27

A Complete Plane Stress FEM Program

TABLE OF CONTENTS

§**27.1. Introduction**

This Chapter describes a complete finite element program for analysis of plane stress problems. Unlike the previous chapters the description is top down, i.e. starts from the main driver down to more specific modules.

The program can support questions given in the take-home final exam, if they pertain to the analysis of given plane stress problems. Consequently this Chapter serves as an informal users manual.

§**27.2. Analysis Stages**

As in all DSM-based FEM programs, the analysis of plane stress problems involves three major stages: (I) preprocessing or model definition, (II) processing, and (III) postprocessing.

The preprocessing portion of the plane stress analysis is done by the first part of the *problem script*, driver program, already encountered in Chapters 21-22. The script directly sets the problem data structures. Preprocessing tasks include:

I.1   Model definition by direct setting of the data structures.
I.2   Plot of the FEM mesh for verification. At the minimum this involves producing a mesh picture that shows nodes and element labels.

The processing stage involves three tasks:

II.1   Assembly of the master stiffness equations $\mathbf{Ku} = \mathbf{f}$. The plane stress assembler is of multiple element type (MET). This kind of assembler was discussed in §27.4. Element types include various plane stress Iso-P models as well as bars.
II.2   Application of displacement BC by a modification method that produces $\hat{\mathbf{K}}\mathbf{u} = \hat{\mathbf{f}}$. The same modules described in §21.3.3 are used, since those are application problem independent.
II.3   Solution of the modified equations for node displacements $\mathbf{u}$. The built-in *Mathematica* function LinearSolve is used for this task.

Upon executing the processing steps, the nodal displacement solution is available. The postprocessing stage embodies three tasks:

III.1   Recovery of node forces including reactions through built-in matrix multiplication $\mathbf{f} = \mathbf{Ku}$.
III.2   Recovery of plate stresses and bar forces (if bars are present). The former are subject to interelement averaging (Chapter 28) to get nodal stresses.
III.3   Print and plotting of results.

§**27.3. Model Definition**

The model-definition data may be broken down into three sets, which are listed below by order of appearance:

$$\text{Model definition} \begin{cases} \text{Geometry data: node coordinates} \\ \text{Element data: type, connectivity, material and fabrication} \\ \text{Degree of freedom (DOF) activity data: force and displacement BCs} \end{cases}$$

(27.1)

The element data is broken down into four subsets: type, connnectivity, material and fabrication, each of which has its own data structure. The degree of freedom data is broken into two subsets:

Table 27.1. Plane Stress Model Definition Data Structures

Long name	Short name	Dimensions	Description
NodeCoordinates	nodxyz	numnod x 2	Node coordinates in global system
ElemTypes	elenod	numele	Element type identifiers
ElemNodes	elenod	numele x \<var\>	Element node lists
ElemMaterials	elemat	numele x \<var\>	Element material properties
ElemFabrications	elefab	numele x \<var\>	Element fabrication properties
NodeDOFTags	nodtag	numnod x 2	Node freedom tags marking BC type
NodeDOFValues	nodval	numnod x 2	Node freedom specified values
ProcessOptions	prcopt	\<var\>	Processing specifications

Notation: numnod: number of nodes (no numbering gaps allowed, each node has two)
displacement DOFs; numele: number of elements; \<var\> variable dimension.
Long names are used in user-written problem scripts. Short names are used in programmed modules.
Dimensions refer to the first two level Dimensions of the *Mathematica* list that implements the
data structure. These give a guide as to implementation in a low level language such as C.

tags and values. In addition there are miscellaneous process options, such as the symbolic versus numeric processing. These options are conveniently collected in a separate data set.

Accordingly, the model-definition input to the plane stress FEM program consists of eight data structures, which are called NodeCoordinates, ElemTypes, ElemNodes, ElemMaterials, ElemFabrications, NodeDOFTags, NodeDOFValues and ProcessOptions. These are summarized in Table 27.1.

The configuration of these data structures are described in the following subsections with reference to the benchmark problems and discretizations shown in Figures 27.1 and 27.2.

§27.3.1. Benchmark Problems

Figure 27.1(a) illustrates a rectangular steel plate in plane stress under uniform uniaxial loading in the y (vertical) direction. Note that the load q is specified as force per unit area (kips per square inch); thus q has not been integrated through the thickness h. The exact analytical solution of the problem is[1] $\sigma_{yy} = q$, $\sigma_{xx} = \sigma_{xy} = 0$, $u_y = qy/E$, $u_x = -\nu u_y = -\nu qx/E$. This problem should be solved exactly by *any* finite element mesh as long as the model is consistent and stable. In particular, the two one-quadrilateral-element models shown in 27.1(b,c).

A similar but more complicated problem is shown in Figure 27.2: a rectangular steel plate dimensioned and loaded as that of Figure 27.1(a) but now with a central circular hole. This problem is defined in Figure 27.2(a). A FEM solution is to be obtained using the two quadrilateral element models (with 4-node and 9-nodes, respectively) depicted in Figure 27.2(b,c). The main result sought is the stress concentration factor on the hole boundary, and comparison of this computed factor with the exact analytical value.

---

[1] The displacement solution $u_y = qy/E$ and $u_x = -\nu u_y$ assumes that the plate centerlines do not translate or rotate, a condition enfoirced in the FEM discretizations shown in Figure 27.1(b,c).
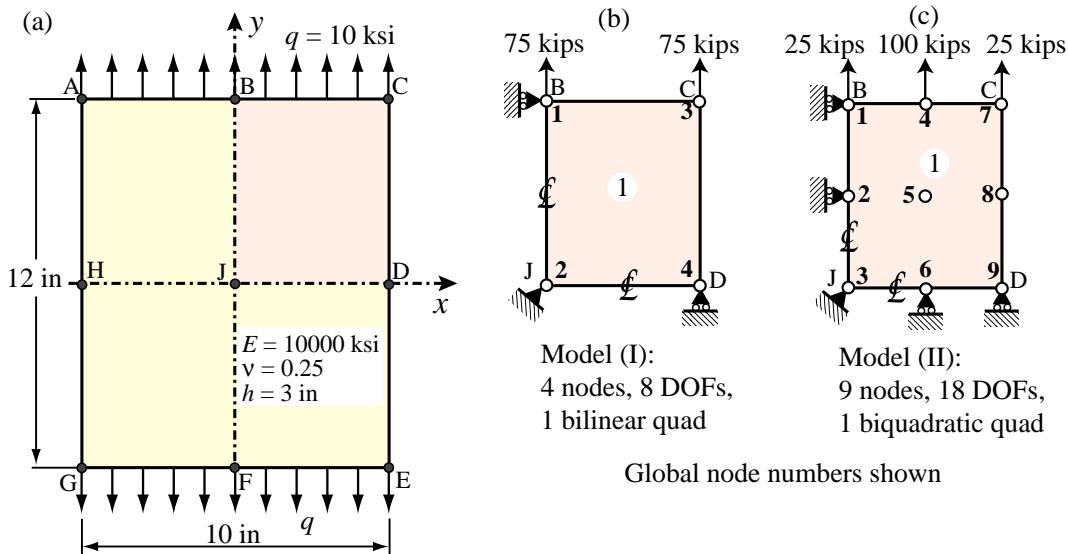
FIGURE 27.1. Rectangular plate under uniform uniaxial loading, and two one-element FEM discretizations of its upper right quadrant.

