F:=DiskFree(0);
    ChDir(CurrentDir);
    writeln(F/sqr(1024.0):7:2,' MB available');
    If F<T then
      begin
      writeln('Error!');
      wait;
      Halt;
```

```
      end;
    write('New *.P files: ');
    TextColor(LightRed);
    write(N/sqr(1024.0):9:4);
    TextColor(LightGreen);
    write(' MB neaded (max), ');
    GetDir(0,CurrentDir);
    ChDir(C_Dir);
    F:=DiskFree(0);
    ChDir(CurrentDir);
    writeln(F/sqr(1024.0):7:2,' MB available');
    If F<N then
      begin
      writeln('Error!');
      wait;
      Halt;
      end;
    end;
end;
{-------------------------------------------------------------}
end.
```

GET.PAS

```
{$N+}
Unit Get;
Interface
Uses
   Declare,Put;
Procedure Get_I(    Name: String255;
                var NumberI: IType);
Procedure Get_Mat(var Mat: MatrixR_Nmatx6);
Procedure Get_MatLabel(var MatLabel: VectS80_Nmat);
Procedure Get_MaxStress(var MaxStress: MatrixR_Nmatx9);
{-------------------------------------------------------------------}
Implementation
Procedure Get_I;
{
Procedure Get_I(    Name: String255;
                var NumberI: IType);
}
{
   Input:
      Name: File name with extension
      NumberI: IType Number
   Output:
      Reads NumberI from File "Name"
}
var
   F: FileI;
begin
   Assign(F,C_Dir+Name);
   {$I-}
   Reset(F);
   {$I-}
   If IoResult=0 then    {read information when file exists}
      begin
      Read(F,NumberI);
      Close(F);
      end
   else
      NumberI:=0;
end;
{-------------------------------------------------------------------}
Procedure InstallMaterial;
{
   Output:
      Mat[1..Nmat,i]: El, Et, Glt, Gtt, nlt, ntt
      MaxStress[iMaterial,j]: iMaterial: Database material number
                              j =1: X
                                =2: X'
                                =3: Y
                                =4: Y'
                                =5: Z
```

```
                                =6: Z'
                                =7: Sxy max
                                =8: Sxz max
                                =9: Syz max
      MatLabel[iMaterial]: Name of material, details
}
Var
   i,j: integer;
   R_Pos: integer;
   Mat: MatrixR_Nmatx6;
   MaxStress: MatrixR_Nmatx9;
   MatLabel: VectS80_Nmat;
begin
{
                  Initialize all materials
}
   For i:=1 to Nmat_max do
      begin
      MatLabel[i]:='';
      For j:=1 to 6 do
         Mat[i,j]:=0.0;
      For j:=1 to 9 do
         MaxStress[i,j]:=BigR;
      end;
{
                  Give: Label (no commas)
                        El, Et, Glt, Gtt, nlt, ntt
                        +S1,-S1,+S2,-S2,+S3,-S3,S12,S13,S23 maximum
}
   MatLabel[1]:='CE 9000 Glass/Epoxy';
   Mat[1,1]:=47.4E9;
   Mat[1,2]:=16.2E9;
   Mat[1,3]:=7.0E9;
   Mat[1,4]:=1.28E9;   {Epoxy}
   Mat[1,5]:=0.26;     {Scotchply 1002}
   Mat[1,6]:=0.35;     {Epoxy}
   MaxStress[1,1]:=1283E6;
   MaxStress[1,2]:=725E6;
   MaxStress[1,3]:=27.0E6;
   MaxStress[1,4]:=179E6;
   MaxStress[1,5]:=27.0E6;
   MaxStress[1,6]:=179E6;
   MaxStress[1,7]:=56.1E6;
   MaxStress[1,8]:=46.6E6;
   MaxStress[1,9]:=72.4E6/2;   {Epoxy}
   MatLabel[2]:='Epoxy: Anhydride-Cured';
   Mat[2,1]:=3.45E9;
   Mat[2,2]:=3.45E9;
   Mat[2,3]:=3.45E9/2/(1+0.35);  {=1.28E9 isotropic}
   Mat[2,4]:=3.45E9/2/(1+0.35);
   Mat[2,5]:=0.35;
   Mat[2,6]:=0.35;
```

```
MaxStress[2,1]:=72.4E6;
MaxStress[2,2]:=72.4E6;
MaxStress[2,3]:=72.4E6;
MaxStress[2,4]:=72.4E6;
MaxStress[2,5]:=72.4E6;
MaxStress[2,6]:=72.4E6;
MaxStress[2,7]:=72.4E6/2;
MaxStress[2,8]:=72.4E6/2;
MaxStress[2,9]:=72.4E6/2;
MatLabel[3]:='T300/5208 Graphite/Epoxy';
Mat[3,1]:=181E9;
Mat[3,2]:=10.3E9;
Mat[3,3]:=7.17E9;
Mat[3,4]:=1.28E9;    (Epoxy)
Mat[3,5]:=0.28;
Mat[3,6]:=0.35;
MaxStress[3,1]:=1500E6;
MaxStress[3,2]:=1500E6;
MaxStress[3,3]:=40E6;
MaxStress[3,4]:=246E6;
MaxStress[3,5]:=40E6;
MaxStress[3,6]:=246E6;
MaxStress[3,7]:=68E6;
MaxStress[3,8]:=68E6;
MaxStress[3,9]:=68E6;
MatLabel[4]:='B(4)/5505 Boron/Epoxy';
Mat[4,1]:=204E9;
Mat[4,2]:=18.5E9;
Mat[4,3]:=5.59E9;
Mat[4,4]:=1.28E9;    (Epoxy)
Mat[4,5]:=0.23;
Mat[4,6]:=0.35;
MaxStress[4,1]:=1260E6;
MaxStress[4,2]:=2500E6;
MaxStress[4,3]:=61E6;
MaxStress[4,4]:=202E6;
MaxStress[4,5]:=61E6;
MaxStress[4,6]:=202E6;
MaxStress[4,7]:=67E6;
MaxStress[4,8]:=67E6;
MaxStress[4,9]:=67E6;
MatLabel[5]:='AS/3501 Graphite/Epoxy';
Mat[5,1]:=138E9;
Mat[5,2]:=8.96E9;
Mat[5,3]:=7.1E9;
Mat[5,4]:=1.28E9;    (Epoxy)
Mat[5,5]:=0.3;
Mat[5,6]:=0.35;
MaxStress[5,1]:=1447E6;
MaxStress[5,2]:=1447E6;
MaxStress[5,3]:=51.7E6;
MaxStress[5,4]:=206E6;
```

```
MaxStress[5,5]:=51.7E6;
MaxStress[5,6]:=206E6;
MaxStress[5,7]:=93E6;
MaxStress[5,8]:=93E6;
MaxStress[5,9]:=93E6;
MatLabel[6]:='ScotchPly 1002 Glass/Epoxy';
Mat[6,1]:=38.6E9;
Mat[6,2]:=8.27E9;
Mat[6,3]:=4.14E9;
Mat[6,4]:=1.28E9;   {Epoxy}
Mat[6,5]:=0.26;
Mat[6,6]:=0.35;
MaxStress[6,1]:=1062E6;
MaxStress[6,2]:=610E6;
MaxStress[6,3]:=31E6;
MaxStress[6,4]:=118E6;
MaxStress[6,5]:=31E6;
MaxStress[6,6]:=118E6;
MaxStress[6,7]:=72E6;
MaxStress[6,8]:=72E6;
MaxStress[6,9]:=72E6;
MatLabel[7]:='Kevlar 49/Epoxy Aramid/Epoxy';
Mat[7,1]:=76E9;
Mat[7,2]:=5.5E9;
Mat[7,3]:=2.3E9;
Mat[7,4]:=1.28E9;   {Epoxy}
Mat[7,5]:=0.34;
Mat[7,6]:=0.35;
MaxStress[7,1]:=1400E6;
MaxStress[7,2]:=235E6;
MaxStress[7,3]:=12E6;
MaxStress[7,4]:=53E6;
MaxStress[7,5]:=12E6;
MaxStress[7,6]:=53E6;
MaxStress[7,7]:=34E6;
MaxStress[7,8]:=34E6;
MaxStress[7,9]:=34E6;
MatLabel[8]:='Aluminum';
Mat[8,1]:=71E9;
Mat[8,2]:=71E9;
Mat[8,3]:=26.2E9;
Mat[8,4]:=26.2E9;
Mat[8,5]:=0.334;
Mat[8,6]:=0.334;
MaxStress[8,1]:=400E6;
MaxStress[8,2]:=400E6;
MaxStress[8,3]:=400E6;
MaxStress[8,4]:=400E6;
MaxStress[8,5]:=400E6;
MaxStress[8,6]:=400E6;
MaxStress[8,7]:=230E6;
MaxStress[8,8]:=230E6;
```

```
MaxStress[8,9]:=230E6;
MatLabel[9]:='Carbon Steel (G10180 HotRolled)';
Mat[9,1]:=207E9;
Mat[9,2]:=207E9;
Mat[9,3]:=79.3E9;
Mat[9,4]:=79.3E9;
Mat[9,5]:=0.292;
Mat[9,6]:=0.292;
MaxStress[9,1]:=220E6;
MaxStress[9,2]:=220E6;
MaxStress[9,3]:=220E6;
MaxStress[9,4]:=220E6;
MaxStress[9,5]:=220E6;
MaxStress[9,6]:=220E6;
MaxStress[9,7]:=110E6;
MaxStress[9,8]:=110E6;
MaxStress[9,9]:=110E6;
MatLabel[10]:='Stainless Steel (S30300)';
Mat[10,1]:=190E9;
Mat[10,2]:=190E9;
Mat[10,3]:=73.1E9;
Mat[10,4]:=73.1E9;
Mat[10,5]:=0.305;
Mat[10,6]:=0.305;
MaxStress[10,1]:=240E6;
MaxStress[10,2]:=240E6;
MaxStress[10,3]:=240E6;
MaxStress[10,4]:=240E6;
MaxStress[10,5]:=240E6;
MaxStress[10,6]:=240E6;
MaxStress[10,7]:=120E6;
MaxStress[10,8]:=120E6;
MaxStress[10,9]:=120E6;
MatLabel[11]:='Copper (C21000)';
Mat[11,1]:=119E9;
Mat[11,2]:=119E9;
Mat[11,3]:=44.7E9;
Mat[11,4]:=44.7E9;
Mat[11,5]:=0.326;
Mat[11,6]:=0.326;
MaxStress[11,1]:=70E6;
MaxStress[11,2]:=70E6;
MaxStress[11,3]:=70E6;
MaxStress[11,4]:=70E6;
MaxStress[11,5]:=70E6;
MaxStress[11,6]:=70E6;
MaxStress[11,7]:=35E6;
MaxStress[11,8]:=35E6;
MaxStress[11,9]:=35E6;
MatLabel[26]:='Exercise Steel 1';
Mat[26,1]:=1E9;
Mat[26,2]:=1E9;
```

```
  Mat[26,3]:=1E9/2/(1+0.3);
  Mat[26,4]:=1E9/2/(1+0.3);
  Mat[26,5]:=0.3;
  Mat[26,6]:=0.3;
  MaxStress[26,1]:=220E6;
  MaxStress[26,2]:=220E6;
  MaxStress[26 3]:=220E6;
  MaxStress[26,4]:=220E6;
  MaxStress[26,5]:=220E6;
  MaxStress[26,6]:=220E6;
  MaxStress[26,7]:=110E6;
  MaxStress[26,8]:=110E6;
  MaxStress[26,9]:=110E6;
  MatLabel[27]:='Exercise Steel Polytechnique';
  Mat[27,1]:=1E9;
  Mat[27,2]:=1E9;
  Mat[27,3]:=1E9/2/(1+0.25);
  Mat[27,4]:=1E9/2/(1+0.25);
  Mat[27,5]:=0.25;
  Mat[27,6]:=0.25;
  MaxStress[27,1]:=220E6;
  MaxStress[27,2]:=220E6;
  MaxStress[27,3]:=220E6;
  MaxStress[27,4]:=220E6;
  MaxStress[27,5]:=220E6;
  MaxStress[27,6]:=220E6;
  MaxStress[27,7]:=110E6;
  MaxStress[27,8]:=110E6;
  MaxStress[27,9]:=110E6;
  R_pos:=Ord('R')-Ord('A')+1 +25;  {=43}
  MatLabel[R_pos]:='Rigid 1000';
  Mat[R_pos,1]:=2E14;
  Mat[R_pos,2]:=2E14;
  Mat[R_pos,3]:=2E14/2/(1+0.28);
  Mat[R_pos,4]:=2E14/2/(1+0.28);
  Mat[R_pos,5]:=0.28;
  Mat[R_pos,6]:=0.28;
  MaxStress[R_pos,1]:=1E30;
  MaxStress[R_pos,2]:=1E30;
  MaxStress[R_pos,3]:=1E30;
  MaxStress[R_pos,4]:=1E30;
  MaxStress[R_pos,5]:=1E30;
  MaxStress[R_pos,6]:=1E30;
  MaxStress[R_pos,7]:=1E30;
  MaxStress[R_pos,8]:=1E30;
  MaxStress[R_pos,9]:=1E30;
  Put_Mat(Mat);
  Put_MaxStress(MaxStress);
  Put_MatLabel(MatLabel);
end;
{-------------------------------------------------------------------------}
Procedure Get_Mat;
```

```
{
Procedure Get_Mat(var Mat: MatrixR_Nmatx6);
}
{
   Input:
      Reads Mat from File Mat.dat
   Output:
      Mat[1..Nmat,i]: El, Et, Glt, Gtt, nlt, ntt
                      El=-1 if undefined
}
var
   F: File of MatrixR_Nmatx6;
begin
   Assign(F,'Mat.M');
   {$I-}
   Reset(F);
   {$I-}
   If IoResult<>0 then
      begin
      InstallMaterial;
      Reset(F);
      Read(F,Mat);
      Close(F);
      end
   else
      begin
      Read(F,Mat);
      Close(F);
      end;
end;
{-------------------------------------------------------------------}
Procedure Get_MatLabel;
{
Procedure Get_MatLabel(var MatLabel: VectS80_Nmat);
}
{
   Output:
      MatLabel[iMaterial]: Name of material, details
}
var
   F: File of VectS80_Nmat;
begin
   Assign(F,'MatLabel.M');
   {$I-}
   Reset(F);
   {$I-}
   If IoResult<>0 then
      begin
      InstallMaterial;
      Reset(F);
      Read(F,MatLabel);
      Close(F);
```

```
      end
   else
      begin
      Read(F,MatLabel);
      Close(F);
      end;
end;
{-------------------------------------------------------------------}
Procedure Get_MaxStress;
{
Procedure Get_MaxStress(var MaxStress: MatrixR_Nmatx9);
}
{
   Output:
      MaxStress[iMaterial,j]: iMaterial: Database material number
                              j =1: X
                                =2: X'
                                =3: Y
                                =4: Y'
                                =5: Z
                                =6: Z'
                                =7: Sxy max
                                =8: Sxz max
                                =9: Syz max
}
var
   F: File of MatrixR_Nmatx9;
begin
   Assign(F,'MaxStress.M');
   {$I-}
   Reset(F);
   {$I-}
   If IoResult<>0 then
      begin
      InstallMaterial;
      Reset(F);
      Read(F,MaxStress);
      Close(F);
      end
   else
      begin
      Read(F,MaxStress);
      Close(F);
      end;
end;
{-------------------------------------------------------------------}
end.
```

HELP.PAS

```
{-------------------------------------------------------------------------
   Help Program
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 20000,0,10500}     {max memory used for drawing elements}
                       {10500 bytes in Heap needed for InitGraph}
Uses
   Crt,Graph,Declare,WaitKey,WhatKey,Sonore,H_Common,H_Featur,H_Lang,H_Batch,
   H_Main,H_Etype;
{-------------------------------------------------------------------------}
Procedure Require(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Requirements\';
   Repeat
      Header(HelpPath);
      writeln('Minimal requirements:');
      writeln('   Any IBM PS/2 (TM), IBM (R), or 100% compatible.');
      writeln('   Any math coprocessor: 8087/80287/80387.');
      writeln('   1 hard disk drive, 1 floppy disk drive.');
      writeln('   Turbo Pascal 4.0 (TM) for its graphic drivers *.BGI.');
      writeln('   Any DOS.');
      writeln('   Any graphic card.');
      writeln('   256 kB of memory.');
      writeln('   A word processor (Turbo Pascal''s (TM) can do).');
      writeln('Suggested improvements:');
      writeln('   Enough memory for a RAM drive or caching program.');
      writeln('   Printer for hard copies using PrintScreen');
      writeln('   High speed 40 MB hard disk or larger.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Sequence(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Sequence\';
   Repeat
      Header(HelpPath);
```

```
      writeln('Typical sequence of events:');
      writeln('You have created a directory for DIRECT named \Direct on your hard disk.');
      writeln('You have copied all files to \Direct.');
      writeln('Copy all *.BGI files from Turbo Pascal 4.0 (TM) to \Direct.');
      writeln('You type DIRECT <Enter> from that directory.');
      writeln('Use HELP to learn more about the program and it''s language (see \Language\)');
      writeln('Exit and set your CONFIG.SYS file in the root directory so that you have:');
      writeln('   Files=20   ');
      writeln('   Buffers=17 ');
      writeln('Use CONFIGURE to specify your setup. Use MATERIAL for material definition.');
      writeln('Write your own geometry file with EDIT.');
      writeln('Use PREPROCESS to debug and compile that file.');
      writeln('Use NUMBERS for some data on the case.');
      writeln('Use VIEW to do just that, and verify your geometry.');
      writeln('Use SOLVE to solve the problem at hand. STRESS will calculate stresses.');
      writeln('Use WORST to find safety factor and failure location with mode.');
      writeln('Use DEFORM to see the deformed body shape.');
      writeln('Use RESULT to get a handle on the results.');
      writeln('Use MANAGER to store solution and selection menus.');
      writeln('Use EXIT if tired or successful.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure FileType(HelpPath: String255);
\ar
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Files\';
   Repeat
      Header(HelpPath);
      writeln('Permanent files (*.P): those used for the geometry, solution and set-up.');
      writeln('   These files will be erased next time someone uses PREPROCESS.');
      writeln('   It is a good idea to store them elsewhere after SOLVE is done.');
      writeln('Configuration files (*.D): those used for DIRECT configuration.');
      writeln('Material files (*.M): those used for material data.');
      writeln('Generation files (*.GEN): those you write to generate geometry.');
        write('Temporary files (*.T): those used during the solution process or node selection.')
;
      writeln('   They are automatically erased after the child process is over.');
      writeln('   If ever you see one, you can delete it.');
      writeln('Program files (*.EXE): erase these as an excuse to get a newer version.');
      writeln('Driver files (*.BGI): Drivers used by this program for graphics.');
      writeln('   Available from Turbo Pascal 4.0 (TM).');
      writeln('Backup files (*.BAK): backup files from your word processor, if applicable.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
```

```
{-------------------------------------------------------------------------}
var
   HelpPath: String255;
   C: char;
   Action: String6;
begin
   ClrScr;
   HelpPath:='\';
   TextColor(LightGreen);
   Repeat
      Header(HelpPath);
      writeln('H- Help on HELP');
      writeln('F- Features');
      writeln('R- Requirements');
      writeln('S- Sequence of sample session');
      writeln('L- Language for geometry generation');
      writeln('E- Elements');
      writeln('M- Main menu');
      writeln('T- File types');
      writeln('B- Batch mode');
      writeln('<Esc>- Exit HELP');
      Get_Key(C,Action); C:=UpCase(C);
      If C='F' then Features(HelpPath);
      If C='L' then Language(HelpPath);
      If C='E' then Element(HelpPath);
      If C='R' then Require(HelpPath);
      If C='M' then MainMenu(HelpPath);
      If C='S' then Sequence(HelpPath);
      If C='T' then FileType(HelpPath);
      If C='H' then MainHelp(HelpPath);
      If C='B' then BatchMode(HelpPath);
      until Action='ESC';
   Beep;
   ClrScr;
end.
```

H_BATCH.PAS

```
{$N+}
Unit H_Batch;
Interface
Uses
   Declare,Sonore,H_Common;
Procedure BatchMode(HelpPath: String255);
{-------------------------------------------------------------------------}
Implementation
Procedure Batch_ADDTOEND(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'ADDTOEND\';
   Repeat
      Header(HelpPath);
      writeln('ADDTOEND CaseName AnyCommand');
      writeln('   CaseName: file name of case without ".GEN" extension.');
      writeln('   AnyCommand: designates any command found in \Language\');
      writeln('   The command "AnyCommand" will be appended to CaseName.GEN.');
      writeln('   ex: AddToEnd Example Edel 2;');
      writeln('       -> Example.GEN is the generation file.');
      writeln('          element 2 will be deleted from the geometry.');
      writeln('ADDTOEND CaseName RemoveWorstElement');
      writeln('   RemoveWorstElement: this parameter will produce the appending');
      writeln('      of "EDEL ElementNumber;" where the element number will be');
      writeln('      the worst element found with the WORST process.');
      writeln('   ex: AddToEnd Example RemoveWorstElement');
      writeln('       The worst element found by the WORST process (2) is');
      writeln('       removed from the geometry defined in example.GEN by');
      writeln('       appending: "Edel 2;"');
      writeln('Note: It is recommended to create a new generation file (*.GEN)');
      writeln('   from the current one with the COPY command (from DOS) to');
      writeln('   avoid modifying the original generation file.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_CONFIG(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'CONFIG\';
```

```
   Repeat
      Header(HelpPath);
      writeln('CONFIG ChangedItem [NewString]');
      writeln('   ChangedItem: "CurrentCaseDir"');
      writeln('              or "SaveRootDir"');
      writeln('              or "GenerationDir"');
      writeln('              or "TemporaryDir"');
      writeln('              or "WpDir"');
      writeln('              or "WpName".');
      writeln('   NewString: New information corresponding to "ChangedItem".');
      writeln('      If NewString is omitted, "ChangedItem" information is erased.');
      writeln('Ex:');
      writeln('   CONFIG GenerationDir C:\Direct\Gen\');
      writeln('   --> The generation directory becomes C:\Direct\Gen\');
      writeln;
      writeln('For more information see the Help screen in the CONFIGURE process.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
Procedure Batch_COPYGEN(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'COPYGEN\';
   Repeat
      Header(HelpPath);
      writeln('COPYGEN FromCaseName ToCaseName');
      writeln('   FromCaseName: Name of case without ".GEN" extension');
      writeln('                 FROM which we copy.');
      writeln('   ToCaseName:   Name of case without ".GEN" extension');
      writeln('                 TO which we copy.');
      writeln('Note: this procedure reduces directory confusion for the used');
      writeln('   ex: CopyGen OldCase NewCase');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
Procedure Batch_DELSTRES(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'DELSTRES\';
   Repeat
```

```
      Header(HelpPath);
      writeln('DELSTRESS [FiberStrain] [FiberStress] [ElemStrain] [ElemStress]');
      writeln('          [GlobalStrain] [GlobalStress]');
      writeln('   FiberStrain:  For deleting strain in the material direction.');
      writeln('   FiberStress:  For deleting stress in the material direction.');
      writeln('   ElemStrain:   For deleting strain in the element  direction.');
      writeln('   ElemStress:   For deleting stress in the element  direction.');
      writeln('   GlobalStrain: For deleting strain in the global   direction.');
      writeln('   GlobalStress: For deleting stress in the global   direction.');
      writeln('   Parameters can be simultaneous and interchanged.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_MANAGER(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'MANAGER\';
   Repeat
      Header(HelpPath);
      writeln('MANAGER Save SubDirectoryName');
      writeln('   SubDirectoryName: sub-directory of the "Current Case"');
      writeln('         directory, see CONFIGURE process.');
      writeln('      When omitted, the current case name is used.');
      writeln('   ex: Manager Save Example');
      writeln('      current case will be saved in sub-directory Example');
      writeln('MANAGER Keep');
      writeln('   This command will keep all the files which are used when');
      writeln('      the current geometry is used to generate another one.');
      writeln('   The current case name will appear as the coarse case name.');
      writeln('   See \Main menu\Status lines\.');
      writeln('MANAGER Restore SubDirectoryName');
      writeln('   This command is the reverse of MANAGER Save SubDirectoryName.');
      writeln('   Previously saved information is restored for further use.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_NEWNAME(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'NEWNAME\';
```

```
   Repeat
      Header(HelpPath);
      writeln('NEWNAME NewCaseName');
      writeln('   NewCaseName: New name of case without ".GEN" extension.');
      writeln('      The current case must be preprocessed.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_PREP(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'PREP\';
   Repeat
      Header(HelpPath);
      writeln('PREP CaseName');
      writeln('   CaseName: file name of case without ".GEN" extension.');
      writeln('      The process will run without waiting is there is no error.');
      writeln('PREP CaseName VariableName1=Value1 ... VariableNameN=ValueN');
      writeln('   CaseName must be the first parameter.');
      writeln('   VariableName,Value: The variable "VariableName" is assigned');
      writeln('      the value "Value".');
      writeln('      This variable is passed into the preprocessed file.');
      writeln('      Attempts to re-define the variable in the file is ignored.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_SOLVE(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'SOLVE\';
   Repeat
      Header(HelpPath);
      writeln('SOLVE NoWait');
      writeln('   NoWait causes the process to run without waiting.');
      writeln('SOLVE Printer');
      writeln('   Printer sends the execution profile to the printer.');
      writeln('SOLVE Printer NoWait');
      writeln('   Parameters can be simultaneous and interchanged.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
```

```
end;
{-------------------------------------------------------------------------}
Procedure Batch_STRESS(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'STRESS\';
   Repeat
      Header(HelpPath);
      writeln('STRESS [FiberStrain] [FiberStress] [ElemStrain] [ElemStress]');
      writeln('       [GlobalStrain] [GlobalStress]');
      writeln('   The process runs without waiting when a parameter is passed.');
      writeln('   FiberStrain:  For calculating strain in the material direction.');
      writeln('   FiberStress:  For calculating stress in the material direction.');
      writeln('   ElemStrain:   For calculating strain in the element  direction.');
      writeln('   ElemStress:   For calculating stress in the element  direction.');
      writeln('   GlobalStrain: For calculating strain in the global   direction.');
      writeln('   GlobalStress: For calculating stress in the global   direction.');
      writeln('   Parameters can be simultaneous and interchanged.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_SUMTONEW(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'SUMTONEW\';
   Repeat
      Header(HelpPath);
      writeln('SUMTONEW CaseName1 Mult1 [CaseName2 Mult2 ...] NewCaseName');
      writeln('This permits to add the solution of many cases:');
      writeln('   CaseName1 ... CaseNameN (without .GEN extension).');
      writeln('The forces, displacements, strains and stresses of each case');
      writeln('   is correspondingly multyplied by Mult1 ... MultN before');
      writeln('   they are summed.');
      writeln('Only the common strains and stresses to all cases are summed.');
      writeln('Each case must be solved (force displacements) and saved with MANAGER.');
      writeln('All cases summed must be saved in the same "Save Root" directory.');
      writeln('   (See \Main menu\MANAGER\)');
      writeln('A single case can be used for the summation.');
      writeln('NewCaseName is the case name to which results are summed.');
      writeln('   Using MANAGER will restore this case.');
      writeln('The geometry of all cases must be the same.');
```

```
      writeln('This process avoids the long solving time for a loading which is');
      writeln('   a linear combination of previous loading cases to a geometry.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Batch_WORST(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'WORST\';
   Repeat
      Header(HelpPath);
      writeln('WORST NoWait');
      writeln('   NoWait causes the process to run without waiting.');
      writeln('WORST Printer');
      writeln('   Printer causes the displayed information to be sent to the');
      writeln('   printer followed with a form feed.');
      writeln('WORST NoWait Printer');
      writeln('   NoWait and Printer parameter can be simultaneous ',
              'and interchanged');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
{
Procedure BatchMode(HelpPath: String255);
}
Procedure BatchMode;
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Batch mode\';
   Repeat
      Header(HelpPath);
      writeln('A batch mode is available. It is used to run many cases one after another.');
      writeln('This is done by passing parameters to the child process programs.');
      writeln;
      writeln('   A- CONFIG       (commands are shown in a natural sequence)');
      writeln('   B- COPYGEN');
      writeln('   C- PREP');
      writeln('   D- NEWNAME');
      writeln('   E- SOLVE');
      writeln('   F- WORST');
```

```
      writeln('   G- ADDTOEND');
      writeln('   H- STRESS');
      writeln('   I- DELSTRES');
      writeln('   J- SUMTONEW');
      writeln('   K- MANAGER');
      writeln;
      writeln('A maximum of 127 characters are allowed for a command line.');
      writeln('These commands can be used inside a batch file (*.BAT).');
      writeln('See you DOS manual for further help.');
      GoBack(HelpPath,long,C,Action);
      If C='A' then Batch_CONFIG(HelpPath);
      If C='B' then Batch_COPYGEN(HelpPath);
      If C='C' then Batch_PREP(HelpPath);
      If C='D' then Batch_NEWNAME(HelpPath);
      If C='E' then Batch_SOLVE(HelpPath);
      If C='F' then Batch_WORST(HelpPath);
      If C='G' then Batch_ADDTOEND(HelpPath);
      If C='H' then Batch_STRESS(HelpPath);
      If C='I' then Batch_DELSTRES(HelpPath);
      If C='J' then Batch_SUMTONEW(HelpPath);
      If C='K' then Batch_MANAGER(HelpPath);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
end.
```

H_COMMON.PAS

```
{$N+}
Unit H_Common;
Interface
Uses
   Crt,Graph,J_Graph,Declare,WhatKey,Sonore;
{
                  HELP variables
}
Const
   Npoint_max = 300;
   Ncon_max = 300;
Type
   MatrixR_PointxD = array[1..Npoint_max,1..3] of Rtype;
   VectI_Ncon = array[1..Ncon_max] of Integer;
   VectS5_Point = array[1..Npoint_max] of String[5];
{-------------------------------------------------------------------------}
Procedure Header(    HelpPath: String255);
Procedure GoBack(    HelpPath: String255;
                     Previouslength: integer;
                 var C: Char;
                 var Action: String6);
Procedure Two(    Title: String80;
              var XYZ: MatrixR_PointxD;  { var to save memory only }
                  Npoint: Integer;
              var Dep,Fin: VectI_Ncon;
              var Name: VectS5_Point;
                  Ntrait: integer;
                  Rho,D,Theta,Phi: Rtype;
              var Action: String6);
{-------------------------------------------------------------------------------}
Implementation
Procedure Header;
{
Procedure Header(    HelpPath: String255);
}
begin
   ClrScr;
   Write('Help Level: ');
   TextColor(LightRed);
   Writeln(HelpPath);
   TextColor(LightGreen);
end;
{-------------------------------------------------------------------------}
Procedure GoBack;
{
Procedure GoBack(    HelpPath: String255;
                     Previouslength: integer;
                 var C: Char;
                 var Action: String6);
}
```

```
{
   Input:
      HelpPath: Present Path in Help
      Previouslength: Previous length of path
   Output:
      C: chosen character;
      Action: Action taken;
      Writes <Esc>- "Previous help level"
      If <Esc> key is hit, Help path is returned to previous one
}
begin
   write('<Esc>- Back to: ',Copy(HelpPath,1,Previouslength));
   Get_Key(C,Action); C:=UpCase(C);
   If Action='ESC' then
      begin
      HelpPath:=Copy(HelpPath,1,Previouslength+1);
      beep;
      end;
end;
{-------------------------------------------------------------------------------}
Procedure Two;
{
Procedure Two(    Title: String80;
              var XYZ: MatrixR_PointxD;        var to save memory only
                  Npoint: Integer;
              var Dep,Fin: Vectl_Ncon;
              var Name: VectS5_Point;
                  Ntrait: integer;
                  Rho,D,Theta,Phi: Rtype;
              var Action: String6);
}
{
   Input:
      Title: Title;
      XYZ[i]: Z Y and Z Coordinates of point i
      Npoints: number of points
      Dep[i]: starting point # for line i
      Fin[i]: ending point # for line i
      Name[i]: Label for node i
      Ntrait: number of line
      Rho: The distance from the viewer to the origin
      D: Distance between the viewer and intersection plane
      Theta: Angle with X axis of view point in Degrees
      Phi: Angle with Z axis of view point in Degrees
   Output:
      Draw the preceding with Labeling of nodes
      Action: Action taken by user:
              'LEFT','RIGHT','UP','DOWN','ESC'
}
var
   XY: array[1..Npoint_Max,1..2] of Rtype;
   Margin,Ratio,amplitude,s1,c1,s2,c2,xe,ye,ze: Rtype;
```

```
   iD,i,GrDriver,GrMode: integer;
   Xasp,Yasp: word;
   a,Max,Min: array[1..2] of Rtype;
   Smax,Smin,Sign: array[1..2] of integer;
   C: Char;
   Mot: String[8];
begin
{
                Transform from degrees to radians
}
   Theta:=Theta*Pi/180;
   Phi:=Phi*Pi/180;
{
                Get Graphic Setup
}
   GrDriver:=Detect;
   InitGraph(grDriver,grMode,'');
   Smax[1]:=GetMaxX;
   Smin[1]:=0;
   Smax[2]:=0;
   Smin[2]:=GetMaxY;
   GetAspectRatio(Xasp,Yasp);
   Ratio:=Yasp/Xasp;
   S1:=Sin(Theta) ; C1:=Cos(Theta);
   S2:=Sin(Phi) ; C2:=Cos(Phi);
{
                Initialize Max Min
}
   For iD:=1 to 2 do
      begin
      Max[iD]:=-1E30;
      Min[iD]:=1E30;
      end;
{
                Perspective projection
}
   for i:=1 to Npoint do
      begin
      Xe:=-xyz[i,1]*S1+xyz[i,2]*C1;
      Ye:=-xyz[i,1]*C1*C2-xyz[i,2]*S1*C2+xyz[i,3]*S2;
      Ze:=-xyz[i,1]*S2*C1-xyz[i,2]*S1*S2-xyz[i,3]*C2+Rho;
      XY[i,1]:=(D*Xe/Ze)*Ratio;
      XY[i,2]:=D*Ye/Ze;
{
                Find Max min
}
      For iD:=1 to 2 do
         begin
         If XY[i,iD]<Min[iD] then Min[iD]:=XY[i,iD];
         If Max[iD]<XY[i,iD] then Max[iD]:=XY[i,iD];
         end;
      end;
```

```
{
                Leave a margin for Max Min
}
  Margin:=0.025*(max[1]-Min[1]);   { X margin } {0.025 for = 2/80 = 2 col.}
  max[1]:=max[1]+Margin;
  min[1]:=min[1]-Margin;
  Margin:=0.03*(max[2]-Min[2]);   { Y margin } {0.03 for > 1/25/2=half a line}
  If Title<>'' then max[2]:=max[2]+Margin+(max[2]-min[2])/20
              else max[2]:=max[2]+Margin;
  min[2]:=min[2]-Margin;
{
                Find appropriate amplitude (lowest ratio)
                in order to keep proportion in geometry
                Take Sign in account
}
  For iD:=1 to 2 do
    a[iD]:=(Smax[iD]-Smin[iD])/(max[iD]-min[iD]);
  If abs(a[1])<=abs(a[2]) then Amplitude:=abs(a[1])
                          else Amplitude:=abs(a[2]);
  For iD:=1 to 2 do
    Sign[iD]:=Round((Smax[iD]-Smin[iD])/abs(Smax[iD]-Smin[iD]));
{
                Scale and center coordinates on screen
}
  For i:=1 TO Npoint do
    For iD:=1 to 2 do
    XY[i,iD]:=(XY[i,iD]-(max[iD]+min[iD])/2)*Sign[iD]*Amplitude
            +(Smax[iD]+Smin[iD])/2;
{
                Draw lines
}
  For i:=1 to Ntrait do
    Line(Round(XY[Dep[i],1]),Round(XY[Dep[i],2]),
        Round(XY[Fin[i],1]),Round(XY[Fin[i],2]));
{
                Get node Labels
}
  For i:=1 to Npoint do
    If Name[i]<>'' then
      Put_Mot(Name[i],XY[i,1],XY[i,2]);
{
                Write Title
}
  If Title<>'' then
    OutTextXY(Round(GetMaxX/2-TextWidth(Title)/2),0,Title);
  Get_Key(C,Action);
  Beep;
  CloseGraph;    {Free heap memory used for graphics}
end;
{-------------------------------------------------------------}
end.
```

H_ETYPE.PAS

```
{$N+}
Unit H_Etype;
Interface
Uses
   Crt,Declare,H_Common,Sonore,WaitKey,Linear;
Procedure Element(HelpPath: String255);
{-------------------------------------------------------------------}
Implementation
Procedure View1;
var
   i Npoint,Ntrait,Inter: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
   Radius,Xc,Yc: Rtype;
begin
   Beep;
   ClrScr;
{
                  corners
}
   XYZ[1,1]:=0.1;   XYZ[1,2]:=0.1;   Name[1]:='';
   XYZ[2,1]:=0.9;   XYZ[2,2]:=0.4;   Name[2]:='';
   XYZ[3,1]:=0.3;   XYZ[3,2]:=0.9;   Name[3]:='';
{
                  Axis
}
   XYZ[4,1]:=0.0;   XYZ[4,2]:=0.0;   Name[4]:='O  ';
   XYZ[5,1]:=1.0;   XYZ[5,2]:=0.0;   Name[5]:='';
   XYZ[6,1]:=0.0;   XYZ[6,2]:=1.0;   Name[6]:='';
   XYZ[7,1]:=0.07;  XYZ[7,2]:=0.07;  Name[7]:='1';
   XYZ[8,1]:=0.95;  XYZ[8,2]:=0.4;   Name[8]:='2';
   XYZ[9,1]:=0.3;   XYZ[9,2]:=0.95;  Name[9]:='3';
   XYZ[10,1]:=1.05; XYZ[10,2]:=0.0;  Name[10]:='X';
   XYZ[11,1]:=0.0;  XYZ[11,2]:=1.05; Name[11]:='Y';
   Npoint:=11;
   Dep[1]:=1;  Fin[1]:=2;
   Dep[2]:=2;  Fin[2]:=3;
   Dep[3]:=3;  Fin[3]:=1;
   Dep[4]:=4;  Fin[4]:=5;
   Dep[5]:=4;  Fin[5]:=6;
   Ntrait:=5;
{
                  Show numbering sequence
}
   Inter:=20;
   Radius:=0.15;
   Xc:=0.43;
   Yc:=0.48;
```

```
   For i:=0 to Inter do
      begin
      xyz[Npoint+i+1,1]:=Xc+Radius*Cos(-Pi+i*3/2*Pi/Inter);
      xyz[Npoint+i+1,2]:=Yc+Radius*Sin(-Pi+i*3/2*Pi/Inter);
      Name[Npoint+i+1]:='';
      end;
   For i:=1 to Inter do
      begin
      Dep[Ntrait+i]:=Npoint+i;
      Fin[Ntrait+i]:=Npoint+i+1;
      end;
   Npoint:=Npoint+Inter+1;
   Ntrait:=Ntrait+Inter;
{
                  Arrow tip after arc
}
   xyz[Npoint+1,1]:=xyz[Npoint,1]-Radius*0.2;
   xyz[Npoint+1,2]:=xyz[Npoint,2];
   xyz[Npoint+2,1]:=xyz[Npoint,1];
   xyz[Npoint+2,2]:=xyz[Npoint,2]+Radius*0.08;
   xyz[Npoint+3,1]:=xyz[Npoint,1];
   xyz[Npoint+3,2]:=xyz[Npoint,2]-Radius*0.08;
   name[Npoint+1]:='';
   name[Npoint+2]:='';
   name[Npoint+3]:='';
   Dep[Ntrait+1]:=Npoint+1;
   Fin[Ntrait+1]:=Npoint+2;
   Dep[Ntrait+2]:=Npoint+1;
   Fin[Ntrait+2]:=Npoint+3;
   Dep[Ntrait+3]:=Npoint+2;
   Fin[Ntrait+3]:=Npoint+3;
   Npoint:=Npoint+3;
   Ntrait:=Ntrait+3;
   For i:=1 to Npoint do XYZ[i,3]:=0;
   Two('Element type 1',XYZ,Npoint,Dep,Fin,Name,Ntrait,
       1000,1000,-90,0,Action);
end;
{-------------------------------------------------------------------------}
Procedure View2;
var
   Edge,Inter,i,Npoint,Ntrait: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Theta,Phi,Per: Rtype;
   Action: String6;
begin
   Beep;
   ClrScr;
   Inter:=10; (number of intervals on each edge )
   Npoint:=12*(Inter+1);
   If Npoint>Npoint_max then
```

```
      begin
      writeln('too many points for element 2');
      wait;
      Exit;
      end;
{
                  Define edge 1
}
   Edge:=1;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=1-0.1*Sin(Per*Pi);
      XYZ[i,2]:=2*Per;
      XYZ[i,3]:=0+0.1*Sin(Per*Pi);
      end;
{
                  Define edge 2
}
   Edge:=2;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=1-Per;
      XYZ[i,2]:=2+0.1*Sin(Per*Pi);
      XYZ[i,3]:=0+0.1*Sin(Per*Pi);
      end;
{
                  Define edge 3
}
   Edge:=3;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=0-0.1*Sin(Per*Pi);
      XYZ[i,2]:=2-2*Per;
      XYZ[i,3]:=0+0.1*Sin(Per*Pi);
      end;
{
                  Define edge 4
}
   Edge:=4;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=0+Per;
      XYZ[i,2]:=0+0.1*Sin(Per*Pi);
      XYZ[i,3]:=0+0.1*Sin(Per*Pi);
      end;
{
                  Define edge 5
}
```

```
   Edge:=5;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=1-0.1*Sin(Per*Pi);
      XYZ[i,2]:=0+0.1*Sin(Per*Pi);
      XYZ[i,3]:=0+Per;
      end;
{
                  Define edge 6
}
   Edge:=6;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=1-0.1*Sin(Per*Pi);
      XYZ[i,2]:=2+0.1*Sin(Per*Pi);
      XYZ[i,3]:=0+Per;
      end;
{
                  Define edge 7
}
   Edge:=7;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=0-0.1*Sin(Per*Pi);
      XYZ[i,2]:=2+0.1*Sin(Per*Pi);
      XYZ[i,3]:=0+Per;
      end;
{
                  Define edge 8
}
   Edge:=8;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=0-0.1*Sin(Per*Pi);
      XYZ[i,2]:=0+0.1*Sin(Per*Pi);
      XYZ[i,3]:=0+Per;
      end;
{
                  Define edge 9
}
   Edge:=9;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=1-0.1*Sin(Per*Pi);
      XYZ[i,2]:=2*Per;
      XYZ[i,3]:=1+0.1*Sin(Per*Pi);
      end;
```

```
{
                Define edge 10
}
   Edge:=10;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=1-Per;
      XYZ[i,2]:=2+0.1*Sin(Per*Pi);
      XYZ[i,3]:=1+0.1*Sin(Per*Pi);
      end;
{
                Define edge 11
}
   Edge:=11;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=0-0.1*Sin(Per*Pi);
      XYZ[i,2]:=2-2*Per;
      XYZ[i,3]:=1+0.1*Sin(Per*Pi);
      end;
{
                Define edge 12
}
   Edge:=12;
   For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1) do
      begin
      Per:=(i-1-(Edge-1)*(Inter+1))/Inter;
      XYZ[i,1]:=0+Per;
      XYZ[i,2]:=0+0.1*Sin(Per*Pi);
      XYZ[i,3]:=1+0.1*Sin(Per*Pi);
      end;
{
                Connect edges, calculate Ntrait
}
   Ntrait:=0;
   For Edge:=1 to 12 do
      For i:=(Edge-1)*(Inter+1)+1 to Edge*(Inter+1)-1 do
         begin
         Ntrait:=Ntrait+1;
         Dep[Ntrait]:=i;
         Fin[Ntrait]:=i+1;
         end;
{
                Set Label
}
   For i:=1 to Npoint do Name[i]:='';
   Name[Lin(1,1,Inter+1)]:='1';
   Name[Lin(1,Round((Inter+1)/2),Inter+1)]:='2';
   Name[Lin(1,Inter+1,Inter+1)]:='3';
   Name[Lin(2,Round((Inter+1)/2),Inter+1)]:='4';
```

```
   Name[Lin(2,Inter+1,Inter+1)]:='5';
   Name[Lin(3,Round((Inter+1)/2),Inter+1)]:='6';
   Name[Lin(3,Inter+1,Inter+1)]:='7';
   Name[Lin(4,Round((Inter+1)/2),Inter+1)]:='8';
   Name[Lin(5,Round((Inter+1)/2),Inter+1)]:='9';
   Name[Lin(6,Round((Inter+1)/2),Inter+1)]:='10';
   Name[Lin(7,Round((Inter+1)/2),Inter+1)]:='11';
   Name[Lin(8,Round((Inter+1)/2),Inter+1)]:='12';
   Name[Lin(9,1,Inter+1)]:='13';
   Name[Lin(9,Round((Inter+1)/2),Inter+1)]:='14';
   Name[Lin(9,Inter+1,Inter+1)]:='15';
   Name[Lin(10,Round((Inter+1)/2),Inter+1)]:='16';
   Name[Lin(10,Inter+1,Inter+1)]:='17';
   Name[Lin(11,Round((Inter+1)/2),Inter+1)]:='18';
   Name[Lin(11,Inter+1,Inter+1)]:='19';
   Name[Lin(12,Round((Inter+1)/2),Inter+1)]:='20';
{
                Add X Y Z axis
}
   XYZ[Npoint+1,1]:=-0.2; XYZ[Npoint+1,2]:=-0.1; XYZ[Npoint+1,3]:=0.0;
   XYZ[Npoint+2,1]:=+1.4; XYZ[Npoint+2,2]:=-0.1; XYZ[Npoint+2,3]:=0.0;
   XYZ[Npoint+3,1]:=-0.2; XYZ[Npoint+3,2]:=+2.4; XYZ[Npoint+3,3]:=0.0;
   XYZ[Npoint+4,1]:=-0.2; XYZ[Npoint+4,2]:=-0.1; XYZ[Npoint+4,3]:=1.4;
   Name[Npoint+1]:='O';
   Name[Npoint+2]:='X';
   Name[Npoint+3]:='Y';
   Name[Npoint+4]:='Z';
   Dep[Ntrait+1]:=Npoint+1;  Fin[Ntrait+1]:=Npoint+2;
   Dep[Ntrait+2]:=Npoint+1;  Fin[Ntrait+2]:=Npoint+3;
   Dep[Ntrait+3]:=Npoint+1;  Fin[Ntrait+3]:=Npoint+4;
   Npoint:=Npoint+4;
   Ntrait:=Ntrait+3;
{
                Add local coordinate System
}
   XYZ[Npoint+1,1]:=+0.5; XYZ[Npoint+1,2]:=+1.0; XYZ[Npoint+1,3]:=+0.5;
   XYZ[Npoint+2,1]:=+0.5; XYZ[Npoint+2,2]:=+1.5; XYZ[Npoint+2,3]:=+0.5;
   XYZ[Npoint+3,1]:=+0.2; XYZ[Npoint+3,2]:=+1.0; XYZ[Npoint+3,3]:=+0.5;
   XYZ[Npoint+4,1]:=+0.5; XYZ[Npoint+4,2]:=+1.0; XYZ[Npoint+4,3]:=+0.8;
   Name[Npoint+1]:='o';
   Name[Npoint+2]:='Xi';
   Name[Npoint+3]:='Eta';
   Name[Npoint+4]:='Zeta';
   Dep[Ntrait+1]:=Npoint+1;  Fin[Ntrait+1]:=Npoint+2;
   Dep[Ntrait+2]:=Npoint+1;  Fin[Ntrait+2]:=Npoint+3;
   Dep[Ntrait+3]:=Npoint+1;  Fin[Ntrait+3]:=Npoint+4;
   Npoint:=Npoint+4;
   Ntrait:=Ntrait+3;
{
                Draw element
}
   Theta:=-32;
```

```
   Phi:=72;
   Repeat
      Two('Element type 2',XYZ,Npoint,Dep,Fin,Name,Ntrait,
          1000,1000,Theta,Phi,Action);
      If Action='LEFT' then Theta:=Theta-8;
      If Action='RIGHT' then Theta:=Theta+8;
      If Action='UP' then Phi:=Phi-8;
      If Action='DOWN' then Phi:=Phi+8;
      until (Action<>'LEFT') and (Action<>'RIGHT')
            and (Action<>'UP') and (Action<>'DOWN');
end;
{-------------------------------------------------------------------------}
Procedure Element;
{
Procedure Element(HelpPath: String255);
}
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   ClrScr;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Elements\';
   Repeat
      Header(HelpPath);
      writeln('Element type 1:');
      writeln('   3 node, 2D, isotropic, linear displacements.');
      writeln('   Face number 1 is 1-2 segment, ... Face 3 is 3-1 segment.');
      writeln('Element type 2:');
      writeln('   20 node, 3D, orthotropic at an angle in reference to the');
      writeln('   curved local coordinate system of the element, quadratic');
      writeln('   displacement. The exterior of the element can follow a');
      writeln('   curved boundary because of its quadratic shape.');
      writeln('   The material properties are given at an angle in the');
      writeln('   local coordinate system Xi Eta Zeta, the angle is in the');
      writeln('   Xi-Eta plane (0 at Xi) and increases with a trigonometric');
      writeln('   rotation about the Zeta axis.');
      writeln('   The local coordinate system curves to follow the element');
      writeln('   shape.');
      writeln('   Face 1 is at Zeta=-1, Face 2 at Eta=-1, Face 3 at Xi=1,');
      writeln('   Face 4 is at Eta=1, Face 5 at Xi=-1 and Face 6 at Zeta=1.');
      writeln('NOTE: It is necessary to define your element in the numbering');
      writeln('   sequence shown when using the ELEM command (see \Language\).');
      writeln('1- View element type 1');
      writeln('2- View element type 2 (use arrows to change viewpoint)');
      GoBack(HelpPath,long,C,Action);
      If C='1' then View1;
      If C='2' then View2;
      until Action='ESC';
end;
```

```
{-------------------------------------------------------------------------}
end.
```

H_FEATUR.PAS

```
{$N+}
Unit H_Featur;
Interface
Uses
  Crt,Declare,Sonore,H_Common,WaitKey,Linear;
Procedure Features(HelpPath: String255);
{-------------------------------------------------------------------------}
Implementation
Procedure Performance(HelpPath: String255);
var
  C: char;
  Action: String6;
  Long: Byte;
begin
  Beep;
  ClrScr;
  Long:=Length(HelpPath);
  HelpPath:=HelpPath+'Performance\';
  Repeat
    Header(HelpPath);
    writeln('The Math Coprocessor is used for increased speed.');
    writeln('The program supports the 8087/80287/80387 chips.');
    writeln('A study of read and write patterns has been done to reduce');
    writeln('   the frequency of disk access. This has been applied in ');
    writeln('   particular for the SOLVE process.');
    writeln('Up to 350 degrees of freedom are kept in RAM during SOLVE,');
    writeln('   depending on available RAM. The rest is kept on disk.');
    writeln('Speed is highest for an optimal number of buffers in your');
    writeln('   CONFIG.SYS file (see \Sequence\). Buffers=17 seems best.');
    writeln('   By default Buffers=2 (3 on an IBM AT).');
    writeln('   Each buffer takes up 528 bytes of memory (17 -> 9 kB).');
    writeln('A second way is to use a commercial disk caching program.');
    writeln('A third way to increase the speed is to define a RAM disk');
    writeln('   and then specify that drive for the temporary files (*.T)');
    writeln('   in CONFIGURE. DOS provides Vdisk. Vdisk can be set in expanded memory.');
    writeln('You can chart or compare changes by looking at the displayed');
    writeln('   elapsed time for solving in the SOLVE process.');
    GoBack(HelpPath,long,C,Action);
    until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Accuracy(HelpPath: String255);
var
  C: char;
  Action: String6;
  Long: Byte;
begin
  Beep;
  ClrScr;
  Long:=Length(HelpPath);
```

```
   HelpPath:=HelpPath+'Accuracy\';
   Repeat
      Header(HelpPath);
      writeln('This program makes its calculation in double precision (16 significant digits)');
      writeln('   with real numbers of 8 bytes. The range is +/- 5.0E-324 to 1.7E308.');
      writeln('The strains and stresses are evaluated at the optimal sampling points ');
      writeln('   (gauss-quadrature) and the extrapolated at the nodes. This method has');
      writeln('   a better precision than evaluating the strains at the nodes directly.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Size(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   ClrScr;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Size\';
   Repeat
      Header(HelpPath);
      writeln('All addressing is done by using long integers (+/- 2147483647)');
      writeln('   numbers of 4 bytes. This permits to have an "infinite" geometry');
      writeln('   size and wave front matrix.');
      writeln('Your only limits are time and the size of your hard disk.');
      writeln('   Example: For a 3 dimensional problem, a 200 node wave front with 3000');
      writeln('            nodes in total will occupy a maximum of 50 MB.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure GenInfo(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   ClrScr;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'General\';
   Repeat
      Header(HelpPath);
      writeln('It was written by Jerome Daoust as part of a Ph.D. in Mechanical');
      writeln('   Engineering in composite materials (1989).');
      writeln('Written in PASCAL, this program has a total of 26000 lines (Bf).');
      writeln('In January 1988 no commercial program was found to solve cases');
      writeln('   using an orthotropic element with a material angle.');
      writeln('This program was designed from the start to be used on a personal computer.');
```

```
      writeln('The environment of Turbo Pascal 4.0 (TM) provided sufficient support.');
      writeln('This program will automatically adapt itself to any graphic card:');
      writeln('   CGA,MCGA,EGA,VGA,Hercules,AT&T 6300,IBM 3270 PC.');
      writeln('You can use your favorite word processor from inside this program.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Bonus(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   ClrScr;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Bonus\';
   Repeat
      Header(HelpPath);
      writeln('I have written a utility program: FILL.exe');
      writeln('It is used to copy files from the hard disk to diskettes.');
      writeln('Those files can be restored any time with the COPY command.');
      writeln('This provides a simpler alternative to the BACKUP and RESTORE');
      writeln('   commands from DOS.');
      writeln('More information is available by typing: FILL<Enter>');
      writeln;
      writeln('Another utility is REGRES.exe. Help is available from within.');
      writeln('It is used for linear regression on files from Lotus 1-2-3.');
      writeln('Formulas can be edited to match a data set of many variables.');
      wri     'Variables can be multyplied within terms of the formula.');
      GoBa    elpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Features;
{
Procedure Features(HelpPath: String255);
}
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Features\';
   Repeat
      Header(HelpPath);
      writeln('P- Performance');
      writeln('A- Accuracy');
      writeln('S- Size of problem');
```

```
      writeln('G- General information');
      writeln('B- Bonus');
      GoBack(HelpPath,long,C,Action);
      If C='A' then Accuracy(HelpPath);
      If C='P' then Performance(HelpPath);
      If C='S' then Size(HelpPath);
      If C='G' then GenInfo(HelpPath);
      If C='B' then Bonus(HelpPath);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
end.
```

H_LANG.PAS

```
{$N+}
Unit H_Lang;
Interface
Uses
   Crt,Declare,Sonore,H_Common;
Procedure Language(HelpPath: String255);
{-------------------------------------------------------------------------}
Implementation
Procedure L_DISPL(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'DISPL\';
   Repeat
      Header(HelpPath);
      writeln('DISPL FromNode,[ToNode],[Repeat],[dNode],[Dx],[Dy],[Dz];');
      writeln('   FromNode (I): Set displacements at nodes starting with FromNode.');
      writeln('   ToNode (I): Set displacements at nodes ending with ToNode.');
      writeln('      Default: ToNode=FromNode.');
      writeln('   Repeat (I): number of times the given pattern is repeated');
      writeln('      excluding the original pattern. Range: 0...N (Default=0).');
      writeln('   dNode (I): increase in node number for each new pattern.');
      writeln('      Default: dNode=ToNode-FromNode+1. This places the new nodes after');
      writeln('      node ToNode. Restriction: dNode > ToNode-FromNode.');
      writeln('   Dx,Dy,Dz (IR): Displacements for the range of nodes.');
      writeln('      Forces are assumed to be known for all nodes initially and equal to 0.');
      writeln('      Only when a value is given is that direction redefined for a');
      writeln('      known displacement.');
      writeln('      This permits to have known forces and displacements for a given node.');
      writeln('example:');
      writeln('   DISPL 4,8,,,0,,0;');
      writeln('   --> set nodes 4 to 8 as having a known Dx=0 and Dz=0.');
      writeln('       The y direction is left at it''s previous state of definition.');
      writeln('note: It is a good practice to use this command after all');
      writeln('      FORCE and TRAC commands.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------- ----------------------------------------------------}
Procedure L_EDEL(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
```

```
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'EDEL\';
   Repeat
      Header(HelpPath);
      writeln('EDEL ElementNumber;');
      writeln('   ElementNumber (I): Number of element to delete.');
      writeln('example:');
      writeln('   EDEL 32;');
      writeln('   --> Element 32 is deleted.');
      writeln('       The node coordinates are kept (deleted later if unused).');
      writeln('Note: This command is mainly used to remove "failed" elements.');
      writeln('   Once the case is solved and the "worst" element found, the');
      writeln('   element can be automatically removed (See \Batch mode\ADDTOEND\)');
      writeln('   to simulate a crack propagation.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_EGEN(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'EGEN\';
   Repeat
      Header(HelpPath);
      writeln('EGEN FromElement,[ToElement],Repeat,[dElement],dNode;');
      writeln('   FromElement (I): First element considered.');
      writeln('   ToElement (I): Last element considered.');
      writeln('      Default: ToElement=FromElement.');
      writeln('   Repeat (I): number of time pattern is repeated. Range: 0...N.');
      writeln('       Pattern is FromElement to ToElement element numbers.');
      writeln('   dElement (I): step for element numbers between each pattern.');
      writeln('       Default: ToElement-FromElement+1');
      writeln('   dNode (I): increase in node numbers for the elements at each');
      writeln('      new pattern.');
      writeln('example:');
      writeln('   EGEN 34,40,10,,100;');
      writeln('   --> Generate 10 new patterns by imitating elements 3 to 5 and');
      writeln('       appending new elements numbers to those existing while');
      writeln('       increasing node numbers by 100 at each new pattern.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_ELEM(HelpPath: String255);
var
   C: char;
   Action: String6;
```

```
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'ELEM\';
   Repeat
      Header(HelpPath);
      writeln('ELEM Number,Mat,Etype,[Angle],Node1,Node2,Node3,[Node4]... ;');
      writeln('   Number (I): element number.');
      writeln('      Your element numbering can be discontinuous.');
      writeln('   Mat (I): material number. It must be previously defined.');
      writeln('   Etype (I): element type (See \Features\Elements\).');
      writeln('   Angle (IR): angle between material direction and local coordinate');
      writeln('      system of the element. Default: angle=0. Angle in DEGREES.');
      writeln('   Node1...NodeX: number of nodes used.');
      writeln('      Be sure to follow the same order as the typical element shown in');
      writeln('      \Features\Elements\. Nodes chosen must be previously defined.');
      writeln('example:');
      writeln('   ELEM 3,1,,67,34,76;');
      writeln('   --> element 3 is the of type 1 and uses nodes 67,34 and 76.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_EMOD(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'EMOD\';
   Repeat
      Header(HelpPath);
      writeln('EMOD ExistingElemNumber,[NewMat],[NewAngle];');
      writeln('   ExistingElemNumber (I): a previously defined element number.');
      writeln('   NewMat (I): material number. It must be previously defined.');
      writeln('      Default: Material number of previous element definition.');
      writeln('   NewAngle (IR): angle between material direction and local coordinate');
      writeln('      system of the element.');
      writeln('      Default: Angle of previous element definition.');
      writeln('      Angle in DEGREES.');
      writeln('example:');
      writeln('   EMOD 4,3,22.5;');
      writeln('   --> The material number and angle of previously defined element 4');
      writeln('       are respectively changed to 3 and 22.5 degrees.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_FORCE(HelpPath: String255);
```

```
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'FORCE\';
   Repeat
      Header(HelpPath);
      writeln('FORCE FromNode,[ToNode],[Repeat],[dNode],[Fx],[Fy],[Fz];');
      writeln('   FromNode (I): Set displacements at nodes starting with FromNode.');
      writeln('   ToNode (I): Set displacements at nodes ending with ToNode.');
      writeln('      Default: ToNode=FromNode.');
      writeln('   Repeat (I): number of times the given pattern is repeated');
      writeln('      excluding the original pattern. Range: 0...N (Default=0).');
      writeln('   dNode (I): increase in node number for each new pattern.');
      writeln('      Default: dNode=ToNode-FromNode+1. This places the new nodes after');
      writeln('      node ToNode. Restriction: dNode > ToNode-FromNode.');
      writeln('   Fx,Fy,Fz (IR): Forces for the range of nodes.');
      writeln('      Forces are assumed to be known for all nodes initially and equal to 0.');
      writeln('      Only when a value is given is that direction redefined for a');
      writeln('      known forces.');
      writeln('      This permits to have known forces and displacements for a given node.');
      writeln('example:');
      writeln('   FORCE 4,8,,,,-1.5;');
      writeln('   --> set nodes 4 to 8 as having a known Fy=-1.5.');
      writeln('       The x and z direction are left at their previous state of definition.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_INTER(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'INTER\';
   Repeat
      Header(HelpPath);
      writeln('INTER Degree,NodeStart,NodeEnd,[NodeInc],[Repeat],[PatternInc],[Insert],');
      writeln('      [InsNodeStart],[InsNodeInc],[InsNodeStartInc],[InsPatternInc];');
      writeln('   Degree (I): Degree of polynomial for interpolation (1 to 10).');
      writeln('   NodeStart,NodeEnd (I): First and last node number in pattern.');
      writeln('   NodeInc (I): Node number increase in pattern, for nodes used to interpolate.');
      writeln('      Default: NodeInc = NodeEnd - NodeStart.');
      writeln('   Repeat (I): Number of times pattern is repeated. Default: Repeat = 0.');
      writeln('   PatternInc (I): Increment in node number for new patterns.');
      writeln('      Default: PatternInc = NodeEnd - NodeStart + NodeInc.');
```

```
      writeln('   Insert (I): Number of nodes inserted between each node of pattern.');
      writeln('      Default: Insert = 1 (2 divisions).');
      writeln('   InsNodeStart (I): First node number inserted between nodes of pattern.');
      writeln('      Default: InsNodeStart = NodeStart + 1.');
      writeln('   InsNodeInc (I): Increment in node numbers inserted between 2 nodes');
      writeln('      of pattern. Default: InsNodeInc = 1.');
      writeln('   InsNodeStartInc (I): Increment in first node numbers inserted between');
      writeln('      nodes of pattern. Default: InsNodeStartInc = NodeInc.');
      writeln('   InsPatternInc (I): Increment in inserted node numbers for each pattern.');
      writeln('      Default: InsPatternInc = (PatternInc/NodeInc)*InsNodeStartInc.');
      writeln('example:');
      writeln('   INTER 1,1,17,4,,,1,3,,4; --> Interpolates nodes 3,7,11,15 between nodes');
      writeln('   1,5,9,13,17. Forces and displacement are linearly interpolated. Coordinates');
      writeln('   are interpolated linearly within local coordinate system.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
Procedure Orientation_Plot;
{
   Output:
      Show unit vector L,M,N orientation
}
var
   Np,Nt: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
begin
   ClrScr;
   Np:=0;
   Nt:=0;
{
                  Axis
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='O  ';
   XYZ[Np+2,1]:=2.0;  XYZ[Np+2,2]:=0.0;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:=''; {X}
   XYZ[Np+3,1]:=0.0;  XYZ[Np+3,2]:=2.0;  XYZ[Np+3,3]:=0.0;  Name[Np+3]:=''; {Y}
   XYZ[Np+4,1]:=0.0;  XYZ[Np+4,2]:=0.0;  XYZ[Np+4,3]:=1.2;  Name[Np+4]:=''; {Z}
   XYZ[Np+5,1]:=2.1;  XYZ[Np+5,2]:=0.0;  XYZ[Np+5,3]:=0.0;  Name[Np+5]:='X';
   XYZ[Np+6,1]:=0.0;  XYZ[Np+6,2]:=2.1;  XYZ[Np+6,3]:=0.0;  Name[Np+6]:='Y';
   XYZ[Np+7,1]:=0.0;  XYZ[Np+7,2]:=0.0;  XYZ[Np+7,3]:=1.3;  Name[Np+7]:='Z';
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+1;  Fin[Nt+2]:=Np+3;
   Dep[Nt+3]:=Np+1;  Fin[Nt+3]:=Np+4;
   Np:=Np+7;
   Nt:=Nt+3;
{
                  Orientation vector, L,M,N
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='';
```

```
   XYZ[Np+2,1]:=1.7;  XYZ[Np+2,2]:=1.7;  XYZ[Np+2,3]:=1.7;  Name[Np+2]:='';
   XYZ[Np+3,1]:=1.8;  XYZ[Np+3,2]:=1.8;  XYZ[Np+3,3]:=1.8;  Name[Np+3]:='V';
   XYZ[Np+4,1]:=1.0;  XYZ[Np+4,2]:=0.0;  XYZ[Np+4,3]:=0.0;  Name[Np+4]:='';
   XYZ[Np+5,1]:=0.0;  XYZ[Np+5,2]:=1.0;  XYZ[Np+5,3]:=0.0;  Name[Np+5]:='';
   XYZ[Np+6,1]:=1.0;  XYZ[Np+6,2]:=1.0;  XYZ[Np+6,3]:=0.0;  Name[Np+6]:='';
   XYZ[Np+7,1]:=1.0;  XYZ[Np+7,2]:=1.0;  XYZ[Np+7,3]:=1.0;  Name[Np+7]:='';
   XYZ[Np+8,1]:=1.0;  XYZ[Np+8,2]:=0.5;  XYZ[Np+8,3]:=0.0;  Name[Np+8]:='M';
   XYZ[Np+9,1]:=0.5;  XYZ[Np+9,2]:=1.0;  XYZ[Np+9,3]:=0.0;  Name[Np+9]:='L';
   XYZ[Np+10,1]:=1.0; XYZ[Np+10,2]:=1.0; XYZ[Np+10,3]:=0.5; Name[Np+10]:='N';
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+4;  Fin[Nt+2]:=Np+6;
   Dep[Nt+3]:=Np+5;  Fin[Nt+3]:=Np+6;
   Dep[Nt+4]:=Np+6;  Fin[Nt+4]:=Np+7;
   Np:=Np+10;
   Nt:=Nt+4;
   Two('L,M,N orientation of vector V',XYZ,Np,Dep,Fin,Name,Nt,
       1000,1000,30,60,Action);
end;
{-------------------------------------------------------------------------}
Procedure Rectangular_Plot;
{
   Output:
      Show local rectangular coordinate system
}
var
   Np,Nt: Integer;
   XYZ: MatrixR_Pointx0;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
begin
   ClrScr;
   Np:=0;
   Nt:=0;
{
                  Axis
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='O'+chr(39)+'  ';
   XYZ[Np+2,1]:=2.0;  XYZ[Np+2,2]:=0.0;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:=''; (X)
   XYZ[Np+3,1]:=0.0;  XYZ[Np+3,2]:=2.0;  XYZ[Np+3,3]:=0.0;  Name[Np+3]:=''; (Y)
   XYZ[Np+4,1]:=0.0;  XYZ[Np+4,2]:=0.0;  XYZ[Np+4,3]:=1.2;  Name[Np+4]:=''; (Z)
   XYZ[Np+5,1]:=2.1;  XYZ[Np+5,2]:=0.0;  XYZ[Np+5,3]:=0.0;  Name[Np+5]:='X'+chr(39);
   XYZ[Np+6,1]:=0.0;  XYZ[Np+6,2]:=2.1;  XYZ[Np+6,3]:=0.0;  Name[Np+6]:='Y'+chr(39);
   XYZ[Np+7,1]:=0.0;  XYZ[Np+7,2]:=0.0;  XYZ[Np+7,3]:=1.3;  Name[Np+7]:='Z'+chr(39);
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+1;  Fin[Nt+2]:=Np+3;
   Dep[Nt+3]:=Np+1;  Fin[Nt+3]:=Np+4;
   Np:=Np+7;
   Nt:=Nt+3;
{
                  x' y' z' coordinates
}
```

```
   XYZ[Np+1,1]:=1.7;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='';
   XYZ[Np+2,1]:=0.0;  XYZ[Np+2,2]:=1.7;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:='';
   XYZ[Np+3,1]:=1.7;  XYZ[Np+3,2]:=1.7;  XYZ[Np+3,3]:=0.0;  Name[Np+3]:='';
   XYZ[Np+4,1]:=1.7;  XYZ[Np+4,2]:=1.7;  XYZ[Np+4,3]:=1.7;  Name[Np+4]:='';
   XYZ[Np+5,1]:=1.7;  XYZ[Np+5,2]:=1.7;  XYZ[Np+5,3]:=1.9;  Name[Np+5]:='P';
   XYZ[Np+6,1]:=1.9;  XYZ[Np+6,2]:=0.8;  XYZ[Np+6,3]:=0.0;  Name[Np+6]:='y'+chr(39);
   XYZ[Np+7,1]:=0.8;  XYZ[Np+7,2]:=1.9;  XYZ[Np+7,3]:=0.0;  Name[Np+7]:='x'+chr(39);
   XYZ[Np+8,1]:=1.9;  XYZ[Np+8,2]:=1.9;  XYZ[Np+8,3]:=0.8;  Name[Np+8]:='z'+chr(39);
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+3;
   Dep[Nt+2]:=Np+2;  Fin[Nt+2]:=Np+3;
   Dep[Nt+3]:=Np+3;  Fin[Nt+3]:=Np+4;
   Np:=Np+8;
   Nt:=Nt+3;
   Two('Local RECTANGULAR coordinate system P(x'',y'',z'')',
       XYZ,Np,Dep,Fin,Name,Nt,1000,1000,30,60,Action);
end;
{-------------------------------------------------------------------------}
Procedure Cylindrical_Plot;
{
   Output:
      Show local cylindrical coordinate system
}
var
   i,Np,Nt,inter: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
begin
   ClrScr;
   Np:=0;
   Nt:=0;
{
                  Axis
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='O'+chr(39)+'  ';
   XYZ[Np+2,1]:=2.0;  XYZ[Np+2,2]:=0.0;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:=''; {X}
   XYZ[Np+3,1]:=0.0;  XYZ[Np+3,2]:=2.0;  XYZ[Np+3,3]:=0.0;  Name[Np+3]:=''; {Y}
   XYZ[Np+4,1]:=0.0;  XYZ[Np+4,2]:=0.0;  XYZ[Np+4,3]:=1.2;  Name[Np+4]:=''; {Z}
   XYZ[Np+5,1]:=2.1;  XYZ[Np+5,2]:=0.0;  XYZ[Np+5,3]:=0.0;  Name[Np+5]:='X'+chr(39);
   XYZ[Np+6,1]:=0.0;  XYZ[Np+6,2]:=2.1;  XYZ[Np+6,3]:=0.0;  Name[Np+6]:='Y'+chr(39);
   XYZ[Np+7,1]:=0.0;  XYZ[Np+7,2]:=0.0;  XYZ[Np+7,3]:=1.3;  Name[Np+7]:='Z'+chr(39);
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+1;  Fin[Nt+2]:=Np+3;
   Dep[Nt+3]:=Np+1;  Fin[Nt+3]:=Np+4;
   Np:=Np+7;
   Nt:=Nt+3;
{
                  theta
}
   inter:=10;
   For i:=0 to inter do
```

