13. Programa Tensión Plana en Mathematica.

En esta sección, en primer lugar se plantean los pasos a seguir para realizar un análisis por elementos finitos, tal y como hay que proceder tanto en programas comerciales, como en los programas implementados en Mathematica comentados a lo largo de las lecciones. Se comenta la estructura de datos implementada para la definición del problema, indicando que se utilizan, convenientemente agrupados, todos los módulos definidos en Mathematica. Para mostrar cómo hay que preparar los datos para poder utilizar con éxito el conjunto de módulos implementados, se presentan dos problemas a resolver: (1) placa rectangular bajo las condiciones de tensión plana sometida a una carga uníaxial uniforme, definida mediante un solo elemento, que simula una cuarta parte de la misma, debido a la existencia de simetría de cargas y de geometría, considerándose en primer lugar un elemento cuadrilátero de 4 nodos, y en segundo lugar un elemento cuadrilátero de 9 nodos; y (2) la misma placa en las mismas condiciones y sometida al mismo tipo de cargas, pero que posee un agujero central, analizada por la misma razón anterior, en una cuarta parte únicamente, definida mediante un conjunto de elementos generados a mano, de los dos tipos utilizados en el problema anterior.

El proceso de PREPARACION DE DATOS consta de las siguientes fases: (1) definición de coordenadas nodales; (2) selección del tipo de elemento, eligiendo uno del conjunto de los disponibles: (a) triangulo lineal de 3 nodos; (b) triangulo cuadrático de 6 nodos; (c) triángulo cúbico de 10 nodos; (d) cuadrilátero bilineal de 4 nodos; y (e) cuadrilátero bicuadrático de 9 nodos; (3) definición de la conectividad de elementos, es decir, las secuencias de nodos que definen los elementos; (4) definición de las propiedades del material de cada elemento; (5) definición de las propiedades geométricas de cada elemento, es decir, de la función espesor; (6) definición de los indicadores de libertad en cada nodo, es decir, en cada nodo se indica si va a estar restringido en desplazamiento, o va a tener una fuerza aplicada, según cada dirección coordenada; (7) definición de los valores de libertad en cada nodo, es decir, del desplazamiento o la fuerza a la que está sometido, en cada dirección coordenada; (8) indicación de procesado, es decir, si se pretende utilizar números en coma flotante, o números enteros; y (9) relación de aspecto para la visualización de la malla, pues se dispone de módulos en Mathematica que permiten visualizar la malla.

El PROCESO DE SOLUCION del problema consta de los siguientes fases: (1) Ensamblado de la Matriz de Rigidez Global, utilizándose un ejemplo de dos triángulos para mostrar el proceso que tiene lugar; (2) Aplicación de las condiciones de contorno y cargas, utilizándose el mismo ejemplo para mostrar el proceso; y (3) Obtención de los desplazamientos nodales y de las fuerzas de reacción en los nodos restringidos. Para mostrar con detalle el proceso de solución comentado se utiliza el primero de los ejemplos.

El POSPROCESADO DEL MODELO consiste en las siguientes fases: (1) Obtención de las tensiones nodales, comentándose el funcionamiento del módulo de Mathematica correspondiente; (2) Visualización de los desplazamientos nodales; y (3) Visualización de las tensiones. Estas dos últimas fases utilizan módulos creados adrede.

Todo este desarrollo figura perfectamente explicado en el Tema 27 del Curso Introductorio al Método de los Elementos Finitos que se cursa en la Universidad de Colorado en Boulder, bajo la dirección del Prof. Carlos A. Felippa, y por ello lo proporcionamos completo en esta sección.

CHAPTER 27. Un Programa de EF para el Problema de Tensión Plana.
Carlos A. Felippa.

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.

§27.2. ANALYSIS STAGES

As in all FEM programs, the analysis of plane stress problems by the Direct Stiffness Method involves three major stages: (I) preprocessing or model definition, (II) processing, and (III) post-processing.

The preprocessing portion of the plane stress analysis is done by the driver program, which directly sets the data structures. These are similar to those outlined in Chapter 22 for frame/truss structures:

I.1 Model definition by direct setting of the data structures.

I.2 Plot of the FEM mesh, including nodes and element labels.

The processing stage involves three steps:

II.1 Assembly of the master stiffness matrix, with a subordinate element stiffness module.

II.2 Application of displacement BC. This is done by the equation modification method that keeps the same number and arrangement of degrees of freedom.

II.3 Solution of the modified equations for displacements. The built in *Mathematica* function `LinearSolve` is used for this task.

Upon executing the processing steps, the nodal displacement solution is available. The following postprocessing steps follow:

III.1 Recovery of forces including reactions, done through the built-in matrix multiplication.