§27.3.2. Node Coordinates

The geometry data is specified through `NodeCoordinates`. This is a list of node coordinates configured as

$$\boxed{\texttt{NodeCoordinates = \{\{}x_1, y_1\texttt{\},\{}x_2, y_2\texttt{\}, } \ldots \texttt{ \{}x_N, y_N\texttt{\}\}}} \qquad (27.2)$$

where N is the number of nodes. Coordinate values should be normally be floating point numbers;[2] use the `N` function to insure that property if necessary. Nodes must be numbered consecutively and no gaps are permitted.

Example 27.1. For Model (I) of Figure 27.1(b):
```
NodeCoordinates=N[{{0,6},{0,0},{5,6},{5,0}}];
```

Example 27.2. For Model (II) of Figure 27.1(c):
```
NodeCoordinates=N[{{0,6},{0,3},{0,0},{5/2,6},{5/2,3},{5/2,0},{5,6},{5,3},{5,0}}];
```

Example 27.3. For Model (I) of Figure 27.2(a), using a bit of coordinate generation:
```
s={1,0.70,0.48,0.30,0.16,0.07,0.0};
xy1={0,6}; xy7={0,1}; xy8={2.5,6}; xy14={Cos[3*Pi/8],Sin[3*Pi/8]};
xy8={2.5,6}; xy21={Cos[Pi/4],Sin[Pi/4]}; xy15={5,6};
xy22={5,2};  xy28={Cos[Pi/8],Sin[Pi/8]}; xy29={5,0}; xy35={1,0};
NodeCoordinates=Table[{0,0},{35}];
Do[NodeCoordinates[[n]]=N[s[[n]]   *xy1+(1-s[[n]])    *xy7], {n,1,7}];
Do[NodeCoordinates[[n]]=N[s[[n-7]] *xy8+(1-s[[n-7]])  *xy14],{n,8,14}];
Do[NodeCoordinates[[n]]=N[s[[n-14]]*xy15+(1-s[[n-14]])*xy21],{n,15,21}];
```

---

[2] Unless one is doing a symbolic or exact-arithmetic analysis. Those are rare at the level of a full FEM analysis.
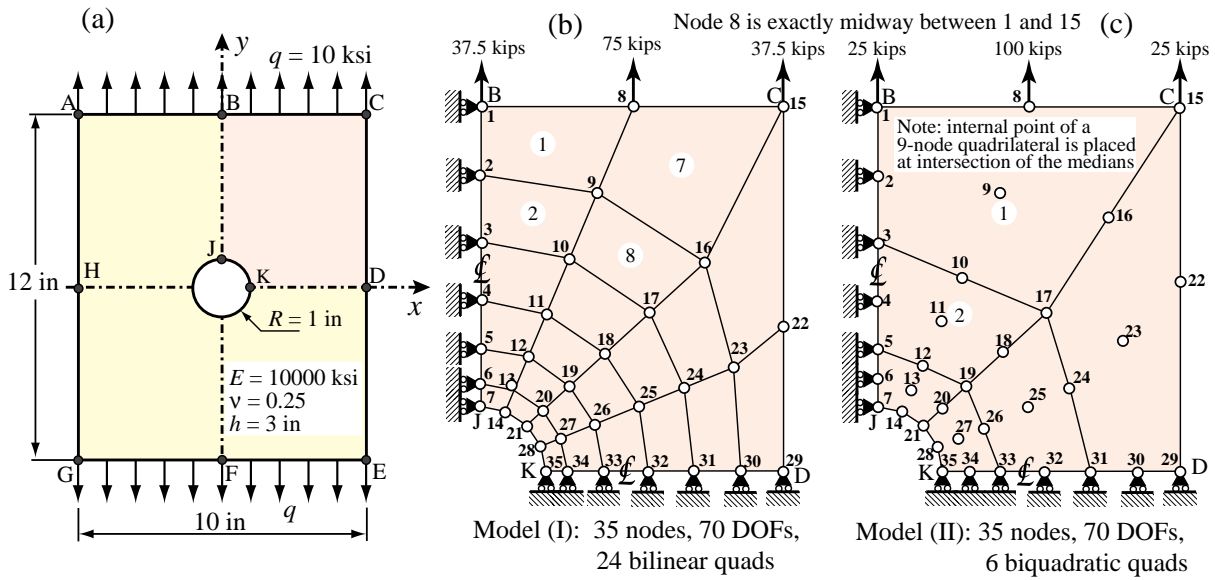
FIGURE 27.2. Plate with a circular hole and two FEM discretizations of its upper right quadrant. Only a few element numbers are shown to reduce clutter. Note: the meshes generated with the example scripts given later differ slightly from those shown above. Compare, for example, mesh in (b) above with that in Figure 27.3.

```
Do[NodeCoordinates[[n]]=N[s[[n-21]]*xy22+(1-s[[n-21]])*xy28],{n,22,28}];
Do[NodeCoordinates[[n]]=N[s[[n-28]]*xy29+(1-s[[n-28]])*xy35],{n,29,35}];
```

The result of this generation is that some of the interior nodes are not in the same positions as sketched in Figure 27.2(b), but this slight change hardly affects the results.

§27.3.3. Element Type

The element type is a label that specifies the type of model to be used. These labels are placed into an element type list:

$$\boxed{\texttt{ElemTypes = \{ type}^{(1)}\texttt{, type}^{(2)}\texttt{, ... type}^{N_e}\texttt{ \}}} \tag{27.3}$$

Here $\texttt{type}^e$ is the type identifier of the e-th element specified as a character string. N_e is the number of elements; no element numbering gaps are allowed. Legal type identifiers are listed in Table 27.2.

