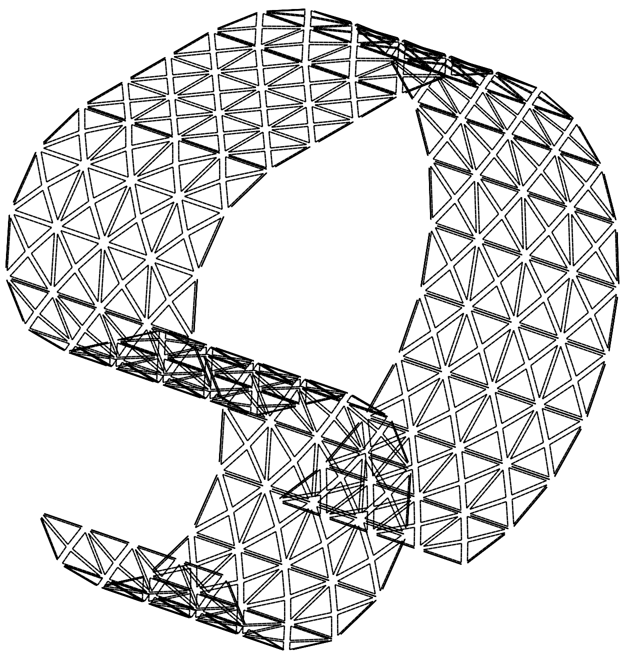
QED:

Static, Dynamic, Stability and Nonlinear Analysis of Solids and Structures



ikayex SOFTWARE TOOLS

Copyright Notice

No part of this publication may be reproduced without prior written consent from *ikayex* Software Tools.
Copyright(c) 1989 – 2008
ikayex Software Tools.
This manual describes:
DiSPtool ver 2.70, GenMesh ver 4.52, NonStaD ver 3.44, PloMeshl ver 4.60, QED ver 3.50, Simplex ver 2.62, StaDyn ver 4.68, released January 2008.

Software License Agreement

The essence of this agreement is that *ikayex* Software Tools grants the purchaser a single license for the use of the accompanying software.

The software may be used at more than one location and by more than one person provided there is no possibility that it is being used by two or more people simultaneously.

Disclaimer

ikayex Software Tools makes no warranties as to the contents of this manual or the accompanying software. Although every effort has been made to insure that the manual is accurate and the software reliable, *ikayex* Software Tools cannot be held responsible for any damages suffered from use of this product.

ikayex Software Tools
Lafayette, Indiana 47905

Contents

Chapter 1

QED the Computer Laboratory

The QED program is a Visual Simulation Tool for Analysis. Its intent is to provide an interactive simulation environment for understanding a variety of problems in solid and structural mechanics. This chapter gives an overview of QED, the underlying mechanics and programs, plus some introductory tutorials on static and dynamic stress analysis. The purpose is not to provide a tutorial on running QED — this is provided in the richly documented text *Guided Explorations in the Mechanics of Structures* [12]; rather, it is a set of tutorials on running the underlying supporting executables associated with the QED package. It is not necessary to understand these programs in order to run QED, however, knowing the material in this manual will enrich the use of QED.

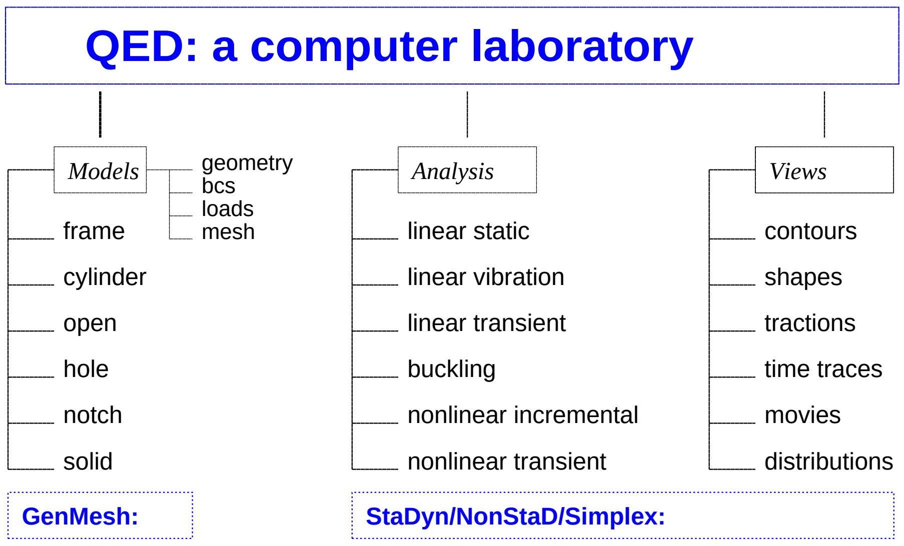
1.1 Overview of QED

An experiment by its nature is a single realization — a single geometry, material, or load case; multiple test cases and examples are just not economically feasible. But engineers, being introduced to something new, need to see other examples as well as variations on the given examples. For instance, in the stress analysis of a symmetrically notched specimen, some logical questions to ask are:

- What if the notches are bigger or smaller?
- What if there is one instead of two notches?
- What if the notches are closer or further apart?
- What if the material is changed?
- What if instead of a notch there is a hole?
- What if the clamped boundary has some elasticity?

These questions are too cumbersome and expensive to answer experimentally but are very appropriate for a simulation program. Furthermore, a very important role of a test engineer is to be able to distinguish those aspects of an experiment that have a significant deleterious effect from those that are insignificant. This can come only through experience and here too the program can help to accelerate the process of accumulating experience.

What is missing in the traditional laboratory is the iterative stage in both analysis and testing that all engineers go through — the process of asking the "What-Ifs", doing parameter sensitivity studies, and re-designing the experiment. Having a flexible sophisticated model running simultaneously on the computer to counterpoint the experiment can profoundly affect the engineers' perception of both theory and experiment. The interplay of both establish an interesting and exciting dynamic. As envisioned, the modeling program runs simultaneously with the experiment, and becomes a resource to be interacted with and tested against the experiment.



The figure above shows a schematic of the functional parts of the program. Its design is such that it isolates the user from having to cope with the full-blown flexibility of the underlying enabling programs and presents each problem in terms of a limited (but richly adaptable) number of choices and combinations.

The process of finite element analysis can be broken down into three separate stages. These are presented as independent modules in QED. The pre-processing stage allows the model geometry to be defined, the boundary conditions imposed, the loads applied and the mesh generated. In the second stage, the analytical solution is obtained. Choices as to the type of solution required and the parameters best suited to guide the procedure are made. In the post-processing stage, results are displayed in a variety of ways. Contour plots of nodal results, the deformed shape, free body diagrams and time history traces are available. The following sections provide fuller explanation of these three analysis steps.

The design philosophy of QED is to make each module very specific but flexible. This is important as it helps to fix focus on the significant aspects of behavior. Each problem has a template of properties, therefore the user need only focus on what they want to change.

Solution techniques are separated from the model building to emphasize the independent, general nature of the analysis. Whether static or transient, model solutions are sought that do not depend on the type of structure or model geometry considered.

In the post-processing, general tools are provided which demonstrate the behavior of different characteristics. Knowing which is best suited to a particular problem is vital and having all available, but being taught which are most appropriate or effective to communicate the desired information, is very valuable to a deep understanding a problem.

As shown in the following chapters, the underlying programs can be menu driven but their operation under QED is by way of driver or script files. Script files That is, QED creates the script files to execute GenMesh and StaDyn/NonStaD. The name of the script files are:

```
StaDyn/NonStaD/Simplex:  instad,   inpost,   inps
GenMesh:                 inmesh,   inmesh#,  insdf
```

In this way, the programs can operate as separate executable programs.

The description of the structure and its material properties is kept in a separate file, referred to as the *Structure DataFile*. This will have the name `qed.sdf`. Creating this file and inputting it properly is a crucial step in the analysis. As the data file is read in, numerous types of checks are performed on it so as to confirm that it was read properly. It is also echoed back into the file `<<StaDyn.LOG>>` or `<<NonStaD.LOG>>` if desired.

For Starters

There is no installation procedure, per se, in getting `QED` up and running — just a matter of copying it from the CD disk onto your hard disk. If you are not familiar with the process do the following.

Open a command prompt window (this is sometimes called an MS-DOS prompt window) off the `start/accessories` menu or by typing “cmd” in the `run` window. The properties of the command window can be adjusted by left clicking on the top bar and selecting `properties`.

Go to the root directory (assuming you are not already there and assuming you are on the C: drive) by

```
C>  cd\
```

(Do not type the `C>`). Make a working directory. Later you can place the programs in your favorite directories but for now type

```
C>  mkdir qed
C>  cd qed
```

Now place the CD in the CD drive and (assuming it is drive `D:`) type

```
C>  copy d:*.*
```

Typing the `dir` command gives some of the contents of the disk as

```
STADYN  .EXE    NONSTAD .EXE   QED     .EXE
GENMESH .EXE    STRIP   .EXE
PLOTMESH.EXE
```

These programs will be explained in due course but to check that all the copying went as it should, type

```
C>  stadyn
```

You should get the opening menu. If so, choose

```
800
```

Note that within `STADYN`, all interaction through the keyboard requires a carriage return or `ENTER` to complete the entry. Having pressed return, exit by typing

```
0
```

Now run `QED` by typing

```
C>  qed
```

and the opening graphical screen should appear. If it does not, then press 'q' to quit. Go to 'settings' off the control panel and change the number of colors to 256. This should cure the problem. Note that while most interaction with `QED` is through the keyboard, a carriage return is not required unless data is being entered.

Setup and Interaction Keys

QED launches GenMesh, StaDyn/NonStad/Simplex, and so on, as separate executable programs. Therefore, QED needs to know where to find these programs. The default location in is the

```
C:\qed
```

directory but this can be changed through the fifth line of the `<<qed.cfg>>` file. The full contents of this file are

```
166500              ::Max memory
 11100       10010  ::Max lines | pts
    75         100  ::size font
    50          20  ::animate pause #
c:\qed
20.000000      -25.000000    0.000000E+00  ::xyz rots
1    1    1    1    1    1    1    1    1    1  ::subs
1  ::grav
@@ DATE:  8-11-2007      TIME:  9:23
end
```

The third line can be used to adjust the size of the QED window and the size of the fonts; both numbers are percentages.

Generally, interaction is through the keyboard. When input in the form of numbers is required, the input is terminated with a carriage return.

Some keys are available at most stages, these are:

`h`	help reminders
`m`	mesh display toggle on/off
`o`	change orientation of axes
`q`	quit the current operation
`s`	show/render the current operation
`t`	tags toggle on/off
`w`	write a PS file of zoomed image

The help reminders are somewhat context sensitive.

A few pointers on running QED:

- It is strongly recommended that QED be launched from the COMMAND window and not from EXPLORER.
- All interaction with QED is via the highlighted keys or the leftmost symbol (number/letter) on the menus.
- The size of the window and of the fonts are changed by editing the third line in the file `<<qed.cfg>>`. These numbers are percentages; the first number is the percentage of the full screen occupied by the QED window.
- If the initial screen is black, use settings of the CONTROL PANEL to change the number of colors and/or resolution.
- PS files of contours can be obtained by pressing "w" in the zoom window. Parameters for the PS contours are changed by editing the file `<<stadyn.ctr>>`. To set up GhostView as the PostScript viewer type SETUP.

- TRACE data files are obtained by zooming on the window; the data is stored in the file `<<qed.dyn>>`.
- For problems with a large number of elements, it may be necessary to change the dynamically allocated memory sizes in `<<stadyn.cfg>>`, `<<nonstad.cfg>>`, and `<<genmesh.cfg>>`.
- The symptom that a supporting .EXE program did not run is that the window pops up and closes immediately. If this happens, look in the appropriate `<<*.log>>` file.

Being productive with QED

QED is set up so as to remember the complete state of a procedure whether it is creating a model or doing an analysis. The relevant information is stored in the `<<***.cfg>>` files such as `<<frame.cfg>>`, `<<solid.cfg>>`, and `<<anal.cfg>>`. These get over-written as changes are made, therefore, to archive a particular frame, say, copy `<<frame.cfg>>` to a new name. When this particular frame is required again, then just copy the file back to `<<frame.cfg>>`. It is important to note that this must be done while QED is not running.

There are two facilities for recording the graphics results from QED. The simplest is to cut and paste the screen into `Paint` and then use `Paint`'s capability to manipulate and print the image. Suppose it is desired to print a copy of the contours, then on the QED top bar menu click `Edit/Select All` then click `Edit/Copy`. Launch `Paint` from `Start/Accessories` and click `Edit/Paste`. The resulting files can be stored in a variety of bitmap formats including `<<***.bmp>>` and `<<***.jpg>>`.

The support for vector graphics is through PostScript. QED produces strictly ASCII files which are easily edited with any text editor if needed. Grey scale images (such as for photoelastic or Moiré fringe patterns) are included via the `image` function.

Chapter 2

Model Building with GenMesh

When structural problems are large, it is essential to have an automatic scheme for the generation of an input data file; this not only removes the drudgery of making the file, but more importantly it helps ensure its integrity. The purpose of this chapter is to show how the program GenMesh (GENerate a MESH) can be used to create structure datafiles for use by StaDyn/NonStaD/Simplex.

The design of StaDyn/NonStaD is such that a frame and a meshed plate have very much in common — primarily the nodes have exactly the same number of degrees of freedom. Thus, any generic mesh can be easily made to represent either of these two structural types. Indeed, both can be combined to form a complex mesh. The Simplex meshes are quite different since they only have translational degrees of freedom and cannot (at present) be combined with StaDyn/NonStaD meshes.

When manipulating complex (or compound meshes) it is essential to avoid having to deal directly with node numbers or element numbers. GenMesh uses the idea of *groups* and *tags* to keep track of special collections of nodes and elements, respectively. These are usually specified at the time of making the component meshes and the complex mesh then inherits them. Operations such as specifying material properties or specifying boundaries for attachment become easier, and most important become independent of the mesh density.

The main capabilities of GenMesh involve generating meshes and performing the following executive functions:

- Create Generic 2-D Shapes
- Create Arbitrary 2-D Meshes
- Create 3-D Structures
- Create 3-D Solids
- Re-Map Mesh
- Re-connect Mesh
- Merge two Meshes
- Make Structure DataFile

Unlike frame elements, plate elements are approximate and therefore many elements are required to accurately model a given region. Some of the generic meshes available are:

- 2-D Plane shapes
  - Quadrilateral block
  - Two to One reduction
  - Generic Cut-out
  - Generic Notch
  - Arbitrary shape
- 3-D Structures
  - Dome w/out stringers
  - General w/out stringers
  - Space Frame
  - Space Truss

The same mesh can be used for a linear analysis by StaDyn or for a nonlinear analysis by NonStaD. The solid element meshes can be analyzed only by Simplex.

2.1 Types of Structures Considered

Structures that can be satisfactorily idealized as a collection line elements are called *frame* or *skeletal* structures. Usually their members are assumed to be connected either by frictionless pins or by rigid joints. Structures that can be satisfactorily idealized as a collection of flat platelets are called *folded plate* or *thin-walled* structures. These platelets are usually connected by frictionless pins or by rigid joints. StaDyn can analyze these structures separately as well as in combination.

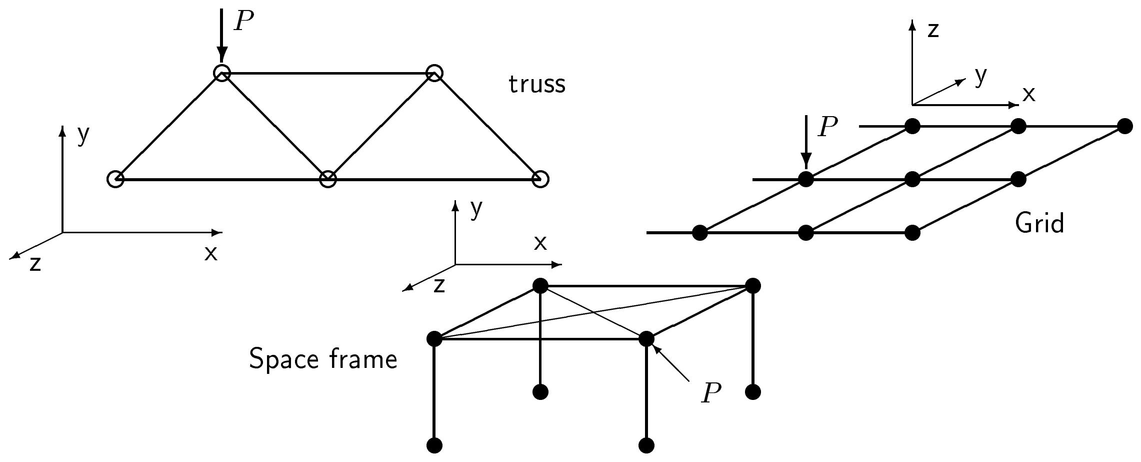


Figure 2.1: Some types of skeletal structures.

A *truss* consists of a collection of arbitrarily oriented rod members that are interconnected at pinned joints. They are loaded only at their joints and (because the joints cannot transmit bending moment) must be triangulated to avoid collapse. A *frame* structure, on the other hand, is one that consists of beam members which are connected rigidly or by pins at the joints. The members can support bending (in any direction) as well as axial loads, and at the rigid joints the relative positions of the members remain unchanged after deformation. Rigidly jointed frames are often loaded along their members as well as at their joints. Plane frames, like plane trusses, are loaded only in their own plane. In contrast, *grids* (or *grills*) are always loaded normal to the plane of the structure. Space frames can be loaded in any plane. The space frame is the most complicated type of jointed framework — each member can undergo axial deformation, torsional deformation, and flexural deformation (in two planes). Its supports may be fixed, pinned, elastic, or there may be roller supports.

Corresponding to the space frame, there is the folded plate structure. Each platelet undergoes in-plane deformations as well as out-of-plane bending and twisting.

StaDyn/NonStaD is set up to analyze the space frame and folded plate because all other types of jointed structures are special cases that can be obtained by reduction from it.

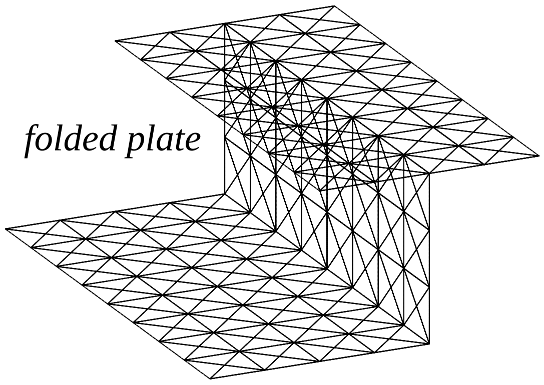


Figure 2.2: A folded plate structure.

The number of possible displacement components at each node is known as the *nodal degree of freedom* (DoF); the nodal degree of freedom for different structural types is shown in the following table:

Structure	Dimension	u	v	w	ϕ_x	ϕ_y	ϕ_z	Type #
Rod	$1-D$	$\surd$						11
Beam	$1-D$		$\surd$				$\surd$	12
Shaft	$1-D$				$\surd$			13
Truss	$2-D$	$\surd$	$\surd$					21
Frame/membrane	$2-D$	$\surd$	$\surd$				$\surd$	22
Grill/Plate	$2-D$			$\surd$	$\surd$	$\surd$		23
Truss	$3-D$	$\surd$	$\surd$	$\surd$				31
Frame/FoldedPlate	$3-D$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	32
GeneralStructure	$3-D$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	33
Solid	$3-D$	$\surd$	$\surd$	$\surd$				111

From this table, it is clear how the frame structure and folded plate structure share common types of degrees of freedom. This essentially is what allows them to be combined together to form complex structures.

A Note on the Elements Used

Since StaDyn/NonStaD is designed to analyze thin-walled 3-D structures comprising a mixture of frame and plate sub-structures, then it simplifies the implementation when both structural types are modeled in a compatible way. This section briefly describes the elements used — more general treatments of the finite element method can be found in References [3, 7, 19, 26] and aspects specific to framed structures are developed in References [1, 17, 25], and aspects specific to StaDyn/NonStaD in References [10, 11]. Three-noded triangular elements were chosen primarily because they can be conveniently mapped to form irregular shapes. Furthermore, we consider only thin plate flexural theory (Kirchhoff plates) and the corresponding slender beam theory (Bernoulli-Euler).

A 3-D frame member has six DoF at each node

$$\{u\} = \{u,\, v,\, w,\, \phi_x,\, \phi_y\, \phi_z\}$$

In local coordinates, this has three behaviors. There is a rod action with axial displacement and force

$$\{u\} = \{u\}\,, \qquad F(x) = EA\frac{\partial u}{\partial x}$$

There are two beam actions; the bending moment and shear force. The corresponding nodal degrees of freedom are the rotation $\phi_z(x)$ (or the slope of the deflection curve at the node) and the vertical displacement $v(x)$.

$$\{u\} = \{v,\, \phi_z\}\,, \qquad M(x) = EI\frac{\partial^2 v}{\partial x^2}\,, \quad V(x) = -EI\frac{\partial^3 v}{\partial x^3}$$

There is also a bending about the y-axis. Finally, there is a twisting about the axis

$$\{u\} = \{\phi_x\}\,, \qquad T(x) = GJ\frac{\partial \phi_z}{\partial x}$$

where EA, EI, and GJ are the axial, bending and torsional stiffnesses, respectively; E and G are the Young's and shear modulus, respectively; and A, I and J are the area, moment of inertia, and polar moment of inertia, respectively. Full details on the matrix implementation for frame structures can be found in Reference [10].

A 3-D plate supports both in-plane (membrane) and out-of-plane (flexural) actions. The in-plane behavior of the plate is analogous to that of a plane 2-D frame. Thus at each node we want the DoF to be

$$\{u\} = \{u,\, v,\, \phi_z\}$$

The usual constant strain triangle (CST) element has only the two displacements in its formulation. The element implemented in `StaDyn` is taken from the paper by Bergan and Felippa [6]. This is a nine-noded triangular element which is shown to have superior in-plane performance over the CST. But more importantly from our perspective is that it correctly implements the drilling DoF (ϕ_z) and therefore makes it suitable for a 3-D incorporation. The 'rotation' implemented is actually that taken from continuum mechanics

$$\phi_z = \tfrac{1}{2}\left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)$$

Coding for the element is given in Reference [6].

The strains are obtained by differentiation of the displacements

$$\epsilon_{xx} = \frac{\partial \bar{u}}{\partial x}\,, \qquad \epsilon_{yy} = \frac{\partial \bar{v}}{\partial y}\,, \qquad \gamma_{xy} = \frac{\partial \bar{u}}{\partial y} + \frac{\partial \bar{v}}{\partial x}$$

The material behavior is represented (for the plane stress case) by

$$\left\{ \begin{array}{c} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{array} \right\} = \frac{E}{1-\nu^2} \left[\begin{array}{ccc} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{array} \right] \left\{ \begin{array}{c} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{array} \right\}$$

where ν is Poison's ratio.

The out-of-plane behavior is analogous to that of a plane 2-D grid. That is, we want an element that has at each node the degrees of freedom

$$\{u\} = \{w,\, \phi_x,\, \phi_y\}$$

The rotations are related to the deflection by

$$\phi_x = \frac{\partial w}{\partial y}\,, \qquad \phi_y = -\frac{\partial w}{\partial x}$$

In local coordinates the three-noded triangle has a total of 9 degrees of freedom. This element, now called the Discrete Kirchhoff Triangular (DKT) element, was first introduced by Stricklin, Haisler, Tisdale, and Gunderson in 1968 [22]. It has been widely researched and documented as being one of the more efficient flexural elements (see Batoz, Bathe, and Ho [4]). Code for the element is given in References [5, 7].

The curvatures are obtained by differentiation of the displacement

$$\kappa_{xx} = \frac{\partial^2 w}{\partial x^2}\,, \qquad \kappa_{yy} = \frac{\partial^2 w}{\partial y^2}\,, \qquad \kappa_{xy} = \frac{\partial^2 w}{\partial x \partial y}$$

The material behavior is represented (for the plane stress case) by

$$\left\{ \begin{array}{c} M_{xx} \\ M_{yy} \\ M_{xy} \end{array} \right\} = \frac{Eh^3}{12(1-\nu^2)} \left[\begin{array}{ccc} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{array} \right] \left\{ \begin{array}{c} \kappa_{xx} \\ \kappa_{yy} \\ \kappa_{xy} \end{array} \right\}$$

where h is the plate thickness.

In this manner, each node whether it is associated with a frame or a plate element has the requisite six DoF and all resolved components of applied loading will be supported.

The 3-D solids are modelled using Hex20 elements. This has 20 nodes and each node has the DoF

$$\{u\} = \{u,\, v,\, w\}^T$$

Formulations for its use are given in a number of texts, two of which are references [2, 8]. This element performs well even when used to construct shells.

2.2 Structure Data File

The description of the structure and its material properties is kept in a separate file, referred to as the *Structure DataFile*. Creating this file and inputting it properly is a crucial step in the analysis.

This section describes the fields for the structure datafile.

Structure DataFile Format

The input datafile can be in free format with blanks or commas used as separators. Any editor or word processor can be used to make changes to a file, but make sure to store the new file as strict ASCII files without any hidden word-processing symbols.

Note that the data input is arranged in groups, and that each group must have the word `END` or `end` as its last line. This acts as an additional data checker.

Header Group

```
TITLE
IGLOBAL
IFLAG1 IFLAG2 IFLAG3 IFLAG4
end
```

TITLE :	Short title (up to 40 characters) describing the mesh or problem.
IGLOBal:	Global problem reduction.[integer] 1-D: 11=rod, 12=beam, 13=shaft 2-D: 21=truss/cable, 22=frame/membrane, 23=grill/plate 3-D: 31=truss/cable, 32=frame/folded-plate, 33=general 3-D: 110=solid plane strain, 111=solid general
IFLAG :	Echo flags for the input. 1=on, 0=off 1=connectivity, 2=material, 3=numbering, 4=loads

Connectivities Group

The number of lines is equal to `NEL`. Every element must have an input for it, although they do not have to be input in strict sequential order. However, the nodes for triangular elements must be input counter-clockwise.

```
NEL
ELM   TYP    NPI    NPJ     NPK
 :     :      :      :       :
 :     :      :      :       :
end
```

NEL	: Number of elements
ELM	: The unique number assigned to each element.
TYP	: The type of element. [integer]
	1 = truss, 2 = cable, 3 = frame, 4 = triangle, 21 = solid.
NPI	: The number of the #1 node on the element. [integer]
NPJ	: The number of the #2 node on the element. [integer]
NPK	: The number of the #3 node on the element. [integer]
	: For truss or frame members set `NPK=NPJ`.
	: For solids, `NP*` ranges from 1 to 21.

Material Tags Group

The number of lines is equal to `NMAT`. Every element must have a material number or tag; however, it is possible to specify overlaying numbers. For example, to make Element 7 different from Elements 1-6 and 8-10, say, specify 1 - 10 followed by 7 - 7.

```
NMAT
ELM1    ELM2     #
  :       :      :
  :       :      :
end
```

NMAT	: Number of element material lines. [integer]
ELM1	: First element of this material. [integer]
ELM2	: Last element of this material. [integer]
#	: Material number, tag number, or sub-structure number. [integer]

Coordinates Group

The number of lines input must equal `NNP`. The lines do not have to be input in strict order from 1 to NNP, but every node must be on a separate line.

```
NNP
NODE     XORD     YORD   ZORD
  :        :        :      :
  :        :        :      :
end
```

NNP	: Number of nodal points
NODE	: The unique number assigned to each node.
XORD	: x-coordinate of the node point.
YORD	: y-coordinate of the node point.
ZORD	: z-coordinate of the node point.

Boundary Conditions Group

The number of lines input must equal `NBC`. Again the numbering need not be strictly sequential. The default value for each degree of freedom is free; that is, the boundary conditions need be imposed only for those nodes that have constrained degrees of freedom.

```
NBC
NODE    XDOF    YDOF   ZDOF   XROT    YROT    ZROT
  :       :       :      :      :       :       :
  :       :       :      :      :       :       :
end
```

NBC :	The number of nodes which are given explicit degree of freedom constraint.
NODE :	The number of a node at which at least one degree of freedom is being fixed.
XDOF :	Motion in the x-direction. [integer]
YDOF :	Motion in the y-direction. [integer]
ZDOF :	Motion in the z-direction. [integer]
XROT :	Motion about the x-axis. [integer]
YROT :	Motion about the y-axis. [integer]
ZROT :	Motion about the z-axis. [integer] In each case: 0=fixed, 1=free. For solids, specify rotations as 0=inactive.

Nodal Loads Group

The number of nodes which have applied loads or concentrated masses must equal `NLOAD`. The specified loaded nodes do not have to be input in strict sequential order. Each node is assumed to have zero applied load and zero concentrated mass unless imposed otherwise. The actual applied load history for dynamic problems is input from a separate file — the values here essentially say where the loads are applied as well as their relative scaling.

```
NLOAD
NODE    XLOAD    YLOAD    ZLOAD    XMOM    YMOM    ZMON   TYPE
  :        :        :         :       :      :       :       :
  :        :        :         :       :      :       :       :
end
```

NLOAD : Number of load nodes or mass points. [integer]

NODE	:	The unique number assigned to each node. [integer]
XLOAD	:	Force applied in the x-direction at the node. [real]
YLOAD	:	Force applied in the y-direction at the node. [real]
ZLOAD	:	Force applied in the z-direction at the node. [real]
XMOM	:	Moment applied about the x-axis at the node. [real]
YMOM	:	Moment applied about the y-axis at the node. [real]
ZMOM	:	Moment applied about the z-axis at the node. [real]
TYPE	:	Additional load feature. [real]
	:	TYPE $\geq$ 0.0: Concentrated mass of value TYPE at the node.
	:	TYPE $= -2$: Second force distribution.
	:	For solids, specify moments as 0.0.