III.2 Computation of element stresses and interelement averaging to get nodal stresses.

III.3 Contour plotting of displacement and stress fields.

§27.3. MODEL DEFINITION

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

Model definition	Geometry data: node coordinates Element data: type, connectivity, material and fabrication Degree of freedom data: forces and support BCs

Note that 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: tag and value. In addition there are miscellaneous process options, which 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`, `ElemNodeLists`, `ElemMaterial`, `ElemFabrication`, `FreedomTags`, `FreedomValues` and `ProcessingOptions`

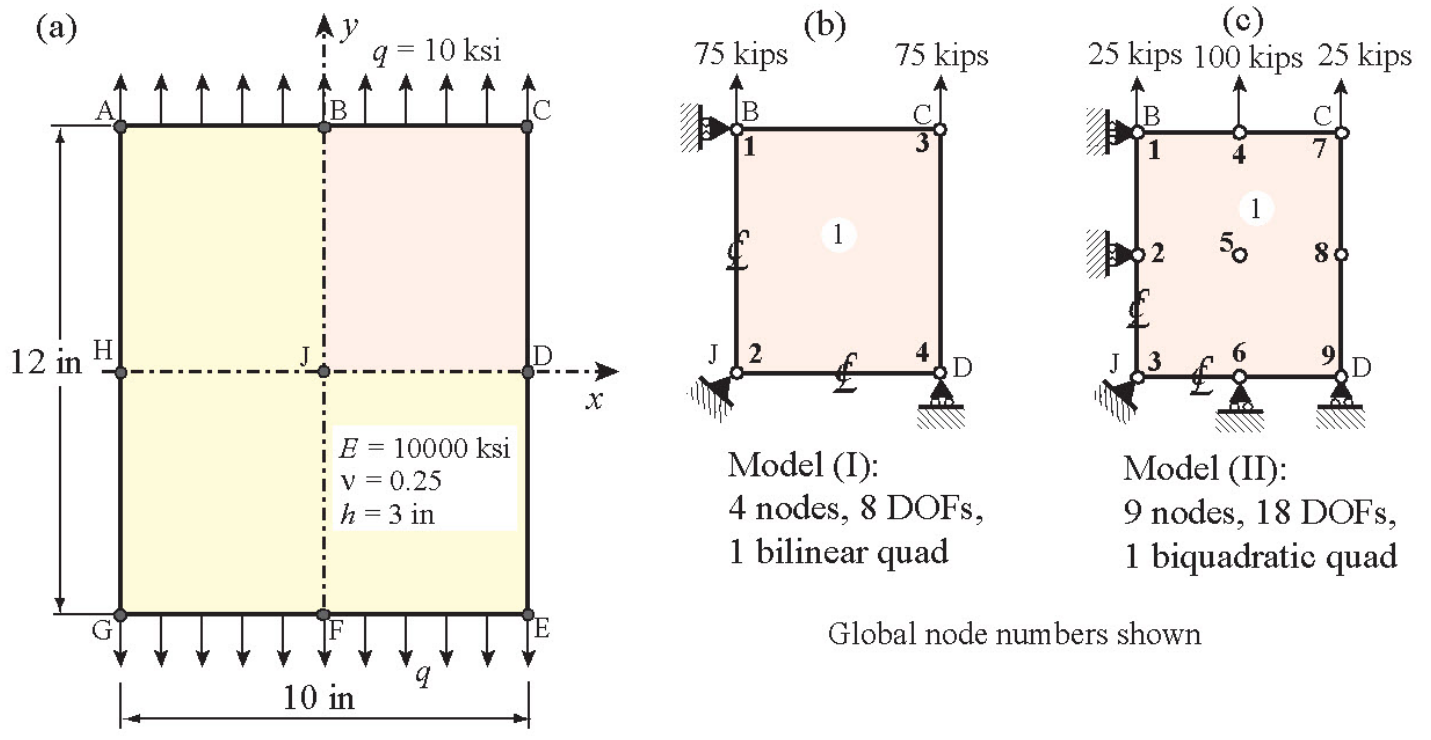


Figure 27.1. Rectangular plate under uniaxial loading, and two one-element FEM discretizations of quadrant BCDJ.

These data sets are described in the following subsections with reference to the problems and discretizations of Figures 27.1 to 27.3.

§27.3.1. Demo Problems

Figure 27.1 is a rectangular plate in plane stress under uniform uniaxial loading. Its analytical solution is $\sigma_{yy} = q$, $\sigma_{xx} = \sigma_{xy} = 0$, $u_y = qy/E$, $u_x = -\nu qx/E$. This problem should be solved exactly by *any* finite element mesh. In particular, the two one-element models shown on the right of that figure.