Example 27.4. For Model (I) in Figure 27.2(a):
```
ElemTypes=Table["Quad4",{numele}];
```

Example 27.5. For Model (II) in Figure 27.2(b):
```
ElemTypes=Table["Quad9",{numele}];
```

Here `numele` is a variable that contains the number of elements. This can be extracted, for example, as `numele=Length[ElemNodes]`, where `ElemNodes` is defined below, if `ElemNodes` is defined first as is often the case.

Table 27.2. Element Type Identifiers Implemented in Plane Stress Program

Identifier	Nodes	Model for	Description
`"Bar2"`	2	bar	2-node bar element
`"Bar3"`	3	bar	3-node bar element; may be curved
`"Trig3"`	3	plate	3-node linear triangle
`"Trig6"`	6	plate	6-node quadratic triangle (3-interior point Gauss rule)
`"Trig6.-3"`	6	plate	6-node quadratic triangle (midpoint Gauss rule)
`"Trig10"`	10	plate	10-node cubic triangle (7 point Gauss rule)
`"Quad2"`	4	plate	RS 4-node iso-P bilinear quad (2×2 Gauss rule)
`"Quad4.1"`	4	plate	RD 4-node iso-P bilinear quad (1-point Gauss rule)
`"Quad8"`	8	plate	RS 8-node iso-P serendipity quad (3×3 Gauss rule)
`"Quad8.2"`	8	plate	RD 8-node iso-P serendipity quad (2×2 Gauss rule)
`"Quad9"`	9	plate	RS 9-node iso-P biquadratic quad (3×3 Gauss rule)
`"Quad9.2"`	9	plate	RD 9-node iso-P biquadratic quad (2×2 Gauss rule)

RS: rank sufficient, RD: Rank deficient.

§27.3.4. Element Connectivity

Element connectivity information specifies how the elements are connected.[3] This information is stored in `ElemNodes`, which is a list of element nodelists:

$$\boxed{\texttt{ElemNodes = \{\{enl}^{(1)}\texttt{\}, \{enl}^{(2)}\texttt{\}, ... \{enL}^{N_e}\texttt{\}\}}} \tag{27.4}$$

Here $\texttt{enl}^e$ denotes the lists of nodes of the element e (as given by global node numbers) and N_e is the total number of elements.

Element boundaries must be traversed counterclockwise (CCW) but you can start at any corner. Numbering elements with midnodes requires more care: begin listing corners CCW, followed by midpoints also CCW (first midpoint is the one that follows first corner when traversing CCW). If element has thirdpoints, as in the case of the 10-node triangle, begin listing corners CCW, followed by thirdpoints CCW (first thirdpoint is the one that follows first corner). When elements have an interior node, as in the 9-node quadrilateral or 10-node triangle, that node goes last.

Example 27.6. For Model (I) of Figure 27.1(b), which has only one 4-node quadrilateral:

```
ElemNodes={{1,2,4,3}};
```

Example 27.7. For Model (II) of Figure 27.1(c), which has only one 9-node quadrilateral:

```
ElemNodes={{1,3,9,7,2,6,8,4,5}};
```

Example 27.8. For Model (I) of Figure 27.2(b), numbering the elements from top to bottom and from left to right:

```
ElemNodes=Table[{0,0,0,0},{24}];
ElemNodes[[1]]={1,2,9,8};
```

---

[3] Some FEM programs call this the "topology" data.

```
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,2,6}];
ElemNodes[[7]]=ElemNodes[[6]]+{2,2,2,2};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,8,12}];
ElemNodes[[13]]=ElemNodes[[12]]+{2,2,2,2};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,14,18}];
ElemNodes[[19]]=ElemNodes[[18]]+{2,2,2,2};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,20,24}];
```

Example 27.9. For Model (II) of Figure 27.2(c), numbering the elements from top to bottom and from left to right:

```
ElemNodes=Table[{0,0,0,0,0,0,0,0,0},{6}];
ElemNodes[[1]]={1,3,17,15,2,10,16,8,9};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{2,2,2,2,2,2,2,2,2},{e,2,3}];
ElemNodes[[4]]=ElemNodes[[3]]+{10,10,10,10,10,10,10,10,10};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{2,2,2,2,2,2,2,2,2},{e,5,6}];
```

Since this particular mesh only has 6 elements, it would be indeed faster to write down the six nodelists.

§27.3.5. Material Properties

Data structure `ElemMaterials` is a list that provides the constitutive properties of the elements:

$$\boxed{\texttt{ElemMaterials = \{mprop}^{(1)}\texttt{, mprop}^{(2)}\texttt{, ... mprop}^{N_e}\texttt{\}}} \tag{27.5}$$

For a plate element, `mprop` is the stress-strain matrix of elastic moduli (also known as elasticity matrix) arranged as `{{E11,E12,E33},{E12,E22,E23},{E13,E23,E33}}`. Note that although this matrix is symmetric, it must be specified as a full 3×3 matrix. For a bar element, `mprop` is simply the longitudinal elastic modulus.

A common case in practice is that (i) all elements are plates, (ii) the plate material is uniform and isotropic. An isotropic elastic material is specified by the elastic modulus E and Poisson's ratio ν. Then this list can be generated by a single `Table` instruction upon building the elastic ity matrix, as in the example below.

Example 27.10. For all FEM discretizations in Figures 27.1 and 27.2 all elements are plates of the same isotropic material. Suppose that values of the elastic modulus E and Poisson's ratio ν are stored in `Em` and `nu`, respectively, which are typically declared at the beginning of the problem script. Let `numele` give the number of elements. Then the material data is compactly declared by saying

```
Emat=Em/(1-nu^2)*{{1,nu,0},{nu,1,0},{0,0,(1-nu)/2}};
ElemMaterials=Table[Emat,{numele}];
```

§27.3.6. Fabrication Properties

Data structure `ElemFabrications` is a list that provides the fabrication properties of the elements:

$$\boxed{\texttt{ElemFabrications = \{fprop}^{(1)}\texttt{, fprop}^{(2)}\texttt{, ... fprop}^{N_e}\texttt{\}}} \tag{27.6}$$

For a plate element, `fprop` is the thickness h of the plate, assumed constant.[4] For a bar element, `fprop` is the cross section area.

If all elements are plates with the same thickness, this list can be easily generated by a `Table` instruction as in the example below.

Example 27.11. For all FEM discretizations in Figures 27.1 and 27.2 all elements are plates with the same thickness h, which is stored in variable `th`. This is typically declared at the start of the problem script. As before, `numele` has the number of elements. Then the fabrication data is compactly declared by saying

```
ElemFabrications=Table[th,{numele}]
```

§27.3.7. Node Freedom Tags

Data structure `NodeDOFTags` is a list that labels each node DOF as to whether the load or the displacement is specified. The configuration of this list is similar to that of `NodeCoordinates`:

$$\texttt{NodeDOFTags=}\{\,\{\,\texttt{tag}_{x1},\texttt{tag}_{y1}\,\},\{\,\texttt{tag}_{x2},\texttt{tag}_{y2}\,\},\ \ldots\ \{\,\texttt{tag}_{xN},\texttt{tag}_{yN}\,\}\,\} \tag{27.7}$$

The tag value is 0 if the force is specified and 1 if the displacement is specified.