Element Material Properties Group

The number of lines is equal to `MATYPE`. There must be a type specified for each material tag. The input format is identical for both the plate and frame elements although a couple of the entries have slightly different interpretations. The frame values are indicated in parenthesis.

```
MATYPE
Mat#    E    G    A  Rho   Gama   Ix   Iy   Iz   ::truss
Mat#    E    G    A  Rho   Gama   Ix   Iy   Iz   ::cable
Mat#    E    G    A  Rho   Gama   Ix   Iy   Iz   ::frame
Mat#    E    G    h  Rho   Gama   Ip   Ia   Ib   ::plate
Mat#    E    G    0  Rho   0      0    0    0    ::solid
Mat#    C00  Alf  0  Rho   0      0    0    0    ::rubber
 :      :    :    :   :     :      :    :    :
 :      :    :    :   :     :      :    :    :
end
```

MATYPE	:	Number of element materials. [integer]
Mat#	:	Material number. [integer]
E	:	Young's modulus of the material. [real]
G	:	Shear modulus of the material. [real]
A	:	Cross-sectional area of the frame elements. [real
h	:	Thickness of the plate element. [real]
Rho	:	The density of the element (input as W/g). [real]
Gama	:	Stiffness modifier. [real]
		cable: $\bar{F}_o = \gamma EA$
		frame: orientation of principal axes
		plate: $\gamma > 0 =$ plane stress
		plate: $\gamma < 0 =$ plane strain
Ix	:	Frame polar moment of area about x-axis. [real]

Iy	:	Frame second moment of area about y-axis. [real]
Iz	:	Frame second moment of area about z-axis. [real]
Ip	:	Bending second moment of area for plate. [real] $I_p = h^3/12$: uniform plate
Ia	:	In-plane drilling parameter, $(\alpha = 1.5)$. [real]
Ib	:	In-plane drilling parameter, $(\beta = 0.5)$. [real]
C00,Alf	:	Mooney-Rivlin, $(C_{10} = (1-\alpha)C_{00},\ C_{01} = \alpha C_{00})$. [real]

Specials Group

This group is a mechanism to allow the input of special global properties. In the present formulation, damping is applied to all members equally, and is made proportional to their mass and/or stiffness matrices.

The number of lines input must equal NSP.

```
NSP
CODE    C1     C2     C3
  :     :      :      :
  :     :      :      :
end
```

NSP	:	Number of lines
CODE	:	Unique code number for each special attribute. [integer] 2110=damping: $[\,C\,] = (c_1/\rho)[\,M\,] + (c_2/\rho)[\,K\,]$ 2120=gravity: $\hat{g} = c_1\hat{e}_x + c_2\hat{e}_y + c_3\hat{e}_z$ 2210=shear effect: rod, $c_1\nu\sqrt{GI_x/EAL^2}$; beam, $c_2 12EI_z/GAL^2$ 2341=plasticity: $\sigma_Y = c_1,\ E_T = c_2$. 2410=reduced integration: $c_1 = 1$; $c_2 = 2$; $c_3 = 1, 2, 3$ (Hex20, 3=full).
C1	:	First constant [real]
C2	:	Second constant [real]
C3	:	Third constant [real]

Examples of Structure DataFiles

The following example is for a simple in-plane plate structure as shown in Figure 2.3(a).

```
ex_ps.1: 8-element plate                    ::header
22
1 1 1 1
end
8                                           ::element
1    4    1    2    5
```

```
2    4    2    3    5
3    4    3    4    5
4    4    4    1    5
5    4    2    6    8
6    4    6    7    8
7    4    7    3    8
8    4    3    2    8
end
1                                             ::material #s
1  8  1
end
8                                             ::node
1     0.00      0.0    0.0
2     2.00      0.0    0.0
3     2.00      2.0    0.0
4     0.00      2.0    0.0
5     1.00      1.0    0.0
6     4.00      0.0    0.0
7     4.00      2.0    0.0
8     3.00      1.0    0.0
end
4                                             ::boundary condns
1    0    0   0   0  0  0
2    1    0   0   0  0  0
6    1    0   0   0  0  0
4    0    1   0   0  0  0
end
2                                             ::applied loads
6    1000.0    0.0    0.0 0.0 0.0 -500.0     0.0
7    1000.0    0.0    0.0 0.0 0.0  500.0     0.0
end
1                                             ::# of materials
1    10.0e6 4.0e6 1.00    2.50e-4  1    1 1.5 0.5
end
0                                             ::specials
end
```

This is the input file for a simple plate problem and a copy of it is on the disk as `<<ex_ps.1>>`. Note that this data is inputted in free format with blank spaces used only as separators. In general, the file can be documented by adding comments on the remainder of a line — a convention followed in all the examples is that comments are separated from the required numbers by double colons, but otherwise there is nothing special about the double colons. For this particular example, the customary units are used — StaDyn/NonStaD, however, will handle any set of units as long as they are consistent. This mesh corresponds to one quarter of a plate with a uniform stress of 1000 psi applied at both ends.

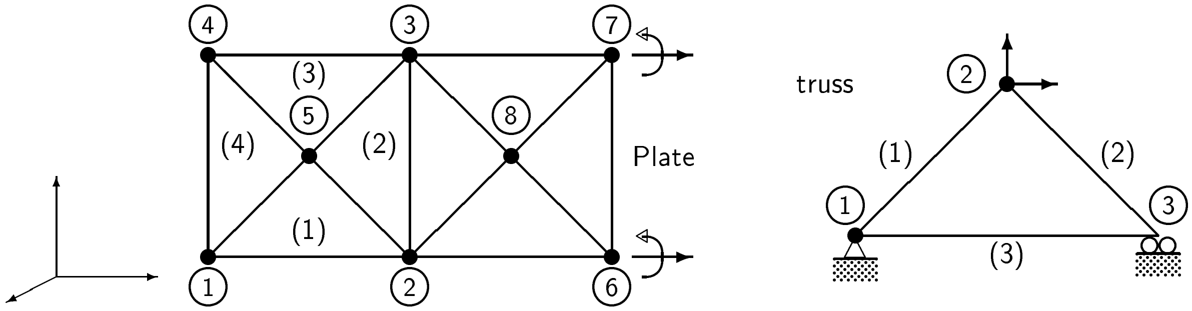


Figure 2.3: Some simple structures. (a) EX_PS.1, an eight element plate. (b) EX_FS.1, a three element truss.

Notice how the boundary conditions are imposed: Node 1 is fixed in all directions, but Node 2 and Node 3 are free to move in the x-direction and not in any of the other directions. That is, they are on horizontal rollers. Likewise, Node 4 is on vertical rollers. The imposition of a stress corresponds to a distributed applied loading; this element has an edge shape function similar to that of a beam hence the consistent load moments are

$$\begin{aligned} P_1 &= \frac{1}{2}\sigma_o tL = +\frac{1}{2} \times 1000 \times 1 \times 2 = 1000 = P_2 \\ T_1 &= +\frac{\alpha}{12}\sigma t L^2 = +\frac{1.5}{12} \times 1000 \times 1 \times 2 \times 2 = 500 = -T_2 \end{aligned}$$

where α is the drilling parameter as appears in the material line. Imposing these moments is only necessary when the mesh is relatively coarse; generally speaking, the lumped approximation will be adequate.

Note that there must be entries in each group. Thus even if there are no loads, say, there must be at least one line indicating so. This is illustrated in the lines for the `Specials Group`. While seemingly unnecessary, this helps the input checker to be more accurate and therefore useful.

The following example is for a simple 2-D truss structure as shown in Figure 2.3(b) and a copy of it is on the disk as `<<ex_fs.1>>`.

```
ex_fs.1: Balfour pp121.                 ::header
21
1 1 1 1
end
3                                       ::element
1 1 1 2 2
```

```
2 1 2 3 3
3 1 1 3 3
end
1                                       ::material #
1 3  1
end
3                                       ::node
1  0.0 0.0 0.0
2  3.0 3.0 0.0
3  6.0 0.0 0.0
end
2                                       ::boundary conditions
1  0   0   0   0   0   0
3  1   0   0   0   0   0
end
1                                       ::applied loads + mass
2  100e3  200e3  0.0   0.0 0.0 0.0   0.0
end
1                                       ::material props
1  200e9   70e9   3000e-6  0.0 0.0   1.0 1.0 1.0
end
0                                       ::specials
end
```

For this particular example, the units are in metric — StaDyn/NonStaD/Simplex will handle any set of units as long as they are consistent. Notice how the boundary conditions are imposed: Node 1 is fixed in all directions, but Node 3 is free to move in the x-direction and not in any of the others. That is, Node 3 is on rollers.

The structure datafiles for both the plate and truss are almost identical — the only differences are in the element number, the specifics of the material properties, and the repeated third connectivity. Consequently, it is very easy to construct a structure that contains both types of elements.

Checking the Input

This tutorial shows how to use some of the built-in diagnostics of StaDyn/NonStaD to help ensure that the structure datafile is correct.

To run the program simply type

```
C>  stadyn
```

Note that `stadyn` is the name of the executable version. The menu will appear and you then just respond to the questions. Since you want to input the structure datafile, choose

```
        2
```

and you are asked for the data filename. Type

```
        ex_ps.1
```

If StaDyn has read the data correctly, it will acknowledge so and then present the main menu again. To quit, choose

```
        0
```

In the working directory there is a newly created file called `<<stadyn.log>>`. It is an ASCII file so peruse it by typing

```
   C>  type StaDyn.log
```

It is a long file so it may be preferable to pipe to `MORE`. There appears to be a big jumble of numbers and symbols such as

```
   @@ StaDyn version 4.40, June 2002
   @@ DATE:  6-19-2002      TIME: 19:38
   @@ MAXimum storage   :     2000000
   @@ MAXimum elements  :        5000
   @@ MAXimum nodes     :        5000
   @@ MAXimum force incs:        1000   500
   @@ ITERmax           :          16
   @@ rtol              :    1.000000E-10
   @@  ALLOCATION succeeded
                  2   ::MAIN
   ex_ps.1
      :
   @@ HEADER GROUP
   @@ Title        = ex_ps.1 8-element plate
   @@ Problem type = 22
      :
```

This file is actually a record, or log, of the session just completed. There are two types of lines in this file. The lines that have double colons (`::`) are generally the inputs given to StaDyn. What follows the colons are brief descriptions of what was typed or where it was typed. The other lines that begin with double ats (`@@`) are StaDyn's information or answers as responses. It gives extra insight into its workings that can be very useful when trying to backtrack to nail a problem. In this particular case it can be used to judge if all the structural data was read as intended. If StaDyn detects inconsistencies in the input data, it will flag them and by looking through this log file it is possible to determine approximately where the inconsistency occurred.

Look further through the file and see the manner in which the structural data is echoed. The single most common source of errors for a finite element analysis is in the data input. You should attempt to become familiar with this file because it can be an immensely useful tool for checking the integrity of the data input.

The datafile can also be visually checked by running it through the plotting utility PlotMesh by simply typing

```
C>  plotmesh
```

By pressing the active keys various forms of the datafile will be presented.

2.3 Rectangular Mesh

Unlike frame elements, plate elements are approximate and therefore very many elements are usually required to accurately model a given region. The purpose of this introductory tutorial is to mesh a rectangular region.

Getting Started

`GenMesh` is designed to run as a console (or command window) program under the various flavors of `MS Windows`. Note that all instructions are case insensitive; we will vary the case only to help make the instructions clearer.

To run the program, type (at the 'C prompt')

```
C>  genmesh
```

(note the name of the executable.) You are given the opening menu

```
MAIN menu:
        0: Quit
                                        <create>
        2: Create Generic 2-D Shapes
           23: Create Generic 3-D Solid Shapes
        3: Create Arbitrary 2-D Mesh
           30: Create Arbitrary 2-D Shapes
        4: Create 3-D Structures
                                    <manipulate>
        5: Re-Map
        6: Re-Mesh
        7: Merge two  Meshes
                                       <realize>
        8: Make Structure DataFile
        9:
                                       <services>
      700: Write PostScript Plot File
      911: Ikayex help
SELECT -->
```

Quit by typing

```
0
```

There are two files worth looking at. The first is the log file `<<genmesh.log>>`:

```
@@ GenMesh version 4.10, April 2002
@@ DATE:  6-11-2002      TIME: 11:11
@@ MaxElem request:    5000
@@ Maxes # 4 8:  80001  75001  5001
@@ ALLOCATION succeeded
          0 ::MAIN
@@ GenMesh OK, exited from MAIN
```

The second is `<<genmesh.cfg>>`. This is the configuration file and has in it

```
        5000                    ::MaxElem
        2.00000   2.00000       ::pos X Y
        20.0000                 ::pen_thick
         .000   .000  .000  1.000 ::rotX Y Z shrink
       1.000  7.000  1.000      ::cap X Y size
       1                        ::mesh type
@@ DATE:  6-11-2002      TIME: 11:22
end
```

This allows the run time setting of the dimensions of the arrays. The other settings are the parameters for making a PostScript drawing of the mesh. If a memory allocation failure occurs, this is the place to make the adjustments.

Making the Mesh

We will now run GenMesh to generate a file containing the same information as in the file `<<ex_ps.1>>`. Type

```
   C>  genmesh
```

to get the opening menu. We wish to mesh a simple plate, so choose

```
       2
```

and you are asked to give the mesh file a new name or keep the default one

```
       SAVE new mesh as:
       0=return  1=GMesh.MSH  2=New Name
```

Choose a new name

```
       2
       m1
```

and the program responds with the choice of generic shapes

```
     MESH types:
              0: Return
              1:
               : 2-D Plane shapes
              2: Quadrilateral block
             22: Diagonal      block
              3: Two to One reduction
              4:
              5: Generic Cut-out
             52: Diagonal Cut-out
             54: Diagonal rect Cut-out
              6: Generic Notch
             62: Diagonal Notch
    SELECT -->
```

Choose quadrilaterals

```
       2
```

We must now tell the program how many elements we want, their orientation, and so on. GenMesh instructs

```
       TYPE:   # of modules    X-dir  |  Y-dir  | elem type  | tag #
```

Four numbers are being asked for here and the above are reminders of the options. A convention followed throughout StaDyn/NonStaD and GenMesh is that the vertical bar | separates the entries in the prompts, and spaces or commas separate the choices. A module is a rectangle divided into four triangles (elements) by its diagonals. The triangles share a common node at the intersection of the diagonals with the other nodes at the corners of the rectangle, as shown in Figure 2.4 for a $[2 \times 2]$ set of modules. By choosing different numbers of modules in the vertical and horizontal directions, we can vary the aspect ratio of the module. (Keep in mind that scaling the mesh is a different operation than choosing the number of modules.) The `elem type` is 3 for a frame and 4 for a plate element. The `tag #` can be used to keep track of the sub-structures used in forming a complicated structure, it is also used to associate different material properties with different groups of elements.

For dynamic problems, we need a great number of elements to effectively distribute the mass, but for this tutorial type

```
       2  2  4  1
```

This gives $2 \times 2 \times 4 = 16$ plate elements.

The physical size of the mesh is now specified when GenMesh begins asking for

```
       SCALE mesh:  0=as_is    1=scale
```

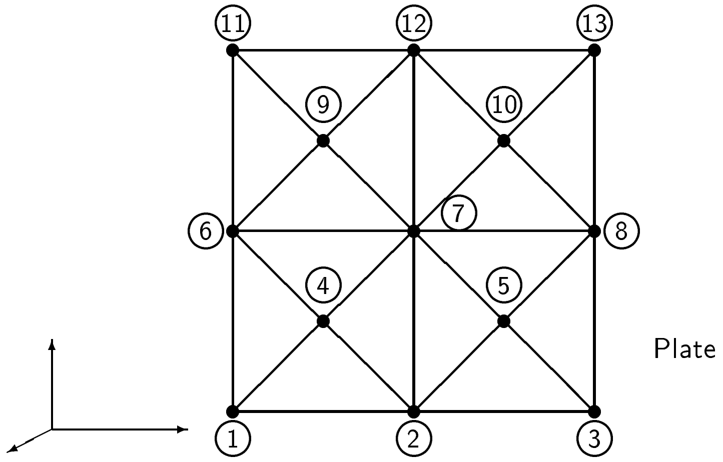


Figure 2.4: A [2 × 2] set of modules.

We will scale it now, so type

```
    1
```

which leads to

```
                   NE          NW          SW          SE
TYPE:          x1 | y1  |  x2 | y2  |  x3 | y3  |  x4 | y4
```

These correspond to the coordinates of four corners — (x_1, y_1) is the coordinate of the northeast corner, (x_2, y_2) the northwest corner, and so on around in an anti-clockwise fashion. Note that it is at this point that the mesh may be configured to any four sided shape; it may be stretched out, squashed, or sheared. For our case, type

```
10.0 10.0    0.0 10.0    0.0  0.0   10.0  0.0
```

This makes it a square plate with 10 units on the side. GenMesh responds with information that `NEW mesh in:  m1`. Back at the main menu quit by typing

```
0
```

Look through the mesh file and you will see

```
Quads
22
0 0 0 0
end
16                          ::ELEMENT GROUP
1  4  1 2 4
2  4  2 7 4
:
:
```

The structure of the file is there plus the connectivities and coordinates. At this stage, this is a generic mesh meaning that it could be used for in-plane loading, bending or 3-D folded plates. Or it could be used to attach to another generic shape to make a more complex mesh involving plates and frames.

Adding Properties

To make the generic mesh into a structure datafile, we need to add properties. In particular, we need to state the type of material, the boundary conditions, and the loads. The generic meshes have a default state inferred from the construction process. This section allows changes to be made to this state.

Initiate `GenMesh`

```
C> genmesh
```

and from the main menu select

```
8
```

to add properties. We are going to read in our generic mesh, add properties to it, and store it under a new name. First you are asked to give the mesh file a new name or keep the default one

```
SAVE new mesh as:
0=return  1=GMesh.MSH  2=New Name
```

Choose to change the name

```
2
```

You are asked for the new name

```
TYPE: NEW_filename -->
```

respond

```
rect.sdf
```

Next you are asked for the input filename

```
TYPE: IN_filename -->
```

Respond

```
m1
```

You are now given a summary of the mesh as

```
        1   Matls
       13   Nodes
        0   BCs
        0   Forces
        0   materials
        0   SPcls
        4 boundary groups
```

and the change properties menu.

```
        0: return
        1: Header     group
        2: Matl tag   group
        3: BCs        group
       41: Loads      group  <new>
       42: Loads      group  <overlap>
        5: Material   group
        6: Specials   group
```

At this stage pick those properties to be changed or to be specified. Start with the title

```
        1
```

and you are asked

```
        TYPE:  Title -->
```

In response type

```
        rect.sdf: square plate
```

This is the title of the problem, not the filename. The program then asks for the problem type

```
   GLOBAL types:
      1-D:  11=rod              12=beam                 13=shaft
      2-D:  21=truss/cable      22=frame/in-plane       23=grill/plate
      3-D:  31=truss/membrane   32=frame/folded plate
   TYPE:  Global type -->
```

Since we are interested in 2-D in-plane behavior choose

```
        22
```

The four echo flags are asked for next

```
    TYPE:  4 flags
```

We will eventually have a large number of elements for the dynamic problem, so we don't want to fill our disk with all this information. But first time through with a new mesh it is a good idea to echo all the information, so type

```
    1  1  1  1
```

Next select information about the substructure tags

```
    2
```

and you are given

```
    CHOOSE material types:
           0=return
           1=re-tag by groups
           2=create single tagged group
           3=re-tag by element
```

The first of these (that is, making no change) will simply associate the material numbers with the tag number, thus all elements tagged 3, say, will have material properties #3. The other three options allow the re-tagging of the elements. For simplicity, choose default material numbers

```
    0
```

Specify the boundary conditions.

```
    3
```

The program gives the options

```
    BOUNDARY condns:
    CHOOSE:        0=continue
                   1=node range  (sequential)
                   2=node range  (bc info)
                   3=node group  (bc info)
                   4=cylinder    (volume )
                   5=nearest     (single )
                   9=interrogate (bc info)
```

The interrogate option is for reminders of the boundary information (although having PlotMesh running simultaneously is a better option). Options 2 and 3 take their information from the boundary nodal information automatically stored at the bottom of each generated mesh file. Since we only have a couple of boundary conditions, input by the boundary information range sequence, hence type

```
        2
```

The node and boundary condition are specified in response to

```
    TYPE:                                          <0-fixed, 1-free>
     Node1 | Node2 | xdof | ydof | zdof | Xrot | Yrot | Zrot
```

Type

```
        11 1   0 0 0  0 0 0
```

corresponding to the fixed boundary condition at the left side. When specifying the boundary conditions, it is possible to specify a range of nodes; the sequence in the range is obtained from the bottom of the mesh file. For plane meshes, the nodes go in a counter clockwise fashion. The boundary condition menu is recurring, so that many boundary conditions may be input. Note that ranges can overlap. Move on by typing

```
        0
```

The applied force data is specified next in response to

```
        41
```

The other load option would be used to add some new loads to an existing load system.

```
        APPLIED LOADS:
        CHOOSE:    0=return
                   1=nodal    load (sequence)
                   2=nodal    load (bc info )
                   3=nodal    load (bc group info )
                   4=nodal    load (nearest xyz   )
                   6=cylinder load (volume)

                  11=equal force       to tagged surface
                  12=uniform pressure  to tagged surface
                  15=traction distribution
                  16=traction distribution   from file
                   9=interrogate
```

There is a single applied force so type

```
        1
```

Give its location in response to

```
TYPE:
Node1 | Node2  |  Px  |  Py  |  Pz  | Tx | Ty | Tz |   cmass
```

and put it at Node 8

```
8  8   1.0   0.0   0.0   0.0  0.0  0.0  0.0
0
```

Just as for the boundary condition input, this would be a recurring request if there are more than one applied force. Note that while the force history will be specified elsewhere this file must say where it is applied. Thus the entry 1.0 above could be used as a scaling factor. Indeed, multiple force sources could be inputted but they will all act as scalings on the applied load history. In this way, distributed and vector loads can be applied. Note also that this is the place where concentrated masses (separate from the distributed mass of the elements) are input.

Next, input the material information.

```
5
```

and receive the request

```
INPUT:   # of different Materials
```

There is only one material

```
1
```

You are given a reminder of the appropriate format for the material data

```
INPUT:
   #  | E  | G  | t(A) | Rho | PLN(gama) | Ip(Ix) | Ia(Iy) | Ib(Iz)
```

In response, let the material be nominally aluminum under plane stress conditions

```
1   10.0e6  4.0e6  1.0  2.50e-4  1   0.08  1.5  0.5
```

The last two numbers are associated with the in-plane behavior of the plate element — generally they should always be specified as above. This would be a recurring sequence if there is more than one material.

We do not want to specify any special material properties so quit

```
0
```

After this you are now put back at the main menu. To exit GenMesh, type

```
0
```

It would be a good idea at this stage to check the structure datafile by running it through StaDyn.

2.4 Compound Meshes

The purpose of this tutorial is to show how GenMesh can be used to create a mesh composed of a number of generic shape meshes. The particular mesh is shown in Figure 2.5, and is comprised of three different meshes: a $[2 \times 4]$ quadrilateral, a 4 module two-to-one transition, and a block with a circular hole.

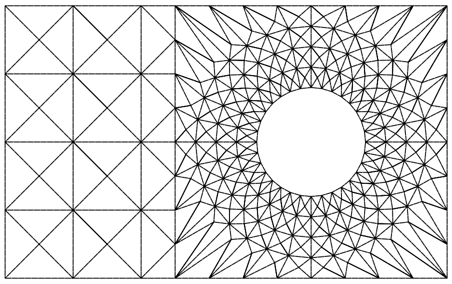


Figure 2.5: Compound mesh comprising three different mesh types.

There are two reasons why a compound mesh might be formed. The first is that the object is composed of two different generic shapes, for example, rectangle and circular; each of these are best meshed in their natural coordinates. The other is that different portions of the structure are composed of different element types. In connecting different meshes together, some transition elements are sometimes required; we will demonstrate the use of these.

Making the Generic Meshes

The script file for the quadrilateral is

```
C>  genmesh
       2 ::MAIN
       2
       m1
       2   ::MESH
       2 4 4 1  ::nxmod,nymod,el,tag
       1     ::1=scale
       1 2  0 2  0 0  1 0
       0 ::MAIN
```

Scaling is done so this is of size 1×2. The script file for the transition mesh is

```
C>  genmesh
       2 ::MAIN
       2
       m2
```

```
          3   ::MESH
          4 4 2   ::nxmod,el,tag
          1     ::1=scale
          2 .25  0 .25  0 0  2 0
          0 ::MAIN
```

This is scaled to size 2×0.25. Use `PlotMesh` to view this mesh; it is the same as in the center section of Figure 2.5 but is horizontal.

We will go through the mesh with a hole a little more in detail. Initiate the program as usual and select the circular hole option

```
  C>  genmesh
          2 ::MAIN
          2
          m3
          5   ::MESH
```

We must now declare how many modules comprising the mesh

```
      INPUT:   # of modules    Hoop   |  Radial  |  elem  |  tag #
```

Many of the generic meshes are basically the same, the differences lie in their boundaries.

In specifying the modules, we do not consider either the boundary modules or the center fans since these require special fix-ups for the different cases. Because the hole is inherently symmetrical, choose a number of hoop modules that is divisible by 8.

Respond

```
       32  4  4  3
```

You are now asked

```
       INPUT:  Rmin | exponent |  0=hollow 1=full
```

The first is the inside radius while the second allows variation of the spacing of the modules in the radial direction. For example, it might be good to put a higher concentration of modules near the inside edge of the hole; this would be achieved by specifying the exponent less than unity. For now, choose

```
       0.4   1  0
```

This gives equi-spaced modules between 0.4 and 1.0. The outer boundary is currently circular but we can map it to a straight side.

```
       MAP straight boundary:  0=as_is   1=map
```

Choose to map

```
       1
```

and you are asked for

```
                  NE      N
       TYPE:     x1,y1   x2,y2
```

We just want a square outer boundary, so type

```
       1 1   0 1
```

Meshes that are mapped may have elements with poor aspect ratio; it is good practice in those situations to smooth or balance the element shapes. In response to

```
       INPUT:  # of smooth cycles  |  type  (1=area,2=coord)
```

reply

```
       0   0
```

because, for the moment, we want to see what an unsmoothed mesh looks like. We also do not want to scale the mesh, so respond

```
       0
       0
```

As each of these generic meshes are being made, it is advisable to use the utility PlotMesh to survey the results. In particular, you need to note the boundary node numbers.

Aligning Meshes

There are a variety of ways that generic meshes can be combined; some of the schemes are based on the information about their boundary nodes. (This information is stored at the end of each mesh file.) Thus the process could be made highly automated for some problems. StaDyn/NonStaD and GenMesh are designed for analyzing general three-dimensional structures and therefore the schemes implemented for merging meshes must also work in those general cases. The basic idea is to first align the substructures as if they are to be physically welded and then attach the nearest nodes.

We will use `M1` as the reference mesh, place `M2` rotated next to it, and finally align `M3`. The two driver files are

```
   C>  genmesh
           5
           2
           m2b
```

```
          m2
          2
          0  0  90
          1.25  0  0
          0
          0
```

Sometimes a little iteration is needed in order to fully predict (in the 3-D cases) where the rotations and translations will leave the new mesh. Again, this is a case were the use of a script file can help considerably.

The mesh with the hole need only be translated

```
   C>  genmesh
          5
          2
          m3b
          m3
          2
          0       0  0
          2.25  1.0  0
          0
          0
```

Merging Meshes

We will first combine `M1` and `M2b`. Initiate `GenMesh` and choose the merge option

```
   C>  genmesh
          7
          2
          m12
```

You are now asked to

```
       TYPE: infileNAME_1 -->
```

It makes a difference which mesh is given first because that is the one to which the second is subservient. Respond

```
       m1
```

Similarly for the second filename

```
       TYPE: infileNAME_2 -->
```

Respond

```
        m2b
```

A brief summary of the two meshes is displayed. Now you are asked about how the two meshes are to be attached

```
        CONNECTion modes:   MODEL_1 <-- MODEL_2
                            0=continue
                            1=node to node
                            2=group to surface
                            3=intersecting surfaces
             INPUT:   mode
```

Note that Model 2 is attached to Model 1. The most basic form of attachment is #1 where, essentially, the two plates are stitch together node by node. The simplest is #3 where any close nodes are automatically connected. We will use the second mode for illustrative purposes.

```
        2
```

and you are asked about the attachment nodes

```
     INPUT: group #  | # of nodes | release as BC 0=none 1=all 2=interior
                      (-# = all)
```

The group referred to is the group of boundary nodes at the bottom of the mesh file. When two meshes are joined what were formerly boundaries now could become interior regions. We therefore may wish to release them and not consider them as boundaries (or special collections of nodes) anymore. Since boundary nodal groups usually overlay at a common node Options 1 and 2 allow distinguishing between the two situations. The sequence of attaching is counter-clockwise for the first mesh, and clockwise for the second. Before getting to this stage, it is good practice to use `PlotMesh` to explore the generic meshes. In particular, that program can determine the boundary nodes.

The appropriate information is

```
        2  5  2
```

The attaching is a recurring menu. Since we do not want to attach any more segments, then to quit type

```
        0
        0
```

Use `PlotMesh` to look at this mesh and note the boundary nodes.

We now wish to merge this new mesh with `M3b`. The procedure is the same as what was just done. To be different, we will show the use of the third mode of attachment.

```
C>  genmesh
       7 ::MAIN
       2
       m123
       m12
       m3b
       3
       0.01  2
       0
       0 ::MAIN
```

The only item of interest here is the 'proximity' number; this can be made large or small depending on the density (or coarseness) of the nodal points. The resulting mesh is shown in Figure 2.5. An important point to note about the mesh file is how, the through the use of the tags, the identity of the individual sub-structure meshes is retained. This is reflected in the different colors used in PlotMesh.

Mesh Smoothing

An obvious feature of Figure 2.5 is that the elements are not in a smooth proportion. This is noticeable in the transition from the coarse rectangular mesh, as well as in the transition of the radial mesh to the rectangular outer boundary. This is a common occurrence in mesh generation, however, some of the obvious irregularities can (and should) be removed and GenMesh has a menu option that does that. Note, however, that this will work only for 2-D plane meshes, in other words, this would be useful before assemblage to the 3-D structure.

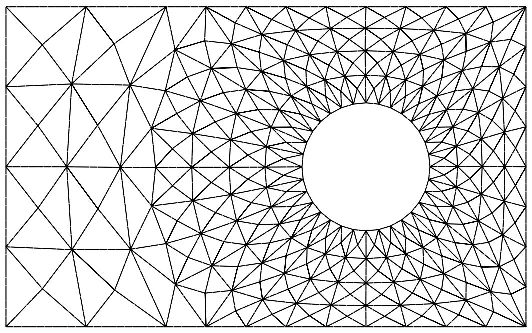


Figure 2.6: Compound mesh with smoothing.

Initiate `GenMesh` and choose the `Re-Map` option

```
C>  genmesh
```

```
          5 ::MAIN
          2
          m123b
          m123
```

The following small menu of the various re-mappings is presented:

```
       CHOOSE mapping:
              0=move on
              1=scale
              2=position (rot & trans)
              3=move single nodes
              5=change boundary groups
              6=smooth
                62=extra smooth
             11=wrap around shape
```

This allows various ways to map or distort a global mesh. We just want to smooth so choose

```
       6
```

You are asked about smoothing

```
       INPUT: # of smooth cycles  |  type  (1=area,2=coord)
```

Choose

```
       11  1
```

What smoothing does is replace the position of every node with the average position (with a weighting based on area or coordinate) of all the nodes of its immediate neighbors. All the nodes belonging to the boundary nodal groups are unaffected thus leaving the outer shape of the mesh intact. (Note that if a frame mesh is to be smoothed then the type based on coordinates should be used.) The number chosen is the number of passes taken — between 5 and 11 passes is usually adequate but check in the `LOG` file to see how rapidly the norm is converging. Quit by typing

```
       0
       0
```

The mesh now looks like Figure 2.6. There is some improvement but obviously the bad effects of incompatible meshes cannot be entirely eliminated. When smoothing a mesh, additional biasing can be obtained by moving individual nodes to a desired location and then specifying it as a boundary node. In this way, the surrounding nodes will adjust appropriately. A much better mesh is obtained by making the uniform section 8 modules deep.