A similar but more complicated problem is shown in Figure 27.2: the axially-loaded rectangular plate dimensioned and loaded as that of Figure 27.1, but now with a central circular hole. This problem is defined in Figure 27.2. A FEM solution is to be obtained using the two quadrilateral element models (4-node and 9-node) depicted in Figure 27.3. The main result sought is the stress concentration factor on the hole boundary.

§27.3.2. Node Coordinates

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

$$\texttt{NodeCoordinates} = \{\{x_1, y_1\}, \{x_2, y_2\}, \ \ldots \ \{x_N, y_N\}\} \tag{27.1}$$

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

Example 1. For Model (I) of Figure 27.1:

```
NodeCoordinates = N[{{0,6},{0,0},{5,6},{5,0}}];
```

Example 2. For Model (II) of Figure 27.1:

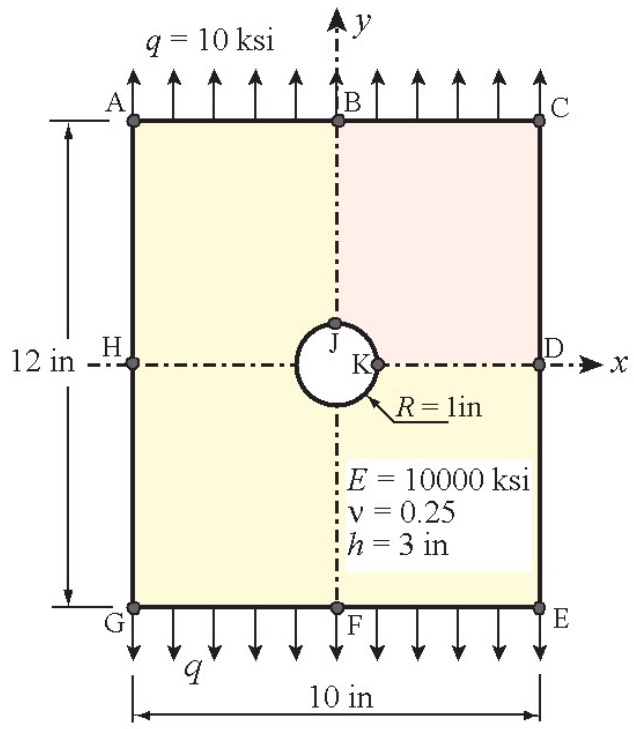


Figure 27.2. Rectangular plate with central circular hole.

```
NodeCoordinates = N[{{0,6},{0,3},{0,0},{5/2,6},{5/2,3},
                     {5/2,0},{5,6},{5,3},{5,0}}];
```

Example 3. For Model (I) of Figure 27.3, using a bit of coordinate generation along lines:

```
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}];
```

The result of this generation is that some of the interior nodes are not in the same positions as sketched in Figure 27.3, but this slight change hardly affects the results.

§27.3.3. Element Type

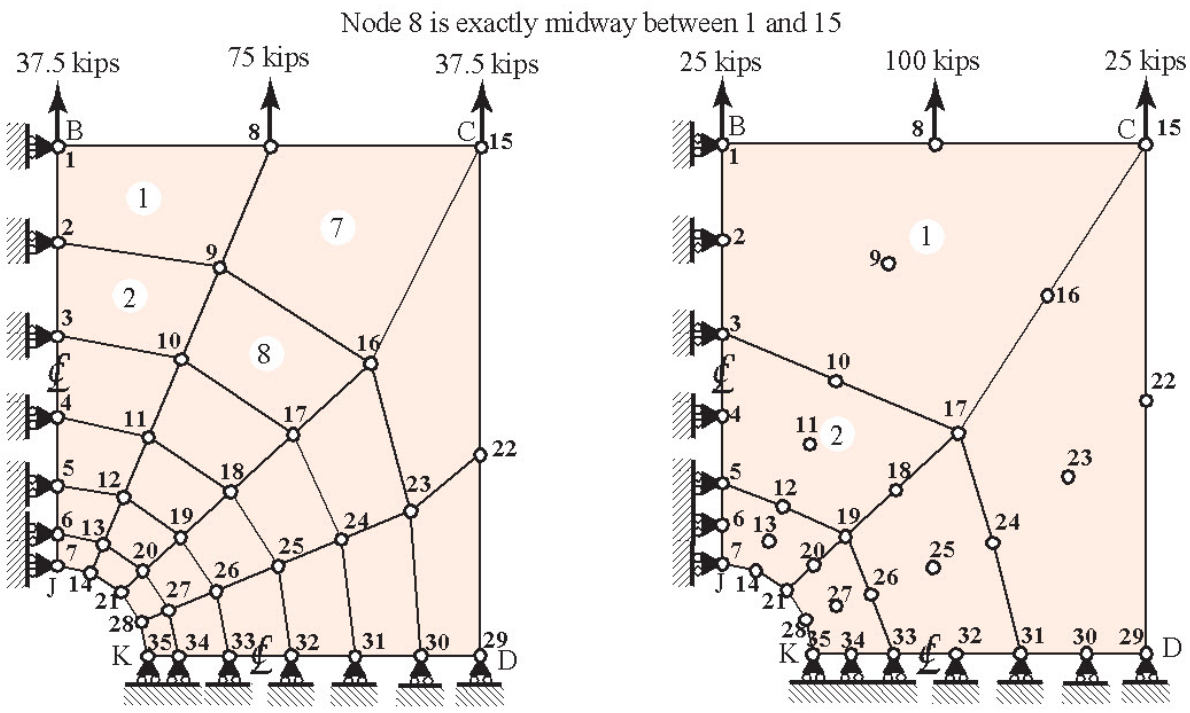
Element type is a label that specifies the type of element to be used.