```
      begin
      XYZ[Np+1+i,1]:=1*Cos(i/inter*Pi/4);
      XYZ[Np+1+i,2]:=1*Sin(i/inter*Pi/4);
      XYZ[Np+1+i,3]:=0.0;
      Name[Np+1+i]:='';
      end;
   For i:=1 to inter do
      begin
      Dep[Nt+i]:=Np+i;
      Fin[Nt+i]:=Np+i+1;
      end;
   Np:=Np+inter+1;
   Nt:=Nt+inter;
   XYZ[Np+1,1]:=1.1;  XYZ[Np+1,2]:=0.5;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='theta';
   Np:=Np+1;
{
                r theta z' coordinates
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='';
   XYZ[Np+2,1]:=1.7;  XYZ[Np+2,2]:=1.7;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:='';
   XYZ[Np+3,1]:=1.7;  XYZ[Np+3,2]:=1.7;  XYZ[Np+3,3]:=1.7;  Name[Np+3]:='';
   XYZ[Np+4,1]:=1.7;  XYZ[Np+4,2]:=1.7;  XYZ[Np+4,3]:=1.9;  Name[Np+4]:='P';
   XYZ[Np+5,1]:=1.0;  XYZ[Np+5,2]:=1.1;  XYZ[Np+5,3]:=0.0;  Name[Np+5]:='r';
   XYZ[Np+6,1]:=1.7;  XYZ[Np+6,2]:=1.8;  XYZ[Np+6,3]:=0.8;  Name[Np+6]:='z'+chr(39);
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+2;  Fin[Nt+2]:=Np+3;
   Np:=Np+6;
   Nt:=Nt+2;
   Two('Local CYLINDRICAL coordinate system P(r,theta,z'')',
       XYZ,Np,Dep,Fin,Name,Nt,1000,1000,30,60,Action);
end;
{-------------------------------------------------------------------------}
Procedure Spherical_Plot;
{
   Output:
      Show local spherical coordinate system
}
var
   i,Np,Nt,inter: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
begin
   ClrScr;
   Np:=0;
   Nt:=0;
{
                Axis
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[Np+1,3]:=0.0;  Name[Np+1]:='O'+chr(39)+'  ';
   XYZ[Np+2,1]:=2.0;  XYZ[Np+2,2]:=0.0;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:=''; {X}
```

```
   XYZ[Np+3,1]:=0.0;   XYZ[Np+3,2]:=2.0;   XYZ[Np+3,3]:=0.0;   Name[Np+3]:=''; {Y}
   XYZ[Np+4,1]:=0.0;   XYZ[Np+4,2]:=0.0;   XYZ[Np+4,3]:=1.2;   Name[Np+4]:=''; {Z}
   XYZ[Np+5,1]:=2.1;   XYZ[Np+5,2]:=0.0;   XYZ[Np+5,3]:=0.0;   Name[Np+5]:='X'+chr(39);
   XYZ[Np+6,1]:=0.0;   XYZ[Np+6,2]:=2.1;   XYZ[Np+6,3]:=0.0;   Name[Np+6]:='Y'+chr(39);
   XYZ[Np+7,1]:=0.0;   XYZ[Np+7,2]:=0.0;   XYZ[Np+7,3]:=1.3;   Name[Np+7]:='Z'+chr(39);
   Dep[Nt+1]:=Np+1;   Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+1;   Fin[Nt+2]:=Np+3;
   Dep[Nt+3]:=Np+1;   Fin[Nt+3]:=Np+4;
   Np:=Np+7;
   Nt:=Nt+3;
(
                theta
)
   inter:=10;
   For i:=0 to inter do
      begin
      XYZ[Np+1+i,1]:=1*Cos(i/inter*Pi/4);
      XYZ[Np+1+i,2]:=1*Sin(i/inter*Pi/4);
      XYZ[Np+1+i,3]:=0.0;
      Name[Np+1+i]:='';
      end;
   For i:=1 to inter do
      begin
      Dep[Nt+i]:=Np+i;
      Fin[Nt+i]:=Np+i+1;
      end;
   Np:=Np+inter+1;
   Nt:=Nt+inter;
   XYZ[Np+1,1]:=1.1;   XYZ[Np+1,2]:=0.5;   XYZ[Np+1,3]:=0.0;   Name[Np+1]:='theta';
   Np:=Np+1;
(
                phi
)
   inter:=10;
   For i:=0 to inter do
      begin
      XYZ[Np+1+i,1]:=sqrt(sqr(Sin(i/inter*Pi/3))/2);
      XYZ[Np+1+i,2]:=XYZ[Np+1+i,1];
      XYZ[Np+1+i,3]:=Cos(i/inter*Pi/3.5);
      Name[Np+1+i]:='';
      end;
   For i:=1 to inter do
      begin
      Dep[Nt+i]:=Np+i;
      Fin[Nt+i]:=Np+i+1;
      end;
   Np:=Np+inter+1;
   Nt:=Nt+inter;
   XYZ[Np+1,1]:=0.5;   XYZ[Np+1,2]:=0.55; XYZ[Np+1,3]:=1.1;   Name[Np+1]:='phi';
   Np:=Np+1;
(
                r theta phi coordinates
```

```
}
   XYZ[Np+1,1]:=0.0;  XYZ[Np+1,2]:=0.0;  XYZ[No+1,3]:=0.0;  Name[Np+1]:='';
   XYZ[Np+2,1]:=1.7;  XYZ[Np+2,2]:=1.7;  XYZ[Np+2,3]:=0.0;  Name[Np+2]:='';
   XYZ[Np+3,1]:=1.7;  XYZ[Np+3,2]:=1.7;  XYZ[Np+3,3]:=1.7;  Name[Np+3]:='';
   XYZ[Np+4,1]:=1.7;  XYZ[Np+4,2]:=1.8;  XYZ[Np+4,3]:=1.8;  Name[Np+4]:='P';
   XYZ[Np+5,1]:=1.2;  XYZ[Np+5,2]:=1.2;  XYZ[Np+5,3]:=1.3;  Name[Np+5]:='r';
   Dep[Nt+1]:=Np+1;  Fin[Nt+1]:=Np+2;
   Dep[Nt+2]:=Np+2;  Fin[Nt+2]:=Np+3;
   Dep[Nt+3]:=Np+1;  Fin[Nt+3]:=Np+3;
   Np:=Np+5;
   Nt:=Nt+3;
   Two('Local SPHERICAL coordinate system P(r,theta,phi)',
       XYZ,Np,Dep,Fin,Name,Nt,1000,1000,30,60,Action);
end;
{-------------------------------------------------------------------------}
Procedure L_LOCAL(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'LOCAL\';
   Repeat
      Header(HelpPath);
      writeln('LOCAL LocalType,x0,y0,[z0],L1,M1,[N1],L2,M2,[N2];');
      writeln('   LocalType (S): Type of local coordinate system:');
      writeln('R-    R[ectangular] for a rectangular coordinate system (x,y,z).');
      writeln('C-    C[ylindrical] for a cylindrical coordinate system (r,theta,z).');
      writeln('S-    S[pherical] for a spherical coordinate system (r,theta,phi).');
      writeln('      r: distance from origin of local coordinate system.');
      writeln('      theta: angle (radian) formed by r projection on X''-Y'' plane and X' .');
      writeln('      phi: angle (radian) formed by r direction and local Z'' axis.');
      writeln('   x0,y0,z0 (IR): local coordinate system origin in global coordinate system.');
      writeln('L- L1,M1,N1 (IR): direction of local X'' axis.');
      writeln('   L2,M2,N2 (IR): direction of local Y'' axis (tentative).');
      writeln('      Z'' axis is set as normal to X''-Y'' plane.');
      writeln('      Then Y'' is corrected as normal to Z''-X'' plane.');
      writeln('example:');
      writeln('   LOCAL Cyl,32,0,,1,0,,0,1;'),
      writeln('   --> Cylindrical coordinate system with origin at (32,0,0)');
      writeln('       and axis direction the same as global coordinate system.');
      writeln('note: By default the local coordinate system is the same as the global.');
      writeln('      The local coordinate system remains valid until changed.');
      GoBack(HelpPath,long,C,Action);
      If C='L' then Orientation_Plot;
      If C='R' then Rectangular_Plot;
      If C='C' then Cylindrical_Plot;
      If C='S' then Spherical_Plot;
      until Action='ESC';
end;
```

```
{-------------------------------------------------------------------------}
Procedure L_MAT(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'MAT\';
   Repeat
      Header(HelpPath);
      writeln('MAT Number,MaterialName;');
      writeln('   Number (I): material number. Numbering can be discontinuous. 50 max.');
      writeln('   MaterialName (S): Name of material (should match database).');
      writeln('example:');
      writeln('   MAT 1,T300/5208 Graphite/Epoxy;');
      writeln('   --> Graphite/epoxy is defined as material 1');
      writeln('Note: When the material is looked up in the database, the');
      writeln('   blanks are taken out of MaterialName and those in database.');
      writeln('   Upper or lower case is ignored.');
      writeln('   The MaterialName can be a subset of the complete name, by');
      writeln('   using the first characters.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_NGEN(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'NGEN\';
   Repeat
      Header(HelpPath);
      writeln('NGEN FromNode,[ToNode],Repeat,[dNode],[dx],[dy],[dz];');
      writeln('   FromNode (I): nodes will be generated from a pattern starting');
      writeln('      with node number FromNode.');
      writeln('   ToNode (I): nodes will be generated from a pattern ending');
      writeln('      with node number ToNode. Default: ToNode=FromNode.');
      writeln('   Repeat (I): number of times the given pattern is repeated');
      writeln('      excluding the original pattern. Range: 1...N.');
      writeln('   dNode (I): increase in node number for each new pattern.');
      writeln('      Default: dNode=ToNode-FromNode+1. This places the new nodes after');
      writeln('      node ToNode. Restriction: dNode > ToNode-FromNode.');
      writeln('   dx,dy,dz (IR): increase in local coordinates for each new pattern.');
      writeln('      Default: dx=dy=dz=0. See \Language\LOCAL\.');
      writeln('example:');
      writeln('   NGEN 3,5,2,6,,-0.3;');
```

```
      writeln('   --> Generate 2 new patterns by imitating nodes 3 to 5 and increasing');
      writeln('       node numbers by 6 and Y coordinate by -0.3 each time.');
      writeln('       Result: nodes 9,10,11, 15,16,17 are defined.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_NODE(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'NODE\';
   Repeat
      Header(HelpPath);
      writeln('NODE Number,[x],[y],[z];');
      writeln('   Number (I): the node number.');
      writeln('      Your node numbering can be discontinuous.');
      writeln('   x,y,z (IR): Local coordinates of your node. Default: x=y=z=0.');
      writeln('      See \Language\LOCAL\.');
      writeln('      When defining a 2D problem, you can give only x and y.');
      writeln('example:');
      writeln('   NODE 34,1.35,,-3.4;');
      writeln('   --> Node 34 is defined as x=1.35, y=0.0, z=-3.4');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_OLD(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'OLD\';
   Repeat
      Header(HelpPath);
      writeln('OLD CNodeStart,[CNodeEnd],[CNodeInc],[Repeat],[CPatternInc],');
      writeln('                [NodeStart],[NodeInc],[PatternInc];');
      writeln('Use: Using a solved case for node coordinate input wit' displacements.');
      writeln('   CNodeStart (I): Starting node number in coarse mesh to copy from.');
      writeln('   CNodeEnd (I): Ending node number in coarse mesh to copy from.');
      writeln('      Default: CNodeEnd = CNodeStart.');
      writeln('   CNodeInc (I): Increment inside CNodeStart to CNodeEnd for nodes used.');
      writeln('      Default: CNodeInc = 1.');
      writeln('   Repeat (I): Number of times we repeat this pattern. Default: Repeat = 0.');
      writeln('   CPatternInc (I): Increment in coarse node number mesh for each pattern.');
```

```
      writeln('      Default: CPattern = CNodeEnd - CNodeStart + CNodeInc.');
      writeln('   NodeStart (I): Starting node number copied. Default: NodeStart = CNodeStart.');
      writeln('   NodeInc (I): Increment in copied node numbers. Default: NodeInc = 1.');
      writeln('   PatternInc (I): Increment for node numbers in copied pattern.');
      writeln('      Default: PatternInc = (CPatternInc/CNodeInc)*NodeInc.');
      writeln('example:');
      writeln('   OLD 1,7,2,,,1,4;');
      writeln('   --> Copies nodes 1,3,5,7 in Coarse mesh already solved, to nodes');
      writeln('       1,5,9,13 in current case. Coordinates and displacement are passed.');
      writeln('       This command is used to apply boundary conditions for a mesh refinement.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_OPER(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'OPER\';
   Repeat
      Header(HelpPath);
      writeln('OPER VariableName,[A],Operator,B;');
      writeln('   VariableName (S): Name of variable.');
      writeln('      = A Operator(B)   /0 Ln(x<=0) Tan(Pi/2) Sqrt(x<0) are checked');
      writeln('   A (IR): Value A. Default: A=1.');
      writeln('   Operator (S): + - * /');
      writeln('      *Abs:     Absolute value');
      writeln('      *Cos *Sin *Tan *ArcTan:  Cosine, Sine, Tangent and ArcTangent (radians)');
      writeln('      *Exp *Exp10:  Natural (base 2.7182818...) and base 10 exponential');
      writeln('      *Ln *Log10:   Natural and base 10 logarithm (0<B)');
      writeln('      *Round:   Round a real number to an integer');
      writeln('      *Sqr:     Square');
      writeln('      *Sqrt:    Square root (0<=B)');
      writeln('      *Trunc:   Truncation of a real to inferior integer number');
      writeln('   B (IR): Value B.');
      writeln('Note: Redefinition attempts will be signaled but ignored.');
      writeln('   It is possible to pass variables through parameters when');
      writeln('   the PREP command is invoqued. See \Batch mode\PREP\.');
      writeln('examples:');
      writeln('   Oper DoubleAngle,2,*,Angle; ');
      writeln('      Variable "DoubleAngle" = double the value of "Angle"');
      writeln('   Oper Root,3.1,*Sqrt,36;   -> Root = 3.1*6 = 18.6');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_REPEAT(HelpPath: String255);
var
```

```
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'REPEAT\';
   Repeat
      Header(HelpPath);
      writeln('REPEAT AgainTimes,[d1],[d2],[d3],...;');
      writeln('   AgainTimes (I): Number of times the following command will');
      writeln('      be repeated (excluding the command itself).');
      writeln('   di (IR): increments for parameters in the following command.');
      writeln('      They must be of same type as in the following command.');
      writeln('      The numbers of increased parameters should equal the');
      writeln('      number of parameters in the following command.');
      writeln('      Do not give a string as an increment.');
      writeln('example:');
      writeln('   REPEAT 2,1,   ,1.0; ');
      writeln('        NODE 4,0.0,0.0; ');
      writeln('   --> Node 4 is defined as x=0.0, y=0.0');
      writeln('        Node 5 is defined as x=0.0, y=1.0');
      writeln('        Node 6 is defined as x=0.0, y=2.0');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
Procedure L_TRAC(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'TRAC\';
   Repeat
      Header(HelpPath);
      writeln('TRAC FromElement,[ToElement],[Repeat],[dElement],Face,Traction;');
      writeln('used to set face or line tractions.');
      writeln('   FromElement (I): First element considered.');
      writeln('   ToElement (I): Last element considered.');
      writeln('      Default: ToElement=FromElement.');
      writeln('   Repeat (I): number of time pattern is repeated. Range: 0...N. Default: 0');
      writeln('      Pattern is FromElement to ToElement element numbers.');
      writeln('   dElement (I): step for element numbers between each pattern.');
      writeln('      Default: ToElement-FromElement+1');
      writeln('   Face (I): face number. See \Features\Elements\');
      writeln('   Traction (IR): traction on face or line boundary of element.');
      writeln('      For a 3-D element: Traction = -Pressure.');
      writeln('example:');
      writeln('   TRAC 3,,2,8,6,1E6;');
```

```
      writeln('   --> set stress of 1E6 on face 6 of element 3, 11 and 19.');
      writeln('note: this command resets nodes on given face to known forces.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure L_VAR(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'VAR\';
   Repeat
      Header(HelpPath);
      writeln('VAR VariableName,Value;');
      writeln('   VariableName (S): Name of variable (',Var_max,' max).');
      writeln('   Value (IRS): Value for variable.');
      writeln('Note: Redefinition attempts will be signaled but ignored.');
      writeln('   It is possible to pass variables through parameters when');
      writeln('   the PREP command is invoqued. See \Batch mode\PREP\.');
      writeln('example:');
      writeln('   VAR Angle,10.3; ');
      writeln('       Variable ANGLE is set to 10.3');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure View2Dexample;
var
   Inter,i,j,Npoint,Ntrait: Integer;
   XYZ: MatrixR_PointxD;
   Dep,Fin: VectI_Ncon;
   Name: VectS5_Point;
   Action: String6;
begin
   Beep;
   ClrScr;
   For i:=1 to Npoint_max do
      begin
      For j:=1 to 3 do
         XYZ[i,j]:=0;
      Name[i]:='';
      end;
{
                  Define Geometry
}
   XYZ[2,1]:=4.0;   XYZ[2,2]:=0.0;   Name[2]:='';
   XYZ[4,1]:=0.0;   XYZ[4,2]:=0.0;   Name[4]:='';
   XYZ[8,1]:=2.0;   XYZ[8,2]:=0.0;   Name[8]:='';
```

```
   XYZ[10,1]:=1.0;  XYZ[10,2]:=2.0;  Name[10]:='';
   XYZ[11,1]:=3.0;  XYZ[11,2]:=2.0;  Name[11]:='';
   XYZ[12,1]:=3.8;  XYZ[12,2]:=-0.2; Name[12]:='2'; (offset labels)
   XYZ[13,1]:=0.2;  XYZ[13,2]:=-0.2; Name[13]:='4';
   XYZ[14,1]:=2.0;  XYZ[14,2]:=-0.2; Name[14]:='8';
   XYZ[15,1]:=1.0;  XYZ[15,2]:=2.2;  Name[15]:='10';
   XYZ[16,1]:=3.2;  XYZ[16,2]:=2.0;  Name[16]:='11';
   Npoint:=16;
   Dep[1]:=4;  Fin[1]:=8;
   Dep[2]:=8;  Fin[2]:=2;
   Dep[3]:=2;  Fin[3]:=11;
   Dep[4]:=11; Fin[4]:=10;
   Dep[5]:=10; Fin[5]:=4;
   Dep[6]:=8;  Fin[6]:=10;
   Dep[7]:=8;  Fin[7]:=11;
   Ntrait:=7;
{
                Define axis
}
   XYZ[Npoint+1,1]:=0.0;  XYZ[Npoint+1,2]:=0.0;  Name[Npoint+1]:='';
   XYZ[Npoint+2,1]:=5.0;  XYZ[Npoint+2,2]:=0.0;  Name[Npoint+2]:='';
   XYZ[Npoint+3,1]:=0.0;  XYZ[Npoint+3,2]:=3.0;  Name[Npoint+3]:='';
   XYZ[Npoint+4,1]:=5.2;  XYZ[Npoint+4,2]:=0.0;  Name[Npoint+4]:='X';
   XYZ[Npoint+5,1]:=0.0;  XYZ[Npoint+5,2]:=3.2;  Name[Npoint+5]:='Y';
   Dep[Ntrait+1]:=Npoint+1;  Fin[Ntrait+1]:=Npoint+2;
   Dep[Ntrait+2]:=Npoint+1;  Fin[Ntrait+2]:=Npoint+3;
   Npoint:=Npoint+5;
   Ntrait:=Ntrait+2;
{
                Define Load
}
   XYZ[Npoint+1,1]:=3.0;  XYZ[Npoint+1,2]:=2.5;  Name[Npoint+1]:='';
   XYZ[Npoint+2,1]:=3.0;  XYZ[Npoint+2,2]:=2.1;  Name[Npoint+2]:='';
   XYZ[Npoint+3,1]:=2.95; XYZ[Npoint+3,2]:=2.2;  Name[Npoint+3]:='';
   XYZ[Npoint+4,1]:=3.05; XYZ[Npoint+4,2]:=2.2;  Name[Npoint+4]:='';
   XYZ[Npoint+5,1]:=3.0;  XYZ[Npoint+5,2]:=2.7;  Name[Npoint+5]:='P';
   Dep[Ntrait+1]:=Npoint+1;  Fin[Ntrait+1]:=Npoint+2;
   Dep[Ntrait+2]:=Npoint+2;  Fin[Ntrait+2]:=Npoint+3;
   Dep[Ntrait+3]:=Npoint+2;  Fin[Ntrait+3]:=Npoint+4;
   Npoint:=Npoint+5;
   Ntrait:=Ntrait+3;
{
                Define Element labels
}
   XYZ[Npoint+1,1]:=1.0;  XYZ[Npoint+1,2]:=0.7;  Name[Npoint+1]:='E 1';
   XYZ[Npoint+2,1]:=3.0;  XYZ[Npoint+2,2]:=0.7;  Name[Npoint+2]:='E 2';
   XYZ[Npoint+3,1]:=2.0;  XYZ[Npoint+3,2]:=1.3;  Name[Npoint+3]:='E 3';
   Npoint:=Npoint+3;
{
                Define left fixed edge
}
   XYZ[Npoint+1,1]:=-0.5; XYZ[Npoint+1,2]:=0.0;  Name[Npoint+1]:='';
```

```
   XYZ[Npoint+2,1]:=0.0;  XYZ[Npoint+2,2]:=0.0;  Name[Npoint+2]:='';
   XYZ[Npoint+3,1]:=0.0;  XYZ[Npoint+3,2]:=-0.5; Name[Npoint+3]:='';
   Dep[Ntrait+1]:=Npoint+1;  Fin[Ntrait+1]:=Npoint+2;
   Dep[Ntrait+2]:=Npoint+2;  Fin[Ntrait+2]:=Npoint+3;
   Npoint:=Npoint+3;
   Ntrait:=Ntrait+2;
{
                Define right fixed edge
}
   XYZ[Npoint+1,1]:=4.0;  XYZ[Npoint+1,2]:=0.0;  Name[Npoint+1]:='';
   XYZ[Npoint+2,1]:=4.0;  XYZ[Npoint+2,2]:=-0.5; Name[Npoint+2]:='';
   Dep[Ntrait+1]:=Npoint+1;  Fin[Ntrait+1]:=Npoint+2;
   Npoint:=Npoint+2;
   Ntrait:=Ntrait+1;
{
                Define left fixed edge hatch
}
   Inter:=7;
   For i:=1 to Inter do
      begin
      xyz[Npoint+1,1]:=-i/Inter*0.5;
      xyz[Npoint+1,2]:=0;
      xyz[Npoint+2,1]:=0;
      xyz[Npoint+2,2]:=-i/Inter*0.5;
      Dep[Ntrait+1]:=Npoint+1;
      Fin[Ntrait+1]:=Npoint+2;
      Npoint:=Npoint+2;
      Ntrait:=Ntrait+1;
      end;
{
                Define right fixed edge hatch
}
   Inter:=7;
   For i:=1 to Inter do
      begin
      xyz[Npoint+1,1]:=4+i/Inter*0.5;
      xyz[Npoint+1,2]:=0;
      xyz[Npoint+2,1]:=4;
      xyz[Npoint+2,2]:=-i/Inter*0.5;
      Dep[Ntrait+1]:=Npoint+1;
      Fin[Ntrait+1]:=Npoint+2;
      Npoint:=Npoint+2;
      Ntrait:=Ntrait+1;
      end;
   Two('Geometry for EXAMPLE.GEN',XYZ,Npoint,Dep,Fin,Name,Ntrait,
       1000,1000,-90,0,Action);
end;
{-------------------------------------------------------------------------}
Procedure L_2D_example(HelpPath: String255);
var
   C: char;
   Action: String6;
```

```
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Example\';
   Repeat
      Header(HelpPath);
      writeln('Here is a simple problem in 2 dimensions:');
      writeln('   using an isotropic material of   E = 1 GPa,  nu = 0.3');
      writeln('   and having   P = 1 N');
      writeln('G- Geometry plot');
      writeln('The problem can be solved by writing in file EXAMPLE.GEN:');
      writeln('   +---------- Top of file ----------+');
      writeln('   ¦ Var Node2,2;           { Set variable Node2 equal to 2   }¦');
      writeln('   ¦ Oper Node4,2,*,Node2;  { Set Node4 = 2*Node2 = 4         }¦');
      writeln('   ¦ Node Node2,4,0;        { Set node 2                      }¦');
      writeln('   ¦ Ngen Node2,,1,2,-4;    { Generate node 4 from 2          }¦');
      writeln('   ¦ Ngen Node4,,1,4,2;     { Generate node 8 from 4          }¦');
      writeln('   ¦ Node 10,1,2;           { Set node 10                     }¦');
      writeln('   ¦ Ngen 10,,1,,2;         { Generate node 11 from 10        }¦');
      writeln('   ¦                        { You can leave empty lines       }¦');
      writeln('   ¦ Mat 1,Exercise Steel 1; { Properties are found in database }¦');
      writeln('   ¦                        {    see: \Main menu\MATERIAL\    }¦');
      writeln('   ¦ Elem 1,1,1,,4,8,10;    { Element 1                       }¦');
      writeln('   ¦ Elem 2,1,1,0,2,11,8;   { Element 2                       }¦');
      writeln('   ¦ Elem 3,1,1,,8,11,10;   { Element 3                       }¦');
      writeln('   ¦ Displ Node2,,,,0,0;    { Node 2 is fixed: Dx=Dy=0        }¦');
      writeln('   ¦ Displ 4,,,,0,0;        { Node 4 is fixed: Dx=Dy=0        }¦');
      writeln('   ¦ Force 11,,,,,-1;       { Node 11 has Fy=-1               }¦');
      writeln('   +---------- Bottom of file ----------+');
      GoBack(HelpPath,long,C,Action);
      If C='G' then View2Dexample;
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Language;
{
Procedure Language(HelpPath: String255);
}
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Language\';
   Repeat
      Header(HelpPath);
      writeln('A- DISPL    ¦  { curly braces are used for comments }');
      writeln('B- EDEL     ¦  All commands are ended with a ; (more than a line)');
      writeln('C- EGEN     ¦  Commands can be written in upper or lower case.');
```

```
      writeln('D- ELEM     ¦  [brackets] indicate an optional statement.');
      writeln('E- EMOD     ¦  (I) is indicated when an Integer should be used.');
      writeln('F- FORCE    ¦  (R) ... for a real.');
      writeln('G- INTER    ¦  (S) ... for a string of 80 characters maximum.');
      writeln('H- LOCAL    ¦');
      writeln('I- MAT      ¦');
      writeln('J- NGEN     ¦');
      writeln('K- NODE     ¦');
      writeln('L- OLD      ¦');
      writeln('M- OPER     ¦');
      writeln('N- REPEAT   ¦');
      writeln('O- TRAC     ¦');
      writeln('P- VAR      ¦');
      writeln('Note:');
      writeln('   Nodes, element, materials, forces and displacements can be redefined.');
      writeln('   In the event of any error the program will indicate where ');
      writeln('      and why it has occurred in your file. Then use your favorite');
      writeln('      word processor to modify accordingly (see \Main menu\Edit\).');
      writeln('   It is a good practice to define displacements last.');
      writeln('S- Sample 2 dimensional problem');
      GoBack(HelpPath,long,C,Action);
      If C='A' then L_Displ(HelpPath);
      If C='B' then L_Edel(HelpPath);
      If C='C' then L_Egen(HelpPath);
      If C='D' then L_Elem(HelpPath);
      If C='E' then L_Emod(HelpPath);
      If C='F' then L_Force(HelpPath);
      If C='G' then L_Inter(HelpPath);
      If C='H' then L_Local(HelpPath);
      If C='I' then L_Mat(HelpPath);
      If C='J' then L_Ngen(HelpPath);
      If C='K' then L_Node(HelpPath);
      If C='L' then L_Old(HelpPath);
      If C='M' then L_Oper(HelpPath);
      If C='N' then L_Repeat(HelpPath);
      If C='O' then L_Trac(HelpPath);
      If C='P' then L_Var(HelpPath);
      If C='S' then L_2d_example(HelpPath);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
end.
```

H_MAIN.PAS

```
{$N+}
Unit H_Main;
Interface
Uses
   Declare,H_Common,Sonore;
Procedure MainHelp(HelpPath: String255);
Procedure MainMenu(HelpPath: String255);
{-------------------------------------------------------------------}
Implementation
Procedure StatusLines(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Status lines\';
   Repeat
      Header(HelpPath);
      writeln('The status lines gives:');
      writeln('   - the file name of the current case,');
      writeln('   - the case used as a coarse mesh for latter refinement');
      writeln('Some processes depend upon the completion of others.');
      writeln('As the current case is analyzed, a status is given to the right');
      writeln('   of the processes.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------}
Procedure MainHelp;
{
Procedure MainHelp(HelpPath: String255);
}
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Help on HELP\';
   Repeat
      Header(HelpPath);
      writeln('This HELP program is structured like a book with chapters and sub-chapters.');
      writeln('The first line gives the current location inside HELP.');
      writeln('Further sub-chapters are indicated by X- in the first column.');
      writeln('All processes available from the main menu are capitalized.');
      writeln('Because this program is in a developing stage, a complete');
      writeln('   help program was preferred over a manual to permit faster');
```

```
      writeln('   adjustments to programming improvements.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainConfigure(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'CONFIGURE\';
   Repeat
      Header(HelpPath);
      writeln('CONFIGURE stores the directories of each type of file. ');
      writeln('Further help is available within that process.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainMaterial(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'MATERIAL\';
   Repeat
      Header(HelpPath);
      writeln('MATERIAL is a database program containing all the information');
      writeln('   pertinent to each material (up to 50). Some materials are');
      writeln('   already set for you.');
      writeln('Note: In the preprocessing stage, the material properties are');
      writeln('   looked up with the material name as a reference, after which');
      writeln('   the number in the database is kept for each element. So you');
      writeln('   should not change the material from one place to another in');
      writeln('   the database when analysing a case preprocessed earlier.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainEdit(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
```

```
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'EDIT\';
   Repeat
      Header(HelpPath);
      writeln('EDIT is used to edit your geometry definition files (*.GEN).');
      writeln('A particular feature here, is using your favorite word processor');
      writeln('   from within this program to edit the *.GEN files.');
      writeln('This program then acts as a "parent" process, where your word');
      writeln('   processor becomes a "child" process.');
      writeln('After you exit your word processor you will conveniently return');
      writeln('   to the main menu.');
      writeln('For this you should use CONFIGURE to install your word');
      writeln('   processor''s name (with extension) and directory.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainPrep(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'PREP\';
   Repeat
      Header(HelpPath);
      writeln('PREPROCESS compiles geometry definition files.');
      writeln('All previous *.P files are erased (see \Files\). ');
      writeln('You choose your file from a *.GEN directory.');
      writeln('The file is first run through to find and verify the compatibility');
      writeln('   between the different element types used. At the same time');
      writeln('   the number of dimensions used in the geometry is found.');
      writeln('   The maximum number of nodes per element is retained.');
      writeln('A complete error checking is done on all commands.');
      writeln('The size of the files used for data is expanded as needed.');
      writeln('The node indexing is compressed to skip gaps left in the node');
      writeln('  numbering. All concerned data files are shuffle and truncated.');
      writeln('Element numbering is also compressed.');
      writeln('The connections between nodes for plots are found.');
      writeln('The number of times a node is used in different elements is counted.');
      writeln('Files necessary to other processes are written (*.P).');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainNumbers(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
```

```
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'NUMBERS\';
   Repeat
      Header(HelpPath);
      writeln('NUMBERS shows you some numbers pertaining to the current case');
      writeln('   and the computer.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainView(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'VIEW\';
   Repeat
      Header(HelpPath);
      writeln('VIEW is used to see all or part of your geometry.');
      writeln('The lines show the connections between the nodes arising from');
      writeln('   the element construction.');
      writeln('This can be used as a verification step.');
      writeln('All geometries are viewed in 3 dimensions.');
      writeln('You can specify any particular viewpoint to look at your model.');
      writeln('Using the PrintScreen key will give you a hard copy.');
      writeln('Perspective has been taken out to leave an undistorted shape.');
      writeln('You should set:  Draw elevation = No');
      writeln('All the selection data is stored for next time.');
      writeln('All plots are automatically scaled to fit the screen without');
      writeln('   distortion.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainSolve(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'SOLVE\';
   Repeat
      Header(HelpPath);
      writeln('A "dry run" of the solving process is done to get the maximum wave front');
      writeln('   size and for an accurate prediction of the necessary disk space.');
```

```
      writeln('SOLVE uses the "Front Solver" method in order to reduce the amount');
      writeln('  of nodes considered at a same time.');
      writeln('  As the solution process is active, the program looks for');
      writeln('  nodes which will not be involved as other elements are ');
      writeln('  added to the global matrix. These nodes are then stored');
      writeln('  in a file along with their relation to the other nodes.');
      writeln('  When all elements are finally treated, the solution is');
      writeln('  back-substituted into the previously stored node equations.');
      writeln('The matrix defined by the wave front is stored by sub-matrices');
      writeln('  which consist of a 2x2 or 3x3 matrices depending on the');
      writeln('  number of dimensions for the problem. While solving the');
      writeln('  linear equation of the front only 2 matrices and a matrix');
      writeln('  of similar size for the multipliers has to be present at the');
      writeln('  same time in memory.');
      writeln('Nodes restrained in all direction are reduced first for speed.');
      writeln('If this process takes over a minute, a beep is heard when finished.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
Procedure MainStress(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'STRESS\';
   Repeat
      Header(HelpPath);
      writeln('The stresses and strains are averaged at each node by considering');
      writeln('   the results from each element including it.');
      writeln('For each element, the stresses are calculated at the optimal flux');
      writeln('   sampling points (gauss-quadrature locations). Then they are');
      writeln('   extrapolated at the nodes.');
      writeln('RESULT is the only process which requires these results. And RESULT');
      writeln('   will still display Displacements and Forces without them.');
      writeln('Note: The strains are tensorial strains.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------------}
Procedure MainWorst(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'WORST\';
```

```
   Repeat
      Header(HelpPath);
      writeln('WORST will find the location inside the node which is closest');
      writeln('   to failure depending on the strength of the materials.');
      writeln('The simplest failure criterion is applied: each stress is');
      writeln('   compared to it''s maximum given in the database.');
      writeln('The on-axis stresses are calculated at the optimal sampling');
      writeln('   points (gauss points) and extrapolated to the nodes of');
      writeln('   the element. Theses stresses are not averaged using the');
      writeln('   other elements node stresses.');
      writeln('Negative stresses are compared with the compressive strengths.');
      writeln('Only the absolute value of the shear stresses is considered.');
      writeln('Once the worst element is found it''s number is saved in a file.');
      writeln('   If WORST is called again, only that element is evaluated.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainDeform(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'DEFORM\';
   Repeat
      Header(HelpPath);
      writeln('DEFORM shows the deformed geometry.');
      writeln('The displacements are exaggerated in order to be seen.');
      writeln('The ratio of exaggeration is given in the graph''s title.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainResult(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'RESULT\';
   Repeat
      Header(HelpPath);
      writeln('RESULT is used to plot the strain, stress, force or');
      writeln('   displacement within a "slice" of your geometry.');
      writeln('You can also use it to list those value or find the extremes.');
      writeln('Plotting process:');
      writeln('   A plane R is fitted by regression to all the chosen nodes.');
```

```
      writeln('   All node are projected on the plane R.');
      writeln('   A new coordinate system with X'' and Y'' in plane R is found.');
      writeln('   Y'' is in the plane formed by the normal of plane R and Z.');
      writeln('   The Y'' axis is oriented towards positive Z.');
      writeln('   A third dimension is created from the results you want to see');
      writeln('      at the chosen nodes.');
      writeln('   Your results are viewed as a landscape elevation in X'' Y'' Z''.');
      writeln('The lines show the connections between the nodes arising from');
      writeln('   the element construction.');
      writeln('You can specify any particular viewpoint to look at your model.');
      writeln('Pressing the <PrtSc> key will give you a hard copy.');
      writeln('Perspective has been taken out to leave an undistorted shape.');
      writeln('All the selection data is stored for next time.');
      writeln('All plots are automatically scaled to fit the screen without');
      writeln('   distortion.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainManager(HelpPath: String255);
var
   C: char;
   Actior: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'MANAGER\';
   Repeat
      Header(HelpPath);
      writeln('MANAGER is used to:');
      writeln('   Save the current case with all selection menus and solution');
      writeln('   Restore a previous case');
      writeln('   Keep current solved case for further mesh refinement');
      writeln('   Delete a previous case');
      writeln('Further help is given within that process.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainExit(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'EXIT\';
   Repeat
      Header(HelpPath);
      writeln('Why do you need help on Exit?');
```

```
      writeln('To make sure you know that your *.P files (see \Files\) will');
      writeln('   be erased the next time someone calls PREPROCESS.');
      writeln('So why not save those precious files in a safe place?');
      writeln('Use MANAGER to save all pertinent data of the solved case.');
      writeln('To free some space on the hard disk, store your data on floppy');
      writeln('   disks using the FILL.EXE program. Typing: Fill<Enter> from ');
      writeln('   the DOS prompt will give you further help on the command.');
      writeln('   Basically, it''s a COPY command that checks for available');
      writeln('   space on the floppy disks.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure MainMenu;
{
Procedure MainMenu(HelpPath: String255);
}
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Main menu\';
   Repeat
      Header(HelpPath);
      writeln('The main menu consists of:');
      writeln('L- Status lines');
      writeln('H- HELP');
      writeln('C- CONFIGURE');
      writeln('A- MATERIAL');
      writeln('E- EDIT');
      writeln('P- PREPROCESS');
      writeln('N- NUMBERS');
      writeln('V- VIEW');
      writeln('S- SOLVE');
      writeln('T- STRESS');
      writeln('W- WORST');
      writeln('D- DEFORM');
      writeln('R- RESULT');
      writeln('M- MANAGER');
      writeln('X- EXIT');
      GoBack(HelpPath,long,C,Action);
      If C='L' then StatusLines(HelpPath);
      If C='H' then MainHelp(HelpPath);
      If C='C' then MainConfigure(HelpPath);
      If C='A' then MainMaterial(HelpPath);
      If C='E' then MainEdit(HelpPath);
      If C='P' then MainPrep(HelpPath);
      If C='N' then MainNumbers(HelpPath);
      If C='V' then MainView(HelpPath);
```

```
      If C='S' then MainSolve(HelpPath);
      If C='T' then MainStress(HelpPath);
      If C='W' then MainWorst(HelpPath);
      If C='D' then MainDeform(HelpPath);
      If C='R' then MainResult(HelpPath);
      If C='M' then MainManager(HelpPath);
      If C='X' then MainExit(HelpPath);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
end.
```

J_GRAPH.PAS

```
{$N+}
Unit J_Graph;
Interface
Uses
   Graph,Declare;
Procedure Put_Mot(     Mot: String6;
                       X,Y: Rtype);
{-------------------------------------------------------------------------*)
Implementation
Procedure Put_Mot;
{
Procedure Put_Mot(     Mot: String6;
                       X,Y: Rtype);
}
{
   Input:
     Mot: word to be displayed
     X,Y: screen coordinate to center the number
   Output:
     Number is centered at screen coordinates (X,Y)
     for the closest row and column
}
begin
   OutTextXY(Round(X-TextWidth(Mot)/2.0),
             Round(Y-TextHeight(Mot)/2.0),Mot);
end;
{-------------------------------------------------------------------------}
end.
```

K_COMMON.PAS

```
{$N+}
Unit K_Common;
Interface
Uses
   Declare,Crt,File_RW;
Procedure NoString(    IRS: VectI_Nc;
                       Qtt: Itype;
                   var Bad: Boolean);
Procedure StretchNodes(var OriginalNnt: Itype;
                           LargerNnt,Dim: Itype;
                       var F_Fd: FileByte;
                       var F_xyz,F_Force,F_Displ: FileR);
{-------------------------------------------------------------}
Implementation
Procedure NoString;
{
Procedure NoString(    IRS: VectI_Nc;
                       Qtt: Itype;
                   var Bad: Boolean);
}
{
   Input:
     IRS[i]: 0 if the (i)th number in the command line is IType
             1 for a real
             2 for a string
            -1 if not specified, or an error occured in conversion
     Qtt: Quantity of Possible number locations before Semicolon (End
          of command line)
   Output:
     Bad: True if a string is used
          False Otherwise
}
var
   iPar: Itype;
begin
   For iPar:=1 to Qtt do
     If (IRS[iPar]=2)and(Bad=False) then
       begin
       writeln('   Parameter ',iPar,' must not be a string');
       Bad:=True;
       end;
end;
{-------------------------------------------------------------}
Procedure StretchNodes;
{
Procedure StretchNodes(var OriginalNnt: Itype;
                           LargerNnt,Dim: Itype;
                       var F_Fd: FileByte;
                       var F_xyz,F_Force,F_Displ: FileR);
}
```

```
{
   Input:
      OriginalNnt: Previous number of nodes
      LargerNnt: New number of nodes
      Dim: Dimension of problem
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
         XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
         each node
      F_Displ: File with Displacements in U,V,W for each node
   Output:
      OriginalNnt: Previous number of nodes set to LargerNnt
}
var
   iScreen,iD,iNode: Itype;
   GxyzR: VectR_D;
   Xs,Ys,Xs2,Ys2: Byte;
begin
   If (OriginalNnt < LargerNnt) then
      begin
      Xs:=WhereX;
      Ys:=WhereY;
      write('   Initializing new coordinates ');
      Xs2:=WhereX;
      Ys2:=WhereY;
{
                Make F_xyz longer
}
      For iD:=1 to Dim do GxyzR[iD]:=BigRmax;        { initialize as unknown }
{
                Make F_xyz, F_Force, F_Displ, F_Fd longer
}
      For iNode:=OriginalNnt+1 to LargerNnt do        { initialize up to last }
        begin
        gotoXY(Xs2,Ys2);
        write(LargerNnt-iNode+1,' ');
        D_R_write(F_xyz,iNode,Dim,GxyzR);
        D_R_0(F_Force,iNode,Dim);
        D_R_0(F_Displ,iNode,Dim);
        D_Byte_0(F_Fd,iNode,Dim);
        end;
      OriginalNnt:=LargerNnt;
      gotoXY(Xs2,Ys2);
      write('  ');
      gotoXY(Xs,Ys);
      For iScreen:=0 to Ys2*80+Xs2-(Ys*80+Xs) do write(' ');
      gotoXY(Xs,Ys);
      end;
end;
{-------------------------------------------------------------}
```

```
end.
```

K_ELEM.PAS

```
{$N+}
Unit K_Elem;
Interface
Uses
   Crt,Declare,K_Common,Nne_Dim,File_RW,Sonore,
   { For element type 2 distortion in ELEM } Elem2,Exyz,VectMat3;
Procedure Set_elem(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                   var CaseToDatabase: VectByte_Nmat;
                   var F_XYZ: FileR;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                   var Net: IType;
                       NneMax,Nnt,Dim: IType;
                   var Bad: Boolean);
Procedure Set_Egen(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                   var F_xyz: FileR;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                   var Net: IType;
                       NneMax,Nnt,Dim: IType;
                   var Bad: Boolean);
Procedure Set_Edel(    IRS,Ni: VectI_Nc;
                       Qtt: IType;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                       NneMax: IType;
                   var Bad: Boolean);
Procedure Set_Emod(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                   var CaseToDatabase: VectByte_Nmat;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                       Net,NneMax: IType;
                   var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Stretch_Elements(    ElemMax,NneMax: Itype;
                           var F_Elem: FileI;
                           var F_Angle: FileR;
                           var Net: Itype);
{
   Input:
      ElemMax: Maximum element number to be used
      NneMax: Maximum number of nodes in an element
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
```

```
                for each element
                Element type = 0 if undefined
        F_Angle: File With angle of elements (radians)
        Net: Number of elements
    Output:
        F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
                for each element
                Element type = 0 if undefined
        F_Angle: File With angle of elements (radians)
        Net: Number of elements, increased if necessary (if Net<ElemMax)
}
var
    iE: Itype;
    XsO,YsO,Xs,Ys: Byte;
begin
    XsO:=WhereX;
    YsO:=WhereY;
    Write('  Initializing new elements ');
    Xs:=WhereX;
    Ys:=WhereY;
{
                    Initialize Elem and Angle between Old Net and Present
                    Readjust Number of Elements
}
    If Net<ElemMax then
        begin
        For iE:=Net+1 to ElemMax-1 do
            begin
            GotoXY(Xs,Ys);
            write(ElemMax-iE,' ');
            Nne2_I_O(F_Elem,iE,NneMax);
            R_O(F_Angle,iE);
            end;
        Net:=ElemMax;
        end;
    GotoXY(XsO,YsO);
    ClrEOL;
end;
{-------------------------------------------------------------------}
Procedure Set_elem;
{
Procedure Set_elem(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                   var CaseToDatabase: VectByte_Nmat;
                   var F_XYZ: FileR;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                   var Net: IType;
                       NneMax,Nnt,Dim: IType;
                   var Bad: Boolean);
}
```

```
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
             1 for a real
             2 for a string
            -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      CaseToDatabase[iMat]: number in database for material iMat in case
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Net: Number of elements
      NneMax: Maximum number of nodes in an element
      Nnt: Number of nodes
      Dim: Dimension of problem
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Net: Number of elements, increased if necessary
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      ELEM Number,mat,Etype,[angle],node1,node2,node3,[node4]...
      ----------------------------------------------
            i    i    i   r    i         i   (i=integer,r=real)
      a real can be given as an integer, but not the reverse
      angle can be omitted: ,, -> 0 is used  (angle in DEGREES)
      according to Etype=1 there must be 7 parameters, no empty space
                         2              24
}
var
   i,k,iNne,Nne: IType;
   Angle: Rtype;
   xyz: VectR_D;
   Elem: VectI_Nne2;
   SameNode: Boolean;
{ for element distortion checking }
   J_det: Rtype;
   Ngq,iGQ: Byte;
   Exyz: MatrixR_Nnex3;
   BF: MatrixR_6xNneD;
   J: MatrixR_3x3;
   enb: MatrixR_NgqxD;
begin
```

```
{
                Verify type of data, no empty spaces
                Look at Writeln statements for more detail
}
   Bad:=False;
   NoString(IRS,Qtt,Bad);
   If Qtt<5 then
      begin
      Bad:=True;
      writeln('   A minimum of 5 parameters needed');
      end;
   If IRS[1]<>0 then
      begin
      Bad:=True;
      writeln('   Element number must be an integer');
      end;
   If IRS[2]<>0 then
      begin
      Bad:=True;
      writeln('   Material number must be an integer');
      end;
   If CaseToDatabase[Ni[2]]<=0 then
      begin
      Bad:=True;
      writeln('   Material number ',Ni[2],' is undefined');
      end
   else
      Ni[2]:=CaseToDatabase[Ni[2]];  {change for database number}
   If IRS[3]<>0 then
      begin
      Bad:=True;
      writeln('   Element type must be an integer');
      end;
   If (Qtt-4)<>Get_Nne(Ni[3]) then      { Verify Qtt with Number of nodes }
      begin
      Bad:=True;
      writeln('   ',Get_Nne(Ni[3]),' nodes neaded for element type ',Ni[3]);
      end;
   If Bad=False then
      For i:=1 to Qtt-4 do
        If IRS[i+4]<>0 then
           begin
           Bad:=True;
           writeln('   Node number given in position ',
                  i,' must be an integer');
           end;
   If Bad=False then
      For i:=1 to Qtt-4 do
         If Nnt<Ni[i+4] then
            begin
            Bad:=True;
            writeln('   Node number given in position ',
```

```
                i,' is beyond highest node number (',Nnt,')');
            end;
   If Bad=False then
      For i:=1 to Qtt-4 do
         begin
         D_R_read(F_xyz,Ni[i+4],dim,xyz);
         If xyz[1]>BigR then
            begin
            Bad:=True;
            writeln('   Node number given in position ',
                  i,' is undefined');
            end;
         end;
{
                Store Data
}
   If Bad=False then
      begin
      Stretch_Elements(Ni[1],NneMax,F_Elem,F_Angle,Net);
      Elem[1]:=Ni[2];
      Elem[2]:=Ni[3];
      Angle:=0;           ( Angle defaults to 0 )
      Case IRS[4] of
         0: Angle:=Ni[4];
         1: Angle:=Nr[4];
         end;
{
                Convert angle from degrees to radians
}
      Angle:=Angle*Pi/180.0;
      For i:=1 to Qtt-4 do
         Elem[2+i]:=Ni[i+4];
      Nne2_I_Write(F_Elem,Ni[1],NneMax,Elem);
      R_Write(F_Angle,Ni[1],Angle);
{
                Warning when a node is used more than once
}
      Nne:=Qtt-4;
      For iNne:=1 to Nne-1 do
         For k:=iNne+1 to Nne do
            If Elem[2+iNne]=Elem[2+k] then SameNode:=True;
      If SameNode=True then
         begin
         WarningSound;
         TextColor(LightRed+Blink);
         Write('   Warning');
         TextColor(LightGreen);
         writeln(': a node is used more than once');
         end;
      end;
{
                Check element type 2 distorsion level
```

```
                The element distortion will be checked for ALL elements
                   in the solving process
)
   If (Bad=False) and (Elem[2]=2) then
      begin
      write('  Checking distortion ');
      Get_Exyz(Elem,F_xyz,Exyz);
      Get_enb(enb);
      Ngq:=27;
      For iGQ:=1 to Ngq do
         begin
         If Bad=False then
            begin
            Get_BF(enb[iGQ,1],enb[iGQ,2],enb[iGQ,3],Exyz,BF,J);
            Determinant3(J,J_Det);
            If J_Det<0 then
               begin
               Bad:=True;
               writeln;
               writeln('This element is too distorted.');
               writeln('          or');
               writeln('Improper node numbering direction.');
               end;
            end;
         end;
      If Bad=False then
         begin
         GotoXY(1,WhereY);
         ClrEOL;
         end;
      end;
end;
{-------------------------------------------------------------------}
Procedure Set_Egen;
{
Procedure Set_Egen(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                   var F_xyz: FileR;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                   var Net: IType;
                       NneMax,Nnt,Dim: IType;
                   var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
```

```
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Net: Number of elements
      NneMax: Maximum number of nodes in an element
      Nnt: Number of nodes
      Dim: Dimension of problem
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Net: Number of elements, increased if necessary
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      EGEN FromElement,[ToElement],Repeat,[dElement],dNode
      ----------------------------------------------------
             i           i        i       i       i    (i=integer)
      FromElement < ToElement
      (ToElement-FromElement) < dElement  (Step in Element generation)
      if ToElement is omitted ToElement=FromElement
      Repeat: Number of times pattern of element FromElement to ToElement is repeated
      if dElement is omitted dElement = (ToElement-FromElement+1)
}
var
   E_last,iEnew,dNode,iRepete,iE,col,dElement,ToElement,FromElement,Repete,i: IType;
   xyz: VectR_D;
   Elem,Elem0: VectI_Nne2;
   Angle,Angle0: Rtype;
   Xs,Ys: Byte;
begin
   Bad:=False;
   NoString(IRS,Qtt,Bad);
   If Qtt<5 then
      begin
      writeln('   5 parameters neaded');
      Bad:=True;
      end;
{
                  Verify data that MUST be given
}
   If IRS[1]<>0 then
      begin
      writeln('   FromElement must be an integer');
      Bad:=True;
```

```
      end;
   If IRS[3]<>0 then
      begin
      writeln('   Repeat must be an integer');
      Bad:=True;
      end;
   If IRS[5]<>0 then
      begin
      writeln('   dNode must be an integer');
      Bad:=True;
      end;
{
                  Verify that ToElement, dElement are not real numbers
}
   If IRS[2]=1 then
      begin
      writeln('   ToElement must not be a real');
      Bad:=True;
      end;
   If IRS[4]=1 then
      begin
      writeln('   dElement must not be a real');
      Bad:=True;
      end;
{
                  Get dNode
}
   dNode:=Ni[5];
{
                  Get FromElement ToElement Repete dElement
}
   If Bad=False then
      begin
      FromElement:=Ni[1];
      If IRS[2]=0 then ToElement:=Ni[2]
                  else ToElement:=FromElement;
      Repete:=Ni[3];
      If IRS[4]=0 then dElement:=Ni[4]
                  else dElement:=ToElement-FromElement+1;
      If FromElement>ToElement then
         begin
         Bad:=True;
         writeln('   FromElement (',FromElement,') > ToElement (',ToElement,')');
         end;
      If Repete<1 then
         begin
         Bad:=True;
         writeln('   Repeat < 1');
         end;
      end;
{
                  Check element number increment
```

```
}
   If Bad=False then
      begin
      If not (ToElement-FromElement < dElement) then
         begin
         writeln('   dElement must be greater than (ToElement-FromElement)');
         writeln('   to avoid re-definition of elements within original pattern');
         bad:=True;
         end;
      end;
{
                  Create new Elements
}
   If Bad=False then
      begin
      E_last:=ToElement+Repete*dElement;
      Stretch_Elements(E_Last,NneMax,F_Elem,F_Angle,Net);
      write('   Generating elements ');
      Xs:=WhereX;
      Ys:=WhereY;
      iE:=FromElement;
      While (iE<=ToElement)and(Bad=False) do
         begin
         Nne2_I_Read(F_Elem,iE,NneMax,Elem0);
         R_Read(F_Angle,iE,Angle0);
         If Elem0[2]=0 then
            begin
            Bad:=True;
            writeln('   Attempt to use an undefined element (',iE,')');
            end;
         iRepete:=1;
         While (iRepete<=Repete)and(Bad=False) do
            begin
            gotoXY(Xs,Ys);
            write((ToElement-iE+1)*Repete-iRepete+1,' ');
            iEnew:=iE+dElement*iRepete;
            Angle:=Angle0;
            For col:=1 to 2 do
               Elem[col]:=Elem0[col];
            Col:=2+1;
            While (Col<=2+Get_Nne(Elem0[2]))and(Bad=False) do
               begin
               Elem[col]:=Elem0[col]+iRepete*dNode;
{
                   Check if node number is valid
}
               If Nnt<Elem[col] then
                  begin
                  Bad:=True;
                  writeln('   Attempt to use a node number (',Elem[col],
                          ') beyond highest defined node number (',Nnt,')');
                  end;
```

```
                If Bad=False then
                   begin
                   D_R_read(F_xyz,Elem[col],dim,xyz);
                   If xyz[1]>BigR then
                      begin
                      Bad:=True;
                      writeln('   Trying to use an undefined node (',
                              Elem[col],') in element ',iEnew);
                      end;
                   end;
                Inc(Col);
                end;
             If Bad=False then
                begin
                Nne2_I_write(F_Elem,iEnew,NneMax,Elem);
                R_write(F_Angle,iEnew,Angle);
                end;
             Inc(iRepete);
             end;
          Inc(iE);
          end;
       end;
    If Bad=False then
       begin
       gotoXY(1,Ys);
       ClrEOL;
       end;
end;
{-------------------------------------------------------------------}
Procedure Set_Edel;
{
Procedure Set_Edel(    IRS,Ni: VectI_Nc;
                       Qtt: IType;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                       NneMax: IType;
                   var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      NneMax: Maximum number of nodes in an element
```

```
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      EDEL Number;
      ------------
            i   (i=integer,r=real)
      Deletes element "Number"
}
var
   Elem: VectI_Nne2;
begin
   Bad:=False;
   NoString(IRS,Qtt,Bad);
   If Qtt<>1 then
      begin
      Bad:=True;
      writeln('  1 parameter needed');
      end;
   If IRS[1]<>0 then
      begin
      Bad:=True;
      writeln('  Element number must be an integer');
      end;
{
                  Delete element (Element type = 0)
}
   If Bad=False then
      begin
      Nne2_I_O(F_Elem,Ni[1],NneMax);
      R_O(F_Angle,Ni[1]);
      end;
end;
{-------------------------------------------------------------------}
Procedure Set_Emod;
{
Procedure Set_Emod(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                   var CaseToDatabase: VectByte_Nmat;
                   var F_Elem: FileI;
                   var F_Angle: FileR;
                       Net,NneMax: IType;
                   var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
```

```
            1 for a real
            2 for a string
           -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      CaseToDatabase[iMat]: number in database for material iMat in case
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Net: Number of elements
      NneMax: Maximum number of nodes in an element
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      Emod Number,[mat],[angle];
      ------------------------------------------
            i     i     ir
      a real can be given as an integer, but not the reverse
      mat and angle (degrees) can be omitted: same as before
}
var
   i,ElemNumber,MatNumber: IType;
   Angle: Rtype;
   Elem: VectI_Nne2;
begin
{
                  Verify data quantity
}
   Bad:=False;
   If Qtt=1 then
      begin
      WarningSound;
      TextColor(LightRed+Blink);
      Write('   Warning');
      TextColor(LightGreen);
      writeln(': nothing changed');
      end;
   If not(Qtt in [1..3]) then
      begin
      Bad:=True;
      writeln('   1 to 3 parameters needed');
      end;
   If Qtt<2 then   {set mat as unspecified}
      begin
```

```
      IRS[2]:=-1;
      Qtt:=2;
      end;
   If Qtt<3 then   {set angle as unspecified}
      begin
      IRS[3]:=-1;
      Qtt:=3;
      end;
{
                  Verify type of data
}
   NoString(IRS,Qtt,Bad);
   If IRS[1]=0 then
      ElemNumber:=Ni[1]
   else
      begin
      Bad:=True;
      writeln('   Element number must be an integer');
      end;
   If ((IRS[2]<>-1) and (IRS[2]<>0)) then
      begin
      Bad:=True;
      writeln('   Material number must be an integer or be unspecified');
      end;
   If ((IRS[3]<>-1) and (IRS[3]<>0) and (IRS[3]<>1)) then
      begin
      Bad:=True;
      writeln('   Angle can be an integer, a real or be unspecified');
      end;
{
                   Look if that element number was defined
}
   If Bad=False then
      begin
      If Net<ElemNumber then
         begin
         Bad:=True;
         writeln('   Element number ',ElemNumber,
                 ' is beyond last defined element (',Net,')');
         end;
      end;
   If Bad=False then
      begin
      Nne2_I_Read(F_Elem,ElemNumber,NneMax,Elem);
      If Elem[2]=0 then
         begin
         Bad:=True;
         writeln('   Element number ',ElemNumber,
                 ' was not previously defined');
         end;
      end;
{
```

```
                    Change material number when defined
}
   If (Bad=False) and (IRS[2]=0) then
      begin
      MatNumber:=Ni[2];
      If CaseToDatabase[MatNumber]<=0 then
         begin
         Bad:=True;
         writeln('   Material number ',MatNumber,' is undefined');
         end
      else
         begin
         Elem[1]:=CaseToDatabase[MatNumber];   {change for database number}
         Nne2_I_Write(F_Elem,ElemNumber,NneMax,Elem);
         end;
      end;
{
                    Change angle when defined
}
   If (Bad=False) and (IRS[3] in [0,1]) then
      begin
      Case IRS[3] of
         0: Angle:=Ni[3]*1.0;
         1: Angle:=Nr[3];
         end;
      Angle:=Angle*Pi/180.0;    {store angle in radians from degrees}
      R_Write(F_Angle,ElemNumber,Angle);
      end;
end;
{-------------------------------------------------------------------}
end.
```

K_FORDIS.PAS

```
{$N+}
Unit K_ForDis;
Interface
Uses
   Declare,K_Common,File_RW,Sonore,Crt;
Procedure Set_ForceDispl(   ForceDispl: Char;
                            IRS,Ni: VectI_Nc;
                            Nr: VectR_Nc;
                            Qtt,Dim,Nnt: IType;
                        var F_xyz: FileR;
                        var F_Force,F_Displ: FileR;
                        var F_Fd: FileByte;
                        var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_ForceDispl;
{
Procedure Set_ForceDispl(   ForceDispl: Char;
                            IRS,Ni: VectI_Nc;
                            Nr: VectR_Nc;
                            Qtt,Dim,Nnt: IType;
                        var F_xyz: FileR;
                        var F_Force,F_Displ: FileR;
                        var F_Fd: FileByte;
                        var Bad: Boolean);
}
{
   Input:
      ForceDispl: ='F' is a force is to be set
                  ='D' is a displacement is to be set
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      Dim: Dimension of problem
      Nnt: Number of nodes, increased if necessary
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
   Output:
      F_Force: modified for added Forces
      F_Displ: modified for added Displacements
```

```
      F_Fd: modified for added forces
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      FORCE FromNode,[ToNode],[Repeat],[dNode],Fx,[Fy],[Fz]
      DISPL FromNode,[ToNode],[Repeat],[dNode],Dx,[Dy],[Dz]
      ------------------------------------------
              i       i       i       i    r   r   r (i=integer, r=real)
      a real can be given as an integer, but not the reverse
      If ToNode is omitted, ToNode=FromNode
      Repeat: is the number of times pattern is repeated (excluding first)
             = 0 when omitted
      dNode: increment in node number at each new pattern
          = ToNode-F_from+1 when omotted
      When a Value is omitted for direction iD,
         Old Fd[iD] for that direction is kept
      Fd is set to 0 (force given) or 1 (Displacement given)
         only when a value is given
}
var
   Node,dNode,Repete,iRepete,iPattern,FromNode,ToNode,i: IType;
   XYZ,Value: VectR_D;
   Fd: VectByte_D;
   Warn: Boolean;
begin
   Bad:=False;
   NoString(IRS,Ott,Bad);
   If Ott<5 then
      begin
      Bad:=True;
      Writeln('   Must have at least 5 parameter');
      end;
   If IRS[1]<>0 then
      begin
      Bad:=True;
      Writeln('   FromNode must be an integer');
      end;
   If IRS[2]=1 then
      begin
      Bad:=True;
      Writeln('   ToNode can not be a real');
      end;
{
                  Avoid FromNode>ToNode
}
   If Bad=False then
      begin
      FromNode:=Ni[1];
      If IRS[2]=-1 then ToNode:=FromNode
                   else ToNode:=Ni[2];
      If FromNode>ToNode then
         begin
```

```
        Bad:=True;
        Writeln('   FromNode > ToNode');
        end;
     end;
{
                Get dNode and Repete
}
   If IRS[3]=1 then
      begin
      Bad:=True;
      writeln('   Repeat must not be a real');
      end;
   If IRS[4]=1 then
      begin
      Bad:=True;
      writeln(' dNode must not be a real');
      end;
   If Bad=False then
      begin
      If IRS[3]=-1 then Repete:=0;   {default}
      If IRS[3]=0 then Repete:=Ni[3];
      If IRS[4]=-1 then dNode:=ToNode-FromNode+1; {default}
      If IRS[4]=0 then dNode:=Ni[4];
      end;
{
                Check if all nodes are in 1< <Nnt range
}
   If Bad=False then
      If Nnt<ToNode+(dNode)*Repete then
         begin
         Bad:=True;
         writeln('   ToNode+(dNode)*Repeat is beyond last node number (',
                Nnt,')');
         end;
   If Bad=False then
      If FromNode<1 then
         begin
         Bad:=True;
         writeln('   FromNode number is under 1');
         end;
{
                Check if nodes have been defined
}
   If Bad=False then
      begin
      Warn:=False;
      For iRepete:=0 to Repete do
         For iPattern:=FromNode to ToNode do
            begin
            Node:=iPattern+dNode*iRepete;
            D_R_Read(F_xyz,Node,Dim,xyz);
            If xyz[1]>BigR then
```

```
                begin
                If Warn=False then
                   begin
                   WarningSound;
                   TextColor(LightRed+Blink);
                   Write('   Warning');
                   TextColor(LightGreen);
                   write(', setting an undefined node: ',Node);
                   Warn:=True;
                   end
                else
                   begin
                   write(' ',Node);
                   end;
                end;
             end;
       If Warn=True then writeln;
       end;
{
                  Get Values for Forces or Displacements
}
   For i:=1 to D_max do      { Set Default to 0 }
      Value[i]:=0;
   For i:=1 to Qtt-4 do
      Case IRS[i+4] of
         0: Value[i]:=Ni[i+4];
         1: Value[i]:=Nr[i+4];
         end;
{
                  Set Forces or Displacements
}
   If Bad=False then
      For iPattern:=FromNode to ToNode do
         For iRepete:=0 to Repete do
            begin
            Node:=iPattern+dNode*iRepete;       { get node number }
            D_Byte_read(F_Fd,Node,Dim,Fd);   { read old Fd }
            For i:=1 to Qtt-4 do
               begin
{
                  When a value is given in a particular direction, reset
                     that direction to be a known Force or Displacement
                  Otherwise keep the old value of Fd[i]
}
               If IRS[i+4] in [0,1] then
                  Case ForceDispl of
                     'F': Fd[i]:=0;  (Force)
                     'D': Fd[i]:=1;  (Displacement)
                     end;
               end;
            Case ForceDispl of
               'F': D_R_write(F_Force,Node,Dim,Value);
```

```
                'D': D_R_write(F_Displ,Node,Dim,Value);
                end;
            D_Byte_write(F_Fd,Node,Dim,Fd);
            end;
end;
{-------------------------------------------------------------------------}
end.
```

K_INTER.PAS

```
{$N+}
Unit K_Inter;
Interface
Uses
   Crt,Declare,WaitKey,K_Common,File_RW,LocGlo,Sonore;
Procedure Set_INTER(    IRS,Ni: VectI_Nc;
                        Nr: VectR_Nc;
                        Qtt,Dim: IType;
                        LocalType: Char;
                        Origin: VectR_3;
                        Laxis: MatrixR_3x3;
                    var Nnt: Itype;
                    var F_Fd: FileByte;
                    var F_xyz,F_Force,F_Displ: FileR;
                    var Bad: Boolean);
{-------------------------------------------------------------------------}
Implementation
Const
   Nmax = 10;          ( Used in GE_SCP.inc)
   Nmax_plus1 = 11;
Type
   MatrixR_NxN = array[1..Nmax,1..Nmax] of Rtype;
   VectR_N = array[1..Nmax] of Rtype;
{-------------------------------------------------------------------------}
Procedure GE_SCP(var A_square: MatrixR_NxN; {var to save memory}
                 var Y: VectR_N;            {var to save memory}
                     n: integer;
                 var X: VectR_N;
                 var Error_Type: Integer);
{
   Gaussian Elimination with Scaled Column Pivoting
      ( better then Maximal Column Pivoting )
   Taken from "Numerical Analysis", third edition, Burden & Faires
   Algorithm 6.3 page 328 along with algorithm 6.2 page 326
                            written by Jerome Daoust in Fall '87
   Solves the  n x n  linear system:
      E1: A(1,1).X(1) + A(1,2).X(2) + ... + A(1,n).X(n) = Y(1)
      E2: A(2,1).X(1) + A(2,2).X(2) + ... + A(2,n).X(n) = Y(2)
      ...
      En: A(n,1).X(1) + A(n,2).X(2) + ... + A(n,n).X(n) = Y(n)
   Input:
      A_square[1..N,1..N]: matrix giving the coefficients
                           of the equations
      Y[1..N]: Constants
      N: number of equations and unknowns
   Output:
      X[1..N]: Solution
      Error_Type: =0 if completed successfully
                  =1 if the system has no unique solution
}
```

```
var
   i,j,k,p,Ncopy: integer;
   Max: Rtype;
   S: VectR_N;
   A: array[1..Nmax,1..Nmax_plus1] of Rtype;
   m: array[1..Nmax,1..Nmax] of Rtype;
   Nrow : array[1..Nmax] of integer;
begin
{
                  Transfer A_square and Y into augmented matrix A
}
   For i:=1 to N do
      begin
      For j:=1 to N do
         A[i,j]:=A_square[i,j];
      A[i,N+1]:=Y[i];
      end;
{
                  Look for a nul row
}
   Error_Type:=0;
   For i:=1 to n do
      begin
      S[i]:=0;
      For j:=1 to n do
         If S[i]<abs(A[i,j]) then S[i]:=abs(A[i,j]);
      If S[i]=0 then Error_Type:=1;
      Nrow[i]:=i;
      end;
   If Error_Type=0 then
      begin
{
                  Elimination Process
}
      For i:=1 to n-1 do
         begin
         Max:=0;
         For j:=i to n do
            If Max<abs(A[Nrow[j],i])/s[Nrow[j]] then
               begin
               p:=j;
               Max:=abs(A[Nrow[j],i])/s[Nrow[j]]
               end;
         If A[Nrow[p],i]=0 then
            Error_Type:=1
         else
            begin
{
                  Simulated row interchange
}
            If Nrow[i]<>Nrow[p] then
               begin
```