2.5 Circular Plate with Pressure

The purpose of this tutorial is to show how `GenMesh` can be used to mesh circular regions. In particular, we will run it to generate a file containing the same information as in the file `<<ex_ps.2>>` that will be used for the circular plate for flexural tutorials.

Making the Mesh

Run the program and select the `Generic Shapes`

```
   C>  genmesh
          2
          2
          m1
```

The program responds with the choice of generic shapes, choose the 'generic cut-out' option

```
        5
```

We must now input how many modules comprising the mesh

```
        TYPE:   # of modules    Hoop   |  radial  |  elem |  tag #
```

Many of the generic meshes are basically the same, the differences lie in their boundaries.

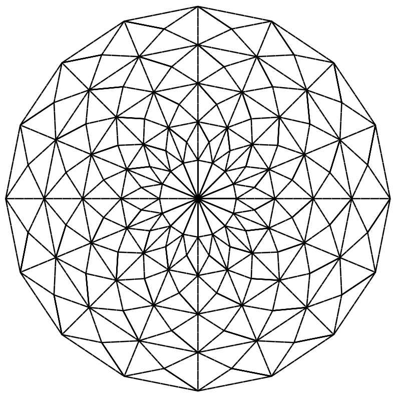


Figure 2.7: Solid circular hole representing a circular plate.

In specifying the modules we do not consider the center fan since this requires special fix-ups. Respond

```
        16  4  4  1
```

You are now asked

```
        INPUT:  Rmin | exponent  |  0=hollow 1=full
```

The first is the inside radius even though it is a solid plate. The radius referred to here is that of the solid fan at the center; it is not possible to carry the module idea down to a point. The second input allows variation of the spacing of the modules. For example, it might be good to put a higher concentration of modules near the inside edge of the hole; this would be achieved by specifying the exponent less than unity. Respond

```
        0.2   1   1
```

This will give equi-spaced modules between 0.2 and 1.0. We do not want to map the outer boundary of the mesh to a rectangle or smooth the interior node distribution, therefore type

```
         0
         0  0
```

Only the generic dimensions of the total mesh has been specified but it is possible at this stage to change it in response to

```
        SCALE mesh:  0=as_is    1=scale
```

We will scale it such that it is 10 in radius.

```
         1
```

which leads to

```
                       NE           NW           SW           SE
        INPUT:      x1 | y1   |   x2 | y2   |   x3 | y3   |   x4 | y4
```

These correspond to the coordinates of four corners — (x_1, y_1) is the coordinate of the northeast corner, (x_2, y_2) the northwest corner, and so on around in an anti-clockwise fashion. Note that it is at this point that the mesh may be configured to any four sided shape. It may be stretched out, squashed, or sheared; in fact the circle could be transformed into an ellipse. For our case, type

```
       10.0 10.0  0.0 10.0  0.0 0.0   10.0 0.0
```

and quit

```
        0
```

You are now informed that the mesh is in `P1`. Display it with `PlotMesh`, it should look like Figure 2.7.

Adding Properties

To make the generic mesh into a structure datafile, we need to add properties. That is, we need to state the type of material, the boundary conditions, and the loads. We have already done this in a previous tutorial, hence the following will dwell only on the significantly different parts.

Initiate `GenMesh` and from the main menu select the `Make Structure DataFile` option.

```
C>  genmesh
        8
        2
        m2
        m1
```

You are now given a summary of the mesh as

```
      1   Matls
    145   Nodes
      0   BCs
      0   Forces
      0   materials
      0   SPcls
      2   boundary groups
```

The series of questions to be answered are the same as the earlier tutorial, so we will just give the responses.

```
      1
      circ.msh: circular plate  uniform pressure
      23
      0  0  0  0
```

We will specify the boundary conditions next.

```
      3
      3
      2    0  0  0    0  0  0
      0
```

corresponding to the fixed boundary condition all around the outside. This was made convenient by realizing that the outer boundary nodes are in group 2 — note that this is a consequence of having used the generic mesh generator to produce the mesh in the first instance. The boundary condition menu is recurring, so that many boundary conditions may be input. Note, also, that ranges can overlap.

The applied force data is to be specified next

```
        41
```

and we are given the options

```
        APPLIED LOADS:
        CHOOSE:    0=return
                   1=nodal load  (sequence)
                   2=nodal load  (bc info )
                   3=nodal load  (bc group info )
                     32=nodal load  (bc group info ) (1/2 ends)
                     34=node traction (bc group) (1/2 ends)
                   4=nodal load  (nearest xyz)
                   6=cylinder    (volume)
                     6=cylinder resultant (volume)

                  11=equal force       to tagged surface
                  12=uniform pressure  to tagged surface
                     122=uniform load  to tagged frame
                     124=uniform pressure to Hex20 surface
                  15=traction distribution
                     152=traction distn (coords in file)
                  16=traction distribution    from file
                   9=interrogate
```

We will use the idea of the tagged element to distribute the pressure over the elements.

```
        12
```

and you are asked for

```
        TYPE:  tag #  |  pressure
```

Although there is only one element tag in this model, it can be imagined that this tag idea is a way of isolating particular sub-structures for application of the pressure. For example, this can be used to isolate the outer skin of an aircraft wing. Respond

```
        1  1.0
```

Note that an applied pressure is treated differently than applied point forces — a pressure acting on a surface produces tractions normal to the surface. The components of the forces are computed automatically and will then be distributed among the nodes of the each element. This is a lumped formulation, the slightly more accurate (for coarse meshes, anyway) consistent method is not as clean to implement here since it requires knowledge of the plate bending shape functions.

The applied loads menu is a recurring one that allows multiple loads to be added, but we will quit here

```
      0
```

The material properties are specified by

```
      5
      1
      1    10.0e6  4.0e6  0.1  2.50e-4  1   0.0833e-3  1.5  0.5
```

The third last number computes the plate moment of inertia for a uniform plate as

$$I_p = \frac{h^3}{12} = \frac{0.1}{12} = 0.08333 \times 10^{-3}$$

Note, however, that for arbitrary complex plates (with interior reinforcement, for example) an effective I_p can be used that is not related to the thickness as above.

We have all our properties specified, so quit

```
      0
      0
```

Survey `<<m2>>` and make sure the properties have been installed as expected.

Minimizing the Bandwidth

When forming the mesh, no consideration is given to reducing the bandwidth. The reason for this is that in all likelihood the mesh will be combined with another and consequently its bandedness properties would be destroyed. The final step in the mesh creation is the reduction of the bandwidth.

Before we reduce the bandwidth, it is instructive to first send the file through StaDyn. Initiate it, read in the mesh `<<m2>>`, and quit.

```
  C>  stadyn
          2
          m2
          0
```

This does two things: first it checks that the mesh is readable by StaDyn, the second thing it does is compute the bandwidth. This is displayed as [387 × 387] in the `stadyn.log` file. The computational cost for analysis with this mesh would be very large. Also given is the profile storage value; this says that the actual storage requirements is currently 25767 or 17% of the bandwidth storage. It is this number that we should concentrate on reducing. Keep in mind however, that in doing its reduction GenMesh does not take the degrees of freedom into account and thus the specific numbers will differ between GenMesh and StaDyn.

To reduce the bandwidth, run GenMesh and select the `Re-mesh` option

```
    C> genmesh
           6
           2
           m3
           m2
```

You are now given the re-meshing menu

```
       CHOOSE:  0=return
                 1=
                 2=remesh plate
                 3=remesh frame
                 4=add frame member
                 5=
                 6=reduce bandwidth
                11=mesh of only boundary elements
                21=separate elements
                23=form cohesive surface
                31=remove snags
                 :
```

Choose

```
       6
```

and you are asked

```
       CHOOSE:    0=return  1=reduce  2=update
```

Choose reduce

```
       1
```

and you are informed that

```
       INITIAL band = 145
       STORAGE  3450 21025     16% of NB
       INPUT:  wavefront center    x0  |  y0  |  z0
```

The initial band is different from that reported by StaDyn because the present program does not take the number of degrees of freedom per node into account; the consequences of this are negligible. Nor does it take the boundary conditions into account. The simple scheme for bandwidth reduction is that of the *wavefront*, that is, the nodes are numbered based on their relative distance from some point (the wavefront center). Try

```
       0  0  0
```

and you are told that

```
INITIAL BAND =     145
FINAL   BAND =      47
INITL STORAGE     3450    21025      16% of NB
FINAL STORAGE     3542     6815      51% of NB
CHOOSE:  0=return  1=reduce  2=update
```

Note that while the bandwidth was reduced, the profile storage was increased. Try

```
1
0  -1000  0
```

and you get an even smaller bandwidth

```
INITIAL BAND =     145
FINAL   BAND =      46
INITL STORAGE     3450    21025      16% of NB
FINAL STORAGE     2279     6670      34% of NB
CHOOSE:  0=return  1=reduce  2=update
```

This is a case where, relative to the original, we have reduction in both bandwidth and profile storage. You could experiment more but we will save it and quit.

```
2
0
0
0
```

If you now run `<<m3>>` through StaDyn you will see that the storage requirements are displayed as [387×120] and a profile storage value that is currently 15823 or 34% of the bandwidth storage.

All the original node numbers have been changed, so you should use PlotMesh to familiarize yourself with the new numbers.

2.6 Arbitrary Meshes

The purpose of this tutorial is to show how the `GenMesh` program can be used to mesh regions where the boundaries are specified in an arbitrary fashion. In particular, we will redo the earlier problem of the circular hole in the rectangular block.

Boundary Information

We will need to input data that describes the boundary control points. These are then connected to form segments. the numbering sequence is such that the area to be meshed is on the left hand side. This takes the form

```
# of control points
node       Xord     Yord
   :         :        :
   :         :        :
end
# of segments
1st node  2nd node  connection type
   :         :        :
   :         :        :
end
```

The `connection type` specifies the geometric shape of the segment. The current implemented forms are

```
straight line        :     1
inside   semi-circle :     2
outside  semi-circle :    -2
bezier   curve       :     3
circular arc         :     4
```

Mesh Generation

Run the program

```
   C>  genmesh
       3
       2
       a1
```

and in response to

```
TYPE: # of control points
      6
```

We now input the location of these points

```
INPUT:   point #  |  x1   |   y1
```

This will cycle through the six points.

```
1 0    0
2 3.25 0
3 3.25 2
4 0    2
5 1.85 1
6 2.65 1
```

We now input the segment connections and the style of their connection. There are six segments

```
TYPE:   # of segments
        6

TYPE:     seg# |  point_1  |  point_2  |  style
```

This will cycle through the six points.

```
1  1 2    1
2  2 3    1
3  3 4    1
4  4 1    1
5  5 6    2
6  6 5    2
end
```

Each segment or overlapping segments can be divided and these will form the boundary edges of the elements.

```
INPUT:  1st seg  |  2nd seg    |  # of divsns    <0 0 0 to end>
```

This is a recurring menu, type

```
1 3 15
2 2 8
4 4 4
5 6 16
0 0 0
```

This will give the same number of modules around the hole and at the left and right ends as in our earlier tutorial. A final piece of information is asked for

```
    INPUT:  maxelements attempted  <  3000
```

This subroutine is a situation were we do not know the final number of elements in advance. This input allows you to set an upper limit on the number of elements formed. Actually, if the program is having difficulty forming a mesh it is a good idea to set this number very and view the partial mesh formed; this will give you an idea of what is being attempted and how you might adjust the input file to correct for it. Choose

```
        1000
```

The program proceeds, periodically echoing its attempts at connecting nodes. The first mesh created is rather crude, the better mesh is obtained by performing a smoothing operation.

```
        INPUT: # of smooth cycles  |  type (1=area 2=coord)
```

Typically five to twelve smoothing cycles are adequate. Type

```
        11   1
```

Finally we need to tag the mesh

```
        INPUT:    elem type  |  tag #
```

Give the information and quit.

```
        4   1
        0
```

Now survey the mesh `A1` using the utility `PlotMesh`. The smoothed mesh should look like Figure 2.8. A judicious use of boundary divisions segments can significantly improve the smoothness of the mesh.

Note that it is essential that these arbitrary meshes be sent through the bandwidth reduction option.

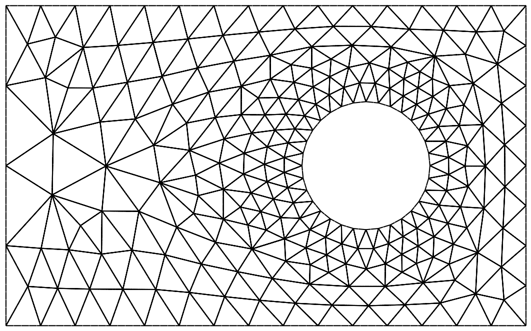


Figure 2.8: Arbitrary mesh.

2.7 Complex Structure

Thin-walled structures are usually reinforced by the addition of stringers; these long slender members are added to give additional bending resistance. The thin-wall is modeled using plate elements and the stringers are modeled using frame elements. Thus we have a structure composed of both types of elements.

The purpose of this tutorial is to show how GenMesh can be used to create such a mesh; in fact, it is the file `<<ex_cs.1>>` to be used in a later tutorial.

Making the Generic Meshes

We want to make two generic meshes: a [4 × 4] quadrilateral, and a frame mesh that will surround it. The script file for the quadrilateral is

```
C>  genmesh
        2 ::MAIN
        2
        m1
        2   ::MESH
        4 4 4 1  ::nxmod,nymod,el,tag
        1      ::1=scale
        4 4  0 4  0 0  4 0
        0 ::MAIN
```

Scaling is done so this is of size [4 × 4].

The simplest way to make the frame is to first create a three element mesh such as the following:

```
    frame
    22
```

```
0 0 0 0
end
3                                     ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
3 3 3 4 4
end
1                                     ::material #s
1 3  2
end
4                                     ::NODE GROUP
1   0.0 0.0 0.0
2   4.0 0.0 0.0
3   4.0 4.0 0.0
4   0.0 4.0 0.0
end
0
end
0
end
0
end
0
end
```

Call the mesh `<<m2>>`. Alternatively, a one module frame can be created and the extraneous lines removed. We will convert the frame into twelve elements using the `Re-Mesh` capability of GenMesh. The responses are

```
C>  genmesh
       6  ::MAIN
       2
       m3
       m2
       3
       1     3      4     ::nel1 nel2 mul
       0     0      0
       0
       0 ::MAIN
```

As each of these generic meshes are being made, it is advisable to use the utility PlotMesh to survey the results. In particular, you need to note the boundary node groups of mesh `<<m1>>`.

Merging Meshes

Generic meshes can be combined based on the information about their boundary nodes or can be combined based on a node by node basis. We will use the former approach.

Initiate GenMesh and choose the merge option

```
C>  genmesh
        7
        2
        m4
        m1
        m3
```

A brief summary of the two meshes is displayed. Now you are asked about how the two meshes are to be attached

```
        CONNECTed modes:   MODEL_1 <-- MODEL_2
                  0=continue
                  1=node to node
                  2=group to surface
                  3=intersecting surfaces
        INPUT:   mode
```

The most basic form of attachment is #1 where, essentially, the two structures are stitched together node by node. This is generally a tedious menu option to respond to but is the preferred scheme when the response file itself is generated automatically. The simplest is the third one and we will use this form of attachment since the two meshes are already aligned,

```
        3
```

We will not release any boundary nodes (that is, they will remain as boundaries which may prove useful for applying loads, say) Respond

```
        0.01  0
```

and quit

```
        0
        0
```

The new mesh is in `m4`, use PlotMesh to look at it and note the boundary nodes. In particular, note the different color around the boundary. A more dramatic confirmation can be obtained by changing `7:mesh_%=75`.

In this manner any number of stringers can be attached to the plate. In deed, if each is given a separate segment number then each can also be given separate properties.

Adding Properties

To make the generic structure into a structure datafile, we need to add properties. That is, we need to state the type of material, the boundary conditions, and the loads.

Initiate GenMesh and from the main menu select the `Make Structure DataFile` option.

```
C>  genmesh
        8
        2
        m5
        m4
```

You are now given a summary of the mesh as

```
 2   Matls
41   Nodes
 0   BCs
 0   Forces
 0   materials
 0   SPcls
 4   boundary groups
```

The series of questions to be answered are the same as an earlier tutorial, so we will just give the responses.

```
1
ex_cs.1:  thin-walled shear beam
22
1  1  1  1
```

Boundary conditions are specified next.

```
3
3
3     0  0  0    0  0  0
0
```

corresponding to a fixed boundary condition at the left side. The boundary condition menu is recurring, so that many boundary conditions may be input. Note that ranges can overlap.

There is a vertical point force to be applied at Node 5, so type

```
41
1
5  5    0   1000   0    0 0 0   0
0
```

Finally, input the material properties.

```
5
2
1  10.0e6  4.0e6  0.1  2.50e-4  1   0.0833e-3  1.5    0.5
2  10.0e6  4.0e6  1.0  2.50e-4  0.0 0.1666     0.0833 0.0833
```

There are no special properties, so finish by typing:

```
0
0
```

Survey `<<ex_cs.1>>` and compare with `<<m5>>` to make sure the properties have been installed as expected.

2.8 Extruded Meshes

There is a class of 3-D meshes that can be viewed as a particular cross-section in the $x - y$ plane that is extruded in the z-direction. A circular cylinder is a simple example: the cross-section shape is a circle and the cylinder is formed by moving this (as a generating surface) along the z-axis. By adding a mapping function, the variety of shapes can be increased substantially. `GenMesh` has two such generic meshes, one which can be considered as a generalized cylinder and the other as a generalized dome; the essential difference between the two is that the latter is closed at one end.

These 3-D structures are usually thin-walled with stringers or stiffeners added for extra support. These are rather complex modules with a significant amount of data input; our first tutorials therefore will concentrate on the meaning of the input terms by creating a triangular box beam with a half-cylinder attached. The last tutorial considers creating a thin-walled structure that resembles a multi-sparred wing.

Thin-walled Reinforced Structure

This structure will be constructed in three stages. First will be the folded plate structure comprising the shell, followed by a frame structure of similar proportions. The final stage will consist of merging the two structures. Unlike the generic meshes of the previous sections, the meshes created here are physical from the beginning.

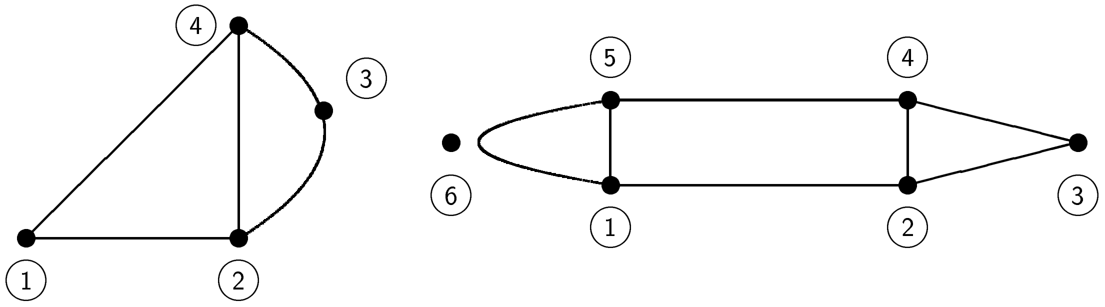


Figure 2.9: Cross-section of extruded structures.

I: Folded Plate Structure

The structure will look like that of Figure 2.9. It will have four control points: three to define the triangle, and a fourth to define the curved surface.

Run the program and choose the name for the mesh as `<<m1>>`

```
C>  genmesh
    4
    2
    m1
```

and you are given the mesh types

```
       MESH types:
                0: return
                << 3-D Extruded shapes >>
                2: Dome      (closed ended)
                   22:
                   24: diag  (closed ended) no stringer
                3:
                4: General   (open   ended) two_planes
                   42:       (open   ended) no stringer
                   44: diag  (open   ended) no stringer
                << 3-D Frame shapes >>
                5: General Rect (open ended) two_planes
                6: 3-D Arbitrary
                 :
```

We will look at the diagonal open-ended shape without stringers

```
       44
```

The first series of information is to allow characterization of the cross-section in the $x - y$ plane. We are asked for the number of control points and their coordinates

```
    TYPE: # of control points
```

Input four control points

```
       4
```

Now input the coordinates of each point

```
                             front        back
    Type:   point #   |  x1|y1|z1  |  x2|y2|z2
```

This gives information about two planes (front and back) and the extrusion will be done between the two. Note that the two planes need not be parallel. For now, make the control points in both planes the same

```
       1   0.0  0.0  0.0   0.0  0.0  8.0
       2  10.0  0.0  0.0  10.0  0.0  8.0
       3  14.0  6.0  0.0  14.0  6.0  8.0
       4  10.0 10.0  0.0  10.0 10.0  8.0
```

This sets the length of the extrusion as `8.0`. Thus, unlike the generic meshes, these meshes are physical from the beginning; in fact, the control points can be used to locate the sub-structure for attachment to other sub-structures.

Next is information about the shape of the structure.

```
SEGment styles:   1=line  2=     3=bezier  4=arc
TYPE: # of connected segments
```

In a complex structure with many control points, there are a variety of ways that the points can be attached to each other. Each connection forms a segment and each segment can be of a different style. The `style` of connection specifies the geometric shape of the segment. The current implemented forms are

```
straight line  :        1
bezier   curve :        3
circular arc   :        4
```

Both the *Bezier* and *arc* actually require three control points to form it, We will connect Points 2, 3, and 4 with an arc, Points 1&2, Points 2&4, and Points 4&1 with straight lines for a total of five segments. Thus respond

```
5
```

Now input which control points are connected and some of the attributes

```
TYPE:  point_1  |  point_2  |  # divs  |  style  |  plate tag
```

A segment is not necessarily a single plate element and the `# divs` allows specification of the number of modules connecting the two control points. Note that multiple connections can be made to the control points. Respond

```
1   2   8   1   1
2   3   10  4   2
3   4   10  4   3
2   4   8   1   4
4   1   8   1   5
```

This gives the flat faces `8` modules across and the curved surfaces `10` each. Each plated segment will have a different tag for easy identification.

The final piece of information to be specified deals with how the cross-section is extruded in the z-direction

```
Z_EXTRUSION
TYPE:      #  z_modules
```

This is the number of modules in the extruded direction. Put 10 modules in the z-direction

```
10
```

Exit the program by typing

```
0
```

Now survey the mesh `m1` using the utility `PlotMesh`; change the orientation by pressing `u/U` and `v/V`. The mesh should look like Figure 2.10(a).

`GenMesh` has the ability to produce the PostScript file necessary to make hard copy version of the mesh files in this form; a later tutorial shows how it is done.

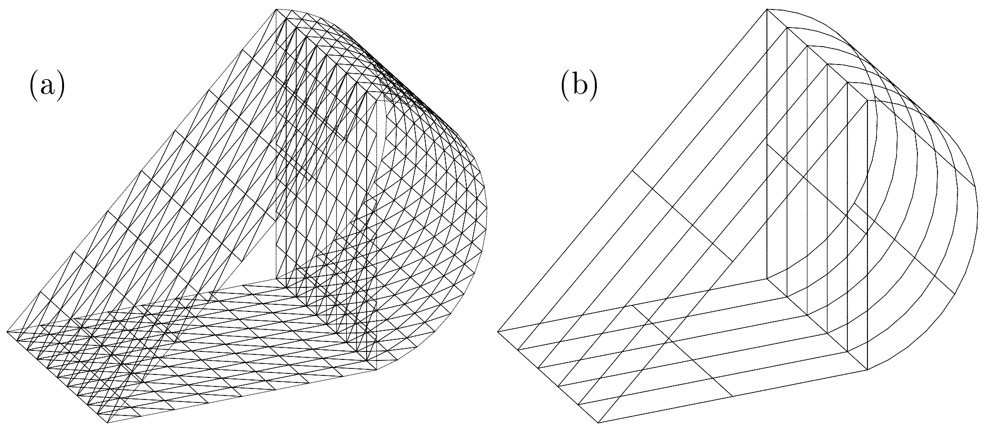


Figure 2.10: Extruded structures. (a) Folded plate. (b) Frame.

II: Extruded Frame Structure

We will make a frame structure of similar shape to the folded plate structure just constructed. A merging of the two structures will then produce a reinforced thin-walled structure.

Run the program and choose the name for the mesh as `<<m2>>`

```
C>  genmesh
    4
    2
    m2
```

and from the menu of mesh types

```
MESH types:
         0: return
         << 3-D Extruded shapes >>
         2: Dome      (closed ended)
            22:
            24: diag  (closed ended) no stringer
         3:
         4: General   (open   ended) two_planes
            42:       (open   ended) no stringer
            44: diag  (open   ended) no stringer
         << 3-D Frame shapes >>
         5: General Rect (open ended) two_planes
         6: 3-D Arbitrary
          :
```

choose the general rectangle with two planes;

```
        5
```

The first series of information is to allow characterization of the cross-section in the $x - y$ plane. First is the number of control points

```
    TYPE: # of control points
          4
```

Then the coordinates and the joint tag

```
                          front        back
    Type:   point #   |  x1|y1|z1   |  x2|y2|z2   |  joint tag
```

This gives information in two planes (front and back) and the extrusion will be done between the two. Note that the two planes need not be parallel. Control points act as joints and therefore it is useful to be able to associate a lengthwise (in the z-direction) property with them that is different from other members. For now make both cross-sections the same and give each joint a different property.

```
        1   0.0  0.0  0.0   0.0  0.0  8.0  11
        2  10.0  0.0  0.0  10.0  0.0  8.0  12
        3  14.0  6.0  0.0  14.0  6.0  8.0  13
        4  10.0 10.0  0.0  10.0 10.0  8.0  14
```

This sets the length of the frame structure as `8.0`. The tags are set different from the plated structure in anticipation of merging the two meshes. Again, unlike the generic meshes, these meshes are physical from the beginning; in fact, the control points can be used to locate the sub-structure for attachment to other sub-structures.

In a complex structure with many control points, there are a variety of points that can be attached to each other. Each connection forms a segment; the choice of connections are:

```
        SEGment styles:   1=line  2=     3=bezier  4=arc
        TYPE: # of connected segments  | # z_divs  |  z_rate
```

Both the *Bezier* and *arc* actually require three control points, we will connect Points 2, 3, and 4 with an arc. The remaining two numbers gives the number of elements in the z direction and the rate at which transverse members are added, respectively. Thus respond

```
        5   10  2
```

Now input which control points are connected and some of the attributes

```
    TYPE:  point_1 | point_2 | # divs | style | t_tag  |  l_tag | rate
```

A segment is not necessarily a single element and the `# divs` allows specification of the number of elements connecting the two control points. The `style` of connection specifies the geometric shape of the segment. The current implemented forms are

```
straight line       :     1
bezier   curve      :     3
circular arc        :     4
```

The tags are for the transverse and lengthwise members, the rate is the rate at which lengthwise members are added along the segment. As a result, there will be lengthwise members associated with the joints, and also a set of intermediate lengthwise members. Note that multiple connections can be made to the control points. Make the discretization similar the folded plate, hence respond

```
1   2   8   1   15  16 4
2   3   10  4   15  16 5
3   4   10  4   15  16 5
2   4   8   1   15  16 4
4   1   8   1   15  16 4
```

This gives the straight segments `8` elements across and the curved segments `10` each. The flat and curved faces of the frame structure will have an intermediate lengthwise member. Note that if `GenMesh` has difficulty placing the intermediate members, then it will not place any for that segment.

Exit the program by typing

```
0
```

and survey the mesh `<<m2>>` using the utility `PlotMesh`; the mesh should look like Figure 2.10(b).

III: Merging the Meshes

We will combine `<<m1>>` with `<<m2>>` to produce a thin-walled structure reinforced by frame members.

Initiate `GenMesh` and choose the merge option

```
C>  genmesh
        7
        2
        m12
```

You are now asked to

```
TYPE: infileNAME_1 -->
```

It makes a difference (for some of the merge options) which mesh is given first because that is the one to which the second is subservient. Respond

```
        m1
```

Similarly for the second filename

```
        TYPE: infileNAME_2 -->
```

Respond

```
        m2
```

A brief summary of the two meshes is displayed. The mode of how the two meshes are to be attached must be decided

```
        CONNECTion modes:   MODEL_1 <-- MODEL_2
                            0=continue
                            1=node to node
                            2=group to surface
                            3=intersecting surfaces
              INPUT:   mode
```

Note that Model 2 is attached to Model 1. The most basic form of attachment is #1 where, essentially, the two structure are stitched together node by node. The simplest is #3 where any close nodes are automatically connected. We will use the third mode

```
        3
```

and you are asked about the attachment nodes

```
      Intersecting Sections:
      INPUT:  proximity   | release as BC 0=none 1=all 2=interior
```

The 'proximity' number is the radius of a sphere and all nodes found inside this sphere are collapsed into a single node. It can be specified large or small depending on the density (or coarseness) of the nodal points; generally it is specified at about 1/100 of the smallest element size. In the present case, care was taken to ensure that many of the nodes coordinates of the frame coincided with those of the folded plate, and hence a very small proximity can be used. When two meshes are joined, what were formerly boundaries now could become interior regions. We therefore may wish to release them and not consider them as boundaries (or special collections of nodes) anymore. In the present case we will keep all former boundaries. Choose

```
      .001  0
```

Quit by typing

```
       0
       0
```

An important point to note about the mesh file is how, through the use of the tags, the identity of the individual sub-structure meshes is retained. This is reflected in the different colors used in PlotMesh.

Reinforced Structures

GenMeshhas the option to create reinforced structures directly. The inputs to these are more involved than in the foregoing, so generally, the approached of creating reinforced structures by merging separately created folded-plate and frame structures is preferable.

Two examples are illustrated, a flat plate and a two-sparred wing.