$$\texttt{ElemTypes} = \{\{\texttt{etyp}^{(1)}\},\ \{\texttt{etyp}^{(2)}\},\ \ldots\ \{\texttt{etyp}^{(N_e)}\}\} \tag{27.2}$$

Here `etyp`(e) is the type descriptor of the e-th element specified as a character string. Allowable types are listed in Table 27.1.

For example, for Model (I) in Figure 27.3:

```
ElemTypes=Table[{"Quad4"},{numele}];
```



Model (I): 35 nodes, 70 DOFs, 24 bilinear quads

Model (II): 35 nodes, 70 DOFs, 6 biquadratic quads

Figure 27.3. Two FEM discretizations of quadrant BCDKJ of the plate of Figure 27.2. Only a few element numbers are shown to reduce clutter.

and for Model (II):

```
ElemTypes=Table[{"Quad9"},{numele}];
```

Here `numele` is the number of elements. This can be extracted as `numele=Length[ElemNodeLists]`, where `ElemNodeLists` is defined below.

Table 27.1 Element Type Descriptors

Identifier	Plane Stress Model
`"Trig3"`	3-node linear triangle
`"Trig6"`	6-node quadratic triangle
`"Trig10"`	10-node cubic triangle
`"Quad4"`	4-node blinear quad
`"Quad9"`	9-node biquadratic quad

§27.3.4. Element Connectivity

Element connectivity information specifies how the elements are connected.[1] This information is stored in `ElemNodeLists`, which is a list of element nodelists:

$$\texttt{ElemNodeLists} = \{\{\texttt{eNL}^{(1)}\},\ \{\texttt{eNL}^{(2)}\},\ \ldots\ \{\texttt{enL}^{(N_e)}\}\} \tag{27.3}$$

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

where $\texttt{eNL}^{(e)}$ denotes the lists of nodes of the element e (given by nglobal node numbers) and N_e is the total number of elements.

Element boundaries must be traversed counterclockwise but you can start at any corner. Numbering elements with midnodes requires more care: first list corners counterclockwise, followed by midpoints (first midpoint is the one that follows first corner when going counterclockwise). When elements have an interior node, as in the 9-node biquadratic quadrilateral, that node goes last.

Example 1. For Model (I) of Figure 27.1, which has only one element:

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

Example 2. For Model (II) of Figure 27.1, which has only one element:

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

Example 3. For Model (I) of Figure 27.3, numbering the elements from top to bottom and from left to right:

```
ElemNodeLists=Table[{0,0,0,0},{24}];
ElemNodeLists[[1]]={1,2,9,8};
Do [ElemNodeLists[[e]]=ElemNodeLists[[e-1]]+{1,1,1,1},{e,2,6}];
ElemNodeLists[[7]]=ElemNodeLists[[6]]+{2,2,2,2};
Do [ElemNodeLists[[e]]=ElemNodeLists[[e-1]]+{1,1,1,1},{e,8,12}];
ElemNodeLists[[13]]=ElemNodeLists[[12]]+{2,2,2,2};
Do [ElemNodeLists[[e]]=ElemNodeLists[[e-1]]+{1,1,1,1},{e,14,18}];
ElemNodeLists[[19]]=ElemNodeLists[[18]]+{2,2,2,2};
Do [ElemNodeLists[[e]]=ElemNodeLists[[e-1]]+{1,1,1,1},{e,20,24}];
```