```
                Ncopy:=Nrow[i];
                Nrow[i]:=Nrow[p];
                Nrow[p]:=Ncopy;
                end;
             For j:=i+1 to n do
                begin
                m[Nrow[j],i]:=A[Nrow[j],i]/A[Nrow[i],i];
                For k:=1 to n+1 do
                   A[Nrow[j],k]:=A[Nrow[j],k]-m[Nrow[j],i]*A[Nrow[i],k];
                end;
             end;
          end;
       end;
    If A[Nrow[n],n]=0 then Error_Type:=1;
    If Error_Type=0 then
(
                  Start back substitition
)
       begin
       X[n]:=A[Nrow[n],n+1]/A[Nrow[n],n];
       For i:=n-1 downto 1 do
          begin
          X[i]:=A[Nrow[i],n+1];
          For j:=i+1 to n do
             X[i]:=X[i]-A[Nrow[i],j]*X[j];
          X[i]:=X[i]/A[Nrow[i],i];
          end;
       end;
end;
{-------------------------------------------------------------------------}
Const
   DegreeMax = 10;
Type
   VectR_DEG = array[0..DegreeMax] of Rtype;
{-------------------------------------------------------------------------}
Procedure Calc_Equation(    Degree: Itype;
                        var T,V: VectR_Deg;   ( var to save memory )
                        var Equation: VectR_Deg);
(
   Input:
      Degree: Number of data points minus 1
      T,V: Vi = A0 + A1.Ti + A2.Ti^2 + A3.Ti^3 + ... + A(Degree).Ti^(Degree)
   Output:
      A: A0 ... A(Degree)
)
var
   M: MatrixR_NxN;
   X,Y: VectR_N;
   row,col,N,Error_Type: Integer;
begin
   N:=Degree+1;
   For row:=1 to N do
```

```
      begin
      Y[row]:=V[row-1];
      M[row,1]:=1;
      end;
{
                        [M][Equation]=[V]
                  ¦ 1  t1 ...  t1^N ¦ ¦a0¦   ¦v0¦
                  ¦ ...             ¦.¦  ¦ = ¦  ¦
                  ¦ 1  tn ...  tn^N ¦ ¦an¦   ¦vn¦
}
   For col:=2 to N do
      For row:=1 to N do
         M[row,col]:=M[row,col-1]*T[row-1];
   GE_SCP(M,Y,N,X,Error_Type);
   If Error_type<>0 then
      begin
      writeln('Error type ',Error_type,' in Calc_Equation');
      wait;
      Halt;
      end;
   For row:=1 to N do
      Equation[row-1]:=X[row];
end;
{-------------------------------------------------------------------}
Procedure Interpol(    Degree: Itype;
                       X: Rtype;
                   var A: VectR_Deg;   { var to save memory }
                   var Y: Rtype);
{
             Evaluates a Polynomial (Y) for X
             Y = aN.x^N + a(N-1).x^(N-1) + ...    ... + a1.x + a0
   Input:
      X: value for polynomial evaluation
      A: polynomial equation constants
   Output:
      Y: value of polynomial
}
var
   k,N: integer;
   B: array [0..Nmax_plus1] of Rtype;
begin
   N:=Degree;    (not plus 1)
   B[N+1]:=0;
   For k:=N downto 0 do
      B[k] := A[k]+X*B[k+1];
   Y:= B[0];
end;
{-------------------------------------------------------------------}
Procedure Set_INTER;
{
Procedure Set_INTER(    IRS,Ni: VectI_Nc;
                        Nr: VectR_Nc;
```

```
                    Qtt,Dim: IType;
                    LocalType: Char;
                    Origin: VectR_3;
                    Laxis: MatrixR_3x3;
                var Nnt: Itype;
                var F_Fd: FileByte;
                var F_xyz,F_Force,F_Displ: FileR;
                var Bad: Boolean);
)
(
  Input:
    IRS[i]: 0 if the (i)th number in the command line is IType
          1 for a real
          2 for a string
         -1 if not specified, or an error occured in conversion
    Ni[i]: IType number at the (i)th position
    Nr[i]: Real number at the (i)th position
    Qtt: Quantity of Possible number locations before Semicolon (End
         of command line)
    Dim: Dimension of problem
    LocalType: "R" for Rectangular
               "C" for Cylindrical
               "S" for Spherical
    Origin: coordinates or origin of local coordinate system
      = ¦ x0 y0 z0 ¦
    Laxis: direction of local coordinate system axis
      = ¦ l1 m1 n1 ¦
        ¦ l2 m2 n2 ¦
        ¦ l3 m3 n3 ¦
    Nnt: Current Number of nodes
    F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                              =1 for Displacement
    F_xyz: File with coordinates of problem
    F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
       each node
    F_Displ: File with Displacements in U,V,W for each node
  Output:
    Nnt: Current Number of nodes, increased if necessary
    F_xyz: File with coordinates of problem taken for nodes taken
          from coarse case
    F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                              =1 for Displacement <— for nodes taken from
                                 coarse case
    F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
       each node, set to 0 for new nodes
    F_Displ: File with Displacements in U,V,W for each node
            for nodes taken from coarse case
    Bad: True if there is an error in the command parameters
         False Otherwise
  Command Format:
    INTER Degree,NodeStart,NodeEnd,[NodeInc],[Repeat],[PatternInc],
          [Insert],[InsNodeStart],[InsNodeInc],[InsNodeStartInc],
```

```
            [InsPatternInc]
               (all integers)
         Coordinates are interpolated linearly in local coordinate system
         Force or displacement are interpolated linearly between nodes
         Default:
            NodeInc = NodeEnd - NodeStart
            Repeat = 0
            PatternInc = NodeEnd - NodeStart + NodeInc
            Insert = 1
            InsNodeStart = NodeStart + 1
            InsNodeInc = 1
            InsNodeStartInc = NodeInc
            InsPatternInc = (PatternInc/NodeInc).InsNodeStartInc
}
var
   iD,iPar: integer;
   NodeStart,NodeEnd,NodeInc,Repete,PatternInc,Insere,iInsere,
      InsNodeStart,InsNodeInc,InsNodeStartInc,InsPatternInc,iEquation,
      Node,iNode0,iRepete,Degree,MinQtt,IncTimes,iTimes: Itype;
   LxyzRCS,GxyzR,Displ,Force: VectR_D;
   Fd: VectByte_D;
   Node_Used: array[0..DegreeMax] of Itype;
   GxyzR_Used,LxyzRCS_Used,Displ_Used,
      Force_Used: array[0..DegreeMax] of VectR_D;
   Fd_Used: array[0..DegreeMax] of VectByte_D;
   Equation_xyz,Equation_Force,Equation_Displ: array[1..D_max] of VectR_Deg;
   Value_Used,T_Used,Equation: VectR_Deg;
   RatioB,T_value: Rtype;
   RedefinedOneOrMore: Boolean;
begin
   Bad:=False;
   NoString(IRS,Qtt,Bad);
   MinQtt:=3;    {minimum qtt of parameters}
{
                  Check quantity and parameter types
}
   If Bad=False then
      If Qtt<MinQtt then
         begin
         Bad:=True;
         writeln('   Must have at least ',MinQtt,' parameters');
         end;
   If Bad=False then
      For iPar:=1 to MinQtt do
         If 0<>IRS[iPar] then
            begin
            Bad:=True;
            writeln('   Parameter ',iPar,' must be an integer');
            end;
   If Bad=False then
      begin
      For iPar:=MinQtt+1 to Qtt do
```

```
      If (IRS[iPar] in [1,2]) then Bad:=True;
   If Bad=True then
      writeln('   Parameters must be integers or default (not given)');
   end;
If Bad=False then
   begin
   Degree:=Ni[1];
   NodeStart:=Ni[2];
   NodeEnd:=Ni[3];
   end;
If Bad=False then
   If not ((1<=Degree)and(Degree<=DegreeMax)) then
      begin
      Bad:=True;
      writeln('   Degree not in 1..',DegreeMax,' range');
      end;
If Bad=False then
   begin
   If IRS[4]=-1 then
      NodeInc:=NodeEnd-NodeStart
   else
      NodeInc:=Ni[4];
   If IRS[5]=-1 then
      Repete:=0
   else
      Repete:=Ni[5];
   If IRS[6]=-1 then
      PatternInc:=NodeEnd-NodeStart+NodeInc
   else
      PatternInc:=Ni[6];
   If IRS[7]=-1 then
      Insere:=1
   else
      Insere:=Ni[7];
   If IRS[8]=-1 then
      InsNodeStart:=NodeStart+1
   else
      InsNodeStart:=Ni[8];
   If IRS[9]=-1 then
      InsNodeInc:=1
   else
      InsNodeInc:=Ni[9];
   If IRS[10]=-1 then
      InsNodeStartInc:=NodeInc
   else
      InsNodeStartInc:=Ni[10];
   If IRS[11]=-1 then
      InsPatternInc:=Round(PatternInc/NodeInc)*InsNodeStartInc
   else
      InsPatternInc:=Ni[11];
   end;
If Bad=False then
```

```
      If NodeInc<1 then
         begin
         Bad:=True;
         writeln('   NodeInc < 1');
         end;
   If Bad=False then
      If NodeEnd<=NodeStart then
         begin
         Bad:=True;
         writeln('   NodeEnd <= NodeStart');
         end;
   If Bad=False then
      begin
      IncTimes:=Round((NodeEnd-NodeStart)/NodeInc/Degree);
      If IncTimes <> (NodeEnd-NodeStart)/NodeInc/Degree then
         begin
         Bad:=True;
         writeln('   (NodeEnd-NodeStart)/NodeInc/Degree must be an integer');
         end;
      end;
   If Bad=False then
      If Nnt<NodeEnd+Repete*PatternInc then
         begin
         Bad:=True;
         writeln('   Node numbers used exceed limit (',Nnt,')');
         end;
   If Bad=False then
      begin
      RedefinedOneOrMore:=False;
      iRepete:=0;
      While (iRepete<=Repete) and (Bad=False) do
         begin
         iTimes:=1;
         While (iTimes<=IncTimes) and (Bad=False) do
            begin
            For iNode0:=0 to Degree do
               begin
               Node_Used[iNode0] := NodeStart
                                  + (iTimes-1)*NodeInc*Degree
                                  + (iNode0)*NodeInc
                                  + iRepete*PatternInc;
               D_R_Read(F_xyz,Node_Used[iNode0],Dim,GxyzR_Used[iNode0]);
{
                   Check if all nodes are defined
}
               If GxyzR_Used[iNode0][1]>BigR then
                  begin
                  Bad:=True;
                  writeln;
                  writeln('   Node ',Node_Used[iNode0],' is undefined');
                  end;
               If Bad=False then
```

```
                begin
                T_Used[iNode0]:=iNode0;     {curvilinear coordinate}
                D_Byte_Read(F_Fd,Node_Used[iNode0],Dim,Fd_Used[iNode0]);
                D_R_Read(F_Displ,Node_Used[iNode0],Dim,Displ_Used[iNode0]);
                D_R_Read(F_Force,Node_Used[iNode0],Dim,Force_Used[iNode0]);
                end;
             end;
{
              Check if all nodes use the same Fd[iD]
}
          If Bad=False then
             begin
             For iNode0:=1 to Degree do       { compare to iNode0=0 }
                begin
                For iD:=1 to Dim do
                   If Fd_Used[0][iD]<>Fd_Used[iNode0][iD] then Bad:=True;
                If Bad=True then
                   begin
                   writeln;
                   writeln('   Nodes ',Node_Used[0],' and ',Node_Used[iNode0],
                           ' do not have the same knowns in same directions');
                   end;
                end;
             end;
{
              Get Local coordinates of nodes used
}
          If Bad=False then
             begin
             For iNode0:=0 to Degree do
                GlobalLocal(Laxis,LocalType,Origin,GxyzR_Used[iNode0],
                            Dim,LxyzRCS_Used[iNode0]);
{
              Get equations that will be used to interpolate
}
             For iEquation:=1 to 2 do
                begin
                For iD:=1 to Dim do
                   begin
                   For iNode0:=0 to Degree do
                      Case iEquation of
                         1: Value_Used[iNode0]:=LxyzRCS_Used[iNode0][iD];
                         2: Case Fd_Used[iNode0][iD] of
                            0: Value_Used[iNode0]:=Force_Used[iNode0][iD];
                            1: Value_Used[iNode0]:=Displ_Used[iNode0][iD];
                            end;
                         end;
                   Calc_Equation(Degree,T_Used,Value_Used,Equation);
                   Case iEquation of
                      1: Equation_xyz[iD]:=Equation;
                      2: Case Fd_Used[iNode0][iD] of
                         0: begin
```

```
                        Equation_Force[iD]:=Equation;
                        For iNode0:=0 to Degree do
                           Equation_Displ[iD][iNode0]:=0;
                        end;
                     1: begin
                        Equation_Displ[iD]:=Equation;
                        For iNode0:=0 to Degree do
                           Equation_Force[iD][iNode0]:=0;
                        end;
                     end;
                  end;
               end;
{
                Interpolate
}
            For iNode0:=0 to Degree-1 do
               begin
               For iInsere:=1 to Insere do
                  begin
                  Node := InsNodeStart
                        + (iTimes-1)*InsNodeStartInc*Degree
                        + (iNode0)*InsNodeStartInc
                        + (iInsere-1)*InsNodeInc
                        + iRepete*InsPatternInc;
                  D_R_Read(F_xyz,Node,Dim,GxyzR);
                  If not (GxyzR[1]>BigR) then
                     begin
                     If RedefinedOneOrMore=False then
                        begin
                        WarningSound;
                        TextColor(LightRed+Blink);
                        Write('   Warning');
                        TextColor(LightGreen);
                        write(', redefined nodes: ');
                        RedefinedOneOrMore:=True;
                        end
                     else
                        write(',');
                     write(Node);
                     end;
{
                  Extend node number if necessary
}
                    StretchNodes(Nnt,Node,Dim,F_Fd,F_xyz,F_Force,F_Displ);
{
                  Interpolate coordinates in Local coordinate system
                  and Forces or Displacements in each direction
                  Set Fd[iD] same as first node used
}
                    RatioB:=(iInsere)/(Insere+1);
                    T_value := T_Used[iNode0]*(1-RatioB)
```

```
                        + T_Used[iNode0+1]*RatioB;
                    For iD:=1 to Dim do
                      begin
                      Interpol(Degree,T_value,Equation_xyz[iD],LxyzRCS[iD]);
                      Fd[iD]:=Fd_Used[0][iD];  {same as first node used}
                      Case Fd[iD] of
                        0: begin
                          Displ[iD]:=0;
                          Interpol(Degree,T_value,Equation_Force[iD],
                                  Force[iD]);
                          end;
                        1: begin
                          Force[iD]:=0;
                          Interpol(Degree,T_value,Equation_Displ[iD],
                                  Displ[iD]);
                          end;
                        end;
                      end;
{
                  Convert back into global coordinate system
                  save coordinates, forces, displacement and state of knowns
}
                    LocalGlobal(Laxis,LocalType,Origin,LxyzRCS,Dim,GxyzR);
                    D_R_write(F_xyz,Node,Dim,GxyzR);
                    D_R_write(F_Force,Node,Dim,Force);
                    D_R_write(F_Displ,Node,Dim,Displ);
                    D_Byte_write(F_Fd,Node,Dim,Fd);
                    end;
                  end;
                end;
              Inc(iTimes);
              end;
            Inc(iRepete);
            end;
        If RedefinedOneOrMore=True then writeln;
        end;
end;
{-------------------------------------------------------------}
end.
```

K_LOCAL.PAS

```
{$N+}
Unit K_Local;
Interface
Uses
   Declare,VectMat3;
Procedure Set_Local(     IRS,Ni: VectI_Nc;
                         Nr: VectR_Nc;
                         Ns80: VectS80_Nc;
                         Qtt: IType;
                    var LocalType: Char;
                    var Origin: VectR_3;
                    var Laxis: MatrixR_3x3;
                    var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_Local;
{
Procedure Set_Local(     IRS,Ni: VectI_Nc;
                         Nr: VectR_Nc;
                         Ns80: VectS80_Nc;
                         Qtt: IType;
                    var LocalType: Char;
                    var Origin: VectR_3;
                    var Laxis: MatrixR_3x3;
                    var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Ns80[i]: String at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
   Output:
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
      Origin: coordinates or origin of local coordinate system
         = ¦ x0 y0 z0 ¦
      Laxis: direction of local coordinate system axis
         = ¦ l1 m1 n1 ¦
           ¦ l2 m2 n2 ¦
           ¦ l3 m3 n3 ¦
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
```

```
      LOCAL LocalType,x0,y0,[z0],L1,M1,[N1],L2,M2,[N2]
      ------------------------------------------------
            String   r  r   r   r  r   r   r  r   r    (i=integer,r=real)
      a real can be given as an integer, but not the reverse
)
var
   iDir,iD,iPar: integer;
begin
   Bad:=False;
{
                  Check quantity and parameter types
}
   If not (Qtt in [9,10]) then
      begin
      Bad:=True;
      Writeln('   Must have 9 or 10 parameter');
      end;
   If Bad=False then
      If IRS[1]<>2 then
         begin
         Bad:=True;
         writeln('   First parameter must be a string');
         end;
   If Bad=False then
      If not (Ns80[1][1] in ['R','C','S']) then
         begin
         Bad:=True;
         writeln('   Local coordinate type does not start with: R C S');
         end;
   If Bad=False then
      For iPar:=1 to Qtt do
         If iPar in [1,2,3,5,6,8,9] then
            If IRS[1]=-1 then
               begin
               Bad:=True;
               Writeln('   Parameter ',iPar,' must be given');
               end;
{
                  Fill default values
}
   If Bad=False then
      begin
      If IRS[4]=-1 then
         begin
         IRS[4]:=1;   (real)
         Nr[4]:=0;
         end;
      If IRS[7]=-1 then
         begin
         IRS[7]:=1;   (real)
         Nr[7]:=0;
         end;
```

```
    If Qtt<10 then
       begin
       Qtt:=10;
       IRS[10]:=1;   (real)
       Nr[10]:=0;
       end;
{
                Set all numbers to real
}
    For iPar:=2 to Qtt do
       If IRS[iPar]=0 then
          begin
          IRS[iPar]:=1;
          Nr[iPar]:=Ni[iPar];
          end;
{
                Get local coordinate system type
}
    LocalType:=Ns80[1][1];
{
                Get origin coordinates
}
    For iPar:=2 to Qtt do
       Case iPar of
          2..4: Origin[iPar-1]:=Nr[iPar];
          5..7: Laxis[1,iPar-4]:=Nr[iPar];
          8..10: Laxis[2,iPar-7]:=Nr[iPar];
          end;
{
                Get direction 1 and 2, normalize
}
    For iDir:=1 to 2 do
       NormV3(Laxis[iDir,1],Laxis[iDir,2],Laxis[iDir,3]);
    end;
{
                Get third direction by making the vector product of
                Dir3 = Dir1 x Dir2
}
  If Bad=False then
    begin
    VectorP3(Laxis[1,1],Laxis[1,2],Laxis[1,3],
             Laxis[2,1],Laxis[2,2],Laxis[2,3],
             Laxis[3,1],Laxis[3,2],Laxis[3,3]);
    NormV3(Laxis[3,1],Laxis[3,2],Laxis[3,3]);
{
                Before, Dir1 and Dir2 where not necessarely normal.
                But V3 and V1 are normal -> V2
                V2 = V3 x V1 (right hand rule)
}
    VectorP3(Laxis[3,1],Laxis[3,2],Laxis[3,3],
             Laxis[1,1],Laxis[1,2],Laxis[1,3],
             Laxis[2,1],Laxis[2,2],Laxis[2,3]);
```

```
        end;
end;
{-------------------------------------------------------------------}
end.
```

K_MAT.PAS

```
{$N+}
Unit K_Mat;
Interface
Uses
   Declare,K_Common;
Procedure Set_Mat(     IRS,Ni: VectI_Nc;
                       Ns80: VectS80_Nc;
                       Qtt: IType;
                   var MatLabel: VectS80_Nmat;
                   var CaseToDatabase: VectByte_Nmat;
                   var Bad: Boolean);
{-------------------------------------------------------------}
Implementation
Procedure Set_Mat;
{
Procedure Set_Mat(     IRS,Ni: VectI_Nc;
                       Ns80: VectS80_Nc;
                       Qtt: IType;
                   var MatLabel: VectS80_Nmat;
                   var CaseToDatabase: VectByte_Nmat;
                   var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Ns80[i]: String at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      MatLabel[iMaterial]: Name of material, details
         Here it is in upper case with blanks removed
   Output:
      CaseToDatabase[iMat]: number in database for material iMat in case
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      MAT #,NameOfMaterial;
      ------------------
          i       s        (i=integer, s=string)
}
var
   DatabaseNumber,iData,Count,i,j: integer;
   MaterialName: String80;
begin
   Bad:=False;
   If Qtt<>2 then
      begin
```

```
      writeln('   Must have 2 parameters');
      Bad:=True;
      end;
   If IRS[1]<>0 then
      begin
      writeln('   Material number must be given, as an integer');
      Bad:=True;
      end;
   If Bad=False then
      If Nmat_max<Ni[1] then
         begin
         writeln('   A maximum of ',Nmat_max,' materials are allowed');
         Bad:=True;
         end;
   If IRS[2]<>2 then
      begin
      writeln('   Material name must be a string');
      Bad:=True;
      end;
   If Bad=False then
      begin
{
                Look for material in database corresponding
                to this name (compare first characters)
}
      MaterialName:=Ns80[2];
      Count:=0;
      DatabaseNumber:=0;
      For i:=1 to Length(MaterialName) do
         MaterialName[i]:=UpCase(MaterialName[i]);
      For iData:=1 to Nmat_max do
         begin
{
                Look for a match
}
         If MaterialName=Copy(MatLabel[iData],1,Length(MaterialName)) then
            begin
            Count:=Count+1;
            DatabaseNumber:=iData;
            end;
         end;
      If Count<>1 then
         begin
         writeln('   ',Count,' materials correspond to ',MaterialName);
         Bad:=True;
         end;
      end;
   If Bad=False then
      CaseToDatabase[Ni[1]]:=DatabaseNumber;
end;
{-------------------------------------------------------------}
end.
```

K_NODE.PAS

```
{$N+}
Unit K_node;
Interface
Uses
   Crt,Declare,K_Common,File_RW,LocGlo,Sonore,    waitkey;
Procedure Set_node(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                       LocalType: Char;
                       Origin: VectR_3;
                       Laxis: MatrixR_3x3;
                   var F_xyz,F_Force,F_Displ: FileR;
                   var F_Fd: FileByte;
                   var Nnt: IType;
                       Dim: IType;
                   var Bad: Boolean);
Procedure Set_Ngen(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt,Dim: IType;
                       LocalType: Char;
                       Origin: VectR_3;
                       Laxis: MatrixR_3x3;
                   var F_xyz,F_Force,F_Displ: FileR;
                   var F_Fd: FileByte;
                   var Nnt: IType;
                   var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_node;
{
Procedure Set_node(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt: IType;
                       LocalType: Char;
                       Origin: VectR_3;
                       Laxis: MatrixR_3x3;
                   var F_xyz,F_Force,F_Displ: FileR;
                   var F_Fd: FileByte;
                   var Nnt: IType;
                       Dim: IType;
                   var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
```

```
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
      Origin: coordinates or origin of local coordinate system
         = ¦ x0 y0 z0 ¦
      Laxis: direction of local coordinate system axis
         = ¦ l1 m1 n1 ¦
           ¦ l2 m2 n2 ¦
           ¦ l3 m3 n3 ¦
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      Nnt: Number of nodes
      Dim: Dimension of problem
   Output:
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
          initialized as expansion needs it
      F_Force,F_Displ,F_Fd: made longer is necessary
      Nnt: Number of nodes, increased if necessary
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      NODE #,[X],[Y],[Z]
      ------
           i  r   r   r  (i=integer, r=real)
      a real can be given as an integer, but not the reverse
      When a coordinate is omitted 0 is assumed, but X Y Z order is kept
}
var
   I,J: IType;
   LxyzRCS,GxyzR: VectR_D;
begin
   Bad:=False;
   NoString(IRS,Qtt,Bad);
   If Qtt<1 then
      begin
      Bad:=True;
      Writeln('   Must have at least 1 parameter');
      end;
   If IRS[1]<>0 then
      begin
      Bad:=True;
      Writeln('   Node number must be an integer');
      end;
```

```
   If Bad=False then
      begin
      StretchNodes(Nnt,Ni[1],Dim,F_Fd,F_xyz,F_Force,F_Displ);
      For i:=1 to Dim do      ( Set Default to 0 )
         LxyzRCS[i]:=0;
      For i:=2 to Qtt do
         Case IRS[i] of
            0: LxyzRCS[i-1]:=Ni[i];
            1: LxyzRCS[i-1]:=Nr[i];
            end;
{
                  Convert from Local Rectangular-Cylindrical-Sherical
                              Global rectangular to
}
      LocalGlobal(Laxis,LocalType,Origin,LxyzRCS,Dim,GxyzR);
      D_R_write(F_xyz,Ni[1],dim,GxyzR);
      end;
end;
{-------------------------------------------------------------------}
Procedure Set_Ngen;
{
Procedure Set_Ngen(    IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Qtt,Dim: IType;
                       LocalType: Char;
                       Origin: VectR_3;
                       Laxis: MatrixR_3x3;
                  var F_xyz,F_Force,F_Displ: FileR;
                  var F_Fd: FileByte;
                  var Nnt: IType;
                  var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      Dim: Dimension of problem
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
      Origin: coordinates or origin of local coordinate system
         = | x0 y0 z0 |
      Laxis: direction of local coordinate system axis
         = | l1 m1 n1 |
           | l2 m2 n2 |
           | l3 m3 n3 |
```

```
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
         XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
         each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
         each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      Nnt: Number of nodes
   Output:
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
         XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Force,F_Displ,F_Fd: made longer is necessary
      Nnt: Number of nodes
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      NGEN FromNode,[ToNode],Repeat,[dNode],dX,[dY],[dZ]
      ---------------------------------------

              i      i      i      i     r   r   r  (i=integer, r=real)
      a real can be given as an integer, but not the reverse
      FromNode <= ToNode
      (ToNode-FromNode) < dNode  (Step in Node generation)
      if ToNode is omitted ToNode:=FromNode
      Repeat: number of times FromNode to ToNode pattern is repeated
      if dNode is omitted dNode = (ToNode-FromNode+1)
      When a Increase (dX..dZ) is omitted 0 is assumed, but order is kept
}
var
   N_last,iRepete,iNode,iD,dNode,ToNode,FromNode,Repete,i: IType;
   Xs,Ys,Xu,Yu: integer;
   LdeltaRCS,DeltaLxyzRCS,GxyzRO,GxyzR,LxyzRCSO,LxyzRCS: VectR_D;
   AtLeastOneUndefined: Boolean;
begin
   Bad:=False;
   AtLeastOneUndefined:=False;
   NoString(IRS,Qtt,Bad);
   If Qtt<5 then
      begin
      writeln('   5 parameters neaded');
      bad:=True;
      end;
{
                  Verify data that MUST be given
}
   If IRS[1]<>0 then
      begin
      writeln('   FromNode must be an integer');
      bad:=True;
      end;
   If IRS[3]<>0 then
      begin
```

```
      writeln('   Repeat must be an integer');
      bad:=True;
      end;
{
                  Verify that ToNode, dNode are not real numbers
}
   If IRS[2]=1 then
      begin
      writeln('   ToNode must not be a real');
      bad:=True;
      end;
   If IRS[4]=1 then
      begin
      writeln('   dNode must not be a real');
      bad:=True;
      end;
{
                  Get dX dY dZ
}
   For i:=1 to D_max do
      LdeltaRCS[i]:=0;          { Initialize increments }
   For i:=5 to Qtt do
      If IRS[i]<>-1 then
         Case IRS[i] of
            0: LdeltaRCS[i-4]:=Ni[i]*1.0;
            1: LdeltaRCS[i-4]:=Nr[i];
            end;;
{
                  Get FromNode ToNode Repete dNode
}
   If Bad=False then
      begin
      FromNode:=Ni[1];
      If IRS[2]=0 then ToNode:=Ni[2]
                  else ToNode:=FromNode;
      Repete:=Ni[3];
      If IRS[4]<>0 then dNode:=Ni[4]
                  else dNode:=ToNode-FromNode+1;
      If FromNode>ToNode then
         begin
         Bad:=True;
         writeln('   FromNode (',FromNode,') > ToNode (',ToNode,')');
         end;
      If Repete<1 then
         begin
         Bad:=True;
         writeln('   Repeat < 1');
         end;
      end;
{
                  Check node number increment
}
```

```
   If Bad=False then
      begin
      If not ((ToNode-FromNode < dNode) or (dNode < FromNode-ToNode)) then
         begin
         writeln('   dNode must be greater than (ToNode-FromNode)');
         writeln('                  or less than (FromNode-ToNode)');
         writeln('   to avoid re-definition of node within original pattern');
         bad:=True;
         end;
      end;
   If Bad=False then
      begin
{
                  Readjust Number of nodes
}
      N_last:=ToNode+Repete*dNode;
      StretchNodes(Nnt,N_last,Dim,F_Fd,F_xyz,F_Force,F_Displ);
{
                  Create new nodes coordinates
}
      write('   Creating new coordinates  ');
      Xs:=WhereX;
      Ys:=WhereY;
      For iNode:=FromNode to ToNode do
         begin
         gotoXY(Xs,Ys);
         write(ToNode-iNode+1,' ');
         D_R_Read(F_xyz,iNode,Dim,GxyzR0);
{
                  Look for At Least One Undefined node being copied
}
         If BigR<GxyzR0[1] then
            begin
            If AtLeastOneUndefined=False then
               begin
               AtLeastOneUndefined:=True;
               WarningSound;
               If Ys=25 then Dec(Ys);    { correct counter line }
               writeln;                  { add a new line }
               TextColor(LightRed+Blink);
               Write('   Warning');
               TextColor(LightGreen);
               write(', undefined nodes being copied: ',iNode);
               Xu:=WhereX;
               Yu:=WhereY;
               GotoXY(Xs,Ys);
               end
            else
               begin
               GotoXY(Xu,Yu);
               Write('...',iNode,'   ');
               GotoXY(Xs,Ys);
```

```
            end;
         end
      else
         begin
{
               Generate new nodes only when when node is defined:
               Convert from Global rectangular
                     to   Local Rectangular-Cylindrical-Sherical
               before incrementing node coordinates and then
               Convert from Local Rectangular-Cylindrical-Sherical
                     to   Global rectangular
}
         GlobalLocal(Laxis,LocalType,Origin,GxyzR0,Dim,LxyzRCS0);
         For iRepete:=1 to Repete do
            begin
            For iD:=1 to Dim do
               LxyzRCS[iD]:=LxyzRCS0[iD]+iRepete*ldeltaRCS[iD];
            LocalGlobal(Laxis,LocalType,Origin,LxyzRCS,Dim,GxyzR);
            D_R_write(F_xyz,iNode+diNode*iRepete,dim,GxyzR);
            end;
         end;
      end;
   GotoXY(1,Ys);
   ClrEOL;
   If AtLeastOneUndefined=True then   { keep warning message }
      begin
      DelLine;
      GotoXY(1,WhereY+1);
      end;
   end;
end;
{-------------------------------------------------------------}
end.
```

K_OLD.PAS

```
{$N+}
Unit K_Old;
Interface
Uses
   Declare,K_Common,File_RW;
Procedure Set_OLD(    IRS,Ni: VectI_Nc;
                      Nr: VectR_Nc;
                      Qtt,Dim,CoarseDim,CoarseNntOld: IType;
                  var Nnt: Itype;
                  var FC_OldNew: FileI;
                  var F_Fd: FileByte;
                  var F_xyz,FC_xyz,F_Force,F_Displ,FC_Displ: FileR;
                  var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_OLD;
{
Procedure Set_OLD(    IRS,Ni: VectI_Nc;
                      Nr: VectR_Nc;
                      Qtt,Dim,CoarseDim,CoarseNntOld: IType;
                  var Nnt: Itype;
                  var FC_OldNew: FileI;
                  var F_Fd: FileByte;
                  var F_xyz,FC_xyz,F_Force,F_Displ,FC_Displ: FileR;
                  var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      Dim: Dimension of problem
      CoarseDim: Dimension of Coarse solution problem
      CoarseNntOld: Number of nodes in coarse solution (user's number)
      Nnt: Current Number of nodes
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      F_xyz: File with coordinates of problem
      FC_xyz: File with coordinates of coarse problem
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Displ: File with Displacements in U,V,W for each node
      FC_Displ: File with Displacements in U,V,W for each node of coarse problem
   Output:
      Nnt: Current Number of nodes, increased if necessary
```

```
      F_xyz: File with coordinates of problem taken for nodes taken
             from coarse case
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement <— for nodes taken from
                                   coarse case
      F_Force[1..Nnt x Dimension]: File with forces in Fx1,Fy1,Fz1 for
          each node, set to 0 for new nodes
      F_Displ: File with Displacements in U,V,W for each node
               for nodes taken from coarse case
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      OLD CNodeStart,[CNodeEnd],[CNodeInc],[Repeat],[CPatternInc],
                    [NodeStart],[NodeInc],[PatternInc]
      ————————————      (all integers)
      Default:
         CNodeEnd = CNodeStart
         CNodeInc = 1
         Repeat = 0
         CPatternInc = CNodeEnd - CNodeStart + CNodeInc
         NodeStart = CNodeStart
         NodeInc = 1
         PatternInc = (CPatternInc/CNodeInc).NodeInc
}
var
   iD,iPar: integer;
   CNodeStart,CNodeEnd,CNodeInc,Repete,CPatternInc,
      NodeStart,NodeInc,PatternInc,
      iNode0,iRepete,OldCNode,Node,NewCNode: Itype;
   Fd_1: VectByte_D;
   GxyzR,Displ: VectR_D;
begin
   Bad:=False;
   NoString(IRS,Qtt,Bad);
{
                  Check quantity and parameter types
}
   If Bad=False then
      If Qtt<1 then
         begin
         Bad:=True;
         writeln('   Must have at least 1 parameter');
         end;
   If Bad=False then
      If Dim<>CoarseDim then
         begin
         Bad:=True;
         writeln('   Problem must be of same dimension as coarse solution');
         end;
   If Bad=False then
      begin
      For iPar:=1 to Qtt do
```

```
      If 0<IRS[iPar] then Bad:=True;
   If Bad=True then
      writeln('   Parameters must be integers or default (not given)');
   end;
If Bad=False then
   If IRS[1]=-1 then
      begin
      Bad:=True;
      writeln('   CNodeStart must be given');
      end
   else
      CNodeStart:=Ni[1];
If Bad=False then
   If IRS[2]=-1 then
      CNodeEnd:=CNodeStart
   else
      CNodeEnd:=Ni[2];
If Bad=False then
   If IRS[3]=-1 then
      CNodeInc:=1
   else
      CNodeInc:=Ni[3];
If Bad=False then
   If IRS[4]=-1 then
      Repete:=0
   else
      Repete:=Ni[4];
If Bad=False then
   If IRS[5]=-1 then
      CPatternInc:=CNodeEnd-CNodeStart+CNodeInc
   else
      CPatternInc:=Ni[5];
If Bad=False then
   If IRS[6]=-1 then
      NodeStart:=CNodeStart
   else
      NodeStart:=Ni[6];
If Bad=False then
   If IRS[7]=-1 then
      NodeInc:=1
   else
      NodeInc:=Ni[7];
If Bad=False then
   If IRS[8]=-1 then
      PatternInc:=Round(CPatternInc/CNodeInc)*CNodeInc
   else
      PatternInc:=Ni[8];
If Bad=False then
   If Round((CNodeEnd-CNodeStart)/CNodeInc)
        <> ((CNodeEnd-CNodeStart)/CNodeInc) then
      begin
      Bad:=True;
```

```
      writeln('   (CNodeEnd-CNodeStart)/CNodeInc must be an integer');
      end;
   If Bad=False then
      If CNodeEnd < CNodeStart then
      begin
      Bad:=True;
      writeln('   CNodeEnd < CNodeStart');
      end;
   If Bad=False then
      If CoarseNntOld<CNodeEnd+Repete*CPatternInc then
      begin
      Bad:=True;
      writeln('   Node numbers of coarse case exceed limit (',
             CoarseNntOld,')');
      end;
   If Bad=False then
      begin
      For iD:=1 to Dim do
         Fd_1[iD]:=1;   (known displacement)
      iRepete:=0;
      While (iRepete<=Repete) and (Bad=False) do
         begin
         iNode0:=0;
         While (iNode0<=Round((CNodeEnd-CNodeStart)/CNodeInc))
               and (Bad=False) do
            begin
            OldCNode:=CNodeStart+iNode0*CnodeInc+iRepete*CPatternInc;
            I_Read(FC_OldNew,OldCNode,NewCNode);
            If NewCNode=0 then
               begin
               Bad:=True;
               writeln('   Node number ',OldCNode,
                      ' of coarse case is undefined');
               end;
            If Bad=False then
               begin
               Node:=NodeStart+iNode0*NodeInc+iRepete*PatternInc;
               StretchNodes(Nnt,Node,Dim,F_Fd,F_xyz,F_Force,F_Displ);
{
                  Copy coordinates and displacements
                  set Fd[iD]=1
}
               D_R_Read(FC_Displ,NewCNode,Dim,Displ);
               D_R_write(F_Displ,Node,Dim,Displ);
               D_R_Read(FC_xyz,NewCNode,Dim,GxyzR);
               D_R_write(F_xyz,Node,Dim,GxyzR);
               D_Byte_write(F_Fd,Node,Dim,Fd_1);
               end;
            Inc(iNode0);
            end;
         Inc(iRepete);
         end;
```

```
        end;
end;
{-------------------------------------------------------------------------}
end.
```

K_OPER.PAS

```
{$N+}
Unit K_Oper;
Interface
Uses
   Crt,Declare,Sonore;
Procedure Set_Oper(     IRS,Ni: VectI_Nc;
                        Nr: VectR_Nc;
                        Ns80: VectS80_Nc;
                        Qtt: IType;
                    var VarName,VarValue: VectS80_Var;
                    var VarQtt: Byte;
                    var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_Oper;
{
Procedure Set_Oper(     IRS,Ni: VectI_Nc;
                        Nr: VectR_Nc;
                        Ns80: VectS80_Nc;
                        Qtt: IType;
                    var VarName,VarValue: VectS80_Var;
                    var VarQtt: Byte;
                    var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: RType number at the (i)th position
      Ns80[i]: String at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      VarName[1..Var_max]: variable names
      VarValue[1..Var_max]: variable values
      VarQtt: quantity of variables
   Output:
      VarName[1..Var_max]: variable names
      VarValue[1..Var_max]: variable values
      VarQtt: quantity of variables
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      OPER NewVariableName,[A],Operator,B;
      ------------------------------------
               s          ir    s    ir  (i=integer,r=real, s=string)
      A=1 if omitted
}
```

```
var
   AI,BI,CI: Itype;
   AR,BR,CR: Rtype;
   i: integer;
   Cstring,Operator: String255;
   Redefine,RealCalc: Boolean;
begin
   Bad:=False;
   Redefine:=False;
   If Qtt<>4 then
      begin
      writeln('   Must have 4 parameters');
      Bad:=True;
      end;
{
                  integer or real -> previously defined -> ignore
}
   If IRS[1] in [0,1] then
      begin
      Redefine:=True;
      WarningSound;
      TextColor(LightRed+Blink);
      Write('   Warning');
      TextColor(LightGreen);
      write(', ignored due to redefinition. First parameter is ');
      Case IRS[1] of
         0: writeln('an integer number');
         1: writeln('a real number');
         end;
      end;
   If IRS[1]=-1 then
      begin
      writeln('   Variable name must be given');
      Bad:=True;
      end;
   If IRS[2]=2 then
      begin
      writeln('   A must not be a string');
      Bad:=True;
      end;
   If IRS[3]<>2 then
      begin
      writeln('   Operator must be a string');
      Bad:=True;
      end;
   If not (IRS[4] in [0,1]) then
      begin
      Case IRS[4] of
         -1: writeln('   B must be an integer, a real number or a variable');
          2: writeln('   Unknown variable: ',Ns80[4]);
          end;
      Bad:=True;
```

```
      end;
{
                  Get A and B as reals if one of them is a real number
}
   If Bad=False then
      begin
      If ((IRS[2]=1) or (IRS[4]=1)) then   (if A or B is a real -> reals)
         begin
         Case IRS[2] of
           -1: AR:=1.0;
            0: AR:=1.0*Ni[2];
            1: AR:=Nr[2];
            end;
         Case IRS[4] of
            0: BR:=1.0*Ni[4];
            1: BR:=Nr[4];
            end;
         RealCalc:=True;  { Real number calculation }
         end
      else
         begin
         Case IRS[2] of
           -1: AI:=1;
            0: AI:=Ni[2];
            end;
         BI:=Ni[4];
         RealCalc:=False;
         end;
      end;
{
                  Get Operator type
}
   If Bad=False then
      begin
      For i:=1 to Length(Ns80[3]) do     { put Operator in upper case }
         Ns80[3][i]:=UpCase(Ns80[3][i]);
      Operator:=''; (none)
      If Ns80[3]='+'        then Operator:=Ns80[3];
      If Ns80[3]='-'        then Operator:=Ns80[3];
      If Ns80[3]='*'        then Operator:=Ns80[3];
      If Ns80[3]='/'        then Operator:=Ns80[3];
      If Ns80[3]='*ABS'     then Operator:=Ns80[3];
      If Ns80[3]='*ARCTAN'  then Operator:=Ns80[3];
      If Ns80[3]='*COS'     then Operator:=Ns80[3];
      If Ns80[3]='*EXP'     then Operator:=Ns80[3];
      If Ns80[3]='*EXP10'   then Operator:=Ns80[3];
      If Ns80[3]='*LN'      then Operator:=Ns80[3];
      If Ns80[3]='*LOG10'   then Operator:=Ns80[3];
      If Ns80[3]='*ROUND'   then Operator:=Ns80[3];
      If Ns80[3]='*SIN'     then Operator:=Ns80[3];
      If Ns80[3]='*SQR'     then Operator:=Ns80[3];
      If Ns80[3]='*SQRT'    then Operator:=Ns80[3];
```

```
      If Ns80[3]='*TAN'    then Operator:=Ns80[3];
      If Ns80[3]='*TRUNC'  then Operator:=Ns80[3];
      If Operator='' then
         begin
         writeln('   Unknown operator: ',Ns80[2]);
         Bad:=True;
         end;
      end;
{
                Check for a valid operation
}
   If Bad=False then
      begin
{
                Avoid division by zero
}
      If Operator='/' then
         begin
         If RealCalc=False then     {integers}
            begin
            If BI=0 then
               begin
               writeln('   Division by zero');
               Bad:=True;
               end;
            end
         else                       {reals}
            begin
            If BR=0.0 then
               begin
               writeln('   Division by zero');
               Bad:=True;
               end;
            end;
         end;
{
                Avoid Tan(Pi/2)  Tan(-Pi/2)
}
      If Operator='*TAN' then
         begin
         If RealCalc=False then     {integers}
            begin
            {nothing to worry about}
            end
         else                       {reals}
            begin
            If BR=Pi/2 then
               begin
               writeln('   ArcTan(Pi/2)=infinity');
               Bad:=True;
               end;
            If BR=-Pi/2 then
```

```
            begin
            writeln('   ArcTan(-Pi/2)=infinity');
            Bad:=True;
            end;
         end;
      end;
{
            Avoid Ln(x) Log10(x) with x<=0
}
   If ((Operator='*LN') or (Operator='*LOG10')) then
      begin
      If RealCalc=False then     (integers)
         begin
         If BI<=0 then
            begin
            writeln('   Logarithm of a number<=0');
            Bad:=True;
            end;
         end
      else                       (reals)
         begin
         If BR<=0.0 then
            begin
            writeln('   Logarithm of a number<=0');
            Bad:=True;
            end;
          end;
      end;
{
            Avoid sqrt(x) with x<=0
}
   If Operator='*SQRT' then
      begin
      If RealCalc=False then     (integers)
         begin
         If BI<0 then
            begin
            writeln('   Square root of a number<=0');
            Bad:=True;
            end;
         end
      else                       (reals)
         begin
         If BR<0.0 then
            begin
            writeln('   Square root of a number<=0');
            Bad:=True;
            end;
          end;
      end;
   end;
{
```

```
                Do operation
}
   If Bad=False then
      begin
      If RealCalc=False then   {integer}
         begin
         If Operator='+'       then CI:=AI+BI;
         If Operator='-'       then CI:=AI-BI;
         If Operator='*'       then CI:=AI*BI;
         If Operator='/'       then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*ABS'    then CI:=AI*ABS(BI);
         If Operator='*ARCTAN' then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*COS'    then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*EXP'    then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*EXP10'  then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*LN'     then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*LOG10'  then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*ROUND'  then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*SIN'    then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*SQR'    then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*SQRT'   then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*TAN'    then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         If Operator='*TRUNC'  then begin AR:=AI; BR:=BI; RealCalc:=True; end;
         end;
      If RealCalc=True then   {reals}
         begin
         If Operator='+'       then CR:=AR+BR;
         If Operator='-'       then CR:=AR-BR;
         If Operator='*'       then CR:=AR*BR;
         If Operator='/'       then CR:=AR/BR;
         If Operator='*ABS'    then CR:=AR*Abs(BR);
         If Operator='*ARCTAN' then CR:=AR*ArcTan(BR);
         If Operator='*COS'    then CR:=AR*Cos(BR);
         If Operator='*EXP'    then CR:=AR*Exp(BR);
         If Operator='*EXP10'  then CR:=AR*Exp(BR*Ln(10.0));
         If Operator='*LN'     then CR:=AR*Ln(BR);
         If Operator='*LOG10'  then CR:=AR*Ln(BR)/Ln(10.0);
         If Operator='*ROUND'  then CR:=AR*Round(BR);
         If Operator='*SIN'    then CR:=AR*Sin(BR);
         If Operator='*SQR'    then CR:=AR*Sqr(BR);
         If Operator='*SQRT'   then CR:=AR*Sqrt(BR);
         If Operator='*TAN'    then CR:=AR*Sin(BR)/Cos(BR);
         If Operator='*TRUNC'  then CR:=AR*Trunc(BR);
{
                Convert back to an integer if real value is
                close enough to an integer
}
         If Abs(CR-Round(CR))<1E-12 then
            begin
            CI:=Round(CR);
            RealCalc:=False;
            end;
```

```
        end;
      end;
{
                Convert result into a string
}
   If RealCalc=False then  {integer}
      begin
      str(CI,Cstring);
      end
   else                    {real}
      begin
      str(CR,Cstring);
      end;
{
                Check if maximum number of variables is reached
}
   If Bad=False then
      If Var_max<=VarQtt then
         begin
         writeln('   Too many variables (max ',Var_max,')');
         bad:=True;
         end;
{
                Do not re-define a variable (when Redefine=True)
}
   If (Bad=False) and (Redefine=False) then
      begin
      For i:=1 to Length(Ns80[1]) do     { put variable name in upper case }
         Ns80[1][i]:=UpCase(Ns80[1][i]);
      VarName[VarQtt+1]:=Ns80[1];
      VarValue[VarQtt+1]:=Cstring;
      Inc(VarQtt);
      end;
end;
{-------------------------------------------------------------------}
end.
```

K_TRAC.PAS

```
{$N+}
Unit K_Trac;
Interface
Uses
   Crt,Declare,K_Common,File_RW,Exyz,Elem1,Elem2;
Procedure Set_Traction(    IRS,Ni: VectI_Nc;
                           Nr: VectR_Nc;
                           Qtt,Net: IType;
                       var F_XYZ: FileR;
                       var F_Elem: FileI;
                       var F_Force: FileR;
                       var F_Fd: FileByte;
                           NneMax,Dim: IType;
                       var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_Traction;
{
Procedure Set_Traction(    IRS,Ni: VectI_Nc;
                           Nr: VectR_Nc;
                           Qtt,Net: IType;
                       var F_XYZ: FileR;
                       var F_Elem: FileI;
                       var F_Force: FileR;
                       var F_Fd: FileByte;
                           NneMax,Dim: IType;
                       var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      Net: Number of elements
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      NneMax: Maximum number of nodes in an element
      Dim: Dimension of problem
```

```
   Output:
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
         each node.
         Forces are set to simulate a traction on a face of an element
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
         Nodes in contact with chosen face are set to Fd=0
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      TRAC FromElement,[ToElement],[Repeat],[dElement],Face,Traction
      ------------------------------------------------------------

            i           i         i         i        i     ir
}
var
   iRepete,dElement,Repete,Etype,FromElement,ToElement,iE,iNt,iNne,iD,Node: Itype;
   Face,nT: Byte;
   Tindex: VectI_Nne;
   Ft: MatrixR_Nnex3;
   Exyz: MatrixR_Nnex3;
   Traction: Rtype;
   Elem: VectI_Nne2;
   Force: VectR_D;
begin
   Bad:=False;
   NoString(IRS,Qtt,Bad);
   If Qtt<>6 then
      begin
      Bad:=True;
      Writeln('   Must have 6 parameter');
      end;
   If IRS[1]<>0 then
      begin
      Bad:=True;
      Writeln('   FromElement must be an integer');
      end;
   If IRS[2]=1 then
      begin
      Bad:=True;
      Writeln('   ToElement must not be a real');
      end;
   If IRS[3]=1 then
      begin
      Bad:=True;
      Writeln('   Repeat must not be a real');
      end;
   If IRS[4]=1 then
      begin
      Bad:=True;
      Writeln('   dElement must not be a real');
      end;
   If IRS[5]<>0 then
```

```
      begin
      Bad:=True;
      Writeln('   Face number must be an integer');
      end;
   If IRS[6]=-1 then
      begin
      Bad:=True;
      Writeln('   Traction value must be given');
      end;
{
                  Check FromElement<=ToElement
}
   If Bad=False then
      begin
      FromElement:=Ni[1];
      Case IRS[2] of
        -1: ToElement:=FromElement;
         0: ToElement:=Ni[2];
         end;
      If ToElement<FromElement then
         begin
         writeln('   FromElement must be less or equal to ToElement');
         Bad:=True;
         end;
      end;
{
                  Get number of times pattern is repeated
}
   If Bad=False then
      If IRS[3]=-1 then
         Repete:=0
      else
         begin
         Repete:=Ni[3];
         If Repete<0 then
            begin
            Bad:=True;
            writeln('   Repeat < 0');
            end;
         end;
{
                  Get increase in element numbers
}
   If Bad=False then
      If IRS[4]=-1 then
         dElement:=ToElement-FromElement+1
      else
         begin
         dElement:=Ni[4];
         If dElement<ToElement-FromElement+1 then
            begin
            Bad:=True;
```

```
            writeln('   dElement < ToElement-FromElement+1');
            end;
         end;
{
                  Check if ToElement+Repete.dElement <= Net
}
   If Bad=False then
      If Net<ToElement+Repete*dElement then
         begin
         Bad:=True;
         writeln('   ToElement+Repeat*dElement beyond last element defined (',
                Net,')');
         end;
{
                  Get face number and Traction value
}
   If Bad=False then
      begin
      Face:=Ni[5];
      Case IRS[6] of
         0: Traction:=Ni[6];
         1: Traction:=Nr[6];
         end;
      end;
{
                  Get Etype
                  Check face number
}
   iE:=FromElement;
   iRepete:=0;
   While (Bad=False) and (iE<=ToElement) and (iRepete<=Repete) do
      begin
      Nne2_I_Read(F_Elem,iE+iRepete*dElement,NneMax,Elem);
      Etype:=Elem[2];
      case Etype of
         1: If ((Face<1)or(3<Face)) then
            begin
            writeln('   Face must be in 1-3 range');
            Bad:=True;
            end;
         2: If ((Face<1)or(6<Face)) then
            begin
            writeln('   Face must be in 1-6 range');
            Bad:=True;
            end;
         end;
{
                  Set Traction on face
}
       If Bad=False then
         begin
         Write('   ',(ToElement+Repete*dElement)-(iE+iRepete*dElement)+1,' ');
```

```
        Get_Exyz(Elem,F_xyz,Exyz);
        Case Etype of
           1: Traction1(Face,Traction,Exyz,Tindex,Nt,Ft);
           2: Traction2(Face,Traction,Exyz,Tindex,Nt,Ft);
           end;
{
                Add to existing forces
}
        For iNt:=1 to Nt do
           begin
           iNne:=Tindex[iNt];
           Node:=Elem[2+iNne];
           D_R_read(F_Force,Node,Dim,Force);     { read previous forces }
           For iD:=1 to Dim do
              Force[iD]:=Force[iD]+Ft[iNne,iD];  { add to forces }
           D_R_write(F_Force,Node,Dim,Force);
           D_Byte_0(F_Fd,Node,Dim);               { set forces as known }
           end;
        GotoXY(1,WhereY);
        Write('              ');
        GotoXY(1,WhereY);
        end;
{
                Increment loop index
}
     If iE<ToElement then
        iE:=iE+1
     else
        begin
        iE:=FromElement;
        iRepete:=iRepete+1;
        end;
     end;
end;
{-------------------------------------------------------------------}
end.
```

K_VAR.PAS

```
{$N+}
Unit K_Var;
Interface
Uses
   Crt,Declare,Sonore;
Procedure Set_Var(     IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Ns80: VectS80_Nc;
                       Qtt: IType;
                   var VarName,VarValue: VectS80_Var;
                   var VarQtt: Byte;
                   var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Set_Var;
{
Procedure Set_Var(     IRS,Ni: VectI_Nc;
                       Nr: VectR_Nc;
                       Ns80: VectS80_Nc;
                       Qtt: IType;
                   var VarName,VarValue: VectS80_Var;
                   var VarQtt: Byte;
                   var Bad: Boolean);
}
{
   Input:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Nr[i]: RType number at the (i)th position
      Ni[i]: IType number at the (i)th position
      Ns80[i]: String at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
          of command line)
      VarName[1..Var_max]: variable names
      VarValue[1..Var_max]: variable values
      VarQtt: quantity of variables
   Output:
      VarName[1..Var_max]: variable names
      VarValue[1..Var_max]: variable values
      VarQtt: quantity of variables
      Bad: True if there is an error in the command parameters
           False Otherwise
   Command Format:
      VAR VariableName,VariableValue;
      ------------------------------
               s          irs        (i=integer,r=real, s=string)
}
var
```

```
   i: integer;
   Name,Mot: String255;
   Redefine: Boolean;
begin
   Bad:=False;
   Redefine:=False;
   If Qtt<>2 then
     begin
     writeln('  Must have 2 parameters');
     Bad:=True;
     end;
   If IRS[1]=-1 then
     begin
     writeln('  Variable name must be given');
     Bad:=True;
     end;
{
                integer or real -> previously defined -> ignore
}
   If IRS[1] in [0,1]  then
     begin
     Redefine:=True;
     WarningSound;
     TextColor(LightRed+Blink);
     Write('  Warning');
     TextColor(LightGreen);
     write(', ignored due to redefinition. First parameter is ');
     Case IRS[1] of
        0: writeln('an integer number');
        1: writeln('a real number');
        end;
     end;
   If (IRS[1]=2) and (Bad=False) then   (first time this variable name is used)
     begin
     If IRS[2]=-1 then
        begin
        writeln('   Variable value must be given');
        Bad:=True;
        end;
{
                Convert to upper case
}
     If Bad=False then
        begin
        Name:=Ns80[1];
        For i:=1 to length(Name) do
           Name[i]:=UpCase(Name[i]);
        end;
     end;
{
                Check if maximum number of variables is reached
}
```

```
   If bad=False then
      If Var_max<=VarQtt then
         begin
         writeln('   Too many variables (max ',Var_max,')');
         bad:=True;
         end;
{
                  Do not re-define a variable (when Redefine=True)
}
   If (Bad=False) and (Redefine=False) then
      begin
      VarName[VarQtt+1]:=Name;
      Case IRS[2] of
         0: Str(Ni[2],Mot);
         1: Str(Nr[2],Mot);
         2: Mot:=Ns80[2];
         end;
      VarValue[VarQtt+1]:=Mot;
      Inc(VarQtt);
      end;
end;
{-------------------------------------------------------------------------}
end.
```

LINEAR.PAS

```
{$N+}
Unit Linear;
Interface
Uses
   Declare;
Function Lin(    row,col,col_max: IType): IType;
{-------------------------------------------------------------------}
Implementation
Function Lin;
{
Function Lin(    row,col,col_max: IType): IType;
}
{
   Give vector position of a matrix position
   Input:
      row: row
      col: column
      col_max: max # of columns
   Output:
      Lin: position in vector
}
begin
   Lin:= col_max*(row-1)+col;
end;
{-------------------------------------------------------------------}
end.
```

LINES.PAS

```
{-------------------------------------------------------------------------
   Program to count number of lines in all Pascal files
   Initially written in May '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
Uses
   Printer,Dos,Crt;
Type
   String255 = String[255];
{------------------------- -------------------------------------------}
Procedure LineCount(    FromDir,Filter: String255;
                        OnPrinter: Boolean);
{
   Input:
      FromDir: Directory FROM where files are
      Filter: filter for files
      OnPrinter: True for printer copy
}
Const
   MaxFiles = 3000;
var
   i,j,iFile,FileQtt,FileLines,TotalLines: LongInt;
   FileName,LineString: String255;
   DirInfo: SearchRec;
   NameOfFile: array[1..MaxFiles] of String[12];
   F: Text;
begin
   FileQtt:=0;
   FindFirst(FromDir+Filter,Archive,DirInfo);
   While DosError=0 do
      begin
      Inc(FileQtt);
      If MaxFiles<FileQtt then
         begin
         writeln('Too many files: over ',MaxFiles);
         Halt;
         end;
      NameOfFile[FileQtt]:=DirInfo.Name;
      FindNext(DirInfo);
      end;
{
                  Sort files by bubble sort
}
   For i:=1 to FileQtt do
      begin
      gotoXY(1,WhereY);
      write(FileQtt-i+1,' ');
      For j:=FileQtt-1 downto 1 do
         If NameOfFile[j]>NameOfFile[j+1] then
            begin
```

```
            FileName:=NameOfFile[j];
            NameOfFile[j]:=NameOfFile[j+1];
            NameOfFile[j+1]:=FileName;
            end;
      end;
   gotoXY(1,WhereY);
{
                  Count number of line for each file
}
   TotalLines:=0;
   For iFile:=1 to FileQtt do
      begin
      write(NameOfFile[iFile]:12,' ');
      FileLines:=0;
      Assign(F,FromDir+NameOfFile[iFile]);
      Reset(F);
      While EOF(F)=False do
         begin
         Readln(F);
         Inc(FileLines);
         end;
      Close(F);
      TotalLines:=TotalLines+FileLines;
      writeln(FileLines:10,' ',TotalLines:10);
      If OnPrinter=True then
         writeln(Lst,NameOfFile[iFile]:12,' ',FileLines:10);
      end;
   writeln('TOTAL':12,' ',TotalLines:10);
   If OnPrinter=True then
      writeln(Lst,'TOTAL':12,' ',TotalLines:10);
end;
{=============================================================================}
var
   FromDir,Filter: String255;
   OnPrinter: Boolean;
begin
   FromDir:='\tp\fe\';
   Filter:='*.PAS';
   OnPrinter:=True;
   LineCount(FromDir,Filter,OnPrinter);
end.
```

LISTING.PAS

```
{$M 60000,0,0}
{
    Program to produce listing of a list of files (Programs)
-------------------------------------------------------------------------
}
Uses
   Crt,Dos,Printer,WaitKey,Sonore;
Const
   MaxFiles = 1000;
Type
   String255 = String[255];
   FileString = String[12];
   VectFS_P = array[1..MaxFiles] of FileString;
Var
   PageToPrinter,DoPrint: Boolean;
{-------------------------------------------------------------------------}
Procedure UpCaseFileNameVect(var FileNameVect: VectFS_P;
                                FileQtt: integer);
var
   i,j: integer;
begin
   For i:=1 to FileQtt do
      For j:=1 to length(FileNameVect[i]) do
         FileNameVect[i][j]:=UpCase(FileNameVect[i][j]);
end;
{-------------------------------------------------------------------------}
Procedure GetFileNameVect(var FromDir: String255;
                             Filter: String255;
                          var FileNameVect:VectFS_P;
                          var FileQtt: Integer);
{
   Input:
      FromDir: Directory FROM where files are
      Filter: filter for files
   Output:
      FromDir: Directory FROM where files are
      FileNameVect[1..N]: Get all files
      FileQtt: quantity of files
}
var
   DirInfo: SearchRec;
begin
   If 0<Length(FromDir) then
      begin
      If FromDir[Length(FromDir)]<>'\' then
         FromDir:=FromDir+'\';
      end;
   FileQtt:=0;
   FindFirst(FromDir+Filter,Archive,DirInfo);
   While DosError=0 do
```

```
      begin
      Inc(FileQtt);
      If MaxFiles<FileQtt then
         begin
         writeln('Too many files: over ',MaxFiles);
         Halt;
         end;
      FileNameVect[FileQtt]:=DirInfo.Name;
      FindNext(DirInfo);
      end;
end;
{-------------------------------------------------------------------}
Procedure OrderFileNameVect(var FileNameVect: VectFS_P;
                                FileQtt: integer);
{
   Input:
      FileNameVect[1..N]: Get all files in alphabetical order
      FileQtt: quantity of files
   Output:
      FileNameVect[1..N]: Get all files in alphabetical order
      FileQtt: quantity of files
}
var
   i,j: integer;
   Temporary: FileString;
   Xs,Ys: Byte;
begin
   Xs:=WhereX;
   Ys:=WhereY;
{
                  Sort files by bubble sort
}
   For i:=1 to FileQtt do
      begin
      gotoXY(Xs,Ys);
      write(FileQtt-i+1,' ');
      For j:=FileQtt-1 downto 1 do
         If FileNameVect[j]>FileNameVect[j+1] then
            begin
            Temporary:=FileNameVect[j];
            FileNameVect[j]:=FileNameVect[j+1];
            FileNameVect[j+1]:=Temporary;
            end;
      end;
   gotoXY(Xs,Ys);
   write(' ');
   gotoXY(Xs,Ys);
end;
{-------------------------------------------------------------------}
Procedure VerifyTwice(var FileNameVect: VectFS_P;
                          FileQtt: integer);
{
```

```
   Input
      FileNameVect[1..N]: all file names
      FileQtt: quantity of files
}
var
   i,j: integer;
begin
   For i:=1 to FileQtt-1 do
      For j:=i+1 to FileQtt do
         If FileNameVect[i]=FileNameVect[j] then
            begin
            writeln('Same file repeated: ',FileNameVect[i]);
            Wait;
            Halt;
            end;
end;
{-------------------------------------------------------------------------}
Procedure ExistFile(    FromDir: String255;
                    var FileNameVect: VectFS_P;
                        FileQtt: integer);
{
   Input
      FromDir: Directory FROM where files are
      FileNameVect[1..N]: all file names
      FileQtt: quantity of files
}
var
   i: integer;
   F: Text;
begin
   For i:=1 to FileQtt-1 do
      begin
      Assign(F,FromDir+FileNameVect[i]);
      Reset(F);
      If IoResult=0 then
         begin
         Close(F);
         end
      else
         begin
         writeln('This file doesn''t exist: ',FromDir+FileNameVect[i]);
         Wait;
         Halt;
         end;
      end;
end;
{-------------------------------------------------------------------------}
Procedure SetUpFX85(var cpi,lpi: real);
{
   Output
      cpi: characters/inch
      lpi: lines/inch
```

```
}
begin
   If DoPrint=True then
      begin
      write(Lst,CHR(27)+'@',chr(13));            {initialize}
      writeln('Align printer head with perforation');
      writeln('then press any key');
      Wait;
      write(Lst,CHR(27)+'3'+chr(31)+chr(13));    {7 lines / inch}
      write(Lst,CHR(15),chr(13));                {Compressed}
      end;
   cpi:=15;
   lpi:=7;
end;
{-------------------------------------------------------------------}
Procedure SetUpLaser(var cpi,lpi: real);
{
   Output
      cpi: characters/inch
      lpi: lines/inch
}
begin
   If DoPrint=True then
      begin
      write('Select 16.6 cpi + 6lpi');
      wait;
      end;
   cpi:=16.6;
   lpi:=6;
end;
{-------------------------------------------------------------------}
Procedure Header(    Page: integer;
                     LeftMargin,RightColumn,
                        TopMargin,NumberColumn,NumberLine: Byte;
                     PagePrefix,PageSuffix: String;
                 var Line: Byte;
                     FileName: FileString);
{
   Input:
      Page: current page number          -> 45
      LeftMargin: Left margin (characters)
      RightColumn: Last column used for text
      TopMargin: top margin (lines)
      NumberColumn: starting column for page number location
      NumberLine: starting line for page number location
      PagePrefix: prefix to page number   -> '-A'
      PageSuffix: Suffix to page number   -> '-'    => -A45-
      Line: current line position
      FileName: File Name (given when desired to print)
   Output:
      Line: current line position
}
```

```
var
   CenterColumn,iLine,i: integer;
begin
   If DoPrint=True then
      begin
      For iLine:=1 to TopMargin do
         begin
         If iLine=NumberLine then
            begin
            For i:=1 to NumberColumn-1 do
               write(Lst,' ');
            write(Lst,PagePrefix,Page,PageSuffix);
            end;
         writeln(Lst);
         end;
      end;
   Line:=TopMargin+1;
{
                  Put file name on line 2
}
   If 0<Length(FileName) then
      begin
      CenterColumn:=LeftMargin
                  +Trunc((RightColumn-LeftMargin-1-Length(FileName))/2);
      If DoPrint=True then
         begin
         For i:=1 to CenterColumn-1 do
            write(Lst,' ');
         write(Lst,FileName,chr(13));
         For i:=1 to CenterColumn-1 do
            write(Lst,' ');
         writeln(Lst,FileName);
         writeln(Lst);
         end;
      Line:=Line+2;
      end;
end;
{-------------------------------------------------------------------------}
Procedure GetLine(var F: Text;
                      CommentOk: Boolean;
                  var InComment: Boolean;
                  var Ls: String255);
{
   Input:
      F: file variable
      CommentOk: True we are in a comment at the beginning of the line
   Output:
      InComment: True if we are still in a comment at the end of the line
      Ls: next valid line to print
}
var
   Line: String255;
```

```
   C: char;
   LineOk: Boolean;
   i: Byte;
begin
   If CommentOk=False then
      begin
      Readln(F,Line);
      Ls:='';
      For i:=1 to Length(Line) do
         begin
         C:=Line[i];
         Case C of
            '(': InComment:=True;
            ')': InComment:=False;
         else
            If InComment=False then
               Ls:=Ls+C;
            end;
         end;
{
                  Set line to nul when only blanks are found
}
      LineOk:=False;
      For i:=1 to Length(Ls) do
         If not (Ls[i] in [' ',chr(13)]) then LineOk:=True;
      If LineOk=False then Ls:='';
      end
   else
      begin
      readln(F,Ls
      end;
end;
{------------------------------------------ ------------------------}
Procedure ASCII_Line(    TabInsert: byte;
                     var L:String255);
{
   Input:
      TabInsert: number of spaces to replace a tab character (#9)
      L: a line of normal and special characters
   Ouput:
      L: a line of normal characters
}
var
   Xs,Ys,i: Byte;
   L2: string;
begin
   For i:=1 to Length(L) do
      begin
      If not (L[i] in [' '..'~']) then
         begin
         Repeat
            Case L[i] of
```

```
'a': L[i]:='a';
'e': L[i]:='e';
'i': L[i]:='i';
'u': L[i]:='u';
'a': L[i]:='a';
'e': L[i]:='e';
'u': L[i]:='u';
'a': L[i]:='a';
'e': L[i]:='e';
'i': L[i]:='i';
'o': L[i]:='o';
'u': L[i]:='u';
'e': L[i]:='e';
'c': L[i]:='c';
'S': L[i]:='S';
'e': L[i]:='e';
'S': L[i]:='S';
'P': L[i]:='P';
'<': L[i]:='<';
'>': L[i]:='>';
'>': L[i]:='>';
'<': L[i]:='<';
'¯': L[i]:='¯';
'x': L[i]:='x';
'¦': L[i]:='¦';
'¦': L[i]:='¦';
'-': L[i]:='-';
'-': L[i]:='-';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'+': L[i]:='+';
'X': L[i]:='X';
'X': L[i]:='X';
```

```
                'X': L[i]:='X';
                'X': L[i]:='X';
                '2': L[i]:='2';
{
                    Insert spaces for horizontal tab character
}
                chr(9): begin
                        L2:=Copy(L,1,i-1);
                        For i:=1 to TabInsert do
                           L2:=L2+' ';
                        L:=L2+Copy(L,i+1,Length(L)-i);
                        end;
             else
                Bad_Beep;
                Xs:=WhereX;
                Ys:=WhereY;
                writeln;
                writeln(L);
                write('character (',L[i],') (#',ord(L[i]),') should become: ');
                readln(L[i]);
                GotoXY(1,Ys+1);
                ClrEOL;
                GotoXY(1,Ys+2);
                ClrEOL;
                GotoXY(Xs,Ys);
                end;
             until L[i] in [' '..'~'];
          end;
       end;
end;
{-------------------------------------------------------------------------}
Procedure PrintFile(    FileName: FileString;
                        FromDir: String255;
                    var Page: integer;
                        TabInsert: byte;
                        PagePrefix,PageSuffix: String;
                        NumberColumn,NumberLine,TopMargin,LeftMargin,
                           LineMax,RightColumn: Byte;
                        CommentOk: Boolean);
{
   Input:
      FileName: File Name
      FromDir: Directory FROM where files are
      Page: current page number          -> 45
      TabInsert: number of spaces to replace a tab character (#9)
      PagePrefix: prefix to page number  -> '-A'
      PageSuffix: Suffix to page number  -> '-'    => -A45-
      NumberColumn: starting column for page number location
      NumberLine: starting line for page number location
      TopMargin: top margin (lines)
      LeftMargin: Left margin (characters)
      LineMax: last line used in a page
```

```
      RightColumn: Last column used for text
      CommentOk: True we are in a comment at the beginning of the line
   Output:
      Page: New page number after file is printed
}
var
   Col,Xs,Ys,LineLength,Line: Byte;
   FileLine: LongInt;
   F: Text;
   Ls: String255;
   InComment: Boolean;
begin
   If DoPrint=True then PageToPrinter:=false;
   If PageToPrinter=true then write(Lst,FileName,' pp.: ',Page);
   write(FileName);
   gotoXY(13,WhereY);
   write(' pp.: ',Page);
   Xs:=WhereX;
   Ys:=WhereY;
   If Ys=25 then Ys:=Ys-1;
   writeln;
   FileLine:=0;
   LineLength:=RightColumn-LeftMargin+1;
   Header(Page,LeftMargin,RightColumn,TopMargin,NumberColumn,NumberLine,
          PagePrefix,PageSuffix,Line,FileName);
   Assign(F,FromDir+FileName);
   Reset(F);
   While EOF(F)=False do
      begin
      Inc(FileLine);
      GotoXY(4,Ys+1);
      write('File:',FileLine,' Page:',Page,' Line:',Line,' ');
      GetLine(F,CommentOk,InComment,Ls);
      ASCII_Line(TabInsert,Ls);
{
                  Skip blank lines
}
      While 0<Length(Ls) do
         begin
         If Line=LineMax+1 then
            begin
            If DoPrint=True then Write(Lst,Chr(12),chr(13));
            Inc(Page);
            Header(Page,LeftMargin,RightColumn,TopMargin,NumberColumn,NumberLine,
                   PagePrefix,PageSuffix,Line,'');
            end;
         If DoPrint=True then
            begin
            For Col:=1 to LeftMargin do
               write(lst,' ');
            writeln(Lst,Copy(Ls,1,LineLength));
            end;
```