I: Reinforced Plate

Run the program and name the mesh as `m1`

```
   C>  genmesh
       4
       2
       m1
```

From the menu of mesh types

```
       MESH types:
                0: return

                 : 3-D Extruded shapes
                2: Dome      (closed ended)
               22
               24: diag      (closed ended) no stringer
                3:
                4: General   (open   ended) two_planes
               42:           (open   ended) no stringer
               44: diag      (open   ended) no stringer

                 : 3-D Frame shapes
                5: General Rect (open ended) two_planes
                6: 3-D Arbitrary
               10:
```

choose the general open-ended shape

```
       4
```

The first series of information is to allow characterization of the cross-section in the $x - y$ plane in terms of the control points and their coordinates.

```
    TYPE: # of control points
```

There are only two control points

```
        2
```

Now input the coordinates and the tag

```
                          (front)       (back)
    Type:   point #   |  x1|y1|z1  |  x2|y2|z2  |  joint tag
```

This gives information about two planes (front and back) and the extrusion will be done between the two. Note that the two planes need not be parallel. Most control points act as joints and therefore it is useful to be able to associate a lengthwise (in the z-direction) stiffener with them. For now we will make both stiffeners the same

```
        1    0.0   0.0   0.0    0.0   0.0 18.0    11
        2   10.0   0.0   0.0   10.0   0.0 18.0    11
```

This sets the width of the plate as `10.0` and its length as `18.0`. Thus, unlike the generic meshes, these meshes are physical from the beginning; in fact, the control points can be used to locate the sub-structure for attachment to other sub-structures.

In a complex structure with many control points, there are a variety of points that can be attached to each other. Each connection forms a segment; for our plate there is only one segment hence in response to

```
        SEGment styles:   1=line  2=     3=bezier  4=arc
        TYPE: # of connected segments
```

respond

```
        1
```

Now input which control points are connected and some of the attributes

```
    TYPE:  point_1  |  point_2  |  # divs  |  style  |  plate tag
```

A segment is not necessarily a single plate element and the `# divs` allows specification of the number of modules connecting the two control points. The `style` of connection specifies the geometric shape of the segment. The current implemented forms are

```
        straight line        :     1
        bezier   curve       :     3
        circular arc         :     4
```

Note that the last two of these actually require three control points; this will be demonstrated in the next tutorial. We are interested in a flat plate so respond

```
        1   2   8   1   1
```

This gives the plate 8 modules across and all plate elements will be tagged as 1

Between the control points, there may be additional stiffening elements coinciding with the boundaries of the element modules. The reinforcers can be transverse and lengthwise. The attributes of these are being asked for in

```
INPUT:   trans tag  |  length tag   |  length divs rate
```

The lengthwise reinforcers must coincide with module boundaries hence the last item just specifies how often they are realized. We will make both stiffeners look different and put the lengthwise ones at every second module

```
12  13   2
```

If there were multiple segments, then both sets of input would be input for each segment, allowing each segment to be different.

The final piece of information to be specified deals with how the cross-section is extruded in the z-direction

```
Z_EXTRUSION
TYPE:      #  z_modules  |  trans rate
```

The first is the number of modules in that direction, and the second is the rate at which the transverse stiffeners are placed. Again the maximum rate is to place them at every module, that is, a rate of 1. We will put 12 modules in the z-direction and place the stiffeners at every third module

```
12   3
```

Exit the program by typing

```
0
```

Survey the mesh <<m1>> using the utility PlotMesh, change the orientation by pressing u/U and v/V. The exploded version of the mesh should look like Figure 2.11.

GenMesh has the ability to produce the PostScript file necessary to make hard copy version of the mesh files is this form; the next small tutorial shows how it is done.

Run the program and choose the Write PostScript option

```
C>  genmesh
    700
```

A PostScript file will be generated and we have the choice as to how to store it

```
SAVE new file as:
0=return  1=GMesh.PS  2=NEW name
```

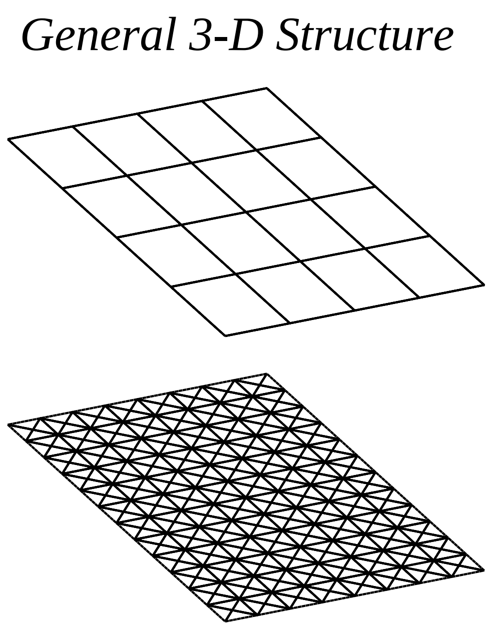


Figure 2.11: Exploded view of a reinforced flat plate.

Specify the new name as `m1.ps`

```
2
m1.ps
```

For the input filename give

```
m1
```

We are given a choice as to the output format

```
OUTPUT format:
        0=return
        2=Shrunk mesh coordinates
        4=PS plot file  (B/W)
        5=PS plot file  (color)
        6=PS plot solid (color)
        7=PS plot solid (grey )
```

`Option 2` does not produce any PostScript output, it just contains the tagged coordinates of the line segments forming each element. This would be useful if some other rendering scheme was used. Color PostScript will appear as grayscale on a black and white output device hence sometimes it may be preferable to specify the black and white option.

```
5
```

We can manipulate the image through the following menu:

```
CHANGE default:
        0=Write PS file
        1=Position mesh
        2=Change orientation
        3=Move tagged surfaces
        4=Position or change title
        5=Change pen
```

For 3-D meshes, it is usually necessary to arrange the perspective so as to get a good view. We will tilt the plate about the x and y axes

```
2
25   25   0    1.0
```

The last item, the `shrinkage`, shrinks each element about its centroid; this is a way of ensuring that there are no missing elements. An alternative way of viewing a composite mesh is to move the tagged surfaces apart. This is very useful especially for thinned-walled reinforced structures.

```
3
```

We will separate the plate elements (tagged 1) from the remaining frame elements

```
INPUT:   tag#  |  Xmove  |  Ymove      <0 0 0 ends>
          1       0.0       -10.
          0       0          0
```

Notice in this case we are dealing with page coordinates. This is a recurring menu so many of the surfaces can be moved. Finally, the title (as read from the mesh header group) can be changed and repositioned.

```
4
Flat Plate
1   7.5   1.0
```

The PostScript file can now be written and we can exit the program

```
0
0
```

The PostScript file is named `<<m1.ps>>` and this can be sent to a PostScript printer or displayed on a screen by use of an interpreter such as `Ghostscript`.

In order to render the figure, it is necessary to also specify other information such as the pen thickness. This information is kept in the `<<genmesh.cfg>>` file. Its format is

```
elem:  max #
posn:  X  Y
pens:  thickness
Rotn:  X  Y  Z  shrink
caps:  X  Y  size
type:  mesh type
```

Generally, once these are set it should not be necessary to adjust them again.

II: Two Spar Wing

The second example is of a multi-spar wing. The cross-section control points are shown in Figure 2.9(b).

Run the program and choose the open-ended z-extrusion option.

```
C>  genmesh
    4
    2
    m1
    4
```

The first series of information is to allow characterization of the cross-section in the $x-y$ planes. There are six control points, the input information is

```
6
1     0.0     0.0  0.0     0.0   0.0  500.0   11
2   140.0    10.0  0.0   140.0  10.0  500.0   12
3   200.0    20.0  0.0   200.0  20.0  500.0   12
4   140.0    30.0  0.0   140.0  30.0  500.0   13
5     0.0    40.0  0.0     0.0  40.0  500.0   12
6  -100.0    20.0  0.0  -100.0  20.0  500.0   12
```

We could have used the second group of coordinates to give the wing a taper; however this will be left to a separate tutorial that demonstrates how 3-D meshes can be remapped. Typing lots of numbers like this can be very tedious and prone to error, it is understood that all these numbers would be typed initially in a driver (response) file and `GenMesh` actually run in batch mode.

In a complex structure with many control points, there are a variety of points that can be attached to each other. Each connection forms a segment; we now connect these control points to form our wing cross-section.

```
SEGment styles:   1=line  2=     3=bezier  4=arc
TYPE: # of connected segments
```

We will make 8 segments. The information to be typed is

```
TYPE:  point_1 | point_2 | # divs | style | t_tag  |  l_tag | rate
TYPE:  trans tag  |  length tag  |  length divs rate

   8
   1   2   6   1   1
  15  16   2
   2   3   3   1   1
  17  18   3
   3   4   3   1   2
  17  18   3
   4   5   6   1   2
  12  13   2
   5   6   4   3   2
  17  18   2
   6   1   4   3   1
  12  13   2
   1   5   4   1   3
  12  13   4
   2   4   4   1   3
  12  13   4
```

Note that the segments connecting points `5-6-1` form a Bezier curve.

The final piece of information to be specified deals with how the cross-section is extruded in the z-direction

```
Z_EXTRUSION
TYPE:      #  z_modules  |  trans rate
```

We will make the mesh 12 modules long and place the transverse stiffeners (ribs) at every second module

```
12  2
```

Note that the rate number should divide evenly into the total number of modules. Quit the program

```
0
```

We will now show how a 3-D mesh such as the wing can be further modified by scaling. The physical dimensions have already been set, where this option is useful is in tapering the extrusion.

```
C> genmesh
   5
   2
```

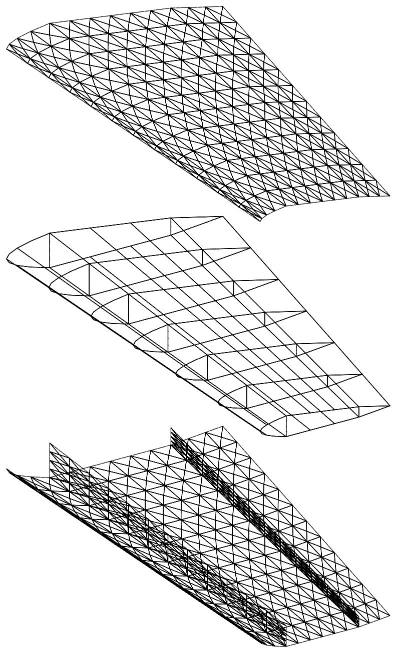


Figure 2.12: Two-spar wing.

```
m2
m1
1
1
```

We will taper the wing in the $x-z$ planform and the $y-z$ depth.

```
1
```

A reminder of the format is given

```
SCALE wrt Z
          |x
  1    x2|________ x1
          |        |
  0    x3|________|x4____z

  TYPE:   x1 | x2 | x3 | x4        <1 1 0 0>
```

The hint numbers will leave the mesh unchanged. We will give a slight taper as

```
0.8   1.0   0.0   0.2
```

For the $y-z$ plane we get

```
 SCALE wrt Z
          |y
   1    y2|________ y1
          |        |
   0    y3|________|y4____z

   TYPE:   y1 | y2 | y3 | y4         <1 1 0 0>
```

Again, we will give a slight taper

```
       0.8   1.0   0.0   0.2
```

Finally for the $z - z$ plane

```
 SCALE wrt Z
          |z
        z2|________ z1
          |        |
        z3|________|z4____z
          0      1
   TYPE:   z1 | z2 | z3 | z4         <1 0 0 1>
```

Note the hint values in this case, these are the ones we will use

```
       1.0   0.0   0.0   1.0
```

It is worth experimenting with these to see the various ways the extrusion can be manipulated. For example, by making the first line `1.2 1.0 0.0 0.4` a swept-back wing can be generated. Quit the program

```
       0
       0
```

Now survey the mesh `<<m2>>` using the utility `PlotMesh`. The exploded version of the mesh should look like Figure 2.12.

2.9 Solid Meshes

The purpose of this tutorial is to show how `GenMesh` can be used to mesh solid regions. In particular, we will run it to generate a file containing the same information as in the file `<<ex_ep.1>>` that will be used for the elastic-plastic tutorial.

Making the Mesh

Run the program and select the `Solid Shapes`

```
   C>  genmesh
          23
          2
          m1
```

The program responds with the choice of shapes

```
     MESH types:
              0: Return
               : 3-D Block shapes
             22: Tet      6 elem block
             24: Hex_8    1 elem block
             26: Hex_20   1 elem block
               :
             52: Tet      6 elem Cut-out
             54: Hex_8      cylinder
             56: Hex_20     cylinder
             58: Hex_20     cyl in rect
             66: Hex_20     notch
    SELECT -->
```

Choose the Hex_20 element block

```
       26
```

We must now tell the program how many elements we want

```
       TYPE:   # of modules    X-dir  |  Y-dir  | Z-dir  | tag #
```

Keep in mind that scaling the mesh is a different operation than choosing the number of modules. The `tag #` can be used to keep track of the sub-structures used in forming a compound mesh, it is also used to associate different material properties with different groups of elements. Respond

```
       10  1  1  1
```

You are now asked

```
        TYPE:  X-dim | Y-dim | Z-dim
```

This is the scaling on the mesh. Respond

```
        10   1.0  0.5
```

Back at the main menu choose to `Quit` with

```
         0
```

Use PlotMesh to view the mesh.

Adding Properties

To make the generic mesh into a structure datafile, we need to add properties. That is, we need to state the type of material, the boundary conditions, and the loads. We have already done this in a previous tutorial, hence the following will dwell only on the significantly different parts.

Initiate `GenMesh` and from the main menu select the `Make Structure DataFile` option.

```
   C>  genmesh
           8
           2
           m2
           m1
```

You are now given a summary of the mesh as

```
          1   Matls
        128   Nodes
          0   BCs
          0   Loads
          0   matls
          0   spcls
          0   boundary groups
```

The series of questions to be answered are the same as the earlier tutorials, so we will just give the responses. We will specify the boundary conditions as fixed on one end by isolating a thin disk of volume

```
        3
        4
        0 0 0   0 0 0
       -0.01  0 0    0.01 0 0
        20
        0
```

The boundary condition menu is recurring, so that many boundary conditions may be input.

The applied load data is to be specified next

```
41
```

and we are given the options

```
APPLIED LOADS:
CHOOSE:    0=return
           1=nodal load  (sequence)
           2=nodal load  (bc info )
           3=nodal load  (bc group info )
             32=nodal load  (bc group info ) (1/2 ends)
             34=node traction (bc group) (1/2 ends)
           4=nodal load  (nearest xyz)
           6=cylinder    (volume)
             6=cylinder resultant (volume)

          11=equal force       to tagged surface
          12=uniform pressure  to tagged surface
             122=uniform load  to tagged frame
             124=uniform pressure to Hex20 surface
          15=traction distribution
             152=traction distn (coords in file)
          16=traction distribution    from file
           9=interrogate
```

We will again isolate a thin disk of volume to distribute a uniform pressure over the right end.

```
 124
-9.99  0 0   10.01 0 0
 20  1.0
 0
```

Note that an applied pressure is treated differently for the Hex20 element than for the other elements — the consistent load formulation is used — consequently both positive and negative nodal load values will appear in the structure datafile. The applied loads menu is a recurring one that allows multiple loads to be added.

The material properties are specified by

```
5
1
1    10.0e6  4.0e6  1.0  2.50e-4  1   1.0  1.0  1.0  1.0
```

The elastic-plastic material properties are specified as part of the `Specials` group. Choose

```
6
```

to be given the recurring input

```
TYPE:   Code  |  c1  |  c2  |  c3    <0 0 0 0 to end>
```

We will also specify full integration on the element matrices. Thus respond

```
2341   30e3  0.5e6   0
2410   1     2       3
0 0 0 0
```

Exit the properties section with

```
0
```

Back at the main menu, we will choose to reduce the bandwidth with

```
6
2
m3
m2
6
1
-100  0  0
2
0
0
```

Survey `<<m3>>` and make sure the properties have been installed as expected. This should be identical to the file `<<ex_ep.1>>` on the disk. Also, use PlotMesh to view the mesh.

2.10 Some Hints

The following are some hints that may prove useful:

- Once the procedures for the problem have been established, convert them into a driver (script) file and use the program in batch mode.

- Whenever possible, use the boundary nodal groups and element tags to realize the structure datafile. This will make the driver files independent of the mesh density and therefore more versatile.

- When smoothing a mesh, additional biasing can be achieved by moving individual nodes to a desired location and then specifying it as a boundary node. In this way, the surrounding nodes will adjust appropriately.

- When making a compound mesh, it is best to leave the property specification until the second to last operation, and the node re-numbering until the very last operation.

- For very large and complex meshes, make each sub-structure complete and self-contained including its absolute positioning. In this way, the assembling is done as the final stage.

Chapter 3

Basic StaDyn Tutorials

This chapter is a collection of short tutorials for running StaDyn. It is advisable the first time through, that all the tutorials be completed and that all the instructions be followed strictly. This is because some of the later tutorials make reference back to the earlier ones.

The program StaDyn (STAtic and DYNamic analysis) is capable of doing the static and dynamic analysis of 3-D structures formed from a combination of frame members and folded plates. Its main capabilities involve determining the member displacements and loads, and structural reactions for:

- Static loading
- Stability analysis
- Vibration analysis
- Forced frequency loading
- Applied transient load history

The tutorials in this and the next chapter cover examples from each of these. The program is menu driven in such a way that its functioning is apparent. This facilitates early understanding of it, but can be somewhat intrusive after familiarity has been developed. It will be shown, however, that it is possible to execute the program without these menus and thus speed all operations.

As StaDyn proceeds, it creates a number of files among which are:

```
stadyn.cfg      stadyn.stf      stadyn.dis
stadyn.log      stadyn.mas      stadyn.snp
stadyn.out      stadyn.cms
stadyn.dyn      stadyn.geo
                stadyn.lod
```

The `LOG` file echoes all the input responses as well as having some extra information that might prove useful during post analysis of the results. The `OUT` file is the usual location for StaDyn output. The second column of files are associated with various system matrices such as the stiffness matrix and the mass matrix — these are left on disk in case they may be of value for some other purpose. They are in binary form. The last column of files are output files. They too are in binary form for compactness but can be read for further post-processing.

Regularly have a look in the `StaDyn.LOG` because there is much information to be found there. Although this information is probably of no direct interest now, it is good planning to get used to interpreting this file in anticipation of more complicated problems.

3.1 Static Example

StaDyn is designed to run as a console (or command window) program under the various flavors of `MS Windows`. Note that all instructions are case insensitive; we will vary the case only to help make the instructions clearer. The class of static problems solved by StaDyn is

$$\Big[[K_E] + \chi[K_G]\Big]\{u\} = \chi_1\{P\} + \chi_2\{G\}$$

where χ_i are various scale factors and $\{G\}$ is the gravity load vector.

Getting Started

To run the program, type (at the 'C prompt')

```
   C>  stadyn
```

(note the name of the executable). The opening menu is

```
    MAIN menu:
           0:  Quit

           2:  Read in structure DataFile

          30:  Form system matrices

          60:  Static loading
          70:  Forced frequency loading
          80:  Transient loading
          90:  Eigen Analysis

         700:  PostScript plot files
         800:  System services
         911:  Ikayex information
  SELECT-->
```

Quit by typing

```
       0
```

There are two files worth looking at. The first is the log file `StaDyn.LOG`; use the `list` utility to peruse it. It contains

```
 @@ StaDyn version 4.40, June 2002
 @@ DATE:  8-13-2002      TIME: 11:11
 @@ MAXimum storage   :     2000000
 @@ MAXimum elements  :        5000
 @@ MAXimum nodes     :        5000
 @@ MAXimum force incs:        5000   200
 @@ ITERmax           :          16
 @@ rtol              :    1.000000E-10
 @@ ALLOCATION succeeded
          0 ::MAIN
 @@ StaDyn OK, ended from MAIN
```

The second is `StaDyn.CFG`. This is the configuration file and has in it

```
     2000000     ::MaxStorage
        5000     ::MaxNode
        5000     ::MaxElem
        5000  200    ::MaxForce
           2      ::ilump
          16      ::itermax
          1.0000000E-010      ::rtol
 @@ DATE:  8-13-2002      TIME: 11:11
```

This file allows changing the run time settings of the dimensions of the arrays. If you get a memory allocation failure, this is the place to make the adjustments.

A Static Analysis

Since we know the data in the file `EX_PS.1` to be correct, we will use it to do a simple static analysis. Run StaDyn and read the structure DataFile

```
   C>  stadyn
       2
       ex_ps.1
```

Before we can obtain the static solution, we must first form the system matrices. Select

```
       30
```

and you are given the choices

```
       FORM System Matrices: sub-menu
            31: [K_E]            <static >
            32:
            33: [K_G]
            34: [K_E],[K_G]      <buckling>
```

```
     35: [M]
     36: [K_E],[M]         <dynamic>
     37: [K_E],[K_G],[M]   <all>
```

Select

```
31
```

and make the choice

```
CHOOSE STIFFness storage:
         0=DEFault               >>MEMory only>>
         1=BINary  [prof]        >>StaDyn.STF >>
         2=ASCii   [prof]        >>StaDyn.OUT >>
         3=ASCii   [NxB]         >>StaDyn.OUT >>
         4=BINary  [NxB]         >>named      >>
         5=DEFault + load        >>StaDyn.OUT >>
        11=BINary  [prof+locn]   >>named      >>
```

Since this is the first example, echo the stiffness matrix to `StaDyn.OUT` just to see what it looks like.

```
3
```

Since the loads were already read from `EX_PS.1` we can now obtain a static solution by typing

```
60
```

and there are two problem types

```
STATIC Problems: sub-menu
     61: Distributed loadings
     62: Pre-stress  dP
     63: Pre-stress   P
     66: All-node loadings
```

In the first, the loads specified in the structure datafile are the applied loads, while in the second those loads are interpreted as pre-loading on the structure. Choose

```
61
```

Each major capability of StaDyn is divided into two parts: the analysis and the post-processing. The first determines the nodal degrees of freedom in the global coordinates, while the second allows post-processing of them to give strains or contours and so on — basically member or local level information. Thus, in response to

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

choose

```
    1
```

since this is the first time through. Sometimes it is convenient to have unit loads in the structure datafile and to use the analysis stage to scale the loads. In response to

```
    INPUT scales:  load  |  gravity
```

type

```
    1.0  0.0
```

After echoing some information on the screen, you are back to

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

Quit and get out of StaDyn by typing

```
    0
    0
```

That is all there is to it!

The output can now be surveyed by viewing the ASCII file `StaDyn.OUT`. This file is fairly well documented so there is little difficulty in interpreting it. The maximum displacement occurs at Node 6 with

```
   6  .400000E-03  .000000  .000000  .000000  .000000  .000000
```

Note that some of the nodes can have a rotation; this is a consequence of the type of in-plane element implemented — it has three degrees of freedom at each node, two displacements and one rotation. In the present case, the rotations are zero indicating that the element arrangement satisfies the patch test.

Also have a look in the `StaDyn.LOG` file to see the type of information it contains.

Using Script Files

Actually, the main reason for doing this simple example is to introduce a feature of `StaDyn` that underlies its design philosophy. One of the advantages of echoing information to the `LOG` file is that it allows the creation of a driver or script file. If you are not familiar with this idea then perform the following little tutorial. Make a copy of the log file by

```
C>  copy  stadyn.log  instatic
```

Now use an editor to edit `instatic`. The only editing you need do is remove blank lines and lines beginning with `@@`. What is left is the same collection of responses as on the previous page but now it is documented. After you exit your editor all you need do to run the example of this section is

```
C>  stadyn < instatic
```

Go ahead, do it. The screen will scroll and everything is done automatically for you.

After a while you will find yourself doing the same operations repeatedly. Then you can make script files for each of them. More important, the script file is a record of how you processed the data. In a week, a month or 6 months if you need to reconstruct what you did, it is all there. The idea is that the script file approach allows the user a greater variety of inputs and gives greater control of the program; quite complicated sequences of instructions can be pieced together to give very fine control over your data manipulations. It also allows the program to be used in batch mode and thereby blend in with your other programs more productively. This is the communication method used by `QED`.

To further encourage use of script files, a small utility is provided on the disk for stripping away the extra lines from the `LOG` file. To run it simply type

```
C>  strip < stadyn.log > instatic
```

3.2 Static Analysis of Trusses and Frames

The following are a collection of tutorials for static analysis. In each case, the information we seek can be obtained without substantial post-processing; we leave those tutorials until Chapter 3. The cases can also be thought of as validation tests for the program, which is why the results are compared to those in a reference.

Truss

We will use the structure datafile `EX_FS.1` to do a static analysis of a simple truss. Run the program and read the structure DataFile as done before

```
   C> stadyn
      2
      ex_fs.1
```

Before we can obtain the static solution we must first form the stiffness matrix

```
      30
      31
      0
```

Since the loads were already read from `EX_FS.1` we can now obtain the static solution by typing

```
      60
      61
      1
      1.0    0.0
```

We also want to look at the nodal forces, so choose the post-analysis

```
       2
```

and you are given the selection

```
     POST menu:
              0=return
                   LOCAL (member coords)
              2=nodal strain [ue]
              3=nodal stress
              4=nodal force
              5=element stress
              6=element (average) stress
              7=force sum
              8=specials
```

```
                 GLOBAL
           11=Global displacement
           12=Global loads
           14=Global assembled DoF loads
           15=Global assembled nodal loads
                 CONTOURs
           31=store displacement data
           32=store strain       data
           33=store stress       data
                        frames
           34=store displacement data
           35=store strain       data
           36=store stress       data
          100=    << select FRAMEs only >>
          200=    << select PLATEs only >>
```

Select the `Global loads` and quit `StaDyn` by typing

```
12
0
0
0
```

The output can now be surveyed by viewing the ASCII file `StaDyn.OUT`. This file is fairly well documented so there is little difficulty in interpreting it. Reference [1] gives for the displacement of Node 2

```
Node 2:        u: 0.457mm   v: 1.664mm   w: 0.0mm
```

whereas the program gives

```
    2     .457107E-03   .166421E-02    .000000   .......
```

These two sets agree. It is also worth noting that the nodal forces for Node 2 exactly add up to the applied forces

```
1   2     150000.    150000.     .000000   .......
2   2     -50000.     50000.     .000000   .......
sum       100000.    200000.     .000000   .......
```

Also have a look in the `StaDyn.LOG` to see the information it contains. It is good planning to get used to interpreting this file in anticipation of more complicated problems.

Frame analysis

The following example is for a simple plane frame structure. It is specified as plane by setting `IGLOBAL=22` and made of frame elements by letting `TYPE=3`.

```
ex_fs.2:  Balfour pp224.                     ::HEADER GROUP
22
1 1 1 1
end
2                                            ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
end
1                                            ::material #s
1 2  1
end
3                                            ::NODE GROUP
1  0.0 0.0  0.0
2  0.0 3.0  0.0
3  7.0 4.0  0.0
end
2                                            ::boundary condns
1  0   0   0   0   0   1
3  0   0   0   0   0   1
end
2                                            ::applied loads
1    0.0       0.0    0.0 0.0 0.0 -100.0e3 0.0
2  150.0e3  -150.0e3  0.0 0.0 0.0    0.0e3 0.0
end
1                                            ::material props
1   200e9 70e9   9000e-6 0.0 0.0     1.0 1.0 0.25e-3
end
0                                            ::SPECIALs GROUP
end
```

A copy of this is on the disk as `EX_FS.2`.

Run the program and read the structure datafile as before

```
C> stadyn
   2
   ex_fs.2
```

Before we can obtain the static solution we must first form the system matrices

```
30
31
0
```

Since the loads were already read from `EX_FS.2` we can now obtain the static solution by typing

```
60
61
1
1.0  0.0
```

We also want to look at the nodal forces, so choose the post-analysis and quit

```
 2
 12
 0
 0
 0
```

The displacements and rotation at Node 2 as given by Reference [1] are

```
u: 0.792mm   v: -0.297mm  zrot: 0.532e-3 rad
```

while those from the program are

```
2    .791626E-03   -.296860E-03  .......   .533942E-03
```

Both sets of numbers agree quite well. Here, it is worth noting that the internal nodal moments cancel to give a resultant of zero at Node 2.

Make a script file for this problem by typing

```
C>  strip < stadyn.log > instatic
```

We will use this for the following tutorials.

Grillage Analysis

The following example is for a simple grill structure and a copy of it is on the disk as `EX_FS.3`.

```
ex_fs.3:  Balfour pp311.                    ::HEADER GROUP
23
1 1 1 1
end
3                                           ::ELEMENT GROUP
1 3 1 2 2
```

```
      2 3 2 3 3
      3 3 2 4 4
      end
      1                                         ::material #s
      1 3 1
      end
      4                                         ::NODE GROUP
      1  9.0 -10.0   0.0
      2  9.0   0.0   0.0
      3  0.0   0.0   0.0
      4  9.0  12.0   0.0
      end
      3                                         ::boundary condns
      1  0   0   0   0   0   0
      3  0   0   0   0   0   0
      4  0   0   0   0   0   0
      end
      1                                         ::applied loads
      2    0.0  0.0  -360.0e3    0.0 -540.0e3 0.0   0.0
      end
      1                                         ::material props
      1    15e9  6e9  9000e-6 0.0 0.0    40e-3 80e-3 80e-3
      end
      0                                         ::SPECIALs GROUP
      end
```

Since this is a grill problem then IGLOBAL=23. The other point of interest is that the load is distributed and this is converted into effective concentrated loads by

$$P_2 = P_3 = \tfrac{1}{2} q_o L \,, \qquad T_2 = -T_3 = -\tfrac{1}{12} q_o L^2$$

where q_o is the uniformly distributed load on Member 2. Note that since Node 3 is fixed there are no applied loads there.

Once INSTATIC has been changed to reflect the new input filename, the problem can now be run simply by typing

```
   C> stadyn < instatic
```

The displacement and rotations at Node 2 as given by Reference [1] are

```
      z: -9.791mm   xrot: -2.376e-4 rad   yrot: 5.724e-4 rad
```

and those of the program

```
   2   .000000  .000000  -.979448E-02  -.237660E-03  .572668E-03 ...
```

Note that internal nodal forces in the z-direction are generated.

3-D Analysis

As another static example, StaDyn is demonstrated on a three-dimensional problem specified by taking `IGLOBAL=32`. All members are made of the same stock. A copy of it is on the disk as `EX_FS.4`.

```
ex_fs.4:  Balfour pp360.               ::HEADER GROUP
32
1 1 1 1
end
3                                      ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
3 3 3 4 3
end
2                                      ::material #s
1 2 1
3 3 2
end
4                                      ::NODE GROUP
1  0.0  0.0  0.0
2  1.0  0.0  0.0
3  1.0 -1.0  0.0
4  1.0 -1.0 -1.0
end
1                                      ::boundary conditions
1  0   0   0   0   0   0
end
1                                      ::applied loads
4   10.0e3    0.0   0.0   0.0    0.0   0.0  0.0
end
2                                      ::material props
1    209e9  80e9  200e-6 0.0  0.0  6.67e-9  1.67e-9 6.67e-9
2    209e9  80e9  200e-6 0.0 90.0  6.67e-9  1.67e-9 6.67e-9
end
 0                                     ::SPECIALs GROUP
end
```

Another point of interest is that the orientation of the last member is rotated 90^o.