When there are a lot of nodes, often the quickest way to specify this list is to create with a `Table` command that initializes it to all zeros. Then displacement BCs are inserted appropriately, as in the example below.

Example 27.12. For Model (I) in Figure 27.2(a):

```
numnod=Length[NodeCoordinates];
NodeDOFTags=Table[{0,0},{numnod}];    (* create and initialize to zero *)
Do[NodeDOFTags[[n]]={1,0},{n,1,7}];   (* vroller @ nodes 1 through 7 *)
Do[NodeDOFTags[[n]]={0,1},{n,29,35}]; (* hroller @ nodes 29 through 35 *)
```

This scheme works well because typically the number of supported nodes is small compared to the total number.

§27.3.8. Node Freedom Values

Data structure `NodeDOFValues` is a list with the same node by node configuration as `NodeDOFTags`:

$$\texttt{NodeDOFValues=}\{\,\{\,\texttt{value}_{x1},\texttt{value}_{y1}\,\},\{\,\texttt{value}_{x2},\texttt{value}_{y2}\,\},\ \ldots\ \{\,\texttt{value}_{xN},\texttt{value}_{yN}\,\}\,\}$$
$$\tag{27.8}$$

Here `value` is the specified value of the applied node force component if the corresponding tag is zero, and of the prescribed displacement component if the tag is one.

Often most of the entries of (27.8) are zero. If so a quick way to build it is to create it with a `Table` command that initializes it to zero. Then nonzero values are inserted as in the example below.

---

[4] It is possible also to specify a variable thickness by making `fprop` a list that contains the thicknesses at the nodes. Since the variable thickness case is comparatively rare, it will not be described here.

Example 27.13. For the model (I) in Figure 27.2(a) only 3 values (for the y forces on nodes 1, 8 and 15) will be nonzero:

```
numnod=Length[NodeCoordinates];
NodeDOFValues=Table[{0,0},{numnod}]; (* create and initialize to zero *)
NodeDOFValues[[1]]=NodeDOFValues[[15]]={0,37.5};
NodeDOFValues[[8]]={0,75};              (* y nodal loads *)
```

§27.3.9. Processing Options

Array `ProcessOptions` is a list of general processing options that presently contains only the `numer` logical flag. This is normally be set to `True` to specify numeric computations:

```
ProcessOptions={True}
```

§27.3.10. Model Display Utility

Only one graphic display utility is presently provided to show the mesh. Nodes and elements of Model (I) of Figure 27.2(a) may be plotted by saying

```
aspect=6/5;
Plot2DElementsAndNodes[NodeCoordinates,
ElemNodes,aspect, "Plate with circular
hole - 4-node quad model",True,True];
```

Here `aspect` is the plot frame aspect ratio (y dimension over x dimension). The 4th argument is a plot title textstring. The last two `True` argument values specify that node labels and element labels, respectively, be shown. The output of the mesh plot command is shown in Figure 27.3.

§27.3.11. Model Definition Print Utilities

Several print utilities are provided in the plane stress program to print out model definition data in tabular form. They are invoked as follows.

FIGURE 27.3. Mesh plot for Model (I) of Figure 27.2(a).

To print the node coordinates:

```
PrintPlaneStressNodeCoordinates[NodeCoordinates,title,digits];
```

To print the element types and nodes:

```
PrintPlaneStressElementTypeNodes[ElemTypes,EleNodes,title,digits];
```

To print the element materials and fabrications:

```
PrintPlaneStressElementMatFab[ElemMaterials,ElemFabrications,title,digits];
```

To print freedom activity data:

```
PrintPlaneStressFreedomActivity[NodeDOFTags,NodeDOFValues,title,digits];
```

In all cases, `title` is an optional character string to be printed as a title before the table; for example `"Node coordinates"`. To eliminate the title, specify `""` (two quote marks together).

The last argument of the print modules: `digits`, is optional. If set to `{d,f}` it specifies that floating point numbers are to be printed with room for at least d digits, with f digits after the decimal point. If `digits` is specified as a void list: `{ }`, a preset default is used for d and f.
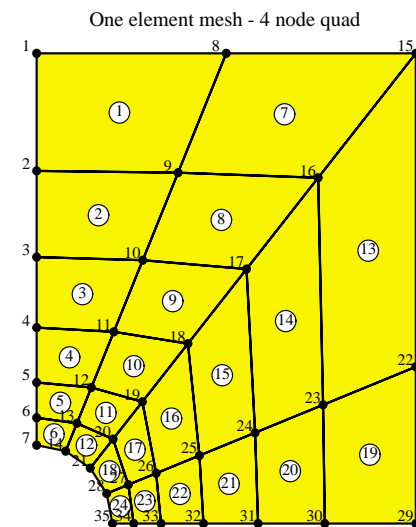
§27.3.12. Model Definition Script Samples

As capstone examples, Figures 27.4 and Figures 27.5 list the preprocessing (model definition) parts of the problem scripts for Model (I) and (II), respectively, of Figure 27.1(b,c).

```
ClearAll[Em,ν,th];
Em=10000; ν=.25;  th=3; aspect=6/5; Nsub=4;
Emat=Em/(1-ν^2)*{{1,ν,0},{ν,1,0},{0,0,(1-ν)/2}};

(*  Define FEM model *)

NodeCoordinates=N[{{0,6},{0,0},{5,6},{5,0}}];
PrintPlaneStressNodeCoordinates[NodeCoordinates,"",{6,4}];
ElemNodes=  {{1,2,4,3}};
numnod=Length[NodeCoordinates]; numele=Length[ElemNodes];
ElemTypes=      Table["Quad4",{numele}];
PrintPlaneStressElementTypeNodes[ElemTypes,ElemNodes,"",{}];
ElemMaterials=  Table[Emat,   {numele}];
ElemFabrications=Table[th,     {numele}];
PrintPlaneStressElementMatFab[ElemMaterials,ElemFabrications,"",{}];
NodeDOFValues=NodeDOFTags=Table[{0,0},{numnod}];
NodeDOFValues[[1]]=NodeDOFValues[[3]]={0,75}; (* nodal loads *)
NodeDOFTags[[1]]={1,0};     (* vroller @ node 1 *)
NodeDOFTags[[2]]={1,1};     (* fixed node 2     *)
NodeDOFTags[[4]]={0,1};     (* hroller @ node 4 *)
PrintPlaneStressFreedomActivity[NodeDOFTags,NodeDOFValues,"",{}];
ProcessOptions={True};
Plot2DElementsAndNodes[NodeCoordinates,ElemNodes,aspect,
    "One element mesh - 4-node quad",True,True];
```