```
        Delete(Ls,1,LineLength);
        Inc(Line);
        end;
      end;
   If DoPrint=True then Write(Lst,Chr(12),chr(13));
   Inc(Page);
   Close(F);
   GotoXY(Xs,Ys);
   write('-',Page-1);
   If PageToPrinter=true then writeln(lst,'-',Page-1);
   GotoXY(1,Ys+1);
   ClrEOL;
end;
{-------------------------------------------------------------------------}
var
   FileNameVect: VectFS_P;
   CharPerLine,Page,i,FileQtt: integer;
   TabInsert: Byte;
   EndFileName,StartFileName,PagePrefix,PageSuffix: String;
   NumberColumn,NumberLine,TopMargin,LeftMargin,LineMax,RightColumn: Byte;
   CommentOk: Boolean;
   FromDir,Filter: String255;
   cpi,lpi: real;
begin
   ClrScr;
{
   SetUpFX85(cpi,lpi);
}
   SetUpLaser(cpi,lpi);
   PageToPrinter:=False;
   DoPrint:=True;
   FromDir:='\tp\fe';
   Filter:='*.pas';
   Page:=5;
   TabInsert:=3;
   PagePrefix:='- B ';
   PageSuffix:=' -';
   NumberLine:=Trunc(lpi*0.2);              {0.5}
   TopMargin:=trunc(lpi*0.8);               {1.1}
   CharPerLine:=Trunc(cpi*8.5);             {8.5}
   LeftMargin:=Trunc(CharPerLine*1.6/8.5);  {1.6}
   LineMax:=Round(lpi*9.4);                 {9.7}
   RightColumn:=Trunc(CharPerLine*7.4/8.5); {7.4}
   NumberColumn:=LeftMargin
                 +Trunc((RightColumn-LeftMargin-1)/2)
                 -Length(PagePrefix);
   CommentOk:=True;
   StartFileName:='AAAAAAAA.PAS';  {in upper case}
   EndFileName  :='ZZZZZZZZ.PAS';  {in upper case}
   GetFileNameVect(FromDir,Filter,FileNameVect,FileQtt);
   UpCaseFileNameVect(FileNameVect,FileQtt);
   VerifyTwice(FileNameVect,FileQtt);
```

```
   OrderFileNameVect(FileNameVect,FileQtt);
   ExistFile(FromDir,FileNameVect,FileQtt);
   For i:=1 to FileQtt do
      begin
      If (StartFileName<=FileNameVect[i]) and
         (FileNameVect[i]<=EndFileName) then
         begin
         PrintFile(FileNameVect[i],FromDir,Page,TabInsert,
                   PagePrefix,PageSuffix,
                   NumberColumn,NumberLine,TopMargin,LeftMargin,
                   LineMax,RightColumn,CommentOk);
         end;
      end;
end.
```

LOCGLO.PAS

```
{$N+}
Unit LocGlo;
Interface
   Uses Declare,Math;
Procedure LocalGlobal(    Laxis: MatrixR_3x3;
                          LocalType: Char;
                          Origin: VectR_3;
                          LxyzRCS: VectR_D;
                          Dim: Byte;
                      var GxyzR: VectR_D);
Procedure GlobalLocal(    Laxis: MatrixR_3x3;
                          LocalType: Char;
                          Origin: VectR_3;
                          GxyzR: VectR_D;
                          Dim: Byte;
                      var LxyzRCS: VectR_D);
{-------------------------------------------------------------------------}
Implementation
Procedure RCStoR(    LxyzRCS: VectR_D;
                     LocalType: Char;
                 var LxyzR: VectR_D);
{
   Input:
      LxyzRCS: Local coordinates in Rectangular coordinate system (x,y,z)
                                    Cylindrical                   (r,theta,z)
                                    Spherical                     (r,theta,phi)
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
   Output:
      LxyzR: Local Rectangular coordinate
}
begin
   Case LocalType of
      'R': begin
           LxyzR[1]:=LxyzRCS[1];
           LxyzR[2]:=LxyzRCS[2];
           LxyzR[3]:=LxyzRCS[3];
           end;
      'C': begin
           LxyzR[1]:=LxyzRCS[1]*cos(LxyzRCS[2]);
           LxyzR[2]:=LxyzRCS[1]*sin(LxyzRCS[2]);
           LxyzR[3]:=LxyzRCS[3];
           end;
      'S': begin
           LxyzR[1]:=LxyzRCS[1]*sin(LxyzRCS[3])*cos(LxyzRCS[2]);
           LxyzR[2]:=LxyzRCS[1]*sin(LxyzRCS[3])*sin(LxyzRCS[2]);
           LxyzR[3]:=LxyzRCS[1]*cos(LxyzRCS[3]);
           end;
      end;
```

```
end;
{-------------------------------------------------------------------}
Procedure TransRotToG(    Laxis: MatrixR_3x3;
                          Origin: VectR_3;
                          LxyzR: VectR_D;
                          Dim: Byte;
                      var GxyzR: VectR_D);
{
   Input:
      Origin: coordinates or origin of local coordinate system
        = ¦ x0 y0 z0 ¦
      Laxis: Direction of local coordinate system axis
        = ¦ l1 m1 n1 ¦
          ¦ l2 m2 n2 ¦
          ¦ l3 m3 n3 ¦
      LxyzR: Local coordinates, rectangular coordinate system
      Dim: Dimension of geometry
   Output:
      GxyzR: Global coordinates, rectangular coordinate system
}
var
   iD: integer;
begin
   For iD:=1 to Dim do
      GxyzR[iD]:=Laxis[1,iD]*LxyzR[1]
                +Laxis[2,iD]*LxyzR[2]
                +Laxis[3,iD]*LxyzR[3]
                +Origin[iD];
end;
{-------------------------------------------------------------------}
Procedure LocalGlobal;
{
Procedure LocalGlobal(    Laxis: MatrixR_3x3;
                          LocalType: Char;
                          Origin: VectR_3;
                          LxyzRCS: VectR_D;
                          Dim: Byte;
                      var GxyzR: VectR_D);
}
{
   Input:
      Laxis: Direction of local coordinate system axis
        = ¦ l1 m1 n1 ¦
          ¦ l2 m2 n2 ¦
          ¦ l3 m3 n3 ¦
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
      Origin: coordinates or origin of local coordinate system
        = ¦ x0 y0 z0 ¦
      LxyzRCS: Local coordinates in Rectangular coordinate system (x,y,z)
                                    Cylindrical                  (r,theta,z)
```

```
                                    Spherical                    (r,theta,phi)
      Dim: Dimension of geometry
   Output:
      GxyzR: Global coordinates, rectangular coordinate system
}
var
   LxyzR: VectR_D;
begin
   RCStoR(LxyzRCS,LocalType,LxyzR);
   TransRotToG(Laxis,Origin,LxyzR,Dim,GxyzR);
end;
{-------------------------------------------------------------------}
Procedure RtoRCS(    LxyzR: VectR_D;
                     LocalType: Char;
                 var LxyzRCS: VectR_D);
{
   Input:
      LxyzR: Local Rectangular coordinate
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
   Output:
      LxyzRCS: Local coordinates in Rectangular coordinate system (x,y,z)
                                    Cylindrical                  (r,theta,z)
                                    Spherical                    (r,theta,phi)
}
begin
   Case LocalType of
      'R': begin
           LxyzRCS[1]:=LxyzR[1];
           LxyzRCS[2]:=LxyzR[2];
           LxyzRCS[3]:=LxyzR[3];
           end;
      'C': begin
           LxyzRCS[1]:=sqrt(sqr(LxyzR[1])+sqr(LxyzR[2]));  { R }
           LxyzRCS[2]:=ArcTan4(LxyzR[1],LxyzR[2]);         { angle with X }
           LxyzRCS[3]:=LxyzR[3];                           { Z }
           end;
      'S': begin
           LxyzRCS[1]:=sqrt(sqr(LxyzR[1])+sqr(LxyzR[2])+sqr(LxyzR[3]));  { r }
           LxyzRCS[2]:=ArcTan4(LxyzR[1],LxyzR[2]);         { angle with X }
           LxyzRCS[3]:=ArcCos(LxyzR[3]/LxyzRCS[1]);        { angle with Z }
           end;
      end;
end;
{-------------------------------------------------------------------}
Procedure TransRotToL(    Laxis: MatrixR_3x3;
                          Origin: VectR_3;
                          GxyzR: VectR_D;
                          Dim: Byte;
                      var LxyzR: VectR_D);
{
```

```
   Input:
      Origin: coordinates or origin of local coordinate system
         = ¦ x0 y0 z0 ¦
      Laxis: Direction of local coordinate system axis
         = ¦ l1 m1 n1 ¦
           ¦ l2 m2 n2 ¦
           ¦ l3 m3 n3 ¦
      GxyzR: Global coordinates, rectangular coordinate system
      Dim: Dimension of geometry
   Output:
      LxyzR: Local coordinates, rectangular coordinate system
}
var
   iD: integer;
begin
   For iD:=1 to Dim do
      LxyzR[iD]:=Laxis[iD,1]*(GxyzR[1]-Origin[1])
                +Laxis[iD,2]*(GxyzR[2]-Origin[2])
                +Laxis[iD,3]*(GxyzR[3]-Origin[3]);
end;
{-------------------------------------------------------------------}
Procedure GlobalLocal;
{
Procedure GlobalLocal(    Laxis: MatrixR_3x3;
                          LocalType: Char;
                          Origin: VectR_3;
                          GxyzR: VectR_D;
                          Dim: Byte;
                      var LxyzRCS: VectR_D);
}
{
   Input:
      Laxis: Direction of local coordinate system axis
         = ¦ l1 m1 n1 ¦
           ¦ l2 m2 n2 ¦
           ¦ l3 m3 n3 ¦
      LocalType: "R" for Rectangular
                 "C" for Cylindrical
                 "S" for Spherical
      Origin: coordinates or origin of local coordinate system
         = ¦ x0 y0 z0 ¦
      GxyzR: Global coordinates, rectangular coordinate system
      Dim: Dimension of geometry
   Output:
      LxyzRCS: Local coordinates in Rectangular coordinate system (x,y,z)
                                    Cylindrical                   (r,theta,z)
                                    Spherical                     (r,theta,phi)
}
var
   LxyzR: VectR_D;
begin
   TransRotTol(Laxis,Origin,GxyzR,Dim,LxyzR);
```

```
   RtoRCS(LxyzR,LocalType,LxyzRCS);
end;
{-------------------------------------------------------------------------}
end.
```

LOGO.PAS

```
Unit Logo;
Interface
Uses
   Crt;
Procedure BatchLogo;
{-------------------------------------------------------------------------}
Implementation
Procedure BatchLogo;
{
                  Identifies the origin of this program when called
                  in batch mode with one ore more parameters
                  Ir  anded for PREP,SOLVE,WORST,STRESS
}
begin
   If 0<ParamCount then
      begin
      ClrScr;
      HighVideo;
      GotoXY(15,5);      writeln('+-----------------------------------------------------+');
      GotoXY(15,WhereY); writeln('| The following process is part of DIRECT created by |');
      GotoXY(15,WhereY); writeln('|                                                     |');
      GotoXY(15,WhereY); writeln('| J. Daoust Ph.D. and S.V. Hoa Ph.D..                 |');
      GotoXY(15,WhereY); writeln('|                                                     |');
      GotoXY(15,WhereY); writeln('| For a complete version of DIRECT,                   |');
      GotoXY(15,WhereY); writeln('|                                                     |');
      GotoXY(15,WhereY); writeln('| information or assistance, write or call:           |');
      GotoXY(15,WhereY); writeln('|                                                     |');
      GotoXY(15,WhereY); writeln('| Dr. S.V. Hoa                                        |');
      GotoXY(15,WhereY); writeln('| Composite Material Laboratory                       |');
      GotoXY(15,WhereY); writeln('| Department of Mechanical Engineering                |');
      GotoXY(15,WhereY); writeln('| Concordia University                                |');
      GotoXY(15,WhereY); writeln('| 1455 de Maisonneuve West                            |');
      GotoXY(15,WhereY); writeln('| Montreal, Quebec, Canada, H3G 1M8                   |');
      GotoXY(15,WhereY); writeln('| (514) 848-3139                                      |');
      GotoXY(15,WhereY); writeln('| (514) 848-8796                                      |');
      GotoXY(15,WhereY); writeln('+-----------------------------------------------------+');
      Delay(10000);
      NormVideo;
      end;
end;
{-------------------------------------------------------------------------}
end.
```

MANAGER.PAS

```
{-------------------------------------------------------------------
   Case Manager
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}
{$M 10000,0,0}
Uses
   Dos,Crt,Declare,Where,WhatKey,WaitKey,PickFile,Status,Param;
{-------------------------------------------------------------------}
procedure menu;
var
   Xleft,Yupper: Byte;
begin
   ClrScr;
   TextColor(LightGreen);
   gotoxy(1,10);
   Xleft:=WhereX;
   Yupper:=WhereY;
   writeln('+-------------------------+');
   writeln('¦ Save current case       ¦');
   writeln('¦ Restore a case          ¦');
   writeln('¦ Keep case for refinement ¦');
   writeln('¦ Delete a previous case  ¦');
   writeln('¦ <Esc>                   ¦');
   writeln('+-------------------------+');
   TextColor(LightRed);
   gotoXY(Xleft+2,Yupper+1); write('S');
   gotoXY(Xleft+2,Yupper+2); write('R');
   gotoXY(Xleft+2,Yupper+3); write('K');
   gotoXY(Xleft+2,Yupper+4); write('D');
   gotoXY(Xleft+2+1,Yupper+5); write('Esc');
   TextColor(LightGreen);
end;
{-------------------------------------------------------------------}
Procedure SaveCase(    SubDir: String255);
(
   Input:
      CaseName: name of case found in "Status.P" file
      SubDir: sub-directory from S_Dir to which case is saved
            if SubDir='' then it will be asked
            if SubDir<>'' then it is accepted
   Output:
      Saves current *.P files in C_Dir directory to a selected directory
)
var
   NewSubDir,CaseName,CoarseCaseName,CurentCoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
```

```
begin
   ClrScr;
   Read_Status(CaseName,CurentCoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   If CaseName='Unknown' then
      begin
      gotoXY(20,10);
      writeln('The case is undefined!');
      wait;
      end
   else
      begin
{
                  Temporarily set CoarseCaseName = Unknown
                  so all *.P file are copied alike including status
                  (but the coarse case name should not be copied)
                  After copying restore original coarse case name
}
      CoarseCaseName:='Unknown';
      Write_Status(CaseName,CoarseCaseName,
                  StatusPrep,StatusForce,StatusDispl,StatusWorst,
                  StatusFiberStrain,StatusFiberStress,
                  StatusElemStrain,StatusElemStress,
                  StatusGlobalStrain,StatusGlobalStress);
{
                  Ask Sub-Directory if not given
}
      If SubDir='' then
         begin
         gotoXY(1,10);
         write('Current case is: ');
         TextColor(LightRed);
         writeln(CaseName);
         TextColor(LightGreen);
         write('*.P files are presently kept in ');
         TextColor(LightRed);
         writeln(C_Dir);
         TextColor(LightGreen);
         SubDir:=S_Dir+CaseName;
         write('Send to which directory (Default=');
         TextColor(LightRed);
         write(SubDir+'\');
         TextColor(LightGreen);
         write('): ');
         TextColor(LightRed);
         NewSubDir:='';
         readln(NewSubDir);
         TextColor(LightGreen);
         If NewSubDir<>'' then SubDir:=NewSubDir;
```

```
        end
      else
        begin
        SubDir:=S_Dir+SubDir;
        end;
{
                Remove last '\'
}
      If SubDir[Length(SubDir)]='\' then Delete(SubDir,Length(SubDir),1);
{
                Create directory
}
      ClrScr;
      write('Saving case ');
      TextColor(LightRed);
      writeln(CaseName);
      TextColor(LightGreen);
{
                Create directory
                Don't use MkDir it bugs the program when the directory
                  already exist (second time around)
}
      Exec('\COMMAND.COM','/C '+'MD '+SubDir);
{
                Delete previous files in subdirectory
}
      Exec('\COMMAND.COM','/C '+'DEL '+SubDir+'\*.P');
{
                Copy files
}
      Exec('\COMMAND.COM','/C '+'COPY '+C_Dir+'*.P'+' '+SubDir+'\*.*');
{
                Restore Original coarse case name
}
      Write_Status(CaseName,CurentCoarseCaseName,
                StatusPrep,StatusForce,StatusDispl,StatusWorst,
                StatusFiberStrain,StatusFiberStress,
                StatusElemStrain,StatusElemStress,
                StatusGlobalStrain,StatusGlobalStress);
   end;
end;
{-------------------------------------------------------------------------}
Procedure RestoreCase(    RestoreCaseName: String255);
{
   Input:
      All subdirectory names to current directory
      RestoreCaseName: sub-directory from S_Dir to which case is saved
            if RestoreCaseName='' then it will be asked
            if RestoreCaseName<>'' then it is accepted
   Output:
      Restores all *.P to D_Dir directory
}
```

```
var
   SubDir,CaseName,CoarseCaseName,
      CurrentCoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   File_ok: Boolean;
begin
   ClrScr;
   If RestoreCaseName='' then
      begin
      writeln('Which case is to be restored: ');
      Pick_File(S_Dir,'*.',Directory,True,RestoreCaseName,File_ok);
      end
   else
      begin
      File_ok:=True;
      end;
   If File_ok=True then
      begin
{
                  Read current CoarseCaseName because the one in the
                  subdirectoring of the case being restored is set to
                  unknown
                  Clean previous *.P files
                  so all *.P file are copied alike including status
                  (but the coarse case name should not be copied)
                  After copying restore original current coarse case name
}
      Read_Status(CaseName,CurrentCoarseCaseName,
                  StatusPrep,StatusForce,StatusDispl,StatusWorst,
                  StatusFiberStrain,StatusFiberStress,
                  StatusElemStrain,StatusElemStress,
                  StatusGlobalStrain,StatusGlobalStress);
      Exec('\Command.com','/C Del '+C_Dir+'*.P');
      ClrScr;
      write('Restoring case ');
      TextColor(LightRed);
      writeln(RestoreCaseName);
      TextColor(LightGreen);
      SubDir:=S_Dir+RestoreCaseName;
      Exec('\COMMAND.COM','/C '+'COPY '+SubDir+'\*.P'+' '+C_Dir+'*.*');
{
                  Read case name and status, restore what was the
                  current coarse case name
}
      Read_Status(CaseName,CoarseCaseName,
                  StatusPrep,StatusForce,StatusDispl,StatusWorst,
                  StatusFiberStrain,StatusFiberStress,
                  StatusElemStrain,StatusElemStress,
                  StatusGlobalStrain,StatusGlobalStress);
```

```
      Write_Status(CaseName,CurrentCoarseCaseName,
                   StatusPrep,StatusForce,StatusDispl,StatusWorst,
                   StatusFiberStrain,StatusFiberStress,
                   StatusElemStrain,StatusElemStress,
                   StatusGlobalStrain,StatusGlobalStress);
      end;
end;
{-------------------------------------------------------------------------}
Procedure KeepCase;
{
   Input:
      Current *.P files
   Output:
      Keep files pertinent for coarse mesh information
         that will be used for mesh refinement
      Add that case name to status file name
           CurrentCaseName <- CoarseMeshCaseName
}
var
   FileName,CoarseCaseName,CaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
begin
   ClrScr;
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   If (StatusDispl='Done') and (StatusForce='Done') then
      begin
      GotoXY(1,7);
      write('Keeping data pertinent to ');
      TextColor(LightRed);
      write(CaseName);
      TextColor(LightGreen);
      writeln(' as a course mesh');
{
                  Save: Old->New relations
                        number of Old nodes
                        Dimension (1, 2 or 3 dimensional)
                        XYZ coordinates
                        Displacements
}
      FileName:='OldToNew';
      write(FileName+'.C',Copy('        ',1,8-Length(FileName)));
      Exec('\COMMAND.COM','/C '+'Copy '+C_Dir+FileName+'.P '+C_Dir+'*.C');
      FileName:='NntOld';
      write(FileName+'.C',Copy('        ',1,8-Length(FileName)));
      Exec('\COMMAND.COM','/C '+'Copy '+C_Dir+FileName+'.P '+C_Dir+'*.C');
```

```
      FileName:='Dim';
      write(FileName+'.C',Copy('        ',1,8-Length(FileName)));
      Exec('\COMMAND.COM','/C '+'Copy '+C_Dir+FileName+'.P '+C_Dir+'*.C');
      FileName:='XYZ';
      write(FileName+'.C',Copy('        ',1,8-Length(FileName)));
      Exec('\COMMAND.COM','/C '+'Copy '+C_Dir+FileName+'.P '+C_Dir+'*.C');
      FileName:='Displ';
      write(FileName+'.C',Copy('        ',1,8-Length(FileName)));
      Exec('\COMMAND.COM','/C '+'Copy '+C_Dir+FileName+'.P '+C_Dir+'*.C');
      CoarseCaseName:=CaseName;
      Write_Status(CaseName,CoarseCaseName,
                   StatusPrep,StatusForce,StatusDispl,StatusWorst,
                   StatusFiberStrain,StatusFiberStress,
                   StatusElemStrain,StatusElemStress,
                   StatusGlobalStrain,StatusGlobalStress);
      end
   else
      begin
      gotoXY(10,12);
      writeln('Case must be solved to be used as a coarse mesh solution');
      wait;
      end;
end;
{-------------------------------------------------------------------------}
Procedure DelCase;
{
   Input:
      All subdirectory names to current directory
   Output:
      Deletes a subdirectory
}
var
   SubDir,CaseName: String255;
   File_ok: Boolean;
begin
   ClrScr;
   writeln('Which case is to be removed (solution is deleted): ');
   Pick_File(S_Dir,'*.',Directory,True,CaseName,File_ok);
   If File_ok=True then
      begin
      SubDir:=S_Dir+CaseName;
      Exec('\COMMAND.COM','/C '+'Del '+SubDir+'\*.P');
{
                   Delete directory
                   Don't use RmDir it bugs the program when the directory
                      doesn't exist (second time around)
}
      Exec('\COMMAND.COM','/C '+'RD '+SubDir);
      end;
end;
{-------------------------------------------------------------------------}
var
```

```
   C: Char;
   Action: String6;
   iParam,i: integer;
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
begin
   Parameters;
{
                  Read directories
}
   ClrScr;
   TextColor(LightGreen);
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   If ParamCount=0 then
      begin
      Repeat
         ClrScr;
         Menu;
         Get_Key(C,Action);
         C:=UpCase(C);
         If C='S' then SaveCase('');
         If C='R' then RestoreCase('');
         If C='K' then KeepCase;
         If C='D' then DelCase;
         until Action='ESC';
      end
   else
      begin
{
                   Act under batch mode
                      MANAGER Save SubDirectory
                      MANAGER Keep
                      MANAGER Restore SubDirectory
}
      If ParameterString[1]='SAVE' then
         Case ParamCount of
            1: SaveCase(CaseName);     { default }
            2: SaveCase(ParameterString[2]);
            end;
      If ParamCount=1 then
         If ParameterString[1]='KEEP' then
            KeepCase;
      If ParamCount=2 then
         If ParameterString[1]='RESTORE' then
```

```
            RestoreCase(ParameterString[2]);
        end;
end.
```

MATERIAL.PAS

```
{-------------------------------------------------------------------
   Program to edit material properties
   Initially written in April '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+} { for math coprocessor }
{$M 11000,0,0}
Uses
   Declare,Crt,Get,Put,WhatKey,Nombre;
{-------------------------------------------------------------------}
Procedure WriteMaxStress(   row,iMat: Byte;
                        var PlaceX,PlaceY: VectByte_25;
                        var Mat: MatrixR_Nmatx6;
                        var MaxStress: MatrixR_Nmatx9; (var for speed)
                        var MatLabel: VectS80_Nmat); {var for speed}
{
   Input:
     iMat:  Database Material number
     Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
     MaxStress[iMaterial,j]: iMaterial: Database material number
                             j  =1: X
                                =2: X'
                                =3: Y
                                =4: Y'
                                =5: Z
                                =6: Z'
                                =7: Sxy max
                                =8: Sxz max
                                =9: Syz max
     MatLabel[iMaterial]: Name of material, details
}
var
   i: integer;
begin
   gotoXY(PlaceX[row],PlaceY[row]);
   Case row of
     1: write(MatLabel[iMat]);
     2..7:  write(RField(Mat[iMat,row-1],12));
     8..16: write(RField(MaxStress[iMat,row-7],12));
     end;
end;
{-------------------------------------------------------------------}
Procedure EditOneMaterial(   iMat: Byte;
                         var Mat: MatrixR_Nmatx6;
                         var MatLabel: VectS80_Nmat;
                         var MaxStress: MatrixR_Nmatx9);
{
   Input:
     iMat:  material number
     Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
     MatLabel[iMaterial]: Name of material, details
```

```
        MaxStress[iMaterial,j]: iMaterial: Database material number
                                j =1: X
                                  =2: X'
                                  =3: Y
                                  =4: Y'
                                  =5: Z
                                  =6: Z'
                                  =7: Sxy max
                                  =8: Sxz max
                                  =9: Syz max
    Output:
        Same as Input
}
var
    RowMax,Row,RowOld: integer;
    PlaceX,PlaceY: VectByte_25;
    IRtype: Byte;
    NumberI: Itype;
    NumberR: Rtype;
    Mot: String255;
    C: Char;
    Action: String6;
begin
    row:=0;
    ClrScr;
    TextColor(LightGreen);
    writeln('<Up>,<Down>,<Home>,<End>,<F2> for isotropic (give E1,Nu12),<Esc>');
    writeln('Material ----------');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    writeln('Modulus ----------');
    write('          E1: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write('          E2: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write('         G12: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write('         G23: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    writeln('Poisson ratio ----------');
    write('         Nu12: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write('         Nu23: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    writeln('Maximum stress ----------');
    write('     Tensile 1: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write(' Compressive 1: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write('     Tensile 2: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
    write(' Compressive 2: ');
    Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
```

```
  write('    Tensile 3: ');
  Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
  write(' Compressive 3: ');
  Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
  write('     Shear 12: ');
  Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
  write('     Shear 13: ');
  Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
  write('     Shear 23: ');
  Inc(row); PlaceX[row]:=WhereX; PlaceY[row]:=WhereY;  writeln;
  writeln('1: Fiber direction.');
  writeln('2: normal to 1, in Xi-Eta plane of local coordinates');
  writeln('3: Normal to 1 and 2, Zeta direction of local coordinates');
  RowMax:=row;
{
                Write all info for the iMat Material
}
  TextColor(LightRed);
  For row:=1 to RowMax do
    WriteMaxStress(row,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
  TextColor(LightGreen);
  RowOld:=0;
  Row:=1;
  Repeat
{
                Write old row
}
    If (0<RowOld) and (RowOld<>Row) then
      begin
      TextColor(LightRed);
      WriteMaxStress(RowOld,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
      TextColor(LightGreen);
      end;
    RowOld:=Row;
{
                Write current row
}
    TextColor(Black);
    TextBackground(LightRed);
    WriteMaxStress(Row,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
    TextColor(LightGreen);
    TextBackground(Black);
    Get_Key(C,Action);
    If (Action='UP') and (1<row) then
      Row:=Row-1;
    If (Action='DOWN') and (row<RowMax) then
      Row:=Row+1;
    If (Action='HOME') and (1<row) then
      Row:=1;
    If (Action='END') and (row<RowMax) then
      Row:=RowMax;
{
```

```
                Set Material as isotropic using only:  E1,nu12
                E2:=E1
                G12:=E/2/(1+Nu)
                G23:=G12
                Nu23:=Nu12
}
     If (Action='F2') then
        begin
        Mat[iMat,2]:=Mat[iMat,1];                    {row 3}
        Mat[iMat,3]:=Mat[iMat,1]/2/(1+Mat[iMat,5]); {row 4}
        Mat[iMat,4]:=Mat[iMat,3];                    {row 5}
        Mat[iMat,6]:=Mat[iMat,5];                    {row 7}
        TextColor(LightRed);
        WriteMaxStress(3,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
        WriteMaxStress(4,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
        WriteMaxStress(5,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
        WriteMaxStress(7,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
        TextColor(LightGreen);
        end;
{
                Capture Number or word
}
     If (C<>chr(0)) then
        begin
{
                Erase old information
}
        TextColor(Black);
        WriteMaxStress(Row,iMat,PlaceX,PlaceY,Mat,MaxStress,MatLabel);
        TextColor(LightGreen);
        TextColor(Yellow);
        TextBackground(Black);
        Mot:=C;
        GotoXY(PlaceX[row],PlaceY[row]);
        write(Mot,' ');
        GotoXY(PlaceX[row]+Length(Mot),PlaceY[row]);
        While (C<>chr(13)) and (Length(Mot)<=79-PlaceX[row]) do
          begin
          Repeat until keypressed;
          C:=ReadKey;
          If (C in [' '..'~']) and (not (C in [','])) then  {no commas}
            Mot:=Mot+C;
          If C=chr(8) then   {back space}
            Delete(Mot,Length(Mot),1);
          GotoXY(PlaceX[row],PlaceY[row]);
          write(Mot,' ');
          GotoXY(PlaceX[row]+Length(Mot),PlaceY[row]);
          end;
        gotoxy(PlaceX[row],PlaceY[row]);
        ClrEOL;
{
                Verify Number
```

```
}
        StoX(Mot,NumberI,NumberR,IRtype);
{
                Store Number
}
        If row=1 then
           MatLabel[iMat]:=Mot;
        If IRtype=1 then              { IType }
           Case row of
              2..7: MaT[iMat,row-1]:=NumberI;
              8..16: MaxStress[iMat,row-7]:=NumberI;
              end;
        If IRtype=2 then              { Rtype }
           Case row of
              2..7: MaT[iMat,row-1]:=NumberR;
              8..16: MaxStress[iMat,row-7]:=NumberR;
              end;
        RowOld:=0;  (to write new value)
        end;
     until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure ChooseMaterial(var Mat: MatrixR_Nmatx6;
                         var MatLabel: VectS80_Nmat;
                         var MaxStress: MatrixR_Nmatx9);
{
   Input:
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      MatLabel[iMaterial]: Name of material, details
      MaxStress[iMaterial,j]: iMaterial: Database material number
                              j =1: X
                                =2: X'
                                =3: Y
                                =4: Y'
                                =5: Z
                                =6: Z'
                                =7: Sxy max
                                =8: Sxz max
                                =9: Syz max
   Output:
      Same as Input
}
Var
   PlaceX,PlaceY,iMat: Byte;
   C: char;
   Action: String6;
Begin
   Repeat
      ClrScr;
      For iMat:=1 to Nmat_max do
         begin
         If iMat<=25 then
```

```
            begin
            PlaceX:=1;
            PlaceY:=iMat;
            C:=chr(96+iMat); {a..y}
            end
         else
            begin
            PlaceX:=41;
            PlaceY:=iMat-25;
            C:=chr(64+iMat-25); {A..Y}
            end;
         GotoXY(PlaceX,PlaceY);
         TextColor(LightRed);
         write(C);
         TextColor(LightGreen);
         write(':',Copy(MatLabel[iMat],1,34));
         If 34<Length(MatLabel[iMat]) then write('...');
         end;
      Get_Key(C,Action);
      Case C of
         'a'..'y': begin
                   iMat:=Ord(C)-96;
                   EditOneMaterial(iMat,Mat,MatLabel,MaxStress);
                   end;
         'A'..'Y': begin
                   iMat:=Ord(C)-64+25;
                   EditOneMaterial(iMat,Mat,MatLabel,MaxStress);
                   end;
         end;
      until Action='ESC'
end;
{-------------------------------------------------------------------------}
Var
   Mat: MatrixR_Nmatx6;
   MaxStress: MatrixR_Nmatx9;
   MatLabel: VectS80_Nmat;
begin
   ClrScr;
   TextColor(LightGreen);
   Get_Mat(Mat);
   Get_MaxStress(MaxStress);
   Get_MatLabel(MatLabel);
   ChooseMaterial(Mat,MatLabel,MaxStress);
   Put_Mat(Mat);
   Put_MaxStress(MaxStress);
   Put_MatLabel(MatLabel);
   ClrScr;
end.
```

MATH.PAS

```
{$N+}
Unit Math;
Interface
   Uses Declare;
Function ArcCos(    X: Rtype): Rtype;
Function ArcTan4(    x,y: Rtype):Rtype;
Function PowerR(    base,puis:Rtype) :Rtype;
{-------------------------------------------------------------------}
Implementation
Function ArcCos;
{
Function ArcCos(    X: Rtype): Rtype;
}
{
   Neuton-Raphson algorithm used to get ArcCos(X)
   Burden & Faires, Numerical Analysis, 3rd edition, Page 42
}
var
   Root_ok: Boolean;
   TOL: Rtype;
   Nmax,i: integer;
   p,p0: Rtype;
begin
   TOL:=1E-10;
   Nmax:=1000;
   p0:=0.5;
   i:=1;
   Root_ok:=False;
   While i<=Nmax do
      begin
      p := p0 - (Cos(p0)-X)/(-Sin(p0));  {x=x0-f/f'}
      If (abs(p-p0)<TOL) then
         begin
         i:=Nmax;
         Root_ok:=True;
         end;
      i:=i+1;
      p0:=p;
      end;
   If Root_ok=False then
      begin
      writeln('Unable to get ArcCos(',X,')');
      Halt;
      end;
   If p0<0 then ArcCos:=-p0    { keep result in 0<=angle<=Pi }
          else ArcCos:= p0;   { knowing that cos(angle)=cos(-angle) }
end;
{-------------------------------------------------------------------}
Function ArcTan4;
{
```

```
Function ArcTan4(     x,y: Rtype):Rtype;
}
{
   Input:
      x,y: 2-D coordinates
   Output:
      4 quadrant ArcTangent angle in radians
}
var
   Angle: Rtype;
begin
   if x=0 then
      begin
      If 0<=y then
         Angle:=Pi/2
      else
         Angle:=-Pi/2;
      end
   else
      Angle:=ArcTan(y/x);
   If x<0 then Angle:=Angle+Pi;
   ArcTan4:=Angle;
end;
{-------------------------------------------------------------------}
function PowerR;
{
function PowerR(     base,puis:Rtype) :Rtype;
}
{
            Returns   (base)^(puis)
                                  Written by Jerome Daoust in Fall '87
}
var
   Crit: Rtype;
begin
   Crit:=1E-5;
   If 0<base then
      PowerR := exp(puis*ln(base))          {base>0}
   else
      begin
      If base=0 then
         PowerR := 0.0                      {base=0}
      else
         begin
         if Abs(Puis-round(Puis))<Crit then
            if Abs(Puis/2-Round(Puis/2))<Crit then
               PowerR := exp(puis*ln(-base))    {base<0 Puis=Pair}
            else
               PowerR := -exp(puis*ln(-base))   {base<0 Puis=Impair}
         else
            if Abs(1/Puis-round(1/Puis))<Crit then
               if Abs(1/Puis/2-Round(1/Puis/2))<Crit then
```

```
                PowerR := exp(puis*ln(-base))     {base<0 1/Puis=Pair}
              else
                PowerR := -exp(puis*ln(-base))    {base<0 1/Puis=Impair}
            else
              Writeln('(ERROR: Base=',base:8,' Power=',Puis:8,')');
          end;
        end;
end;
{-------------------------------------------------------------------------}
end.
```

NEWNAME.PAS

```
{-------------------------------------------------------------------------
   Program to change case name
   Initially written in May '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 2000,0,0}
Uses
   Crt,Declare,Status,Where,WaitKey;
var
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
begin
{
      NEWNAME NewCaseName
         NewCaseName: New name of case without ".GEN" extension.
            The current case must be preprocessed.
}
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   If (StatusPrep='Done') then
      begin
      TextColor(LightRed);
      write(CaseName);
      TextColor(LightGreen);
      CaseName:=ParamStr(1);
      write(' --> ');
      TextColor(LightRed);
      writeln(CaseName);
      TextColor(LightGreen);
      Write_Status(CaseName,CoarseCaseName,
                   StatusPrep,StatusForce,StatusDispl,StatusWorst,
                   StatusFiberStrain,StatusFiberStress,
                   StatusElemStrain,StatusElemStress,
                   StatusGlobalStrain,StatusGlobalStress);
      end
   else
      begin
      writeln('You must preprocess before!');
      wait;
      end;
end.
```

NNE_DIM.PAS

```
{$N+}
Unit Nne_Dim;
Interface
Uses
   Declare;
Function Get_Nne(     Etype: IType):IType;
Function Get_D(     Etype: IType):IType;
Function N_Stress(     Dim: IType): IType;
{-------------------------------------------------------------------}
Implementation
Function Get_Nne;
{
Function Get_Nne(     Etype: IType):IType;
}
{
   Input:
     Etype: Type of element (1,2...)
   Output:
     Nne: Number of nodes used by the element type
}
begin
   Get_Nne:=0;      { default }
   case Etype of
     1: Get_Nne:=3;
     2: Get_Nne:=20;
     end;
end;
{-------------------------------------------------------------------}
Function Get_D;
{
Function Get_D(     Etype: IType):IType;
}
{
   Input:
     Etype: Type of element (1,2...)
   Output:
     Get_D: 2 for 2-D element type
            3 for 2-D element type
}
begin
   Get_D:=0;      { default }
   case Etype of
     1: Get_D:=2;
     2: Get_D:=3;
     end;
end;
{-------------------------------------------------------------------}
Function N_Stress;
{
Function N_Stress(     Dim: IType): IType;
```

```
)
(
    Input:
      Dim: Dimensions in geometry
    Output:
      N_Stress: Number of Stress Components
)
begin
    Case Dim of
      1: N_Stress:=1;
      2: N_Stress:=3;
      3: N_Stress:=6;
      end;
end;
(-------------------------------------------------------------------)
end.
```

NOMBRE.PAS

```
{$N+}
Unit Nombre;
Interface
Uses
   Declare;
procedure Separate(    Number: Rtype;
                   var Prefix: Rtype;
                   var Power: Integer);
Function RField(    NumberR: Rtype;
                    Field: Integer): String255;
Function RWord(    NumberR: Rtype;
                   Field: Integer): String255;
Procedure StoX(    S: String;     (do not use Var (modified))
               var Ni: Itype;
               var Nr: Rtype;
               var IRtype: Byte);
{-------------------------------------------------------------}
Implementation
procedure Separate;
{
procedure Separate(    Number: Rtype;
                   var Prefix: Rtype;
                   var Power: Integer);
}
{
          Separate Prefix from Power in a real number
}
var
   Ln10,P: Rtype;
begin
   Ln10:=ln(10);
   If number=0.0 then
     begin
     Power:=0;
     Prefix:=0.0;
     end
   else
     begin
     P:=ln(abs(number))/ln10;
     If abs(Round(P)-P)<1E-8 then
        Power:=Round(P)
     else
        Power:=Trunc(P);
     If P<0 then Dec(Power);
     Prefix:=number/exp(Power*ln10);
     end;
end;
{-------------------------------------------------------------}
Function RField;
{
```

```
Function RField(    NumberR: Rtype;
                    Field: Integer): String255;
)
{
   Input:
     NumberR: Real number
     Field: Length of field for display
   Output:
     Real number is put into a word of length FIELD
}
var
   Prefix: Rtype;
   PowerOfPrefix,After,PowerMin,PowerMax,Power: Integer;
   Mot,PreW,PowW: String255;
begin
{
                  Find maximum and minimum power to avoid exponent
}
   If 0<=NumberR then PowerMax:=Field-1
                 else PowerMax:=Field-2;
   If 6<PowerMax then PowerMax:=6;
   If 0<=NumberR then PowerMin:=-Trunc(Field/3)
                 else PowerMin:=-Trunc(Field/3)+1;
   If PowerMin<-3 then PowerMin:=-3;
{
                  Separate number:  -3.45E-0056 ->  -3.45  -56
}
   Separate(NumberR,Prefix,Power);
   If (PowerMin<=Power) and (Power<=PowerMax) then
     begin
     If 0<=Power then
        After:=Field-Power-2  (for a positive Power)
     else
        After:=Field-2;       (for a negative Power)
     If NumberR<0 then
        Dec(After);
     str(NumberR:Field:After,Mot)
     end
   else
     begin
{
                  Put in Engineering exponents: 3 6 9 -3 -6 -9 if possible
}
     PowerOfPrefix:=0;
     Case Power of
        -17,-14,-11,-8,-5,-2,1,4,7,10,13,16: begin
           Prefix:=Prefix*10.0;
           Power:=Power-1;
           PowerOfPrefix:=1;
           end;
        -16,-13,-10,-7,-4,-1,2,5,8,11,14,17: begin
           Prefix:=Prefix*100.0;
```

```
            Power:=Power-2;
            PowerOfPrefix:=2;
            end;
         end;
      str(Power,PowW);
      If Power>=0 then
         PowW:='+'+PowW;
      PowW:='E'+PowW;
{
                  erase 0 just after 'E+' or 'E-'
                  E+04 ->  E+4
}
      While PowW[Pos('E',PowW)+2]='0' do
         Delete(PowW,Pos('E',PowW)+2,1);
{
                  Erase '+' just after 'E'
}
      If PowW[Pos('E',PowW)+1]='+' then
         Delete(PowW,Pos('E',PowW)+1,1);
{
                  Get Prefix and back fill with 0's to match Field
                  524.3:4:0  -> ' 524'
}
      If Prefix>=0 then
         str(Prefix:Field-Length(PowW)
                  :Field-(Length(PowW)+2+PowerOfPrefix),PreW)
      else
         str(Prefix:Field-Length(PowW)
                  :Field-(Length(PowW)+3+PowerOfPrefix),PreW);
{
                  Truncate 100.0000E+12 to
                           100.000E+12      when number is 99.999999E+12
}
      While Field<Length(PreW)+Length(PowW) do
         Delete(PreW,Length(PreW),1);
      Mot:=PreW+PowW;
      end;
   RField:=Mot;
end;
{-------------------------------------------------------------------}
Function RWord;
{
Function RWord(    NumberR: Rtype;
                   Field: Integer): String255;
}
{
   Input:
      NumberR: Real number
      Field: Maximum Length of field for display
   Output:
      Real number is put into a word of length FIELD
         unneccessary 0's are taken out
```

```
)
var
   Mot: String255;
begin
   Mot:=RField(NumberR,Field);
{
                  Delete Blanks
                  524.3:4:0 -> ' 524' -> '524'
}
   While Mot[1]=' ' do
      Delete(Mot,1,1);
{
           erase '0' or '.' just before 'E' only if '.' is present
           3.45000E+04 ->  3.45E+04
           500E6 -> 500E6
}
   If 0<Pos('E',mot) then
      While (0<Pos('.',Mot)) and (Mot[Pos('E',mot)-1] in ['0','.']) do
         Delete(Mot,Pos('E',mot)-1,1);
{
           erase 0 at the end if no 'E' is found and a '.' is found
           3.45000E+04 ->  3.45E+04
}
   If Pos('E',Mot)<=0 then
      If 0<Pos('.',Mot) then
         begin
         While Mot[Length(Mot)] in ['0'] do
            Delete(Mot,Length(Mot),1);
         If Mot[Length(Mot)]='.' then
            Delete(Mot,Length(Mot),1);
         end;
   RWord:=Mot;
end;
{-------------------------------------------------------------------------}
Procedure StoX;
{
Procedure StoX(    S: String;   do not use Var (modified)
              var Ni: Itype;
              var Nr: Rtype;
              var IRtype: Byte);
}
{
   Input:
      S: string
   Output:
      Ni: integer number
      Nr: real number
      IRtype: 0 if no conversion can be done (string only)
              1 if integer conversion done
              2 if real conversion done
}
Var
```

```
   Result: integer;
begin
   IRtype:=0;
   If S<>'' then
      begin
{
                so +X is considered as number X
                If a number is given as   .3  -> 0.3
                                         -.3 -> -0.3
                                          3.  -> 3.0
}
      If (copy(S,1,2)='-.') then
         Insert('0',S,2)
      else
         begin
         If S[1]='+' then Delete(S,1,1);
         If (S[1]='.') then S:='0'+S;
         end;
      If (S[Length(S)]='.') then S:=S+'0';
{
                Store IType or Real number in appropriate place
                if possible
}
      Val(S,Ni,Result);
      If Result=0 then
         IRtype:=1
      else
         begin
         val(S,Nr,Result);
         If Result=0 then IRtype:=2;
         end;
      end;
end;
{-------------------------------------------------------------}
end.
```

NUMBERS.PAS

```
{-------------------------------------------------------------------------
   Program to see geometry data
   Initially written in February '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 50000,0,655360}   {same as in SOLVE.pas}
Uses
   Crt,Dos,Graph,Declare,WaitKey,Where,Timer,Nombre,Get,Linear,
   Nne_Dim,File_RW,Status;
{-------------------------------------------------------------------------}
Procedure DisplayWait;
var
   Xs,Ys: Byte;
begin
   Xs:=WhereX;
   Ys:=WhereY;
   gotoXY(76,1);
   TextColor(Black+Blink);
   TextBackground(LightRed);
   write('WAIT');
   gotoXY(Xs,Ys);
   TextColor(LightGreen);
   TextBackground(Black);
end;
{-------------------------------------------------------------------------}
Procedure EraseWait;
var
   Xs,Ys: Byte;
begin
   Xs:=WhereX;
   Ys:=WhereY;
   gotoXY(76,1);
   write('    ');
   gotoXY(Xs,Ys);
end;
{-------------------------------------------------------------------------}
Type
   BlockType = array[1..10,1..10] of Byte;
var
   FreeSize,SizeOfFile,FileReadQtt,FileWriteQtt,FileReadRandomQtt,FilePointer,
      i,j,Net,Nnt,NntOld,Dim,NneMax,ID,DOF,iNode: Itype;
   A,B,C,D,E,R: Rtype;
   Ns: String[255];
   Xs,Ys: Byte;
   CurrentDir,WriteDirectory,CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   GrDriver, GrMode: Integer;
```

```
   Fblock: BlockType;
   F: File of BlockType;
   F_xyz: FileR;
   F_Fd: FileByte;
   Max_xyz,Min_xyz,xyz: VectR_D;
   Fd: VectByte_D;
   DiskPerformanceAnswer: Char;
begin
   ClrScr;
   TextColor(LightGreen);
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   If T_Dir<>'' then
      WriteDirectory:=T_Dir
   else
      WriteDirectory:='DEFAULT';
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
{
                  Get geometry data from disk
}
   Get_I('Net.P',Net);
   Get_I('Nnt.P',Nnt);
   Get_I('NntOld.P',NntOld);
   Get_I('Dim.P',Dim);
   Get_I('NneMax.P',NneMax);
{
                  Display geometry data
}
   write('          Some numbers on the geometry of ');
   TextColor(LightRed);
   writeln(CaseName);
   TextColor(LightGreen);
   writeln;
   Writeln(Net,' elements used');
   Writeln(Nnt,' nodes used');
   Writeln(NntOld,' is the highest node number given by the user');
   Writeln(Dim,' dimensional problem');
   Writeln(NneMax,' is the highest number of nodes found in the elements used');
   DisplayWait;
{
                  Get exact number of degrees of freedom
}
   Xs:=WhereX;
   Ys:=WhereY;
   Assign(F_Fd,C_Dir+'Fd.p');
   Reset(F_Fd);
   DOF:=0;
   For iNode:=1 to Nnt do
      begin
```

```
      GotoXY(Xs,Ys);
      write(Nnt-iNode+1,' ');
      D_Byte_read(F_Fd,iNode,Dim,Fd);
      For iD:=1 to Dim do
         If Fd[iD]=0 then Inc(DOF);
      end;
   Close(F_Fd);
   GotoXY(Xs,Ys);
   Writeln(DOF,' degrees of freedom (unknown displacements)');
{
                  Get coordinate's limits
}
   Xs:=WhereX;
   Ys:=WhereY;
   For iD:=1 to Dim do
      begin
      Max_xyz[iD]:=-BigR;
      Min_xyz[iD]:=+BigR;
      end;
   Assign(F_xyz,C_Dir+'XYZ.p');
   Reset(F_xyz);
   For iNode:=1 to Nnt do
      begin
      GotoXY(Xs,Ys);
      write(Nnt-iNode+1,' ');
      D_R_read(F_xyz,iNode,Dim,xyz);
      For iD:=1 to Dim do
         begin
         If xyz[iD]<Min_xyz[iD] then Min_xyz[iD]:=xyz[iD];
         If Max_xyz[iD]<xyz[iD] then Max_xyz[iD]:=xyz[iD];
         end;
      end;
   Close(F_xyz);
   GotoXY(Xs,Ys);
   writeln('Coordinates: ',Rfield(Min_xyz[1],12),' < X < ',
          Rfield(Max_xyz[1],12));
   If 2<=Dim then
      writeln('             ',Rfield(Min_xyz[2],12),' < Y < ',
             Rfield(Max_xyz[2],12));
   If 3<=Dim then
      writeln('             ',Rfield(Min_xyz[3],12),' < Z < ',
             Rfield(Max_xyz[3],12));
{
                  Show memory of the machine
}
   writeln;
   writeln('          Some numbers on the computer...');
   writeln;
   Str(MemAvail/1024:10:0,Ns);
   While Ns[1]=' ' do Delete(Ns,1,1);
   Writeln(Ns,' kB of RAM available');
{
```

```
                Detect graphic card
}
   GrDriver:=Detect;
   DetectGraph(GrDriver,GrMode);
   Case GrDriver of
      -2: Ns:='None';
      1: Ns:='CGA';
      2: Ns:='MCGA';
      3: Ns:='EGA';
      4: Ns:='EGA64';
      5: Ns:='EGAMono';
      6: Ns:='RESERVED';
      7: Ns:='HercMono';
      8: Ns:='ATT400';
      9: Ns:='VGA';
      10: Ns:='PC3270';
      end;
   writeln(Ns,' graphic card installed');
{
                Show speed of the machine
                Execute calculation loop
                   minimize loop index time with a lot of calculations
                   inside loop
}
   A:=Pi;
   B:=A*A;
   C:=Pi/2;
   D:=Pi/3;
   E:=Pi/7;
   ResetTimer(1);
   StartTimer(1);
   For i:=1 to 300 do
      begin
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      R:=A+B-C*D/E;
      end;
   StopTimer(1);
   Str(300*10*4/ReadTimer(1):20:0,Ns);
   While Ns[1]=' ' do Delete(Ns,1,1);
   writeln(Ns,' Flops (FLotting-point OPerations per Second)');
   EraseWait;
{
                Ask if disk performance is desired
}
```

```
   write('Do you want a disk performance test (Y/N)? ');
   While KeyPressed=True do DiskPerformanceAnswer:=ReadKey; {empty buffer}
   Repeat
      DiskPerformanceAnswer:=UpCase(ReadKey);
      Until DiskPerformanceAnswer in ['Y','N'];
   GotoXY(1,WhereY);
   write('                                          ');
   GotoXY(1,WhereY);
{
                Display disk writing speed (Temporary directory)
}
   If DiskPerformanceAnswer='Y' then
      begin
      DisplayWait;
      FileWriteQtt:=4000;
      FileReadQtt:=1000;
      FileReadRandomQtt:=100;
      SizeOfFile:=FileWriteQtt*SizeOf(Fblock);
      GetDir(0,CurrentDir);
      ChDir(T_Dir);
      FreeSize:=DiskFree(0);
      ChDir(CurrentDir);
      If FreeSize<SizeOfFile then
         begin
         writeln('Insufficient free disk space for disk test!');
         wait;
         Halt;
         end;
      Assign(F,T_Dir+'Rien.t');
      Rewrite(F);
      ResetTimer(1);
      StartTimer(1);
      For i:=1 to FileWriteQtt do
         write(F,Fblock);
      StopTimer(1);
      gotoXY(1,WhereY);
      Str(FileWriteQtt*SizeOf(Fblock)/1024/ReadTimer(1):20:2,Ns);
      While Ns[1]=' ' do Delete(Ns,1,1);
      writeln(Ns,' kB/s writing (',SizeOf(Fblock),
              ' byte data) to ',WriteDirectory,' directory');
{
                Display disk reading speed (Temporary directory)
}
      Seek(F,0);
      ResetTimer(1);
      StartTimer(1);
      For i:=1 to FileReadQtt do
         read(F,Fblock);
      StopTimer(1);
      Str(FileReadQtt*SizeOf(Fblock)/1024/ReadTimer(1):20:2,Ns);
      While Ns[1]=' ' do Delete(Ns,1,1);
      writeln(Ns,' kB/s reading (',SizeOf(Fblock),
```

```
            ' byte data) to ',WriteDirectory,' directory');
{
                Display disk acces time (Temporary directory)
}
     Randomize;
     ResetTimer(1);
     StartTimer(1);
     For i:=0 to FileReadRandomQtt do
        begin
        FilePointer:=Random(FileWriteQtt);
        Seek(F,FilePointer);
        read(F,Fblock);
        end;
     StopTimer(1);
     Str(ReadTimer(1)/FileReadRandomQtt*1024:20:3,Ns);
     While Ns[1]=' ' do Delete(Ns,1,1);
     writeln(Ns,' ms average access time (random read a ',SizeOf(Fblock),
            ' byte data)');
     While Ns[1]=' ' do Delete(Ns,1,1);
     writeln('   in a file size of ',SizeOfFile,' bytes in ',
            WriteDirectory,' directory');
{
                Wait for leaving
}
     Close(F);
     Erase(F);
     EraseWait;
     end;
  Wait;
  ClrScr;
end.
```

PARAM.PAS

```
Unit Param;
Interface
Uses
   Declare,WaitKey;
Procedure Parameters;
{-------------------------------------------------------------------------------}
Implementation
Procedure Parameters;
Var
   i: Byte;
   Mot: String[255];
begin
{
                  Check number of parameters
}
   If Param_max<ParamCount then
      begin
      Writeln('Too many parameters: ',ParamCount,' (Max=',Param_max,')');
      Wait;
      Halt;
      end;
{
                  Treat parameters passed with program name
                  Check parameters for a maximum length
                  Set parameters to upper case
}
   For iParam:=1 to ParamCount do
      begin
      Mot:=ParamStr(iParam);
      If Param_Long<Length(Mot) then
         begin
         Writeln('Parameter too long (',Param_Long,' characters max):');
         writeln(Mot);
         Wait;
         Halt;
         end;
      For i:=1 to Length(Mot) do
         Mot[i]:=UpCase(Mot[i]);
      ParameterString[iParam]:=Mot;
      end;
{
                  Set NoWait=True if 'NoWait' parameter is passed
}
   NoWait:=False;
   For iParam:=1 to ParamCount do
      If ParameterString[iParam]='NOWAIT' then NoWait:=True;
{
                  Set OnPrinter=True if 'Printer' parameter is passed
}
   OnPrinter:=False;
```

```
   For iParam:=1 to ParamCount do
      If ParameterString[iParam]='PRINTER' then OnPrinter:=True;
end;
{-------------------------------------------------------------------------}
end.
```

PICKFILE.PAS

```
{$N+}
Unit PickFile;
Interface
Uses
   Crt,Dos,Declare,waitkey,WhatKey;
Procedure Pick_File(    PathName,DirChoice: String255;
                        Attribute: Byte;
                        NotDots: Boolean;
                    var FileName: String255;
                    var File_ok: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Pick_File;
{
Procedure Pick_File(    PathName,DirChoice: String255;
                        Attribute: Byte;
                        NotDots: Boolean;
                    var FileName: String255;
                    var File_ok: Boolean);
}
{
   Consumes about 3600 bytes
   Input:
      PathName: Path of directory
      DirChoice: Directory selection. Example: '*.PAS'
      Attribute: File attribute:
         ReadOnly
         Hidden
         SysFile
         VolumeID
         Directory
         Archive
         AnyFile
      NotDots: True if the file name must have a character<>'.'
               False otherwise
   Output:
      FileName: Chosen File name with extension and without path
      File_ok: True if the file exists
               False otherwise
}
var
   Slong,iNew,iOld,ColMax,Count,i,j,Xstart,Ystart: Integer;
   DirInfo: SearchRec;
   Place: array[1..250,1..2] of Byte;
   Fname: array[1..250] of String[12];
   Action: String6;
   C: Char;
   Valid: Boolean;
begin
   File_ok:=False;
```

```
   FileName:='';
{
                terminate path by '\' when not a nul string
}
   If 0<Length(PathName) then
      If PathName[length(PathName)]<>'\' then PathName:=PathName+'\';
   ClrScr;
   TextColor(LightGreen);
   write('Directory: ');
   TextColor(LightRed);
   writeln(PathName,DirChoice);
   TextColor(LightGreen);
   writeln('use: ',chr(18),chr(29),' <Home> <End> <First letter> <PgUp>',
          ' <PgDn> <Esc>, <Enter> to select.');
   Xstart:=WhereX;
   Ystart:=WhereY;
{
                Read Directory
}
   Slong:=12; {full name with extension}
   ColMax:=Trunc((80-1)/(Slong+1));
{
                Find first file matching description
}
   FindFirst(PathName+DirChoice,Attribute,DirInfo);
   Count:=0;
{
                Find other files under same description
}
   While DosError=0 do
      begin
      FileName:=DirInfo.name;
{
                Add Backslash after directory name
}
      If DirInfo.Attr = Directory then FileName:=FileName+'\';
{
                Verify if file name has a character<>'.'
}
      Valid:=True;
      If NotDots=True then
         begin
         Valid:=False;
         i:=1;
         While i<=length(FileName) do
            begin
            If not (FileName[i] in ['.','\']) then
               begin
               Valid:=True;
               i:=Length(FileName);  {exit loop}
               end;
            i:=i+1;
```

```
            end;
         end;
      If Valid=True then
         begin
         Count:=Count+1;
         gotoXY(Xstart,Ystart);
         write(count);
         Place[count,1]:=Xstart+
                        (Slong+1)*((Count-1)-ColMax*Trunc((Count-1)/ColMax));
         Place[count,2]:=Ystart+Trunc((Count-1)/ColMax);
{
                  Put File name in Vector of names
}
         Fname[Count]:=FileName;
         end;
      FindNext(DirInfo);
      end;
{
                  Sort Directory by Bubble Sort Technique
                  For the quantity of items, this method is fast enough
}
   If 0=Count then
      begin
      writeln;
      writeln('No files found!');
      writeln('Try giving a new directory in the configuration.');
      wait;
      end
   else
      begin
{
                  Sort files by bubble sort
}
      For i:=1 to Count do
         begin
         gotoXY(Xstart,Ystart);
         write(count-i+1,' ');
         For j:=Count-1 downto 1 do
            If Fname[j]>Fname[j+1] then
               begin
               FileName:=Fname[j];
               Fname[j]:=Fname[j+1];
               Fname[j+1]:=FileName;
               end;
         end;
{
                  Fill name with leading blanks until length=Slong
}
      For i:=1 to Count do
         While Length(Fname[i])<Slong do
            Fname[i]:=' '+Fname[i];
{
```

```
                Display Directory
}
     For i:=1 to Count do
        begin
        GotoXY(Place[i,1],Place[i,2]);
        write(Fname[i]);
        end;
{
                Display First File name
}
     TextColor(Black);
     TextBackground(LightGreen);
     GotoXY(Place[1,1],Place[1,2]);
     write(Fname[1]);
     TextColor(LightGreen);
     TextBackground(Black);
{
}
     iNew:=1;
     Action:='NIL';
     While (Action<>'ENTER') and (Action<>'ESC') do
        begin
        iOld:=iNew;
        Get_Key(C,Action);
{
                Move according to:  Home End arrows
}
        If Action='HOME' then
           iNew:=1;
        If Action='END' then
           iNew:=Count;
        If (Action='RIGHT') and (iOld<Count) then
           iNew:=iOld+1;
        If (Action='LEFT') and (1<iOld) then
           iNew:=iOld-1;
        If (Action='UP') and (1+ColMax<=iOld) then
           iNew:=iOld-ColMax;
        If (Action='DOWN') and (iOld<=Count-ColMax) then
           iNew:=iOld+ColMax;
        If (Action='PGDN') then
           iNew:=iOld+ColMax*Trunc((Count-iOld)/ColMax);
        If (Action='PGUP') then
           iNew:=iOld-ColMax*Trunc((iOld-1)/ColMax);
{
                Move according to:  First letter of File name
}
        If C<>chr(0) then
           begin
           C:=UpCase(C);
           If C in ['A'..'Z','0'..'9'] then
              begin
{
```

```
                Look for File Name after current position
}
            iNew:=0;
            i:=iOld+1;
            While i<=Count do
               begin
               j:=1;          { Find first character <> blank in name }
               While (Fname[i][j]=' ')and(j<=Slong) do j:=j+1;
               If C=Fname[i][j] then
                  begin
                  iNew:=i;
                  i:=Count; { exit loop }
                  end;
               i:=i+1;
               end;
{
                Look for File Name befor or at current position
}
            if iNew=0 then
               begin
               i:=1;
               While i<=iOld do
                  begin
                  j:=1;          { Find first character <> blank in name }
                  While (Fname[i][j]=' ')and(j<=Slong) do j:=j+1;
                  If C=Fname[i][j] then
                     begin
                     iNew:=i;
                     i:=Count; { exit loop }
                     end;
                  i:=i+1;
                  end;
               end;
            If iNew=0 then iNew:=iOld; (no change)
            end;
         end;
{
                Display new location
}
      If iOld<>iNew then
         begin
         GotoXY(Place[iOld,1],Place[iOld,2]);
         write(Fname[iOld]);
         TextColor(Black);
         TextBackground(LightGreen);
         GotoXY(Place[iNew,1],Place[iNew,2]);
         write(Fname[iNew]);
         TextColor(LightGreen);
         TextBackground(Black);
         end;
      end;
   If Action='ENTER' then File_ok:=True;
```

```
{
                Get File Name
}
     FileName:=Fname[iNew];
     While Copy(FileName,1,1)=' ' do   { erase leading spaces }
        Delete(FileName,1,1);
{
                Remove Backslash after directory name
}
     If FileName[Length(FileName)]='\' then
        Delete(FileName,Length(FileName),1);
     end;
  ClrScr;
  If Action='ESC' then FileName:='';
end;
{-------------------------------------------------------------------}
end.
```

PREP.PAS

```
{-------------------------------------------------------------------
   PreProcessor
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 40000,0,0}
Uses
   Dos,Crt,Declare,WaitKey,Timer,WhatKey,Nombre,PickFile,Where,
   Nne_Dim,Linear,File_RW,Get,Put,Sonore,Status,Translat,Param,Logo,
   K_Var,K_Elem,K_Node,K_Mat,K_ForDis,K_Trac,K_Local,K_Old,K_Inter,K_Oper;
{-------------------------------------------------------------------}
Procedure Write_Line(    KeyWord: String255;
                         IRS,Ni: VectI_Nc;
                         Nr: VectR_Nc;
                     var Ns80: VectS80_Nc;
                         Qtt: Itype);
{
   Input:
      KeyWord: Command word
           example: NGEN
      IRS[i]: 0 if the (i)th number in the command line is IType
             1 for a real
             2 for a string
            -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Ns80[i]: String at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
   Output:
      Displays interpreted command line according to a given indentation
              ----------
}
var
   i,j,Indent,L,count: IType;
   Full: array[1..1000] of char;
   Mot: String255;
begin
   Indent:=3;   { Indent }
{
                  Get Full character vector
}
   For i:=1 to Length(KeyWord) do
      Full[i]:=KeyWord[i];
   L:=0;
   If 0<length(KeyWord) then
      begin
      Full[Length(KeyWord)+1]:=' ';
      L:=Length(KeyWord)+1;
      end;
```

```
  For i:=1 to Qtt do
     begin
     Mot:='';
     If IRS[i]=0 then  { Integer }
        str(Ni[i],Mot);
     If IRS[i]=1 then  { Real }
        Mot:=RWord(Nr[i],15);
     If IRS[i]=2 then  { string }
        Mot:=Ns80[i];
     If i<Qtt then Mot:=Mot+',';
     While Mot[1]=' ' do Delete(Mot,1,1); { remove leading blanks }
     For j:=1 to Length(Mot) do
        Full[L+j]:=Mot[j];
     L:=L+Length(Mot);
     end;
  Full[L+1]:=';';
  L:=L+1;
{
                Write Vector
}
  TextColor(LightGreen);
  i:=1;
  While i<=L do
     begin
     If (WhereX<=Indent) and (Indent<i) and (i<L) then
        GotoXY(Indent+1,WhereY);
     If (Full[i] in [',',';']) then
        TextColor(LightRed);
     If i<=Length(KeyWord) then
        TextColor(LightCyan);
     Write(Full[i]);
     TextColor(LightGreen);
{
                Look if enough space is left on line to completly write
                next parameter
}
     If Full[i]=',' then
        begin
        Count:=1;
        While (not (Full[i+Count] in [',',';'])) and (i+Count<=L) do
            Inc(Count);
        If Col_Max<WhereX-1+Count then
           writeln;
        end;
     Inc(i);
     end;
  If i<WhereX then writeln;
end;
{-------------------------------------------------------------------}
Procedure ElementCrush(var F_Elem: FileI;
                       var F_Angle: FileR;
                           NneMax: IType;
```

```
                        var Net: IType);
{
   Input:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
      F_Angle: File With angle of elements (radians)
      NneMax: Maximum number of nodes in an element
      Net: Number of elements
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Compacted for elements used in structure
      F_Angle: File With angle of elements (radians)
              Compacted for elements used in structure
      Net: New number of elements (compacted index)
      Old element number is kept in "OldElem.P" file
}
var
   NetOriginal,iOld,iNew: IType;
   Angle: Rtype;
   Elem: VectI_Nne2;
   Xs,Ys: integer;
   F_OldElem: FileI;
begin
   Assign(F_OldElem,C_Dir+'OldElem.P');
   Rewrite(F_OldElem);
   write('Establishing compacted element index ');
   Xs:=WhereX;
   Ys:=WhereY;
   NetOriginal:=Net;
   iNew:=1;
   For iOld:=1 to NetOriginal do
      begin
      gotoXY(Xs,Ys);
      write(NetOriginal-iOld+1,' ');
      Nne2_I_Read(F_Elem,iOld,NneMax,Elem);
{
                  Check if element type is defined, 0=Etype if undefined
}
      If 0<Elem[2] then
         begin
         I_Write(F_OldElem,iNew,iOld);
         Nne2_I_Write(F_Elem,iNew,NneMax,Elem);
         R_Read(F_Angle,iOld,Angle);    { correct angles }
         R_Write(F_Angle,iNew,Angle);
         iNew:=iNew+1;
         end
      else
         Net:=Net-1;
      end;
   Close(F_OldElem);
```

```
   gotoXY(Xs,Ys);
   writeln('  ');
{
                  Truncate F_Elem File after Net node
                          F_Angle
}
   Truncate(F_Elem);
   Truncate(F_Angle);
end;
{-------------------------------------------------------------------}
Procedure NodeCrush(var F_Elem: FileI;
                    var Nnt: IType;
                        NneMax,Net: IType;
                    var F_XYZ,F_Force,F_Displ: FileR;
                    var F_Fd: FileByte;
                    var Dim: IType);
{
   Input:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Element numbering must be compacted already (ElementCrush)
      Nnt: Maximum node number
      NneMax: Maximum number of nodes in an element
      Ne: Number of elements;
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
          XYZ[Lin(i,1,Dim)-1] > BigR for undefined position
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      Nne: number of nodes in an element
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
            Nodes # readjusted for new numbers (Compacted)
      Nnt: Number of nodes used
      F_XYZ: repositioned coordinates according to new node numbers
      F_Force,F_Displ,F_Fd: repositioned forces and Displacement like F_xyz
      F_Used: File giving number of time a node is used
      F_NewOld: original node number given by user for each "compacted" node #
      Dim: Dimension of Problem (=2 for 2D, =3 for 3D)
           =0 if elements of different dimensions were used
      F_OldNew: compacted node number for node numbers given by user
                0 when a node is not used
}
var
   iOld,iNew,NntOriginal,Gap,jGood,From_node,To_node,iD,i,j,iE,Old,New: IType;
   Xs,Ys: Integer;
```

```
   Used,ZeroByte: Byte;
   XYZ,Force,Displ: VectR_D;
   Fd: VectByte_D;
   Elem: VectI_Nne2;
   F_NewOld,F_OldNew: FileI;
   F_Used: FileByte;
begin
   ZeroByte:=0;
   Assign(F_NewOld,C_Dir+'NewToOld.P');
   Assign(F_OldNew,T_Dir+'OldToNew.P');
   Assign(F_Used,C_Dir+'Used.P');
   Rewrite(F_NewOld);
   Rewrite(F_OldNew);
   Rewrite(F_Used);
{
                initialize F_Used F_NewOld
                Initialize Old -> New File (before Nnt reduced)
}
   write('Initialization of index for compacted node numbers ');
   Xs:=WhereX;
   Ys:=WhereY;
   For i:=1 to Nnt do
      begin
      gotoXY(Xs,Ys);
      write(Nnt-i+1,' ');
      Byte_Write(F_Used,i,ZeroByte);
      I_Write(F_NewOld,i,i);
      I_0(F_OldNew,i);   {0 will remain for unused node numbers}
      end;
   gotoXY(Xs,Ys);
   writeln('  ');
{
                Count number of times a node is used --> Used
}
   write('Counting the number of times a node is used ');
   Xs:=WhereX;
   Ys:=WhereY;
   For iE:=1 to Net do
      begin
      gotoXY(Xs,Ys);
      write(Net-iE+1,' ');
      Nne2_I_Read(F_Elem,iE,NneMax,Elem);
      For j:=1 to Get_Nne(Elem[2]) do
         begin
         Byte_Read(F_Used,Elem[2+j],Used);
         Inc(Used);
         Byte_Write(F_Used,Elem[2+j],Used);
         end;
      end;
   gotoXY(Xs,Ys);
   writeln('  ');
{
```