Run the program as before, preferably using the script file. The displacements and rotations at Node 4 as given by Reference [1] are

```
u: 59.4  v: 3.64  w: 14.3   Rx: 0.454e-4  Ry: -0.0510  Rz: 0.0108
```

The values from the program are

```
  4   59.3474   3.58672   14.3254   .469410E-06  -50.9782  10.7602
```

The digits are in close agreement, but the orders of magnitude are off for the rotations. This is due to the fact that the reference uses a combination of kN and mm for its dimensions.

Gravity Loading

As a final static example, StaDyn's ability to compute gravity loading will be demonstrated. To make it more interesting, we will seek to find the total mass of the structure plus the location of the center of mass. For simplicity, all members are made of the same stock, nominally of aluminum. A copy of the structure datafile is on the disk as `EX_FS.5`.

```
ex_fs.5 gravity                         ::HEADER GROUP
32
1 1 1 1
end
4                                       ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
3 3 3 4 4
4 3 5 1 1
end
2                                       ::material props
1 3 1
4 4 2
end
5                                       ::NODE GROUP
1   0.0   0.0  0.0
2   0.0  10.0  0.0
3  10.0  10.0  0.0
4  10.0  10.0 10.0
5   0.0  -1.0  0.0
end
1                                       ::boundary conditions
5  0   0   0   0   0   0
end
1                                       ::applied loads
4   0.0    0.0   0.0   0.0    0.0   0.0  0.0
end
2                                       ::material props
1   10e6  4e6   1.0  2.5e-4  0.0    .1  .1  .1
```

```
    2   10e6  4e6   1.0  0.0e-4  0.0    .1  .1  .1
    end
     1                                    ::SPECIALs GROUP
     2120   0.0 -1.0   0.0
    end
```

The structure itself is comprised of members 1 to 3, Member 4 is a massless rigid support that will act effectively as the weigh scale. Three components of gravity may be specified, right now it is assumed to be acting in the vertical (y) direction; this information is specified in the `SPECIALS` category with a `CODE=2120`. Normally, a value such as 386in/s^2 would be specified when interested in weight loading, but now we want the mass so unity is used. Also, note that there are no applied loads (other than gravity).

Run the program and form the stiffness matrix as usual,

```
  C> stadyn
     2
     ex_fs.5
     30
     31
     0
```

Gravity acts on the mass and forms a load vector with components at every translation degree of freedom. That is, it appears as a load vector similar to $\{P\}$. Now we are in a position to solve the static problem.

```
     60
     61
     1
     0.0   1.0
     2
     12
```

All the information we require can the obtained from the global information for the connection between Member 5 and the structure at Node 1. Hence quit by typing

```
     0
     0
     0
```

For verification purposes, the displacements and rotations at Node 1 are given as

```
  1  .18750E-07 -.75000E-09 .62500E-08 .12500E-07 .0000 -.37500E-07
```

The information we seek is given in the `GLOBAL nodal forces` row for Element 4 at Node 5. The force and moments at this point are

```
        Fy = .750000E-02   Mx = -.125000E-01   Mz = .375000E-01
```

The mass is given as

$$M = W/g = F_y/g = .0075/1.0 = .0075$$

The x_c and z_c centroids can be calculated as

$$x_c = \frac{M_z}{F_y} = \frac{.0375}{.0075} = 5.0\,, \qquad z_c = \frac{-M_x}{F_y} = \frac{.0125}{.0075} = 1.67$$

Repeating the process, but this time with the gravity vector in the x direction, say, will allow the other coordinate of the centroid to be determined.

This completes the static introduction. These examples were devised mainly to give a simple means of accessing the way StaDyn does things. All the examples are such that you can verify them by hand or go to the source of the reference to get more details. Redo these examples and try some of the other features not explicitly mentioned.

3.3 Static Analysis of Plates

We will do two examples for the analysis of plates. The first is a static analysis where we look at the deflections and moments in a plate in flexure. The second is an example of a stress concentration in a bar with a hole.

Plate in Flexure

The file `EX_PS.2` describes a flat circular plate. The loading corresponds to a uniform pressure as computed in `GenMesh`. Thus the distributed load is implemented in its lumped form. Run `StaDyn` and read the data as before

```
C>  stadyn
    2
    ex_ps.2
```

You are informed that the size is $[387 \times 120]$; this is fairly large, so we will do the minimum of echoing. Before we can obtain the static solution, we must first form the stiffness matrix. Select

```
    30
    31
    0
```

The assembly stage for flexural and in-plane elements is significantly more time consuming than for the frame elements, so ellipses are shown in order to give an idea of how much time it will take. Since the loads were already read from `EX_PS.2`, we can now obtain a solution by typing

```
    60
    61
```

Each major capability is divided into two parts: the analysis and the post-processing. The first determines the nodal degrees of freedom while the second allows post-processing of them to give strains or contours and so on. Thus, in response to

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

choose

```
     1
```

since this is the first time through. Although `EX_PS.2` contains the loads, sometimes it may be convenient to scale them; this is done next

```
    INPUT scales:   load  |  gravity
```

Input

```
       1.0  0.0
```

and after echoing some information on the screen, you are back to

```
       CHOOSE:   0=return   1=analysis  2=post_analysis
```

Choose the post analysis

```
        2
```

You are now asked to choose the form of nodal output

```
     CHOOSE output:
     POST menu:
              0=return
                   LOCAL (member coords)
              2=nodal strain [ue]
              3=nodal stress
              4=nodal force
              5=element stress
              6=element (average) stress
              7=force sum
              8=specials
                   GLOBAL
             11=Global displacement
             12=Global loads
             15=Global assembled DoF loads
             16=Global assembled nodal loads
                   CONTOURs
             31=store displacement data
             32=store strain       data
             33=store stress       data
                           frames
             34=store displacement data
             35=store strain       data
             36=store stress       data
            100=    << select FRAMEs only >>
            200=    << select PLATEs only >>
```

By default, the displacements are stored as part of the analysis, for the post analysis we want the moments (or bending stress), so choose

```
       3
```

To quit type

```
        0
        0
        0
```

The output can now be surveyed by viewing the ASCII file `StaDyn.OUT`. This file is fairly well documented so there is little difficulty in interpreting it. The maximum displacement occurs at Node 68 and is

```
    68    .0000    .0000  .172764  -.406007E-08  .847288E-09  .0000
```

Theory [11, 24] gives that the maximum displacement at the center is

$$w(r) = \frac{pa^4}{64D}[1-(\frac{r}{a})^2]^2\,, \qquad D \equiv \frac{Eh^3}{12(1-\nu^2)}$$

For our parameters with $\nu = 0.25$ and $D = 888.9$, this becomes

$$w(r) = 0.1758\,[1-(\frac{r}{a})^2]^2$$

This gives a nice comparison, a difference of little more than 1%.

The computed (stress) moments at the center (Node 68) and at the edge (Node 78) are given in the file captioned as

```
  NODAL STRESS averages:
   Node:    Sxx     Syy     Sxy       Mxx        Myy        Mxy
    68     .000    .000    .000    7.9064     7.9064     .41452E-06
    78     .000    .000    .000  -14.238     -4.3198     .71264E-06
```

The theoretical values [24, 11] for the moments are

$$M_r(r) = \frac{pa^2}{16}[(1+\nu)-(\frac{r}{a})^2(3+\nu)]\,, \qquad M_t(r) = \frac{pa^2}{16}[(1+\nu)-(\frac{r}{a})^2(1+3\nu)]$$

This gives at the center and at the boundary, respectively,

$$M_r = M_t = 7.8125\,; \qquad M_r = -12.5\,, \qquad M_t = -3.125$$

Again, we get a nice comparison, especially at the center. We expect the values at the boundary to be off because we are replacing a circular boundary with a polygon. Improved results can be obtained by making a finer mesh [11].

In-Plane Loading

The file `EX_PS.3` describes a bar with a hole in it. The full bar is modeled, although because of symmetry it would be feasible to model only one-quarter of it. In this tutorial we will look at the effects of stress concentrations.

Run `StaDyn` and read the data as before

```
C>  stadyn
    2
    ex_ps.3
```

You are informed that the size is $[1511 \times 102]$; this is fairly large which is typical of problems involving stress concentrations. Form the stiffness matrix and do the static analysis

```
    30
    31
    0
    60
    61
    1
    1.0  0.0
```

After echoing some information on the screen, you are back to

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

Now choose the post analysis

```
     2
```

You are now asked to choose the form of nodal output

```
  POST menu:
           0=return
                LOCAL (member coords)
           2=nodal strain [ue]
           3=nodal stress
           4=nodal force
           6=element stress
           7=force sum
                GLOBAL
          11=Global displacement
          12=Global loads
                CONTOURs
          31=store displacement data
          32=store strain       data
          33=store stress       data
```

We want the stresses, so choose

```
    3
```

To quit type

```
    0
    0
    0
```

The output can now be surveyed by viewing the ASCII file `StaDyn.OUT`. This file is fairly well documented so there is little difficulty in interpreting it. The maximum stress occurs at Node 259 and the corresponding line is

```
 NODAL STRESS averages:
  Node:    Sxx      Syy        Sxy        Mxx       Myy       Mxy
   259   10052.  1729.1     -50.43     .00000    .00000    .00000
```

From a handbook on stress concentration factors [24] we get that the SCF should be

$$SCF = 3.8$$

This compares with the above value of

$$SCF = \frac{10052.}{1000/0.4} = 4.02$$

This type of difference is typical.

3.4 Complex Structure

We end these introductory tutorials with an example of a complex structure; that is, a structure which contains both frame and plate elements. The structure itself is simple being just a cantilevered shear beam lying in the $x - y$ plane with a single vertical applied load.

The mesh is given in `EX_CS.1` and was created as one of the tutorials in the previous chapter. Notice that we specify the global type as being `IGLOBAL=22`, that is, the shear beam will behave essentially as a plane frame. Remember, however, that if in doubt about the global type then just specify it as its most general form, that is, as `IGLOBAL=32`.

Analysis

Run StaDyn, read the structure datafile, form the stiffness matrix, and do the analysis

```
C>  stadyn
    2
    ex_cs.1
    30
    31
    0
    60
    61
    1
    1.0  0.0
```

After echoing some information on the screen, you are back to

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

Quit the program

```
    0
    0
```

The output can now be surveyed by viewing the ASCII file `StaDyn.OUT`. This file is fairly well documented so there is little difficulty in interpreting it. The maximum displacement occurs at Node 5 and is

```
  5   .185939E-03  .176154E-02  0.000   0.000  0.000  .332471E-03
```

An energy approach to this problem where the center panel is treated as having a constant shear flow [26] gives the maximum displacement at the load point as

$$v = \frac{Pc}{3EA} + \frac{PL^3}{3EAc^2} + \frac{PL}{Gtc} = (1.33 + 1.33 + 26.0) \times 10^{-4} = 0.0029$$

There is a difference, indicating that the actual structure is stiffer than that assumed in the simple theory. If, on the other hand, we instead treat the structure as a beam in bending, then the tip deflection is given by

$$v = \frac{PL^3}{3EI} = \frac{1000 \times 4^3}{3 \times 10^7 \times 8.5} = 0.00025 = 2.5 \times 10^{-4}$$

This relatively small value indicates that the shear deformation is a significant portion of the total deformation.

We will look at the stresses to see if that clarifies the assumptions. Run StaDyn and read the data as before

```
C>  stadyn
    2
    ex_cs.1
```

Choose the post analysis and look at the element stresses,

```
    60
    61
     2
     200
     223
```

To quit type

```
    0
    0
    0
```

Look at the output in `StaDyn.OUT`, it is clear that the shear stress is the dominant stress giving, for example, in Element 27

```
   27    -87.428  -153.40   1905.5          0    0    0
```

The shear stress in the elementary theory is given as

$$\tau = \frac{P}{ct} = \frac{1000}{4 \times 0.1} = 2500$$

From this it is clear that the stringers are supporting a significant part of the load.

PostScript Contours

Sometimes the full behavior of a structure is not apparent by looking at individual numbers at individual nodes. Plotting the data as a contour plot can often help and this tutorial shows the type of support given by StaDyn for accomplishing this.

We will assume that this is the first time through the analysis; run `StaDyn`, read the structure datafile, form the stiffness matrix, and do the analysis.

```
C>  stadyn
    2
    ex_cs.1
    30
    31
    0
    60
    61
    1
    1.0  0.0
```

After echoing some information on the screen, you are back to

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

Choose the post analysis

```
     2
```

You are now asked to choose the form of nodal output, we want to look at the contours of displacement, so choose

```
    31
```

This stores the response data in the file `StaDyn.OUT`; however, it is the contour module that will now post-process it. We get to that menu off the main menu by

```
    0
    0
    700
    2
    p1.ps
```

where we elected to name the file. The contour menu is

```
     CONTOURs:
            0: Quit
                 <<RENDER PS file>>
            4: Deformed shape
```

```
                 5: Contours
                      <<CHANGE defaults>>
                11: Position model
                12: Rotate   model
                13: Move     tagged surfaces
                14: Add      caption
                15: Toggle   tags
                16: Change   pens
```

At the moment we will insure that the orientation is correct but otherwise accept the default values. (Note that the default values are stored in the file `StaDyn.CTR`; this is an ASCII file that is easily edited.)

```
12
0  0  0
```

Choose to draw the contours.

```
5
```

The choice of outputs are:

```
     DoF:       1=u   2=v   3=w   4=Rx  5=Ry  6=Rz
             or  1=Exx 2=Eyy 3=Exy 4=Kxx 5=Kyy 6=Kxy
             or  1=Sxx 2=Syy 3=Sxy 4=Mxx 5=Myy 6=Mxy
  Mesh:          1=Yes 0=No -1=Black
  Legend:        1=Yes 0=No -1=Black

  INPUT:  DoF  | scale | mesh ? | legend ?
```

Remember we have already stored the displacements, hence it is the first line that applies. Contour the vertical displacements, display it superposed on the mesh, and show the legend all without exaggerated scale

```
2  1.0 1 1
```

Quit the program

```
0
0
```

To obtain the other displacement contours, we need only re-enter the PostScript subroutine off the main menu. To get stress contours, however, we would first need to post-process the displacement data off the analysis menu.

Look at the output in `p1.ps`, this is the PostScript file of the contours. This can be sent directly to a PostScript printer, or can be viewed using the GhostScript

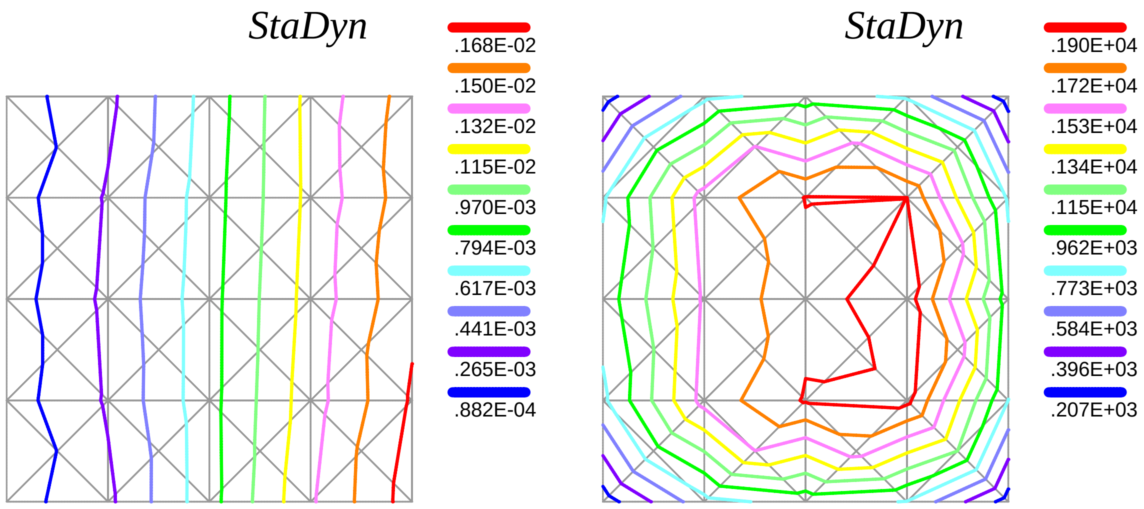


Figure 3.1: Contours of vertical deflection and shear stress.

program. The file will produce color contours as shown in Figure 3.1. It is clear that slightly more deformation occurs at the lower right edge where the load is applied.

Make a script file for the postprocessing and look at the stresses. As shown in Figure 3.1(b) the shear stress distribution is highly non-uniform. Also look at the bending stress σ_{xx}.

The configuration for the contour plots is stored in the file `StaDyn.CTR`. This has the organization

```
::geometry
::position X Y
::pen thickness mesh, contour
::rotations X Y Z
::caption positions X Y and size
::modes mass
::sub-structure flags
```

Sometimes, it may be more convenient to make the adjustment through this file rather than through the program.

3.5 Some Hints

StaDyn is a complex program, so it is difficult to give tutorials that cover all of its features. Additional advanced tutorials are given in the next chapter, but the best approach to using it usefully, is to start with the examples given, modify them (slightly at first) until you are studying the problems of interest to you. Experiment!

The following are some hints that may prove useful:

- Once the procedures for the problem have been established, convert them into a script files and use it in batch mode.
- Whenever possible, use the boundary condition lines and the global-type variable to remove unnecessary degrees of freedom. This will reduce the size of the system of equations to be solved and improve performance.
- The input structure file is the source of all that `StaDyn` can do, so generally, it is required that this file be read in before any of the processing functions can be used.

One feature of StaDyn of note, is that it is fairly open, meaning that you can access many of the system matrices. When there is a problem, you may consider looking at the system matrices directly.

Chapter 4

Advanced StaDyn Analyses

In addition to static analyses, StaDyn is designed to perform dynamic (transient, vibration, forced frequency) and stability (buckling) analyses. This chapter presents a few tutorials that cover the basics of these analyses.

Obtaining the buckling loads and buckled mode shapes for a 3-D structure is an eigenvalue problem similar to finding the vibration natural frequencies and mode shapes. In fact, once the appropriate matrices are established, StaDyn treats the two problems almost identically. The number of modes possible is equal to the number of free degrees of freedom in the system. But, usually, it is the lower loads and frequencies of most interest, therefore, it is prudent to keep the system size small. However, to get accurate lower loads (or frequencies) it is necessary to introduce more degrees of freedom (and thus be required to evaluate the higher loads also.) As an approximate rule of thumb, a system size N gives about $N/2$ good values of buckling load. This must be treated with caution since it depends on the location of the nodes, and also if the mode shapes themselves are of interest. StaDyn gives a choice of using the Jacob rotations or the sub-space iteration schemes to solve the eigenvalue problem; the latter is the appropriate one for large systems.

A different set of issues arise when the applied loading is transient; here the equations of motion must be integrated step by step. StaDyn uses the implicit Newmark integration scheme for this because it is an unconditionally stable algorithm.

4.1 Stability Analysis

All stability analyses occurs in two stages, the pre-buckle analysis and the buckling analysis. The onset of buckling depends on the presence of the in-plane stresses which we usually do not know in advance, so an important part of determining the geometric stiffness matrix is in determining these in-plane stresses. Hence it is important to allow the degrees of freedom be such that this can occur.

The eigenvalue problem to be solved is

$$\Big[[K_E] - \lambda[K_G]\Big]\{\phi\} = 0$$

Both the elastic stiffness and geometric stiffness must be assembled before the stability analysis itself can be performed.

Truss Buckling

On the disk is a structure datafile called <<ex_fb.1>> for a simple two member truss structure.

```
ex_fb.1 Przemieniecki pp397             ::HEADER GROUP
21
1 1 1 1
end
2                                       ::ELEMENT GROUP
1 1 1 2 2
2 1 1 3 3
end
1                                       ::material #s
1 2 1
end
3                                       ::NODE GROUP
1  10.0  10.0   0.0
3   0.0   0.0   0.0
2  10.0   0.0   0.0
end
2                                       ::boundary conditions
2  0   0   0   0   0   0
3  0   0   0   0   0   0
end
1                                       ::applied loads
1  0.0 -1.0    0.0 0.0 0.0 0.0 0.0
end
1                                       ::material props
```

```
1  10e6  4e6  1.0   0.0e-4  0.0     1.0 1.0 1.0
end
 0                                   ::SPECIALs GROUP
end
```

The material is nominally aluminum and its properties are given in customary units.

Both the stiffness and geometric matrices must be assembled before the stability analysis itself can be performed. Do this by typing:

```
C> stadyn
   2
   ex_fb.1
   30
```

to be presented with the system matrices choices

```
FORM System Matrices: sub-menu
31: [K_E]            <static>
32:
33: [K_G]
34: [K_E],[K_G]      <buckling>
35: [M  ]
36: [K_E],[M  ]      <dynamic>
37: [K_E],[K_G],[M]  <all>
```

Choose

```
34
```

In response to the storage options

```
CHOOSE storage:
        0=DEFault BINary [prof]       >>StaDyn.GEO>>
        2=ASCii          [prof]       >>StaDyn.OUT>>
        3=ASCii          [NxB]        >>StaDyn.OUT>>
        4=BINary         [NxB]        >>named     >>
        5=DEFault + load              >>StaDyn.OUT>>
       11=BINary         [prof+locn]  >>named     >>
```

choose

```
0
```

Note that in order to establish the geometric matrix, it is first necessary for StaDyn to solve the corresponding static problem for the press loads. There are two main sources of pre-stress: that from $\{P\}$ specified in the SDF, and that from gravity. Both of these can be scaled separately, thus the required input

```
 @@    [K_G] = s_1 [K_G(P)] + s_2 [K_G(G)]'
 INPUT scales:  load | gravity '
```

Choose

```
1.0  0.0
0
```

Solving the statics problems is accomplished automatically and the material that is echoed to the screen is consistent with this. The required information for determining the stresses for use in the stability analysis is stored in the file `<<stadyn.dis>>`.

Back at the main menu, select the eigenanalysis

```
 90
```

StaDyn does two types of eigenvalue problems; vibration and stability, and has implemented two eigensolvers; Jacobi rotations and Sub-space iterations. The sub-menu choices are

```
EIGENvalue Problems: sub-menu
92:  Vibration w/ Jacobi
93:  Vibration w/ Sub-space
96:  Buckling  w/ Jacobi
97:  Buckling  w/ Sub-space
```

Jacobi rotations are very robust and give all the eigenvalues. Unfortunately, it is also computationally very expensive and generally should not be used for systems larger than 100. Sub-space iteration is appropriate for large problems but has difficulties with very small problems. The present problem is only of size $[2 \times 2]$ so Jacobi rotations is a good choice. Choose

```
  96
```

Each major capability of StaDyn is divided into two parts: the analysis and the post-processing. The first determines the nodal degrees of freedom while the second allows post-processing of them to give strains or contours and so on. Thus, in response to

```
 CHOOSE:   0=return   1=analysis  2=post_analysis
```

choose

```
  1
```

since this is the first time through. For large problems, a good deal of information would now be echoed to the screen; lines to watch are

```
D_NORM: #     Rotns = #
# of sweeps =
```

Finding eigenvalues is an iterative process and these can be used to monitor the rate of convergence. Note that StaDyn will automatically quit after 15 sweeps. For this problem we get normal convergence after only 1 sweep; about 7 sweeps is typical for larger problems. It should be realized that even if convergence has not occurred, the eigenvalues and eigenvectors may be of acceptable quality.

Finally, we are asked again

```
CHOOSE:   0=return   1=analysis  2=post_analysis
```

For this exercise, we are only interested in the first two buckling loads and this information is automatically stored in `<<stadyn.out>>` as part of the analysis, so we could opt to quit the program now. However, we will proceed to obtain the eigenvectors also. Choose

```
 2
```

This can also be re-entered starting from the beginning as long as the file `<<stadyn.snp>>` has not been altered. This would be used if the various mode shapes, for example, are to be accessed. The post-processing menu is

```
MODAL storage:
      0=return    |  0                <0 0 to end>
      # of modes  |  1=modal values
           :      |  2=mode  shapes
           :      |  3=modal vectors
                  |
      mode #      |  31=store contour data
```

The difference between mode shape and modal vector is that the former already has the boundary conditions factored in. This is a recurring prompt so, if desired, all the information can be stored. The last option would be used if the results are to be piped into a modal analysis post-processor. This option also wants the number of modes to be reported. It can be from one to the system size. If a number outside this range is given, then the program takes the appropriate allowable limit. For this exercise, we are only interested in the first two buckling loads, so select the first two modal values

```
2  2
```

and we are again given the output menu. Choose

```
0  0
```

and to quit the program, type

```
0
0
```

The output can now be surveyed by viewing the ASCII text file `<<stadyn.out>>`. The modal results are formatted similarly to a static analysis. Reference [19] shows that the first buckling load should be

$$P_{crit} = \frac{2\sqrt{2}-1}{7} EA = 0.2612\, EA$$

The output from StaDyn gives the critical loads as

```
0.261204E+07        -.676777E+18
```

which is very close to the theory. The very high second value indicates that there is no second buckling mode. It must be realized that this truss buckling is not the same as would occur if a frame member has pinned joints; this latter occurrence requires a bending action. It can be said, however, that for a given geometry the column or frame buckling is more likely to occur first.

Frame Buckling

As a second example, consider the simple frame described in the structure file `<<ex_fb.2>>`; this corresponds to an inverted L-frame fixed at its two ends.

```
ex_fb.2 Eisley pp273                    ::HEADER GROUP
22
1 1 1 1
end
6                                       ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
3 3 3 4 4
4 3 4 5 5
5 3 5 6 6
6 3 6 7 7
end
1                                       ::material #s
1 6 1
end
7                                       ::NODE GROUP
1  0.0  0.00  0.0
2  0.0  0.50  0.0
```

```
3  0.0  1.00  0.0
4  0.0  1.50  0.0
5  0.0  2.00  0.0
6  0.5  2.00  0.0
7  1.0  2.00  0.0
end
2                                       ::boundary condns
1  0   0   0   0   0   0
7  0   0   0   0   0   0
end
2                                       ::applied loads
3  1.00   0.0    0.0 0.0 0.0 0.0 0.0
5  0.0  -1.00   0.0 0.0 0.0 0.0 0.0
end
1                                       ::material props
1   68.95e9 26.52e9 576e-6 1e-4 1.0  50e-9 27.648e-9 27.648e-9
end
0                                       ::SPECIALs GROUP
end
```

This problem is stated in metric [SI] units with the material being aluminum. This is used as an example problem in Reference [13] but beware that the material was inadvertently stated as being steel.

Run the program as before with no echoing

```
C> stadyn
   2
   ex_fb.2
   30
   34
   0
   1 0
   0
   90
   96
   1
```

Since the system size is larger (15) than the last example, more Jacobi iterations are required. Store only the first three mode shapes

```
 2
 3  2
```

Now exit the program by typing

```
0  0
0
0
```

Scan through the output file `<<stadyn.out>>` to see the results. Reference [13] shows that the first eigenvalue and corresponding x displacements are:

$$
\begin{aligned}
P_{crit} &= 12078 \\
1: &\quad 0.0 \\
2: &\quad 0.4406 \\
3: &\quad 1.0 \\
4: &\quad 0.70235 \\
5: &\quad -0.00002 \\
6: &\quad -0.00001 \\
7: &\quad 0.0
\end{aligned}
$$

The buckling loads (eigenvalues) from the program are

```
12075.0      26152.2    50872.0
```

Again the complete physical interpretation of the modal vectors is given in a format identical to that of the static output. Picking off the values corresponding to the above gives

```
node         u          v            rotz
 1:         .000        .000          .000
 2:         .249     -2.05E-4       -.815
 3:         .565     -4.17E-4       -.223
 4:         .397     -7.13E-4        .790
 5:       -1.124E-5  -7.85E-4        .536
 6:       -5.622E-6   7.07E-2       -.138
 7:         .000        .000          .000
```

If these values are normalized with respect to the displacement at Node 3 then it is seen that they correspond with the above.

Experiment with reversing the sign of the loads and note that StaDyn still obtains the correct buckling load. This is important in complex structures where it is not obvious (from a cursory glance) which members are in compression.

Stability of Plates

We will finish the stability tutorials with an example of determining the stability of plates. The file `<<ex_pb.8x4>>` models a rectangular plate (aspect ratio 2:1) with clamped edges but free to expand laterally.

All stability analysis occurs in two stages, the pre-buckle analysis and the buckling analysis. The onset of buckling depends on the presence of the in-plane (membrane) stresses which we usually do not know in advance, so an important part of determining the geometric stiffness matrix is in determining these in-plane stresses. Hence it is important to allow the degrees of freedom be such that this can occur. Consequently, although the boundary conditions are referred to as being 'clamped', note that in fact they are allowed to move in the in-plane direction in such a way that only a σ_{xx} stress is generated. Note also that we set `IGLOBAL=32`.

Run StaDyn, read in the structure datafile and form the stiffness matrices

```
C>  stadyn
    2
    ex_pb.8x4
    30
    34
    0
    1  0
    0
```

Back at the main menu, select the eigenanalysis

```
    90
```

StaDyn does two types of eigenvalue problems; vibration and stability, and has implemented two eigensolvers; Jacobi rotations and Sub-space iterations. The sub-menu choices are

```
EIGENvalue Problems: sub-menu
92:  Vibration w/ Jacobi
93:  Vibration w/ Sub-space
96:  Buckling  w/ Jacobi
97:  Buckling  w/ Sub-space
```

Jacobi rotations are very robust, and give all the eigenvalues. Unfortunately, it is also computationally very expensive and generally should not be used for systems larger than 100. Sub-space iteration is appropriate for large problems but has difficulties with very small problems. Plate problems are large (especially when a convergence study must be done) so sub-space iterations is the appropriate choice. Choose

```
     97
```

Each major capability is divided into two parts: the analysis and the post-processing. The first determines the nodal degrees of freedom while the second allows post-processing of them to give strains or contours and so on. Thus, in response to

```
CHOOSE:   0=return   1=analysis  2=post_analysis
```

choose

```
        1
```

since this is the first time through. Unlike the Jacobi rotation solver, subspace iteration scheme need not solve for all the eigenvalues. Thus in response to

```
        INPUT:  # of modes of interest
        -->
```

choose

```
        10
```

This will compute the first (lowest) ten eigenvalues. Note that internally, StaDyn actually solves for about twice that number so as to ensure that the vector space is appropriately covered.

A good deal of information is now echoed to the screen, some of which may be recognized as coming from the Jacobi rotations. Note that StaDyn will automatically quit after 16 iterations (a number that can be changed in the `StaDyn.CFG` file). It should be realized that even if full convergence has not occurred, the eigenvalues and eigenvectors may be of acceptable quality, this can be judged by looking at the quality number.

Quit the program by typing

```
        0
        0
```

The buckling loads are stored in `StaDyn.OUT` as part of the analysis.