FIGURE 27.4. Model definition part of Model (I) of Figure 27.1(b).

```
ClearAll[Em,ν,th];
Em=10000; ν=.25;  th=3; aspect=6/5; Nsub=4;
Emat=Em/(1-ν^2)*{{1,ν,0},{ν,1,0},{0,0,(1-ν)/2}};

(*  Define FEM model *)

NodeCoordinates=N[{{0,6},{0,3},{0,0},{5/2,6},{5/2,3},
    {5/2,0},{5,6},{5,3},{5,0}}];
PrintPlaneStressNodeCoordinates[NodeCoordinates,"",{6,4}];
ElemNodes=  {{1,3,9,7,2,6,8,4,5}};
numnod=Length[NodeCoordinates]; numele=Length[ElemNodes];
ElemTypes=      Table["Quad9",{numele}];
PrintPlaneStressElementTypeNodes[ElemTypes,ElemNodes,"",{}];
ElemMaterials=  Table[Emat,   {numele}];
ElemFabrications=Table[th,     {numele}];
PrintPlaneStressElementMatFab[ElemMaterials,ElemFabrications,"",{}];
NodeDOFValues=NodeDOFTags=Table[{0,0},{numnod}];
NodeDOFValues[[1]]=NodeDOFValues[[7]]={0,25};
NodeDOFValues[[4]]={0,100};                (* nodal loads *)
NodeDOFTags[[1]]=NodeDOFTags[[2]]={1,0}; (* vroller @ nodes 1,2 *)
NodeDOFTags[[3]]={1,1};                    (* fixed node 3 *)
NodeDOFTags[[6]]=NodeDOFTags[[9]]={0,1}; (* hroller @ nodes 6,9 *)
PrintPlaneStressFreedomActivity[NodeDOFTags,NodeDOFValues,"",{}];
ProcessOptions={True};
Plot2DElementsAndNodes[NodeCoordinates,ElemNodes,aspect,
    "One element mesh - 9-node quad",True,True];
```

FIGURE 27.5. Model definition part of Model (II) of Figure 27.1(c).

```
PlaneStressSolution[nodxyz_,eletyp_,elenod_,elemat_,elefab_,
   nodtag_,nodval_,prcopt_]:= Module[{K,Kmod,f,fmod,u,numer=True,
   noddis,nodfor,nodpnc,nodsig,barele,barfor},
   If [Length[prcopt]>=1, numer=prcopt[[1]]];
   K=PlaneStressMasterStiffness[nodxyz,eletyp,elenod,elemat,elefab,prcopt];
   If [K==Null,Return[Table[Null,{6}]]];
   Kmod=ModifiedMasterStiffness[nodtag,K];
   f=FlatNodePartVector[nodval];
   fmod=ModifiedNodeForces[nodtag,nodval,K,f];
   u=LinearSolve[Kmod,fmod]; If [numer,u=Chop[u]];
   f=K.u;                    If [numer,f=Chop[f]];
   nodfor=NodePartFlatVector[2,f]; noddis=NodePartFlatVector[2,u];
   {nodpnc,nodsig}=PlaneStressPlateStresses[nodxyz,eletyp,elenod,elemat,
               elefab,noddis,prcopt];
   {barele,barfor}=PlaneStressBarForces[nodxyz,eletyp,elenod,elemat,
               elefab,noddis,prcopt];
   ClearAll[K,Kmod];
   Return[{noddis,nodfor,nodpnc,nodsig,barele,barfor}];
];
```

FIGURE 27.6. Module to drive the analysis of the plane stress problem.

§27.4. Processing

Once the model definition is complete, the plane stress analysis is carried out by calling module
LinearSolutionOfPlaneStressModel listed in Figure 27.6. This module is invoked from the
problem driver as

```
{NodeDisplacements,NodeForces,NodePlateCounts,NodePlateStresses,
   ElemBarNumbers,ElemBarForces}= PlaneStressSolution[NodeCoordinates,
   ElemTypes,ElemNodes,ElemMaterials,ElemFabrications,
   NodeDOFTags,NodeDOFValues,ProcessOptions];
```

The module arguments: NodeCoordinates, ElemTypes, ElemNodes , ElemMaterials,
ElemFabrications, NodeDOFTags, NodeDOFValues and ProcessOptions are the data struc-
tures described in the previous section.

The module returns the following:

NodeDisplacements	Computed node displacements, in node-partitioned form.
NodeForces	Recovered node forces including reactions, in node-partitioned form.
NodePlateCounts	For each node, number of plate elements attached to that node. A zero count means that no plate elements are attached to that node.
NodePlateStresses	Averaged stresses at plate nodes with a nonzero NodePlateCounts.
ElemBarNumbers	A list of bar elements if any specified, else an empty list.
ElemBarForces	A list of bar internal forces if any bar elements were specified, else an empty list.