```
                correct Nnt, Used, establish New -> Old relation
                This is a bit slower than just taking the last item to
                   fill vacant node numbers, but the node order is kept.
}
   write('Establishing compacted node index ');
   Xs:=WhereX;
   Ys:=WhereY;
   NntOriginal:=Nnt;
   iNew:=1;
   For iOld:=1 to NntOriginal do
      begin
      gotoXY(Xs,Ys);
      write(NntOriginal-iOld+1,' ');
      Byte_Read(F_Used,iOld,Used);
      If 0<Used then
         begin
         I_Read(F_NewOld,iOld,Old);
         I_Write(F_NewOld,iNew,Old);
         Byte_Write(F_Used,iNew,Used);
         iNew:=iNew+1;
         end
      else
         Nnt:=Nnt-1;
      end;
   gotoXY(Xs,Ys);
   writeln('  ');
{
                correct XYZ, Force, Displ, Fd, establish Old_to_New
}
   write('Correcting coordinates, forces and displacements ');
   Xs:=WhereX;
   Ys:=WhereY;
   For i:=1 to Nnt do
      begin
      gotoXY(Xs,Ys);
      write(Nnt-i+1,' ');
      I_Read(F_NewOld,i,Old);
      I_Write(F_OldNew,Old,i);
      D_R_Read(F_xyz,Old,dim,xyz);
      D_R_write(F_xyz,i,dim,xyz);
      D_R_Read(F_Force,Old,dim,Force);
      D_R_write(F_Force,i,dim,Force);
      D_R_Read(F_Displ,Old,dim,Displ);
      D_R_write(F_Displ,i,dim,Displ);
      D_Byte_Read(F_Fd,Old,dim,Fd);
      D_Byte_write(F_Fd,i,dim,Fd);
      end;
   gotoXY(Xs,Ys);
   writeln('  ');
{
                Truncate Files after Nnt node
}
```

```
   Truncate(F_NewOld);
   Truncate(F_xyz);
   Truncate(F_Force);
   Truncate(F_Displ);
   Truncate(F_Fd);
{
                  Correct Element
}
   write('Correcting elements for compacted node numbers ');
   Xs:=WhereX;
   Ys:=WhereY;
   For iE:=1 to Net do
      begin
      gotoXY(Xs,Ys);
      write(Net-iE+1,' ');
      Nne2_I_Read(F_Elem,iE,NneMax,Elem);
      For j:=1 to Get_Nne(Elem[2]) do
         begin
         I_Read(F_OldNew,Elem[2+j],New);
         Elem[2+j]:=New;
         end;
      Nne2_I_Write(F_Elem,iE,NneMax,Elem);
      end;
   gotoXY(Xs,Ys);
   writeln('  ');
   Close(F_NewOld);
   Close(F_OldNew);
   Close(F_Used);
end;
{-------------------------------------------------------------------}
Procedure Node_Last(    Nnt,Net,NneMax: IType;
                    var F_Elem: FileI);
{
   Input:
      Nnt: number of nodes in total
      Net: Number of elements;
      NneMax: Maximum number of nodes in an element
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
   Output:
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Node are set to negative in Element matrix
              when they are use for the last time according
              to the numbering of the elements
   Note: When a node is used more than once in an element, it is only set
      for reduction once.
}
var
   iE,iNnt,iNne,OldNode,Nne: IType;
```

```
  Xs,Ys: Integer;
  Elem: VectI_Nne2;
  F_NewOld: FileI;
begin
  write('Finding last occurence of nodes in solution process ');
  Xs:=WhereX;
  Ys:=WhereY;
{
                Write all nodes as negative in New -> Old reference
                  a negative number indicates that it hasn't been
                  used yet in any element.
}
  Assign(F_NewOld,C_Dir+'NewToOld.P');
  reset(F_NewOld);
  For iNnt:=1 to Nnt do
     begin
     GotoXY(Xs,Ys);
     write(Net+Nnt-iNnt+1,' ');
     I_Read(F_NewOld,iNnt,OldNode);
     I_Write(F_NewOld,iNnt,-OldNode);    ( negative )
     end;
{
                Read element information in reverse order of solving process
                Look for negative OLD node # referenced by element node
                  numbers -> set element node # to negative
                             restore OLD node # to positive
}
  For iE:=Net downto 1 do
     begin
     GotoXY(Xs,Ys);
     write(iE,' ');
     Nne2_I_Read(F_Elem,iE,NneMax,Elem);
     Nne:=Get_Nne(Elem[2]);
     For iNne:=1 to Nne do
        begin
        I_Read(F_NewOld,Elem[2+iNne],OldNode);
        If OldNode<0 then
          begin
          Elem[2+iNne]:=-abs(Elem[2+iNne]);       ( negative node in element )
          I_Write(F_NewOld,abs(Elem[2+iNne]),abs(OldNode));    ( positive )
          end;
        end;
     Nne2_I_Write(F_Elem,iE,NneMax,Elem);
     end;
  GotoXY(Xs,Ys);
  writeln(' ');
  close(F_NewOld);
end;
{-------------------------------------------------------------------}
Procedure PassVariables(var VarName,VarValue: VectS80_Var;
                        var VarOtt: Byte);
{
```

```
   Output:
     VarName[1..Var_max]: variable names
     VarValue[1..Var_max]: variable values
     VarQtt: quantity of variables
}
Var
   i,iPar: Byte;
begin
{
                Transfer Parameter variables
}
   For iPar:=2 to ParamCount do
     begin
{
                Get VarName
}
     VarName[iPar-1]:='';
     i:=1;
     While (ParameterString[iPar][i]<>'=') and
           (i<=Length(ParameterString[iPar])) do
       begin
       VarName[iPar-1]:=VarName[iPar-1]+ParameterString[iPar][i];
       Inc(i);
       end;
{
                Get VarValue
}
     VarValue[iPar-1]:='';
     i:=Pos('=',ParameterString[iPar])+1;      { one after , }
     While (i<=Length(ParameterString[iPar])) do
       begin
       VarValue[iPar-1]:=VarValue[iPar-1]+ParameterString[iPar][i];
       Inc(i);
       end;
     Inc(VarQtt);
     end;
end;
{-------------------------------------------------------------------}
Procedure Get_NneMax_Dim(var F_Code: FileC;
                         var VarName,VarValue: VectS80_Var;
                         var VarQtt: Byte;
                         var NneMax,Dim: IType;
                         var Bad: Boolean);
{
   Input:
     F_Code: File with geometry definition
     VarName[1..Var_max]: variable names
     VarValue[1..Var_max]: variable values
     VarQtt: quantity of variables
   Output:
     NneMax: Maximum number of nodes in an element
     Bad: True if no known element was found
```

```
      Dim: Dimension of element types
}
var
   i,C_Type,Qtt,Nne: Itype;
   Finish: Boolean;
   KeyWord,Say: String255;
   IRS,Ni: VectI_Nc;
   Nr: VectR_Nc;
   Ns80: VectS80_Nc;
   ParameterVarQtt: Byte;
begin
   ParameterVarQtt:=VarQtt;   (remember original parameter qtt)
   ClrScr;
   writeln('Scanning for maximum number of nodes in an element ',
           'and dimension compatibility');
   writeln;
   Bad:=False;
   Seek(F_Code,0);
   NneMax:=0;
   Finish:=False;
   Dim:=0;
   While (Finish=False)and(Bad=False) do
      begin
      Get_command(F_Code,C_type,KeyWord,Say);
      If (C_type<>-1) and (C_type<>0) then
         Get_Number_pos(Say,VarName,VarValue,VarQtt,IRS,Ni,Nr,Ns80,Qtt,Bad);
      If C_Type<>0 then
         Write_Line(KeyWord,IRS,Ni,Nr,Ns80,Qtt);
      Case C_type of
        -1: begin
            Bad:=True;
            Write('   Unknown command!');
            end;
         0: Finish:=True;
         1: write('   ');   ( indent for next command after REPEAT )
         2: Set_Var(IRS,Ni,Nr,Ns80,Qtt,VarName,VarValue,VarQtt,Bad);
         3: begin
            If IRS[3]=0 then
               If (0<Ni[3]) and (Ni[3]<=EtypeMax) then
                  begin
                  TextColor(White);
                  writeln('Element type ',Ni[3]);
                  TextColor(LightGreen);
                  Nne:=Get_Nne(Ni[3]);
                  If NneMax<Nne then NneMax:=Nne;
                  If Dim=0 then
                     Dim:=Get_D(Ni[3])
                  else
                     If Dim<>Get_D(Ni[3]) then
                        writeln('Elements of different dimension mixed');
                  end;
            end;
```

```
         15: Set_Oper(IRS,Ni,Nr,Ns80,Qtt,VarName,VarValue,VarQtt,Bad);
         end;
      end;
   If (0=NneMax) and (Bad=False) then
      begin
      Bad:=True;
      writeln;
      writeln('   No valid element found');
      end;
{
                  Reset pointer to beginning of file
}
   Seek(F_Code,0);
{
                  Forget variables defined after parameter variables
}
   For i:=ParameterVarQtt+1 to VarQtt do
      begin
      VarName[i]:='';
      VarValue[i]:='';
      end;
   VarQtt:=ParameterVarQtt;
   If Bad=False then ClrScr;
end;
{-------------------------------------------------------------------}
var
   CoarseNntOld,CoarseDim,NneMax,Qtt,Qtt_0,Qtt_d,iParam,AgainTimes,i_Again,
      i,j,Dim,Nnt,NntOld,Net,C_type: IType;
   IRS,Ni,IRS_0,Ni_0,IRS_d,Ni_d: VectI_Nc;
   Nr,Nr_0,Nr_d: VectR_Nc;
   Ns80,Ns80_0,Ns80_d: VectS80_Nc;
   Bad,Finish,File_ok: Boolean;
   CurrentDir,KeyWordCommand,Say: String255;
   F_Code: FileC;
   F_xyz,F_Angle,F_Force,F_Displ,FC_xyz,FC_Displ: FileR;
   F_Elem,FC_OldNew: FileI;
   F_Fd: FileByte;
   FileName,CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   Origin: VectR_3;
   Laxis: MatrixR_3x3;
   LocalType: Char;
   MatLabel: VectS80_Nmat;
   VarName,VarValue: VectS80_Var;
   iPar,VarQtt: Byte;
   CaseToDatabase: VectByte_Nmat;
begin
{
      PREP CaseName
```

```
        CaseName: file name of case without ".GEN" extension.
           The process will run without waiting is there is no error.
      PREP CaseName VariableName1=Value1 ... VariableNameN=ValueN
        CaseName must be the first parameter.
        VariableName,Value: The variable "VariableName" is assigned
           the value "Value".
           This variable is passed into the preprocessed file.
           Attempts to re-define the variable in the file is ignored.
}
   BatchLogo;
   Bad:=False;
   Parameters;
{
                  Read directories
}
   ClrScr;
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
{
                  Clean previous *.P files
                  Set Status to Error
}
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   Exec('\Command.com','/C Del '+C_Dir+'*.P');
{
                  Initialize status
}
   CaseName:='Unknown';
   CoarseCaseName:=CoarseCaseName;   (same)
   StatusPrep:='';
   StatusForce:='';
   StatusDispl:='';
   StatusWorst:='';
   StatusFiberStrain:='';
   StatusFiberStress:='';
   StatusElemStrain:='';
   StatusElemStress:='';
   StatusGlobalStrain:='';
   StatusGlobalStress:='';
   Write_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
{
                  Pick current case
                  Case can be chosen in batch mode: PREP 2D.GEN<Enter>
}
   If ParamCount=0 then
```

```
      Pick_File(G_Dir,'*.GEN',AnyFile,True,FileName,File_ok)
   else
      begin
      File_ok:=False;
      If 1<=ParamCount then
         begin
         File_ok:=True;
         FileName:=ParameterString[1]+'.GEN';
         end;
      end;
{
                  Get Case Name = File Name without directory and extension
}
   CaseName:=FileName;
   While 0<Pos('.',CaseName) do Delete(CaseName,Length(CaseName),1);
   ResetTimerAll;
   StartTimer(1);
   If File_ok=True then
      begin
      Assign(F_Code,G_Dir+FileName);
      Reset(F_Code);
{
                  Set Number to a minimum: Dimension, Number of nodes,
                     Number of elements, Number of materials
}
      Dim:=0;
      Nnt:=0;
      Net:=0;
{
                  Assign Files
}
      Assign(F_xyz,C_Dir+'xyz.P');
      Assign(F_Force,C_Dir+'Force.P');
      Assign(F_Displ,C_Dir+'Displ.P');
      Assign(F_Fd,C_Dir+'Fd.P');
      Assign(F_Elem,C_Dir+'Elem.P');
      Assign(F_Angle,C_Dir+'Angle.P');
      Rewrite(F_xyz);
      Rewrite(F_Force);
      Rewrite(F_Displ);
      Rewrite(F_Fd);
      Rewrite(F_Elem);
      Rewrite(F_Angle);
      If CoarseCaseName<>'Unknown' then
         begin
         Get_I('NntOld.C',CoarseNntOld);
         Get_I('Dim.C',CoarseDim);
         Assign(FC_xyz,C_Dir+'xyz.C');
         Assign(FC_OldNew,C_Dir+'OldToNew.C');
         Assign(FC_Displ,C_Dir+'Displ.C');
         Reset(FC_xyz);
         Reset(FC_OldNew);
```

```
      Reset(FC_Displ);
      end;
{
                Initialize Variables
}
   VarQtt:=0;
   For i:=1 to Var_max do
      begin
      VarName[i]:='';
      VarValue[i]:='';
      end;
{
                Verify format of Parameter variables: VarName,Value
}
   For iPar:=2 to ParamCount do
      begin
      If Pos('=',ParameterString[iPar])=0 then
         begin
         Bad:=True;
         Writeln('   Parameter variable ',iPar-1,' is missing a equal sign:');
         writeln('   ',ParameterString[iPar]);
         end
      end;
{
                Transfer Parameter variables
}
   If Bad=False then
      PassVariables(VarName,VarValue,VarQtt);
{
                Get Material Labels -> upper case without blanks
                   because blanks are ignored in *.GEN file
                Initialize Case to Database material number relation
}
   If Bad=False then
      begin
      Get_MatLabel(MatLabel);
      For i:=1 to Nmat_max do
         begin
         for j:=1 to Length(MatLabel[i]) do
            MatLabel[i][j]:=UpCase(MatLabel[i][j]);
         While 0<Pos(' ',MatLabel[i]) do
            Delete(MatLabel[i],Pos(' ',MatLabel[i]),1);
         end;
      For i:=1 to Nmat_max do
         CaseToDatabase[i]:=0;
      end;
{
                Initialize Local coordinate system direction and origin
}
   If Bad=False then
      begin
      For i:=1 to Dim3 do
```

```
            For j:=1 to Dim3 do
               Laxis[i,j]:=0;
         Laxis[1,1]:=1;
         Laxis[2,2]:=1;
         Laxis[3,3]:=1;
         For i:=1 to Dim3 do
            Origin[i]:=0;
         LocalType:='R';  { rectangular coordinate system }
         end;
{
                  Get Highest number of nodes in an element
}
      If Bad=False then
         Get_NneMax_Dim(F_Code,VarName,VarValue,VarQtt,NneMax,Dim,Bad);
{
                  Interpret the File with PREP commands
}
      If Bad=False then
         begin
         write('Interpreting file ');
         TextColor(LightRed);
         write(FileName);
         TextColor(LightGreen);
         writeln(':');
         end;
      Finish:=False;           { END command not encountered }
      While (Bad=False) and (Finish=False) do
         begin
         Get_command(F_Code,C_type,KeyWordCommand,Say);
         If (C_type<>-1) and (C_type<>0) then
            Get_Number_pos(Say,VarName,VarValue,VarQtt,IRS,Ni,Nr,Ns80,Qtt,Bad);
         If C_Type<>0 then
            Write_Line(KeyWordCommand,IRS,Ni,Nr,Ns80,Qtt);
         Case C_type of
           -1: begin
               Bad:=True;
               Write('   Unknown command!');
               end;
            0: Finish:=True;
            1: begin
{
                  REPEAT command
                  Get number of times the following command is to be repeated
}
               If Qtt<2 then
                  begin
                  Bad:=True;
                  writeln('   At least 2 parameters needed');
                  end;
               If Bad=False then
                  If IRS[1]<>0 then
                     begin
```

```
                    Bad:=True;
                    writeln('   AgainTimes must be an integer');
                    end;
                 If Bad=False then
                   begin
                   AgainTimes:=Ni[1];
                   If AgainTimes<=0 then
                     begin
                     Bad:=True;
                     writeln('   REPEAT must be done at least once');
                     end;
                   end;
{
                    Get quantity of items to increment
                    Get increments
                    Get type of increments (Integer-Real-String)
}
                 If Bad=False then
                   begin
                   Qtt_d:=Qtt-1;
                   For iParam:=1 to Qtt_d do
                     begin
                     IRS_d[iParam]:=IRS[iParam+1];
                     Case IRS_d[iParam] of
                       0: Ni_d[iParam]:=Ni[iParam+1];
                       1: Nr_d[iParam]:=Nr[iParam+1];
                       2: Ns80_d[iParam]:=Ns80[iParam+1];
                       end;
                     end;
{
                    Get command after REPEAT with initial parameters
}
                   Get_command(F_Code,C_type,KeyWordCommand,Say);
                   If (C_type<>-1) and (C_type<>0) then
                     Get_Number_pos(Say,VarName,VarValue,VarQtt,
                                    IRS_0,Ni_0,Nr_0,Ns80_0,Qtt_0,Bad);
                   If C_Type<>0 then
                     begin
                     write('   ');
                     Write_Line(KeyWordCommand,IRS_0,Ni_0,Nr_0,Ns80_0,Qtt_0);
                     end;
                   Case C_type of
                    -1: begin
                        Bad:=True;
                        Write('   Unknown command!');
                        end;
                     0: begin
                        Finish:=True;
                        Bad:=True;
                        Write('   REPEAT must be followed by another command!');
                        end;
                     1: begin
```

```
                    Bad:=True;
                    Write('   REPEAT must not be repeated!');
                    end;
                 end;
              end;
(
               Compare REPEAT and following command for a match
               in length and types
)
           If (C_Type in [2..16]) and (Bad=False) then
              begin
              If Qtt_0<>Qtt_d then
                 begin
                 Bad:=True;
                 writeln('   REPEAT and ',KeyWordCommand,
                         ' must have a corresponding number of parameters');
                 end
              else
                 Qtt:=Qtt_0;
              iParam:=1;
              While (iParam<=Qtt) and (Bad=False) do
                 begin
                 If IRS_d[iParam]=2 then
                    begin
                    Bad:=True;
                    writeln('   A string is not incremented');
                    end;
                 If (IRS_0[iParam]<>IRS_d[iParam]) and
                    (0 <= IRS_d[iParam]) then
                    begin
                    Bad:=True;
                    writeln('   Number used for increase must be of ',
                            'same type as parameter ',iParam,
                            ' in ',KeyWordCommand);
                    end;
                 Inc(iParam);
                 end;
              end;
(
               Get values to be passed to following command
)
           If (C_Type in [2..16]) and (Bad=False) then
              begin
              For i_Again:=0 to AgainTimes do
                 begin
                 For iParam:=1 to Qtt do
                    begin
                    Case IRS_d[iParam] of
                      -1: begin
                          Ni[iParam]:=Ni_0[iParam];
                          Nr[iParam]:=Nr_0[iParam];
                          Ns80[iParam]:=Ns80_0[iParam];
```

```
                    end;
                  0: Ni[iParam]:=Ni_0[iParam]+i_Again*Ni_d[iParam];
                  1: Nr[iParam]:=Nr_0[iParam]+i_Again*Nr_d[iParam];
                  2: Ns80[iParam]:=Ns80_0[iParam];
                  end;
               end;
            IRS:=IRS_0;
{
            Execute following command with appropriate parameters
}
            If 1<=i_Again then
               begin
               write('-> ');
               Write_Line(KeyWordCommand,IRS,Ni,Nr,Ns80,Qtt);
               end;
            Case C_Type of
               2: Set_Var(IRS,Ni,Nr,Ns80,Qtt,
                          VarName,VarValue,VarQtt,Bad);
               3: Set_Elem(IRS,Ni,Nr,Qtt,CaseToDatabase,F_XYZ,
                           F_Elem,F_Angle,Net,NneMax,Nnt,Dim,Bad);
               4: Set_Node(IRS,Ni,Nr,Qtt,LocalType,Origin,Laxis,
                           F_xyz,F_Force,F_Displ,F_Fd,Nnt,Dim,Bad);
               5: Set_Mat(IRS,Ni,Ns80,Qtt,MatLabel,CaseToDatabase,Bad);
               6: Set_ForceDispl('F',IRS,Ni,Nr,Qtt,Dim,Nnt,
                                 F_xyz,F_Force,F_Displ,F_Fd,Bad);
               7: Set_ForceDispl('D',IRS,Ni,Nr,Qtt,Dim,Nnt,
                                 F_xyz,F_Force,F_Displ,F_Fd,Bad);
               8: Set_Ngen(IRS,Ni,Nr,Qtt,Dim,LocalType,Origin,Laxis,
                           F_XYZ,F_Force,F_Displ,F_Fd,Nnt,Bad);
               9: Set_Egen(IRS,Ni,Nr,Qtt,F_XYZ,F_Elem,F_Angle,Net,
                           NneMax,Nnt,Dim,Bad);
               10: Set_Traction(IRS,Ni,Nr,Qtt,Net,F_XYZ,F_Elem,F_Force,F_Fd,
                                NneMax,Dim,Bad);
               11: Set_Local(IRS,Ni,Nr,Ns80,Qtt,LocalType,Origin,Laxis,Bad);
               12: Set_Old(IRS,Ni,Nr,Qtt,Dim,CoarseDim,CoarseNntOld,
                           Nnt,FC_OldNew,F_Fd,F_xyz,FC_xyz,F_Force,F_Displ,
                           FC_Displ,Bad);
               13: Set_Inter(IRS,Ni,Nr,Qtt,Dim,LocalType,Origin,Laxis,
                             Nnt,F_Fd,F_xyz,F_Force,F_Displ,Bad);
               14: Set_Edel(IRS,Ni,Qtt,F_Elem,F_Angle,NneMax,Bad);
               15: Set_Oper(IRS,Ni,Nr,Ns80,Qtt,
                            VarName,VarValue,VarQtt,Bad);
               16: Set_Emod(IRS,Ni,Nr,Qtt,CaseToDatabase,
                            F_Elem,F_Angle,Net,NneMax,Bad);
               end;
            end;
         end;
{
  - - - - - - Execute all other commands - -  - - - - - - - - - -
}
         2: Set_Var(IRS,Ni,Nr,Ns80,Qtt,VarName,VarValue,VarQtt,Bad);
```

```
            3: Set_Elem(IRS,Ni,Nr,Qtt,CaseToDatabase,F_XYZ,
                       F_Elem,F_Angle,Net,NneMax,Nnt,Dim,Bad);
            4: Set_Node(IRS,Ni,Nr,Qtt,LocalType,Origin,Laxis,
                       F_xyz,F_Force,F_Displ,F_Fd,Nnt,Dim,Bad);
            5: Set_Mat(IRS,Ni,Ns80,Qtt,MatLabel,CaseToDatabase,Bad);
            6: Set_ForceDispl('F',IRS,Ni,Nr,Qtt,Dim,Nnt,
                            F_xyz,F_Force,F_Displ,F_Fd,Bad);
            7: Set_ForceDispl('D',IRS,Ni,Nr,Qtt,Dim,Nnt,
                            F_xyz,F_Force,F_Displ,F_Fd,Bad);
            8: Set_Ngen(IRS,Ni,Nr,Qtt,Dim,LocalType,Origin,Laxis,
                       F_XYZ,F_Force,F_Displ,F_Fd,Nnt,Bad);
            9: Set_Egen(IRS,Ni,Nr,Qtt,F_XYZ,F_Elem,F_Angle,Net,
                       NneMax,Nnt,Dim,Bad);
           10: Set_Traction(IRS,Ni,Nr,Qtt,Net,F_XYZ,F_Elem,F_Force,F_Fd,
                          NneMax,Dim,Bad);
           11: Set_Local(IRS,Ni,Nr,Ns80,Qtt,LocalType,Origin,Laxis,Bad);
           12: Set_Old(IRS,Ni,Nr,Qtt,Dim,CoarseDim,CoarseNntOld,
                       Nnt,FC_OldNew,F_Fd,F_xyz,FC_xyz,F_Force,F_Displ,
                       FC_Displ,Bad);
           13: Set_Inter(IRS,Ni,Nr,Qtt,Dim,LocalType,Origin,Laxis,
                         Nnt,F_Fd,F_xyz,F_Force,F_Displ,Bad);
           14: Set_Edel(IRS,Ni,Qtt,F_Elem,F_Angle,NneMax,Bad);
           15: Set_Oper(IRS,Ni,Nr,Ns80,Qtt,VarName,VarValue,VarQtt,Bad);
           16: Set_Emod(IRS,Ni,Nr,Qtt,CaseToDatabase,
                        F_Elem,F_Angle,Net,NneMax,Bad);
         end;
      end;
{
                Close files which are of no further use
}
   Close(F_Code);
   If CoarseCaseName<>'Unknown' then
      begin
      Close(FC_xyz);
      Close(FC_OldNew);
      Close(FC_Displ);
      end;
{
                Pause on error
}
   If Bad=True then
      begin
      Bad_Beep;
      StatusPrep:='Error';
      Write_Status(CaseName,CoarseCaseName,
                   StatusPrep,StatusForce,StatusDispl,StatusWorst,
                   StatusFiberStrain,StatusFiberStress,
                   StatusElemStrain,StatusElemStress,
                   StatusGlobalStrain,StatusGlobalStress);
      Wait;
      end;
{
```

```
                Compress node and element numbers
}
     If Bad=False then
        begin
        TextColor(LightRed+Blink);
        writeln;
        Writeln('Success!');
        TextColor(LightGreen);
{
                Compact element numbering
}
        ElementCrush(F_Elem,F_Angle,NneMax,Net);
{
                Compact node index
}
        NntOld:=Nnt;            (Save uncompacted node number)
        NodeCrush(F_Elem,Nnt,NneMax,Net,F_XYZ,F_Force,F_Displ,F_Fd,Dim);
{
                Find last occurence of nodes in solving process
}
        Node_Last(Nnt,Net,NneMax,F_Elem);
{
                Find out maximum front matrix size during solving process
}
        Write('Now writting files: ');
        Put_I('Net.P',Net);
        Put_I('Nnt.P',Nnt);
        Put_I('NntOld.P',NntOld);
        Put_I('Dim.P',Dim);
        Put_I('NneMax.P',NneMax);
        StatusPrep:='Done';
        Write_Status(CaseName,CoarseCaseName,
                     StatusPrep,StatusForce,StatusDispl,StatusWorst,
                     StatusFiberStrain,StatusFiberStress,
                     StatusElemStrain,StatusElemStress,
                     StatusGlobalStrain,StatusGlobalStress);
        end;
{
                Close Files
}
     Close(F_xyz);
     Close(F_Force);
     Close(F_Displ);
     Close(F_Fd);
     Close(F_Elem);
     Close(F_Angle);
     end;
   StopTimer(1);
   If (60<ReadTimer(1)) and (Bad=False) then Alert;
   ClrScr;
end.
```

PUT.PAS

```
{$N+}
Unit Put;
Interface
Uses
   Declare,Crt;
Procedure Put_I(   Name: String255;
                   NumberI: IType);
Procedure Put_Mat(var Mat: MatrixR_Nmatx6);
Procedure Put_MatLabel(var MatLabel: VectS80_Nmat);
Procedure Put_MaxStress(var MaxStress: MatrixR_Nmatx9);
{-------------------------------------------------------------------}
Implementation
Procedure Put_I;
{
Procedure Put_I(   Name: String255;
                   NumberI: IType);
}
{
   Input:
      Name: File name with extension
      NumberI: IType Number
   Output:
      Writes NumberI to File "Name"
}
var
   F: FileI;
   i,X,Y: IType;
begin
   X:=WhereX;
   Y:=WhereY;
   Name:=C_Dir+Name;
   write(Name);
   Assign(F,Name);
   Rewrite(F);
   Write(F,NumberI);
   Close(F);
   GotoXY(X,Y);
   For i:=1 to Length(Name) do write(' ');
   GotoXY(X,Y);
end;
{-------------------------------------------------------------------}
Procedure Put_Mat;
{
Procedure Put_Mat(var Mat: MatrixR_Nmatx6);
}
{
   Input:
      Mat[1..Nmat,i]: El, Et, Glt, Gtt, nlt, ntt
                      El=-1 if undefined
}
```

```
var
   F: File of MatrixR_Nmatx6;
begin
   Assign(F,'Mat.M');
   Rewrite(F);
   Write(F,Mat);
   Close(F);
end;
{-------------------------------------------------------------------}
Procedure Put_MatLabel;
{
Procedure Put_MatLabel(var MatLabel: VectS80_Nmat);
}
{
   Input:
      MatLabel[iMaterial]: Name of material, details
}
var
   F: File of VectS80_Nmat;
begin
   Assign(F,'MatLabel.M');
   Rewrite(F);
   Write(F,MatLabel);
   Close(F);
end;
{-------------------------------------------------------------------}
Procedure Put_MaxStress;
{
Procedure Put_MaxStress(var MaxStress: MatrixR_Nmatx9);
}
{
   Input:
      MaxStress[iMaterial,j]: iMaterial: Database material number
                              j =1: X
                                =2: X'
                                =3: Y
                                =4: Y'
                                =5: Z
                                =6: Z'
                                =7: Sxy max
                                =8: Sxz max
                                =9: Syz max
}
var
   F: File of MatrixR_Nmatx9;
begin
   Assign(F,'MaxStress.M');
   Rewrite(F);
   Write(F,MaxStress);
   Close(F);
end;
{-------------------------------------------------------------------}
```

end.

REGRES.PAS

```
{-------------------------------------------------------------------
  Program to match data points to a formula
  Initially written in May '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 35000,0,655360}   (381k used in heap -> 390144 bytes)
Uses
  Dos,Crt,WhatKey,Edit,PickFile,WaitKey,Nombre,Sonore,Math,H_common,
  Timer,Declare;
{-------------------------------------------------------------------}
Const
  Nmax = 127;       {90 with Double reals, 127 with single reals}
  Nmax_plus1 = 128;
Type
  SmallRtype = Single;
  MatrixR_NxN = array[1..Nmax,1..Nmax] of SmallRtype;
  MatrixR_Ptr = ^MatrixR_NxN;
  VectR_N = array[1..Nmax] of SmallRtype;
  VectR_Ptr = ^VectR_N;
  VectI_N = array[1..Nmax] of integer;   {Not Itype}
  VectI_Ptr = ^VectI_N;
  AugmentedMatrixR_NxN1 = array[1..Nmax,1..Nmax_plus1] of SmallRtype;
  AugmentedMatrixR_Ptr = ^AugmentedMatrixR_NxN1;
  VectR_27 = array[1..27] of SmallRtype;
{-------------------------------------------------------------------}
Procedure Message;
var
  Xl: Byte;
begin
  ClrScr;
  TextColor(LightGreen);
  writeln;
  GotoXY(28,WhereY); Writeln('  x      x     x  x      ');
  GotoXY(28,WhereY); Writeln('x+x--~x----x~----~--x-~+~');
  GotoXY(28,WhereY); Writeln(' |                   ~| ');
  GotoXY(28,WhereY); Writeln('~|x    R E G R E S    x|~');
  GotoXY(28,WhereY); Writeln('~|x                   | ');
  GotoXY(28,WhereY); Writeln('x+--x---x-~--x--~-~-x-~+~');
  GotoXY(28,WhereY); Writeln(' ~    ~               ~  ');
  GotoXY(35,5);
  TextColor(White);
  write('R E G R E S');
  GotoXY(1,11);
  TextColor(LightCyan);
  Xl:=19;
  GotoXY(Xl-1,WhereY); Writeln('+----+-------------------------------------+');
  GotoXY(Xl-1,WhereY); Writeln('| xx |                                     |');
  GotoXY(Xl-1,WhereY); Writeln('+----+                                     |');
  GotoXY(Xl-1,WhereY); Writeln('|                                          |');
  GotoXY(Xl-1,WhereY); Writeln('|                                          |');
```

```
   GotoXY(Xl-1,WhereY); Writeln('¦                                                ¦');
   GotoXY(Xl-1,WhereY); Writeln('¦                                                ¦');
   GotoXY(Xl-1,WhereY); Writeln('¦                                                ¦');
   GotoXY(Xl-1,WhereY); Writeln('+----------------------------------------------+');
   TextColor(LightRed);
   GotoXY(Xl+1,12);       write('1a');
   GotoXY(Xl+5,12);       write('       Jerome Daoust, Ph.D.');
   GotoXY(Xl+5,WhereY+1); write('  Composite Material Laboratory');
   GotoXY(Xl,WhereY+1); write('   Department of Mechanical Engineering    ');
   GotoXY(Xl,WhereY+1); write('          Concordia University             ');
   GotoXY(Xl,WhereY+1); write('        1455 de Maisonneuve West           ');
   GotoXY(Xl,WhereY+1); write('     Montreal, Quebec, Canada, H3G 1M8     ');
   GotoXY(Xl,WhereY+1); write('             (514) 259-4145                ');
   GotoXY(79,25);
   TextColor(LightGreen);
   Wait;
end;
{-------------------------------------------------------------------------}
Procedure HelpGeneral(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'General information\';
   Repeat
      Header(HelpPath);
      writeln('This program makes its calculation in single precision (8 significant digits).');
      writeln('It was written by Jerome Daoust as part of a Ph.D. in Mechanical');
      writeln('   Engineering in composite materials (1988).');
      writeln('Written in PASCAL, this program makes linear regressions.');
      writeln('It was ment as a tool within DIRECT (by the same author).');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure HelpLotus(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Lotus (*.PRN) files\';
   Repeat
      Header(HelpPath);
      writeln('Using Lotus 1-2-3, you can print a worksheet (or a part)');
      writeln('into a file in the same way you would print to the printer');
      writeln('with \ Print File. Give the file a Name and a Range.');
```

```
      writeln('Press Go and Quit. Further help is available from Lotus with <F1>.');
      writeln('Your range should be formatted as: Y=f(a,b, ... ,z)');
      writeln('   y1 a1 [b1 c1 ... z1]');
      writeln('   y2 a2 [b2 c2 ... z2]');
      writeln('   ...');
      writeln('   yN aN [bN cN ... zN]');
      writeln('A top margin in the file is ignored.');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure HelpFormula(HelpPath: String255);
var
   C: char;
   Action: String6;
   Long: Byte;
begin
   Beep;
   Long:=Length(HelpPath);
   HelpPath:=HelpPath+'Formula\';
   Repeat
      Header(HelpPath);
      writeln('You can have up to ',Nmax,' terms in the formula.');
      writeln('You can have 26 variables: a...z.');
      writeln('Letter case is ignored.');
      writeln('Example: aa+a+ab+b+1');
      writeln(' --> (C1)aa + (C2)a + (C3)ab + (C4)b + (C5)1 = Y');
      writeln('      C1...C5 will be determined.');
      writeln('Terms can be as the following: 1 a  ab  aa  a2  a2b  a-0.33b');
      writeln('A power in exponent form is not allowed: 3E4. Give 30000 instead. ');
      writeln('Spaces are ignored: "a + 1" is the same as "a+1".');
      writeln('If all variables were normalized (-1...1), the contribution');
      writeln('   of each term is shown by the constants found (Ci).');
      GoBack(HelpPath,long,C,Action);
      until Action='ESC';
end;
{-------------------------------------------------------------------------}
Procedure Help;
var
   HelpPath: String255;
   C: char;
   Action: String6;
begin
   ClrScr;
   HelpPath:='\';
   TextColor(LightGreen);
   Repeat
      Header(HelpPath);
      writeln('G- General information');
      writeln('L- Lotus (*.PRN) files');
      writeln('F- Formula format');
      writeln('<Esc>- Exit HELP');
```

```
      Get_Key(C,Action); C:=UpCase(C);
      If C='G' then HelpGeneral(HelpPath);
      If C='L' then HelpLotus(HelpPath);
      If C='F' then HelpFormula(HelpPath);
      until Action='ESC';
   Beep;
   ClrScr;
end;
{------------ ------------------------------------------------------------------}
Procedure GenerateFormula(var VarQtt: Itype;
                          var F: VectC_Screen;
                          var L,Long: integer);
{
   Input:
      VarQtt: quantity of variable excluding Y in Y=f(A,B,...Z)
      Long: maximum value of L
   Output:
      F[1..L]: character for function
      L: quantity of charactes in F
}
var
   Xs,Ys,iS,i: byte;
   v,max: array[0..26] of byte;
   S: String255;
   AllZero,AllMax,IncrementNext: Boolean;
begin
   ClrScr;
{
                  Input maximum degree for each variable
}
   For i:=0 to VarQtt do
      v[i]:=0;
   For i:=1 to VarQtt do
      begin
      max[i]:=0;
      write('Maximum degree for variable ',Chr(64+i),': ');
      Xs:=WhereX;
      Ys:=WhereY;
      readln(max[i]);
      GotoXY(Xs,Ys);
      TextColor(LightRed);
      writeln(max[i],'          ');
      TextColor(LightGreen);
      end;
{
                  Generate formula starting with term: 1
}
   L:=0;
   AllZero:=True;
   Repeat
{
                  Set first term as '1'
```

```
                Put '+' sign between terms
}
      If AllZero=True then
         begin
         Inc(L);
         If L<=Long then F[L]:='1';
         AllZero:=False;
         end
      else
         begin
         Inc(L);
         If L<=Long then F[L]:='+';
         end;
{
                Put term
}
      For i:=1 to VarQtt do
         begin
         If 0<>v[i] then
            begin
            Inc(L);
            If L<=Long then F[L]:=Char(96+i);   { a...z }
            Str(v[i],S);
            If S<>'1' then
               begin
               For iS:=1 to Length(S) do
                  begin
                  Inc(L);
                  If L<=Long then F[L]:=S[iS];
                  end;
               end;
            end;
         end;
{
                Look if all powers are at maximum
}
      AllMax:=True;
      For i:=1 to VarQtt do
         If v[i]<max[i] then AllMax:=False;
{
                Increment powers
}
      If AllMax=False then
         begin
         IncrementNext:=True;
         For i:=VarQtt downTo 1 do
            begin
            If IncrementNext=True then
               begin
               If max[i]=v[i] then
                  begin
                  v[i]:=0;
```

```
                IncrementNext:=True;
                end
             else
                begin
                Inc(v[i]);
                IncrementNext:=False;
                end;
             end;
          end;
       end;
    until ((AllMax=True) or (Long<L));
{
                Verify length of formula
}
   If Long<L then
      begin
      writeln('The formula is too long');
      L:=Long;
      Wait;
      end;
end;
{-------------------------------------------------------------------------}
Procedure GE_SCP(var A_square: MatrixR_Ptr;  {var to save memory}
                 var Y: VectR_N;             {var to save memory}
                     n: integer;
                 var X: VectR_N;
                 var Error_Type: Integer);
{
   Gaussian Elimination with Scaled Column Pivoting
      ( better then Maximal Column Pivoting )
   Taken from "Numerical Analysis", third edition, Burden & Faires
   Algorithm 6.3 page 328 along with algorithm 6.2 page 326
                            written by Jerome Daoust in Fall '87
   Solves the  n x n  linear system:
      E1: A(1,1).X(1) + A(1,2).X(2) + ... + A(1,n).X(n) = Y(1)
      E2: A(2,1).X(1) + A(2,2).X(2) + ... + A(2,n).X(n) = Y(2)
      ...
      En: A(n,1).X(1) + A(n,2).X(2) + ... + A(n,n).X(n) = Y(n)
   Input:
      A_square[1..N,1..N]: matrix giving the coefficients
                           of the equations
      Y[1..N]: Constants
      N: number of equations and unknowns
   Output:
      X[1..N]: Solution
      Error_Type: =0 if completed successfully
                  =1 if the system has no unique solution
}
var
   i,j,k,p,Ncopy: integer;
   Max: Rtype;
   A: AugmentedMatrixR_Ptr;
```

```
   m: MatrixR_Ptr;
   S: VectR_Ptr;
   Nrow : VectI_Ptr;
begin
   New(A);
   New(m);
   New(S);
   New(Nrow);
{
                  Transfer A_square and Y into augmented matrix A
}
   For i:=1 to N do
      begin
      For j:=1 to N do
         A^[i,j]:=A_square^[i,j];
      A^[i,N+1]:=Y[i];
      end;
{
                  Look for a nul row
}
   Error_Type:=0;
   For i:=1 to n do
      begin
      S^[i]:=0;
      For j:=1 to n do
         If S^[i]<abs(A^[i,j]) then S^[i]:=abs(A^[i,j]);
      If S^[i]=0 then Error_Type:=1;
      Nrow^[i]:=i;
      end;
   If Error_Type=0 then
      begin
{
                  Elimination Process
}
      For i:=1 to n-1 do
         begin
         Max:=0;
         For j:=i to n do
            If Max<abs(A^[Nrow^[j],i])/S^[Nrow^[j]] then
               begin
               p:=j;
               Max:=abs(A^[Nrow^[j],i])/S^[Nrow^[j]]
               end;
         If A^[Nrow^[p],i]=0 then
            Error_Type:=1
         else
            begin
{
                  Simulated row interchange
}
            If Nrow^[i]<>Nrow^[p] then
               begin
```

```
                Ncopy:=Nrow^[i];
                Nrow^[i]:=Nrow^[p];
                Nrow^[p]:=Ncopy;
                end;
             For j:=i+1 to n do
                begin
                m^[Nrow^[j],i]:=A^[Nrow^[j],i]/A^[Nrow^[i],i];
                For k:=1 to n+1 do
                   A^[Nrow^[j],k]:=A^[Nrow^[j],k]-m^[Nrow^[j],i]*A^[Nrow^[i],k];
                end;
             end;
          end;
       end;
   If A^[Nrow^[n],n]=0 then Error_Type:=1;
   If Error_Type=0 then
{
                  Start back substitition
}
       begin
       X[n]:=A^[Nrow^[n],n+1]/A^[Nrow^[n],n];
       For i:=n-1 downto 1 do
          begin
          X[i]:=A^[Nrow^[i],n+1];
          For j:=i+1 to n do
             X[i]:=X[i]-A^[Nrow^[i],j]*X[j];
          X[i]:=X[i]/A^[Nrow^[i],i];
          end;
       end;
   Dispose(Nrow);
   Dispose(S);
   Dispose(m);
   Dispose(A);
end;
{-------------------------------------------------------------------}
Procedure Get_Data(var F_Data: Text;
                   var VarQtt,DataQtt: Itype;
                      DisplayLine: Byte);
{
   Input:
      F_Data: Data file:  Y1 a1 b1 c1 ...
                          ...
                          Yn an bn cn ...
      DisplayLine: Line to display messages
   Output:
      VarQtt: quantity of variable excluding Y in Y=f(A,B,...Z)
      DataQtt: quantity of data lines
}
var
   i: integer;
   DataStr: String255;
   Xs,Ys: Byte;
begin
```

```
   GotoXY(1,DisplayLine);
   write('Getting data quantity: ');
   Xs:=WhereX;
   Ys:=WhereY;
{
                  Get VarQtt
}
   Reset(F_Data);
   Repeat
      Readln(F_Data,DataStr);
      until ((DataStr<>'') or (EOF(F_Data)=True));
   DataStr:=DataStr+' ';              { put a space at the end }
   While DataStr[1]=' ' do            { remove leading spaces }
      Delete(DataStr,1,1);
   While 0<Pos('  ',DataStr) do       { remove double spaces }
      Delete(DataStr,Pos('  ',DataStr),1);
   VarQtt:=-1;
   For i:=1 to Length(DataStr) do
      If DataStr[i]=' ' then
         Inc(VarQtt);
{
                  Get DataQtt (some line are empty due to page formatting)
}
   Reset(F_Data);
   DataQtt:=0;
   While EOF(F_Data)=False do
      begin
      Readln(F_Data,DataStr);
      If DataStr<>'' then Inc(DataQtt);
      GotoXY(Xs,Ys);
      write(DataQtt)
      end;
   GotoXY(1,DisplayLine);
   ClrEol;
   GotoXY(79,25);
end;
{-------------------------------------------------------------------------------}
Procedure LineToVect(var DataStr: String255;
                     var d: VectR_27;
                     var Bad: Boolean);
{
   Input:
      DataStr: Line of data: Ystring Astring ... Zstring
   Output:
      d[1..27]: Y A ... Z   where Y=F(A,B,...Z)
      Bad = True if a string could not be converted into a number
}
var
   IRtype,i,ID: Byte;
   NumberS: String255;
   NumberI: Itype;
   NumberR: Rtype;
```

```
begin
{
                Transform line into a vector: d[1..VarQtt+1]
                Ns+' '+Ns+' '+Ns+' '
}
   iD:=0;
   DataStr:=DataStr+' ';              { put a space at the end }
   While DataStr[1]=' ' do            { remove leading spaces }
      Delete(DataStr,1,1);
   While 0<Pos('  ',DataStr) do       { remove double spaces }
      Delete(DataStr,Pos('  ',DataStr),1);
   NumberS:='';
   For i:=1 to length(DataStr) do
      begin
      If DataStr[i]=' ' then
         begin
         StoX(NumberS,NumberI,NumberR,IRtype);
         NumberS:='';
         Inc(iD);
         Case IRtype of
            0: begin
               writeln('A string is found in the data file: ',NumberS);
               Bad:=True;
               end;
            1: d[iD]:=NumberI*1.0;
            2: d[iD]:=NumberR;
            end;
         end
      else
         NumberS:=NumberS+DataStr[i];
      end;
end;
{-------------------------------------------------------------------}
Procedure EvalString(    S: String;
                     var d: VectR_27;
                     var Value: Rtype;
                     var Bad: Boolean);
{
   Input:
      S: string to be evaluated
         examples:    ABC  1  A^12.B^3.C
      d[1..27]: Y A ... Z   where Y=F(A,B,...Z)
   Output:
      Value: Value of string
      Bad = True if an error occured
}
var
   i,iP,Ls,IRtype,DotPlace,Fin: byte;
   NumberI: Itype;
   NumberR,PowR: Rtype;
   PowS: String255;
begin
```

```
   Ls:=Length(S);
   Value:=1.0;
   If S<>'1' then
      begin
      i:=1;
      While (i<=Ls) do
         begin
         If S[i] in ['A'..'Z'] then
            begin
            If (S[i+1] in ['0'..'9']) and (i<Ls) then
               begin
               iP:=i+1;
               While (iP+1<=Ls) and (not (S[iP+1] in ['A'..'Z'])) do
                  Inc(iP);
               PowS:=Copy(S,i+1,iP-i);
               StoX(PowS,NumberI,NumberR,IRtype);
               Case IRtype of
                  0: begin
                     WriteLn('Exponent in ',S,' is not a number');
                     Bad:=True;
                     iP:=Ls;   {exit loop)
                     end;
                  1: PowR:=NumberI*1.0;
                  2: PowR:=NumberR;
                  end;
               Value:=Value*PowerR(d[Ord(S[i])-63],PowR);
               i:=iP+1;
               end
            else
               begin
               Value:=Value*d[Ord(S[i])-63];  ( A is d[2] ...)
               Inc(i);
               end;
            end
         else
            begin
            writeln('Variable is not in the A...Z range: ',S[i]);
            Bad:=True;
            end;
         end;
      end;
end;
{-------------------------------------------------------------------}
Procedure Regression(var F_Data: Text;
                         F: VectC_Screen; ( Not var (modified) )
                         L: integer;
                         VarOtt,DataOtt: Itype;
                         FormulaFileName: String255);
{
   Input:
      F_Data: Data file:  Y1 a1 b1 c1 ...
                          ...
```

```
                        Yn an bn cn ...
      F[1..L]: character for function
      L: quantity of charactes in F
      VarQtt: quantity of variable excluding Y in Y=f(A,B,...Z)
      DataQtt: quantity of data lines
      FormulaFileName: File name of formula
}
Const
   Lterm = 40;
var
   A: MatrixR_Ptr;
   C,Y: VectR_N;
   d: VectR_27;
   Error_Type,iL,iCte,CteQtt: integer;
   S,DataStr: String255;
   Bad: Boolean;
   iData: Itype;
   DifferenceMax,Difference,AlertTime,Sum2,Tr,Fr: Rtype;
   Ys: array[1..Nmax] of String[Lterm];
   Yr: array[1..Nmax] of Rtype;
   iC,jC,IRtype,Xscr,Yscr: Byte;
begin
   New(A);
   ClrScr;
   AlertTime:=2*60.0;
   ResetTimer(1);
   StartTimer(1);
   Bad:=False;
{
                  Check formula length
}
   If Bad=False then
      begin
      writeln('Checking presence of a formula');
      If L<=0 then
         begin
         Bad:=True;
         Writeln('No formula entered');
         end;
      end;
{
                  Change F to upper case
}
   If Bad=False then
      begin
      writeln('Checking range of variables in formula with quantity in file');
      For iL:=1 to L do
         begin
         F[iL]:=UpCase(F[iL]);
         If (VarQtt<Ord(F[iL])-64) and (F[iL] in ['A'..'Z']) then
            begin
            Writeln('Insufficient number of variables in file');
```

```
          Bad:=True;
          end;
        end;
    end;
{
                Add '+' to F
}
  F[L+1]:='+';
  Inc(L);
{
                Check for a leading '+'
}
  If Bad=False then
    begin
    writeln('Checking formula format');
    If F[1]='+' then
       begin
       Bad:=True;
       Writeln('Formula should start with a variable, not "+"');
       end;
    end;
{
                Determine number of constants to evaluate
                Get formulas for [A] and [Y] matrices
}
  If Bad=False then
    begin
    writeln('Counting number of constants in formula');
    iL:=1;
    iCte:=0;
    S:='';
    While iL<=L do
       begin
       If F[iL]='+' then
         begin
         Inc(iCte);
         If Lterm<Length(S) then
           begin
           write('Term ',iCte,' (',S,') has over ',Lterm,' characters');
           Bad:=True;
           end;
         Ys[iCte]:=S;
         S:='';
         end
       else
         If F[iL]<>' ' then S:=S+F[iL];  { ignore spaces }
       Inc(iL);
       end;
    CteQtt:=iCte;
    If Nmax<CteQtt then
       begin
       writeln('Too many terms in formula (Max=',Nmax,'): ',CteQtt);
```

```
        Bad:=True;
        end;
      end;
{
                Look for same terms
}
   If Bad=False then
      begin
      writeln('Looking for identical terms in formula');
      For iC:=1 to CteQtt-1 do
        For jC:=iC+1 to CteQtt do
          If Ys[iC]=Ys[jC] then
             begin
             Bad:=True;
             Writeln('Term ',Ys[iC],' is found more than once');
             end;
      end;
{
              Verify for sufficient data quantity
}
   If Bad=False then
      begin
      writeln('Checking data quantity');
      If DataQtt<CteQtt then
        begin
        Writeln('Insufficient data quantity (',DataQtt,') for the ',
                CteQtt,' terms');
        Bad:=True;
        end;
      end;
{
                Initialize matrices
}
   If Bad=False then
      begin
      For iC:=1 to CteQtt do
        begin
        For jC:=1 to CteQtt do
          A^[iC,jC]:=0.0;
        Y[iC]:=0.0;
        end;
      end;
{
                Summation to matrix:
                example: F = C1.a + C2.ab + C3.b + C4.1
                then: ¦ a^2    a^2.b   a.b   a   ¦       ¦ y.a  ¦
                      ¦ a^2.b  a^2.b^2 a.b^2 a.b ¦ [C] = ¦ y.a.b ¦
                      ¦ a.b    a.b^2   b^2   b   ¦       ¦ y.b  ¦
                      ¦ a      a.b     b     1   ¦       ¦ y.1  ¦
                                [A]                  [C] =  [Y]
                first line = F without constants multiplied by a , a
                             F without constants multiplied by ab , ab
```

```
                                F without constants multiplied by b , b
                                F without constants multiplied by 1 , 1
}
  If Bad=False then
    begin
    Reset(F_Data);
    write('Establishing system of equations for constants ');
    Xscr:=WhereX;
    Yscr:=WhereY;
    iData:=1;
    While (iData<=DataQtt) and (Bad=False) do
       begin
       GotoXY(Xscr,Yscr);
       Write(DataQtt-iData+1,' ');
       Repeat
          Readln(F_Data,DataStr);
          until (DataStr<>'');
{
                Transform line into a vector:
                  d[1]=Y
                  d[2...27]=A...Z
}
       LineToVect(DataStr,d,Bad);
{
                Sum to matrix [A]
}
       If Bad=False then
          begin
          For iC:=1 to CteQtt do
             EvalString(Ys[iC],d,Yr[iC],Bad);
          For iC:=1 to CteQtt do
             begin
             Y[iC]:=Y[iC]+d[1]*Yr[iC];
             For jC:=iC to CteQtt do
                A^[iC,jC]:=A^[iC,jC]+Yr[jC]*Yr[iC];
             end;
          end;
       Inc(iData);
       end;
    GotoXY(Xscr,Yscr);
    Writeln(' ');
    end;
{
                Copy A(i,j) to A(j,i)
}
  If Bad=False then
    begin
    For iC:=1 to CteQtt do
       For jC:=iC+1 to CteQtt do
          A^[jC,iC]:=A^[iC,jC];
{
                Solve
```

```
}
     Write('Solving a ',CteQtt,'x',CteQtt,' system of equations');
     GE_SCP(A,Y,CteQtt,C,Error_Type);
     Writeln;
     If Error_Type<>0 then
        begin
        writeln('Error during gaussian elimination');
        Bad:=True;
        end
     else
        begin
        ClrScr;
        iL:=1;
        TextColor(LightRed);
        write('F');
        TextColor(LightGreen);
        Write('=');
        For iC:=1 to CteQtt do
           begin
           write('(',Rword(C[iC],8),')');
           TextColor(LightRed);
           write(Ys[iC]);
           TextColor(LightGreen);
           If iC<>CteQtt then write('+')
                         else writeln;
           end;
        end;
     end;
{
                Get sum(Y-F)^2
                and maximum absolute (Y-F)
}
   If     =False then
      b  n
      Sum2:=0.0;
      TextColor(LightRed);
      write(FormulaFileName);
      TextColor(LightGreen);
      Write(': Sum of (Yi-Fi)^2 = ');
      Xscr:=WhereX;
      Yscr:=WhereY;
      DifferenceMax:=-BigR;
      Reset(F_Data);
      For iData:=1 to DataQtt do
         begin
         GotoXY(Xscr,Yscr);
         Write(DataQtt-iData+1,' ');
         Repeat
            Readln(F_Data,DataStr);
            until (DataStr<>'');
{
                Transform line into a vector:
```

```
                    d[1]=Y
                    d[2...27]=A...Z
)
        LineToVect(DataStr,d,Bad);
{
                Evaluate function
}
        Fr:=0.0;
        For iC:=1 to CteQtt do
          begin
          EvalString(Ys[iC],d,Tr,Bad);
          Fr:=Fr+C[iC]*Tr;
          end;
        Difference:=abs(d[1]-Fr);
        If DifferenceMax<Difference then DifferenceMax:=Difference;
        Sum2:=Sum2+Sqr(Difference);
        end;
{
                write result
}
     GotoXY(Xscr,Yscr);
     TextColor(LightRed);
     write(Rword(Sum2,8));
     TextColor(LightGreen);
     write(', Max |Yi-Fi| = ');
     TextColor(LightRed);
     write(Rword(DifferenceMax,8));
     TextColor(LightGreen);
     StopTimer(1);
     If AlertTime<ReadTimer(1) then Alert;
     Wait;
     end;
{
                Wait on error message
}
  If Bad=True then Wait;
  Dispose(A);
end;
{-------------------------------------------------------------------------}
Var
  Long,i,L: integer;
  MemNeeded: LongInt;
  iData,DataQtt,VarQtt: Itype;
  C: Char;
  d: VectR_27;
  Action: String6;
  DataStr,PathName,DirChoice: String255;
  NewFormulaFileName,FormulaFileName,DataFileName: String255;
  Bad,DataDone,FormulaFile_ok,DataFileOpen,DataFile_ok: Boolean;
  F_Formula: FileC;
  F_Data: Text;
  F: VectC_Screen;
```

```
  DisplayLine,Xleft,Xright: Byte;
Begin
{
                Check memory space:
                   [A] in Regression(): Nmax.Nmax.Rtype
                   [m] in GE_SCP():     Nmax.Nmax.Rtype
                   [A] in GE_SCP():     Nmax.(Nmax+1).Rtype
                   [S] in GE_SCP():     Nmax.Rtype
                   [Nrow] in GE_SCP():  Nmax.Itype
}
  MemNeeded:=(2*Nmax*Nmax+Nmax*(Nmax+1)+Nmax)*SizeOf(Rtype)
           +Nmax*SizeOf(Itype);
  If MemAvail<MemNeeded then
    begin
    writeln('This program requires ',Trunc(MemNeeded/1024)+1,'k but only ',
           Trunc(MemAvail/1024),'k is free.');
    Wait;
    Halt;
    end;
  Message;
  DataDone:=False;
  DataFileOpen:=False;
  L:=0;
  F[1]:=' ';
  DataFileName:='NoName';
  FormulaFileName:='NoName';
  Repeat
    ClrScr;
    TextColor(LightGreen);
    Write('H- Help    Data:');
       TextColor(LightRed);
       write(DataFileName);
       TextColor(LightGreen);
       Write(' Formula: ');
       TextColor(LightRed);
       Writeln(FormulaFileName);
       TextColor(LightGreen);
    Writeln('I- Input file: Lotus (*.PRN) file');
    Writeln('V- View data');
    Writeln('N- Number of variables');
    Writeln('R- Retrieve formula');
    Writeln('G- Generate formula');
    Writeln('E- Edit formula');
    Writeln('S- Save formula');
    Writeln('D- Delete formula');
    Writeln('P- Perform regression');
    Writeln('Q- Quit');
    DisplayLine:=WhereY;
    Xright:=80;
    Xleft:=1;
    Long:=(Xright-Xleft-1)*(25-DisplayLine-1)-1;
    GotoXY(79,25);
```

```
      Get_Key(C,Action);
      C:=UpCase(C);
      Case C of
         'H': Help;
         'I': begin
              DataDone:=False;
              GotoXY(1,DisplayLine);
              write('Path name (Default=C:\Lotus\Files\): ');
              Readln(PathName);
              If PathName='' then PathName:='C:\Lotus\Files\';
{
                   Erase last '\' in path name
}
              If 0<Length(PathName) then
                 If PathName[Length(PathName)]='\' then
                    Delete(PathName,Length(PathName),1);
              If DataFileOpen=True then
                 begin
                 Close(F_Data);
                 DataFileOpen:=False;
                 end;
              Pick_File(PathName,'*.PRN',AnyFile,False,
                        DataFileName,DataFile_ok);
              If DataFile_ok then
                 begin
                 Assign(F_data,PathName+'\'+DataFileName);
                 Reset(F_Data);
                 DataFileOpen:=True;
                 end;
              end;
         'V': begin
              If DataFileOpen then
                 begin
                 If DataDone=False then
                    begin
                    Get_Data(F_Data,VarQtt,DataQtt,DisplayLine);
                    DataDone:=True;
                    end;
                 ClrScr;
                 Reset(F_Data);
                 For iData:=1 to DataQtt do
                    begin
                    Repeat
                       Readln(F_Data,DataStr);
                       until (DataStr<>'');
                    LineToVect(DataStr,d,Bad);
                    If Bad=True then wait;
                    Write(iData:3);
                    For i:=1 to VarQtt+1 do write(' ',Rfield(d[i],8));
                    If Round(iData/25)*25=iData then
                       begin
                       wait;
```

```
            ClrScr;
            end
         else writeln;
         end;
      If Round(DataQtt/25)*25<>DataQtt then wait;
      end
   else
      begin
      GotoXY(1,DisplayLine);
      writeln('Data file not open');
      Wait;
      end;
   end;
'N': begin
   If DataFileOpen then
      begin
      If DataDone=False then
         begin
         Get_Data(F_Data,VarQtt,DataQtt,DisplayLine);
         DataDone:=True;
         end;
      GotoXY(1,DisplayLine);
      writeln('File ',DataFileName);
      writeln('   ',VarQtt,' variable(s)');
      writeln('   ',DataQtt,' data line(s)');
      Wait;
      end
   else
      begin
      GotoXY(1,DisplayLine);
      writeln('Data file not open');
      Wait;
      end;
   end;
'R': begin
   Pick_File('','*.F',AnyFile,False,FormulaFileName,FormulaFile_ok);
   If FormulaFile_ok=True then
      begin
      L:=0;
      F[1]:=' ';
      Assign(F_Formula,FormulaFileName);
      Reset(F_Formula);
      While EOF(F_Formula)=False do
         begin
         Inc(L);
         Read(F_Formula,F[L]);
         end;
      Close(F_Formula);
      end;
   end;
'G': begin
   If (DataFileOpen=True) then
```

```
        begin
        If DataDone=False then
          begin
          Get_Data(F_Data,VarQtt,DataQtt,DisplayLine);
          DataDone:=True;
          end;
        GenerateFormula(VarQtt,F,L,Long);
        end
      else
        begin
        GotoXY(1,DisplayLine);
        writeln('Data file not open');
        Wait;
        end;
      end;
  'E': begin
      gotoXY(35,DisplayLine-1);
      write(FormulaFileName);
      get_Paragraph(Long,DisplayLine,Xleft,Xright,LightRed,False,F,L);
      end;
  'S': begin
      If 0<L then
        begin
        GotoXY(1,DisplayLine);
        Write('Formula file name (Default = ',FormulaFileName,'): ');
        Readln(NewFormulaFileName);
        If NewFormulaFileName<>'' then
          FormulaFileName:=NewFormulaFileName+'.F';
        Assign(F_Formula,FormulaFileName);
        Rewrite(F_Formula);
        For i:=1 to L do
          write(F_Formula,F[i]);
        Close(F_Formula);
        end
      else
        begin
        GotoXY(1,DisplayLine);
        writeln('No formula entered');
        Wait;
        end;
      end;
  'D': begin
      Pick_File('','*.F',AnyFile,False,FormulaFileName,FormulaFile_ok);
      If FormulaFile_ok=True then
        begin
        Assign(F_Formula,FormulaFileName);
        Erase(F_Formula);
        end;
      end;
  'P': begin
      If (DataFileOpen=True) then
        begin
```

```
                ClrScr;
                If DataDone=False then
                   begin
                   Get_Data(F_Data,VarQtt,DataQtt,DisplayLine);
                   DataDone:=True;
                   end;
                Regression(F_Data,F,L,VarQtt,DataQtt,FormulaFileName);
                end
             else
                begin
                GotoXY(1,DisplayLine);
                writeln('Data file not open');
                Wait;
                end;
             end;
          'Q': ClrScr;
          end;
       Until C='Q';
    If DataFileOpen then Close(F_Data);
end.
```

RESULT.PAS

```
{-------------------------------------------------------------------
   PostProcessor
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 10000,0,76004}     (max memory used for drawing elements)
                       (10500 bytes in Heap needed for InitGraph)
                       (65504=2047*8*4 bytes in Heap needed for Connect.pas)
Uses
   Crt,Graph,Printer,Dos,Declare,Draw,WhatKey,Where,WaitKey,Nombre,Timer,
   Connect,Nne_Dim,Get,Linear,File_RW,VectMat3,Status;
{-------------------------------------------------------------------}
Procedure Display_Key(    ColorType,col,row: integer;
                       var Key_pos: MatrixByte_3x10x2;
                       var Key_W: MatrixS30_3x10;
                       Var StatusForce,StatusDispl,
                              StatusFiberStrain,StatusFiberStress,
                              StatusElemStrain,StatusElemStress,
                              StatusGlobalStrain,StatusGlobalStress: String12);
{
   Input:
      ColorType: 1 for normal (available stresses highlighted)
                 2 to indicate column position
                 3 to indicate current position
      Key_pos[col,row,1..2]: X and Y of position of keyword
      Key_W[col,row]: Keyword
   Output:
      Write appropriate keyword at it's location
}
begin
   GotoXY(Key_pos[col,row,1],Key_pos[col,row,2]);
   Case ColorType of
      1: begin
         If Col=2 then
            begin
            If (Row=1) and (StatusDispl='Done') then TextColor(LightRed);
            If (Row=2) and (StatusForce='Done') then TextColor(LightRed);
            If (Row=3) and (StatusFiberStrain='Done') then TextColor(LightRed);
            If (Row=4) and (StatusFiberStress='Done') then TextColor(LightRed);
            If (Row=5) and (StatusElemStrain='Done') then TextColor(LightRed);
            If (Row=6) and (StatusElemStress='Done') then TextColor(LightRed);
            If (Row=7) and (StatusGlobalStrain='Done') then TextColor(LightRed);
            If (Row=8) and (StatusGlobalStress='Done') then TextColor(LightRed);
            end;
         end;
      2: begin
         TextColor(Black);
         TextBackGround(LightRed);
         end;
      3: begin
```

```
            TextColor(Black);
            TextBackGround(White);
            end;
         end;
      write(Key_W[col,row]);
      TextColor(LightGreen);
      TextBackground(Black);
end;
{-------------------------------------------------------------}
Procedure HelpAsk;
{
   Output: Help on Ask procedure
}
begin
   ClrScr;
   writeln('You can use the <arrow> <Home> <End> <PgUp> <PgDn> keys ');
   writeln('   to move and select the output you want.');
   writeln('   If you press a <letter> key the cursor will point to the');
   writeln('   next item that starts with this letter.');
   writeln('When selected, press the <F10> key.');
   writeln('<F2> allows you to return to the point where you select');
   writeln('   all or part of the geometry you want to see and the viewpoint.');
   writeln('Fiber strain: strain along the material direction.');
   writeln('Element strain: strain along the element coordinate direction.');
   writeln('Global strain: strain along the global coordinate direction.');
   writeln('<Esc> will return you to the main menu.');
   wait;
end;
{-------------------------------------------------------------}
Procedure Ask(    Dim: IType;
             var StatusForce,StatusDispl,
                    StatusFiberStrain,StatusFiberStress,
                    StatusElemStrain,StatusElemStress,
                    StatusGlobalStrain,StatusGlobalStress: String12;
             var Faire: VectI_3;
             var Title: String80;
             var Finish: Boolean);
{
   Input:
      Dim: Dimension # of problem
      StatusForce,StatusDispl,
         StatusFiberStrain,StatusFiberStress,
         StatusElemStrain,StatusElemStress,
         StatusGlobalStrain,StatusGlobalStress: "Done" if calculated
   Output:
      Faire[3]: Faire[1] = -2 for Help
                         = -1 when Exit
                         = 0 for redefining SELECT parameters
                             (nodes chosen, Phi, Theta)
                         =1 for Screen printout
                         =2 for Printer printout
                         =3 for Extremes
```

```
                    =4 for Graph
                    =5 for Lotus 123
                Faire[2] = 1 for Displacement   If StatusDispl='Done'
                    = 2 for Force          If StatusForce='Done'
                    = 3 for Fiber Strain   If StatusFiberStrain='Done'
                    = 4 for Fiber Stress   If StatusFiberStress='Done'
                    = 5 for Element Strain If StatusElemStrain='Done'
                    = 6 for Element Stress If StatusElemStress='Done'
                    = 7 for Global Strain  If StatusGlobalStrain='Done'
                    = 8 for Global Stress  If StatusGlobalStress='Done'
                If Dim=2
                   Faire[3] = 1 for direction X
                            = 2 for direction Y
                            = 3 for direction XY
                If Dim=3
                   Faire[3] = 1 for direction X
                            = 2 for direction Y
                            = 3 for direction Z
                            = 4 for direction XY
                            = 5 for direction XZ
                            = 6 for direction YZ
      Title: Title for graph
      Finish: =True if that's what we want
              =False otherwise
}
var
   Old_col,New_col,row,col: IType;
   MaxPos,New_row,Old_row: array[1..3] of IType;
   Key_pos: MatrixByte_3x10x2;
   Key_W: MatrixS30_3x10;
   C: char;
   Action: String6;
   GoodPos,Found: boolean;
   MessageX,MessageY: Byte;
begin
   MessageX:=1;
   MessageY:=20;
{
                Display choices
}
   ClrScr;
{
                Get key positions and keywords
}
   Key_pos[1,1,1]:=1;   Key_pos[1,1,2]:=4;  Key_W[1,1]:='Screen listing';
   Key_pos[1,2,1]:=1;   Key_pos[1,2,2]:=5;  Key_W[1,2]:='Printer listing';
   Key_pos[1,3,1]:=1;   Key_pos[1,3,2]:=6;  Key_W[1,3]:='Extremes';
   Key_pos[1,4,1]:=1;   Key_pos[1,4,2]:=7;  Key_W[1,4]:='Graph';
   Key_pos[1,5,1]:=1;   Key_pos[1,5,2]:=8;  Key_W[1,5]:='Lotus 123';
   MaxPos[1]:=5;
   Key_pos[2,1,1]:=20;  Key_pos[2,1,2]:=4;  Key_W[2,1]:='Displacement';
   Key_pos[2,2,1]:=20;  Key_pos[2,2,2]:=5;  Key_W[2,2]:='Force';
```