The output can now be surveyed by viewing the ASCII file `<<stadyn.out>>`. It contains

```
        EIGENvalues:
           N      lambda [load]
           1        28.4584
           2        28.5276
           3        40.3622
           :          :
```

Theory [23] gives the lowest buckling load for a rectangular plate with aspect ratio $a/b = 2$ as

$$\sigma_{cr} h = 7.88 \, \frac{\pi^2 D4}{a^2} = 28465.$$

The difference in the factor of 1000 is due to the fact that 1000 psi was the applied in-plane load. The comparison is reasonable considering the number of elements used.

4.2 Vibration Analysis

The vibration eigenvalue problem and the stability eigenvalue problem are very similar, although it can generally be said that convergence occurs sooner for the former than for the latter.

The eigenvalue problem to be solved is

$$\Big[[K_E] + \chi[K_G] - \lambda\lceil M \rfloor\Big]\{\phi\} = 0\,, \qquad \lambda \equiv \omega^2$$

Both the elastic stiffness and mass matrix must be assembled before the vibration analysis can be performed. The effects of pre-stress can be included by adding the geometric stiffness $[K_G]$.

Vibrating Beam

On the disk is a structure datafile called `<<ex_fv.1>>` for a three element cantilever beam:

```
ex_fv.1 Lalanne pp164.                 ::HEADER GROUP
12
1 1 1 1
end
3                                      ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
3 3 3 4 4
end
1                                      ::material #s
1 3  1
end
4                                      ::NODE GROUP
1  0.0 0.0  0.0
2 10.0 0.0  0.0
3 20.0 0.0  0.0
4 30.0 0.0  0.0
end
1                                      ::boundary condns
1  0   0   0   0   0   0
end
1                                      ::applied loads
2   0.0  0.0  0.0 0.0 0.0 0.0 0.0
end
1                                      ::material props
```

```
      1   10.0e6 4.0e6  1.68 2.5e-4 0.0   0.0 0.0 1.0
      end
       0                                  ::SPECIALs GROUP
      end
```

There is nothing special to note about this file except, perhaps, that while it is not necessary to have any applied loads in vibration analyses, it is nonetheless necessary to have some corresponding entries in the input data file. The material is nominally aluminum and its properties are given in customary units. The units for the density are

$$\rho = 0.25 \times 10^{-3} \quad \mathrm{lb} \cdot \mathrm{s/in}^4$$

This is different from the weight per unit volume by the gravitational constant. That is, $\rho = W^*/g$ where W^* is the weight per unit volume.

Both the stiffness and mass matrices must be assembled before the analysis itself can be performed. Do this by typing

```
   C> stadyn
      2
      ex_fv.1
      30
```

to be presented with the system matrices choices

```
      FORM System Matrices: sub-menu
      31: [K_E]            <static>
      32:
      33: [K_G]
      34: [K_E],[K_G]      <buckling>
      35: [M  ]
      36: [K_E],[M  ]      <dynamic>
      37: [K_E],[K_G],[M]  <all>
```

To do a dynamic analysis, we also need the mass matrix. Choose

```
      36
      3
      3
```

This will echo the stiffness and mass matrices into the <<stadyn.out>> file.

Vibration problems are eigenvalue problems, so choose

```
       90
```

StaDyn does two types of eigenvalue problems; vibration and stability, and has implemented two eigensolvers; Jacobi rotations and Sub-space iterations. The sub-menu choices are

```
EIGENvalue Problems: sub-menu
92:  Vibration w/ Jacobi
93:  Vibration w/ Sub-space
96:  Buckling  w/ Jacobi
97:  Buckling  w/ Sub-space
```

Jacobi rotations are very robust, and give all the eigenvalues. Unfortunately, it is also computationally very expensive and generally should not be used for systems larger than 100. Sub-space iterations, on the other hand, can solve the partial eigenvalue problem; that is, it can solve for the lowest m eigenvalues where $m \ll N$. Since these are the ones of most interest in structural analysis, it makes sub-space very attractive. It is the one that we will choose later for the plate problems, for now choose

```
92
```

Again, each major capability is divided into two parts: the analysis and the post-processing. The first determines the eigenvalues and nodal degrees of freedom while the second allows post-processing of them to give contours and so on. Thus, in response to

```
CHOOSE:   0=return   1=analysis  2=post_analysis
```

choose

```
 1
```

since this is the first time through. All vibration analyses can include the effects of pre-stress as a scaling on the geometric stiffness matrix. We will ignore these, hence in response to

```
INPUT pre-stress scale:   scale X [K_G}
```

type

```
0.0
```

The program now performs the complete analysis. A good deal of information is now echoed to the screen; an interesting quantity to keep your eye on is the `DIFF NORM`, finding eigenvalues is an iterative process and this quantity conveys an idea of the rate of convergence. It should be realized that even if convergence has not occurred, the eigenvalues and eigenvectors may be of acceptable quality. Thus the final echoing of information is an assessment of the quality of the eigenvectors.

Finally, we are asked

```
CHOOSE:   0=return   1=analysis  2=post_analysis
```

To access the eigenvector results, we must post process the binary files. Choose

```
        2
```

and you are given the storage menu

```
        MODAL storage:
              0=return     |  0               <0 0 to end>
              # of modes   |  1=modal values
                   :       |  2=mode shapes
                   :       |  3=modal vectors
                           |
              mode #       | 31=store contour data
```

The difference between the mode shapes and the modal vectors is that the former has the boundary conditions already factored in. This is a recurring prompt so, if desired, all the information can be stored. The last option would be used if the results are to be piped into a plotting analysis post-processor. This option also wants the mode number to be reported. It can be from one to the system size. For this exercise, we are interested in all the eigenvalues and eigenvectors, so choose

```
        10   2
```

The information is now stored in `<<stadyn.out>>` and we are given again the output menu. We could re-enter the post analysis either now (or re-enter later) because the complete information is also stored in binary form in the file `<<stadyn.snp>>`. This would be used if the various mode shapes, for example, are to be accessed. We will just quit

```
        0  0
        0
        0
```

Scan through the file `<<stadyn.out>>` to see the results. The first portion of it contains the stiffness matrix written in band storage form and the mass matrix written in diagonal (lumped) form. The resonant frequencies according to Reference [15] are

$$\omega_1 = \frac{3.516}{L^2}\sqrt{\frac{EI}{\rho A}} = 602.8\,\mathrm{r/s}\,, \qquad \omega_2 = \frac{22.11}{L^2}\sqrt{\frac{EI}{\rho A}} = 3790.7\,\mathrm{r/s}$$

and these compare to the program values of

```
    EIGENvalues:
    N    omega [r/s]    freq [Hz]
     1  570.785         90.8432       real
     2  3159.01         502.772       real
     3  7882.65         1254.56       real
     4  23081.9         3673.60       real
     5  30933.7         4923.24       real
     6  36772.0         5852.44       real
```

Clearly, the results are not fully converged. The complete physical interpretation of the modal vectors is given in the file in a format identical to that of the static output.

The reference also gives the modal vectors (in the format $\{v, \phi, \ldots\}$) as

$$\begin{aligned}\phi_1 &= \{1\,,\,0.18210\,,\,3.3040\,,\,0.2636\,,\,6.041\,,\,0.2772\}\\ \phi_2 &= \{1\,,\,0.09966\,,\,0.7179\,,\,-0.1672\,,\,-1.695\,,\,-0.2704\}\end{aligned}$$

On the face of it, these do not compare with the results from StaDyn. However, if these results are normalized with respect to the first entry, then both sets agree remarkably well. The difference lies in how the results are normalized, the one chosen in the program is based on the mass matrix.

A point worth mentioning in regards to dynamic problems is that after a structure input datafile has been created, it can be cumbersome to modify it to increase the number of elements (this would be required in order to improve the quality of the results). This is such a recurring need that the program `GenMesh` has a remeshing feature built in. The tutorial on transient responses demonstrates how this is used.

Frame Vibration

As a second example, consider the simple frame described in the structure file `<<ex_fv.2>>`

```
ex_fv.2 Lalanne pp172.                ::HEADER GROUP
22
1 1 1 1
end
2                                     ::ELEMENT GROUP
1 3 1 2 2
2 3 2 3 3
end
2                                     ::material #s
1 1 1
2 2 2
end
3                                     ::NODE GROUP
1  0.0    0.0   0.0
2  1.0    0.0   0.0
3  1.707 -0.707 0.0
end
2                                     ::boundary conditions
1  0   0   0   0   0   0
3  0   0   0   0   0   0
end
1                                     ::applied loads
```

```
2  10.0  0.0  0.0   0.0 0.0 0.0   0.0
end
2                                    ::material props
1    200.0e9 80e9  1.0e-2  7800  0.0  1.0 1.0 1.0e-6
2     70.0e9 25e9  1.0e-2  2700  0.0  1.0 1.0 1.0e-6
end
 0                                   ::SPECIALs GROUP
end
```

This problem is stated in metric [MKS] units and corresponds to an aluminum beam connected to a steel beam at a 45^o angle.

Make a driver (script) file using the `STRIP` utility on the `LOG` file of the previous tutorial and call it `<<invibn>>` say. Now change the name of the input file to `<<ex_fv.2>>` and run the problem as

```
C> stadyn < invibn
```

Scan through the output file `<<stadyn.out>>` to see the results. The stiffness matrix written in band storage form is the same as given in Reference [15], but the mass matrix written is different; the reason is that the reference uses the consistent mass matrix.

The reference also shows that the three eigenvalues and eigenvectors are

$$\begin{aligned} \omega_1 &= 1030 \qquad \phi_1 = \{-0.0010\,,\,-0.0087\,,\,1\} \\ \omega_2 &= 3131 \qquad \phi_2 = \{0.0609\,,\,0.3170\,,\,1\} \\ \omega_3 &= 8619 \qquad \phi_2 = \{-0.4794\,,\,0.1653\,,\,1\} \end{aligned}$$

The eigenvalues from the program are

```
1:  992.39
2:  2364.8
3:  6775.3
```

Again the complete physical interpretation of the modal vectors is given further on in the file in a format identical to that of the static output.

Spring Mass System

As a final example, we depart a little from the continuous systems considered above and look at a discrete spring-mass system. This is only to demonstrate that StaDyn can handle concentrated inertias. The structure data file is `<<ex_fv.3>>` and consists of three masses and three springs in a collinear arrangement.

```
ex_fv.3 Meirovitz pp175.                ::HEADER GROUP
11
1 1 1 1
end
3                                       ::ELEMENT GROUP
1 1 1 2 2
2 1 2 3 3
3 1 3 4 4
end
2                                       ::material #s
1 2 1
3 3 2
end
4                                       ::NODE GROUP
1  0.0 0.0 0.0
2 10.0 0.0 0.0
3 20.0 0.0 0.0
4 30.0 0.0 0.0
end
1                                       ::boundary conds
1  0   0   0   0   0   0
end
3                                       ::applied loads
2   0.0 0.0 0.0    0.0 0.0 0.0    1.0
3   0.0 0.0 0.0    0.0 0.0 0.0    1.0
4   0.0 0.0 0.0    0.0 0.0 0.0    2.0
end
2                                       ::material props
1    10.0e6 4e6  1.0  0.0 0.0    0.0  0.0  0.0
2    20.0e6 4e6  1.0  0.0 0.0    0.0  0.0  0.0
end
 0                                      ::SPECIALs GROUP
end
```

Note that the density is given as zero but concentrated masses are stated as the last entries on the applied loads line. The units are mass units, that is, weight/gravity. Further, since there are only axial degrees of freedom then `IGLOBAL=11`.

Run this as above using the script file and compare the results as given in Reference [18]

$$\omega_1 = 373\,, \qquad \phi_1 = \{1.0000\,,\, 1.8608\,,\, 2.1617\}$$
$$\omega_2 = 1321\,, \qquad \phi_2 = \{1.0000\,,\, 0.2542\,,\, -0.3407\}$$
$$\omega_3 = 2029\,, \qquad \phi_2 = \{1.0000\,,\, -2.1152\,,\, 0.679\}$$

The eigenvalues from the program are

```
1:    373.09
2:   1321.3
3:   2028.5
```

This reference further shows that the eigenvectors when normalized according to the scheme of StaDyn are

$$\phi_1 = \{0.2691\,,\, 0.5008\,,\, 0.5817\}$$
$$\phi_2 = \{0.8781\,,\, 0.2232\,,\, -0.2992\}$$
$$\phi_2 = \{0.3954\,,\, -0.8363\,,\, 0.2685\}$$

These are the same as reported by StaDyn.

Flexural Vibrations of Plates

In this tutorial we show the use of sub-space iteration as the means for solving the eigenvalue problem. This is the generally preferred scheme for large problems.

We will look at the vibrations of a circular plate as described in the datafile `<<ex_ps.2>>`. Run StaDyn, read the structure datafile and form the stiffness and mass matrices as before

```
C>  stadyn
    2
    ex_ps.2
    30
    36
    0
    0
```

Vibration problems are eigenvalue problems, so choose

```
    90
```

and off the sub-menu choices

```
    EIGENvalue Problems: sub-menu
    92:  Vibration w/ Jacobi
    93:  Vibration w/ Sub-space
    96:  Buckling  w/ Jacobi
    97:  Buckling  w/ Sub-space
```

choose

```
   93
```

Jacobi rotations are very robust, and give all the eigenvalues. Unfortunately, it is also computationally very expensive and generally should not be used for systems larger than 100. Sub-space iterations on the other hand can solve the partial eigenvalue problem; that is, it can solve for the lowest M eigenvalues where $M \ll N$. Since these are the ones of most interest in structural analysis, it makes sub-space very attractive.

Each major analysis capability is divided into two parts: the analysis and the post-processing. The first determines the nodal degrees of freedom while the second allows post-processing of them to give strains or contours and so on. Thus, in response to

```
CHOOSE:   0=return   1=analysis  2=post_analysis
```

choose

```
 1
```

since this is the first time through. All vibration analyses can include the effects of pre-stress as a scaling on the geometric stiffness matrix. We will ignore these, hence in response to

```
INPUT pre-stress scale:   scale X [K_G}
```

type

```
0.0
```

When using sub-space iterations, it is necessary to choose the number of modes of interest;

```
INPUT:   # of modes of interest
```

Internally, StaDyn will automatically increase the number by about ten to ensure that the vector subspace is adequately spanned. Choose

```
10
```

A good deal of information is now echoed to the screen; part of the reason for this is that Jacobi rotations are also performed as part of the sub-space iteration scheme. A number to keep your eye on is

```
TRIGGER rtolv:     #
```

This says which eigenvalue failed the convergence test. Generally, convergence occurs from the lowest upward, and this can be used to monitor the rate of convergence. When this number passes our selected number of modes, the iteration quits. If this does not happen, then StaDyn will automatically quit after 15 iterations (this number can be changed in the ¡¡stadyn.cfg¿¿ file). For this problem, we get normal convergence after about 12 iterations; this is typical. It should be realized that even if convergence has not occurred, the eigenvalues and eigenvectors may be of acceptable quality. Thus the final echoing of information is an assessment of the quality of the eigenvectors.

Finally, we are asked

```
    CHOOSE:   0=return   1=analysis  2=post_analysis
```

The eigenvalues are automatically stored as part of the analysis, but to access the eigenvectors we must post-process the binary files. Choose

```
    2
```

and you are given the storage menu

```
     MODAL storage:
           0=return     |  0               <0 0 to end>
           # of modes   |  1=modal values
                :       |  2=mode shapes
                :       |  3=modal vectors
                        |
           mode #       | 31=store contour data
```

For this exercise, we are only interested in the eigenvalues, but choose the mode shapes anyway

```
    10      2
```

The information is now stored in `<<stadyn.out>>` and we are given the storage menu again. We could ask for more data to be stored but for now we will just quit

```
    0   0
    0
    0
```

The output can now be surveyed by viewing the ASCII file `<<stadyn.out>>`. It contains

```
  EIGENvalues:
     N   omega [r/s]     freq [Hz]
     1   612.400         97.4665         real
     2   1261.50         200.773         real
     3   1261.50         200.773         real
     4   2037.19         324.229         real
     5   2037.19         324.229         real
     6   2322.68         369.665         real
```

```
   7   2934.76          467.081          real
   8   2934.76          467.081          real
   9   3482.00          554.178          real
  10   3482.00          554.178          real
```

Also have a look in `<<stadyn.log>>`, you will find much information about the eigen-solver performance.

Theory [11, 16, 21] gives that the resonant frequencies for a clamped circular plate are

$$\omega = \frac{\lambda_{rs}^2}{a^2}\sqrt{\frac{D}{\rho h}} = \lambda_{rs}^2\, 59.63$$

Some values for λ_{rs}^2 are

$$\lambda_{00}^2 = 10.216 \quad \lambda_{01}^2 = 21.26 \quad \lambda_{02}^2 = 34.88 \quad \lambda_{10}^2 = 39.77 \quad \lambda_{03}^2 = 51.04 \quad \lambda_{11}^2 = 60.82$$

This gives frequencies of

```
   1    609.2
   2   1268.
   3   1268.
   4   2080.
   5   2080.
   6   2371.
   7   3043.
   8   3043.
   9   3627.
  10   3627.
```

Again, we get a nice comparison. Except for the fundamental frequency, all the computed frequencies are lower than the corresponding theoretical values.

Note also that the algorithm had little difficulty in coping with the repeated roots. In fact, a look at the output file will show that the corresponding eigenvectors are different.

4.3 Forced Frequency Response

Allied to the idea of vibration is the concept of forced frequency response. Here the structure is excited at a frequency that is not necessarily the resonant frequency.

In comparison to a modal or vibration analysis, the significant difference here is that a force must be applied. Generally, this is known in the form of a spectrum (different amplitudes at different frequencies) but for now it is taken as uniform. The response obtained is referred to as the *impulse response.*

The problem to be solved is

$$\Big[[K_E] + \chi[K_G] + i\omega[\,C\,] - \omega^2\lceil\, M\,\rfloor\Big]\,\{\hat{u}\} = \chi_1\{\hat{P}\}$$

Both the elastic stiffness and mass matrix must be assembled before the analysis can be performed. The effects of pre-stress can be included by adding the geometric stiffness $[K_G]$.

Vibrating Rod

To demonstrate the concepts of forced frequency analysis, we will consider the forced vibration of a fixed-fixed rod. The data file is called `<<ex_ff.1>>`:

```
ex_ff.1 fixed-fixed rod            ::HEADER GROUP
11
1 1 1 1
end
8                                  ::ELEMENT GROUP
1 1 1 2 2
2 1 2 3 3
3 1 3 4 4
4 1 4 5 5
5 1 5 6 6
6 1 6 7 7
7 1 7 8 8
8 1 8 9 9
end
1                                  ::material #s
1 8  1
end
9                                  ::NODE GROUP
1   0.0 0.0  0.0
2  12.5 0.0  0.0
3  25.0 0.0  0.0
4  37.5 0.0  0.0
```

```
5  50.0 0.0  0.0
6  62.5 0.0  0.0
7  75.0 0.0  0.0
8  87.5 0.0  0.0
9 100.0 0.0  0.0
end
2                                   ::boundary condns
1  0   0   0   0   0   0
9  0   0   0   0   0   0
end
1                                   ::applied loads
5   1.0  0.0  0.0 0.0 0.0 0.0 0.0
end
1                                   ::material props
1    10.0e6 4.0e6  1.0  2.5e-4 0.0    0.0 0.0 0.0
end
 1                                  ::SPECIALs GROUP
2110   1.0e-2   0   0
end
```

Note that the rod has a small amount of mass proportional damping as specified through the `specials` code `[2110]`. Also note that the load is applied at the center of the rod.

Run the program, read in the structural datafile, and form the stiffness and mass matrices, by typing

```
C> stadyn
   2
   ex_ff.1
   30
   36
   0
   0
```

Choose the forced frequency analysis

```
70
```

which leads to the menu choices

```
FORCED Frequency Problems: sub-menu
71:  Distributed loadings {P}
72:  Pre-stress           {dP}
```

In the first of these the loads specified in the input file are made oscillatory, whereas in the second they form the pre-stress and the oscillatory load is a point load separately specified. We will choose the distributed loading analysis

```
71
1
```

Although the loads are specified in the structure datafile, sometimes it is useful to be able to specify a scale on the distribution; in response to

```
INPUT load scale:   scale X {P}
```

type

```
1.0
```

Now input the frequency range information

```
INPUT:  freq_0 (Hz)  |  d_freq (Hz)  |  # of incs
```

It is a practice of StaDyn that a vertical bar | separates multiple fields that must be entered. So this wants three numbers: the first being the beginning of the frequency range, then the spacing of the frequencies, and ending with the total number of frequencies to be tested. Note that these frequencies are to be given in Hz. For now choose

```
0.0    10.0    601
```

This gives 601 samples over a frequency range of 6000 Hz. Be careful when looking at freely supported structures, that the initial frequency is not specified as identically zero, since then the system is singular. The next information required is the form of the output. The choices are

```
CHOOSE output:
              0=return
              1=Re-Im nodal  displacements
              2=Mag   nodal  displacements
              3=norm of all  displacement
```

An interesting thing about the normalized displacements is that in the vicinity of a resonance they approach the eigenvectors. For now choose

```
3
```

since the norm gives a global picture of the dynamic behavior. Exit the program by typing

```
0
0
```

Look through the file `<<stadyn.dyn>>` (not `<<stadyn.out>>`) and see three unlabeled columns in the form

```
freq (Hz)     displacement norm     log( disp norm)
```

Notice where the maxima occur — this is in the vicinity of a resonance. A more precise value can be determined by re-running the example but having narrowed the frequency down to the band of interest. There is no header in the file because it is intended to be displayed by a program such as DiSPtool.

For the above parameters, the resonances are expected at [10]

$$f = \frac{\pi N}{L}\sqrt{\frac{EA}{\rho A}} = 2\pi N \text{ k-rad/s} = N \text{ kHz}$$

where N is the resonance number. Thus, counting and identifying the resonances for this problem will be easy.

Now scan `<<stadyn.dyn>>` and see that the peaks appear close to

```
1000     3000     4000      5000
```

It is obviously missing some resonances. In fact, it is picking up every other one. Now change the location of the force to Node 3 and re-run the program. (This is where using the script file can save a lot of time.) When this is done it will be seen that the peaks occur at

```
1000    2000    3000     5000     6000    7000
```

Again, it is missing some peaks. Experiment with other locations. What this is showing is that where the force is applied will determine what modes play a role in the response. That is, not all modes may be significant for a given type of loading. Try putting a force at every node; contrary to expectation not all modes are excited! Now try alternating the directions of the forces.

4.4 Transient Response

These tutorials show how StaDyn can be used to compute the forced transient response of a structure. The problem to be solved is

$$\Big[[K_E] + \chi[K_G]\Big]\{u\} + [\ C\ ]\{\dot{u}\} + \lceil M \rfloor\{\ddot{u}\} = \chi_1\{P(t)\} + \chi_2\{G(t)\}$$

Both the elastic stiffness and mass matrix must be assembled before the analysis can be performed. The effects of pre-stress can be included by adding the geometric stiffness $[K_G]$. Note that gravity can be applied with a specified history.

Two sets of files need to be ready. The first is of the structure itself and the other is of the input force and gravity histories. On the disk is a file `<<force.t1>>` which is a five point representation of a triangular pulse and we will use this as the forcing history. A separate file could be used for the gravity history, but we will use an additional column is the force file. Before we proceed, however, have a look at the `<<force.t1>>` file — it should have the following data

```
0         0.0   0.0
10e-6     0.0   1.0
60e-6     1.0   1.0
110e-6    0.0   1.0
500e-6    0.0   1.0
```

The first column is time in seconds, while the second column is load. The third column is the gravity history although it will not be used in this example. Note that there are only five data points, thus depending on the incremental rate chosen, this file will be interpolated as needed. This makes it convenient for inputting a variety of force histories — they can even be experimentally measured, for example, with many points.

The second file `<<ex_ft.1>>` is the structure datafile. A transient loading problem usually requires significantly more elements for analysis than the corresponding static or vibration problem. Consequently, a need often arises where, for a given structure, the density of elements needs to be increased. We first demonstrate this with a rod.

Remeshing a Frame

On the disk is a structure file called `<<ex_ft.1>>` for a two noded free-free rod.

```
ex_ft.1 free-free rod                  ::HEADER GROUP
11
0 0 0 0
end
1                                      ::ELEMENT GROUP
1 1 1 2 2
```

```
end
1                                      ::material #s
1 1  1
end
2                                      ::NODE GROUP
1   0.0 0.0 0.0
2  50.0 0.0 0.0
end
0                                      ::boundary conditions
end
1                                      ::applied loads
1  1.0   0.0     0.0 0.0 0.0 0.0 0.0
end
1                                      ::material props
1  10.0e6 4.0e6 1.0   2.5e-4 0.0   1.0 1.0 1.0
end
1                                      ::SPECIALs GROUP
2110  0.0 0.0 0.0
end
```

There are no boundary conditions for this problem but we must still include the section that specifies boundaries; hence the null group for the boundary conditions. Note that while the force history itself will be specified elsewhere, this file must say where it is applied. Thus, the entry 1.0 above could be used as a scaling factor. Indeed, multiple forces could be inputted and they will all act as scalings on the applied load history. In this way, distributed and vector loads can be applied. Keep in mind, however, that each of these forces will have the same time history. Note that the rod has no damping but a small amount of mass and/or stiffness proportional damping could be specified through the `specials` code `[2110]`.

It is obvious that it is not possible to perform wave propagation studies with only a single element. Hence the purpose of this tutorial is to show how the program `GenMesh` can be used to remesh a structure to the required number of elements.

The interface of `GenMesh` is very similar to that of `StaDyn`, so run it as

```
C>  genmesh
```

We wish to remesh the inputted structure, so choose

```
6
```

and you are asked for the new name and old name, respond

```
2
ex_ft.2
ex_ft.1
```

which leads to the re-mesh menu:

```
CHOOSE:   0=return
          1=
          2=remesh plate
          3=remesh frame
          4=add frame member
          5=
          6=reduce bandwidth
          :
```

Choose to remesh a frame

```
3
```

The program wants to know how the structure is to be remeshed.

```
MULT factors:
INPUT:  elm1  |   elm2  |   mult factor    <0 0 0 to end>
```

It is looking for a range of elements and their corresponding multiplication factor. It is possible to overlap these ranges and thus save some typing. We have only one element so type

```
1   1   100
0   0   0
```

This has divided the original element into 100. Exit the program by typing

```
0
0
```

Look through the file `<<ex_ft.2>>`. It is seen that indeed it has made 100 elements; however, notice that the last element has connectivity

```
100  1  101 2 2
```

While the program has remeshed the structure, it has not numbered the nodes in any optimum fashion — the original numbering is left intact. This can be useful sometimes, but generally, it means that the bandwidth has become unacceptably large and the nodes must be renumbered.

We will now repeat the procedure from the beginning and this time also renumber all the nodes.

```
   C> genmesh
      6
      2
      ex_ft.2
      ex_ft.1
      3
      1  1  100
      0  0  0
```

We now want to renumber the nodes.

```
      6
      1
```

Notice that the initial bandwidth is 100. We are using the wavefront method which basically renumbers based on the proximity of nodes to each other. This is determined relative to the focus point, so in response to

```
       INPUT wavefront center:    x0  |  y0  |  z0
```

respond

```
       -100  0  0
```

Note that the bandwidth is reduced to 2. Also, keep in mind that the renumbering is based on nodal number (not on the number of degrees of freedom of the reduced system) and so the bandwidth numbers may appear different from that obtained when using StaDyn.

Since the file is acceptable to us, update it and exit

```
       2
       0
       0
       0
```

Now look at `<<ex_ft.2>>` to see that the connectivity of the last element is

```
       100  1  100 101 101
```

This is what we wanted.

Impact of a Rod

Run StaDyn, read in the structure datafile, and form the stiffness and mass matrices

```
C>  stadyn
    2
    ex_ft.2
    30
    36
    0
    0
```

With both the stiffness and mass matrices formed, the transient analysis can be chosen

```
80
```

This leads to the sub-menu

```
TRANSient Problems: submenu
81:  Distributed loadings  {P}+{G}
82:  Multiple       ..
82:  Pre-stress     ..      {dP}
84:  Moving         ..
86:  All-node       ..
```

We wish to use the loads specified in the structure datafile as the distribution of transient applied loads, hence choose

```
81
```

There are two stages to the transient analysis: a first time analysis when all the basic data is generated, and a post-processing stage when the snapshots are analyzed. (Snapshots are complete records of the nodal displacements at a particular instant, they are needed to obtain the member loads and element stresses.) Right now we just want to solve the problem, so in response to

```
CHOOSE:  0=return  1=analysis  2=post_analysis
```

choose

```
1
```

Sometimes it is convenient to have unit loads in the structure datafile or a load history that is normalized somehow and to use the analysis stage to scale the loads. In response to

```
INPUT scales:  load  |  gravity
```

type

```
        1000    0.0
```

At this stage we must input information about the time integration parameters

```
      TYPE:  time inc   |  # of incs  |   print count  |  snap count
```

The equations of motion are integrated incrementally in time. The program wants to know the time increment and how many of them. There is no hard and fast rule for this especially since the algorithm implemented is unconditionally stable; it is best to experiment. Since the time step may be finer than what is necessary for display of the results, the print count allows the output rate to be varied. Snapshots are necessary for the post-processing of the results to obtain stresses and strains; this is stored in binary form. For now we will not bother with snapshots so choose

```
        1e-6   501      2   2
```

For dynamic problems, it is not feasible to output the results for every node because there is just too many of them and 99.9% of the information would probably not be used. So StaDyn allows selection of a limited number of nodes as output

```
      CHOOSE nodal output:
      TYPE:      node #  |  DoF  |  rate (2=accn)    < 0 0 0 to end>
```

Since each node can have up to six degrees of freedom (3 displacements and 3 rotations) and up to 3 rates (displacement, velocity, and acceleration), it is easy to see how the amount of information can mushroom. For now just select the velocity of Nodes 1, 51 and 101 as output. Velocities are usually good quantities to choose because they are related more directly to the loads.

```
        1     1      1
        51    1      1
        101   1      1
        0     0      0
```

This is a recurring menu allowing the nodes to be chosen randomly. Now the snapshot information

```
      CHOOSE snapshot output:  0=none
                               1=partial  disp
                               2=full     disp
                               4=Full     disp, vel, acc
```

For now choose

```
        0
```

The final piece of information required is about the force history

```
    @@ get  {P} load history
    @@ TYPE:  Load_Filename -->
```

Respond

```
        force.t1
```

The file is then parsed for the appropriate columns

```
     @@ INPUT:  Time col  |  load col  |  # cols
```

Respond

```
        1 2 3
```

Similar input is required for gravity

```
    @@ get  {Grav} load history
    @@ TYPE:  Load_Filename -->
```

Give the responses

```
        force.t1
        1 3 3
```

The program now proceeds on its way, periodically echoing some information to the screen. Finally, back at the menu, exit by typing

```
        0
        0
```

The results are located in the file `<<stadyn.dyn>>` and are stored as

```
        time   P_load  G_load    resp 1     resp 2  ....
```

and in the present case it will have five columns. Note that the force is the linearly interpolated force. The times traces of these responses are shown in Figure 4.1. These numbers are best viewed using the utility program DiSPtool.