Example 4. For Model (II) of Figure 27.3, numbering the elements from top to bottom and from left to right:

```
ElemNodeLists=Table[{0,0,0,0,0,0,0,0,0},{6}];
ElemNodeLists[[1]]={1,3,17,15,2,10,16,8,9};
Do [ElemNodeLists[[e]]=ElemNodeLists[[e-1]]+{2,2,2,2,2,2,2,2,2},{e,2,3}];
ElemNodeLists[[4]]=ElemNodeLists[[3]]+{10,10,10,10,10,10,10,10,10};
Do [ElemNodeLists[[e]]=ElemNodeLists[[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 `ElemMaterial` lists the elastic modulus E and the Poisson's ratio ν for each element:

$$\texttt{ElemMaterial} = \{\{E^{(1)}, \nu^{(1)}\},\ \{E^{(2)}, \nu^{(2)}\},\ \ldots\ \{E^{(N_e)}, \nu^{(N_e)}\}\} \tag{27.4}$$

In the common case in which all elements have the same E and ν, this list can be easily generated by a `Table` instruction.

for all models shown in Figures 27.1 and 27.3,

```
ElemMaterial=Table[{Em,nu},{numele}],
```

in which `numele=Length[ElemeNodeLists]` is the number of elements (24 there) and the values of `Em` and `nu` are typically declared separately.

§27.3.6. Fabrication Properties

`ElemMaterial` lists the plate thickness for each element:

$$\texttt{ElemFabrication} = \{\{h(1)\},\ \{h^{(2)}\},\ \ldots\ \{h^{(N_e)}\}\} \tag{27.5}$$

where N_e is the number of elements.

If all elements have the same thickness, this list can be easily generated by a `Table` instruction. For example, for the model of Figure 27.3, `ElemFabrication=Table[th,{numele}]`, Here `numele` is the number of elements (24 in that case; this can be extracted as the `Length` of `ElemNodeList`) and the value of `th=3` is set at the start of the input cell.

§27.3.7. Freedom Tags

`FreedomTags` labels each node degree of freedom as to whether the load or the displacement is specified. The configuration of this list is similar to that of `NodeCoordinates`:

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

The tag value is 0 if the force is specified and 1 if the displacement is specified. When there are a lot of nodes, the quickest way to specify this list is to start from all zeros, and then insert the boundary conditions appropriately. For example, for the Model (I) of Figure 27.3:

```
numnod=Length[NodeCoordinates];
FreedomTags=Table[{0,0},{numnod}];     (* initialize and reserve space *)
Do[FreedomTags[[n]]={1,0},{n,1,7}];    (* vroller @ nodes 1 through 7 *)
Do[FreedomTags[[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. Freedom Values

`FreedomValues` has the same node by node configuration as `FreedomTags`. It lists the specified values of the applied node force component if the corresponding tag is zero, and of the prescribed displacement component if the tag is one. Typically most of the list entries are zero. For example, in the model (I) of Figure 27.3 only 3 values (for the *y* forces on nodes 1, 8 and 15) will be nonzero:

```
numnod=Length[NodeCoordinates];
FreedomValues=Table[{0,0},{numnod}]; (* initialize and reserve space *)
FreedomValues[[1]]=FreedomValues[[15]]={0,37.5};
FreedomValues[[8]]={0,75};           (* y nodal loads *)
```

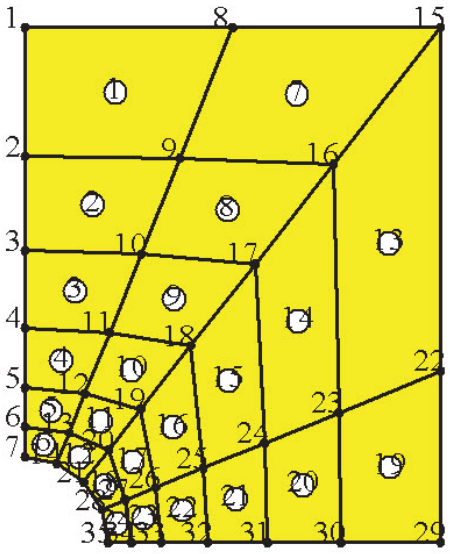


Figure 27.4. Mesh plot showing node and element numbers for Model (I) of Figure 27.3.

§27.3.9. Processing Options

Array `ProcessingOptions` should normally set as follows:

```
ProcessingOptions={True};
```

This specifies floating point numerical computations.