```
   Key_pos[2,3,1]:=20;  Key_pos[2,3,2]:=6;  Key_W[2,3]:='Fiber strain';
   Key_pos[2,4,1]:=20;  Key_pos[2,4,2]:=7;  Key_W[2,4]:='Fiber stress';
   Key_pos[2,5,1]:=20;  Key_pos[2,5,2]:=8;  Key_W[2,5]:='Element strain';
   Key_pos[2,6,1]:=20;  Key_pos[2,6,2]:=9;  Key_W[2,6]:='Element stress';
   Key_pos[2,7,1]:=20;  Key_pos[2,7,2]:=10; Key_W[2,7]:='Global strain';
   Key_pos[2,8,1]:=20;  Key_pos[2,8,2]:=11; Key_W[2,8]:='Global stress';
   MaxPos[2]:=8;
   If Dim=2 then
      begin
      MaxPos[3]:=3;
      Key_pos[3,1,1]:=38; Key_pos[3,1,2]:=4; Key_W[3,1]:='X direction';
      Key_pos[3,2,1]:=38; Key_pos[3,2,2]:=5; Key_W[3,2]:='Y direction';
      Key_pos[3,3,1]:=38; Key_pos[3,3,2]:=6; Key_W[3,3]:='XY direction';
      end;
   If Dim=3 then
      begin
      MaxPos[3]:=6;
      Key_pos[3,1,1]:=38; Key_pos[3,1,2]:=4; Key_W[3,1]:='X direction';
      Key_pos[3,2,1]:=38; Key_pos[3,2,2]:=5; Key_W[3,2]:='Y direction';
      Key_pos[3,3,1]:=38; Key_pos[3,3,2]:=6; Key_W[3,3]:='Z direction';
      Key_pos[3,4,1]:=38; Key_pos[3,4,2]:=7; Key_W[3,4]:='XY direction';
      Key_pos[3,5,1]:=38; Key_pos[3,5,2]:=8; Key_W[3,5]:='XZ direction';
      Key_pos[3,6,1]:=38; Key_pos[3,6,2]:=9; Key_W[3,6]:='YZ direction';
      end;
{
                  Display choices
}
   gotoXY(1,1);
   TextColor(LightRed);
   write('<F1>');
   TextColor(LightGreen);
   writeln(': Help');
   TextColor(LightRed);
   write('<F2>');
   TextColor(LightGreen);
   writeln(': Redefine node selection, View axis');
   TextColor(LightRed);
   write('<F10>');
   TextColor(LightGreen);
   write(': accept the following request:');
   gotoXY(1,Key_pos[1,MaxPos[1],2]+1);
   TextColor(LightRed);
   write('<Esc>');
   TextColor(LightGreen);
   write(': exit');
{
                  Highlight Keywords
}
   New_row[1]:=1;
   New_row[2]:=1;
   New_row[3]:=1;
   For Col:=1 to 3 do
```

```
    begin
    For Row:=1 to MaxPos[Col] do
       Display_Key(1,col,row,Key_pos,Key_W,
       StatusForce,StatusDispl,
       StatusFiberStrain,StatusFiberStress,
       StatusElemStrain,StatusElemStress,
       StatusGlobalStrain,StatusGlobalStress);
    Display_Key(2,col,New_row[col],Key_pos,Key_W,
       StatusForce,StatusDispl,
       StatusFiberStrain,StatusFiberStress,
       StatusElemStrain,StatusElemStress,
       StatusGlobalStrain,StatusGlobalStress);
    end;
  New_Col:=1;
  Display_Key(3,New_Col,New_row[New_col],Key_pos,Key_W,
       StatusForce,StatusDispl,
       StatusFiberStrain,StatusFiberStress,
       StatusElemStrain,StatusElemStress,
       StatusGlobalStrain,StatusGlobalStress);
{
                Modify choice
}
  Action:='NUL';
  While (Action<>'ESC')and(Action<>'F1')and(Action<>'F2')and(Action<>'F10') do
    begin
    Old_col:=New_Col;
    For col:=1 to 3 do
       Old_row[col]:=New_row[col];
    Get_Key(C,Action);
    C:=UpCase(C);
{
                Move
}
    If C<>chr(0) then
       begin
       Found:=False;
       row:=Old_row[Old_Col]+1;
       While (row<=MaxPos[Old_Col]) and (Found=False) do
          begin
          If Key_W[Old_col,row][1]=C then
             begin
             New_row[Old_col]:=row;
             found:=True;
             end;
          row:=row+1;
          end;
       row:=1;
       While (row<=Old_row[Old_Col]) and (Found=False) do
          begin
          If Key_W[Old_col,row][1]=C then
             begin
             New_row[Old_col]:=row;
```

```
                found:=True;
                end;
             row:=row+1;
             end;
          end;
       If (Action='HOME') then
          begin
          New_col:=1;
          New_row[1]:=1;
          end;
       If (Action='END') then
          begin
          New_col:=3;
          New_row[3]:=MaxPos[3];
          end;
       If (Action='PGUP') then
          New_row[old_col]:=1;
       If (Action='PGDN') then
          New_row[old_col]:=MaxPos[old_col];
       If (Action='RIGHT') then New_col:=Old_col+1;
       If (Action='LEFT') then New_col:=Old_col-1;
       If (Action='UP') then New_row[Old_col]:=New_row[Old_col]-1;
       If (Action='DOWN') then New_row[Old_col]:=New_row[Old_col]+1;
{
                Check limits
}
       If New_Col<1 then New_col:=1;
       If 3<New_Col then New_col:=3;
       If new_row[new_col]<1 then
          New_row[new_col]:=1;
       If MaxPos[new_col]<New_row[new_col] then
          New_row[new_col]:=MaxPos[new_col];
{
                Verify position before leaving menu
}
       If (Action='F10') then
          begin
          GoodPos:=True;
{
                Avoid shear directions with Force and Displacement
}
          If New_row[2] in [1,2] then        { displacement - force }
             begin
             If Dim=2 then
                If 2<new_Row[3] then
                   begin
                   New_row[3]:=2;
                   GotoXY(MessageX,MessageY);
                   write('No shear direction with ',Key_W[2,New_row[2]]);
                   GoodPos:=False;
                   end;
             If Dim=3 then
```

```
            If 3<new_Row[3] then
               begin
               New_row[3]:=3;
               GotoXY(MessageX,MessageY);
               write('No shear direction with ',Key_W[2,New_row[2]]);
               GoodPos:=False;
               end;
         end;
{
               Verify with status
}
      If (StatusDispl<>'Done') and (New_Row[2]=1) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
         end;
      If (StatusForce<>'Done') and (New_Row[2]=2) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
         end;
      If (StatusFiberStrain<>'Done') and (New_Row[2]=3) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
         end;
      If (StatusFiberStress<>'Done') and (New_Row[2]=4) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
         end;
      If (StatusElemStrain<>'Done') and (New_Row[2]=5) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
         end;
      If (StatusElemStress<>'Done') and (New_Row[2]=6) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
         end;
      If (StatusGlobalStrain<>'Done') and (New_Row[2]=7) then
         begin
         GotoXY(MessageX,MessageY);
         write(Key_W[2,New_row[2]],' has not been calculated');
         GoodPos:=False;
```

```
          end;
        If (StatusGlobalStress<>'Done') and (New_Row[2]=8) then
          begin
          GotoXY(MessageX,MessageY);
          write(Key_W[2,New_row[2]],' has not been calculated');
          GoodPos:=False;
          end;
{
                Wait, erase error message, prevent exit
}
        If GoodPos=False then
          begin
          Wait;
          GotoXY(1,WhereY);
          ClrEOL;
          Action:='NUL';   (no exit)
          end;
        end;
{
                Display new position
}
      For col:=1 to 3 do
        begin
        Display_Key(1,Col,Old_row[col],Key_pos,Key_W,
        StatusForce,StatusDispl,
        StatusFiberStrain,StatusFiberStress,
        StatusElemStrain,StatusElemStress,
        StatusGlobalStrain,StatusGlobalStress);
        Display_Key(2,Col,New_row[col],Key_pos,Key_W,
        StatusForce,StatusDispl,
        StatusFiberStrain,StatusFiberStress,
        StatusElemStrain,StatusElemStress,
        StatusGlobalStrain,StatusGlobalStress);
        end;
      Display_Key(3,New_Col,New_row[New_col],Key_pos,Key_W,
        StatusForce,StatusDispl,
        StatusFiberStrain,StatusFiberStress,
        StatusElemStrain,StatusElemStress,
        StatusGlobalStrain,StatusGlobalStress);
      end;
{
                Establish Faire
}
   ClrScr;
   If Action='ESC' then
     begin
     Finish:=True;
     Faire[1]:=-1;
     end;
   If Action='F1' then Faire[1]:=-2;
   If Action='F2' then Faire[1]:=0;
   If Action='F10' then
```

```
      For col:=1 to 3 do
        Faire[col]:=New_row[col];
   Title:=Key_W[2,new_row[2]];
   Title:=Title+' in '+Key_W[3,new_row[3]];
end;
{-------------------------------------------------------------------}
Procedure Plane(    Nnt: IType;
                var F_Connect: FileI;
                var F_xyz3D: FileR);
{
   Input:
      Nnt: Number of nodes in total
      F_Connect: File Giving Connection for each node
                 Connected to nodes Connect[1..Con], =0 for last connection
                 Connect[1]=-1 if node not selected
      F_xyz3D[1..Dim*Nnt]: File with X Y Z of nodes
   Output:
      F_xyz3D[1..Dim*Nnt]: File with X Y Z of nodes
      Where X Y of nodes are a projection on a regression plane
         from the selected nodes
}
var
   i,j,iNode,NodeSelected,Dim3: IType;
   Connect: VectI_Con;
   A,B,C,D,t,L,l1,l2,l3,m1,m2,m3,n1,n2,n3: Rtype;
   Coef,iC: MatrixR_3x3;
   Constant,meanXYZ: Array[1..3] of Rtype;
   Nxyz,xyz1,xyz: VectR_D;
   Same: array[1..3] of boolean;
   FirstNode: Boolean;
   Xs,Ys: Byte;
begin
   Dim3:=3;
   write('Fitting plane to chosen nodes ');
   Xs:=WhereX;
   Ys:=WhereY;
{
        least square for a plane: Z = aX + bY + d  for N points
           a.Sum(Xi.Xi) + b.Sum(XiYi) + d.Sum(Xi) = Sum(ZiXi)
           a.Sum(Xi.Yi) + b.Sum(YiYi) + d.Sum(Yi) = Sum(ZiYi)
           a.Sum(Xi)    + b.Sum(Yi)   + d.N       = Sum(Zi)
}
   For i:=1 to 3 do
     begin
     For j:=1 to 3 do
        Coef[i,j]:=0.0;
     Constant[i]:=0.0;
     Same[i]:=True;
     end;
   NodeSelected:=0;
   FirstNode:=True;
   For iNode:=1 to Nnt do
```

```
      begin
      GotoXY(Xs,Ys);
      write(3*Nnt-iNode+1,'  ');
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then   (nodes selected)
         begin
         D_R_Read(F_xyz3D,iNode,Dim3,xyz);
         If FirstNode=True then
            begin
            For i:=1 to Dim3 do
               xyz1[i]:=xyz[i];
            FirstNode:=False;
            end
         else
            begin
            For i:=1 to Dim3 do
               If xyz1[i]<>xyz[i] then
                  same[i]:=False;
            end;
         Coef[1,1]:=Coef[1,1]+XYZ[1]*XYZ[1];
         Coef[1,2]:=Coef[1,2]+XYZ[1]*XYZ[2];
         Coef[1,3]:=Coef[1,3]+XYZ[1];
         Coef[2,2]:=Coef[2,2]+XYZ[2]*XYZ[2];
         Coef[2,3]:=Coef[2,3]+XYZ[2];
         Constant[1]:=Constant[1]+XYZ[3]*XYZ[1];
         Constant[2]:=Constant[2]+XYZ[3]*XYZ[2];
         Constant[3]:=Constant[3]+XYZ[3];
         Inc(NodeSelected);
         end;
      end;
   Coef[2,1]:=Coef[1,2];
   Coef[3,1]:=Coef[1,3];
   Coef[3,2]:=Coef[2,3];
   Coef[3,3]:=NodeSelected;
{
                  If one of the coordinates is fixed,
                  no need to solve equations
}
   For i:=1 to Dim3 do
      begin
      If Same[i]=True then
         begin
         A:=0;
         B:=0;
         C:=0;
         Case i of
            1: A:=1;
            2: B:=1;
            3: C:=1;
            end;
         D:=-(A*xyz1[1]+B*xyz1[2]+C*xyz1[3]);
         end;
```

```
    end;
{
                Solve by first inverting matrix
}
  If (Same[1]=False)and(Same[2]=False)and(Same[3]=False) then
    begin
    Invert3(Coef,iC);
    A:=iC[1,1]*Constant[1]+iC[1,2]*Constant[2]+iC[1,3]*Constant[3];
    B:=iC[2,1]*Constant[1]+iC[2,2]*Constant[2]+iC[2,3]*Constant[3];
    D:=iC[3,1]*Constant[1]+iC[3,2]*Constant[2]+iC[3,3]*Constant[3];
{
                Keep plane constants as aX+bY+cZ+D=0
}
    C:=-1.0;
    A:=A;
    B:=B;
    D:=D;
    end;
{
                Find Mean for the points
                Projection of coordinates onto plane.
                substituting line equation into plane:
                  A(At+X1)+B(Bt+Y1)+C(Ct+Z1)+D=0
                we get:
                  t = (A.X1+B.Y1+C.Z1+D) / (A.A+B.B+C.C)
                  Xp = A.t+X1
                  Yp = B.t+Y1
                  Zp = C.t+Z1
}
  For i:=1 to Dim3 do                          { init mean sum }
    meanXYZ[i]:=0.0;
  L:=A*A+B*B+C*C;
  For iNode:=1 to Nnt do
    begin
    GotoXY(Xs,Ys);
    write(2*Nnt-iNode+1,'  ');
    Con_I_Read(F_Connect,iNode,Connect);
    If Connect[1]<>-1 then   {nodes selected}
      begin
      D_R_Read(F_xyz3D,iNode,Dim3,xyz);
      t:=(A*XYZ[1]+B*XYZ[2]+C*XYZ[3]+D)/L;
      For i:=1 to Dim3 do                          { sum for mean }
        meanXYZ[i]:=meanXYZ[i]+XYZ[i];
      XYZ[1]:=A*t+XYZ[1];
      XYZ[2]:=B*t+XYZ[2];
      XYZ[3]:=C*t+XYZ[3];
      D_R_write(F_xyz3D,iNode,Dim3,xyz);
      end;
    end;
  For i:=1 to Dim3 do                          { get mean }
    meanXYZ[i]:=meanXYZ[i]/NodeSelected;
{
```

```
                Projection of mean onto plane
}
  L:=A*A+B*B+C*C;
  t:=(A*maanXYZ[1]+B*meanXYZ[2]+C*meanXYZ[3]+D)/L;
  meanXYZ[1]:=A*t+meanXYZ[1];
  meanXYZ[2]:=B*t+meanXYZ[2];
  meanXYZ[3]:=C*t+meanXYZ[3];
{
                Find Z' axis, Normal to plane.
                The normal to plane Ax+By+Cz+D is paralel to
                the vector (A,B,C).
}
  L:=sqrt(A*A+B*B+C*C);
  l3:=A/L;
  m3:=B/L;
  n3:=C/L;
{
                If Vector Z(0,0,1) and Z'(l3,m3,n3) makes an angle
                bigger than 90 degrees, change sign of Z' vector
                => cos(angle)<0
                In this manner the normal to the plane will be
                directed towards Z positive
}
  If n3<0 then
    begin
    l3:=-l3;
    m3:=-m3;
    n3:=-n3;
    end;
{
                Find X' axis, Normal to ZZ' plane -> ZxZ'
                    i   j   k
                Z   0   0   1
                Z'  l3  m3  n3
}
  If (abs(l3)<1E-5) and (abs(m3)<1E-5) then   { Z' parallel to Z }
    begin
    l1:=1;          { use X' = X }
    m1:=0;
    n1:=0;
    end
  else
    begin
    l1:=-m3;
    m1:=-(-l3);
    n1:=0.0;
    L:=sqrt(l1*l1+m1*m1+n1*n1);
    l1:=l1/L;
    m1:=m1/L;
    n1:=n1/L;
    end;
{
```

```
                Find Y' axis, Normal to Z'X' plane -> Z'xX'
                    i  j  k
                Z'  l3 m3 n3
                X'  l1 m1 n1
)
   l2:=m3*n1-m1*n3;
   m2:=-(l3*n1-l1*n3);
   n2:=l3*m1-l1*m3;
   L:=sqrt(l2*l2+m2*m2+n2*n2);
   l2:=l2/L;
   m2:=m2/L;
   n2:=n2/L;
{
                Transform projected points into local coordinate
                system X' Y' Z'
)
   For iNode:=1 to Nnt do
      begin
      GotoXY(Xs,Ys);
      write(Nnt-iNode+1,'  ');
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then   {nodes selected}
         begin
         D_R_Read(F_xyz3D,iNode,Dim3,xyz);
         Nxyz[1]:=l1*(XYZ[1]-meanXYZ[1])
                 +m1*(XYZ[2]-meanXYZ[2])
                 +n1*(XYZ[3]-meanXYZ[3]);
         Nxyz[2]:=l2*(XYZ[1]-meanXYZ[1])
                 +m2*(XYZ[2]-meanXYZ[2])
                 +n2*(XYZ[3]-meanXYZ[3]);
         Nxyz[3]:=0.0;
         D_R_write(F_xyz3D,iNode,Dim3,Nxyz);
         end;
      end;
   gotoxy(Xs,Ys);
   writeln('  ');
end;
{-------------------------------------------------------------------}
Procedure Get_Value(    Faire2,Faire3,Node,Dim,NStress: Itype;
                  var F_Displ,F_Force,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                        F_Gstrain,F_Gstress: FileR;
                  var Value: Rtype);
{
   Input:
      Faire2 = 1 for Displacement
             = 2 for Force
             = 3 for Fiber Strain
             = 4 for Fiber Stress
             = 5 for Element Strain
             = 6 for Element Stress
             = 7 for Global Strain
             = 8 for Global Stress
```

```
      If Dim=2
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction XY
      If Dim=3
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction Z
                = 4 for direction XY
                = 5 for direction XZ
                = 6 for direction YZ
      Node: Node number considered
      Dim: Dimension # of problem
      Nstress: Number of stress components (3 in 2D, 6 in 3D)
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Fstrain: File with material direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Fstress: File with material direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Estrain: File with element direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Estress: File with element direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Gstrain: File with Global direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Gstress: File with Global direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
   Output:
      Value: Desired value according to Faire2 and Faire3
}
var
   Force,Displ: VectR_D;
begin
   Case Faire2 of
      1: begin
         D_R_Read(F_Displ,Node,Dim,Displ);
         Value:=Displ[Faire3];
         end;
      2: begin
         D_R_Read(F_Force,Node,Dim,Force);
         Value:=Force[Faire3];
         end;
      3: begin
         Seek(F_Fstrain,Lin(Node,Faire3,Nstress)-1);
         Read(F_Fstrain,Value);
         end;
      4: begin
         Seek(F_Fstress,Lin(Node,Faire3,Nstress)-1);
         Read(F_Fstress,Value);
```

```
         end;
      5: begin
         Seek(F_Estrain,Lin(Node,Faire3,Nstress)-1);
         Read(F_Estrain,Value);
         end;
      6: begin
         Seek(F_Estress,Lin(Node,Faire3,Nstress)-1);
         Read(F_Estress,Value);
         end;
      7: begin
         Seek(F_Gstrain,Lin(Node,Faire3,Nstress)-1);
         Read(F_Gstrain,Value);
         end;
      8: begin
         Seek(F_Gstress,Lin(Node,Faire3,Nstress)-1);
         Read(F_Gstress,Value);
         end;
      end;
end;
{-------------------------------------------------------------}
Procedure New_Z(    Faire2,Faire3,Nnt,Dim: IType;
                var F_Connect: FileI;
                var F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                       F_Gstrain,F_Gstress,F_Force,F_Displ: FileR;
                var F_xyz3D: FileR;
                var Zscale: Rtype);
{
   Input:
      Faire2 = 1 for Displacement
             = 2 for Force
             = 3 for Fiber Strain
             = 4 for Fiber Stress
             = 5 for Element Strain
             = 6 for Element Stress
             = 7 for Global Strain
             = 8 for Global Stress
      If Dim=2
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction XY
      If Dim=3
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction Z
                = 4 for direction XY
                = 5 for direction XZ
                = 6 for direction YZ
      Nnt: Number of nodes in geometry
      Dim: Dimension # of problem
      F_Connect: File Giving Connection for each node
                 Connected to nodes Connect[1..Con], =0 for last connection
                 Connect[1]=-1 if node not selected
```

```
      F_xyz3D[1..Dim*Nnt]: File with X Y Z of nodes
         X Y of nodes projected on a regression plane
         from the selected nodes
      F_Fstrain: File with material direction Strain[1..N_Stress(Dim).Nnt]:
               e1 e2 e3 ...e23  for each node
      F_Fstress: File with material direction Stress[1..N_Stress(Dim).Nnt]:
               s1 s2 s3 ...s23  for each node
      F_Estrain: File with element direction Strain[1..N_Stress(Dim).Nnt]:
               e1 e2 e3 ...e23  for each node
      F_Estress: File with element direction Stress[1..N_Stress(Dim).Nnt]:
               s1 s2 s3 ...s23  for each node
      F_Gstrain: File with Global direction Strain[1..N_Stress(Dim).Nnt]:
               e1 e2 e3 ...e23  for each node
      F_Gstress: File with Global direction Stress[1..N_Stress(Dim).Nnt]:
               s1 s2 s3 ...s23  for each node
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
   Output:
      F_xyz3D: File with X Y of nodes projected on a regression plane
         from the selected nodes, but Z dimension added for desired
         strain or stress representation
      Zscale: Scale factor for Z
}
var
   Nstress,iNode,iD: IType;
   max,min: array[1..3] of Rtype;
   Value,Variation: Rtype;
   xyz: VectR_D;
   Connect: VectI_Con;
   Xs,Ys: Byte;
begin
   write('Adding selected parameter to coordinates ');
   Xs:=WhereX;
   Ys:=WhereY;
{
                  Initialize max, min for all directions
}
   For iD:=1 to 3 do
      begin
      max[iD]:=-1E30;
      min[iD]:=1E30;
      end;
{
                  Transfer Values as third dimension
}
   Nstress:=N_Stress(Dim);
   For iNode:=1 to Nnt do
      begin
      gotoXY(Xs,Ys);
      write(Nnt-iNode+1,' ');
```

```
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then   (nodes selected)
         begin
         Get_Value(Faire2,Faire3,iNode,Dim,NStress,
                   F_Displ,F_Force,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                   F_Gstrain,F_Gstress,Value);
         D_R_Read(F_xyz3D,iNode,3,xyz);
         XYZ[3]:=Value;
         D_R_write(F_xyz3D,iNode,3,xyz);
{
                   Find max,min
}
         For iD:=1 to 3 do
            begin
            If Max[iD]<XYZ[iD] then Max[iD]:=XYZ[iD];
            If XYZ[iD]<Min[iD] then Min[iD]:=XYZ[iD];
            end;
         end;
      end;
   gotoXY(Xs,Ys);
   writeln(' ');
{
                   Use smallest of the 2 first dimension variation
}
   If (Max[1]-Min[1])<(Max[2]-Min[2]) then
      Variation:=Max[1]-Min[1]
   else
      Variation:=Max[2]-Min[2];
   Variation:=Variation*0.5;  { Further reduce variation }
{
                   Get Z Scaling factor
}
   If abs(Min[3])<abs(Max[3]) then
      Zscale:=Variation/abs(Max[3])
   else
      Zscale:=Variation/abs(Min[3]);
end;
{-------------------------------------------------------------------}
Procedure Extreme(    Title: String80;
                      Faire2,Faire3,Nnt,Dim: IType;
                  var F_Connect: FileI;
                  var F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                        F_Gstrain,F_Gstress,F_Force,F_Displ: FileR;
                  var F_NewOld: FileI);
{
   Input:
      Title: Title
      Faire2 = 1 for Displacement
             = 2 for Force
             = 3 for Fiber Strain
             = 4 for Fiber Stress
             = 5 for Element Strain
```

```
            = 6 for Element Stress
            = 7 for Global Strain
            = 8 for Global Stress
      If Dim=2
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction XY
      If Dim=3
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction Z
                = 4 for direction XY
                = 5 for direction XZ
                = 6 for direction YZ
      Nnt: Number of nodes in geometry
      Dim: Dimension # of problem
      F_Connect: File Giving Connection for each node
                 Connected to nodes Connect[1..Con], =0 for last connection
                 Connect[1]=-1 if node not selected
      Force[1..3Nnt]: Force for nodes
      Displ[1..3Nnt]: Displacement for nodes
      F_NewOld: original node number given by user for each "compacted" node #
      F_Fstrain: File with material direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Fstress: File with material direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Estrain: File with element direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Estress: File with element direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Gstrain: File with Global direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Gstress: File with Global direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
         each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
         each node
   Output:
      Display extreme values of eS according to Faire2, Faire3
}
var
   CountGood,Old,Nstress,iNode,MaxNumber,MinNumber: IType;
   STD,SumSquare,SumValue,Average,Value,Max,Min: Rtype;
   Connect: VectI_Con;
   SignificantDigits,Xs,Ys: Byte;
begin
{
                  Find max min
}
   Max:=-1E30;
   Min:=1E30;
```

```
   MaxNumber:=0;
   MinNumber:=0;
   Nstress:=N_Stress(Dim);
   writeln(Title);
   writeln;
   Xs:=WhereX;
   Ys:=WhereY;
   SumValue:=0.0;
   SumSquare:=0.0;
   CountGood:=0;
   For iNode:=1 to Nnt do
      begin
      gotoXY(Xs,Ys);
      write(Nnt-iNode+1,' ');
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then   {nodes selected}
         begin
         I_Read(F_NewOld,iNode,Old);
         Get_Value(Faire2,Faire3,iNode,Dim,NStress,
                   F_Displ,F_Force,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                   F_Gstrain,F_Gstress,Value);
         Inc(CountGood);
         SumValue:=SumValue+Value;
         SumSquare:=SumSquare+Sqr(Value);
         If Max<Value then
            begin
            Max:=Value;
            MaxNumber:=Old;
            end;
         If Value<Min then
            begin
            Min:=Value;
            MinNumber:=Old;
            end;
         end;
      end;
{
                   Average = Sum(Xi)/n
                         n.Sum(Xi^2) - [Sum(Xi)]^2
                   s^2 = ------------------------
                                  n.(n-1)
                   Standard deviation = s
}
   Average:=SumValue/CountGood;
   STD:=sqrt((CountGood*SumSquare-sqr(SumValue))/(CountGood*(CountGood-1)));
   gotoXY(Xs,Ys);
   SignificantDigits:=2*SizeOf(RType)+3;
   writeln('Max = ',RField(Max,SignificantDigits),' at node ',MaxNumber);
   writeln('Min = ',RField(Min,SignificantDigits),' at node ',MinNumber);
   writeln('Average = ',RField(Average,SignificantDigits));
   writeln('Standard deviation = ',RField(STD,SignificantDigits));
   Wait;
```

```
end;
{-------------------------------------------------------------------}
Procedure Lister(    Title: String80;
                     OutDevice: Byte;
                     Faire2,Faire3,Nnt,Dim: IType;
                var F_Connect: FileI;
                var F_xyz,F_Fstrain,F_Fstress,F_Estrain,F_Estress,F_Gstrain,F_Gstress,
                       F_Force,F_Displ: FileR;
                var F_NewOld: FileI);
{
   Input:
      Title: Title
      OutDevice: =0 for Screen output
                 =1 for Printer output
                 =2 for Lotus 123 (*.PRN file)
      Faire2 = 1 for Displacement
             = 2 for Force
             = 3 for Fiber Strain
             = 4 for Fiber Stress
             = 5 for Element Strain
             = 6 for Element Stress
             = 7 for Global Strain
             = 8 for Global Stress
      If Dim=2
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction XY
      If Dim=3
         Faire3 = 1 for direction X
                = 2 for direction Y
                = 3 for direction Z
                = 4 for direction XY
                = 5 for direction XZ
                = 6 for direction YZ
      Nnt: Number of nodes in geometry
      Dim: Dimension # of problem
      F_Connect: File Giving Connection for each node
                 Connected to nodes Connect[1..Con], =0 for last connection
                 Connect[1]=-1 if node not selected
      F_xyz: File with coordinates of nodes
      F_Fstrain: File with material direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Fstress: File with material direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Estrain: File with element direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Estress: File with element direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Gstrain: File with Global direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Gstress: File with Global direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
```

```
      F_Force[1..Nnt x Dimension]: File with forces in Fx1,Fy1,Fz1 for
         each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
         each node
      F_NewOld: original node number given by user for each "compacted" node #
   Output:
      Display on screen, values of eS according to Faire2, Faire3
)
var
   i,Old,Nstress,Line,iNode,LineMax: IType;
   Xs,Ys,iD,TopMargin,BottomMargin: integer;
   Value: Rtype;
   xyz: VectR_D;
   MotOld,Fname: String255;
   F_Lotus: Text;
   Connect: VectI_Con;
begin
   If OutDevice=0 then          { Screen }
      begin
      LineMax:=CrtLine_max;
      TopMargin:=0;
      BottomMargin:=0;
      end;
   If OutDevice=1 then          { Printer }
      begin
      LineMax:=PrtLine_max;
      TopMargin:=2;
      BottomMargin:=2;
      end;
   If OutDevice=2 then          { Lotus File }
      begin
      LineMax:=2000000000;
      TopMargin:=0;
      BottomMargin:=0;
{
                  Ask File Name, open file
}
      Repeat
         ClrScr;
         gotoXY(1,10);
         writeln('Previous file with same name will be erased');
         write('Lotus file name (no extension): ');
         readln(Fname);
         until Pos('.',Fname)=0;
      Fname.=Fname+'.PRN';
      Assign(F_Lotus,C_Dir+Fname);
      Rewrite(F_Lotus);
      end;
{
                  Write title
}
   For i:=1 to TopMargin do
```

```
      Case OutDevice of
        0: writeln;
        1: writeln(Lst);
        2: {skip);
        end;
    Case OutDevice of
      0: writeln('Node, ',Title);
      1: writeln(Lst,'Node, ',Title);
      2: begin
        writeln(F_Lotus,'"'+'    Node '
                        +'       X '
                        +'       Y '
                        +'       Z '
                        +' '+Title+'"');
        write('Writting to file ');
        Xs:=WhereX;
        Ys:=WhereY;
        end;
      end;
    Line:=TopMargin+2;  { present line number }
{
                Write value at each node
}
    Nstress:=N_Stress(Dim);
    For iNode:=1 to Nnt do
      begin
      Con_I_Read(F_Connect,iNode,Connect);
      If Connect[1]<>-1 then   (nodes selected)
        begin
        Get_Value(Faire2,Faire3,iNode,Dim,NStress,
                  F_Displ,F_Force,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                  F_Gstrain,F_Gstress,Value);
        I_Read(F_NewOld,iNode,Old);
{
                  Get coodinates (if necessary)
}
        Case OutDevice of
          0: (nothing);
          1: (nothing);
          2: D_R_Read(F_xyz,iNode,Dim,xyz);
          end;
{
                  Create top margin
}
        While Line<=TopMargin do
          begin
          Case OutDevice of
            0: writeln;
            1: writeln(Lst);
            2: (skip);
            end;
          Line:=Line+1;
```

```
            end;
{
                  Write value
}
         str(Old:10,MotOld);
         Case OutDevice of
            0: write(MotOld,': ',RField(Value,2*SizeOf(Rtype)+3));
            1: write(Lst,MotOld,': ',RField(Value,2*SizeOf(Rtype)+3));
            2: begin
               gotoXY(Xs,Ys);
               write(Nnt-iNode+1,' ');
               write(F_Lotus,MotOld);
               For iD:=1 to Dim do
                  write(F_Lotus,' '+RField(xyz[iD],2*SizeOf(Rtype)+3));
               write(F_Lotus,' '+RField(Value,2*SizeOf(Rtype)+3));
               end;
            end;
{
                  Act upon end of page
}
         If 'Node<=Nnt then
            begin
            If (Line=LineMax-BottomMargin) then
               begin
               Case OutDevice of
                  0: begin
                     wait;
                     ClrScr;
                     end;
                  1: writeln(Lst,chr(12));
                  2: {skip};
                  end;
               Line:=1;
               end
            else
               begin
{
                   Move to next line
}
               Case OutDevice of
                  0: writeln;
                  1: writeln(Lst);
                  2: writeln(F_Lotus);
                  end;
               Line:=Line+1;
               end;
            end;
         end;
      end;
{
                  Final page flush
}
```

```
   Case OutDevice of
     0: begin
        wait;
        ClrScr;
        end;
     1: writeln(Lst,chr(12));
     2: {skip};
     end;
{
                Close device
}
   Case OutDevice of
     0: {skip};
     1: {skip};
     2: close(F_Lotus);
     end;
end;
{-------------------------------------------------------------------}
var
   NneMax,i,iD,Net,Nnt,Dim,Numbering,Draw_Axis,Draw_Elev: IType;
   Modified: Boolean;
   Faire: VectI_3;
   Zscale,Phi,Theta: Rtype;
   Finish: Boolean;
   Title: String80:
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
     StatusFiberStrain,StatusFiberStress,
     StatusElemStrain,StatusElemStress,
     StatusGlobalStrain,StatusGlobalStress: String12;
   F_xyz,F_xyz3D: FileR;
   F_Fstrain,F_Fstress,F_Estrain,F_Estress,F_Gstrain,F_Gstress,
     F_Force,F_Displ: FileR;
   F_Elem,F_NewOld,F_Connect: FileI;
begin
   ClrScr;
   Writeln('Loading geometry and results');
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   Get_I('Net.P',Net);
   Get_I('Nnt.P',Nnt);
   Get_I('Dim.P',Dim);
   Get_I('NneMax.P',NneMax);
{
                Assign file names
}
   Assign(F_xyz,C_Dir+'xyz.P');
   Assign(F_xyz3D,T_Dir+'xyz_3d.T');
```

```
Assign(F_Force,C_Dir+'Force.P');
Assign(F_Displ,C_Dir+'Displ.P');
Assign(F_Elem,C_Dir+'Elem.P');
Assign(F_NewOld,C_Dir+'NewToOld.P');
Assign(F_Connect,T_Dir+'Connect.T');  {Don't call it 'CON.dat'}
Reset(F_xyz);
Rewrite(F_xyz3D);
Reset(F_Force);
Reset(F_Displ);
Reset(F_Elem);
Reset(F_NewOld);
Rewrite(F_Connect);
Assign(F_Fstrain,C_Dir+'Fstrain.P');
Assign(F_Fstress,C_Dir+'Fstress.P');
Assign(F_Estrain,C_Dir+'Estrain.P');
Assign(F_Estress,C_Dir+'Estress.P');
Assign(F_Gstrain,C_Dir+'Gstrain.P');
Assign(F_Gstress,C_Dir+'Gstress.P');
If StatusFiberStrain ='Done' then Reset(F_Fstrain);
If StatusFiberStress ='Done' then Reset(F_Fstress);
If StatusElemStrain  ='Done' then Reset(F_Estrain);
If StatusElemStress  ='Done' then Reset(F_Estress);
If StatusGlobalStrain='Done' then Reset(F_Gstrain);
If StatusGlobalStress='Done' then Reset(F_Gstress);
Modified:=True;
xyz_3d(Nnt,F_xyz,F_xyz3D,Dim,Modified);
Finish:=False;
InitConnectHeapRAM;
Select('RESULT',Net,Nnt,NneMax,F_xyz3D,F_Elem,F_NewOld,F_Connect,
      Phi,Theta,Numbering,Draw_Axis,Draw_Elev);
While Finish=False do
  begin
  Ask(Dim,StatusForce,StatusDispl,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress,
      Faire,Title,Finish);
  Case Faire[1] of
    -2: HelpAsk;
     0: begin
        xyz_3d(Nnt,F_xyz,F_xyz3D,Dim,Modified);
        DisposeConnectHeapRAM;
        InitConnectHeapRAM;
        Select('RESULT',Net,Nnt,NneMax,F_xyz3D,F_Elem,F_NewOld,
              F_Connect,Phi,Theta,Numbering,
              Draw_Axis,Draw_Elev);
        end;
     1: Lister(Title,0,Faire[2],Faire[3],Nnt,Dim,F_Connect,
              F_xyz,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
              F_Gstrain,F_Gstress,F_Force,F_Displ,F_NewOld);
     2: Lister(Title,1,Faire[2],Faire[3],Nnt,Dim,F_Connect,
              F_xyz,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
```

```
                F_Gstrain,F_Gstress,F_Force,F_Displ,F_NewOld);
        3: begin
           Extreme(Title,Faire[2],Faire[3],Nnt,Dim,F_Connect,
                   F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                   F_Gstrain,F_Gstress,F_Force,F_Displ,F_NewOld);
           end;
        4: begin
           xyz_3d(Nnt,F_xyz,F_xyz3D,Dim,Modified);
           Plane(Nnt,F_Connect,F_xyz3D);
           New_Z(Faire[2],Faire[3],Nnt,Dim,F_Connect,
                 F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                 F_Gstrain,F_Gstress,F_Force,F_Displ,F_xyz3D,Zscale);
           Dessine(Title,Nnt,
                   Phi,Theta,Numbering,Draw_Axis,Draw_Elev,
                   F_xyz3D,Zscale,F_NewOld,F_Connect);
           Modified:=True;
           end;
        5: Lister(Title,2,Faire[2],Faire[3],Nnt,Dim,F_Connect,
                  F_xyz,F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                  F_Gstrain,F_Gstress,F_Force,F_Displ,F_NewOld);
        end;
      end;
   DisposeConnectHeapRAM;
   Close(F_xyz);
   Close(F_xyz3D);     Erase(F_xyz3D);
   If StatusFiberStrain ='Done' then Close(F_Fstrain);
   If StatusFiberStress ='Done' then Close(F_Fstress);
   If StatusElemStrain  ='Done' then Close(F_Estrain);
   If StatusElemStress  ='Done' then Close(F_Estress);
   If StatusGlobalStrain='Done' then Close(F_Gstrain);
   If StatusGlobalStress='Done' then Close(F_Gstress);
   Close(F_Force);
   Close(F_Displ);
   Close(F_Elem);
   Close(F_NewOld);
   Close(F_Connect);   Erase(F_Connect);
   ClrScr;
end.
```

SHUFFLE.PAS

```
{$N+}
Unit Shuffle;
Interface
Uses
   Crt,Declare,File_RW,Linear;
Procedure Reshape(var K: MatrixR_NneDxNneD;
                      Nne,Dim: IType);
Procedure Extract(var Ke: MatrixR_NneDxNneD;
                      Dim,Nne: IType;
                  var F_Displ: FileR;
                  var F_Fd: FileByte;
                      Elem: VectI_Nne2;
                  var R: VectR_NneD);
{-------------------------------------------------------------------}
Implementation
Function Pos_uvuv(    i,N,Dim: Byte):Byte;
{
   Input:
      i: position in vector uuvv[u1...uN v1...vN w1...wN]
         numbered from 1 to N.Dim
      N: number of nodes in vector uuvv = (vector length)/Dim
      Dim: number of dimensions: u = 1
                                 uv = 2
                                 uvw = 3
   Output:
      Pos_uvuv: new position in vector [u1 v1 w1 ... uN vN wN]
}
begin
{
                  Decrement i by 1
}
   Dec(i);
   Pos_uvuv := Dim*(i mod N)+(i div N)+1;
end;
{-------------------------------------------------------------------}
Procedure Reshape;
{
Procedure Reshape(var K: MatrixR_NneDxNneD;
                      Nne,Dim: Byte);
}
{
   Input:
      K: (Nne.Dim)x(Nne.Dim) stiff
      Nne: Number of nodes in this element
      Dim: Dimensions
   Output:
      K: Reshaped from | k1,1   ... k1,2n   | |u1|   |fx1|
                       | .                 | |. |   | . |
                       | kn,1   ... kn,2n   | |un| = |fxn|
                       | kn+1,1 ... kn+1,2n | |v1|   |fy1|
```

```
                    ¦ .                  ¦ ¦. ¦   ¦ . ¦
                    ¦ k2n,1  ... k2n,2n  ¦ ¦vn¦   ¦fyn¦
                 to ¦                    ¦ ¦u1¦   ¦fx1¦
                    ¦                    ¦ ¦v1¦   ¦fy1¦
                    ¦      new K         ¦ ¦. ¦ = ¦ . ¦
                    ¦                    ¦ ¦un¦   ¦fxn¦
                    ¦                    ¦ ¦vn¦   ¦fyn¦
                    ¦       ¦ ¦d1¦   ¦q1¦
                    ¦ new K ¦ ¦  ¦ = ¦  ¦
                    ¦       ¦ ¦dn¦   ¦qn¦
}
var
   DimNne,c,r,i,i1,i2: Byte;
   w: array[1..NneD_max] of integer;  (integer so the sign can be inversed)
   L: array[1..NneD_max] of Rtype;
   KeepR: Rtype;
begin
   write('   Reshaping ');
   DimNne:=Dim*Nne;
{
                  Calculate index chage
}
   For i:=1 to DimNne do
      w[i]:=Pos_uvuv(i,Nne,Dim);
{
                  swap rows
}
   For i:=2 to DimNne-1 do  { first and last index don't change }
      begin
      i1:=i;
      While 0<w[i1] do
         begin
         i2:=w[i1];
         w[i1]:=-w[i1];  { negative to indicate a used index }
         If i2<>i1 then
            begin
            If i1=i then
               For c:=1 to DimNne do
                  L[c]:=K[i1,c];
            For c:=1 to DimNne do
               begin
               KeepR:=K[i2,c];   (swap new row with L )
               K[i2,c]:=L[c];
               L[c]:=KeepR;
               end;
            end;
         i1:=i2;
         end;
      end;
{
                  restore original index change
}
```

```
   For i:=1 to DimNne do
      w[i]:=abs(w[i]);
(
                  swap columns
)
   For i:=2 to DimNne-1 do  ( first and last index don't change )
      begin
      i1:=i;
      While 0<w[i1] do
         begin
         i2:=w[i1];
         w[i1]:=-w[i1];  ( negative to indicate a used index )
         If i2<>i1 then
           begin
           If i1=i then
              For r:=1 to DimNne do
                 L[r]:=K[r,i1];
           For r:=1 to DimNne do
              begin
              KeepR:=K[r,i2];   (swap new row with L )
              K[r,i2]:=L[r];
              L[r]:=KeepR;
              end;
           end;
         i1:=i2;
         end;
      end;
   writeln;
end;
{-------------------------------------------------------------------}
Procedure ExtractOne(    Pivot,N: Byte;
                         Displacement: Rtype;
                    var K: MatrixR_NneDxNneD;
                    var R:  VectR_NneD);
(
   Input:
      Pivot: position in matrix where we have a fixed displacement
      N: size of matrix
      D: Displacement value
      K: (Nne.Dim)x(Nne.Dim) SYMMETRIC stiffness matrix for element
      R: constants in equation: [K].[D]=[R]
   Output:
      K: modified for unknowns, still SYMMETRIC
      R: modified constants in equation: [K].[D]=[R]
      example:    |k11 k12 k13| |D1 |   |R1'|  X' are knowns
                  |k21 k22 k23| |D2'| = |R2 |
                  |k31 k32 k33| |D3 |   |R3'|
                  Symmetric
                  |k11  0  k13| |D1|   |R1'-k12.D2'|
                  | 0   1   0 | |D2| = |    D2'    |
                  |k31  0  k33| |D3|   |R3'-k32.D2'|
                  Symmetric
```

```
}
var
   row,col: Byte;
begin
   For row:=1 to N do
      If row<>Pivot then
         begin
         R[row]:=R[row]-K[row,Pivot]*Displacement;
         K[row,Pivot]:=0.0;
         end;
   For col:=1 to N do
      If col<>Pivot then
         K[Pivot,col]:=0.0;
   R[Pivot]:=Displacement;
   K[Pivot,Pivot]:=1.0;
end;
{-------------------------------------------------------------------}
Procedure Extract;
{
Procedure Extract(var Ke: MatrixR_NneDxNneD;
                      Dim,Nne: Byte;
                  var F_Displ: FileR;
                  var F_Fd: FileByte;
                      Elem: VectI_Nne2;
                  var R: VectR_NneD);
}
{
   Input:
      K: (Nne.Dim)x(Nne.Dim) SYMMETRIC stiffness matrix for element
      Dim: Dimensions for geometry
      Nne: number of nodes in element
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacement
      Elem[1..Net+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      R: constants in equation: [K].[D]=[R]
   Output:
      K: modified for unknowns, still SYMMETRIC
      R: modified constants in equation: [K].[D]=[R]
}
var
   iNne,iD: Byte;
   OneFixed: Boolean;
   Displ: VectR_D;
   Fd: VectByte_D;
begin
   write('   Extracting unknowns ');
   For iNne:=1 to Nne do
      begin
```

```
      OneFixed:=False;
      D_Byte_Read(F_Fd,abs(Elem[2+iNne]),Dim,Fd);
      For iD:=1 to Dim do
         If Fd[iD]=1 then OneFixed:=True;
      If OneFixed=True then
         begin
         D_R_Read(F_Displ,abs(Elem[2+iNne]),Dim,Displ);
         For iD:=1 to Dim do
            If Fd[iD]=1 then
               ExtractOne(Lin(iNne,iD,Dim),Nne*Dim,Displ[iD],Ke,R);
         end;
      end;
   writeln;
end;
{-------------------------------------------------------------}
end.
```

SOLVE.PAS

```
{-------------------------------------------------------------------
   Finite element program
   Initially written in January '88 by Jerome Daoust
   Solves 2 or 3 dimension problems
   Stiffness elements:
      1: 2D isotropic 3 node, linear displacement
      2: 3D orthotropic 20 node, principal direction for material properties
            at an angle from local coordinate system, quadratic displacement
-------------------------------------------------------------------}
{$N+) { for math coprocessor }
{$M 50000,0,655360}
Uses
   Printer,Dos,Crt,Declare,WaitKey,Nombre,Timer,Where,Nne_Dim,Linear,File_RW,
   Get,Put,Sonore,VectMat3,Status,Exyz,BackSub,Front,Fixed,Sub,Shuffle,Logo,
   Param,Displace,Elem1,Elem2;
{-------------------------------------------------------------------}
Procedure GetConstants(var EFpos: VectI_Nne;
                       var Elem: VectI_Nne2;  { var for speed }
                       var PosKe: MatrixI_NnexD; { var for speed }
                           Nne,Dim: Itype;
                       var F_Force,F_CF: FileR;
                       var CF_Elem: VectR_NneD);
{
   Input:
      EFpos: Element to Front position (in actual matrix)
         negative for new positions
      Elem[1..Net+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      PosKe[iNne,iD]:= Lin(iNne,iD,Dim)
      Nne: number of nodes in element
      Dim: Dimensions for geometry
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_CF: File with Coefficient of front matrix
   Output:
      EFpos: Element to Front position (in actual matrix)
         positive
      CF_Element: Constants in element's equation
}
var
   iNne,iD: Itype;
   Vs,Force: VectR_D;
begin
   For iNne:=1 to Nne do
      If EFpos[iNne]<0 then  { new node in front matrix }
         begin
         EFpos[iNne]:=abs(EFpos[iNne]);
         D_R_Read(F_Force,abs(Elem[2+iNne]),Dim,Force);
         For iD:=1 to Dim do
```

```
            CF_Elem[PosKe[iNne,iD]]:=Force[iD];
         end
      else
         begin
         D_R_Read(F_CF,EFpos[iNne],Dim,Vs);
         for iD:=1 to Dim do
            CF_Elem[PosKe[iNne,iD]]:=Vs[iD];
         end;
end;
{-------------------------------------------------------------------}
Procedure K_elem(    Elem: VectI_Nne2;
                 var F_XYZ: FileR;
                     Mat: MatrixR_Nmatx6;
                     Angle: Rtype;
                 var Ke: MatrixR_NneDxNneD);
{
   Input:
      Elem[1..Nne+2]: defines Material # (i=1),
                              Element Type (i=2),
                              nodes for element (i=3...)
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      Angle: angle for the material in the element
   Output:
      Ke: Element Stiffness matrix:  |Fx1|          |u1 |
                                     |Fy1|          |v1 |
                                     |...|  =  [Ke]. |...|
                                     |FxN|          |uN |
                                     |FyN|          |vN |
}
var
   Etype,iMat: IType;
   Exyz: MatrixR_Nnex3;
begin
   Etype:=Elem[2];
   iMat:=Elem[1];
   Get_Exyz(Elem,F_xyz,Exyz);
   Case Etype of
      1: K_elem1(Exyz[1,1],Exyz[1,2],Exyz[2,1],Exyz[2,2],Exyz[3,1],Exyz[3,2],
                 Mat[iMat,1],Mat[iMat,5],Ke);
      2: K_elem2(iMat,Exyz,Mat,Angle,Ke);
      end;
end;
{-------------------------------------------------------------------}
Procedure GetSubSym(var F_Front: FileR;
                        row,col,FrontNodeSize,Dim: Itype;
                    var AllZero: Boolean;
                    var Fs: MatrixR_DxD);
{
   Input:
      F_Front: File containing Sub-Matrices of Front Matrix
      row,col: desired position for sub-matrix
```

```
    FrontNodeSize: Maximum number of nodes in Front
    Dim: Dimension of problem =2 for 2-D, =3 for 3-D
  Output:
    AllZero: True if all elements of sub-matrix are 0
    Fs: Sub-matrix
  Note: Sub-matrices are stored in upper triangular array
        with one 0 row used as a buffer.
             ¦ k01 k02 k03 ¦
             ¦ k11 k12 k13 ¦
             ¦     k22 k23 ¦
             ¦         k33 ¦
}
var
  Srow,Scol: integer;
  FsT: MatrixR_DxD;
begin
  If row <= col then
    DD_R_Read(F_Front,AllZero,FrontNodeSize,Row,Col,Dim,Fs)
  else
    begin
    DD_R_Read(F_Front,AllZero,FrontNodeSize,Col,Row,Dim,FsT);
{
                Get Transpose of sub-matrix
}
    Case Dim of
       1..2: begin
          For Srow:=1 to Dim do
             For Scol:=1 to Dim do
                Fs[Scol,Srow]:=FsT[Srow,Scol];
          end;
{
                Explicit equations for increased speed in 3D
}
       3: begin
          Fs[1,1]:=FsT[1,1];
          Fs[2,1]:=FsT[1,2];
          Fs[3,1]:=FsT[1,3];
          Fs[1,2]:=FsT[2,1];
          Fs[2,2]:=FsT[2,2];
          Fs[3,2]:=FsT[2,3];
          Fs[1,3]:=FsT[3,1];
          Fs[2,3]:=FsT[3,2];
          Fs[3,3]:=FsT[3,3];
          end;
       end;
    end;
end;
{-------------------------------------------------------------}
Procedure Reduce(var F_Front: FileR;
                 FrontNodeSize,Dim,N_Front,RiFront,Enode: IType;
              var Elem: Vect1_Nne2;  { var for speed }
              var EFpos: Vect1_Nne;  { var for speed }
```

```
                var F_Front_pos: File1;
                var F_CF: FileR);
{
  Input:
    F_Front: File containing Sub-Matrices of Front Matrix
    FrontNodeSize: Maximum number of nodes in Front
    Dim: Dimension of problem  =2 for 2-D, =3 for 3-D
    N_Front: number of active node in front
    RiFront: index number to be removed in front
    Enode: Node number (as element ordering is defined) to be removed
           Removal is done in reverse order
    Elem[1..i+2]: defines Material # (i=1),
                          Element Type (i=2),
                          nodes for element (i=3...)
    EFpos: Element to Front position (in actual matrix)
    F_Front_pos: File with position of nodes in front
    F_CF: File with Coefficient of front matrix
  Output:
    F_Front,F_CF: Front matrix, Gauss-elimination for Front_node[RiFront]
  Example:
    Dim=2
    N_Front=3
    Front_pos=2
    ¦c1¦   ¦ k11 k12 k13 k14 k15 k16 ¦
    ¦c2¦   ¦ k21                     ¦
    ¦c3¦ = ¦ k31     k33 k34         ¦    Let [kP] = ¦ k33 k34 ¦  be the
    ¦c4¦   ¦ k41     k43 k44         ¦               ¦ k43 k44 ¦
    ¦c5¦   ¦ k51                     ¦    pivoting matrix
    ¦c6¦   ¦ k61                 k66 ¦
    ¦ [c1] ¦   ¦ [k11] [k12] [k13] ¦
    ¦ [c2] ¦ = ¦ [k21]  [kP] [k23] ¦
    ¦ [c3] ¦   ¦ [k31] [k32] [k33] ¦
    Step 1: Reduce [kP], store it in row 0, and remember Multipliers
            M = ¦  0   0  0 ¦   (for 3-D case)
                ¦ m21  0  0 ¦   here m21:= k43/k33
                ¦ m31 m32 0 ¦
            Where kij[i,1..3]:=kij[i,1..3]-M[i,1]*kij[1,1..3] i:=2..3
                  kij[i,1..3]:=kij[i,1..3]-M[i,2]*kij[2,1..3] i:=3
            ¦ k01 k02 k03 ¦
            ¦ k11 k12 k13 ¦
            ¦     k22 k23 ¦
            ¦         k33 ¦
            Reset to 0 original position of [kP]
    Step 2: Reduce all other [kij] on pivoting row, including constants
            by using M, store in row 0
            ¦ [c1]  ¦   ¦ [k11]  [k12]  [k13]  ¦
            ¦ [c'2] ¦ = ¦ [k'21] [k'22] [k'23] ¦
            ¦ [c3]  ¦   ¦ [k31]  [k32]  [k33]  ¦
    Step 3: For each other row:
            - Load [kij] in considered row and same column as pivot node
            - Reduce that [kij], remember multipliers as
              M = ¦ m11 m12 m13 ¦   (for 3-D case)
```

```
                    ¦ m21 m22 m23 ¦
                    ¦ m31 m32 m33 ¦
            - Re-initialize to 0 that [kij] -> useless now
            - Reduce all other [kij] and Constants, on row using M,
              store them
}
Var
   Srow,Scol,iNne,Count,Pivot,FposRow,FposCol,FrontRow,FrontCol,
      FposPivot,PivotMaxRcRow: IType;
   FsKeep,M,FsPivot,Fs,FsPCol,FsPRow: MatrixR_DxD;
   VsKeep,Vs,VsProw: VectR_D;
   Mult,Rkeep,PivotMax: Rtype;
   Xs,Ys: integer;
   S: array[1..Nne_max,1..D_max,1..D_max] of Rtype;
   DoPivot: Boolean;
   Spos: VectI_Nne;
   AllZeroPCol,AllZero: Boolean;
{
                Buffer related variables
}
Const
   FrontBufferMax = 10;    { uses FrontBufferMax(9x8+4) bytes, here 0.76 kB }
Var
   { Pivot Row Reading buffer }
   iBuf,ToBuf: integer;
   FsPRowBuf: array[1..FrontBufferMax] of MatrixR_DxD;
   FposColBuf: array[1..FrontBufferMax] of Itype;
begin
   Xs:=WhereX;
   Ys:=WhereY;
   FposPivot:=EFpos[Encode];
{
                Read, reduce, store pivoting Sub-Matrix (row 0)
                Remember multipliers
}
   DD_R_Read(F_Front,AllZero,FrontNodeSize,FposPivot,FposPivot,Dim,FsPivot);
   Case Dim of
      1..2: begin
         For Pivot:=1 to Dim do
            For Srow:=Pivot+1 to Dim do
               begin
               M[Srow,Pivot]:=FsPivot[Srow,Pivot]/FsPivot[Pivot,Pivot];
               FsPivot[Srow,Pivot]:=0;
               For Scol:=Pivot+1 to Dim do
                  FsPivot[Srow,Scol]:=FsPivot[Srow,Scol]
                       -M[Srow,Pivot]*FsPivot[Pivot,Scol];
               end;
         end;
{
                Explicit equations for increased speed in 3D
}
      3: begin
```

```
            M[2,1]:=FsPivot[2,1]/FsPivot[1,1];
            FsPivot[2,1]:=0;
            FsPivot[2,2]:=FsPivot[2,2]
                 -M[2,1]*FsPivot[1,2];
            FsPivot[2,3]:=FsPivot[2,3]
                 -M[2,1]*FsPivot[1,3];
            M[3,1]:=FsPivot[3,1]/FsPivot[1,1];
            FsPivot[3,1]:=0;
            FsPivot[3,2]:=FsPivot[3,2]
                 -M[3,1]*FsPivot[1,2];
            FsPivot[3,3]:=FsPivot[3,3]
                 -M[3,1]*FsPivot[1,3];
            M[3,2]:=FsPivot[3,2]/FsPivot[2,2];
            FsPivot[3,2]:=0;
            FsPivot[3,3]:=FsPivot[3,3]
                 -M[3,2]*FsPivot[2,3];
            end;
         end;
      DD_R_Write(F_Front,False,FrontNodeSize,0,FposPivot,Dim,FsPivot);
      DD_R_Write(F_Front,True,FrontNodeSize,FposPivot,FposPivot,Dim,FsPivot); (reset)
{
                  Consider all other Sub-Matrices on pivoting node row,
                  including constants
                  save Reduced pivot row in row 0
}
      Count:=0;
      GotoXY(Xs,Ys);
      write(N_Front-Count,' ');
      Count:=Count+1;
      For FrontCol:=1 to N_Front do
         begin
         Front_pos_I_Read(F_Front_pos,FrontCol,FposCol);
         If RiFront<>FrontCol then
            begin
            GetSubSym(F_Front,FposPivot,FposCol,FrontNodeSize,Dim,AllZero,FsPRow);
{
                  Reduce sub-matrix
}
            If AllZero=False then
               begin
               Case Dim of
                  1..2: begin
                     For Pivot:=1 to Dim-1 do
                        For Srow:=Pivot+1 to Dim do
                           For Scol:=1 to Dim do
                              FsPRow[Srow,Scol]:=FsPRow[Srow,Scol]
                                                 -M[Srow,Pivot]*FsPRow[Pivot,Scol];
                     end;
{
                   Explicit equations for increased speed in 3D
}
                  3: begin
```

```
                FsPRow[2,1]:=FsPRow[2,1]-M[2,1]*FsPRow[1,1];
                FsPRow[2,2]:=FsPRow[2,2]-M[2,1]*FsPRow[1,2];
                FsPRow[2,3]:=FsPRow[2,3]-M[2,1]*FsPRow[1,3];
                FsPRow[3,1]:=FsPRow[3,1]-M[3,1]*FsPRow[1,1];
                FsPRow[3,2]:=FsPRow[3,2]-M[3,1]*FsPRow[1,2];
                FsPRow[3,3]:=FsPRow[3,3]-M[3,1]*FsPRow[1,3];
                FsPRow[3,1]:=FsPRow[3,1]-M[3,2]*FsPRow[2,1];
                FsPRow[3,2]:=FsPRow[3,2]-M[3,2]*FsPRow[2,2];
                FsPRow[3,3]:=FsPRow[3,3]-M[3,2]*FsPRow[2,3];
                end;
             end;
{
                Save Reduced Pivot Row in row 0
}
          DD_R_Write(F_Front,False,FrontNodeSize,0,FposCol,Dim,FsPRow);
          end;
       end;
    end;
{
                Reduce constants on same Node row as Pivoting sub-matrix.
}
  D_R_Read(F_CF,FposPivot,Dim,VsPRow);
  Case Dim of
    1..2: begin
       For Pivot:=1 to Dim-1 do
          For Srow:=Pivot+1 to Dim do
             VsProw[Srow]:=VsProw[Srow]-M[Srow,Pivot]*VsProw[Pivot];
       end;
{
                Explicit equations for increased speed in 3D
}
    3: begin
       VsProw[2]:=VsProw[2]-M[2,1]*VsProw[1];
       VsProw[3]:=VsProw[3]-M[3,1]*VsProw[1];
       VsProw[3]:=VsProw[3]-M[3,2]*VsProw[2];
       end;
    end;
  D_R_Write(F_CF,FposPivot,Dim,VsProw);
{
                Reduce each other row
}
  For FrontRow:=1 to N_Front do
    begin
    Front_pos_I_Read(F_Front_pos,FrontRow,FposRow);
    If RiFront<>FrontRow then
       begin
       GotoXY(Xs,Ys);
       write(N_Front-Count,' ');
       Count:=Count+1;
{
                Load [kij] in pivoting column and considered row
                Reduce that [kij], store it, remember multipliers
```

```
}
        GetSubSym(F_Front,FposRow,FposPivot,FrontNodeSize,Dim,
           AllZeroPCol,FsPCol);
        If AllZeroPCol=False then
           begin
           Case Dim of
              1..2: begin
                 For Pivot:=1 to Dim do
                    For Srow:=1 to Dim do
                       begin
                       M[Srow,Pivot]:=FsPCol[Srow,Pivot]/FsPivot[Pivot,Pivot];
                       FsPCol[Srow,Pivot]:=0;
                       For Scol:=Pivot+1 to Dim do
                          FsPCol[Srow,Scol]:=FsPCol[Srow,Scol]
                                -M[Srow,Pivot]*FsPivot[Pivot,Scol];
                       end;
                 end;
{
                  Explicit equations for increased speed in 3D
}
              3: begin
                 M[1,1]:=FsPCol[1,1]/FsPivot[1,1];
                 FsPCol[1,1]:=0;
                    FsPCol[1,2]:=FsPCol[1,2]-M[1,1]*FsPivot[1,2];
                    FsPCol[1,3]:=FsPCol[1,3]-M[1,1]*FsPivot[1,3];
                 M[2,1]:=FsPCol[2,1]/FsPivot[1,1];
                 FsPCol[2,1]:=0;
                    FsPCol[2,2]:=FsPCol[2,2]-M[2,1]*FsPivot[1,2];
                    FsPCol[2,3]:=FsPCol[2,3]-M[2,1]*FsPivot[1,3];
                 M[3,1]:=FsPCol[3,1]/FsPivot[1,1];
                 FsPCol[3,1]:=0;
                    FsPCol[3,2]:=FsPCol[3,2]-M[3,1]*FsPivot[1,2];
                    FsPCol[3,3]:=FsPCol[3,3]-M[3,1]*FsPivot[1,3];
                 M[1,2]:=FsPCol[1,2]/FsPivot[2,2];
                 FsPCol[1,2]:=0;
                    FsPCol[1,3]:=FsPCol[1,3]-M[1,2]*FsPivot[2,3];
                 M[2,2]:=FsPCol[2,2]/FsPivot[2,2];
                 FsPCol[2,2]:=0;
                    FsPCol[2,3]:=FsPCol[2,3]-M[2,2]*FsPivot[2,3];
                 M[3,2]:=FsPCol[3,2]/FsPivot[2,2];
                 FsPCol[3,2]:=0;
                    FsPCol[3,3]:=FsPCol[3,3]-M[3,2]*FsPivot[2,3];
                 M[1,3]:=FsPCol[1,3]/FsPivot[3,3];
                 FsPCol[1,3]:=0;
                 M[2,3]:=FsPCol[2,3]/FsPivot[3,3];
                 FsPCol[2,3]:=0;
                 M[3,3]:=FsPCol[3,3]/FsPivot[3,3];
                 FsPCol[3,3]:=0;
                 end;
              end;
{
                  After we have obtained the Multipliers we re-initialize
```

```
                    that Sub-Matrix
                 Here we use a quick initialization technique
                 At the end the upper row and column corresponding to
                    the pivot location are re-initialized to 0
}
              If FposRow <= FposPivot then
                 DD_R_Write(F_Front,True,FrontNodeSize,FposRow,FposPivot,Dim,FsPCol)
              else
                 DD_R_Write(F_Front,True,FrontNodeSize,FposPivot,FposRow,Dim,FsPCol);
              end;
{
                    Only modify Node row if Multipliers are significant
                       to save time ( not all known displacements ).
}
           If AllZeroPCol=False then
              begin
{
                    Reduce all other [kij] and Constants, on row using M,
                    store them
}
              FrontCol:=1;
              While FrontCol<=N_Front do
                 begin
{
                    Store as many sub-matrices in same row as pivot
                       to reduce the disk acces time when reading from
                       the node row and the row being reduced.
                    consider sub-matrices with row<=col and col <> pivot col
}
                 iBuf:=1;
                 While (FrontCol<=N_Front) and (iBuf<=FrontBufferMax) do
                    begin
                    Front_pos_I_Read(F_Front_pos,FrontCol,FposColBuf[iBuf]);
                    If (FposRow <= FposColBuf[iBuf]) and
                       (FposPivot <> FposColBuf[iBuf]) then
                       begin
                       DD_R_Read(F_Front,AllZero,FrontNodeSize,0,
                                FposColBuf[iBuf],Dim,FsPRowBuf[iBuf]);
                       If AllZero=False then
                          iBuf:=iBuf+1;
                       end;
                    FrontCol:=FrontCol+1;
                    end;
                 ToBuf:=iBuf-1;
{
                     Reduce all valid sub-matrice found by buffer
}
                 For iBuf:=1 to ToBuf do
                    begin
                    DD_R_Read(F_Front,AllZero,FrontNodeSize,FposRow,
                             FposColBuf[iBuf],Dim,Fs);
                    Case Dim of
```

```
                    1..2: begin
                       For Pivot:=1 to Dim do
                       For Srow:=1 to Dim do
                          For Scol:=1 to Dim do
                             Fs[Srow,Scol]:=Fs[Srow,Scol]
                                -M[Srow,Pivot]*FsPRowBuf[iBuf][Pivot,Scol];
                       end;
{
                  Explicit equations for increased speed in 3D
}
                    3: begin
                       Fs[1,1]:=Fs[1,1]-M[1,1]*FsPRowBuf[iBuf][1,1]
                                       -M[1,2]*FsPRowBuf[iBuf][2,1]
                                       -M[1,3]*FsPRowBuf[iBuf][3,1];
                       Fs[1,2]:=Fs[1,2]-M[1,1]*FsPRowBuf[iBuf][1,2]
                                       -M[1,2]*FsPRowBuf[iBuf][2,2]
                                       -M[1,3]*FsPRowBuf[iBuf][3,2];
                       Fs[1,3]:=Fs[1,3]-M[1,1]*FsPRowBuf[iBuf][1,3]
                                       -M[1,2]*FsPRowBuf[iBuf][2,3]
                                       -M[1,3]*FsPRowBuf[iBuf][3,3];
                       Fs[2,1]:=Fs[2,1]-M[2,1]*FsPRowBuf[iBuf][1,1]
                                       -M[2,2]*FsPRowBuf[iBuf][2,1]
                                       -M[2,3]*FsPRowBuf[iBuf][3,1];
                       Fs[2,2]:=Fs[2,2]-M[2,1]*FsPRowBuf[iBuf][1,2]
                                       -M[2,2]*FsPRowBuf[iBuf][2,2]
                                       -M[2,3]*FsPRowBuf[iBuf][3,2];
                       Fs[2,3]:=Fs[2,3]-M[2,1]*FsPRowBuf[iBuf][1,3]
                                       -M[2,2]*FsPRowBuf[iBuf][2,3]
                                       -M[2,3]*FsPRowBuf[iBuf][3,3];
                       Fs[3,1]:=Fs[3,1]-M[3,1]*FsPRowBuf[iBuf][1,1]
                                       -M[3,2]*FsPRowBuf[iBuf][2,1]
                                       -M[3,3]*FsPRowBuf[iBuf][3,1];
                       Fs[3,2]:=Fs[3,2]-M[3,1]*FsPRowBuf[iBuf][1,2]
                                       -M[3,2]*FsPRowBuf[iBuf][2,2]
                                       -M[3,3]*FsPRowBuf[iBuf][3,2];
                       Fs[3,3]:=Fs[3,3]-M[3,1]*FsPRowBuf[iBuf][1,3]
                                       -M[3,2]*FsPRowBuf[iBuf][2,3]
                                       -M[3,3]*FsPRowBuf[iBuf][3,3];
                       end;
                    end;
                 DD_R_Write(F_Front,False,FrontNodeSize,FposRow,
                           FposColBuf[iBuf],Dim,Fs);
                 end;
              end;
{
                  Reduce Constants
}
           D_R_Read(F_CF,FposRow,Dim,Vs);
           Case Dim of
              1..2: begin
                 For Pivot:=1 to Dim do
                    For Srow:=1 to Dim do
```

```
                    Vs[Srow]:=Vs[Srow]-M[Srow,Pivot]*VsPRow[Pivot];
                end;
{
                 Explicit equations for increased speed in 3D
}
             3: begin
                Vs[1]:=Vs[1]-M[1,1]*VsPRow[1]
                           -M[1,2]*VsPRow[2]
                           -M[1,3]*VsPRow[3];
                Vs[2]:=Vs[2]-M[2,1]*VsPRow[1]
                           -M[2,2]*VsPRow[2]
                           -M[2,3]*VsPRow[3];
                Vs[3]:=Vs[3]-M[3,1]*VsPRow[1]
                           -M[3,2]*VsPRow[2]
                           -M[3,3]*VsPRow[3];
                end;
              end;
           D_R_Write(F_CF,FposRow,Dim,Vs);
           end;
        end;
     end;
  GotoXY(Xs,Ys);
end;
{-------------------------------------------------------------------}
Procedure Store_Fixed(    Dim,iE: Itype;
                    var Elem: VectI_Nne2;  { var for speed }
                    var PosKe: MatrixI_NnexD; { var for speed }
                    var Ke: MatrixR_NneDxNneD; { var for speed }
                    var F_Fd: FileByte;
                    var F_Rev: FileI;
                    var F_Eq: FileR);
{
  Input:
    Dim: Dimension of problem  =2 for 2-D, =3 for 3-D
    iE: Element number
    Elem[1..i+2]: defines Material # (i=1),
                          Element Type (i=2),
                          nodes for element (i=3...)
    PosKe[iNne,iD]:= Lin(iNne,iD,Dim)
    Ke: (Nne.Dim)x(Nne.Dim) SYMMETRIC stiffness matrix for element
    F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                              =1 for Displacement
    F_Rev: File containing order of back substitution
    F_Eq: File containing equation constants
  Output (Disk File);
    F_Rev: File containing order of back substitution
          |Last:    Element number (negative)
           ...
          |First:   Element number (negative)
    F_Eq: File containing equation constants
          |last:    rows needed in stiffness matrix
          |         ...
```