§27.4.1. Processing Tasks

PlaneStressSolution carries out the following tasks. It assembles the free-free master stiffness matrix K by
calling the MET assembler PlaneStressMasterStiffness. This module is listed in Figure 27.7. A study

```
PlaneStressMasterStiffness[nodxyz_,eletyp_,elenod_,elemat_,
  elefab_,prcopt_]:=Module[{numele=Length[elenod],
  numnod=Length[nodxyz],ncoor,type,e,enl,neldof,OKtyp,OKenl,
  i,j,n,ii,jj,eft,Emat,th,numer,Ke,K},
  OKtyp={"Bar2","Bar3","Quad4","Quad4.1","Quad8","Quad8.2","Quad9",
       "Quad9.2","Trig3","Trig6","Trig6.-3","Trig10","Trig10.6"};
  OKenl= {2,3,4,4,8,8,9,9,3,6,6,10,10};
  K=Table[0,{2*numnod},{2*numnod}];  numer=prcopt[[1]];
  For [e=1,e<=numele,e++, type=eletyp[[e]];
      If [!MemberQ[OKtyp,type], Print["Illegal type:",type,
      " element e=",e," Assembly aborted"]; Return[Null] ];
      enl=elenod[[e]]; n=Length[enl]; {{j}}=Position[OKtyp,type];
      If [OKenl[[j]]!=n, Print ["Wrong node list length, element=",e,
          " Assembly aborted"]; Return[Null] ];
      eft=Flatten[Table[{2*enl[[i]]-1,2*enl[[i]]},{i,1,n}]];
      ncoor=Table[nodxyz[[enl[[i]]]],{i,1,n}];
      If [type=="Bar2",     Em=elemat[[e]]; A=elefab[[e]];
          Ke=PlaneBar2Stiffness[ncoor,Em,A,{numer}] ];
      If [type=="Bar3",     Em=elemat[[e]]; A=elefab[[e]];
          Ke=PlaneBar3Stiffness[ncoor,Em,A,{numer}] ];
      If [type=="Quad4",    Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Quad4IsoPMembraneStiffness[ncoor,Emat,th,{numer,2}] ];
      If [type=="Quad4.1",  Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Quad4IsoPMembraneStiffness[ncoor,Emat,th,{numer,1}] ];
      If [type=="Quad8",    Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Quad8IsoPMembraneStiffness[ncoor,Emat,th,{numer,3}] ];
      If [type=="Quad8.2",  Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Quad8IsoPMembraneStiffness[ncoor,Emat,th,{numer,2}] ];
      If [type=="Quad9",    Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Quad9IsoPMembraneStiffness[ncoor,Emat,th,{numer,3}] ];
      If [type=="Quad9.2",  Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Quad9IsoPMembraneStiffness[ncoor,Emat,th,{numer,2}] ];
      If [type=="Trig3",    Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Trig3IsoPMembraneStiffness[ncoor,Emat,th,{numer}] ];
      If [type=="Trig6",    Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Trig6IsoPMembraneStiffness[ncoor,Emat,th,{numer,3}] ];
      If [type=="Trig6.-3", Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Trig6IsoPMembraneStiffness[ncoor,Emat,th,{numer,-3}] ];
      If [type=="Trig10",   Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Trig10IsoPMembraneStiffness[ncoor,Emat,th,{numer,7}] ];
      If [type=="Trig10.6", Emat=elemat[[e]]; th=elefab[[e]];
          Ke=Trig10IsoPMembraneStiffness[ncoor,Emat,th,{numer,6}] ];
      neldof=Length[Ke];
      For [i=1,i<=neldof,i++, ii=eft[[i]];
          For [j=i,j<=neldof,j++, jj=eft[[j]];
              K[[jj,ii]]=K[[ii,jj]]+=Ke[[i,j]] ];
      ];
  ];  Return[K];
];
```

FIGURE 27.7. Plane stress assembler module.

of its code reveals that it handle the multiple element types listed in Table 27.2. The modules that compute the element stiffness matrices have been studied in previous Chapters and need not be listed here.

The displacement BCs are applied by `ModifiedMasterStiffness` and `ModifiedNodeForces`. These are the same modules used in Chapter 21 for the space truss program, and thus need not be described further.

The unknown node displacements u are obtained through the built in `LinearSolve` function, as `u=LinearSolve[Kmod,fmod]`. This function is of course restricted to small systems, typically less than 200 equations (it gets extremely slow for something bigger) but it has the advantages of simplicity. The node

```
PlaneStressPlateStresses[nodxyz_,eletyp_,elenod_,elemat_,
  elefab_,noddis_,prcopt_]:=Module[{numele=Length[elenod],
  numnod=Length[nodxyz],ncoor,type,e,enl,i,k,n,Emat,th,
  numer=True,nodpnc,nodsig}, If [Length[prcopt]>0, numer=prcopt[[1]]];
  nodpnc=Table[0,{numnod}]; nodsig=Table[{0,0,0},{numnod}];
  If [Length[prcopt]>=1, numer=prcopt[[1]]];
  For [e=1,e<=numele,e++, type=eletyp[[e]];
       enl=elenod[[e]]; k=Length[enl];
       ncoor=Table[nodxyz[[enl[[i]]]],{i,1,k}];
       ue=Table[{noddis[[enl[[i]],1]],noddis[[enl[[i]],2]]},{i,1,k}];
       ue=Flatten[ue];
       If [StringTake[type,3]=="Bar", Continue[]];
       If [type=="Quad4",    Emat=elemat[[e]]; th=elefab[[e]];
          sige=Quad4IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Quad4.1",  Emat=elemat[[e]]; th=elefab[[e]];
          sige=Quad4IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Quad8",    Emat=elemat[[e]]; th=elefab[[e]];
          sige=Quad8IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Quad8.2",  Emat=elemat[[e]]; th=elefab[[e]];
          sige=Quad8IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Quad9",    Emat=elemat[[e]]; th=elefab[[e]];
          sige=Quad9IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Quad9.2",  Emat=elemat[[e]]; th=elefab[[e]];
          sige=Quad9IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Trig3",    Emat=elemat[[e]]; th=elefab[[e]];
          sige=Trig3IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Trig6",    Emat=elemat[[e]]; th=elefab[[e]];
          sige=Trig6IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Trig6.-3", Emat=elemat[[e]]; th=elefab[[e]];
          sige=Trig6IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Trig10",   Emat=elemat[[e]]; th=elefab[[e]];
          sige=Trig10IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       If [type=="Trig10.6", Emat=elemat[[e]]; th=elefab[[e]];
          sige=Trig10IsoPMembraneStresses[ncoor,Emat,th,ue,{numer}] ];
       Do [n=enl[[i]]; nodpnc[[n]]++; nodsig[[n]]+=sige[[i]], {i,1,k}];
     ];
  Do [k=nodpnc[[n]]; If [k>1,nodsig[[n]]/=k], {n,1,numnod}];
  If [numer, nodsig=Chop[nodsig]];
  Return[{nodpnc,nodsig}];
];
```

FIGURE 27.8. Module that recovers averaged nodal stresses at plate nodes.

forces including reactions are recovered by the matrix multiplication f = K.u, where K is the unmodified master stiffness.

Finally, plate stresses and bar internal forces are recovered by the modules PlaneStressPlateStresses and PlaneStressBarForces, respectively. The former is listed in Figure 27.8. This computation is actually part of the postprocessing stage. It is not described here since stress recovery is treated in more detail in a subsequent Chapter.

The bar force recovery module is not described here as it has not been used in assigned problems so far.

§27.5. Postprocessing

As noted above, module PlaneStressSolution carries out preprocessing tasks: recover node forces, plate stresses and bar forces. But those are not under control of the user. Here we describe result printing and plotting activities that can be specified in the problem script.

§27.5.1. Result Print Utilities

Several utilities are provided in the plane stress program to print solution data in tabular form. They are invoked as follows.