From the theory for the impact of rods [9], it can be established that the relationship between force and velocity is

$$\dot{u} = -\frac{Pc_o}{EA} = 0.02P$$

where $c_o = \sqrt{EA/\rho A} = 200000$ in/s is the wave speed. For the current maximum load this gives a maximum velocity of 20 in/s. This is close to that given by the program as 19.939 which occurs at 60 μs.

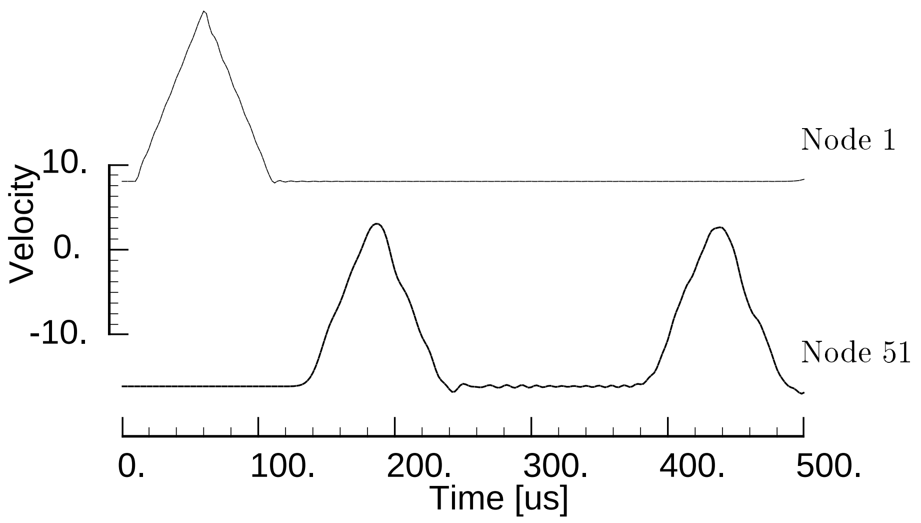


Figure 4.1: Time traces for impacted rod.

Now look at the behavior of the fourth column which corresponds to Node 51 that is half way along the rod at a distance of 20 in. Notice that the maximum value has decreased to 19.039 in/s and its time of occurrence is at 186 μs. The further the wave propagates the more filtering the elements do but the propagation speed of 1 in in 5 μs is hardly affected. Looking at the reflection at time = 438 μs, the peak has further decreased to 18.629.

To understand what is happening, it is necessary to realize that the input force history with its sharp edges is rich in high frequencies. These high frequencies are filtered because the mesh density (element size) is not fully converged. To improve the results it is necessary to either use smaller elements (so that they pass a greater range of frequencies) or smooth the edges on the input force so that it is within the frequency range of the given element size.

As a good exercise, repeat all the above starting with `<<ex_ft.1>>` but convert it into 200 elements. This will give a feel for the proper element size and the sort of performance to be expected from StaDyn.

Post-Processing the Snapshots

Keep in mind that the basic set of unknowns in StaDyn are the nodal degrees of freedom and their time derivatives. Thus, during the dynamic event these are the easiest to record and store. Member information such as stresses and strains must be obtained as a post-processing operation on the displacements. Consequently, if at all possible, it is advisable to keep them out of the transient loop. This is why the concept of the snapshot is introduced.

Run StaDyn as above until the information about the snapshot is required.

```
C>   stadyn
```

```
     2
     ex_ft.2
     30
     36
     0
     0
     80
     81
     1
     1000    0.0
     1e-6   501      2   2
     1    1      1
    51    1      1
   101    1      1
     0    0      0

    CHOOSE snapshot: 0=none  1=partial   2=full
```

Now select a partial snapshot

```
     1
```

Since it is only a partial snapshot, the program wants to know what nodes are involved

```
    TYPE:    node #       < 0 to end>
```

This is a recurring prompt, therefore many unconnected nodes can be chosen. Keep in mind, however, that determining member values requires knowledge of all nodes associated with an element; StaDyn insures that the appropriate nodes are included for the element of interest. In the present case we will only look at one node near the center of the rod, so choose

```
     51
     0
```

The rest of the responses are the same as before

```
     force.t1
     1  2  3
     force.t1
     1  3  3
     0
     0
```

The output for the partial snapshot is in `<<stadyn.out>>`. It is in ASCII text form, so look at it. The data is stored as node number followed by the three displacements and three rotations. The beginning of each snapshot contains the time and the number of elements stored. Note that there is duplication of the nodes; this occurs because the information stored is based on the triangular elements. Thus each group of three lines represent the element completely. Parenthetically, the record for the full snapshot would be stored in `<<stadyn.snp>>`; it is in BINARY form, so you cannot look at it without re-running StaDyn.

This snapshot information can now be converted to member loads and so on by re-running StaDyn in post-analysis mode. That is,

```
    C> stadyn
       2
       ex_ft.2
       80
       81
```

Now choose the post-analysis for partial snap shots

```
       2
       1
```

We have a choice of the type of output

```
       OUTPUT:
              0=return
              1=displacement
              2=nodal strain [ue]
              3=nodal stress
              4=crack
              5=element stress
              7=force sum
              <<plate>>
             21=displacement
             22=nodal strain [ue]
             23=nodal stress
              <<frame>>
             25=displacement
             26=nodal strain [ue]
             27=nodal stress [F/A M/I]
```

Choose the nodal stresses

```
       27
```

The final piece of information we give is the particular node of interest

```
        INPUT:  node #

        51
```

The program will search through the snapshots to locate the nodal data, it will then check to insure that they belong to the same member. Exit the program by typing

```
        0
        0
```

The member stresses are stored in the file `<<stadyn.dyn>>` in the form of seven columns

```
        time      F_x/A  F_y/A  F_z/A   M_x/I_x  M_y/I_y  M_z/I_z
```

The first is time, the next three are stresses, and the last three are bending like stresses (the actual stress is obtained by multiplying by the appropriate distance from the neutral axis). The maximum stress is -950.20 and it occurs at 186 μs. The complete history is best viewed using the utility program DiSPtool.

Wave propagation is a complicated subject and it puts great demand on the program and the user. The above tutorial was to show the basic functioning of StaDyn and does not indicate its full potential. It is recommended to consult References [9, 14, 20] for more examples that can be tested against StaDyn.

In-Plane Impact of a Long Plate

In this tutorial, we look at wave propagation in a 2-D in-plane plate. The plate resembles, somewhat, a beam but because it is modeled two-dimensionally it will exhibit the evolution of waveguide types of behaviors. Only one half of the plate is modeled since there is a line of symmetry at the left face.

Run StaDyn, read in the structure datafile, form the stiffness and mass matrices.

```
    C>  stadyn
        2
        ex_pt.1
        30
        36
        0
        0
```

With both the stiffness and mass matrices formed, the transient analysis option can be chosen from the main menu.

```
 80
 81
 1
 1000    0.0
```

Now input the parameters for the time integration and the nodal outputs

```
 1e-6   401      1     10
 1    2   1
73    2   1
197   2   1
 0    0   0
```

This will store the velocities at every 1 μs and the snapshots at every 10 μs. We will demonstrate the use of snapshots for obtaining contour plots of the stresses. So in response to

```
CHOOSE snapshot output:   0=none  1=partial  2=full
```

choose full snapshots

```
2
```

The last set of information required concerns the force and gravity histories

```
force.t1
1 2 3
force.t1
1 3 3
0
0
```

The program figures out how much data is in the force file and interpolates as needed based on the time increment.

The nodal velocity results are located in the file `<<stadyn.dyn>>` and are stored as

```
Time  P_load  G_load   resp1  resp2  resp3   ....
```

By scanning this file, some of the maximum velocities occur at

```
 :
 60.00e-06  1000   0    304.26     245.68      229,13
 :
260.00E-06     0   0    366.25     352.27      336.19
 :
400.00E-06     0   0   -154.91    -160.42     -156.36
 :
```

It is interesting how, over time, all monitored nodes tend to the same behavior.

As a good exercise, repeat the above but use more modules in the modeling, and try different step sizes. This will give a feel for the proper element size and the sort of performance to be expected from StaDyn.

We will now demonstrate how the snapshots can be post-processed to obtain contours of stress. Recall that we already have stored the full snapshots; to access this information type

```
  c>   stadyn
       2
       ex_pt.1
       80
       81
       2
```

until the information about the snapshot is required

```
      CHOOSE snapshot: 0=none
                       1=partial   disp
                       2=full      disp
                       4=full      disp, vel, acc
```

We stored the full snapshots, hence

```
       2
```

Now we will retrieve a particular snapshot

```
       CHOOSE snapshot:
              0=return
              #=which snap
             -1=all   snaps at one node
             -2=all   snaps at all nodes
             -#=rate at #
```

The -1 option allows a single node to be monitored across all snapshots, this is used for generating time traces. Here we will look at a single snapshot. Choose the eleventh snapshot (i.e., time at 100 μs)

```
       11
```

and after you see the counter you are launched into the Post menu

```
     POST menu:
              0=return
                   LOCAL (member coords)
```

```
             2=nodal strain [ue]
             3=nodal stress
             4=nodal force
             6=element stress
             7=force sum
             8=specials
                  GLOBAL
            11=Global displacement
            12=Global loads
            14=Global assembled DoF loads
            15=Global assembled nodal loads
                  CONTOURs
            31=store displacement data
            32=store strain       data
            33=store stress       data
                         frames
            34=store displacement data
            35=store strain       data
            36=store stress       data
           100=      << select FRAMEs    only   >>
           200=      << select F_PLATEs  only   >>
```

This is almost identical to the Post menu for static problems. We want to see the stresses, so select

```
       33
```

Quit this menu and go to the PostScript output off the main menu.

```
       0
       0
       700
       2
       p1.ps
```

At this sage we are given the contours menu

```
       CONTOURs:
              0:  Quit
                    <<RENDER PS file>>
              4:  Deformed shape
              5:  Contours
                     <<CHANGE defaults>>
             11:  Position model
             12:  Rotate model
```

```
              13:  Move tagged surfaces
              14:  Add caption
              15:  Toggle tags
              16:  Change pens
```

The menu is divided into two parts: the top renders the PS file, while the bottom allows changing the defaults for the drawing parameters. We will change the pen thicknesses

```
       16
```

and the choices are

```
       INPUT pens:   mesh  |  contour
       current:      40.00    80.00
       -->
```

Make the mesh lines thinner

```
       15    80
```

Now choose to render the contours

```
       5
```

to be given the DoF menu

```
      DoF:       1=u   2=v   3=w   4=Rx  5=Ry  6=Rz
            or   1=Exx 2=Eyy 3=Exy 4=Kxx 5=Kyy 6=Kxy
            or   1=Sxx 2=Syy 3=Sxy 4=Mxx 5=Myy 6=Mxy
   Mesh:         1=Yes 0=No -1=Black
   Legend:       1=Yes 0=No -1=Black

   INPUT:  DoF  |  scale  |  mesh ? | legend ?
```

Of interest now are the x-direction stresses since they relate to the bending stresses in beams. Choose this component and quit

```
       1   1.0  1  1
       0
       0
```

Before we look at the contours, it is worth pointing out that to access the other components of stress for this snapshot it is only necessary to enter the PostScript files menu option. If, however, a different snapshot is desired then all of the above procedure needs to be redone.

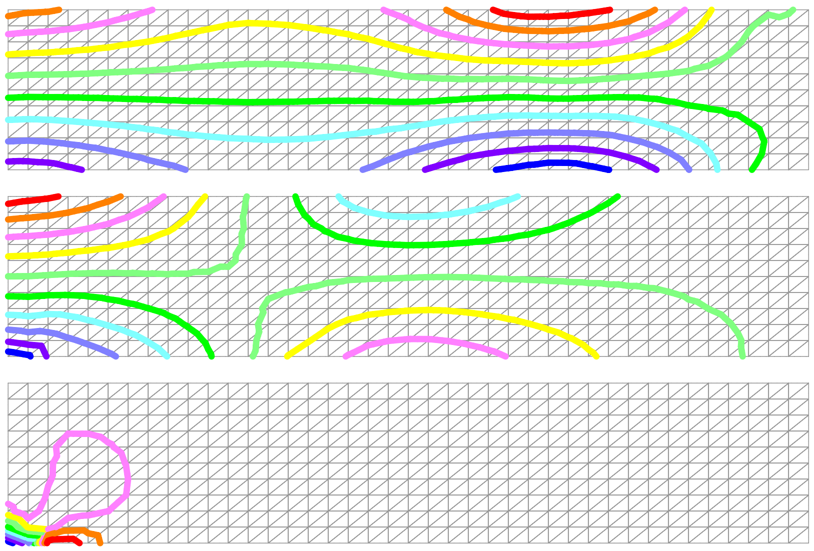


Figure 4.2: Contours of bending stress σ_{xx} at three different times.

Look at the output in `<<p1.ps>>`, this is the PostScript file of the contours. This can be sent directly to a PostScript printer, or can be viewed using the GhostScript program. The file will produce color contours as is shown in Figure 4.2. These contours clearly show the alternating stress sign that is typical in flexural wave propagation.

Make a script file for the postprocessing and look at the stresses. As shown in Figure 4.2 the stress distribution is highly non-uniform; hence also look at the shear stress σ_{xy} and note that its peaks correspond to zeros in the bending action.

The configuration for the contour plots is stored in the file `S<<stadyn.ctr>>`. This has the organization

```
geometry::
X Y position
mesh, contour  pen thickness
rotations X Y Z
label positions X Y and size
modes::
sub-structure flags
```

Sometimes it may be more convenient to make the adjustment through this file rather than through the program.

Chapter 5

Nonlinear Analyses with NonStaD and Simplex

NonStaD (NONlinear STAtic and Dynamic analysis) is designed to perform nonlinear static and dynamic analyses of thin-walled structures whereas Simplex does the same for 3-D solids. This chapter presents a few tutorials that cover the basics of these nonlinear analyses. Note that both the computational cost and computer resources increases substantially for these types of problems.

The main nonlinear capabilities of NonStaD/Simplex involve determining the member displacements and loads, and structural reactions for:

- Static incremental analysis
- Dynamic implicit incremental analysis
- Dynamic explicit incremental analysis

Simplex has the additional capability of nonlinear constitutive relations in the form of plasticity and rubber elasticity. The programs are menu driven in much the same way as for StaDyn; indeed as they proceed, they create a number of files which share the same names (and internal format) as for StaDyn. The primary files are:

```
nonstad.cfg      stadyn.stf      stadyn.dis
nonstad.log      stadyn.mas      stadyn.snp
 stadyn.out      stadyn.mat
 stadyn.dyn      stadyn.geo
 stadyn.mon      stadyn.lod
```

The `.LOG` file echoes all the input responses as well as having some extra information that might prove useful during post analysis of the results. The `.DYN` and `.OUT` files are the usual locations for NonStaD/Simplex output; there is the additional output file `<<stadyn.mon>` that contains monitored information (such as vibration eigenvalues) that are very useful in nonlinear analyses. The second column of files are associated with various system matrices such as the stiffness matrix and the mass matrix — these are left on disk in case they may be of value for some other purpose. They are in binary form. The last column of files are output files. They too are in binary form for compactness but can be read for further post-processing.

Regularly have a look in the `<<stadyn.log>>` because it contains lots of additional information about the functioning of NonStaD and Simplex.

5.1 Large Deflection Analysis of a Beam

This tutorial considers the large deflection of a cantilevered beam subjected to a transverse load. Elementary linear beam theory would indicate only a transverse deflection but such a situation would generate enormous axial forces in the beam. The nonlinear analysis accounts for these axial loads correctly in an incremental/iterative manner.

Getting Started with NonStaD

NonStaD is designed to run as a console program under `MS Windows`. Note that all instructions are case insensitive; we will vary the case only to help make the instructions clearer.

To run the program, type (at the 'C prompt')

```
C>  nonstad
```

(note the name of the executable.) to be given the opening menu

```
MAIN menu:
        0: Quit
        2: Read in structure DataFile
           <<Elastic>>
       13: Static Incremental Analysis     (full NR)
       14: Incremental Time Analysis       (imp,NR)
      142:    ..        ..  Analysis       (imp,NR)
       18: Incremental CoRot Wave Analysis (exp)
      182:    ..        ..   Wave Analysis (exp)
            << Cohesive >>
       26: Incremental CoRot cohesive  (exp)
       27: Incremental CoRot cohesive  (exp,init)
           <<Elastic/Plastic>>
      342: Incremental Time Mult_loads     (imp,NR)
       38: Incremental CoRot Wave Analysis (exp)
      382:    ..        ..   Mult_loads    (exp)
            << Extras >>
      700: PostScript plot files
      800: System services
      911: Ikayex information
```

The biggest difference in comparison to the linear analyses of StaDyn is the absence of the option to form the system matrices; in nonlinear analyses, the system matrices change during the analyses and are therefore part of the particular analyses. Quit by typing

```
      0
```

There are two files worth looking at. The first is the log file `<<nonstad.log>>`; use the `list` utility to peruse it. It contains

```
 @@ NonStaD version 3.10, August 2001
 @@ DATE:  8- 1-2001      TIME: 11:11
 @@ MAXimum storage   :      500000
 @@ MAXimum elements  :        3000
 @@ MAXimum nodes     :        3000
 @@ MAXimum force incs:        3000
 @@ ITERmax           :          16
 @@ rtol              :    1.000000E-10
 @@ ALLOCATION succeeded
          0 ::MAIN
 @@ NonStaD OK, ended from MAIN
```

The second is `<<nonstad.cfg>>`. This is the configuration file and has in it

```
      500000     ::MaxStorage
        3000     ::MaxNode
        3000     ::MaxElem
       10000     ::MaxForce
           1      ::ilump
          16      ::itermax
          1.0000000E-010      ::rtol
 @@ DATE:  8- 1-2001      TIME: 11:11
```

This allows the run time setting of the dimensions of the arrays. If a memory allocation failure occurs, this is the place to make the adjustments.

Incremental/Iterative Solution

The element formulation for the linear static analysis of a beam is exact irrespective of the element size. This is not true for a nonlinear analysis. The one element structure datafile for our beam is in the file `<<ex_fn.1>>` and contains

```
     ex_fn.1:  doyle pp219              ::HEADER GROUP
     22
     1   1   1   1
     end
     1                                  ::ELEMENT GROUP
     1     3     1   2   2
     end
```

```
1                                  ::matl distn
1    1    1
end
2                                  ::NODE GROUP
1   .0000000      0.000000      .0000000
2 10.0000000     0.0000000      .0000000
end
1                                  ::boundary condns
1    0  0  0    0  0  0
end
1                                  ::applied loads
2   0.0  1.0  0.0   0.0  0.0  0.0   0.0
end
1                                  ::matl props
1   10e6  4e6  0.1  2.5e-4  0   16.66e-5  16.66e-5  8.33e-5
end
0                                  ::SPECIALs GROUP
end
```

It is cantilevered at one end and has a transverse load at the other. The `Remesh` capability of `GenMesh` was used to convert this into a model with ten elements which is called `<<ex_fn.2>>`.

NonStaD's nonlinear solvers use a combination of multiple load increments plus Newton-Raphson equilibrium iterations at each increment. This tutorial shows a simple example.

Run NonStad and read the structure datafile

```
C>  nonstad
    2
    ex_fn.2
```

The choices for implicit nonlinear analyses are

```
        13: Static Incremental Analysis  (full NR)
        14: Incremental Time Analysis    (imp,NR)
       142:    ..        ..  Mult_loads  (imp,NR)
```

The second of these uses a pseudo-time for static problems. We will choose the first option to do an analysis

```
    13
    1
```

The information required is the load level plus the number of increments.

```
    TYPE: force max | # of incs | max # inc  | follower (0=N,1=Y)
```

We will load to 100 using five increments. If NonStaD has difficulty converging it will divide the load increment; the maximum number of increments is set to prevent an infinite loop. Finally, unlike a linear analysis, the load in a large deflection problem can either keep its original direction or follow the deforming shape of the structure. We will keep the load pointing in the same direction.

```
        100   10   20   0
```

The algorithm has a number of parameters to control the radius of convergence; two of the parameters are

```
          @@      u = u + beta*du   K = K_E + gamma*K_G
          INPUT:   beta   |  gamma  '
```

Input

```
        1.0   1.0
```

The Newton-Raphson iterations use a tolerance criterion for convergence, however to prevent locking it is good to set a reasonable upper limit on the maximum number of iterations. In response to

```
    INPUT:   iter max  |  tolerance (<1.0e-5)
```

input

```
        50    1e-6
```

Finally, we need to choose the nodal outputs.

```
        CHOOSE nodal output:
        TYPE:  node #  |  DoF               <0 0 to end>
```

We will monitor the end (load) point

```
        11  1
        11  2
        11  6
         0  0
```

Quit the program

```
        0
        0
```

The results are in the file <<stadyn.dyn>>. The final line gives

```
    100.000  -5.92907   8.29978   1.46854   5.00000
```

This is a large deflection. The last column in the output is the number of iterations at that load step. In general, as the increment is increased so too do the number of iterations. It is difficult to know in advance the right balance between load increments versus number of iterations.

Pseudo-Time Incremental Analysis

Because nonlinear static problems require a series of load steps, then we can think of these increments as occurring over time. We will repeat the previous problem by applying a load history from the force file `<<force.14>>` with the following data

```
0       0
10    100
20    100
```

The first column is time in seconds, while the second column is load. The time scale is so slow that inertia effects are negligible. Note that there are only three data points, thus depending on the incremental rate chosen, this file will be interpolated as needed. This makes it convenient for inputting a variety of forces — they can even be experimentally measured, for example, with many points.

Run NonStaD as before

```
C>  nonstad
    2
    ex_fn.2
```

The transient implicit option is chosen

```
    14
```

There are two stages to the transient analysis: a first time analysis when all the basic data is generated, and a post-processing stage when the snapshots are analyzed. (Snapshots are complete records of the nodal displacements at a particular instant, they are needed to obtain the member loads.) Right now we just want to solve the problem, so in response to

```
    CHOOSE:  0=return  1=analysis  2=post-analysis
```

choose

```
    1
```

You are now asked for

```
   TYPE:  time inc  |  # of incs  |   print count  |  snap count
```

The equations of motion are integrated incrementally in time. The program wants to know the time increment and how many of them. There is no hard and fast rule for this especially since the algorithm implemented is unconditionally stable; it is best to experiment. Since the time step may be finer than what is necessary for display of the results, the print count allows the output rate to be varied. Snapshots are necessary for the post-processing of the results to obtain moments and forces. This can be stored in binary form and used later to obtain the forces and moments. For now we will not bother with snapshots so choose

```
    1      12       1   0
```

For the parameters

```
         @@      u = u + beta*du   K = K_E + gamma*K_G
         INPUT:   beta   |  gamma  |  ramp  '
```

input

```
    1.0   1.0    5
```

The `ramp` modifies `beta` so that the full increment of displacement is not applied until the ramp integer number; this is particularly useful for problems (such as beams) where the initial linear deflections are large. The Newton-Raphson iterations use a tolerance criterion for convergence, however to prevent locking it is good to set a reasonable upper limit on the maximum number of iterations. In response to

```
    @@ algor 1=full N-R  2=mod N-R
    INPUT:   algor  |  iter max  |  tolerance (<1.0e-5)
```

input

```
    1    50    1e-6
```

For dynamic problems, it is not feasible to output the results for every node because there is just too many of them and 99.9% of the information would probably not be used. So `StaDyn` allows you to select a limited number of nodes as output

```
      CHOOSE nodal output:
      TYPE:      node #  |  DoF  |  rate (2=accn)    < 0 0 0 to end>
```

Since each node can have up to six degrees of freedom (3 displacements and 3 rotations) and up to 3 rates (displacement, velocity, and acceleration) it is easy see how the amount of information can mushroom. We will monitor the displacements of the end (load) point

```
    11  1   0
    11  2   0
    11  6   0
     0  0   0
```

This is a recurring menu where the nodes (and DoF) can be chosen in any preferred sequence. The next information required is

```
   @@ Eigen type:   0=none 1=?  2=subspace
   INPUT Eigen: type | rate | # vectors |  scale 1 | 2
```

A characteristic of nonlinear problems is that the stiffness can change with load. Indeed, the stiffness can go to zero resulting in a structurally unstable condition. A unique feature of NonStaD is that it can monitor the changing stiffness by reporting the vibration eigenvalues. This will be explored in the next tutorial, for now input

```
0 100  1  0  0
```

The final piece of information required is the force history

```
TYPE:  Force_Filename -->
```

To which respond

```
force.14
```

The file is then parsed for the appropriate columns

```
@@ INPUT:  Time col  |  Force col  |  # cols
```

Respond

```
1 2 2
```

The force histories can be scaled by

```
@@                  {P}  | {Grav} | 3rd  |
TYPE force scales:   P1  |  P2    |  P3  |  P4
```

We will simply choose

```
1.0  0.0  0.0  0.0
```

The program now proceeds on its way, periodically echoing some numbers to the screen. Finally you are given the menu again, so exit by typing

```
0
0
```

The results are located in the file `<<stadyn.dyn>>` and is stored as

```
time  load_1 ...   resp_1  resp_2  ....  iter  hist_1 ....
```

and in the present case it will have nine columns. Note that the force is the linearly interpolated force.

```
 11.00  100.0   -5.92874   8.29962   1.46851    6.0    .000  .000
```

These are almost identical to the results for the previous tutorial.

5.2 Structural Instability

It is a characteristic of nonlinear systems that the structural stiffness can change under load. This is investigated here.

We consider a plate of dimensions [$6 \times 2 \times 0.1\,\mathrm{in}^3$] made of aluminum and loaded uniformly along the east edge. Such a load causes only an inplane uniform σ_{xx} stress. We will apply the load smoothly to a level above the first buckling load to see what happens. More details on the problem can be found in Reference [11].

The mesh file is called `<<ex_pn.1>>` and the load is in file `<<force.14p>>`. The shape of the history is shown in Figure 5.1(a) as $P(t)$.

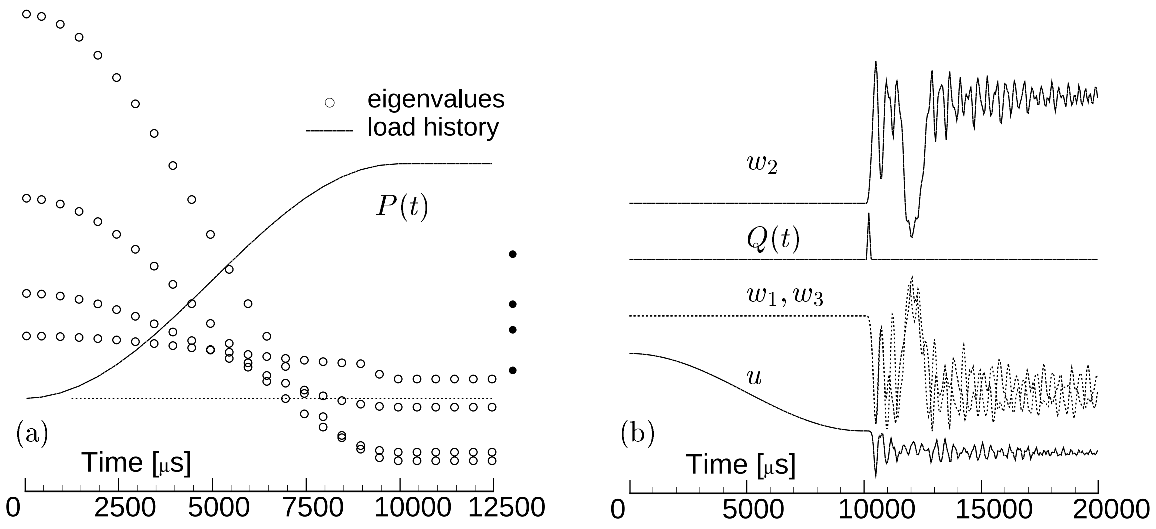


Figure 5.1: Axially loaded plate. (a) Eigenvalue behavior against load. (b) Displacements after critical load and transverse ping applied.

Changing Eigenvalues

Run NonStaD as for the pseudo-time problem

```
C>  nonstad
    2
    ex_pn.1
    14
    1
```

When asked for

```
      TYPE:  time inc  |  # of incs  |   print count  |  snap count
```

respond

```
        50e-6      251      1   10
```

We will do this problem on a time scale where inertia effects are significant. Set the algorithm parameters as

```
        1  1   5
        1  50  1e-6
```

We wish to monitor both the in-plane and out-of-plane behavior, so in response to

```
       CHOOSE nodal output:
       TYPE:      node #  |  DoF  |  rate (2=accn)    < 0 0 0 to end>
```

select

```
       125  1   0
       107  3   0
       113  3   0
       119  3   0
         0  0   0
```

This monitors the displacement at the loaded edge, and the out-of-plane behavior at the three quarter points along the centerline. The next monitoring information required is

```
    @@ Eigen type:   0=none 1=?  2=subspace
    INPUT Eigen: type | rate | # vectors |  scale 1 | 2
```

A unique feature of NonStaD is that it can monitor the changing stiffness by reporting the vibration eigenvalues. This is the main point to be explored in the this tutorial. The `TYPE=1` option chooses vector iteration which reports only the lowest eigenvalue while the `TYPE=2` option chooses subspace iteration which reports the ten lowest eigenvalues. In both cases the eigenvalues are stored in the file `<<stadyn.mon>>`. The remaining option refer to the number of vectors (mode shapes) to be recorded. These are relevant here, so choose

```
        2 10   5  1.0  1.0
```

The final piece of information required is the force history

```
        @@ get {P} history:
        TYPE:  Force_Filename -->
```

Respond with the name and relevant column information

```
        force.14p
        1 2 2
```

The force histories must now be scaled

```
@@                  {P}  | {Grav} | 3rd  |
TYPE force scales:   P1  |  P2    |  P3  |  P4
```

For dynamic problems, there are three contributions to the total magnitude of an applied load. First, there is the scale set in the structure datafile, second there is the scale set in the history file, and third, is the scale that can be set here. The history file has a maximum magnitude of 1000, and a linear buckling analysis gave a lowest eigenvalue of $86,000$, hence to load the plate beyond its first buckling load, input

```
-110  0.0  0.0  0.0
```

The negative sign makes the load compressive. The scales on P_2 and P_3 refer to additional loads — we consider this in the next example.