§27.3.10. Mesh Display

Nodes and elements of Model (I) of Figure 27.3 may be plotted by the following statement:

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

Here `aspect` is the plot frame aspect ratio (y dimension over x dimension), and the last two `True` argument values specify that node labels and element labels, respectively, be shown. The output is shown in Figure 27.4.

```
MembraneSolution[nodcoor_,eletyp_,elenod_,elemat_,
  elefab_,eleopt_,doftag_,dofval_]:= Module[{K,Kmod,u,f,sig,j,n,ns,
  supdof,supval,numnod=Length[nodcoor],numele=Length[elenod]},
  u=f=sig={};
  K=MembraneMasterStiffness[nodcoor,
          eletyp,elenod,elemat,elefab,eleopt];  K=N[K];
  ns=0; Do [Do [If[doftag[[n,j]]>0,ns++],{j,1,2}],{n,1,numnod}];
  supdof=supval=Table[0,{ns}];
  k=0; Do [Do [If[doftag[[n,j]]>0,k++;supdof[[k]]=2*(n-1)+j;
         supval[[k]]=dofval[[n,j]]],{j,1,2}],{n,1,numnod}];
  f=ModifiedNodeForces[supdof,supval,K,Flatten[dofval]];
  Kmod=ModifiedMasterStiffness[supdof,K];
  u=Simplify[Inverse[Kmod].f]; u=Chop[u];
  f=Simplify[K.u]; f=Chop[f];
  sig=MembraneNodalStresses[nodcoor,
        eletyp,elenod,elemat,elefab,eleopt,u];
  sig=Chop[sig];
  Return[{u,f,sig}];
];
```

Figure 27.5. Module to drive the analysis of the plane stress problem.

§27.4. PROCESSING

The static solution is carried out by calling module `LinearSolutionOfPlaneStressModel` shown in Figure 27.5. The function call is

```
{u,f,sig}=MembraneSolution[NodeCoordinates,ElemTypes,
          ElemNodeLists,ElemMaterial,ElemFabrication,
          ProcessingOptions,FreedomTags,FreedomValues];
```

The function begins by assembling the free-free master stiffness matrix `K` by calling the module `MembraneMasterStiffness`. this module is listed in Figure 27.5. As a study of its code reveals, it can handle the five element types described in the previous section. The modules that compute the element stiffness matrices have been studied in previous Chapters and need not be listed here.

The displacement boundary conditions are applied by modules `ModifiedmasterStiffness` and `ModifiedNodeForces`, which return the modified stiffness matrix $\hat{\mathbf{K}}$ and the modified force vector $\hat{\mathbf{f}}$ in `Khat` and `fnat`, respectively. The logic of these modules has been explained in Chapter 22 and need not be described again here.

The unknown node displacements `u` are then obtained through the built in `LinearSolve` function, as `u=LinearSolve[Khat,fhat]`. This solution strategy is of course restricted to very small systems, but it has the advantages of simplicity.

The function returns arrays `u`, `f` and `p`, which are lists of length 12, 12 and 13, respectively. Array `u` contains the computed node displacements ordered $u_{x1} < u_{y1}, u_{x2}, \ldots u_{y8}$. Array `f` contains the node forces recovered from $\mathbf{f} = \mathbf{K}\mathbf{u}$; this includes the reactions f_{x1}, f_{y1} and f_{y8}.

Finally, array `sig` contains the nodal stresses σ_{xx}, σ_{yy}, σ_{xy} at each node, recovered from the displacement solution. This computation is driven by module `MembraneNodeStresses`, which is

```
MembraneMasterStiffness[nodcoor_,eletyp_,elenod_,
  elemat_,elefab_,eleopt_]:=
  Module[{numele=Length[elenod],numnod=Length[nodcoor],numer,
   ne,eNL,eftab,neldof,i,n,Em,ν,Emat,th,ncoor,Ke,K},
  K=Table[0,{2*numnod},{2*numnod}];  numer=eleopt[[1]];
  For [ne=1,ne<=numele,ne++,
       {type}=eletyp[[ne]];
       If [type!="Trig3"&&type!="Trig6"&&type!="Trig10"&&
           type!="Quad4"&&type!="Quad9",
           Print["Illegal element type,ne=",ne]; Return[Null]];
       eNL=elenod[[ne]]; n=Length[eNL];
       eftab=Flatten[Table[{2*eNL[[i]]-1,2*eNL[[i]]},{i,n}]];
       ncoor=Table[nodcoor[[eNL[[i]]]],{i,n}];
       {Em,ν}=elemat[[ne]];
       Emat=Em/(1-ν^2)*{{1,ν,0},{ν,1,0},{0,0,(1-ν)/2}};
       {th}=elefab[[ne]];
       If [type=="Trig3", Ke=Trig3IsoPMembraneStiffness[ncoor,
          {Emat,0,0},{th},{numer}] ];
       If [type=="Quad4", Ke=Quad4IsoPMembraneStiffness[ncoor,
          {Emat,0,0},{th},{numer,2}] ];
       If [type=="Trig6", Ke=Trig6IsoPMembraneStiffness[ncoor,
          {Emat,0,0},{th},{numer,3}] ];
       If [type=="Quad9", Ke=Quad9IsoPMembraneStiffness[ncoor,
          {Emat,0,0},{th},{numer,3}] ];
       If [type=="Trig10",Ke=Trig10IsoPMembraneStiffness[ncoor,
          {Emat,0,0},{th},{numer,3}] ];
       neldof=Length[Ke];
       For [i=1,i<=neldof,i++,ii=eftab[[i]];
         For [j=i,j<=neldof,j++,jj=eftab[[j]];
             K[[jj,ii]]=K[[ii,jj]]+=Ke[[i,j]]
         ];
       ];
     ];
  Return[K];
];
```