```
                {        rows needed in stiffness matrix
                 ...
                {First:  rows needed in stiffness matrix
                {        ...
                {        rows needed in stiffness matrix
}
var
   col,iD,iNns,Nne: integer;
   Negative_iE: Itype;
   StoreElem: Boolean;
   Fd_Elem: MatrixByte_NnexD;
   Xs,Ys: Byte;
begin
   write('  Storing rows for unknown forces ');
   Xs:=WhereX;
   Ys:=WhereY;
   StoreElem:=False;
   Nne:=Get_Nne(Elem[2]);
   get_Fd_Elem(Dim,Elem,F_Fd,Fd_Elem);
{
                Store needed row in the element's stiffness matrix
                Only for nodes used for last time (negative)
}
   For iNne:=1 to Nne do
      begin
      GotoXY(Xs,Ys);
      Write(Nne-iNne+1,' ');
{
                Count then number of unknown forces in any direction
                for nodes used for the last time in this element
}
      For iD:=1 to Dim do
         If (Fd_Elem[iNne,iD]=1) and (Elem[2+iNne]<0) then
            begin
            For col:=1 to Nne*Dim do
               write(F_Eq,Ke[PosKe[iNne,iD],col]);
            StoreElem:=True;
            end;
      end;
{
                Store NEGATIVE element number when a force is missing
}
   If StoreElem=True then
      begin
      Negative_iE:=-abs(iE);
      write(F_Rev,Negative_iE);
      end;
   GotoXY(Xs,Ys);
   Writeln('  ');
end;
{-------------------------------------------------------------}
Procedure Store_node(var F_Front: FileR;
```

```
                    FrontNodeSize,Dim: IType;
                    N_Front: IType;
                var F_Front_pos,F_Front_node: FileI;
                    RiFront: IType;
                var F_CF: FileR;
                var F_Rev: FileI;
                var F_Eq: FileR);
{
  Input:
    F_Front: File containing Sub-Matrices
    FrontNodeSize: Maximum number of nodes in Front
    Dim: Dimension of problem  =2 for 2-D, =3 for 3-D
    N_Front: number of active node in front
    F_Front_pos: File with position of working nodes in front
    F_Front_node: File with node number at Front_pos,
                  negative for nodes used for the last time
    RiFront: position for Front_pos[RiFront] and Front_node[RiFront]
         to be removed
    F_CF: File with Coefficient vector of Front Matrix
    F_Rev: File containing order of back substitution
    F_Eq: File containing equation constants
  Output (Disk File);
    F_Rev: File containing order of back substitution
           |last:  node # to be solved
           |         node used
           |         ...
           |         node used
           |         # of nodes used (positive)
            ...
           |First: node # to be solved
           |         node used
           |         ...
           |         node used
           |         # of nodes used (positive)
    F_Eq: File containing equation constants
           |last:  Constants for node to be solved
           |          Constant for node used
           |          ...
           |          Constant for node used
           |       Constants of equations
            ...
           |First: Constants for node to be solved
           |          Constant for node used
           |          ...
           |          Constant for node used
           |       Constants of equations
  Note:
    Complete row for node is given as row number 0
}
var
  QttStored,FposCol,FposPivot,FnodeI,Fnode,iFront,i,j: IType;
  Vs: VectR_D;
```

```
   Fs: MatrixR_DxD;
   Prefix,Name: String12;
   Xs,Ys: integer;
   AllZero: Boolean;
begin
   Xs:=WhereX;
   Ys:=WhereY;
   write('Storing equations ');
{
                Append node number to File describing reverse order
}
   Front_node_I_Read(F_Front_node,RiFront,Fnode);
   Fnode:=abs(Fnode);
   Write(F_Rev,Fnode);
{
                Store equations for node FRONT_NODE[RiFront]
                the removed node stiffness matrix is placed in front
                example:  k11 k12 k13 k14 k15 k16
                          k21 k22  0  k24 k25 k26
                replaced: k13 k14 k11 k12 k15 k16
                           0  k24 k21 k22 k25 k26
                This information is stored in consecutive Column order
                -> k13 0 k14 k24 ...  k16 k26 C1 C2
                Store C_F coefficients
                Re-initialize constants in pivot node row
}
   Front_pos_I_Read(F_Front_pos,RiFront,FposPivot);
   DD_R_Read(F_Front,AllZero,FrontNodeSize,0,FposPivot,Dim,Fs);
   For j:=1 to Dim do
      For i:=1 to Dim do
         Write(F_Eq,Fs[i,j]);
{
                Fast init
}
   DD_R_Write(F_Front,True,FrontNodeSize,0,FposPivot,Dim,Fs);
{
                Save submatrices in row 0 wich are not 0
                Re-Initialize them
                Store the number of the nodes used without the
                   node which is removed
}
   QttStored:=1;
   For iFront:=1 to N_Front do
      If iFront<>RiFront then
         begin
         Front_pos_I_Read(F_Front_pos,iFront,FposCol);
         DD_R_Read(F_Front,AllZero,FrontNodeSize,0,FposCol,Dim,Fs);
         If AllZero=False then
            begin
            QttStored:=QttStored+1;
            Front_node_I_Read(F_Front_node,iFront,FnodeI);
            FnodeI:=abs(FnodeI);    {avoid a negative value}
```

```
            Write(F_Rev,FnodeI);
            For j:=1 to Dim do
               For i:=1 to Dim do
                  Write(F_Eq,Fs[i,j]);
{
                  Fast Init
}
            DD_R_Write(F_Front,True,FrontNodeSize,0,FposCol,Dim,Fs);
            end;
         end;
{
                  Store number of nodes used, including reduced node
}
   Write(F_Rev,QttStored);
{
                  Save Constants
                  Re-Initialize
}
   D_R_Read(F_CF,FposPivot,Dim,Vs);
   For i:=1 to Dim do
      write(F_Eq,Vs[i]);
   D_R_0(F_CF,FposPivot,Dim);
   GotoXY(Xs,YS);
   write('                ');
   GotoXY(Xs,YS);
end;
{-------------------------------------------------------------------}
Procedure AddKe(    Nne,Dim,FrontNodeSize: Itype;
                var Fixed: VectB_Nne;        (var for speed)
                var F_CF,F_Front: FileR;
                var Ke: MatrixR_NneDxNneD;   (var for speed)
                var CF_Elem: VectR_NneD;        (var for speed)
                var PosKe: MatrixI_NnexD;    (var for speed)
                var EFpos: VectI_Nne);       (var for speed)
{
   Input:
      Nne: number of nodes in the element
      Dim: Dimensions for geometry
      FrontNodeSize: Maximum number of nodes in Front
      Fixed[iNne]: True if a node is fixed in all directions
      F_CF: File containing constants of Front
      F_Front: File containing Front matrix
      Ke: (Nne.Dim)x(Nne.Dim) stiffness matrix for element
      CF_Elem: constants in the new equation: [C]=[K].[x]
      PosKe[iNne,iD]:= Lin(iNne,iD,Dim)
      EFpos: Element to Front position (in actual matrix)
   Output:
      Add Ke to existing Front matrix
      Replace Constants in Front Matrix by the element's
}
var
   Xs,Ys: integer;
```

```
   iD,jD,NewRow,NewCol,RowNne,ColNne: Itype;
   Fs: MatrixR_DxD;
   Vs: VectR_D;
   AllZero: Boolean;
begin
   write('   Adding to front ');
   Xs:=WhereX;
   Ys:=WhereY;
{
                  Constants are replaced (they were summed before)
}
   For RowNne:=1 to Nne do
      begin
{
                  avoid adding constants in a fixed node row
                  -> no operation is done for a completely fixed node
}
      If Fixed[RowNne]=False then
         begin
         NewRow:=EFpos[RowNne];
         For iD:=1 to Dim do
            Vs[iD]:=CF_Elem[PosKe[RowNne,iD]];
         D_R_Write(F_CF,newRow,Dim,Vs);
         end;
      end;
{
                  Making the row the slowest loop decreases disk acces time
                  Only add upper triangular portion of stiffess matrix
}
   For RowNne:=1 to Nne do
      begin
      GotoXY(Xs,Ys);
      write(Nne-RowNne+1,' ');
      For ColNne:=1 to Nne do
         begin
         NewRow:=EFpos[RowNne];
         NewCol:=EFpos[ColNne];
         If NewRow <= NewCol then
            begin
{
                  avoid adding a sub-matrix in fixed node column or row
                  -> no operation is done for a completely fixed node
}
            If (Fixed[RowNne]=False)and(Fixed[ColNne]=False) then
               begin
               DD_R_Read(F_Front,AllZero,FrontNodeSize,newRow,newCol,Dim,Fs);
               For iD:=1 to Dim do
                  For jD:=1 to Dim do
                     Fs[iD,jD] := Fs[iD,jD]
                                 + Ke[PosKe[RowNne,iD],PosKe[ColNne,jD]];
               DD_R_Write(F_Front,False,FrontNodeSize,newRow,newCol,Dim,Fs);
               end;
```

```
          end;
        end;
      end;
   GotoXY(Xs,Ys);
   writeln(' ');
end;
{-------------------------------------------------------------------}
Procedure Solve(    FrontNodeSize: IType;
                var F_XYZ: FileR;
                var F_Force,F_Displ: FileR;
                var F_Fd: FileByte;
                    Net,Nnt,NneMax,Dim: IType;
                var F_Elem: FileI;
                var F_Angle: FileR;
                    Mat: MatrixR_Nmatx6;
                    CaseName: String255);
{
   Input:
      FrontNodeSize: Maximum number of nodes in Front
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
          each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_Fd[1..Nnt x Dimension]: File says if a force is known =0,
                                =1 for Displacemont
      Net: number of elements;
      NneMax: Maximum number of nodes in an element
      Dim: Dimensions for geometry
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Node are negative in Element matrix when they are use for
              the last time according to the numbering of the elements
      F_Angle: File With angle of elements (radians)
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      CaseName: Name of current case to solve
   Output:
      Force: Force vector: Fx1 Fy1 Fz1 ... FxN FyN FzN
      Displ: Displacement vector: u1 v1 w1 ... uN vN wN
      Before this Subroutine they were only partially known
}
var
   i,Fpos,iNne,iD,jNne,jD,row,col,Nne,iFront,iE,N_Front,
      RiNne,Rnode,Fnode: IType;
   Xs,Ys: integer;
   Ke: MatrixR_NneDxNneD;
   Order,EFpos,EFindex: VectI_Nne;
   Fixed: VectB_Nne;
   Fs: MatrixR_DxD;
   Force,Displ,Vs: VectR_D;
   Fd: VectByte_D;
```

```
   CF_Elem: VectR_NneD;
   ZeroI: IType;
   Elem: VectI_Nne2;
   Angle: Rtype;
   PosKe: MatrixI_NnexD;
   F_Rev: FileI;
   F_Eq: FileR;
   F_Front_pos,F_Front_node: FileI;
   F_CF,F_Front: FileR;
   CheckTime,CorrectTime,StoreTime,IndexTime,KnowTime,ReshapTime,BackTime,
     OpenTime,DelTime,AddTime,IniTime,StiffTime,ReduceTime,SolveTime: Rtype;
begin
   ResetTimerAll;
   StartTimer(1);
   StartTimer(2);
   Check_Size(Net,Nnt,Dim,NneMax,F_Elem,F_Fd,FrontNodeSize);
   StopTimer(2);
   CheckTime:=ReadTimer(2);
{
                Initialize Maximum Front matrix and constants
}
   ResetTimer(2);
   StartTimer(2);
   ZeroI:=0;
   N_Front:=0;            { Actual Front dimension }
   Assign(F_Front_Pos,T_Dir+'Fpos.T');
   Assign(F_Front_node,T_Dir+'Fnode.T');
   Assign(F_Eq,T_Dir+'Eq.T');
   Assign(F_Rev,T_Dir+'Rev.T');
   Assign(F_Front,T_Dir+'Front.T');
   Assign(F_CF,T_Dir+'FC.T');
   Rewrite(F_Front_Pos);
   Rewrite(F_Front_node);
   Rewrite(F_Eq);
   Rewrite(F_Rev);
   Rewrite(F_Front);
   Rewrite(F_CF);
   StopTimer(2);
   OpenTime:=ReadTimer(2);
{
                Initialize Front Matrix
                See Sub.inc for order of initialization
}
   ResetTimer(2);
   StartTimer(2);
{
                Initialization
}
   write('Initialization for a ',FrontNodeSize,' node wave front  ');
   Xs:=WhereX;
   Ys:=WhereY;
   For iD:=1 to D_max do
```

```
    For jD:=1 to D_max do
      Fs[iD,jD]:=0;
  Fs[1,1]:=BigRmax;
{
              Initialize row 0
}
  GotoXY(Xs,Ys);
  write(FrontNodeSize-0+1,' ');
  For col:=1 to FrontNodeSize do
    DD_R_Write(F_Front,False,FrontNodeSize,0,col,Dim,Fs);
{
              Initialize rest of matrix, column by column
}
  For col:=1 to FrontNodeSize do
    begin
    GotoXY(Xs,Ys);
    write(FrontNodeSize-Col+1,' ');
    For row:=1 to FrontNodeSize do
      If row<=col then
        DD_R_Write(F_Front,False,FrontNodeSize,row,col,Dim,Fs);
    end;
  For row:=1 to FrontNodeSize do
    D_R_O(F_CF,row,Dim);
  GotoXY(Xs,Ys);
  writeln('  ');
{
              Calculate PosKe[iNne,iD]=Lin(iNne,iD,Dim)
              A lot of speed is due to calculating this only once
}
  For iNne:=1 to Nne_max do
    For iD:=1 to Dim do
      PosKe[iNne,iD]:=Lin(iNne,iD,Dim);
{
              Initialize time sums
}
  ReduceTime:=0;
  StiffTime:=0;
  ReshapTime:=0;
  KnowTime:=0;
  IndexTime:=0;
  AddTime:=0;
  StoreTime:=0;
  CorrectTime:=0;
  StopTimer(2);
  IniTime:=ReadTimer(2);
{
              Create Front for all elements,
              then reduce and store equations
}
  ResetTimer(2);  {stiffness}
  ResetTimer(3);  {reshape}
  ResetTimer(4);  {store}
```

```
    ResetTimer(5); {index Element -> front}
    ResetTimer(6); {extracting known vector in element equations}
    ResetTimer(7); {add element to front}
    ResetTimer(8); {reduction}
    ResetTimer(9); {correct front node index}
    For iE:=1 to Net do
       begin
       StartTimer(2);
       Nne2_I_Read(F_Elem,iE,NneMax,Elem);
       R_Read(F_Angle,iE,Angle);
       Nne:=Get_Nne(Elem[2]);
{
                Get element stiffness matrix
}
       TextColor(LightRed);
       write(Net-iE+1);
       TextColor(LightGreen);
       writeln(' elements left for case ',CaseName);
       K_elem(Elem,F_XYZ,Mat,Angle,Ke);
       StopTimer(2);
{
                Reshape Ke
                  to have   u1 v1 w1 ... uN vN wN
                  instead of  u1 .. uN  v1 .. vN  w1 .. wN
}
       StartTimer(3);
       Reshape(Ke,Nne,Dim);
       StopTimer(3);
{
                Store rows needed for unknown forces
                  when node is used for the last time
                This must be done before Storing of equations
                  of nodes with given forces
                And before row are reset to 0 for fixed displacement
                  in stiffness matrix within EXTRACT subroutine
}
       StartTimer(4);
       Store_Fixed(Dim,iE,Elem,PosKe,Ke,F_Fd,F_Rev,F_Eq);
       StopTimer(4);
{
                update the node # used in the front (Front_pos,
                Front_node,N_Front), see the subroutine's explanations
                Estabish correspondance between order of the nodes in
                the element and where this node is placed in the Front
}
       StartTimer(5);
       write('    Element->Front ');
       Front_Place(Elem,N_Front,F_Front_pos,F_Front_node,EFpos,EFindex);
       writeln;
       StopTimer(5);
{
                Reshape element to keep unknowns of Displacement and Forces
```

```
                as variables
}
      StartTimer(6);
      GetConstants(EFpos, Elem,PosKe, Nne,Dim, F_Force,F_CF,CF_Elem);
      Extract(Ke,Dim,Nne, F_Displ,F_Fd,Elem,CF_Elem);
      StopTimer(6);
{
                Add Element to Front Matrix
}
      StartTimer(7);
      FixedNode(F_Fd,Elem,Nne,Dim,Order,Fixed);
      AddKe(Nne,Dim,FrontNodeSize,Fixed,F_CF,F_Front,Ke,CF_Elem,PosKe,EFpos);
      StopTimer(7);
{
                Reduce Front matrix where nodes are used for last time
                Store equations
                1 node removed at a time ( clear rows and columns
                  for that node)
}
      write('    Reducing ');
      Xs:=WhereX;
      Ys:=WhereY;
      For iNne:=1 to Nne do
        begin
        gotoXY(Xs,Ys);
        write(Nne-iNne+1,' ');
        RiNne:=Order[iNne];
        Rnode:=Elem[2+RiNne];
{
                only reduce nodes used for last time (negative node value)
}
        If Rnode<0 then
          begin
          iFront:=EFindex[RiNne];
{
                Reduce node
                When a node is completely fixed, this is not necessary
                  because rows or columns for a completely fixed node are
                  never added to the front matrix and front constants
}
          StartTimer(8);
          If  Fixed[RiNne]=false then
             Reduce(F_Front,FrontNodeSize,Dim,N_Front,EFindex[RiNne],RiNne,
                  Elem,EFpos,F_Front_pos,F_CF);
          StopTimer(8);
{
                Store equations for reduced node
}
          StartTimer(4);
          If  Fixed[RiNne]=false then
             Store_node(F_Front,FrontNodeSize,Dim,N_Front,
                       F_Front_pos,F_Front_node,iFront,F_CF,F_Rev,F_Eq);
```

```
            StopTimer(4);
{
                    Correct Front index
}
            StartTimer(9);
            Correct_Front(RiNne,Nne,N_Front,F_Front_pos,F_Front_node,EFindex);
            StopTimer(9);
            end
          end;
        gotoXY(Xs,Ys);
        writeln(' ');
        end;
{
                    Get summed times
}
   StiffTime:=ReadTimer(2);
   ReshapTime:=ReadTimer(3);
   StoreTime:=ReadTimer(4);
   IndexTime:=ReadTimer(5);
   KnowTime:=ReadTimer(6);
   AddTime:=ReadTimer(7);
   ReduceTime:=ReadTimer(8);
   CorrectTime:=ReadTimer(9);
{
                  Dispay exact neaded storage space
                  Erase Sub-Matrices and Pointer
}
   ResetTimer(2);
   StartTimer(2);
   TextColor(LightRed);
   write((FileSize(F_Front)*SizeOf(RType)
         +FileSize(F_CF)*SizeOf(RType)
         +FileSize(F_Rev)*SizeOf(Itype)
         +FileSize(F_Eq)*SizeOf(RType)
         +FileSize(F_Front_Pos)*SizeOf(Itype)
         +FileSize(F_Front_Node)*SizeOf(Itype)
         )/Sqr(1024.0):8:4);
   TextColor(LightGreen);
   writeln(' MB were really needed for disk storage of equations.');
   DisposeSubHeapRAM;
   writeln('Deleting file containing sub-matrices');
   Close(F_Front);
   Erase(F_Front);
   Close(F_CF);
   Erase(F_CF);
   Close(F_Front_Pos);
   Erase(F_Front_Pos);
   Close(F_Front_node);
   Erase(F_Front_node);
   StopTimer(2);
   DelTime:=ReadTimer(2);
{
```

```
                Back substitute
}
  ResetTimer(2);
  StartTimer(2);
  InitDisplHeapRAM(Dim,Nnt,F_Displ);
  Back(F_Displ,F_Force,Dim,Nnt,NneMax,F_Fd,F_Elem,F_Rev,F_Eq);
{
                save displacements at this point
                Remove memory allocated for displacements
}
  RamDisplToDisk(Dim,F_Displ);
  DisposeDisplHeapRAM;
  StopTimer(2);
  BackTime:=ReadTimer(2);
{
                If solution takes more that 1 minute -> Beep
}
  StopTimer(1);
  SolveTime:=ReadTimer(1);
  If 60<SolveTime then Alert;
{
                Execution time profile
}
  ClrScr;
  Write('                        EXECUTION PROFILE for ');
  TextColor(LightRed);
  writeln(CaseName);
  TextColor(LightGreen);
  Write(SolveTime:15:2,'s (',HourMinSec(SolveTime),') total time, ');
  writeln(SolveTime-(CheckTime+OpenTime+IniTime+StiffTime+ReshapTime
        +KnowTime+IndexTime+AddTime+ReduceTime+StoreTime+CorrectTime
        +DelTime+BackTime):10:2,'s unaccounted');
  writeln;
  Writeln(CheckTime:15:2,'s for size prediction');
  Writeln(OpenTime:15:2,'s for opening files');
  Writeln(IniTime:15:2,'s for initializing front matrix');
  Writeln(StiffTime:15:2,'s for calculating stiffness');
  writeln('   ',StiffTime/Net:15:2,'s per element');
  Writeln(ReshapTime:15:2,'s for reshaping elements');
  writeln('   ',ReshapTime/Net:15:2,'s per element');
  Writeln(KnowTime:15:2,'s for extracting unknowns in elements');
  writeln('   ',KnowTime/Net:15:2,'s per element');
  Writeln(IndexTime:15:2,'s for indexing elements to front');
  writeln('   ',IndexTime/Net:15:2,'s per element');
  Writeln(AddTime:15:2,'s for adding elements to front');
  writeln('   ',AddTime/Net:15:2,'s per element');
  Writeln(ReduceTime:15:2,'s for reducing nodes in front');
  write('   ',ReduceTime/Net:15:2,'s per element');
  writeln('   ',ReduceTime/Nnt:15:2,'s per node');
  Writeln(StoreTime:15:2,'s for storing reduced nodes');
  writeln('   ',StoreTime/Nnt:15:2,'s per node');
  Writeln(CorrectTime:15:2,'s for correcting front index after node removal');
```

```
   writeln('   ',CorrectTime/Nnt:15:2,'s per node');
   Writeln(DelTime:15:2,'s for deleting files');
   Writeln(BackTime:15:2,'s for backsubstitution');
   If OnPrinter=True then
      begin
      Writeln(Lst,'                          EXECUTION PROFILE for ',CaseName);
        Write(Lst,SolveTime:15:2,'s (',HourMinSec(SolveTime),') total time, ');
      writeln(Lst,SolveTime-(CheckTime+OpenTime+IniTime+StiffTime+ReshapTime
             +KnowTime+IndexTime+AddTime+ReduceTime+StoreTime+CorrectTime
             +DelTime+BackTime):10:2,'s unaccounted');
      writeln(Lst);
      Writeln(Lst,CheckTime:15:2,'s for size prediction');
      Writeln(Lst,OpenTime:15:2,'s for opening files');
      Writeln(Lst,IniTime:15:2,'s for initializing front matrix');
      Writeln(Lst,StiffTime:15:2,'s for calculating stiffness');
      writeln(Lst,'   ',StiffTime/Net:15:2,'s per element');
      Writeln(Lst,ReshapTime:15:2,'s for reshaping elements');
      writeln(Lst,'   ',ReshapTime/Net:15:2,'s per element');
      Writeln(Lst,KnowTime:15:2,'s for extracting unknowns in elements');
      writeln(Lst,'   ',KnowTime/Net:15:2,'s per element');
      Writeln(Lst,IndexTime:15:2,'s for indexing elements to front');
      writeln(Lst,'   ',IndexTime/Net:15:2,'s per element');
      Writeln(Lst,AddTime:15:2,'s for adding elements to front');
      writeln(Lst,'   ',AddTime/Net:15:2,'s per element');
      Writeln(Lst,ReduceTime:15:2,'s for reducing nodes in front');
        write(Lst,'   ',ReduceTime/Net:15:2,'s per element');
      writeln(Lst,'   ',ReduceTime/Nnt:15:2,'s per node');
      Writeln(Lst,StoreTime:15:2,'s for storing reduced nodes');
      writeln(Lst,'   ',StoreTime/Nnt:15:2,'s per node');
      Writeln(Lst,CorrectTime:15:2,'s for correcting front index after node removal');
      writeln(Lst,'   ',CorrectTime/Nnt:15:2,'s per node');
      Writeln(Lst,DelTime:15:2,'s for deleting files');
      Writeln(Lst,BackTime:15:2,'s for backsubstitution');
{
                Form Feed
}
      Write(Lst,Chr(12),chr(13));
      end;
end;
{-------------------------------------------------------------------}
Procedure Display_F_d(    Nnt,Dim: IType;
                      var F_NewOld: FileI;
                      var F_Force,F_Displ: FileR);
{
   Input:
      Nnt: Maximum node number
      Dim: Dimension of problem  =2 for 2-D, =3 for 3-D
      F_NewOld: original node number given by user for each "compacted" node #
      F_Force[1..Nnt x Dimension]: File with Forces in Fx1,Fy1,Fz1 for
         each node
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
         each node
```

```
   Output:
      Prints Nodal forces and displacements
)
var
   Old,iNnt,iD: IType;
   Force,Displ: VectR_D;
   StopNnt: Itype;
begin
   ClrScr;
   If Nnt<=20 then StopNnt:=Nnt
              Else StopNnt:=20;
   writeln('  ',StopNnt,' first nodes');
   If Dim=2 then
      writeln('         Fx,         Fy,         U,          V')
   else
      writeln('         Fx,         Fy,         Fz,         U,          V,          W');
   For iNnt:=1 to StopNnt do
      begin
      D_R_Read(F_Force,iNnt,Dim,Force);
      D_R_Read(F_Displ,iNnt,Dim,Displ);
      I_Read(F_NewOld,iNnt,Old);
      write(Old:3,': ');
      For iD:=1 to Dim do write(RField(Force[iD],11),' ');
      For iD:=1 to Dim do write(RField(Displ[iD],11),' ');
      writeln;
      end;
end;
{-------------------------------------------------------------------}
var
   NneMax,FrontNodeSize,Nnt,Dim,Net: IType;
   Mat: MatrixR_Nmatx6;
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   F_XYZ,F_Angle,F_Force,F_Displ: FileR;
   F_Fd: FileByte;
   F_NewOld,F_Elem: FileI;
begin
{
      SOLVE NoWait
         NoWait causes the process to run without waiting.
      SOLVE Printer
         Printer sends the execution profile to the printer.
      SOLVE Printer NoWait
         Parameters can be simultaneous and interchanged.
}
   BatchLogo;
   ClrScr;
   TextColor(LightRed);
   Writeln('                    Solve for Displacements and Forces');
```

```
   TextColor(LightGreen);
   Parameters;
{
                Read Directories and Status
}
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
            StatusPrep,StatusForce,StatusDispl,StatusWorst,
            StatusFiberStrain,StatusFiberStress,
            StatusElemStrain,StatusElemStress,
            StatusGlobalStrain,StatusGlobalStress);
   TextColor(LightGreen);
   write('Loading geometry files for ');
   TextColor(LightRed);
   writeln(CaseName);
   TextColor(LightGreen);
   Get_I('Net.P',Net);
   Get_I('Nnt.P',Nnt);
   Get_I('Dim.P',Dim);
   Get_I('NneMax.P',NneMax);
   Get_Mat(Mat);
   Assign(F_xyz,C_Dir+'xyz.P');
   Assign(F_Force,C_Dir+'Force.P');
   Assign(F_Displ,C_Dir+'Displ.P');
   Assign(F_Fd,C_Dir+'Fd.P');
   Assign(F_Elem,C_Dir+'Elem.P');
   Assign(F_Angle,C_Dir+'Angle.P');
   Assign(F_NewOld,C_Dir+'NewToOld.P');
   Reset(F_xyz);
   Reset(F_Force);
   Reset(F_Displ);
   Reset(F_Fd);
   Reset(F_Elem);
   Reset(F_Angle);
   Reset(F_NewOld);
   Solve(FrontNodeSize,F_XYZ,F_Force,F_Displ,F_Fd,Net,Nnt,NneMax,Dim,
        F_Elem,F_Angle,Mat,CaseName);
   If NoWait=False then Wait;
{
                Display results
}
   Display_F_d(Nnt,Dim,F_NewOld,F_Force,F_Displ);
{
                close files
}
   Close(F_xyz);
   Close(F_Force);
   Close(F_Displ);
   Close(F_Fd);
   Close(F_Elem);
   Close(F_Angle);
   Close(F_NewOld);
```

```
{
                Correct Status
}
   StatusForce:='Done';
   StatusDispl:='Done';
   Write_Status(CaseName,CoarseCaseName,
                StatusPrep,StatusForce,StatusDispl,StatusWorst,
                StatusFiberStrain,StatusFiberStress,
                StatusElemStrain,StatusElemStress,
                StatusGlobalStrain,StatusGlobalStress);
   If NoWait=False then WaitIfNo;
   ClrScr;
end.
```

SONORE.PAS

```
{$N+}
Unit sonore;
Interface
Uses
   Crt;
Procedure Alert;
Procedure WarningSound;
Procedure Beep;
Procedure Bad_Beep;
{-------------------------------------------------------------------}
Implementation
Procedure Alert;
{
                  Beep to call the user which has wandered
}
var
   iSound: Byte;
begin
   For iSound:=1 to 10 do
      begin
      Sound(3000); Delay(50);
      NoSound;     Delay(50);
      end;
end;
{-------------------------------------------------------------------}
Procedure WarningSound;
{
                  Give a warning sound
}
var
   iSound: Byte;
begin
   For iSound:=1 to 3 do
      begin
      Sound(3000); Delay(50);
      NoSound;     Delay(50);
      end;
end;
{-------------------------------------------------------------------}
procedure Beep;
begin
   Sound(3300);
   Delay(5);
   NoSound;
end;
{-------------------------------------------------------------------}
procedure Bad_Beep;
var
   i: integer;
begin
```

```
   For i:=1 to 5 do
      begin
      Sound(3300);
      Delay(100);
      NoSound;
      Delay(200);
      end;
end;
{-------------------------------------------------------------}
end.
```

STATUS.PAS

```
{$N+}
Unit Status;
Interface
Uses
   Declare;
Procedure Read_Status(var CaseName,CoarseCaseName: String255;
                      var StatusPrep,StatusForce,StatusDispl,StatusWorst,
                         StatusFiberStrain,StatusFiberStress,
                         StatusElemStrain,StatusElemStress,
                         StatusGlobalStrain,StatusGlobalStress: String12);
Procedure Write_Status(var CaseName,CoarseCaseName: String255;
                       var StatusPrep,StatusForce,StatusDispl,StatusWorst,
                          StatusFiberStrain,StatusFiberStress,
                          StatusElemStrain,StatusElemStress,
                          StatusGlobalStrain,StatusGlobalStress: String12);
{-------------------------------------------------------------------}
Implementation
Procedure Read_Status;
{
Procedure Read_Status(var CaseName,CoarseCaseName: String255;
                      var StatusPrep,StatusForce,StatusDispl,StatusWorst,
                         StatusFiberStrain,StatusFiberStress,
                         StatusElemStrain,StatusElemStress,
                         StatusGlobalStrain,StatusGlobalStress: String12);
}
{
   Input:
     File "Status.P": contains Name and Status of solving steps
     C_Dir: Current directory
   Output:
     CaseName: File name being considered (Without extension)
     CoarseCaseName: Solved case used as a coarse mesh for refinement
     StatusPrep,StatusForce,StatusDispl,StatusWorst,
        StatusFiberStrain,StatusFiberStress,
        StatusElemStrain,StatusElemStress,
        StatusGlobalStrain,StatusGlobalStress: "Done", " ", "Error"
}
var
   i: Byte;
   F: Text;
begin
   Assign(F,C_Dir+'Status.P');
   {$I-}
   Reset(F);
   {$I+}
   If IoResult=0 then
     begin
     Readln(F,CaseName);
     Readln(F,CoarseCaseName);
     Readln(F,StatusPrep);
```

```
      Readln(F,StatusForce);
      Readln(F,StatusDispl);
      Readln(F,StatusWorst);
      Readln(F,StatusFiberStrain);
      Readln(F,StatusFiberStress);
      Readln(F,StatusElemStrain);
      Readln(F,StatusElemStress);
      Readln(F,StatusGlobalStrain);
      Readln(F,StatusGlobalStress);
      Close(F);
     end
   else
      begin
      CaseName:='Unknown';
      CoarseCaseName:='Unknown';
      StatusPrep:='';
      StatusForce:='';
      StatusDispl:='';
      StatusWorst:='';
      StatusFiberStrain:='';
      StatusFiberStress:='';
      StatusElemStrain:='';
      StatusElemStress:='';
      StatusGlobalStrain:='';
      StatusGlobalStress:='';
      Write_Status(CaseName,CoarseCaseName,
                   StatusPrep,StatusForce,StatusDispl,StatusWorst,
                   StatusFiberStrain,StatusFiberStress,
                   StatusElemStrain,StatusElemStress,
                   StatusGlobalStrain,StatusGlobalStress);
      end;
end;
{-------------------------------------------------------------------------}
Procedure Write_Status;
{
Procedure Write_Status(var CaseName,CoarseCaseName: String255;
                       var StatusPrep,StatusForce,StatusDispl,StatusWorst,
                           StatusFiberStrain,StatusFiberStress,
                           StatusElemStrain,StatusElemStress,
                           StatusGlobalStrain,StatusGlobalStress: String12);
}
{
   Input:
      CaseName: File name being considered (Without extension)
      CoarseCaseName: Solved case used as a coarse mesh for refinement
      StatusPrep,StatusForce,StatusDispl,StatusWorst,
        StatusFiberStrain,StatusFiberStress,
        StatusElemStrain,StatusElemStress,
        StatusGlobalStrain,StatusGlobalStress: "Done", " ", "Error"
      C_Dir: Current directory
   Output:
      File "Status.P": contains Name and Status of solving steps
```

```
)
var
   i: Byte;
   F: Text;
begin
   Assign(F,C_Dir+'Status.P');
   Rewrite(F);
   writeln(F,CaseName);
   writeln(F,CoarseCaseName);
   Writeln(F,StatusPrep);
   Writeln(F,StatusForce);
   Writeln(F,StatusDisp );
   Writeln(F,StatusWorst);
   Writeln(F,StatusFiberStrain);
   Writeln(F,StatusFiberStress);
   Writeln(F,StatusElemStrain);
   Writeln(F,StatusElemStress);
   Writeln(F,StatusGlobalStrain);
   Writeln(F,StatusGlobalStress);
   Close(F);
end;
{-------------------------------------------------------------------------}
end.
```

STRESS.PAS

```
{-------------------------------------------------------------------
  Program to calculate Local strain and stress
                       Global  —  —   —
  Initially written in January '88 by Jerome Daoust
-------------------------------------------------------------------}
{$N+}
{$M 30000,0,655360}
Uses
  Crt,Declare,Displace,Timer,Nne_Dim,File_RW,Exyz,Linear,Sonore,Logo,
  Nombre,WaitKey,Where,Status,Get,Param,WhatKey,Elem1,Elem2;
{-------------------------------------------------------------------}
Procedure AskSave(Var StatusFiberStrain,StatusFiberStress,
                     StatusElemStrain,StatusElemStress,
                     StatusGlobalStrain,StatusGlobalStress: String12;
                  Var SaveFiberStrain,SaveFiberStress,
                     SaveElemStrain,SaveElemStress,
                     SaveGlobalStrain,SaveGlobalStress: Boolean);
{
  Output:
    StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: "Done", " ", "Error"
    SaveFiberStrain: True if Strain in material direction is to be saved
    SaveFiberStress: True if Stress in material direction is to be saved
    SaveElemStrain: True if Strain in element direction is to be saved
    SaveElemStress: True if Stress in element direction is to be saved
    SaveGlobalStrain: True if Strain in global direction is to be saved
    SaveGlobalStress: True if Stress in global direction is to be saved
}
Var
  Cx,Cy,Row,NewRow,OldRow: Byte;
  Ans: array[1..10] of String[5];
  C: Char;
  Action: String6;
begin
  Cx:=31;
  Cy:=10;
  writeln('<F2> to change, <F10> to accept, <Esc> to exit, <arrows> to move');
  GotoXY(1,Cy);
  writeln('Do you want to save:');
  writeln('Strain in material direction? ');
  writeln('Stress in material direction? ');
  writeln('Strain in element  direction? ');
  writeln('Stress in element  direction? ');
  writeln('Strain in global   direction? ');
  writeln('Stress in global   direction? ');
  For row:=1 to 6 do
    Ans[row]:='No ';
  If StatusFiberStrain='Done'  then Ans[1]:='Done';
  If StatusFiberStress='Done'  then Ans[2]:='Done';
```

```
    If StatusElemStrain='Done'   then Ans[3]:='Done';
    If StatusElemStress='Done'   then Ans[4]:='Done';
    If StatusGlobalStrain='Done' then Ans[5]:='Done';
    If StatusGlobalStress='Done' then Ans[6]:='Done';
    TextColor(LightRed);
    For row:=1 to 6 do
       begin
       GotoXY(Cx,Cy+Row);
       write(Ans[row]);
       end;
    TextColor(LightGreen);
    NewRow:=1;
    TextColor(Black);
    TextBackground(LightRed);
    GotoXY(Cx,Cy   wRow);
    write(Ans[NewRow]);
    TextColor(LightGreen);
    TextBackground(Black);
    Action:='NUL';
    While (Action<>'F10') and (Action<>'ESC') do
       begin
       OldRow:=NewRow;
       Get_Key(C,Action);
       C:=UpCase(C);
       If (Action='UP') and (1<OldRow) then NewRow:=OldRow-1;
       If (Action='DOWN') and (OldRow<6) then NewRow:=OldRow+1;
       If (Action='HOME') and (1<OldRow) then NewRow:=1;
       If (Action='END') and (OldRow<6) then NewRow:=6;
       If (Action='PGUP') and (1<OldRow) then NewRow:=1;
       If (Action='PGDN') and (OldRow<6) then NewRow:=6;
       If Action='F2' then
          begin
          If Ans[OldRow]='Yes' then
             Ans[OldRow]:='No '
          else
             If Ans[OldRow]='No ' then Ans[OldRow]:='Yes';
          end;
       TextColor(LightRed);
       GotoXY(Cx,Cy+OldRow);
       write(Ans[Oldrow]);
       TextColor(LightGreen);
       TextColor(Black);
       TextBackground(LightRed);
       GotoXY(Cx,Cy+NewRow);
       write(Ans[NewRow]);
       TextColor(LightGreen);
       TextBackground(Black);
       end;
{
                  Interpret answers
}
   If Action='ESC' then
```

```
      For row:=1 to 6 do
         Ans[row]:='No ';
   SaveFiberStrain:=False;
   SaveFiberStress:=False;
   SaveElemStrain:=False;
   SaveElemStress:=False;
   SaveGlobalStrain:=False;
   SaveGlobalStress:=False;
   If Ans[1]='Yes' then SaveFiberStrain:=True;
   If Ans[2]='Yes' then SaveFiberStress:=True;
   If Ans[3]='Yes' then SaveElemStrain:=True;
   If Ans[4]='Yes' then SaveElemStress:=True;
   If Ans[5]='Yes' then SaveGlobalStrain:=True;
   If Ans[6]='Yes' then SaveGlobalStress:=True;
   ClrScr;
end;
{-------------------------------------------------------------------------}
Procedure Strain_Stress(    Nnt,Dim:IType;
                        var F_Displ: FileR;
                        var F_XYZ: FileR;
                            Net,NneMax: IType;
                        var F_Elem: FileI;
                        var Mat: MatrixR_Nmatx6;
                        var F_Angle: FileR;
                        var F_Fstrain,F_Fstress,F_Estrain,F_Estress,
                               F_Gstrain,F_Gstress: FileR;
                        var SaveFiberStrain,SaveFiberStress,
                               SaveElemStrain,SaveElemStress,
                               SaveGlobalStrain,SaveGlobalStress: Boolean);
{
   Input:
      Nnt: number of nodes;
      Dim: Dimensions in geometry
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
      Net: Number of elements;
      NneMax: Maximum number of nodes in an element
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Node are negative in Element matrix when they are use for
              the last time according to the numbering of the elements
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      F_Used: File giving number of time a node is used
      F_Angle: File With angle of elements (radians)
      SaveFiberStrain: True if Strain in material direction is to be saved
      SaveFiberStress: True if Stress in material direction is to be saved
      SaveElemStrain: True if Strain in element direction is to be saved
      SaveElemStress: True if Stress in element direction is to be saved
      SaveGlobalStrain: True if Strain in global direction is to be saved
      SaveGlobalStress: True if Stress in global direction is to be saved
```

```
   Output:
      F_Fstrain: File with material direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Fstress: File with material direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Estrain: File with element direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Estress: File with element direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
      F_Gstrain: File with Global direction Strain[1..N_Stress(Dim).Nnt]:
                 e1 e2 e3 ...e23  for each node
      F_Gstress: File with Global direction Stress[1..N_Stress(Dim).Nnt]:
                 s1 s2 s3 ...s23  for each node
}
var
   F_Used: FileByte;
   Nne,i,j,Nstress,Etype,iE,node,iWholePos,iElemPos: IType;
   iNne,iNstress: Byte;
   Fstrain,Fstress,Estrain,Estress,Gstrain,Gstress,Angle: Rtype;
   FiberStrain,FiberStress,ElemStrain,ElemStress,
     GlobalStrain,GlobalStress: VectR_NstressNne;
   Elem: VectI_Nne2;
   Used: Byte;
   Exyz: MatrixR_Nnex3;
   Xs,Ys,Xs2,Ys2: integer;
   ZeroR: RType;
   CalcTime: Rtype;
   Ns: String255;
   Node_Used: array[1..Nne_max] of Byte;
   Node_FilePos: array[1..Nne_max,1..Nstress_max] of Itype;
   Node_SSpos: array[1..Nne_max,1..Nstress_max] of word;
begin
   ResetTimer(1);
   StartTimer(1);
   write('Initializing ');
   Xs:=WhereX;
   Ys:=WhereY;
   ZeroR:=0;
   Nstress:=N_Stress(Dim);
{
                 Files are initialize one after the other to
                 reduce disk acces time
}
   If SaveFiberStrain=True then
      begin
      gotoXY(Xs,Ys);
      write('Fiber Strain ');
      Xs2:=WhereX;
      Ys2:=WhereY;
      Seek(F_Fstrain,0);
      For i:=1 to Nnt do
         begin
```

```
      gotoXY(Xs2,Ys2);
      write(Nnt-i+1,' ');
      For j:=1 to NStress do
        Write(F_Fstrain,ZeroR);
      end;
   end;
If SaveFiberStress=True then
   begin
   gotoXY(Xs,Ys);
   write('Fiber Stress ');
   Xs2:=WhereX;
   Ys2:=WhereY;
   Seek(F_Fstress,0);
   For i:=1 to Nnt do
      begin
      gotoXY(Xs2,Ys2);
      write(Nnt-i+1,' ');
      For j:=1 to NStress do
        Write(F_Fstress,ZeroR);
      end;
   end;
If SaveElemStrain=True then
   begin
   gotoXY(Xs,Ys);
   write('Element Strain ');
   Xs2:=WhereX;
   Ys2:=WhereY;
   Seak(F_Estrain,0);
   For i:=1 to Nnt do
      begin
      gotoXY(Xs2,Ys2);
      write(Nnt-i+1,' ');
      For j:=1 to NStress do
        Write(F_Estrain,ZeroR);
      end;
   end;
If SaveElemStress=True then
   begin
   gotoXY(Xs,Ys);
   write('Element Stress ');
   Xs2:=WhereX;
   Ys2:=WhereY;
   Seek(F_Estress,0);
   For i:=1 to Nnt do
      begin
      gotoXY(Xs2,Ys2);
      write(Nnt-i+1,' ');
      For j:=1 to NStress do
        Write(F_Estress,ZeroR);
      end;
   end;
If SaveGlobalStrain=True then
```

```
      begin
      gotoXY(Xs,Ys);
      write('Global Strain ');
      Xs2:=WhereX;
      Ys2:=WhereY;
      Seek(F_Gstrain,0);
      For i:=1 to Nnt do
         begin
         gotoXY(Xs2,Ys2);
         write(Nnt-i+1,' ');
         For j:=1 to NStress do
            Write(F_Gstrain,ZeroR);
         end;
      end;
   If SaveGlobalStress=True then
      begin
      gotoXY(Xs,Ys);
      write('Global Stress ');
      Xs2:=WhereX;
      Ys2:=WhereY;
      Seek(F_Gstress,0);
      For i:=1 to Nnt do
         begin
         gotoXY(Xs2,Ys2);
         write(Nnt-i+1,' ');
         For j:=1 to NStress do
            Write(F_Gstress,ZeroR);
         end;
      end;
{
                  erase 0 countdown number
}
   gotoXY(Xs,Ys);
   ClrEOL;
   writeln;
   Xs:=WhereX;
   Ys:=WhereY;
{
                  Get Averaged strain and stress at each node
                  in Local (Element) and Global Coordinate system
}
   Assign(F_Used,C_Dir+'Used.P');
   Reset(F_Used);
   For iE:=1 to Net do
      begin
      gotoXY(Xs,Ys);
      write(Net-iE+1,' ');
      Nne2_I_Read(F_Elem,iE,NneMax,Elem);
      R_Read(F_Angle,iE,Angle);
      Etype:=Elem[2];
      Nne:=Get_Nne(Etype);
      Get_Exyz(Elem,F_xyz,Exyz);
```

```
      case Etype of
         1: SS1(F_Displ,Exyz,Elem,Mat,FiberStrain,FiberStress,
               ElemStrain,ElemStress,GlobalStrain,GlobalStress);
         2: SS2(F_Displ,Exyz,Elem,Mat,Angle,FiberStrain,FiberStress,
               ElemStrain,ElemStress,GlobalStrain,GlobalStress);
         end;
{
                  Get all pointers for files
}
      For iNne:=1 to Nne do
         begin
         node:=abs(Elem[iNne+2]);
         Byte_Read(F_Used,node,Node_Used[iNne]);
         For iNstress:=1 to Nstress do
            begin
                 { First file position is 0 }
            Node_FilePos[iNne,iNstress]:= Nstress*(node-1)+iNstress-1;
            Node_SSpos[iNne,iNstress] := Nstress*(iNne-1)+iNstress;
            end;
         end;
{
                  Get the average at the nodes
                  To increase writing speed, all operation on a
                     strain-stress file is done before the next
                  Read old values of Local and Global Strain and Stress
                  Add this element's contribution to averaged node value
                  Write new values of Local and Global Strain and Stress
}
      If SaveFiberStrain=True then
         For iNne:=1 to Nne do
            begin
            For iNstress:=1 to Nstress do
               begin
               Seek(F_Fstrain,Node_FilePos[iNne,iNstress]);
               Read(F_Fstrain,Fstrain);
               Fstrain:=Fstrain
                      +ElemStrain[Node_SSpos[iNne,iNstress]]/Node_Used[iNne];
               Seek(F_Fstrain,Node_FilePos[iNne,iNstress]);
               Write(F_Fstrain,Fstrain);
               end;
            end;
      If SaveFiberStress=True then
         For iNne:=1 to Nne do
            begin
            For iNstress:=1 to Nstress do
               begin
               Seek(F_Fstress,Node_FilePos[iNne,iNstress]);
               Read(F_Fstress,Fstress);
               Fstress:=Fstress
                      +ElemStress[Node_SSpos[iNne,iNstress]]/Node_Used[iNne];
               Seek(F_Fstress,Node_FilePos[iNne,iNstress]);
               Write(F_Fstress,Fstress);
```

```
        end;
      end;
If SaveElemStrain=True then
   For iNne:=1 to Nne do
      begin
      For iNstress:=1 to Nstress do
         begin
         Seek(F_Estrain,Node_FilePos[iNne,iNstress]);
         Read(F_Estrain,Estrain);
         Estrain:=Estrain
                +ElemStrain[Node_SSpos[iNne,iNstress]]/Node_Used[iNne];
         Seek(F_Estrain,Node_FilePos[iNne,iNstress]);
         Write(F_Estrain,Estrain);
         end;
      end;
If SaveElemStress=True then
   For iNne:=1 to Nne do
      begin
      For iNstress:=1 to Nstress do
         begin
         Seek(F_Estress,Node_FilePos[iNne,iNstress]);
         Read(F_Estress,Estress);
         Estress:=Estress
                +ElemStress[Node_SSpos[iNne,iNstress]]/Node_Used[iNne];
         Seek(F_Estress,Node_FilePos[iNne,iNstress]);
         Write(F_Estress,Estress);
         end;
      end;
If SaveGlobalStrain=True then
   For iNne:=1 to Nne do
      begin
      For iNstress:=1 to Nstress do
         begin
         Seek(F_Gstrain,Node_FilePos[iNne,iNstress]);
         Read(F_Gstrain,Gstrain);
         Gstrain:=Gstrain
                +GlobalStrain[Node_SSpos[iNne,iNstress]]/Node_Used[iNne];
         Seek(F_Gstrain,Node_FilePos[iNne,iNstress]);
         Write(F_Gstrain,Gstrain);
         end;
      end;
If SaveGlobalStress=True then
   For iNne:=1 to Nne do
      begin
      For iNstress:=1 to Nstress do
         begin
         Seek(F_Gstress,Node_FilePos[iNne,iNstress]);
         Read(F_Gstress,Gstress);
         Gstress:=Gstress
                +GlobalStress[Node_SSpos[iNne,iNstress]]/Node_Used[iNne];
         Seek(F_Gstress,Node_FilePos[iNne,iNstress]);
         Write(F_Gstress,Gstress);
```

```
            end;
          end;
      end;
   Close(F_Used);
{
                Erase countdown message
}
   gotoXY(1,Ys);
   write('                                                    ');
   gotoXY(1,Ys);
   StopTimer(1);
   CalcTime:=ReadTimer(1);
   Str(CalcTime:20:2,Ns);
   While Ns[1]=' ' do Delete(Ns,1,1);
   writeln(Ns,'s for strain and stress calculations');
   If 30<CalcTime then Alert;
end;
{-------------------------------------------------------------------}
Procedure Display_eS(    Nnt,Dim: IType;
                     var F_Strain,F_Stress: FileR;
                     var F_NewOld: FileI);
{
   Input:
      Nnt: number of nodes
      Dim: Dimensions in geometry
      F_NewOld: original node number given by user for each "compacted" node #
      F_Strain: File with Strain[1..N_Stress(Dim).Nnt]:
                e1 e2 e3 ...e23  for each node
      F_Stress: File with Stress[1..N_Stress(Dim).Nnt]:
                s1 s2 s3 ...s23  for each node
   Output:
      Display strain and stress at each node
}
var
   Old,Nstress,i,j: IType;
   Strain,Stress: Rtype;
   StopNnt: Itype;
begin
   Nstress:=N_Stress(Dim);
   If Nnt<=20 then StopNnt:=Nnt
              Else StopNnt:=20;
   writeln('  ',StopNnt,' first nodes');
   If Dim=2 then
      writeln('strain:  e1          e2          e12')
   else
      writeln('strain:  e1          e2          e3          e12          e13          e23');
   Seek(F_Strain,0);
   For i:=1 to StopNnt do
      begin
      I_Read(F_NewOld,i,Old);
      write(Old:3,': ');
      For j:=1 to Nstress do
```

```
         begin
         read(F_Strain,Strain);
         write(RField(Strain,11),' ');
         end;
      writeln;
      end;
   If NoWait=False then Wait;
   writeln('  ',StopNnt,' first nodes');
   If Dim=2 then
      writeln('stress:  S1          S2         S12')
   else
      writeln('stress:  S1          S2          S3         S12         S13         S23');
   Seek(F_Stress,0);
   For i:=1 to StopNnt do
      begin
      I_Read(F_NewOld,i,Old);
      write(Old:3,': ');
      For j:=1 to Nstress do
         begin
         read(F_Stress,Stress);
         write(RField(Stress,11),' ');
         end;
      writeln;
      end;
end;
{-------------------------------------------------------------------}
var
   NnsMax,Nnt,Dim,Net: IType;
   Mat: MatrixR_Nmatx6;
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   F_XYZ,F_Angle,F_Displ,
      F_Fstrain,F_Fstress,F_Estrain,F_Estress,F_Gstrain,F_Gstress: FileR;
   F_NewOld,F_Elem: FileI;
   SaveFiberStrain,SaveFiberStress,
      SaveElemStrain,SaveElemStress,
      SaveGlobalStrain.SaveGlobalStress: Boolean;
begin
{
      STRESS [FiberStrain] [FiberStress] [ElemStrain] [ElemStress]
             [GlobalStrain] [GlobalStress]
         The process runs without waiting when a parameter is passed.
         FiberStrain:  For calculating strain in the material direction.
         FiberStress:  For calculating stress in the material direction.
         ElemStrain:   For calculating strain in the element  direction.
         ElemStress:   For calculating stress in the element  direction.
         GlobalStrain: For calculating strain in the global   direction.
         GlobalStress: For calculating stress in the global   direction.
         Parameters can be simultaneous and interchanged.
```

```
}
   BatchLogo;
   ClrScr;
   Parameters;
{
                  Read Directories and Status
}
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   If ParamCount=0 then
      begin
      AskSave(StatusFiberStrain,StatusFiberStress,
              StatusElemStrain,StatusElemStress,
              StatusGlobalStrain,StatusGlobalStress,
              SaveFiberStrain,SaveFiberStress,
              SaveElemStrain,SaveElemStress,
              SaveGlobalStrain,SaveGlobalStress);
      end
   else
      begin
      NoWait:=True;
      SaveFiberStrain:=False;
      SaveFiberStress:=False;
      SaveElemStrain:=False;
      SaveElemStress:=False;
      SaveGlobalStrain:=False;
      SaveGlobalStress:=False;
      For iParam:=1 to ParamCount do
         begin
         If ParameterString[iParam]='FIBERSTRAIN' then SaveFiberStrain:=True;
         If ParameterString[iParam]='FIBERSTRESS' then SaveFiberStress:=True;
         If ParameterString[iParam]='ELEMSTRAIN' then SaveElemStrain:=True;
         If ParameterString[iParam]='ELEMSTRESS' then SaveElemStress:=True;
         If ParameterString[iParam]='GLOBALSTRAIN' then SaveGlobalStrain:=True;
         If ParameterString[iParam]='GLOBALSTRESS' then SaveGlobalStress:=True;
         end;
      end;
{
                  Avoid re-writting a stress file
}
   If StatusFiberStrain ='Done' then SaveFiberStrain :=False;
   If StatusFiberStress ='Done' then SaveFiberStress :=False;
   If StatusElemStrain  ='Done' then SaveElemStrain  :=False;
   If StatusElemStress  ='Done' then SaveElemStress  :=False;
   If StatusGlobalStrain='Done' then SaveGlobalStrain:=False;
   If StatusGlobalStress='Done' then SaveGlobalStress:=False;
{
                  One strain or stress must be desired for process
```

```
                to be invoqued
}
   If (SaveFiberStrain or SaveFiberStress or
       SaveElemStrain or SaveElemStress or
       SaveGlobalStrain or SaveGlobalStress) then
      begin
      Write('                    Strain and Stress evaluation for ');
      TextColor(LightRed);
      writeln(CaseName);
      TextColor(LightGreen);
      TextColor(LightGreen);
      write('Loading geometry files for ');
      TextColor(LightRed);
      writeln(CaseName);
      TextColor(LightGreen);
      Get_I('Net.P',Net);
      Get_I('Nnt.P',Nnt);
      Get_I('Dim.P',Dim);
      Get_I('NneMax.P',NneMax);
      Get_Mat(Mat);
      Assign(F_xyz,C_Dir+'xyz.P');
      Assign(F_Fstrain,C_Dir+'Fstrain.P');
      Assign(F_Fstress,C_Dir+'Fstress.P');
      Assign(F_Estrain,C_Dir+'Estrain.P');
      Assign(F_Estress,C_Dir+'Estress.P');
      Assign(F_Gstrain,C_Dir+'Gstrain.P');
      Assign(F_Gstress,C_Dir+'Gstress.P');
      Assign(F_Displ,C_Dir+'Displ.P');
      Assign(F_Elem,C_Dir+'Elem.P');
      Assign(F_Angle,C_Dir+'Angle.P');
      Assign(F_NewOld,C_Dir+'NewToOld.P');
      Reset(F_xyz);
      Reset(F_Displ);
      Reset(F_Elem);
      Reset(F_Angle);
      Reset(F_NewOld);
      If SaveFiberStrain=True then Rewrite(F_Fstrain);
      If SaveFiberStress=True then Rewrite(F_Fstress);
      If SaveElemStrain=True then Rewrite(F_Estrain);
      If SaveElemStress=True then Rewrite(F_Estress);
      If SaveGlobalStrain=True then Rewrite(F_Gstrain);
      If SaveGlobalStress=True then Rewrite(F_Gstress);
{
                read displacements obtained before
}
      InitDisplHeapRAM(Dim,Nnt,F_Displ);
      Strain_Stress(Nnt,Dim,F_Displ,F_XYZ,Net,NneMax,F_Elem,Mat,F_Angle,
                    F_Fstrain,F_Fstress,F_Estrain,F_Estress,F_Gstrain,F_Gstress,
                    SaveFiberStrain,SaveFiberStress,
                    SaveElemStrain,SaveElemStress,
                    SaveGlobalStrain,SaveGlobalStress);
{
```

```
                Remove memory allocated for displacements
                already saved at the end of back substitution
                and not modified since,
                but kept until this point for speed
}
     DisposeDisplHeapRAM;
{
                Display results
}
     If (SaveGlobalStrain=True) and (SaveGlobalStress=True) then
        begin
        writeln('Global coordinate system:');
        Display_eS(Nnt,Dim,F_Gstrain,F_Gstress,F_NewOld);
        end;
{
                close files
}
     Close(F_xyz);
     If SaveFiberStrain=True  then Close(F_Fstrain);
     If SaveFiberStress=True  then Close(F_Fstress);
     If SaveElemStrain=True   then Close(F_Estrain);
     If SaveElemStress=True   then Close(F_Estress);
     If SaveGlobalStrain=True then Close(F_Gstrain);
     If SaveGlobalStress=True then Close(F_Gstress);
     Close(F_Displ);
     Close(F_Elem);
     Close(F_Angle);
     Close(F_NewOld);
{
                Correct Status
}
     If SaveFiberStrain=True then  StatusFiberStrain:='Done';
     If SaveFiberStress=True then  StatusFiberStress:='Done';
     If SaveElemStrain=True then   StatusElemStrain:='Done';
     If SaveElemStress=True then   StatusElemStress:='Done';
     If SaveGlobalStrain=True then StatusGlobalStrain:='Done';
     If SaveGlobalStress=True then StatusGlobalStress:='Done';
     Write_Status(CaseName,CoarseCaseName,
                StatusPrep,StatusForce,StatusDispl,StatusWorst,
                StatusFiberStrain,StatusFiberStress,
                StatusElemStrain,StatusElemStress,
                StatusGlobalStrain,StatusGlobalStress);
     If NoWait=False then WaitIfNo;
     end;
  ClrScr;
end.
```

SUB.PAS

```
{$N+}
Unit Sub;
Interface
Uses
   Crt,Declare,WaitKey;
Procedure InitSubHeapRAM(    Dim,FrontNodeSize: Itype);
Procedure DisposeSubHeapRAM;
Procedure DD_R_Read(var F: FileR;
                    var AllZero: Boolean;
                        FrontNodeSize,row,col,Dim: IType;
                    var S: MatrixR_DxD);
Procedure DD_R_Write(var F: FileR;
                         AllZero: Boolean;
                         FrontNodeSize,row,col,Dim: IType;
                     var S: MatrixR_DxD):                  {var for speed}
{-------------------------------------------------------------------------}
const
   MaxSubVectHeapRAM = 800;  {3.2 k max for pointers array}
                          {MaxSubVectHeapRAM*[4=SizeOf(SubVectHeapPtr)]}
{
                   In order to optimize speed of
                   Dim=1, Dim=2 and Dim=3   cases
                   set SubVectHeapLong as a product of 1*2*3*N
                   So a submatrix is all contained inside
                      a same heap vector
}
   SubVectHeapLong = 108; { (1^2*2^2*3^2)*3 }
                          {691200 Bytes max for heap storage of Rtype}
type
   SubVectHeap = array[1..SubVectHeapLong] of Rtype;
   SubVectHeapPtr = ^SubVectHeap;
var
   SubHeapRAM: array[1..MaxSubVectHeapRAM] of SubVectHeapPtr;
   SubNumberMaxRAM,SubNumberMax,LastSubVectHeapRAM: Itype;
{-------------------------------------------------------------------------}
Implementation
Procedure PosInRAM(    Dim,SubNumber: Itype;
                   var StartVect,PosInVect: Itype);
{
   Input:
      Dim: Dimension of problem
      SubNumber: SubMatrix number:
         1...N
      SubVectHeapLong (Constant): Length of vectors kept in Heap RAM
   Output:
      StartVect: Vector in which first element of SubMatrix is found:
         1...N
      PosInVect: position inside StartVect vector of first element:
         1...SubVectHeapLong
  Check:   1.2.3.4.5.6.   SubNelem=2
```

```
          1..2..3..4..   SubVectHeapLong=3
          1->1,1
          2->1,3
          3->2,2
          4->3,1
}
var
   SubNElem,PosMinus1: Itype;
begin
   Case Dim of
      1: SubNelem:=1;   (sqr(Dim))
      2: SubNelem:=4;
      3: SubNelem:=9;
      end;
   PosMinus1 := (SubNumber-1)*SubNelem;
   StartVect := Trunc(PosMinus1/SubVectHeapLong)+1;
   PosInVect := PosMinus1+1-(StartVect-1)*SubVectHeapLong;
end;
{-------------------------------------------------------------------------}
Procedure SubPlace(    FrontNodeSize,row,col,Dim: IType;
                   var Sub_Number,FilePoint: Itype;
                   var InRAM: Boolean);
{
   Input:
      FrontNodeSize: Maximum number of nodes in Matrix
      row,col: position of node in Matrix
      Dim: Dimension of problem
   Output:
      Sub_Number: SubMatrix number
      FilePoint: File position of first element of sub-matrix
      InRAM: True if Sub-matrix is kept in RAM
}
begin
{
                   Sub-Matrix arrangement:
                     ¦ k01 k02 k03 ¦
                     ¦ k11 k12 k13 ¦
                     ¦     k22 k23 ¦
                     ¦         k33 ¦
                   Order:
                      ¦ 1  2  3 ¦
                      ¦ 4  5  7 ¦
                      ¦    6  8 ¦
                      ¦       9 ¦
}
   If row=0 then
      Sub_Number:=Col
   else
      Sub_Number:=FrontNodeSize+Round((Col-1)/2*Col)+Row;
   If Sub_Number <= SubNumberMaxRAM then
      InRAM:=True
   else
```

```
      begin
      InRAM:=False;
{
                  0 based
                  Reduced file length for portion in RAM
}
      FilePoint:=(Sub_Number-1-SubNumberMaxRAM)*sqr(Dim);
      end;
end;
{------------------------------------------------------------------------}
Procedure InitSubHeapRAM;
{
Procedure InitSubHeapRAM(    Dim,FrontNodeSize: Itype);
}
{
   Input:
      Dim: Dimension of problem
      FrontNodeSize: Maximum number of nodes in Matrix
      SubVectHeapLong (Constant): Length of vectors kept in Heap RAM
      MaxSubVectHeapRAM (Constant): Number of vectors of length (SubVectHeapLong)
         kept in Heap RAM
   Output:
      LastSubVectHeapRAM: Number of vectors of length (SubVectHeapLong) kept in Heap RAM
      SubHeapRAM: Pointers for SubVectHeap kept in heap RAM
      SubNumberMaxRAM: maximum number of sub-matrices which can be stored in RAM
}
var
   i,FilePoint: Itype;
   InRAM: Boolean;
begin
   LastSubVectHeapRAM:=Trunc(MaxAvail/SubVectHeapLong/SizeOf(Rtype));
   If MaxSubVectHeapRAM<LastSubVectHeapRAM then
      LastSubVectHeapRAM:=MaxSubVectHeapRAM;
{
                  Get Maximum number of submatrices: SubNumberMax
}
   SubPlace(FrontNodeSize,FrontNodeSize,FrontNodeSize,Dim,
            SubNumberMax,FilePoint,InRAM);
{
                  Correct last heap vector number used with SubNumberMax
}
   If  Trunc(SubNumberMax*Dim*Dim/SubVectHeapLong)+1 < LastSubVectHeapRAM then
      LastSubVectHeapRAM := Trunc(SubNumberMax*Dim*Dim/SubVectHeapLong)+1;
{
      SubNumberMaxRAM: maximum number of sub-matrices which can be stored in RAM
}
   SubNumberMaxRAM:=Trunc(LastSubVectHeapRAM*SubVectHeapLong/Dim/Dim);
{
                  Correct maximum number of submatrices kept in RAM
                  with maximum submatrix number if inferior
}
   If SubNumberMax<SubNumberMaxRAM then SubNumberMaxRAM := SubNumberMax;
```

```
{
                Show Heap Status
}
   TextColor(LightRed);
   write(Trunc(LastSubVectHeapRAM*SubVectHeapLong*SizeOf(Rtype)/1024)+1);
   TextColor(LightGreen);
   writeln(' kB used (out of ',Trunc(MaxAvail/1024)+1,') for heap RAM storage');
   TextColor(LightRed);
   write((SubNumberMaxRAM/SubNumberMax)*100:5:1);
   TextColor(LightGreen);
   writeln('% of front matrix can be kept in RAM');
{
                Initialize Heap Pointers
                Dispose to Check that Pointers can be disposed
                  (instead of bugging at the complete end)
                  And it only takes 0.16s on a XT at 4.77 MHz
                Initialize Heap Pointers
}
   For i:=1 to LastSubVectHeapRAM do
      New(SubHeapRAM[i]);
   For i:=LastSubVectHeapRAM downto 1 do
      Dispose(SubHeapRAM[i]);
   For i:=1 to LastSubVectHeapRAM do
      New(SubHeapRAM[i]);
end;
{-------------------------------------------------------------------}
Procedure DisposeSubHeapRAM;
{
   Dispose of Heap Pointers
   Input:
      SubHeapRAM: Pointers for SubVectHeap kept in heap RAM
      LastSubVectHeapRAM: Number of vectors of length (SubVectHeapLong) kept in Heap RAM
}
var
   i: Integer;
begin
   For i:=LastSubVectHeapRAM downto 1 do  (reverse release)
      dispose(SubHeapRAM[i]);
end;
{-------------------------------------------------------------------}
Procedure DD_R_Read;
{
Procedure DD_R_Read(var F: FileR;
                    var AllZero: Boolean;
                        FrontNodeSize,row,col,Dim: IType;
                    var S: MatrixR_DxD);
}
{
   Input:
      F: File containing sub-matrices
      FrontNodeSize: Maximum number of nodes in Matrix
      row,col: position of node in Matrix
```

```
      Dim: Dimension of problem
   Output:
      AllZero: True if all elements of sub-matrix are 0
      S: Sub-Matrix
         Reads the S Sub-Matrix from a file
)
Var
   SubNumber,FilePoint,VectNumber,PosInVect: IType;
   InRAM: Boolean;
begin
   if row>col then
      begin
      writeln('bad sub-matrix read');
      wait;
      Halt;
      end;
   SubPlace(FrontNodeSize,row,col,Dim,SubNumber,FilePoint,InRAM);
   If InRAM=True then
      begin
      PosInRAM(Dim,SubNumber,VectNumber,PosInVect);
      S[1,1]:=SubHeapRAM[VectNumber]^[PosInVect];
      If S[1,1]>BigR then
         AllZero:=True
      else
         begin
         AllZero:=False;
{
                  The submatrix vector is all contained in one
                     heap vector thanks to the heap vector length being a
                     multiple of (1^2*2^2*3^2)=(1*2*3)^2=36.
                  See SubVectHeapLong = (1*4*9)*N in constant declaration.
}
         Case Dim of
            2: begin
               S[1,2]:=SubHeapRAM[VectNumber]^[PosInVect+1];
               S[2,1]:=SubHeapRAM[VectNumber]^[PosInVect+2];
               S[2,2]:=SubHeapRAM[VectNumber]^[PosInVect+3];
               end;
            3: begin
               S[1,2]:=SubHeapRAM[VectNumber]^[PosInVect+1];
               S[1,3]:=SubHeapRAM[VectNumber]^[PosInVect+2];
               S[2,1]:=SubHeapRAM[VectNumber]^[PosInVect+3];
               S[2,2]:=SubHeapRAM[VectNumber]^[PosInVect+4];
               S[2,3]:=SubHeapRAM[VectNumber]^[PosInVect+5];
               S[3,1]:=SubHeapRAM[VectNumber]^[PosInVect+6];
               S[3,2]:=SubHeapRAM[VectNumber]^[PosInVect+7];
               S[3,3]:=SubHeapRAM[VectNumber]^[PosInVect+8];
               end;
            end;
         end;
      end
   else
```

```
begin
Seek(F,FilePoint);
Read(F,S[1,1]);
If S[1,1]>BigR then
   AllZero:=True
else
   begin
   AllZero:=False;
   Case Dim of
      2: begin
         Read(F,S[1,2]);
         Read(F,S[2,1]);
         Read(F,S[2,2]);
         end;
      3: begin
         Read(F,S[1,2]);
         Read(F,S[1,3]);
         Read(F,S[2,1]);
         Read(F,S[2,2]);
         Read(F,S[2,3]);
         Read(F,S[3,1]);
         Read(F,S[3,2]);
         Read(F,S[3,3]);
         end;
      end;
   end;
end;
If AllZero=True then
   begin
   Case Dim of
      1: begin
         S[1,1]:=0;
         end;
      2: begin
         S[1,1]:=0;
         S[1,2]:=0;
         S[2,1]:=0;
         S[2,2]:=0;
         end;
      3: begin
         S[1,1]:=0;
         S[1,2]:=0;
         S[1,3]:=0;
         S[2,1]:=0;
         S[2,2]:=0;
         S[2,3]:=0;
         S[3,1]:=0;
         S[3,2]:=0;
         S[3,3]:=0;
         end;
      end;
   end;
```

```
end;
{-------------------------------------------------------------------}
Procedure DD_R_Write;
{
Procedure DD_R_Write(var F: FileR;
                         AllZero: Boolean;
                         FrontNodeSize,row,col,Dim: IType;
                     var S: MatrixR_DxD);       var for speed
}
{
   Input:
      F: File containing sub-matrices
      AllZero: True if all elements of sub-matrix are 0
      FrontNodeSize: Maximum number of nodes in Matrix
      row,col: position of node in Matrix
      Dim: Dimension of problem
      S: Sub-Matrix
   Output:
      Store the S Sub-Matrix into a file
}
var
   SubNumber,FilePoint,VectNumber,PosInVect: IType;
   X_Rtype: Rtype;
   InRAM: Boolean;
begin
   if row>col then
      begin
      writeln('bad sub-matrix write');
      wait;
      Halt;
      end;
   SubPlace(FrontNodeSize,row,col,Dim,SubNumber,FilePoint,InRAM);
   If InRAM=True then
      begin
      PosInRAM(Dim,SubNumber,VectNumber,PosInVect);
      If AllZero=True then
         SubHeapRAM[VectNumber]^[PosInVect] := BigRmax
      else
         begin
{
                  The submatrix vector is all contained in one
                     heap vector thanks to the heap vector length being a
                     multiple of (1^2*2^2*3^2)=(1*2*3)^2=36.
                  See SubVectHeapLong = (1*4*9)*N in constant declaration.
}
         Case Dim of
            1: begin
               SubHeapRAM[VectNumber]^[PosInVect] := S[1,1];
               end;
            2: begin
               SubHeapRAM[VectNumber]^[PosInVect] := S[1,1];
               SubHeapRAM[VectNumber]^[PosInVect+1] := S[1,2];
```