The program now proceeds on its way, periodically echoing numbers to the screen; some of these numbers may be recognized as those from a vibration eigenanalysis as discussed in Chapter 4. Finally, exit the program by typing

```
0
0
```

The response results are located in the file `<<stadyn.dyn>>` and are stored as

```
time   load_1 load_2 ....  resp_1  resp_2  .... iter hist_1 ...
```

and in the present case it will have 12 columns. The final line is

```
.1250E-01 -11000.0 ....  -.660E-01  .0  .0  .0  ....
```

Note that the only displacement is the in-plane displacement of the loaded edge. The interesting results are in the `<<stadyn.mon>>` file which gives the vibration eigenvalues. These are shown plotted in Figure5.1(a). The results indicate three points of interest. First, as the load changes so do the vibrational characteristics. Since the mass is not changing then we conclude the stiffness is changing. Second, the mode interchanges. Third, it is possible for the eigenvalue to become negative. This means that the frequency $\omega = \sqrt{\lambda}$ becomes complex which in turn says that the system is unstable. However, the displacements do not indicate the large displacements usually associated with an instability. This point is taken up in the next example.

Initiating the Instability

Because a structure becomes unstable does not mean that it will actually exhibit the instability in terms of a large displacement or a bifurcation — an agent is necessary. In this tutorial, we will make the structure unstable just as in the previous tutorial, then we will apply a short duration (ping) transverse loading to initiate the bifurcation behavior.

We need to change `<<ex_pn.1>>` in two ways. First, append to the load lines

```
        113   0 0 1   0 0 0  -3
```

and change the total number of loads to `10 ::boundary loads`. This indicates that a transverse point load is applied at the center of the plate; the history will be read in as the third force history (the second is gravity, if present). The second change is that damping must be added so as to settle the vibrations in a reasonable time.

```
       1    ::SPECIALs GROUP
       2110   8e-2  0.0  0.0
       end
```

The resulting file is labelled `<<ex_pn.1b>>` on the disk. The ping history is given in the file `<<ping.14p>>` and is shown in Figure 5.1(b) as $Q(t)$.

Run NonStaD similarly to the previous tutorial

```
   C>  nonstad
        2
        ex_pn.1b
        14
        1
        50e-6  401   2
        1.0    1.0   5
        1      50    1e-6
       125  1   0
       107  3   0
       113  3   0
       119  3   0
        0   0   0
        2  10   5  1.0 1.0
        force.14p
        1 2 2
```

The only difference is that another force filename must be inputted and the response is monitored over a longer period of time.

```
        @@ get {P_3} history:
        TYPE:  Force_Filename -->
```

Input

```
        ping.14p
        1 2 2
```

The third force can have a spatial distribution specified within the structure file or as a combination of the vibration eigenvectors. A nonzero scale on P_4 in

```
@@                  {P}  | {Grav} | 3rd  |
TYPE force scales:   P1  |  P2    |  P3  |  P4
```

would construct a third force whose spatial distribution is a combination of the first two vibration mode shapes when $\lambda < 0$. We will simply choose

```
-110.0  0.0  1.0e-1  0.0
```

A nonzero scale on P_3 would construct a second force whose spatial distribution is a combination of the first two current vibration mode shapes.

The program now proceeds on its way, periodically echoing numbers to the screen. Finally, exit the program by typing

```
0
0
```

The response results are located in the file <<stadyn.dyn>> and the final line is

```
.19950E-01 1000.0  -.82575E-01 -.51505E-01 .89515E-01 -.63884E-01
                    41.957    -160.82     -227.36     103.06     .00
```

Use the utility DiSPtool to view these results. Note that there are displacements at all the nodes. The complete histories are shown in Figure 5.1(b); note that the transverse displacements begin only after the application of the ping. Furthermore, although the transverse load is symmetric (point load at the center) and the geometry is symmetric, the motion of the plate is nonsymmetric as witnessed by the w_1 and w_3 displacement histories being slightly different. It is also interesting to note the final eigenvalues given in the file <<stadyn.mon>>, (these are shown plotted in Figure5.1(b) as the full circles); they are all positive showing that the plate has found a new equilibrium position. The final deflected shape is a [31] mode as expected from buckling theory.

Dynamic Instability: follower loads

This tutorial solves a nonlinear dynamic problem using the explicit integration scheme. The particular problem considered is the type of instability occurring under follower loads. More details of the problem can be found in Reference [11].

We will use a beam similar to that of <<ex_fn.2>> but modified for damping. That is, the specials section becomes

```
1    ::SPECIALs GROUP
2110   8e-2  0.0  0.0
2310   1.0   0.0  0.0
end
```

The [2310] code indicates that all loads are follower loads. The resulting file is labelled <<ex_fn.5>> on the disk.

Run NonStaD as before but choose the explicit time integration option

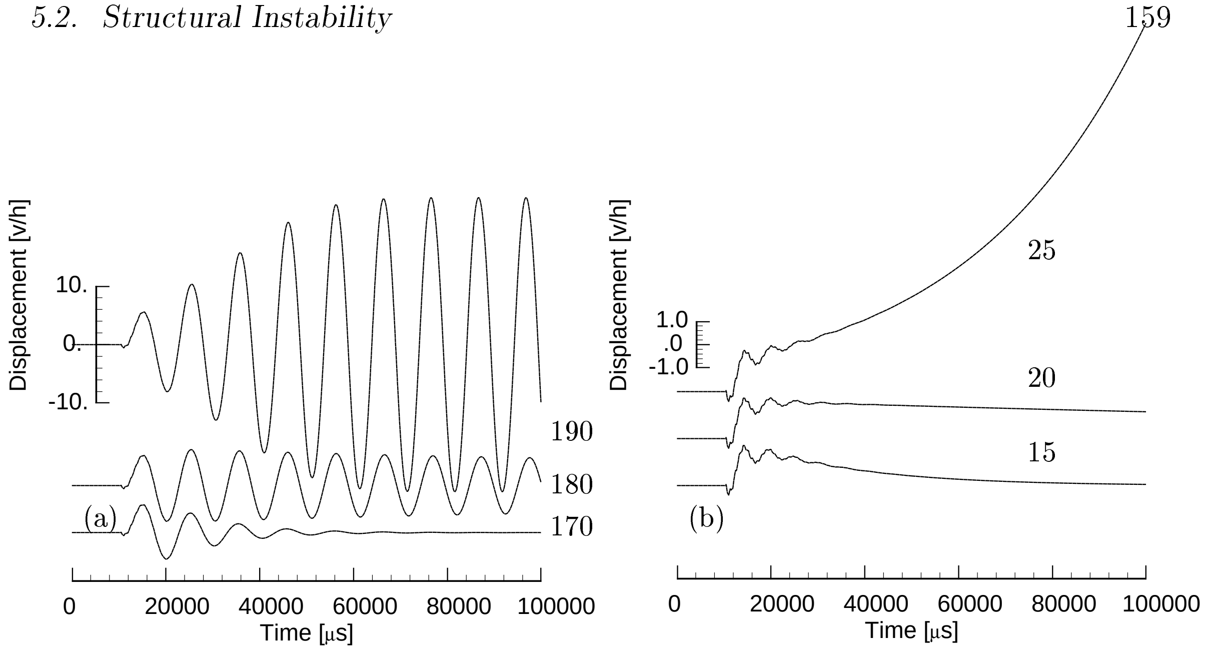


Figure 5.2: Axially loaded beam. (a) Follower force. (b) Constant orientation force.

```
C>  nonstad
    2
    ex_fn.5
    18
    1
```

When asked for

```
TYPE:  time inc  |  # of incs  |   print count
```

respond

```
 5e-6      20001      10
```

Typically, explicit integration requires a very small time step, hence the print count is used to thin the stored data.

We wish to monitor the tip behavior of the beam, so in response to

```
CHOOSE nodal output:
TYPE:      node #  |  DoF  |  rate (2=accn)    < 0 0 0 to end>
```

select

```
 11  1   0
 11  2   0
 11  1   1
 11  2   1
  0  0   0
```

The type and rate of storing the snapshots is selectable

```
@@ SNAPshot type:  0=none  1=partial  2=full
@@ CHOOSE : type  |  count
```

Choose

```
2   200
```

The final piece of information required are the force histories: primary, gravity, and additional. Respond

```
force.14p
1 2 2
force.14p
1 2 2
1  2
ping.14p
1 2 2
```

The ping load is placed at the beam center

```
@@ TYPE:   x y z   |  DoF   |  P_scale
           5 0 0      2        0.1
```

The force histories can be scaled

```
@@                  {P}  | {Grav} | 3rd  |
TYPE force scales:   P1  |  P2    |  P3  |  P4
```

Input

```
-0.170  0.0  1.0  0.0
```

The negative sign makes the primary load compressive and the file `<<force.14p>>` already is scaled to 1000.

The program now proceeds on its way, periodically echoing some numbers to the screen. Finally you are given the menu again so exit by typing

```
0
0
```

The results are located in the file `StaDyn.DYN` and is stored as

```
time    load_1 load_2 load_3    resp_1  resp_2  ....  hist_1 ...
```

and in the present case it will have 10 columns. Use the utility `DiSPtool` to view these results and they should look like those in Figure 5.2(a). For reference, the final line is

```
.10000     -170.0  .000  .000  -.1700E-02 -.113169E-03  .738069E-05
                               ...  1000.00  1000.00  .000  .000
```

which indicates a relatively small displacement.

Now repeat the run but having changed the load scales to

```
 TYPE force scales:   P1  |  P2   |  P3  |  P4
-.190  0.0   1.0  0.0
```

The final line is

```
.10000     -190.0  .000  .000  -.776172E-01 -1.12609  -234.720
                                 ...   1000.00  1000.00 .000 .000
```

indicating a much larger displacement as shown in Figure 5.2(a).

By way of contrast, repeat the run having set the follower load off.

```
1    ::SPECIALs GROUP
2110   8e-2  0.0  0.0
2310   0.0   0.0  0.0
end
```

The results are shown in Figure 5.2(b). First note that the load to cause the instability is considerably lower than the case of the follower load. Second, the characteristic of the instability is different in that for this latter case the displacement increases monotonically, whereas for the follower load case it does so while oscillating. This is the difference between a static instability and a dynamic instability.

5.3 Elastic-Plastic Behavior using Simplex

Simplex is designed to run as a console program under `MS Windows`. Note that all instructions are case insensitive; we will vary the case only to help make the instructions clearer. This tutorial will demonstrate it ability to model elastic-plastic behavior.

Getting Started

To run the program, type (at the 'C prompt')

```
C>  simplex
```

(note the name of the executable.) to be given the opening menu

```
MAIN menu:
        0: Quit
        2: Read in structure DataFile
           <<linear analyses>>
       30: Form system matrices
       60: Static loading
       80: Transient loading
       90: Eigenalysis
           <<imposed large deformation>>
      200: Incremental 3-D
           <<nonlinear elastic, elastic-plastic>>
      400: Incremental 3-D
           <<heat conduction>>
      500: Quasi static
         :
      700: PostScript plot files
      911: Ikayex information
```

One difference in comparison to NonStaD is that Simplex has a full implementation of the linear analyses including the option to form the system matrices. Quit by typing

```
0
```

There are two files worth looking at. The first is the log file `<<simplex.log>>`; use the `list` utility to peruse it. It contains

```
@@ Simplex version 2.62, January 2007
@@ DATE:  8-10-2007      TIME: 15:44
@@ MAXimum storage   :     5000000
@@ MAXimum elements  :        3000
@@ MAXimum nodes     :        3000
@@ MAXimum force incs:        3000   200
@@ ITERmax           :          16
@@ rtol              :    1.000000E-10
@@ ALLOCATION succeeded
          0 ::MAIN
@@ Simplex OK, ended from MAIN
```

The second is <<simplex.cfg>>. This is the configuration file and has in it

```
    5000000     ::MaxStorage
       3000     ::MaxNode
       3000     ::MaxElem
       3000  200     ::MaxForce
          1      ::ilump
         16      ::itermax
         1.0000000E-010      ::rtol
@@ DATE:  8-10-2007      TIME: 15:54
```

This allows the run time setting of the dimensions of the arrays. If a memory allocation failure occurs, this is the place to make the adjustments.

Uniaxial Loading-Unloading Behavior

We consider a bar of dimensions $[10 \times 1.0 \times 0.5\,\text{in}^3]$ made of aluminum and loaded uniformly at one end. Such a load generates essentially a uniaxial state of stress σ_{xx}. We will apply the load history smoothly to a level above yield, then unload and reload to above yield again.

The mesh file is called <<ex_ep.1>> and is the same as generated in Section 2.9; the load history is specified in file <<force.53p>> and contains

```
       0   0    0
       2   0.6  0
       10  1.0  0
       12  1.1  0
       14  0.0  0
       15  0.0  0
       14  0.0  0
       19  1.6  0
       20  1.6  0
```

This will be interpolated as needed.

Run Simplex as for the pseudo-time problem using NonStaD

```
   C>  simplex
```

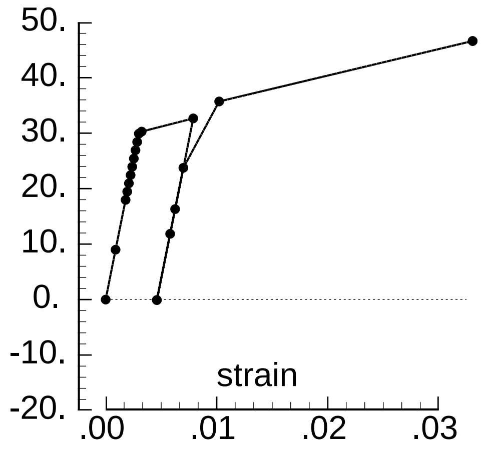


Figure 5.3: Stress-strain behavior of axially loaded bar.

```
2
ex_ep.1
400
```

This accesses the nonlinear analyses menu

```
410: Incremental Static  E   3-D TL
420: Incremental dynamic E   3-D TL
424:    ..         ..    E   3-D TL mult
440: Incremental dynamic E   3-D EXP
442:    ..         ..    gen 3-D EXP
450: Incremental Static  Rubber  3-D TL
460: Incremental Static  E/P 3-D TL
```

The designations TL refer to the implicit Total Lagrangian and explicit Central Difference schemes, respectively. Choose the elastic/plastic analysis

```
460
1
```

When asked for

```
INPUT:  time inc  |  # of incs  |   print count  |  snap count
```

respond ➤

```
1.0    21   1   1
```

For the parameters

```
@@       u = u + beta*du   K = K_E + gamma*K_G
INPUT:   beta   |  gamma  |  ramp  '
```

input

```
      1.0   1.0    5
```

The `ramp` modifies `beta` so that the full increment of displacement is not applied until the ramp integer number; this is particularly useful for problems (such as beams) where the initial linear deflections are large. The Newton-Raphson iterations use a tolerance criterion for convergence, however to prevent locking it is good to set a reasonable upper limit on the maximum number of iterations. In response to

```
    @@ algor: 1=full N-R  2=mod N-R
    INPUT:   algor  |  iter max  |  tolerance (<1.0e-5)
```

input

```
      1    60    1e-6
```

Most of the information we require will come from the post-processing, but this time through the analysis we will monitor some node behaviors to confirm the correct solution of the problem. In response to

```
      CHOOSE nodal output:
      TYPE:      node #  |  DoF                  < 0 0 to end>
```

choose

```
      61  1
      61  2
      61  3
       0  0
```

This monitors the displacements midway along the bar. The next monitoring information required is

```
   @@ Eigen types:   0=none 1=?  2=subspace
   INPUT Eigen: type | rate | # vectors |  scale 1 | 2
```

A unique feature of NonStaD and Simplex is that they can monitor the changing stiffness by reporting the vibration eigenvalues. This will not be explored in the this tutorial, so type

```
      0 10   5
```

During the formation of the mass matrix, the option is given to store it, we will choose not to store it with

```
      0
```

The final information required are the load histories.

```
@@ get {P} history:
TYPE:  Load_Filename -->
```

Respond with the name and relevant column information

```
force.53p
1 2 3
```

Then the gravity history

```
@@ get {Grav} history:
TYPE:  Load_Filename -->
```

Respond with

```
force.53p
1 3 3
```

The load histories must now be scaled

```
@@                  {P}  | {Grav} | 3rd  |
TYPE force scales:   P1  |  P2    |  P3  |  P4
```

For dynamic problems, there are three contributions to the total magnitude of an applied load. First, there is the scale set in the structure datafile, second there is the scale set in the history file, and third, is the scale that can be set here. The history file has a nominal value of order 1.0 and yielding is expected at $\sigma_{xx} = \sigma_Y = 30000$ psi. Since 1 psi was applied at then end, then we will scale the loads as

```
30000   0.0  0.0  0.0
```

The negative sign makes the load compressive.

The program now proceeds on its way, periodically echoing numbers to the screen. Finally, exit the program by typing

```
0
```

The response results are located in the file `<<stadyn.dyn>>` and are stored as

```
time   load_1 load_2 load_3  resp_1  resp_2  .... iter hist_1 ...
```

and in the present case it will have 15 columns. The final line is

```
20.00    48000.0  0.0  0.0   .153508  .773796E-02  .386638E-02 ....
```

Note that the axial displacement corresponds to an axial strain of about $\epsilon_{xx} \approx .153508/5 = .0307$in/in $\approx 3.1\%$ and is therefore well into the yield region of $\epsilon_Y = \sigma_Y/E = 30000/10e6 = 0.3\%$.

Post-processing the Results

All the displacement data is stored in the snapshot file `<<stadyn.snp>>`, quantities such as stress, plastic strain, and so on must be obtained by post-processing these snapshots.

Run Simplex to get to the elastic-plastic post-processing menu

```
C>  simplex
    2
    ex_ep.1
    400
    460
    2
```

The snapshots can be interrogated individually or a sequence

```
 CHOOSE snapshot:
        0=return
        #=which snap
       -#=all   snaps
```

We will look at traces so choose

```
       -1
```

and get the menu

```
    CHOOSE output:
           0=return
             <movie>
          111=node displacement
          141=store displacement data
            <<Hex20>>
      41x=displacement history
          411=displacement
      42x=Lagrange strain  history
          421=element strain IP
          422=element strain N
          424=nodal   strain average
          427=element strain special
          428=nodal   strain special
      43x=Kirchhoff stress  history
          431=element stress IP
          432=element stress N
          434=nodal   stress average
          437=element stress special
          438=nodal   stress special
      44x=Cauchy stress  history
          441=element stress IP
          442=element stress N
          444=nodal   stress average
          447=element stress special
          448=nodal   stress special
      450=force  history
      46x=Plastic strain  history
          461=element all IP
          462=element strain N
          464=nodal   strain average
          468=nodal   strain special
          469=nodal   all average
```

Choose to store all the stress and strains at the particular Node 61 and quit

```
469
61
0
```

The results are in the file `<<stadyn.dyn>>` in the format

```
time   Exx Eyy .... SKxx SKyy .... SCxx SCyy .... EPxx EPyy ....
```

where `SK` are the Kirchhoff stresses and `SC` the Cauchy stresses. The last line contains

```
20.00   .330446E-01 ... 46623.7 ... 49664.6 ... .283885E-01 ...
```

The complete stress-strain response is shown in Figure 5.3.

Chapter 6

Utilities

Since it is cumbersome for a single program such as QED to do all that is required (analysis, postprocessing, model generation, plotting, editing, documentation of results, and so on), the design decision was taken to shift some of these functions onto external programs. An obvious example of this is the fact that GenMesh and StaDyn are executable programs separate from QED itself. Thus the user, potentially, has the option of substituting their own favorite programs for the ones provided.

This chapter reviews some of the additional utilities provided as part of the QED package.

6.1 PlotMesh

There is a need for visual verification of the input data and this is what PlotMesh provides. It is not a graphics environment for data input but a simple means of visually verifying and interrogating the structure input datafile. It works on the same files that is used as input to StaDyn/NonStaD/Simplex as well as the intermediate meshes generated by GenMesh.

To run this utility, simply type

```
C> plotmesh
```

The input file must adhere to the conventions of StaDyn/NonStaD/Simplex — after all it is its input file that is being checked. Interaction is through the keyboard, generally by pressing the highlighted key. When input in the form of numbers is required, the input is terminated with a carriage return. To read in a file, press

```
0
```

and type the filename

```
filename
```

Once the structure is displayed, then various keys can be used to enhance the information. The help key

```
h
```

displays the various active keys. This is a *toggle* switch which means it is turned off by pressing it a second time.

After PlotMesh reads in the structure datafile, it makes a best guess as to how to display the mesh on the screen. If it is too large or too small use the magnification to size appropriately. If white bands appear across the screen then this is an indication that the scales are too large. The structural information is displayed through the use of the three cursor switches

```
a/A  b/B  c/C
```

These check the numbering of the nodes (triangle Apexes), Boundaries, and the element Centroids, respectively. The nearest node to any point can be found by `[pressing nx]` and inputting the coordinates.

There are the refresh and home keys

```
r/R
```

Refresh means the screen is redrawn as is, and `R` means the initial settings are re-established (except for any scalings). All of the keys can be used in combination if so desired. The program is exited by pressing the

```
q
```

key.

Meshes can also be viewed three-dimensionally by pressing

```
d
```

for a 3-D animated rotation. The keys

```
x/X  y/Y  z/Z
```

animate the rotation about these three axes. Incremental rotation of the structure is obtained by pressing the keys

```
u/U  v/V w/W
```

Finally, the `%size` allows the display to show the difference between plate and frame elements.

The size of the PlotMesh window can be changed by editing the second line of the configuration file `<<plotmesh.cfg>>` (while Plotmesh is not running).

6.2 FormGen

There is not a general graphical user interface for running GenMesh. Further, the standard models produced by QED (while varied and flexible) are only a subset of those produced by GenMesh. The program FormGen (FORM GENerator) was written to be the interface for GenMesh. This program is now part of the QED program itself and is accessed off the `[Create Models]` menu as the `[:General]` option.

The interface motif for FormGen is that of forms: the information required by GenMesh for its model building is laid out as fields to be entered in a form. There are six forms:

```
Plane 2-D models
Arbitrary 2-D Models
Thin-wall 3-D Structures
Frame 3-D Structures
Merge Models
Render Structure DataFile
```

The information for each form is stored in the file `<<fgen.cfg>>` so that each working form is easily recreated.

When the form is completed, it is converted into script files (either `<<inmesh>>` or `<<insdf>>`) and the model rendered by running GenMesh in batch mode in the background. The produced model can then be viewed using PlotMesh. If the final name of the model is given as `<<qed.sdf>>`, then it can be used directly for analysis by QED.

All interaction is via the keyboard, generally by pressing the first uppercase letter key or the number/letter before the colon. When input in the form of numbers is required, the input is terminated with a carriage return.

If a particular form is not general enough for the model at hand; for example, the 3-D frame has more than 15 control points, a simple way to overcome the limitation is to edit and amplify the generated script file. That is, use the forms to produce a (complete) reduced model, then directly modify the produced script file(s) to include the additional aspects. The model is then rendered by typing

```
genmesh  < inmesh
```

6.3 DiSPtool

Many of the results produced by StaDyn/NonStaD are in the form of columns of numbers against time. There is a need for visual interpretation of these numbers as well as for further manipulation. This is what DiSPtool (DIgital Signal Processing TOOL) provides.

Interaction is through the keyboard, generally by pressing the highlighted key. When input in the form of numbers is required, the input is terminated with a carriage return.

To run this program, type

```
C> disptool
```

to be given the opening menu

```
Quit
Time Domain Input:
     One_Channel
     Two_Channel
Freq Domain Input:
     One_Channel
     Two_Channel
View Multiple Columns
```

`Time Domain Input` takes one or two channels of time dependent data and does various transforms on them including smoothing, windowing, and Fourier transforms. It can also generate particular histories that can be used. for example, as input force histories. A reminder of the available keys is given at the bottom of the screen.

`Freq Domain Input` takes one or two channels of frequency dependent data and does various transforms on them including windowing, and Inverse Fourier transforms. A reminder of the available keys is given at the bottom of the screen.

`View Multiple Columns` reads a file with up to 99 columns of data, determines automatically how many columns, and displays the columns in various ways. A reminder of the available keys is given at the bottom of the screen.

In each case, a window can be zoomed by pressing the window number (1,2,3, or 4) and a second file can be superposed by `[pressing i]`. Also, selected columns can be stored by `[pressing s]`.

Re-reading of a file (and re-doing the various manipulations) can be forced by `[pressing R]`. This is useful if the file was updated.

The program is exited by pressing the

```
q
```

key.

6.4 GSview and GhostScript

Both StaDyn/NonStaD and GenMesh provide some support for producing figure and plots in PostScript. GSview is the utility through which these can be viewed on the screen or rendered as hard copy by a printer.

GSview is a graphical interface for Ghostscript under MS-Windows. Ghostscript is an interpreter for the PostScript page description language used by laser printers. For documents following the Adobe PostScript Document Structuring Conventions, GSview allows selected pages to be viewed or printed. It supports an impressive list of printers. Some of its features include:
- view pages in arbitrary order (Next, Previous, Goto)
- selectable display resolution, depth, alpha
- single button zoom
- print selected pages using Ghostscript, page offset can be applied
- extract selected pages to another file
- copy display bitmap to clipboard
- graphically select and show bounding box for EPS file
- display PDF files.

The complete GhostView package is included on the disk as a possible way to support graphical visualization of the meshes and the contours. It is also used to view the manual. The package contains

```
setup.exe         MS-Windows installation program
gsview.zip
wizunz32.dll      MS-Windows Info-ZIP unzip utility
gs403ini.zip
gs403w32.zip      MS-Windows Win32 Aladdin Ghostscript
gs403fn1.zip      Aladdin Ghostscript fonts.
```

Installing the package is simple, it just requires typing SETUP and a wizard controls the rest. Configuration of GSview will occur the first time it is run.

The latest version of GSview should be available from

ftp://ftp.cs.wisc.edu/ghost/rjl/gsview*.zip

A World Wide Web home page for Ghostscript, Ghostview and GSview is at

http://www.cs.wisc.edu/ ghost/

GSview uses Ghostscript to display the contents of the PostScript files. Ghostscript is written and owned by Aladdin Enterprises. Aladdin Ghostscript comes with a licence that is more restrictive than the GNU Licence; in particular, it restricts the distribution of Aladdin Ghostscript in commercial contexts. Please see the file PUBLIC that accompanies Aladdin Ghostscript for more details.

6.5 Automating Running in Batch Mode

The stand-alone executables in the QED package can be run in batch mode. Indeed, this is how QED itself interacts with them. When tasks have to be repeated with only slight changes in parameters, then it can be very beneficial to automate the process. This esction describes how this might be done.

Assume an executive program (written in Fortran or MatLab) can produce the changed scripts then the following is some code that can automatically run the executables.

```
      subroutine do_batch(ibat)
         character*80 commnd,exe_locn
         exe_locn='c:\qed'
         ilent=lentrim(exe_locn)
         if (ibat .eq. 11) then
             commnd=exe_locn(1:ilent)//'\genmesh < inmesh  '
         elseif (ibat .eq. 12) then
             commnd=exe_locn(1:ilent)//'\genmesh < insdf   '
         elseif (ibat .eq. 21) then
             commnd=exe_locn(1:ilent)//'\stadyn  < instad  '
         elseif (ibat .eq. 22) then
             commnd=exe_locn(1:ilent)//'\stadyn  < inpost  '
         endif
         call syscom(commnd)
      return
      end
c
      subroutine syscom(commnd)
c        different OS's will have slightly different
c        system command syntex
         use msflib
         character*(*) commnd
         logical*4     success
         success=systemqq(commnd)
c        call system(commnd)
      return
      end
c
c
```

References

[1] **Balfour, J.A.D.**, *Computer Analysis of Structural Frameworks*, Nichols, New York, 1986.

[2] **Bathe, K.-J.**, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1982, 2/E 1996.

[3] **Bathe, K.-J.**, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[4] **Batoz, J-L., Bathe, K-J. and Ho, L-W.**, "A Study of Three-Node Triangular Plate Bending Elements," *International Journal for Numerical Methods in Engineering*, **15**, pp. 1771–1812, 1980.

[5] **Batoz, J-L.**, "An Explicit Formulation for an Efficient Triangular Plate-Bending Element," *International Journal for Numerical Methods in Engineering*, **18**, pp. 1077–1089, 1982.

[6] **Bergan, P.G. and Felippa, C.A.**, "A Triangular Membrane Element with Rotational Degrees of Freedom," *Computer Methods in Applied Mechanics and Engineering*, **50**, pp. 25–69, 1985.

[7] **Cook, R.D., Malkus, D.S. and Plesha, M.E.**, *Concepts and Applications of Finite Element Analysis, 3/E*, Wiley & Sons, New York, 1989.

[8] **Dhondt, G.**, *The Finite Element Method for Three-Dimensional Thermomechanical Applications*, Wiley & Sons, Chichester, England, 2004.

[9] **Doyle, J.F.**, *Wave Propagation in Structures*, Springer-Verlag, New York, 1989, 2/E 1997.

[10] **Doyle, J.F.**, *Static and Dynamic Analysis of Structures*, Kluwer, The Netherlands, 1991.

[11] **Doyle, J.F.**, *Nonlinear Analysis of Thin-walled Structures: Statics, Dynamics, and Stability*, Springer-Verlag, New York, 2001.

[12] **Doyle, J.F.**, *Guided Explorations in the Mechanics of Structures: a modern course on principles and methods of solution*, Class Notes, Purdue University, 2007.

[13] **Eisley, J.G.**, *Mechanics of Elastic Structures*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[14] **Graff, K.F.**, *Wave Motion in Elastic Solids*, Ohio State University Press, Columbus, 1975.

[15] **Lalanne, M., Berthier, P. and Der Hagopian, J.**, *Mechanical Vibrations for Engineers*, Wiley & Sons, New York, 1983.

[16] **Leissa, A.W.**, *Vibration of Plates*, NASA SP-160, 1969.

[17] **Martin, H.C.**, *Introduction to Matrix Methods of Structural Analysis*, McGraw-Hill, New York, 1966.

[18] **Meirovitch, L.**, *Elements of Vibration Analysis*, McGraw-Hill, 1986.

[19] **Przemieniecki, J.S.**, *Theory of Matrix Structural Analysis*, Dover, New York, 1985.

[20] **Redwood, M.**, *Mechanical Waveguides*, Pergamon Press, New York, 1960.

[21] **Soedel, W.**, *Vibrations of Shells and Plates*, Marcel Dekker, New York, 1981.

[22] **Stricklin, J.A, Haisler, E.E., Tisdale, P.R. and Gunderson, R.**, "A Rapidly Converging Triangular Plate Element," *AAIA Journal*, **7**(1), pp. 180–181, 1969.

[23] **Timoshenko, S.P. and Gere, J.M.**, *Theory of Elastic Stability*, McGraw-Hill, New York, 1963.

[24] **Timoshenko, S.P. and Woinowsky-Krieger, S.**, *Theory of Plates and Shells*, McGraw-Hill, New York, 1968.

[25] **Weaver, W. and Gere, J.M.**, *Matrix Analysis of Framed Structures*, Van Nostrand, New York, 1980.

[26] **Yang, T.Y.**, *Finite Element Structural Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

Index