Figure 27.6. Module to assemble the master stiffness matrix.

listed in Figure 27.7. 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.

§27.5. POSTPROCESSING

Postprocessing are activities undertaken on return from `membraneSoution`. They include optional printout of `u`, `f` and `p`, plus some simple graphic displays described below.

§27.5.1. Displacement Field Contour Plots

Contour plots of the displacement components u_x and u_y interpolated over elements from the computed node displacements cand be obtained by the following script:

```
aspect=6/5; Nsub=4; ux=uy=Table[0,{numnod}];
Do[ux[[n]]=u[[2*n-1]]; uy[[n]]=u[[2*n]], {n,1,numnod}];
uxmax=uymax=0;
```

```
MembraneNodalStresses[nodcoor_,eletyp_,elenod_,
  elemat_,elefab_,eleopt_,u_]:= Module[{numele=Length[elenod],
  numnod=Length[nodcoor],numer,type,ne,eNL,eftab,i,ii,j,k,n,
  Em,ν,Emat,th,ncoor,ue,ncount,esig,sige,sig},
  ncount=Table[0,{numnod}]; {numer}=eleopt;
  sige=  Table[0,{numele}]; sig=Table[{0,0,0},{numnod}];
  For [ne=1,ne<=numele,ne++,
       {type}=eletyp[[ne]];
       eNL=elenod[[ne]]; n=Length[eNL]; ue=Table[0,{2*n}];
       eftab=Flatten[Table[{2*eNL[[i]]-1,2*eNL[[i]]},{i,1,n}]];
       ncoor=Table[nodcoor[[eNL[[i]]]],{i,n}];
       Do [ii=eftab[[i]];ue[[i]]=u[[ii]],{i,1,2*n}];
       {Em,ν}=elemat[[ne]];
       Emat=Em/(1-ν^2)*{{1,ν,0},{ν,1,0},{0,0,(1-ν)/2}};
       th=elefab[[ne]];
       If [type=="Trig3", esig=Trig3IsoPMembraneStresses[ncoor,
          {Emat,0,0},{th},{numer},ue] ];
       If [type=="Quad4", esig=Quad4IsoPMembraneStresses[ncoor,
          {Emat,0,0},{th},{numer,2},ue] ];
       If [type=="Trig6", esig=Trig6IsoPMembraneStresses[ncoor,
          {Emat,0,0},{th},{numer,3},ue] ];
       If [type=="Quad9", esig=Quad9IsoPMembraneStresses[ncoor,
          {Emat,0,0},{th},{numer,3},ue] ];
       If [type=="Trig10",esig=Trig10IsoPMembraneStresses[ncoor,
          {Emat,0,0},{th},{numer,3},ue] ];
       esig=Chop[esig]; sige[[ne]]=esig;
       For [i=1,i<=n,i++,k=eNL[[i]]; ncount[[k]]++;
            Do[ sig[[k,j]]+=esig[[i,j]],{j,3}] ];
      ];
  For [n=1,n<=numnod,n++,k=ncount[[n]];If [k>1,sig[[n]]/=k]
      ];
  Return[sig];
];
```

Figure 27.7. Module to compute averaged nodal stresses.

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

§27.5.2. Stress Field Contour Plots

Contour plots of the stress components σ_{xx}, σ_{yy} and σ_{xy} returned by `MembraneSolution` can be produced by the following script:

```
aspect=6/5; Nsub=4; sxx=syy=sxy=Table[0,{numnod}];
Do[{sxx[[n]],syy[[n]],sxy[[n]]}=sig[[n]],{n,1,numnod}];
sxxmax=syymax=sxymax=0;
Do[sxxmax=Max[Abs[sxx[[n]]],sxxmax];
 syymax=Max[Abs[syy[[n]]],syymax];
```

Figure 27.8. Stress contour plots for Model (I) of Figure 27.3.