To print computed node displacements:

```
PrintPlaneStressNodeDisplacements[NodeDisplacements,title,digits];
```

To print receoverd node forces including reactions:

```
PrintPlaneStressNodeForces[NodeForces,title,digits];
```

To print plate node stresses:

```
PrintPlaneStressPlateNodeStresses[NodePlateCounts,NodePlateStresses,title,digits];
```

To print node displacements, node forces and node plate stresses in one table:

```
PrintPlaneStressSolution[NodeDisplacements,NodeForces,
                         NodePlateCounts,NodePlateStresses,title,digits];
```

No utility is provided to print bar forces, as problems involving plates and bars had not been assigned.

In all utilities listed above, `title` is an optional character string to be printed as a title before the table; for example `"Node displacements for plate with a hole"`. To eliminate the title, specify `""` (two quote marks together).

The last argument of the print modules: `digits`, is optional. If set to `{d,f}` it specifies that floating point numbers are to be printed with room for at least d digits, with f digits after the decimal point. If `digits` is specified as a void list: `{ }`, a preset default is used for d and f.

§27.5.2. Displacement Field Contour Plots

Contour plots of displacement components u_x and u_y (interpolated over elements from the computed node displacements) may be produced. Displacement magnitudes are shown using a internally set color scheme based on "hue interpolation", in which white means zero. Plots can be obtained using a script typified by

```
ux=Table[NodeDisplacements[[n,1]],{n,numnod}];
uy=Table[NodeDisplacements[[n,2]],{n,numnod}];
{uxmax,uymax}=Abs[{Max[ux],Max[uy]}];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,ux,
        uxmax,Nsub,aspect,"Displacement component ux"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,uy,
        uymax,Nsub,aspect,"Displacement component uy"];
```

The third argument of `ContourPlotNodeFuncOver2DMesh` is the function to be plotted, specified at nodes. A function maximum (in absolute value) is supplied as fourth argument. The reason for supplying this from the outside is that in many cases it is convenient to alter the actual maximum for zooming or animation purposes.

As for the other arguments, `Nsub` is the number of element subdivisions in each direction when breaking down the element area into plot polygons. Typically `Nsub` is set to 4 at the start of the script and is the same for all plots of a script. Plot smoothness in terms of color grading increases with `Nsub`, but plot time grows as `Nsub`-squared. So an appropriate tradeoff is to use a high `Nsub`, say 8, for coarse meshes containing few elements whereas for finer meshes a value of 4 or 2 may work fine. The `aspect` argument specifies the ratio between the *y* (vertical) and *x* (horizontal) dimensions of the plot frame, and is usaully defined at the script start as a symbol so it is the same for all plots. The last argument is a plot title.
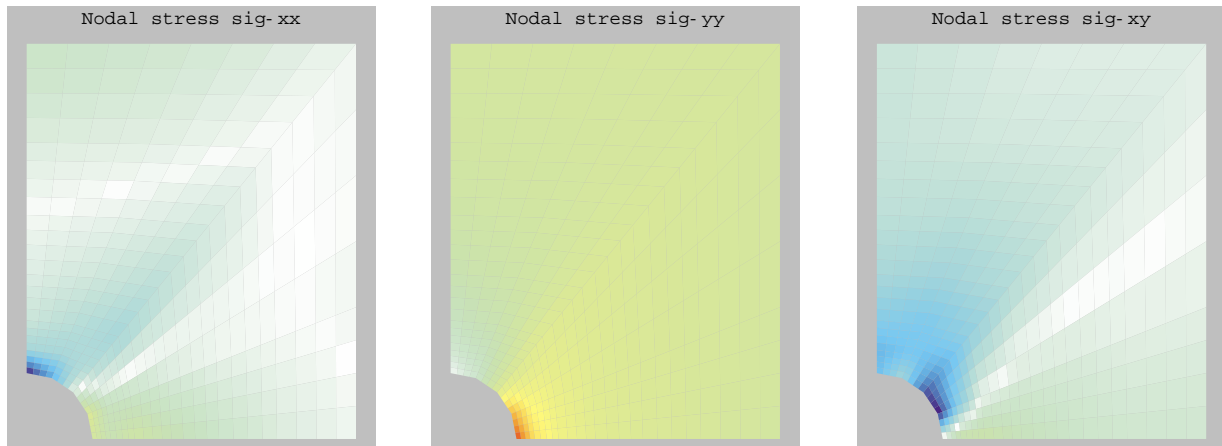
FIGURE 27.9. Stress component contour plots for rectangular plate with
a circular central hole, Model (I) of Figure 27.2(b).

§27.5.3. Stress Field Contour Plots

Contour plots of stress components σ_{xx}, σ_{yy} and σ_{xy} (interpolated over elements from the computed node displacements) may be produced. Stress magnitudes are shown using a internally set color scheme based on "hue interpolation", in which white means zero. Plots can be obtained using a script typified by

```
sxx=Table[NodePlateStresses[[n,1]],{n,numnod}];
syy=Table[NodePlateStresses[[n,2]],{n,numnod}];
sxy=Table[NodePlateStresses[[n,3]],{n,numnod}];
{sxxmax,syymax,sxymax}=Abs[{Max[sxx],Max[syy],Max[sxy]}];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,
        sxx,sxxmax,Nsub,aspect,"Nodal stress sig-xx"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,
        syy,syymax,Nsub,aspect,"Nodal stress sig-yy"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,
        sxy,sxymax,Nsub,aspect,"Nodal stress sig-xy"];
```

For a description of `ContourPlotNodeFuncOver2DMesh` arguments, see the previous subsection.

As an example, stress component contour plots for Model (I) of the rectangular plate with a circular central hole are shown in Figure 27.9.

Remark 27.1. Sometimes it is useful to show contour plots of principal stresses, Von Mises stresses, maximum shears, etc. To do that, the appropriate function should be constructed first from the Cartesian components available in `NodePlateStresses`, and then the plot utility called.

§27.5.4. Animation

Occasionally it is useful to animate results such as displacements or stress fields when a load changes as a function of a time-like parameter t.