```
            SubHeapRAM[VectNumber]^[PosInVect+2] := S[2,1];
            SubHeapRAM[VectNumber]^[PosInVect+3] := S[2,2];
            end;
         3: begin
            SubHeapRAM[VectNumber]^[PosInVect] := S[1,1];
            SubHeapRAM[VectNumber]^[PosInVect+1] := S[1,2];
            SubHeapRAM[VectNumber]^[PosInVect+2] := S[1,3];
            SubHeapRAM[VectNumber]^[PosInVect+3] := S[2,1];
            SubHeapRAM[VectNumber]^[PosInVect+4] := S[2,2];
            SubHeapRAM[VectNumber]^[PosInVect+5] := S[2,3];
            SubHeapRAM[VectNumber]^[PosInVect+6] := S[3,1];
            SubHeapRAM[VectNumber]^[PosInVect+7] := S[3,2];
            SubHeapRAM[VectNumber]^[PosInVect+8] := S[3,3];
            end;
         end;
      end;
   end
else
   begin
   Seek(F,FilePoint);
   If AllZero=True then
      begin
      X_Rtype:=BigRmax;
      write(F,X_Rtype);
      end
   else
      begin
      Case Dim of
         1: write(F,S[1,1]);
         2: begin
            write(F,S[1,1]);
            write(F,S[1,2]);
            write(F,S[2,1]);
            write(F,S[2,2]);
            end;
         3: begin
            write(F,S[1,1]);
            write(F,S[1,2]);
            write(F,S[1,3]);
            write(F,S[2,1]);
            write(F,S[2,2]);
            write(F,S[2,3]);
            write(F,S[3,1]);
            write(F,S[3,2]);
            write(F,S[3,3]);
            end;
         end;
      end;
   end;
end;
{-------------------------------------------------------------------}
end.
```

SUM.PAS

```
{
                Program to help create a batch file for
                combined loading conditions using SUMTONEW
}
{$N+}
Uses
   Declare,Nombre;
var
   iS,iC,SF_A_max,Cmax: integer;
   SF: array[1..50,1..3] of Rtype;
   C: array[1..50,1..3] of integer;
   A: array[1..50] of string[2];
   F: Text;
begin
{
                Safety factors for each angle under:
                   Tension, Bending, Torsion
}
   SF[1,1]:= 1.704;  SF[1,2]:=0.000548;  SF[1,3]:=0.000283;
   SF[2,1]:= 1.690;  SF[2,2]:=0.000551;  SF[2,3]:=0.000278;
   SF_A_max:=2;
{
                Angles (Truncated)
                same row qtt as in [SF] matrix
}
   A[1]:='00';
   A[2]:='90';
{
                Combined loading conditions:
                048 means: No Tension
                           4/8 of maximum Bending
                           8/8 of maximum Torsion
}
   C[ 1,1]:=0;  C[ 1,2]:=8;  C[ 1,3]:=8;  {088}
   C[ 2,1]:=0;  C[ 2,2]:=4;  C[ 2,3]:=8;  {048}
   C[ 3,1]:=0;  C[ 3,2]:=8;  C[ 3,3]:=4;  {084}
   C[ 4,1]:=8;  C[ 4,2]:=0;  C[ 4,3]:=8;  {808}
   C[ 5,1]:=4;  C[ 5,2]:=0;  C[ 5,3]:=8;  {408}
   C[ 6,1]:=8;  C[ 6,2]:=8;  C[ 6,3]:=0;  {880}
   C[ 7,1]:=8;  C[ 7,2]:=8;  C[ 7,3]:=8;  {888}
   C[ 8,1]:=4;  C[ 8,2]:=4;  C[ 8,3]:=8;  {448}
   C[ 9,1]:=4;  C[ 9,2]:=8;  C[ 9,3]:=0;  {480}
   C[10,1]:=4;  C[10,2]:=8;  C[10,3]:=4;  {484}
   C[11,1]:=4;  C[11,2]:=8;  C[11,3]:=8;  {488}
   C[12,1]:=8;  C[12,2]:=0;  C[12,3]:=4;  {804}
   C[13,1]:=8;  C[13,2]:=4;  C[13,3]:=0;  {840}
   C[14,1]:=8;  C[14,2]:=4;  C[14,3]:=4;  {844}
   C[15,1]:=8;  C[15,2]:=4;  C[15,3]:=8;  {848}
   C[16,1]:=8;  C[16,2]:=8;  C[16,3]:=4;  {884}
   Cmax:=16;
```

```
   Assign(F,'C:\TP\FE\GEN\Sum.bat');
   Rewrite(F);
   for iS:=1 to SF_A_max do
      begin
      For iC:=1 to Cmax do
         begin
         Write(F,'SumToNew');
{
                                    EL[00][U]
}
         If 0<C[iC,1] then write(F,' EL',A[iS],'U ',SF[iS,1]*C[iC,1]/8:5:3);
         If 0<C[iC,2] then write(F,' EL',A[iS],'B ',SF[iS,2]*C[iC,2]/8:8:6);
         If 0<C[iC,3] then write(F,' EL',A[iS],'T ',SF[iS,3]*C[iC,3]/8:8:6);
         writeln(F,' Rien');
         writeln(F,'Manager Restore Rien');
{
                            Ec[00]c[888]
}
         writeln(F,'NewName Ec',A[iS],'c',C[iC,1]:1,C[iC,2]:1,C[iC,3]:1);
         writeln(F,'Worst NoWait Printer');
         writeln(F);
         end;
      writeln(F);
      writeln(F);
      writeln(F);
      end;
   Close(F);
end.
```

SUMTONEW.PAS

```
{-------------------------------------------------------------------------
   Program to sum force and displacements from many cases
   Initially written in May '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 2000,0,0}
Uses
   Crt,Dos,Declare,Param,WaitKey,Where,Nombre,Status,Get,File_RW,Nne_Dim;
{-------------------------------------------------------------------------}
{
      SUMTONEW CaseName1 Mult1 [CaseName2 Mult2 ...] NewCaseName
      This permits to add the solution of many cases:
         CaseName1 ... CaseNameN (without .GEN extension).
      The forces, displacements, strains and stresses of each case
         is correspondingly multyplied by Mult1 ... MultN before
         they are summed.
      Only the common strains and stresses to all cases are summed.
      Each case must be solved (force displacements) and saved with MANAGER.
      All cases summed must be saved in the same "Save Root" directory.
         (See \Main menu\MANAGER\)
      A single case can be used for the summation.
      NewCaseName is the case name to which results are summed.
         Using MANAGER will restore this case.
      The geometry of all cases must be the same.
      This process avoids the long solving time for a loading which is
         a linear combination of previous loading cases to a geometry.
}
Const
   Case_max = 20;
   Buf_max = 100;
var
   StressName,NewCaseName,OriginalCurrentCaseDir: String255;
   CaseNameVect: array[1..Case_max] of String255;
   CaseMultVect: array[1..Case_max] of Rtype;
   NStress,iFileStress,Xs,Ys,iD,IRtype,iCase,CaseQtt: Byte;
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
      StatusFiberStrain,StatusFiberStress,
      StatusElemStrain,StatusElemStress,
      StatusGlobalStrain,StatusGlobalStress: String12;
   NumberI: Itype;
   ZeroR,NumberR: Rtype;
   Force,Displ: VectR_D;
   i,j,Node,NodeOld,Nnt,Dim,NntAdd,DimAdd: Itype;
   CommonSS: array[1..10] of Boolean;
   ValueAdd: array[1..Buf_max] of VectR_D;
   SSadd: array[1..Buf_max,1..Nstress_max] of Rtype;
   SS: array[1..Nstress_max] of Rtype;
   F_SS,F_SSadd,F_Force,F_Displ,F_ForceAdd,F_DisplAdd: FileR;
begin
```

```
   TextColor(LightGreen);
   Parameters;
{
                  Read Directories and Status
                  Remember original C_Dir
}
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   OriginalCurrentCaseDir:=C_Dir;
{
                  Verify and interpret parameters
}
   If 2*Round(ParamCount/2)=ParamCount then
      begin
      writeln('Error: there must be an odd number of parameters.');
      Wait;
      Halt;
      end;
   CaseQtt:=Trunc(ParamCount/2);
   For iCase:=1 to CaseQtt do
      begin
      iParam:=(iCase-1)*2+1;
      StoX(ParameterString[iParam],NumberI,NumberR,IRtype);
      If IRtype<>0 then
         begin
         writeln('Error: parameter ',iParam,' must be a string.');
         Wait;
         Halt;
         end;
      CaseNameVect[iCase]:=ParameterString[iParam];
      StoX(ParameterString[iParam+1],NumberI,NumberR,IRtype);
      Case IRtype of
         0: begin
            writeln('Error: parameter ',iParam+1,' must be a number.');
            Wait;
            Halt;
            end;
         1: CaseMultVect[iCase]:=1.0*NumberI;
         2: CaseMultVect[iCase]:=NumberR;
         end;
      end;
   NewCaseName:=ParameterString[ParamCount];
{
                  Make shure that NewCaseName is a new case name
}
   For iCase:=1 to CaseQtt do
      begin
      If NewCaseName=CaseNameVect[iCase] then
         begin
         writeln('Error: NewCaseName must be a new case name');
         Wait;
         Halt;
         end;
```

```
      end;
{
                  Check that all CaseName to add have a:
                     StatusForce='Done'
                     StatusDispl='Done'
}
   For iCase:=1 to CaseQtt do
      begin
      C_Dir:=S_Dir+CaseNameVect[iCase]+'\';
      Read_Status(CaseName,CoarseCaseName,
                  StatusPrep,StatusForce,StatusDispl,StatusWorst,
                  StatusFiberStrain,StatusFiberStress,
                  StatusElemStrain,StatusElemStress,
                  StatusGlobalStrain,StatusGlobalStress);
      C_Dir:=OriginalCurrentCaseDir;
      If ((StatusPrep<>'Done') or
          (StatusForce<>'Done') or
          (StatusDispl<>'Done')) then
        begin
        Writeln('Case name ',CaseNameVect[iCase],' has not been solved or saved.');
        Wait;
        Halt;
        end;
      end;
{
                  Create directory
                  Don't use MkDir it bugs the program when the directory
                     already exist (second time around)
}
   writeln('Creating directory: ',S_Dir+NewCaseName);
   Exec('\COMMAND.COM','/C '+'MD '+S_Dir+NewCaseName);
{
                  Delete previous files in subdirectory
}
   writeln('Deleting previous files in new case directory');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\*.P');
{
                  Copy files from first CaseName
}
   writeln('Copying files from first case (',CaseNameVect[1],'):');
   Exec('\COMMAND.COM','/C '+'COPY '+S_Dir+CaseNameVect[1]+'\*.P'
                            +' '+S_Dir+NewCaseName+'\*.*');
{
                  Delete files:
                          strain and stress
                          worst element
                          status
                          force displacement
}
   Writeln('Deleting unnecessary files:');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Fstrain.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Fstress.P');
```

```
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Estrain.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Estress.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Gstrain.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Gstress.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\BadElem.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Status.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Force.P');
   Exec('\COMMAND.COM','/C '+'DEL '+S_Dir+NewCaseName+'\Displ.P');
{
                Verify that all cases to add have the same number of nodes
                and dimension
}
   writeln('Verifying case compatibility');
   C_Dir:=S_Dir+NewCaseName+'\',
   Get_I('Nnt.P',Nnt);
   Get_I('Dim.P',Dim);
   C_Dir:=OriginalCurrentCaseDir;
   For iCase:=2 to CaseQtt do      (First case is okay)
      begin
      C_Dir:=S_Dir+CaseNameVect[iCase]+'\';
      Get_I('Nnt.P',NntAdd);
      Get_I('Dim.P',DimAdd);
      C_Dir:=OriginalCurrentCaseDir;
      If Nnt<>NntAdd then
         begin
         writeln('Case name ',CaseNameVect[1],' and ',CaseName[iCase]);
         writeln('   do not have the same number of nodes.');
         Wait;
         Halt;
         end;
      If Dim<>DimAdd then
         begin
         writeln('Case name ',CaseNameVect[1],' and ',CaseName[iCase]);
         writeln('   do not have the same number of dimensions.');
         Wait;
         Halt;
         end;
      end;
{
                Look for common strain and stress files
}
   For iFileStress:=1 to 6 do
      CommonSS[iFileStress]:=True;
   For iCase:=1 to CaseQtt do
      begin
      C_Dir:=S_Dir+CaseNameVect[iCase]+'\';
      Read_Status(CaseName,CoarseCaseName,
                  StatusPrep,StatusForce,StatusDispl,StatusWorst,
                  StatusFiberStrain,StatusFiberStress,
                  StatusElemStrain,StatusElemStress,
                  StatusGlobalStrain,StatusGlobalStress);
      C_Dir:=OriginalCurrentCaseDir;
```

```
      If StatusFiberStrain<>'Done'  then CommonSS[1]:=False;
      If StatusFiberStress<>'Done'  then CommonSS[2]:=False;
      If StatusElemStrain<>'Done'   then CommonSS[3]:=False;
      If StatusElemStress<>'Done'   then CommonSS[4]:=False;
      If StatusGlobalStrain<>'Done' then CommonSS[5]:=False;
      If StatusGlobalStress<>'Done' then CommonSS[6]:=False;
      end;
{
                Set Status for NewCaseName
}
   writeln('Setting status for new case');
   CaseName:=NewCaseName;
   CoarseCaseName:='Unknown';
   StatusPrep:='Done';
   StatusForce:='Done';
   StatusDispl:='Done';
   StatusWorst:='';
   If CommonSS[1]=True then StatusFiberStrain:='Done'
                       else StatusFiberStrain:='';
   If CommonSS[2]=True then StatusFiberStress:='Done'
                       else StatusFiberStress:='';
   If CommonSS[3]=True then StatusElemStrain:='Done'
                       else StatusElemStrain:='';
   If CommonSS[4]=True then StatusElemStress:='Done'
                       else StatusElemStress:='';
   If CommonSS[5]=True then StatusGlobalStrain:='Done'
                       else StatusGlobalStrain:='';
   If CommonSS[6]=True then StatusGlobalStress:='Done'
                       else StatusGlobalStress:='';
   C_Dir:=S_Dir+NewCaseName+'\';
   Write_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
     Dir:=OriginalCurrentCaseDir;
{
                  Initialize Force and Displacement files
}
   Assign(F_Force,S_Dir+NewCaseName+'\Force.P');
   Assign(F_Displ,S_Dir+NewCaseName+'\Displ.P');
   Rewrite(F_Force);
   Rewrite(F_Displ);
   write('Initializing forces ');
   Xs:=WhereX;
   Ys:=WhereY;
   For Node:=1 to Nnt do
      begin
      gotoXY(Xs,Ys);
      write(Nnt-Node+1,' ');
      D_R_0(F_Force,Node,Dim);
      end;
```

```
   gotoXY(Xs,Ys);
   writeln(' ');
   write('Initializing displacements ');
   Xs:=WhereX;
   Ys:=WhereY;
   For Node:=1 to Nnt do
      begin
      gotoXY(Xs,Ys);
      write(Nnt-Node+1,' ');
      D_R_0(F_Displ,Node,Dim);
      end;
   gotoXY(Xs,Ys);
   writeln(' ');
{
                  Add force and displacements from cases
}
   For iCase:=1 to CaseQtt do
      begin
      writeln('   Adding case ',CaseNameVect[iCase]);
      Assign(F_ForceAdd,S_Dir+CaseNameVect[iCase]+'\Force.P');
      Assign(F_DisplAdd,S_Dir+CaseNameVect[iCase]+'\Displ.P');
      Reset(F_ForceAdd);
      Reset(F_DisplAdd);
{
                  Add forces
}
      write('      Forces ');
      Xs:=WhereX;
      Ys:=WhereY;
      NodeOld:=0;
      Node:=0;
      While Node<Nnt do
         begin
         GotoXY(Xs,Ys);
         write(Nnt-Node,' ');
         Inc(Node);
         D_R_read(F_ForceAdd,Node,Dim,ValueAdd[Node-NodeOld]);
         If ((Node-NodeOld=Buf_max) or (Node=Nnt)) then
            begin
            For i:=1 to Node-NodeOld do
               begin
               D_R_read(F_Force,NodeOld+i,Dim,Force);
               For iD:=1 to Dim do
                   Force[iD]:=Force[iD]+CaseMultVect[iCase]*ValueAdd[i][iD];
               D_R_write(F_Force,NodeOld+i,Dim,Force);
               end;
            NodeOld:=Node;
            end;
         end;
      GotoXY(Xs,Ys);
      writeln(' ');
{
```

```
                Add displacements
}
     write('      Displacements ');
     Xs:=WhereX;
     Ys:=WhereY;
     NodeOld:=0;
     Node:=0;
     While Node<Nnt do
        begin
        GotoXY(Xs,Ys);
        write(Nnt-Node,' ');
        Inc(Node);
        D_R_read(F_DisplAdd,Node,Dim,ValueAdd[Node-NodeOld]);
        If ((Node-NodeOld=Buf_max) or (Node=Nnt)) then
           begin
           For i:=1 to Node-NodeOld do
              begin
              D_R_read(F_Displ,NodeOld+i,Dim,Displ);
              For iD:=1 to Dim do
                  Displ[iD]:=Displ[iD]+CaseMultVect[iCase]*ValueAdd[i][iD];
              D_R_write(F_Displ,NodeOld+i,Dim,Displ);
              end;
           NodeOld:=Node;
           end;
        end;
     GotoXY(Xs,Ys);
     writeln(' ');
{
                Close case files
}
     Close(F_ForceAdd);
     Close(F_DisplAdd);
     end;
{
                Close new case files
}
  Close(F_Force);
  Close(F_Displ);
{
                Sum common strain or stress
}
  ZeroR:=0.0;
  Nstress:=N_Stress(Dim);
  For iFileStress:=1 to 6 do
     begin
     If CommonSS[iFileStress]=True then
        begin
{
                Initialize strain or stress files
}
        Case iFileStress of
           1: StressName:='Fstrain';
```

```
            2: StressName:='Fstress';
            3: StressName:='Estrain';
            4: StressName:='Estress';
            5: StressName:='Gstrain';
            6: StressName:='Gstress';
            end;
         Assign(F_SS,S_Dir+NewCaseName+'\'+StressName+'.p');
         Rewrite(F_SS);
         write('Initializing ',StressName,' ');
         Xs:=WhereX;
         Ys:=WhereY;
         For Node:=1 to Nnt do
            begin
            gotoXY(Xs,Ys);
            write(Nnt-Node+1,' ');
            For j:=1 to NStress do
               Write(F_SS,ZeroR);
            end;
         gotoXY(Xs,Ys);
         writeln(' ');
{
                  Add strain or stress from cases
}
         For iCase:=1 to CaseQtt do
            begin
            w ite('   Adding case ',CaseNameVect[iCase],' ');
            Xs:=WhereX;
            Ys:=WhereY;
            Assign(F_SSadd,S_Dir+CaseNameVect[iCase]+'\'+StressName+'.p');
            Reset(F_SSadd);
            Seek(F_SS,0);   (set file pointer to top of file)
            NodeOld:=0;
            Node:=0;
            While Node<Nnt do
               begin
               GotoXY(Xs,Ys);
               write(Nnt-Node,' ');
               Inc(Node);
               For j:=1 to NStress do
                  Read(F_SSadd,SSadd[Node-NodeOld,j]);
               If ((Node-NodeOld=Buf_max) or (Node=Nnt)) then
                  begin
                  For i:=1 to Node-NodeOld do
                     begin
                     For j:=1 to NStress do
                        begin
                        Read(F_SS,SS[j]);
                        SS[j]:=SS[j]+CaseMultVect[iCase]*SSadd[i,j];
                        end;
                     Seek(F_SS,FilePos(F_SS)-Nstress);
                     For j:=1 to NStress do write(F_SS,SS[j]);
                     end;
```

```
                NodeOld:=Node;
                end;
             end;
          GotoXY(Xs,Ys);
          writeln(' ');
{
                Close case files
}
          Close(F_SSadd);
          end;
{
                Close new case files
}
       Close(F_SS);
       end;
    end;
  GotoXY(1,WhereY-1);
end.
```

TIMER.PAS

```
{$N+}
Unit Timer;
Interface
Uses
   Crt,Declare,Dos;
Procedure ResetTimerAll;
Procedure ResetTimer(    iTimer: Byte);
Procedure StartTimer(    iTimer: Byte);
Procedure StopTimer(    iTimer: Byte);
Function ReadTimer(    iTimer: Byte): Rtype;
Function HourMinSec(    Seconds: Rtype): String80;
Procedure Show_Time(    Xcoord,Ycoord: integer);
{-------------------------------------------------------------------}
Implementation
Const
   TimerMax = 20;
Var
   SumDelayTimerSecR,StartTimerSecR: array[1..TimerMax] of Rtype;
   StartDate: array[1..TimerMax,1..3] of Word;
{-------------------------------------------------------------------}
Function TimeSecR: Rtype;
var
   Hour,Min,Sec,Sec100: word;
begin
{
                  Important notice:
                  The HOUR must be multiplied by 3600.0 and not 3600
                  otherwise the result stays in WORD type but can exceed
                  a WORD type integer (0-65520) and give an incorrect result
}
   GetTime(Hour,Min,Sec,Sec100);
   TimeSecR:=Hour*3600.0+Min*60.0+Sec+Sec100/100;
end;
{-------------------------------------------------------------------}
Procedure ResetTimerAll;
{
                  Reset to 0 all timers
}
var
   iT: Byte;
begin
   For iT:=1 to TimerMax do
      SumDelayTimerSecR[iT]:=0;
end;
{-------------------------------------------------------------------}
Procedure ResetTimer;
{
Procedure ResetTimer(    iTimer: Byte);
}
{
```

```
   Input:
      iTimer: timer number to be reset to 0
)
begin
   If TimerMax<iTimer then
      begin
      writeln('Bad timer number!');
      Halt;
      end;
   SumDelayTimerSecR[iTimer]:=0;
end;
(-------------------------------------------------------------------)
Procedure StartTimer;
(
Procedure StartTimer(    iTimer: Byte);
)
(
   Input:
      iTimer: timer number which is started
)
var
   Year,Month,Day,DayOfWeek: word;
begin
   StartTimerSecR[iTimer]:=TimeSecR;
   GetDate(Year,Month,Day,DayOfWeek);
   StartDate[iTimer,1]:=Year;
   StartDate[iTimer,2]:=Month;
   StartDate[iTimer,3]:=Day;
end;
(-------------------------------------------------------------------)
Function Bissextile(    Year: word): Boolean;
(
   Input:
      Year: year
   Output:
      True if this year has 29 days in february
      False if              28
)
begin
   If abs((Year/4)-Round(Year/4)) <= 1E-4 then
      Bissextile := True
   else
      Bissextile := False;
end;
(-------------------------------------------------------------------)
Procedure StopTimer;
(
Procedure StopTimer(    iTimer: Byte);
)
(
   Input:
      iTimer: timer number which is stopped
```

```
   Note: This algorithm is best suited for ellapsed times up to a month
}
var
   Y,M,D,Year,Month,Day,DayOfWeek: word;
   DaysPassed: LongInt;
   LastDayOfMonth: array[1..12] of Byte;
   DelayTimerSecR: Rtype;
begin
{
                  Added time can be negative if the day has changed
}
   DelayTimerSecR:=TimeSecR-StartTimerSecR[iTimer];
   SumDelayTimerSecR[iTimer]   := SumDelayTimerSecR[iTimer]
                               + DelayTimerSecR;
   GetDate(Year,Month,Day,DayOfWeek);
   If (Year=StartDate[iTimer,1]) and
      (Month=StartDate[iTimer,2]) and
      (Day=StartDate[iTimer,3]) then
      begin
      { nothing is done because date has not changed }
      end
   else
      begin
{
                  Find out how many days have passed
                  since last StartTimer(iTimer)
}
      Y:=StartDate[iTimer,1];
      M:=StartDate[iTimer 2];
      D:=StartDate[iTimer,3];
{
                  get last day of each month
}
      LastDayOfMonth[1]:=31;
      If Bissextile(Y) then
         LastDayOfMonth[2]:=29
      else
         LastDayOfMonth[2]:=28;
      LastDayOfMonth[3]:=31;
      LastDayOfMonth[4]:=30;
      LastDayOfMonth[5]:=31;
      LastDayOfMonth[6]:=30;
      LastDayOfMonth[7]:=31;
      LastDayOfMonth[8]:=31;
      LastDayOfMonth[9]:=30;
      LastDayOfMonth[10]:=31;
      LastDayOfMonth[11]:=30;
      LastDayOfMonth[12]:=31;
      DaysPassed:=0;
      While not ((Y=Year)and(M=Month)and(D=Day)) do
         begin
         Inc(DaysPassed);
```

```
        If D<LastDayOfMonth[M] then
           Inc(D)
        else
           begin
           Case Month of
              1..11: begin
                 D:=1;
                 Inc(M);
                 end;
              12: begin    { New Year }
                 D:=1;
                 M:=1;
                 Inc(Y);
                 If Bissextile(Y) then
                    LastDayOfMonth[2]:=29
                 else
                    LastDayOfMonth[2]:=28;
                 end;
              end;
           end;
        end;
{
                 Add time corresponding to number of days passed
}
      If 0<DaysPassed then
         SumDelayTimerSecR[iTimer] := SumDelayTimerSecR[iTimer]
                                    + DaysPassed*86400.0;  {1 day = 86400s}
{
                 Reset start date for this timer
}
      StartDate[iTimer,1]:=Year;
      StartDate[iTimer,2]:=Month;
      StartDate[iTimer,3]:=Day;
      end;
end;
{-------------------------------------------    ----------------}
Function ReadTimer;
{
Function ReadTimer(    iTimer: Byte): Rtype;
}
{
   Input:
      iTimer: timer number which is read
   Output:
      ReadTimer: time in secends which have passed since StartTimer[iTimer]
}
begin
   ReadTimer:=SumDelayTimerSecR[iTimer];
end;
{-------------------------------------------------------------}
Function HourMinSec;
{
```

```
Function HourMinSec(    Seconds: Rtype): String80;
Input
   Seconds: elapsed time in seconds
Output:
   HourMinSec: string giving Hour:MM:SS
)
var
   Hour,Min: Itype;
   HourS,MinS,SecS: String[10];
begin
   Hour:=Trunc(Seconds/3600.0);
   Seconds:=Seconds-Hour*3600.0;
   Min:=Trunc(Seconds/60.0);
   Seconds:=Seconds-Min*60.0;
   Str(Hour,HourS);
   Str(Min,MinS);
   While Length(MinS)<2 do MinS:='0'+MinS;
   Str(Seconds:5:2,SecS);
   If SecS[1]=' ' then SecS[1]:='0';
   HourMinSec := HourS+':'+MinS+':'+SecS;
end;
{-------------------------------------------------------------------}
Procedure Show_Time;
{
Procedure Show_Time(    Xcoord,Ycoord: integer);
}
{
   Input:
      Xcoord,Ycoord: Location on screen for writing time.
   Output:
      Writes a 11 character word giving the time of day
}
var
   hourS,minS,secS,sec100S,TimeS: string[255];
   Hour,min,sec,sec100: word;
begin
{
      Convert Time from 0..23 to 1..12 hour system
}
   GetTime(Hour,min,sec,sec100);
   str(min,MinS);
   str(sec,secS);
   If Hour in [0..11] then
      SecS:=SecS+' AM'
   else
      SecS:=SecS+' PM';
   Case Hour of
      0:      HourS:='12';
      1..12:  str(Hour,HourS);
      13..24: begin
              Hour:=Hour-12;
              str(Hour,HourS);
```

```
            end;
        end;
   While Length(HourS)<2 do HourS:=' '+HourS;
   While Length(MinS)<2 do MinS:='0'+MinS;
   While Length(SecS)<5 do SecS:='0'+SecS;
   timeS := hourS+':'+minS+':'+secS;
   gotoXY(Xcoord,Ycoord);
   write(TimeS);
end;
{-------------------------------------------------------------------------}
end.
```

TRANSLAT.PAS

```
{$N+}
Unit Translat;
Interface
Uses
   Declare,Nombre;
Procedure Get_command(var F_Code: FileC;
                      var C_type: IType;
                      var KeyWord,Say: String255);
Procedure Get_Number_pos(    Say: String255;
                         var VarName,VarValue: VectS80_Var;
                             VarQtt: Byte;
                         var IRS,Ni: VectI_Nc;
                         var Nr: VectR_Nc;
                         var Ns80: VectS80_Nc;
                         var Qtt: IType;
                         var Bad: Boolean);
{-------------------------------------------------------------------}
Implementation
Procedure Get_command;
{
Procedure Get_command(var F_Code: FileC;
                      var C_type: IType;
                      var KeyWord,Say: String255);
}
{
   Input:
      F_Code: File pointer of data file (given by user)
   Output:
      C_type: Command type:
           -1:  Error in a command;
            0:  END was found
            1:  Repeat
            2:  Var
            3:  Elem
            4:  Node
            5:  Mat
            6:  Force
            7:  Displ
            8:  Ngen
            9:  Egen
            10: Trac
            11: Local
            12: Old
            13: Inter
            14: Edel
            15: Oper
            16: Emod
      KeyWord: Command word
           example: NGEN
      Say: String following the command name (UpperCase)
```

```
          example: 1,2,,,3.34E-4,5,,7,Bozo,3
      Comments are nested in curly braces (accolades)
}
var
   I: IType;
   L: char;
   Start: Boolean;
   Mot: String[255];
begin
{
                  Get Command name
}
   start:=False;
   L:=chr(0);
   Mot:='';
   While (EOF(F_Code)=False) and ((Start=False)or(L<>' ')) do
      begin
      read(F_Code,L);
{
                  Skip comments between curly braces (like Turbo Pascal's)
                  left curly braces: chr(123), rigth: chr(125)
}
      If L=chr(123) then
         While (EOF(F_Code)=False) and (L<>Chr(125)) do
            read(F_Code,L);
{
                  Add characters to command name
}
      If Start=False then
         If L in ['a'..'z','A'..'Z'] then start:=True;
      If Start=True then
         If L<>' ' then
            Mot:=Mot+L;
      end;
{
                  Change command to Uppercase
}
   For i:=1 to Length(Mot) do
      Mot[i]:=UpCase(Mot[i]);
   KeyWord:=Mot;
{
                  Find Command Type number
}
   C_Type:=-1;
   If EOF(F_Code)=True then C_Type:=0;
   If KeyWord='REPEAT' then C_Type:=1;
   If KeyWord='VAR'    then C_Type:=2;
   If KeyWord='ELEM'   then C_Type:=3;
   If KeyWord='NODE'   then C_Type:=4;
   If KeyWord='MAT'    then C_Type:=5;
   If KeyWord='FORCE'  then C_Type:=6;
   If KeyWord='DISPL'  then C_Type:=7;
```

```
   If KeyWord='NGEN'   then C_Type:=8;
   If KeyWord='EGEN'   then C_Type:=9;
   If KeyWord='TRAC'   then C_Type:=10;
   If KeyWord='LOCAL'  then C_Type:=11;
   If KeyWord='OLD'    then C_Type:=12;
   If KeyWord='INTER'  then C_Type:=13;
   If KeyWord='EDEL'   then C_Type:=14;
   If KeyWord='OPER'   then C_Type:=15;
   If KeyWord='EMOD'   then C_Type:=16;
{
                Get Command Parameters
}
   Say:='',
   If C_type in [1..16] then
      begin
      Mot:='';
      Repeat
         read(F_Code,L);
{
                skip comments between curly braces (like Turbo Pascal's)
                left curly braces: chr(123), rigth: chr(125)
}
         If L='(' then
            While (EOF(F_Code)=False) and (L<>')') do
               read(F_Code,L);
         If L in ['!'..'z','¦','~'] then
            Mot:=Mot+L;
         until ((EOF(F_Code)=True)or      { End of file or }
               (L=';'));                  { ; terminates a line }
{
                Change Command information to Uppercase
}
      For i:=1 to Length(Mot) do
         Mot[i]:=UpCase(Mot[i]);
      Say:=Mot;
      end;
end;
{-------------------------------------------------------------------}
Procedure Get_Number_pos;
{
Procedure Get_Number_pos(    Say: String255;
                         var VarName,VarValue: VectS80_Var;
                             VarQtt: Byte;
                         var IRS,Ni: VectI_Nc;
                         var Nr: VectR_Nc;
                         var Ns80: VectS80_Nc;
                         var Qtt: IType;
                         var Bad: Boolean);
}
{
   Input:
      Say: String following the command name (UpperCase)
```

```
      VarName[1..Var_max]: variable names
      VarValue[1..Var_max]: variable values
      VarQtt: quantity of variables
   Output:
      IRS[i]: 0 if the (i)th number in the command line is IType
              1 for a real
              2 for a string
             -1 if not specified, or an error occured in conversion
      Ni[i]: IType number at the (i)th position
      Nr[i]: Real number at the (i)th position
      Ns80[i]: String at the (i)th position
      Qtt: Quantity of Possible number locations before Semicolon (End
           of command line)
      Bad: Error while interpreting
   example:
          1, 0,4.78E-12,  ,2,Jer,;
        i  1  2   3       4 5   6   7   Qtt = 7 (last comma counted)
      IRS  0  0   1      -1 0   2  -1
       Ni  1  0   0       0 2   0   0
       Nr  0  0 4.78E-12  0 0   0   0
     Ns80 '' ''  ''     '' '' 'Jer' ''
}
var
   iSay,i: integer;
   iVar,Lmot: Byte;
   NumberI: IType;
   Result: Integer;
   NumberR: Rtype;
   Mot: String[255];
   L: char;
   SameAsVar,FoundVar: Boolean;
   IRtype: Byte;
begin
{
                    Initialize Vectors;
}
   For i:=1 to Nc_max do
      begin
      Ni[i]:=0;
      Nr[i]:=0.0;
      Ns80[i]:='';
      IRS[i]:=-1;    { Initialize as a string }
      end;
   Mot:='';
   Qtt:=0;
   For iSay:=1 to Length(Say) do
      begin
      L:=Say[iSay];
      If not (L in [',',';']) then
         Mot:=Mot+L;
      If ((L=',')or(iSay=Length(Say))) then
         begin
```

```
            Inc(Qtt);
            If Nc_max<Qtt then
               begin
               Bad:=True;
               writeln;
               writeln('Too many parameters:');
               writeln(Say);
               end;
            Lmot:=Length(Mot);
            If 0<Lmot then
               begin
               StoX(Mot,NumberI,NumberR,IRtype);
{
                     Replace Variable name by it's value
}
               If IRtype=0 then   {String}
                  begin
                  FoundVar:=False;  {corresponding variable not found}
                  iVar:=1;
                  While (iVar<=VarQtt) and (FoundVar=False) do
                     begin
                     If Lmot=Length(VarName[iVar]) then SameAsVar:=True
                                                   else SameAsVar:=False;
                     i:=1;
                     While (i<=Lmot) and (SameAsVar=True) do
                        begin
                        If UpCase(Mot[i])<>VarName[iVar][i] then SameAsVar:=False;
                        Inc(i);
                        end;
                     If SameAsVar=True then
                        begin
                        Mot:=VarValue[iVar];
                        FoundVar:=True;
                        end;
                     Inc(iVar);
                     end;
                  StoX(Mot,NumberI,NumberR,IRtype);
                  end;
{
                   Store IType or Real number in appropriate place
                   if possible
}
               Case IRtype of
                  1: begin
                     IRS[Qtt]:=0;
                     Ni[Qtt]:=NumberI;
                     end;
                  2: begin
                     IRS[Qtt]:=1;
                     Nr[Qtt]:=NumberR;
                     end;
                  end;
```

```
{
                Set string in vector
}
          If IRtype=0 then  {string}
             begin
             Ns80[Qtt]:=Mot;
             If Ns80[Qtt]<>Mot then
                begin
                Bad:=True;
                writeln('String is too long: ',Mot);
                end
             else
                IRS[Qtt]:=2;  {string}
             end;
          Mot:='';
          end;
       end;
    end;
end;
{-------------------------------------------------------------}
end.
```

VECTMAT3.PAS

```
{$N+}
Unit VectMat3;
Interface
Uses
   Declare,WaitKey;
Procedure Determinant3(    A: MatrixR_3x3;
                       var Det: Rtype);
Procedure Invert3(    A: MatrixR_3x3;
                  var B: MatrixR_3x3);
Procedure VectorP3(    l1,m1,n1,l2,m2,n2: Rtype;
                   var l3,m3,n3: Rtype);
Procedure NormV3(var L,M,N: Rtype);
{-------------------------------------------------------------------}
Implementation
Procedure Determinant3;
{
Procedure Determinant3(    A: MatrixR_3x3;
                       var Det: Rtype);
}
{
   Input:
      A: 3x3 matrix
   Output:
      Det: determinant of matrix A
}
begin
   Det:=+A[1,1]*(A[2,2]*A[3,3]-A[2,3]*A[3,2])
        -A[1,2]*(A[2,1]*A[3,3]-A[2,3]*A[3,1])
        +A[1,3]*(A[2,1]*A[3,2]-A[2,2]*A[3,1]);
end;
{-------------------------------------------------------------------}
procedure Invert3;
{
procedure Invert3(    A: MatrixR_3x3;
                  var B: MatrixR_3x3);
}
{
   Input:
      A: 3x3 matrix
   Output:
      B: inverse of A
}
var
   i,j: integer;
   delta : Rtype;
   T: MatrixR_3x3;
begin
{
                  Get determinant
}
```

```
   Determinant3(A,Delta);
{
                  Change sign of elements
}
   For i:=1 to 3 do
      For j:=1 to 3 do
         begin
         T[i,j]:=A[j,i];
         If ((i+j) MOD 2)=0 then T[i,j]:=-T[i,j];
         end;
{
                  Complete inversion
}
   B[1,1]:=(T[2,2]*T[3,3]-T[2,3]*T[3,2])/delta;
   B[1,2]:=(T[2,1]*T[3,3]-T[2,3]*T[3,1])/delta;
   B[1,3]:=(T[2,1]*T[3,2]-T[2,2]*T[3,1])/delta;
   B[2,1]:=(T[1,2]*T[3,3]-T[1,3]*T[3,2])/delta;
   B[2,2]:=(T[1,1]*T[3,3]-T[1,3]*T[3,1])/delta;
   B[2,3]:=(T[1,1]*T[3,2]-T[1,2]*T[3,1])/delta;
   B[3,1]:=(T[1,2]*T[2,3]-T[1,3]*T[2,2])/delta;
   B[3,2]:=(T[1,1]*T[2,3]-T[1,3]*T[2,1])/delta;
   B[3,3]:=(T[1,1]*T[2,2]-T[1,2]*T[2,1])/delta;
end;
{-------------------------------------------------------------------}
Procedure VectorP3;
{
Procedure VectorP3(    l1,m1,n1,l2,m2,n2: Rtype;
                  var l3,m3,n3: Rtype);
}
{
   Input:
      l1,m1,n1: direction of first vector V1
      l2,m2,n2: direction of second vector V2
   Output:
      l3,m3,n3: Vector product of  V1 x V2
                  note: V1 x V2 = |  i   j   k |
                                  | l1  m1  n1 | V1
                                  | l2  m2  n2 | V2
                                =  (m1.n2-m2.n1)i+
                                  -(l1.n2-l2.n1)j+
                                   (l1.m2-l2.m1)k
}
begin
   l3 := +(m1*n2-m2*n1);
   m3 := -(l1*n2-l2*n1);
   n3 := +(l1*m2-l2*m1);
end;
{-------------------------------------------------------------------}
Procedure NormV3;
{
Procedure NormV3(var L,M,N: Rtype);
}
```

```
{
   Input:
      L,M,N: Vector components in 3-D
   Output:
      L,M,N: Normalized (Length=1) Vector components in 3-D
}
var
   Long: Rtype;
begin
   Long:=sqrt(sqr(L)+sqr(M)+sqr(N));
   If Long < (1/BigR) then
      begin
      writeln('   Error: trying to normalize vector of length: ',Long:15);
      wait;
      Halt;
      end
   else
      begin
      L:=L/Long;
      M:=M/Long;
      N:=N/Long;
      end;
end;
{--------- ------------------------------------------------------------}
end.
```

VIEW.PAS

```
{-------------------------------------------------------------------
   Program to see geometry
   Initially written in January '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 10000,0,76004}     {max memory used for drawing elements}
                       {10500 bytes in Heap needed for InitGraph}
                       {65504=2047*8*4 bytes in Heap needed for Connect.pas}
Uses
   Crt,Graph,Dos,Declare,Where,WaitKey,Draw,WhatKey,Timer,Linear,Nne_Dim,
   File_RW,Get,Sonore,Connect;
{-------------------------------------------------------------------}
var
   F_xyz,F_xyz3D: FileR;
   F_Connect,F_NewOld,F_Elem: FileI;
   NneMax,i,iD,Net,Nnt,Dim,Draw_Axis,Draw_Elev,Numbering: IType;
   Phi,Theta: Rtype;
   Modified: Boolean;
begin
   ClrScr;
   Writeln('Loading geometry');
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Get_I('Net.P',Net);
   Get_I('Nnt.P',Nnt);
   Get_I('Dim.P',Dim);
   Get_I('NneMax.P',NneMax);
{
                  Get XYZ coordinates
}
   Assign(F_xyz,C_Dir+'xyz.P');
   Assign(F_xyz3D,T_Dir+'xyz_3d.T');
   Assign(F_Elem,C_Dir+'Elem.P');
   Assign(F_NewOld,C_Dir+'NewToOld.P');
   Assign(F_Connect,T_Dir+'Connect.T');   {Don't call it 'CON.dat'}
   Reset(F_xyz);
   Rewrite(F_xyz3D);
   Reset(F_Elem);
   Reset(F_NewOld);
   Rewrite(F_Connect);
   Modified:=True;
   xyz_3d(Nnt,F_xyz,F_xyz3D,Dim,Modified);
   InitConnectHeapRAM;
   Select('VIEW',Net,Nnt,NneMax,F_xyz3D,F_Elem,F_NewOld,F_connect,
          Phi,Theta,Numbering,Draw_Axis,Draw_Elev);
   Dessine('',Nnt,Phi,Theta,
           Numbering,Draw_Axis,Draw_Elev,F_xyz3D,1.0,F_NewOld,F_Connect);
   DisposeConnectHeapRAM;
   Close(F_xyz);
   Close(F_xyz3D);     Erase(F_xyz3D);
   Close(F_Elem);
```

```
    Close(F_NewOld);
    Close(F_Connect);   Erase(F_Connect);
    ClrScr;
end.
```

WAITKEY.PAS

```
{$N+}
Unit WaitKey;
Interface
Uses
   Crt;
procedure Wait;
procedure WaitIfNo;
Implementation
{-------------------------------------------------------------------}
procedure Wait;
{
                  waits until a key is pressed
                  a Key previously pressed is ignored
}
var
   Ch: char;
begin
   While KeyPressed do   { Empty keyboard buffer }
      Ch:=ReadKey;
   Ch:=ReadKey;          { keyboard buffer is left empty }
end;
{-------------------------------------------------------------------}
procedure WaitIfNo;
{
                  waits until a key is pressed
                  if none were previously pressed
}
var
   Ch: char;
begin
   Ch:=ReadKey;
   While KeyPressed do   { Empty keyboard buffer }
      Ch:=ReadKey;
end;
{-------------------------------------------------------------------}
end.
```

WHATKEY.PAS

```
{$N+}
Unit WhatKey;
Interface
Uses
   Crt,Declare,
procedure get_key(var C: Char;
                  var action: String6);
Implementation
{-------------------------------------------------------------------
                  Taken from \TP\INC\GET_KEY.INC
-------------------------------------------------------------------}
procedure get_key;
{
procedure get_key(var C: Char;
                  var action: String6);
}
{
   output:
      C<>chr(0) or Action<>''
      C: chr(0) if no character pressed
         else: [' '..'~','c','a','a','e','e','e','e',
               'i','i','o','u','u','u']  that include A..Z a..z 0..9
      Action: '' if no action key was pressed
             else: (UP,DOWN,RIGHT,LEFT,PGDN,PGUP,HOME,END,INS,DEL)
                   (CHOME,CEND,CPGUP,CPGDN,CLEFT,CRIGHT)
                   (ENTER,BS,ESC)
                   (F1...F10, SF1..SF10, CF1...CF10, AF1...AF10)
}
var
   Ch1,Ch2,Ch3,Char_0,Char_Esc: char;
begin
   Char_0:=Chr(0);
   Char_Esc:=Chr(27);
   Repeat
      C:=Char_0;
      Action:='';
      Ch1:=Char_0;
      Ch2:=Char_0;
      Ch3:=Char_0;
{
                  Empty keyboard buffer
}
      While Keypressed do Ch1:=ReadKey;
      Ch1:=ReadKey;
{
                  get action
}
      if Ch1=Char_0 then
         begin
         Ch2:=readKey;
```

```
        case Ch2 of
           'H': action:='UP';
           'P': action:='DOWN';
           'M': action:='RIGHT';
           'K': action:='LEFT';
           'G': action:='HOME';
           'O': action:='END';
           'Q': action:='PGDN';
           'I': action:='PGUP';
           'R': action:='INS';
           'S': action:='DEL';
           't': action:='CRIGHT';
           's': action:='CLEFT';
           'w': action:='CHOME';
           'u': action:='CEND';
           'a': action:='CPGUP';
           'v': action:='CPGDN';
           end;
{
              F1..F10
}
        If ord(Ch2) in [59..68] then
           begin
           str(ord(Ch2)-58,action);
           action:='F'+action;
           end;
{
              SF1..SF10
}
        If ord(Ch2) in [84..93] then
           begin
           str(ord(Ch2)-83,action);
           action:='SF'+action;
           end;
{
              CF1..CF10
}
        If ord(Ch2) in [94..103] then
           begin
           str(ord(Ch2)-93,action);
           action:='CF'+action;
           end;
{
              AF1..AF10
}
        If ord(Ch2) in [104..113] then
           begin
           str(ord(Ch2)-103,action);
           action:='AF'+action;
           end;
        end;
{
```

```
                    Get single character actions
)
      If Ch2=Char_0 then
         case ord(Ch1) of
            8: action:='BS';
            13: action:='ENTER';
            27: action:='ESC';
            end;
      if (Ch1 in [' '..'~','c','a','a','e','e','e','e','i','i','o','u','u','u'])
         and (Ch2=Char_0) then C:=Ch1;
      until ((C<>Char_0) or (Action<>''));
end;
(-------------------------------------------------------------------)
end.
```

WHERE.PAS

```
{$N+}
Unit Where;
Interface
Uses
   Crt,Declare;
Procedure ReadDir(var C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255);
Procedure WriteDir(var C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255);
{-------------------------------------------------------------------)
Implementation
Procedure ReadDir;
{
Procedure ReadDir(var C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255);
}
{
   Input:
      Reads C_Dir ... WPname from file "CONFIG.D"
   Output:
      C_Dir: Where current case is stored (*.P)
      S_Dir: Where cases are permanently stored (root directory) (*.P)
      G_Dir: Where Generation files are (*.gen)
      T_Dir: Where Temporary files are (*.T)
      W_Dir: Where Word Processor is
      WPname: Word Processor name
}
var
   F: Text;
begin
{
                  Read Previous directories and information:
}
   Assign(F,'Config.D');
   {$I-}
   Reset(F);
   {$I+}
   If IOresult=0 then
      begin
      Readln(F,C_Dir);
      Readln(F,S_Dir);
      Readln(F,G_Dir);
      Readln(F,T_Dir);
      Readln(F,W_Dir);
      Readln(F,WPname);
      close(F);
      end
   else
      begin
      C_Dir:='';
      S_Dir:='';
      G_Dir:='';
      T_Dir:='';
```

```
      W_Dir:='';
      WPname:='';
      end;
end;
{-------------------------------------------------------------------------}
Procedure WriteDir;
{
Procedure WriteDir(var C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname: String255);
}
{
   Input:
      C_Dir: Where current case is stored (*.P)
      S_Dir: Where cases are permanently stored (root directory) (*.P)
      G_Dir: Where Generation files are (*.gen)
      T_Dir: Where Temporary files are (*.T)
      W_Dir: Where Word Processor is
      WPname: Word Processor name
   Output:
      Writes C_Dir ... WPname to file "CONFIG.D"
}
var
   F: Text;
begin
{
                  Read Previous directories and information:
}
   Assign(F,'Config.D');
   Rewrite(F);
   Writeln(F,C_Dir);
   Writeln(F,S_Dir);
   Writeln(F,G_Dir);
   Writeln(F,T_Dir);
   Writeln(F,W_Dir);
   Writeln(F,WPname);
   Close(F);
end;
{-------------------------------------------------------------------------}
end.
```

WORST.PAS

```
{-------------------------------------------------------------------
   Program to find WORST stress location (with respect to failure)
   Initially written in April '88 by Jerome Daoust for Ph.D.
-------------------------------------------------------------------}
{$N+}  { for math coprocessor }
{$M 39000,0,655360}
Uses
   Declare,Crt,File_RW,Nne_Dim,Exyz,Elem1,Elem2,Get,Timer,Sonore,Nombre,
   WaitKey,Where,Status,Displace,Param,Printer,Linear,Logo;
{-------------------------------------------------------------------}
Procedure WorstLocate(     Net,Dim,NneMax: Itype;
                       var Mat: MatrixR_Nmatx6;
                       var F_Elem,F_NodeOld,F_ElemOld: File1;
                       var F_Angle,F_xyz,F_Displ: FileR;
                       var MaxStress: MatrixR_Nmatx9;
                       var StatusWorst: String12;
                       var WorstNodeOld,WorstElementOld,WorstDirection,
                             WorstMaterial: Itype;
                       var FailureRatio: Rtype);
{
   Input:
      Net: Number of elements
      Dim: number of dimensions (1..3)
      NneMax: Maximum number of nodes in an element
      Mat[1..Number of materials,i]: El Et Glt Gtt Nlt Ntt  for 1<i<6
      F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
              for each element
              Element type = 0 if undefined
              Node are negative in Element matrix when they are use for
              the last time according to the numbering of the elements
      F_NodeOld: original node number given by user
                 for each "compacted" node #
      F_ElemOld: original element number given by user
                 for each "compacted" element #
      F_Angle: File With angle of elements (radians)
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      MaxStress[iMaterial,j]: iMaterial: Case material number
                              j =1: X
                                =2: X'
                                =3: Y
                                =4: Y'
                                =5: Z
                                =6: Z'
                                =7: Sxy max
                                =8: Sxz max
                                =9: Syz max
      StatusWorst: 'Done' if it has already been calculated
   Output:
```

```
      WorstNodeOld: node closest to failure
      WorstElementOld: Element closest to failure
      WorstDirection: Direction 1..9 of stress failure
                     X X' Y Y' Z Z' Sxy Sxz Syz
      WorstMaterial: case material number of weakest element
      FailureRatio: Stress/MaxStress ratio
}
var
   iStress,Nstress,i,iE,Etype,Emat,Nne,Node,iNne,
      WorstNodeNew,WorstElementNew: Itype;
   Elem: VectI_Nne2;
   Angle: Rtype;
   StressRatio: array[1..9] of Rtype;
   Xs,Ys: Byte;
   Exyz: MatrixR_Nnex3;
   OnAxisStressNode: array [1..Nstress_max] of Rtype;
   F_BadElem: File1;
   FiberStrain,FiberStress,ElemStrain,ElemStress,
      GlobalStrain,GlobalStress: VectR_NstressNne;
begin
   Xs:=WhereX;
   Ys:=WhereY;
   Nstress:=N_Stress(Dim);
   FailureRatio:=-BigR;
{
                  Look if worst stress location is already found
                  Found if C_Dir+'BadElem.P' exists
}
   Assign(F_BadElem,C_Dir+'BadElem.P');
   If StatusWorst='Done' then
      begin
      Reset(F_BadElem);
      Read(F_BadElem,WorstElementNew);
      Close(F_BadElem);
      iE:=WorstElementNew;
      Net:=WorstElementNew;
      end
   else
      iE:=1;
{
                  Look for worst element
}
   While iE<=Net do
      begin
      GotoXY(Xs,Ys);
      write(Net-iE+1,' ');
      Nne2_I_Read(F_Elem,iE,NneMax,Elem);
      Emat:=Elem[1]; (material number)
      R_Read(F_Angle,iE,Angle);  (radians)
      Etype:=Elem[2];
      Nne:=Get_Nne(Etype);
      Get_Exyz(Elem,F_xyz,Exyz);
```

```
    case Etype of
       1: SS1(F_Displ,Exyz,Elem,Mat,FiberStrain,FiberStress,
             ElemStrain,ElemStress,GlobalStrain,GlobalStress);
       2: SS2(F_Displ,Exyz,Elem,Mat,Angle,FiberStrain,FiberStress,
             ElemStrain,ElemStress,GlobalStrain,GlobalStress);
       end;
{
                Convert Fiber stress from a format which depends upon the
                dimension of the problem to the
                6 stress components of 3 dimensionnal cases
}
    For iNne:=1 to Nne do
       begin
       Case Dim of
          1: begin
             OnAxisStressNode[1]:=FiberStress[Lin(iNne,1,Nstress)];
             OnAxisStressNode[2]:=0.0;
             OnAxisStressNode[3]:=0.0;
             OnAxisStressNode[4]:=0.0;
             OnAxisStressNode[5]:=0.0;
             OnAxisStressNode[6]:=0.0;
             end;
          2: begin
             OnAxisStressNode[1]:=FiberStress[Lin(iNne,1,Nstress)];
             OnAxisStressNode[2]:=FiberStress[Lin(iNne,2,Nstress)];
             OnAxisStressNode[3]:=0.0;
             OnAxisStressNode[4]:=FiberStress[Lin(iNne,3,Nstress)];
             OnAxisStressNode[5]:=0.0;
             OnAxisStressNode[6]:=0.0;
             end;
          3: begin
             OnAxisStressNode[1]:=FiberStress[Lin(iNne,1,Nstress)];
             OnAxisStressNode[2]:=FiberStress[Lin(iNne,2,Nstress)];
             OnAxisStressNode[3]:=FiberStress[Lin(iNne,3,Nstress)];
             OnAxisStressNode[4]:=FiberStress[Lin(iNne,4,Nstress)];
             OnAxisStressNode[5]:=FiberStress[Lin(iNne,5,Nstress)];
             OnAxisStressNode[6]:=FiberStress[Lin(iNne,6,Nstress)];
             end;
          end;
{
                Get Highest positive ratio of:
                   S1/X
                   S1/-X'
                   S2/Y
                   S2/-Y'
                   S3/Z
                   S3/-Z
                   abs(S12/S12max)
                   abs(S13/S13max)
                   abs(S23/S23max)
}
        Node:=abs(Elem[2+iNne]);
```

```
        StressRatio[1]:=      OnAxisStressNode[1]/MaxStress[Emat,1];
        StressRatio[2]:=     -OnAxisStressNode[1]/MaxStress[Emat,2];
        StressRatio[3]:=      OnAxisStressNode[2]/MaxStress[Emat,3];
        StressRatio[4]:=     -OnAxisStressNode[2]/MaxStress[Emat,4];
        StressRatio[5]:=      OnAxisStressNode[3]/MaxStress[Emat,5];
        StressRatio[6]:=     -OnAxisStressNode[3]/MaxStress[Emat,6];
        StressRatio[7]:= abs(OnAxisStressNode[4]/MaxStress[Emat,7]);
        StressRatio[8]:= abs(OnAxisStressNode[5]/MaxStress[Emat,8]);
        StressRatio[9]:= abs(OnAxisStressNode[6]/MaxStress[Emat,9]);
        For i:=1 to 9 do
          If FailureRatio<StressRatio[i] then
            begin
            FailureRatio:=StressRatio[i];
            WorstNodeNew:=Node;
            WorstElementNew:=iE;
            WorstMaterial:=Emat;
            WorstDirection:=i;
            end;
        end;
      iE:=iE+1;
      end;
  GotoXY(Xs,Ys);
  write(' ');
  GotoXY(Xs,Ys);
{
                Save worst element number in it's compacted number
}
  ReWrite(F_BadElem);
  Write(F_BadElem,WorstElementNew);
  Close(F_BadElem);
{
                Get user number of worst node and element
}
  I_Read(F_NodeOld,WorstNodeNew,WorstNodeOld);
  I_Read(F_ElemOld,WorstElementNew,WorstElementOld);
end;
{-------------------------------------------------------------------------}
Procedure FindWorst(    CaseName: String255;
                        Net,Dim,NneMax: Itype;
                  var F_Elem,F_NodeOld,F_ElemOld: FileI;
                  var F_Angle: FileR;
                  var F_xyz,F_Displ: FileR;
                  var StatusWorst: String12);
{
  Input:
    OnPrinter: True if a printout is desired (1 full page)
    CaseName: Name of current case
    Net: Number of elements
    Dim: number of dimensions (1..3)
    NneMax: Maximum number of nodes in an element
    F_Elem: File with (Material Number,Element Type,Node1 ... NodeN)
            for each element
```

```
            Element type = 0 if undefined
            Node are negative in Element matrix when they are use for
            the last time according to the numbering of the elements
      F_NodeOld: original node number given by user
                 for each "compacted" node #
      F_ElemOld: original element number given by user
                 for each "compacted" element #
      F_Angle: File With angle of elements (radians)
      F_XYZ[1..Dim*Nnt]: File with X Y Z of nodes
      F_Displ[1..Nnt x Dimension]: File with Displacements in U,V,W for
          each node
      StatusWorst: 'Done' if it has already been calculated
}
var
   Mat: MatrixR_Nmatx6;
   MaxStress: MatrixR_Nmatx9;
   MatLabel: VectS80_Nmat;
   Bad: Boolean;
   WorstNodeOld,WorstElementOld,WorstDirection,WorstMaterial: Itype;
   FailureRatio: Rtype;
begin
   Get_Mat(Mat);
   Get_MaxStress(MaxStress);
   Get_MatLabel(MatLabel);
   ResetTimer(1);
   StartTimer(1);
   WorstLocate(Net,Dim,NneMax,Mat,F_Elem,F_NodeOld,F_ElemOld,
               F_Angle,F_xyz,F_Displ,MaxStress,StatusWorst,
               WorstNodeOld,WorstElementOld,WorstDirection,
               WorstMaterial,FailureRatio);
   StopTimer(1);
   If 60<ReadTimer(1) then Alert;
   ClrScr;
   GotoXY(1,8);
   write('Case: ');
   TextColor(LightRed);
   writeln(CaseName);
   TextColor(LightGreen);
   writeln;
   write('First failure due to stress in direction ');
   TextColor(LightRed);
   Case WorstDirection of
     1: write('1 (tension) = ',
              RField( FailureRatio*MaxStress[WorstMaterial,1],12));
     2: write('1 (compression) = ',
              RField(-FailureRatio*MaxStress[WorstMaterial,2],12));
     3: write('2 (tension) = ',
              RField( FailureRatio*MaxStress[WorstMaterial,3],12));
     4: write('2 (compression) = ',
              RField(-FailureRatio*MaxStress[WorstMaterial,4],12));
     5: write('3 (tension) = ',
              RField( FailureRatio*MaxStress[WorstMaterial,5],12));
```

```
      6: write('3 (compression) = ',
              RField(-FailureRatio*MaxStress[WorstMaterial,6],12));
      7: write('1-2 (shear) = +/-',
              RField( FailureRatio*MaxStress[WorstMaterial,7],12));
      8: write('1-3 (shear) = +/-',
              RField( FailureRatio*MaxStress[WorstMaterial,8],12));
      9: write('2-3 (shear) = +/-',
              RField( FailureRatio*MaxStress[WorstMaterial,9],12));
      end;
    TextColor(LightGreen);
    writeln;
    write('   at element number ');
    TextColor(LightRed);
    write(WorstElementOld);
    TextColor(LightGreen);
    write(' and node ');
    TextColor(LightRed);
    write(WorstNodeOld);
    TextColor(LightGreen);
    writeln('.');
    write('Material of this element is ');
    TextColor(LightRed);
    writeln(MatLabel[WorstMaterial]);
    TextColor(LightGreen);
    write('Maximum stress allowed = ');
    TextColor(LightRed);
    Case WorstDirection of
      1: writeln(RField( MaxStress[WorstMaterial,1],12));
      2: writeln(RField(-MaxStress[WorstMaterial,2],12));
      3: writeln(RField( MaxStress[WorstMaterial,3],12));
      4: writeln(RField(-MaxStress[WorstMaterial,4],12));
      5: writeln(RField( MaxStress[WorstMaterial,5],12));
      6: writeln(RField(-MaxStress[WorstMaterial,6],12));
      7: writeln(RField( MaxStress[WorstMaterial,7],12));
      8: writeln(RField( MaxStress[WorstMaterial,8],12));
      9: writeln(RField( MaxStress[WorstMaterial,9],12));
      end;
    TextColor(LightGreen);
    write('Ratio Stress/MaxStress = ');
    TextColor(LightRed);
    writeln(Rfield(FailureRatio,12));
    TextColor(LightGreen);
    write('Safety factor = ');
    TextColor(LightRed);
    writeln(Rfield(1/FailureRatio,12));
    TextColor(LightGreen);
  {
                  Write on printer also
  }
    If OnPrinter=True then
      begin
      Write(Lst,'Case: ');
```

```
      Writeln(Lst,CaseName);
      writeln(Lst);
      Write(Lst,'First failure due to stress in direction ');
      Case WorstDirection of
         1: Write(Lst,'1 (tension) = ',
                  RField( FailureRatio*MaxStress[WorstMaterial,1],12));
         2: Write(Lst,'1 (compression) = ',
                  RField(-FailureRatio*MaxStress[WorstMaterial,2],12));
         3: Write(Lst,'2 (tension) = ',
                  RField( FailureRatio*MaxStress[WorstMaterial,3],12));
         4: Write(Lst,'2 (compression) = ',
                  RField(-FailureRatio*MaxStress[WorstMaterial,4],12));
         5: Write(Lst,'3 (tension) = ',
                  RField( FailureRatio*MaxStress[WorstMaterial,5],12));
         6: Write(Lst,'3 (compression) = ',
                  RField(-FailureRatio*MaxStress[WorstMaterial,6],12));
         7: Write(Lst,'1-2 (shear) = +/-',
                  RField( FailureRatio*MaxStress[WorstMaterial,7],12));
         8: Write(Lst,'1-3 (shear) = +/-',
                  RField( FailureRatio*MaxStress[WorstMaterial,8],12));
         9: Write(Lst,'2-3 (shear) = +/-',
                  RField( FailureRatio*MaxStress[WorstMaterial,9],12));
         end;
      writeln(Lst);
      Write(Lst,'   at element number ');
      Write(Lst,WorstElementOld);
      Write(Lst,' and node ');
      Write(Lst,WorstNodeOld);
      Writeln(Lst,'.');
      Write(Lst,'Material of this element is ');
      Writeln(Lst,MatLabel[WorstMaterial]);
      Write(Lst,'Maximum stress allowed = ');
      Case WorstDirection of
         1: Writeln(Lst,RField( MaxStress[WorstMaterial,1],12));
         2: Writeln(Lst,RField(-MaxStress[WorstMaterial,2],12));
         3: Writeln(Lst,RField( MaxStress[WorstMaterial,3],12));
         4: Writeln(Lst,RField(-MaxStress[WorstMaterial,4],12));
         5: Writeln(Lst,RField( MaxStress[WorstMaterial,5],12));
         6: Writeln(Lst,RField(-MaxStress[WorstMaterial,6],12));
         7: Writeln(Lst,RField( MaxStress[WorstMaterial,7],12));
         8: Writeln(Lst,RField( MaxStress[WorstMaterial,8],12));
         9: Writeln(Lst,RField( MaxStress[WorstMaterial,9],12));
         end;
      Write(Lst,'Ratio Stress/MaxStress = ');
      Writeln(Lst,Rfield(FailureRatio,12));
      Write(Lst,'Safety factor = ');
      Writeln(Lst,Rfield(1/FailureRatio,12));
{
                  Form Feed
}
      Write(Lst,Chr(12),chr(13));
      end;
```

```
end;
{-------------------------------------------------------------------------}
var
   F_xyz,F_Angle,F_Displ: FileR;
   F_NodeOld,F_ElemOld,F_Elem: FileI;
   Nnt,Dim,Net,NneMax: IType;
   CaseName,CoarseCaseName: String255;
   StatusPrep,StatusForce,StatusDispl,StatusWorst,
     StatusFiberStrain,StatusFiberStress,
     StatusElemStrain,StatusElemStress,
     StatusGlobalStrain,StatusGlobalStress: String12;
begin
{
     WORST NoWait
       NoWait causes the process to run without waiting.
     WORST Printer
       Printer causes the displayed information to be sent to the
       printer followed with a form feed.
     WORST NoWait Printer
       NoWait and Printer parameter can be simultaneous and interchanged
}
   BatchLogo;
   ClrScr;
   Parameters;
   ReadDir(C_Dir,S_Dir,G_Dir,T_Dir,W_Dir,WPname);
   Read_Status(CaseName,CoarseCaseName,
               StatusPrep,StatusForce,StatusDispl,StatusWorst,
               StatusFiberStrain,StatusFiberStress,
               StatusElemStrain,StatusElemStress,
               StatusGlobalStrain,StatusGlobalStress);
   Write('                        Worst location search for ');
   TextColor(LightRed);
   writeln(CaseName);
   TextColor(LightGreen);
   Get_I('Net.P',Net);
   Get_I('Nnt.P',Nnt);
   Get_I('Dim.P',Dim);
   Get_I('NneMax.P',NneMax);
{
                  Get XYZ coordinates
}
   Assign(F_xyz,C_Dir+'xyz.P');
   Assign(F_Displ,C_Dir+'Displ.P');
   Assign(F_Elem,C_Dir+'Elem.P');
   Assign(F_NodeOld,C_Dir+'NewToOld.P');
   Assign(F_ElemOld,C_Dir+'OldElem.P');
   Assign(F_Angle,C_Dir+'Angle.P');
   Reset(F_xyz);
   Reset(F_Displ);
   Reset(F_Elem);
   Reset(F_NodeOld);
   Reset(F_ElemOld);
```

```
   Reset(F_Angle);
   InitDisplHeapRAM(Dim,Nnt,F_Displ); {read displacements obtained before}
   FindWorst(CaseName,Net,Dim,NneMax,
             F_Elem,F_NodeOld,F_ElemOld,F_Angle,F_xyz,F_Displ,StatusWorst);
   DisposeDisplHeapRAM;
   Close(F_xyz);
   Close(F_Displ);
   Close(F_Elem);
   Close(F_NodeOld);
   Close(F_ElemOld);
   Close(F_Angle);
{
                          Correct Status
}
   StatusWorst:='Done';
   Write_Status(CaseName,CoarseCaseName,
                StatusPrep,StatusForce,StatusDispl,StatusWorst,
                StatusFiberStrain,StatusFiberStress,
                StatusElemStrain,StatusElemStress,
                StatusGlobalStrain,StatusGlobalStress);
{
                   Wait
}
   If NoWait=False then Wait;
   ClrScr;
end.
```

APPENDIX C

PREPROCESSING LANGUAGE

This section describes the preprocessing language created for the finite element program, named *DIRECT*®. Parameters in brackets accept default values. Any parameter can be replaced by a variable, a strong point of this language. Comments can be inserted anywhere within or between commands. Error detection can return a warning or a fatal error message with accompanying comments. This is useful when considering that other commercial finite element preprocessors like *ANSYS*® allow you to make errors, then find out too late, or worse, not find out. Out of an extensive list of error checking, here are some of them:

- valid parameter in respect to other parameters in a command
- previous node and element existence when generating new ones
- variable, node or element redefinition
- compatibility between element types when different element types are used

Here are the preprocessing commands available (each parameter is explained in the help menu of *DIRECT*®):

- **DISPL** FromNode,[ToNode],[Repeat],[dNode],[Dx],[Dy],[Dz];
  sets displacements for a set of nodes

- **EDEL** ElementNumber;

  deletes an element

- **EGEN** FromElement,[ToElement],Repeat,[dElement],dNode;

  generates new elements from a set of defined elements

- **ELEM** Number,MaterialNumber,ElementType,[Angle],Node1,Node2,...;

  Defines an element

- **FORCE** FromNode,[ToNode],[Repeat],[dNode],[Dx],[Dy],[Dz];

  sets forces for a set of nodes

- **INTER** DegreeOfInterpolation,NodeStart,NodeEnd,
  [NodeIncrement],[Repeat],[PatternInc],
  [InsertQtt],[InsertNodeStart],[InsertNodeInc],
  [InsertNodeStartInc],[InsPatternInc];

  interpolates new node coordinates and force / displacement

  from other nodes to any degree

- **LOCAL** RectangularCylindricalSpherical,x0,y0,[z0],
  L1,M1,[N1],L2,M2,[N2];

  sets a working coordinate reference system

- **MAT** MaterialNumber,MaterialName;

  looks up the material name in the database and picks up the

  material properties

- **NGEN** FromNode,[ToNode],Repeat,[dNode],[dx],[dy],[dz];

  generates new nodes from a set of defined nodes

- **NODE** NodeNumber,[x],[y],[z];

  sets a node in space coordinates

- **OLD** OldNodeStart,[OldNodeEnd],[OldNodeInc],[Repeat],[OldSetInc],
  NewNodeStart,NewNodeInc,NewSetIncrement;
  passes coordinates and force-displacement for nodes of a previous case to the current case
- **OPER** NewVariableName,ValueA,OperationType,ValueB;
  does mathematical operations upon variables
- **REPEAT** AgainTimes,[d1],[d2],...;
  repeats any following command with parameter increments
- **TRAC** FromElement,[ToElement],[Repeat],[dElement],Face,Traction;
  sets a traction on a face of an element
- **VAR** VariableName,Value;
  defines a variable and pass a value