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

The stress plots are shown in Figure 27.8.

§27.5.3. Animation

Sometimes it is useful to animate results such as 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 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.

§27.6. A COMPLETE DRIVER CELL

A complete driver cell, (which is Cell 11 in the plane stress demo program placed on the web) is listed in Figures 27.9 through 27.11. This driver cell does the one-element test of Figure 27.1(b), which involves only one four node quadrilateral element.

The driver cell is broken down into three segements for reading convenience. Figure 27.9 lists the model definition script followed by a mesh plot command. Figure 27.10 lists the processing script and printout of displacements, forces and stresses. Figure 27.12 lists the postprocessing script to produce contour plots of displacements and stresses.

Of course one element should give the exact solution of the problem of Figure 27.1(a) with one element. That is precisely the test.

```
ClearAll[Em,ν,th];
Em=10000; ν=.25;  th=3; aspect=6/5; Nsub=4;

(*  Define FEM model *)

ElemTypes=Table[{"Quad4"},{numele}];
NodeCoordinates=N[{{0,6},{0,0},{5,6},{5,0}}];
ElemNodeLists=  {{1,2,4,3}};
numnod=Length[NodeCoordinates]; numele=Length[ElemNodeLists];
ElemMaterial=   Table[{Em,ν},{numele}];
ElemFabrication=Table[{th},{numele}];
FreedomValues=FreedomTags=Table[{0,0},{numnod}];
FreedomValues[[1]]=FreedomValues[[3]]={0,75}; (* nodal loads *)
FreedomTags[[1]]={1,0};     (* vroller @ node 1 *)
FreedomTags[[2]]={1,1};     (* fixed node 2 *)
FreedomTags[[4]]={0,1};     (* hroller @ node 4 *)
ElemOptions={True};

Plot2DElementsAndNodes[NodeCoordinates,ElemNodeLists,aspect,
    "One element mesh - 4-node quad",True,True];
```

Figure 27.9. Model definition script for problem of Figure 27.1(b).

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

{u,f,sig}=MembraneSolution[
    NodeCoordinates,ElemTypes,ElemNodeLists,
    ElemMaterial,ElemFabrication,
    ElemOptions,FreedomTags,FreedomValues];

Print["Computed displacements=",u];
Print["Recovered forces=",f];
Print["Avg nodal stresses=",sig];
```

Figure 27.10. Processing script for problem of Figure 27.1(b).

Other driver cell examples for more complicated problems 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. What changes is the model definition script portion, which often benefits from node and element generation constructs.

```
(*  Plot Displacement Components Distribution *)

ux=uy=Table[0,{numnod}];
Do[ux[[n]]=u[[2*n-1]]; uy[[n]]=u[[2*n]],{n,1,numnod}];
uxmax=uymax=0;
Do[uxmax=Max[Abs[ux[[n]]],uxmax]; uymax=Max[Abs[uy[[n]]],uymax],
  {n,1,numnod}];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodeLists,ux,
       uxmax,Nsub,aspect,"Displacement component ux"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodeLists,uy,
       uymax,Nsub,aspect,"Displacement component uy"];

(*  Plot Averaged Nodal Stresses Distribution *)

sxx=syy=sxy=Table[0,{numnod}];
Do[{sxx[[n]],syy[[n]],sxy[[n]]}=sig[[n]],{n,1,numnod}];
sxxmax=syymax=sxymax=0;
Do[sxxmax=Max[Abs[sxx[[n]]],sxxmax];
   syymax=Max[Abs[syy[[n]]],syymax];
   sxymax=Max[Abs[sxy[[n]]],sxymax],{n,1,numnod}];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodeLists,
       sxx,sxxmax,Nsub,aspect,"Nodal stress sig-xx"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodeLists,
       syy,syymax,Nsub,aspect,"Nodal stress sig-yy"];
ContourPlotNodeFuncOver2DMesh[NodeCoordinates,ElemNodeLists,
       sxy,sxymax,Nsub,aspect,"Nodal stress sig-xy"];
```

Figure 27.11. Postprocessing script for problem of Figure 27.1(b).

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 reliance on low-level languages such as Fortran. Even the simplest program would runs to throusands of code lines. At 50 lines/page that becomes difficult to fit snugly in a textbook.

Another historical problem is plotting. Languages such as C or Fortran do not include plotting libraries for the simple reason that low-level plotting standards never existed. The situation changes when using high-level languages like *Matlab* or *Mathematica*. The language engine provides the necessary fit to available computer hardware, and plotting scripts are transportable.