This can be done by performing a sequence of analyses in a `For` loop. Such sequence produces a series of plot frames which may be animated by doubly clicking on one of them. The speed of the animation may be controlled by pressing on the righmost buttons at the bottom of the *Mathematica* window. The second button from the left, if pressed, changes the display sequence to back and forth motion.

```
ClearAll[Em,ν,th,aspect,Nsub];
Em=10000; ν=.25;  th=3; aspect=6/5; Nsub=4;
Emat=Em/(1-ν^2)*{{1,ν,0},{ν,1,0},{0,0,(1-ν)/2}};

(*  Define FEM model *)

s={1,0.70,0.48,0.30,0.16,0.07,0.0};
xy1={0,6}; xy7={0,1}; xy8={2.5,6}; xy14={Cos[3*Pi/8],Sin[3*Pi/8]};
xy8={2.5,6}; xy21={Cos[Pi/4],Sin[Pi/4]}; xy15={5,6};
xy22={5,2};  xy28={Cos[Pi/8],Sin[Pi/8]}; xy29={5,0}; xy35={1,0};
NodeCoordinates=Table[{0,0},{35}];
Do[NodeCoordinates[[n]]=N[s[[n]]    *xy1+(1-s[[n]])     *xy7],{n,1,7}];
Do[NodeCoordinates[[n]]=N[s[[n-7]] *xy8+(1-s[[n-7]])   *xy14],{n,8,14}];
Do[NodeCoordinates[[n]]=N[s[[n-14]]*xy15+(1-s[[n-14]])*xy21],{n,15,21}];
Do[NodeCoordinates[[n]]=N[s[[n-21]]*xy22+(1-s[[n-21]])*xy28],{n,22,28}];
Do[NodeCoordinates[[n]]=N[s[[n-28]]*xy29+(1-s[[n-28]])*xy35],{n,29,35}];
PrintPlaneStressNodeCoordinates[NodeCoordinates,"",{6,4}];
ElemNodes=Table[{0,0,0,0},{24}];
ElemNodes[[1]]={1,2,9,8};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,2,6}];
ElemNodes[[7]]=ElemNodes[[6]]+{2,2,2,2};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,8,12}];
ElemNodes[[13]]=ElemNodes[[12]]+{2,2,2,2};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,14,18}];
ElemNodes[[19]]=ElemNodes[[18]]+{2,2,2,2};
Do [ElemNodes[[e]]=ElemNodes[[e-1]]+{1,1,1,1},{e,20,24}];
numnod=Length[NodeCoordinates]; numele=Length[ElemNodes];

ElemTypes=       Table["Quad4",{numele}];
PrintPlaneStressElementTypeNodes[ElemTypes,ElemNodes,"",{}];
ElemMaterials=   Table[Emat,   {numele}];
ElemFabrications=Table[th,     {numele}];
(*PrintPlaneStressElementMatFab[ElemMaterials,ElemFabrications,"",{}];*)
NodeDOFValues=NodeDOFTags=Table[{0,0},{numnod}];
NodeDOFValues[[1]]=NodeDOFValues[[15]]={0,37.5};
NodeDOFValues[[8]]={0,75}; (* nodal loads *)
Do[NodeDOFTags[[n]]={1,0},{n,1,7}];    (* vroller @ nodes 1-7 *)
Do[NodeDOFTags[[n]]={0,1},{n,29,35}]; (* hroller @ node 4 *)
PrintPlaneStressFreedomActivity[NodeDOFTags,NodeDOFValues,"",{}];
ProcessOptions={True};
Plot2DElementsAndNodes[NodeCoordinates,ElemNodes,aspect,
    "One element mesh - 4-node quad",True,True];
```

FIGURE 27.10. Preprocessing (model definition) script for problem of Figure 27.2(b).

§**27.6. A Complete Problem Script Cell**

A complete problem script cell (which is Cell 13 in the plane stress Notebook placed in the web index of this Chapter) is listed in Figures 27.10 through 27.12. This driver cell does Model (I) of the plate with hole problem of Figure 27.2(b), which uses 4-node quadrilateral elements.

The script is divided into 3 parts for convenience. Figure 27.10 lists the model definition script followed by a mesh plot command. Figure 27.11 shows the call to `PlaneStressSolution` analysis driver and the call to get printout of node displacements, forces and stresses. Finally, Figure 27.12 shows commands to produce contour plots of displacements (skipped) and stresses.

Other driver cell examples may be studied in the `PlaneStress.nb` Notebook posted on the course web site. It can be observed that the processing and postprocessing scripts are largely the same.

```
(*  Solve problem and print results *)

{NodeDisplacements,NodeForces,NodePlateCounts,NodePlateStresses,
    ElemBarNumbers,ElemBarForces}= PlaneStressSolution[
    NodeCoordinates,ElemTypes,ElemNodes,
    ElemMaterials,ElemFabrications,
    NodeDOFTags,NodeDOFValues,ProcessOptions];

PrintPlaneStressSolution[NodeDisplacements,NodeForces,NodePlateCounts,
    NodePlateStresses,"Computed Solution:",{}];
```

FIGURE 27.11. Processing script and solution print for problem of Figure 27.2(b).

```
(*  Plot Displacement Components Distribution - skipped *)

(* ux=Table[NodeDisplacements[[n,1]],{n,numnod}];
uy=Table[NodeDisplacements[[n,2]],{n,numnod}];
{uxmax,uymax}=Abs[{Max[ux],Max[uy]}];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,ux,
      uxmax,Nsub,aspect,"Displacement component ux"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,uy,
      uymax,Nsub,aspect,"Displacement component uy"]; *)

(*  Plot Averaged Nodal Stresses Distribution *)

sxx=Table[NodePlateStresses[[n,1]],{n,numnod}];
syy=Table[NodePlateStresses[[n,2]],{n,numnod}];
sxy=Table[NodePlateStresses[[n,3]],{n,numnod}];
{sxxmax,syymax,sxymax}={Max[Abs[sxx]],Max[Abs[syy]],Max[Abs[sxy]]};
Print["sxxmax,syymax,sxymax=",{sxxmax,syymax,sxymax}];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,
      sxx,sxxmax,Nsub,aspect,"Nodal stress sig-xx"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,
      syy,syymax,Nsub,aspect,"Nodal stress sig-yy"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodes,
      sxy,sxymax,Nsub,aspect,"Nodal stress sig-xy"];
```

FIGURE 27.12. Result plotting script for problem of Figure 27.2(b). Note that displacement
contour plots have been skipped by commenting out the code.

What changes is the model definition script portion, which often benefits from ad-hoc node and element generation constructs.

Notes and Bibliography

Few FEM books show a complete program. More common is the display of snipets of code. These are left dangling chapter after chapter, with no attempt at coherent interfacing. This historical problem comes from early reliance on low-level languages such as Fortran. Even the simplest program may run to throusands of code lines. At 50 lines/page that becomes difficult to display snugly in a textbook while providing a running commentary.

Another ages-old problem is plotting. Languages such as C or Fortran do not include plotting libraries for the simple reason that low-level universal plotting standards never existed. The situation changed when widely used high-level languages like *Matlab* or *Mathematica* appeared. The language engine provides the necessary fit to available computer hardware, concealing system dependent details. Thus plotting scripts are transportable.