6

A Complete Axisymmetric FEM Program

TABLE OF CONTENTS

§6.1. Introduction

This chapter describes a complete axisymmetric FEM program for static analysis of axisymmetric solids with the quadrilateral ring elements described in the previous Chapter. The program is implemented in the *Mathematica* Notebook `QuadSOR.nb`, which is made available on the course web site for assigned computer exercises.

Unlike the previous chapter, the description is top down, i.e. starts from the main driver programs. Three benchmark examples that illustrate the operation of the code as well as the operation of the code and the level of accuracy attainable with 4-node versus 8-node configurations, are given in the next Chapter.

§6.2. FEM Analysis Stages

As in all FEM programs, the analysis of a structure by the Direct Stiffness Method involves three major stages: (I) preprocessing or model definition, (II) processing, and (III) postprocessing. The overall program organization is shown in Figure 6.1.

The preprocessing portion involves:

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

I.2 Printing data structures (optional).

I.3 Plot of the FEM mesh, including nodes and element labels (optional).

The processing stage is done by an *analysis driver* module that performs the following steps:

II.1 Assembly of the master stiffness matrix and force vector, using the element library.

II.2 Application of displacement BC. This is done by the equation modification method.

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

Upon executing these three processing steps, the node displacements are available. Several post-processing steps follow:

III.1 Recovery of forces including reactions, carried out element by element.

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

III.3 Printing computed results, including displacements, forces and stresses. If an analytical solution is available, might be printed here for comparison purposes.

III.4 Contour plot of stress fields (optional).

Steps I.1, I.2, I.3, III.3 and III.4 are done directly by a *driver script* written by the analyst. Steps II.1, II.2, II.3, III.1 and III.2 are done by the *analysis driver* program supplied as part of the code.

The *driver script*, shown on top of Figure 6.1, is written by the analyst to carry out the analysis. In *Mathematica* the driver script is placed on its own Notebook cell. The script is problem dependent: each different problem requires a new one to be writtten.

A driver script is divided into three sections that are roughly associated with the foregoing steps:

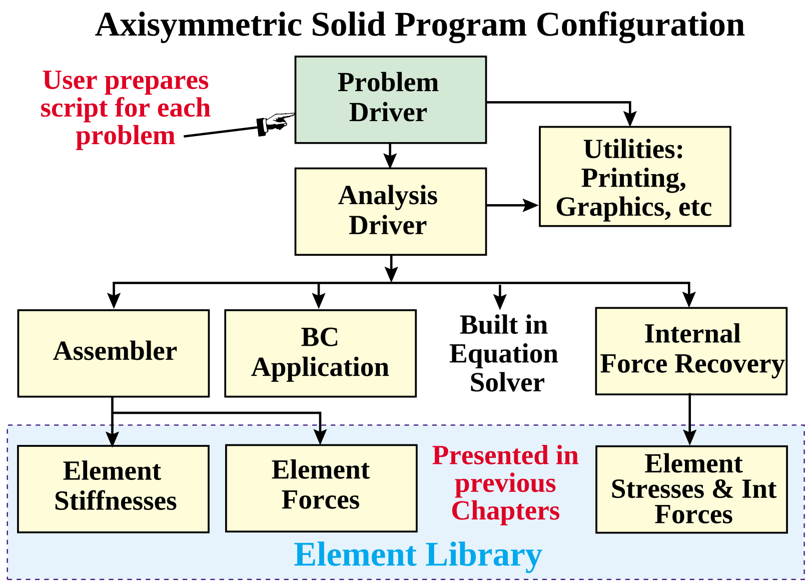
Axisymmetric Solid Program Configuration
User prepares script for each problem
Problem Driver
Utilities: Printing, Graphics, etc
Analysis Driver
Assembler
BC Application
Built in Equation Solver
Internal Force Recovery
Element Stiffnesses
Element Forces
Presented in previous Chapters
Element Stresses & Int Forces
Element Library

FIGURE 6.1. Organization of the axisymmetric solid program QuadSOR.

$$\text{Input data}\ \left\{\begin{array}{l}\text{Preprocessing phase: problem definition data}\\ \text{Processing phase: problem solving data}\\ \text{Postprocessing phase: result display data}\end{array}\right.$$

These are described in the following sections.

§6.3. Problem Definition

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

$$\text{Model data}\ \left\{\begin{array}{l}\text{Geometry data: node coordinates}\\ \text{Element data: type, connectivity, material, body and surface traction forces.}\\ \text{Degree of freedom data: node forces and support BCs}\end{array}\right.$$

Note that the element data is split into five subsets: type, connnectivity, material, body force and surface traction forces, each of which has its own data structure.[1] The degree of freedom data is split into two subsets: tags and values. In addition there are generic processing options, which are conveniently collected in a separate data set.

The input to the axisymmetric solid program `QuadSOR` consists of nine data structures, which are called `NodeCoordinates`, `ElemTypes`, `ElemNodes`, `ElemMaterials`, `ElemBodyForces`, `ElemTractionForces`, `FreedomTags`, `FreedomValues` and `DefaultOptions`. These data structures are described in the following subsections. For specific examples, reference is made to the benchmark problem and FEM discretization shown in Figures 6.2 and 6.3.

[1] For axisymmetric solid models, the fabrication data that appears, for example, in plane stress programs is empty.

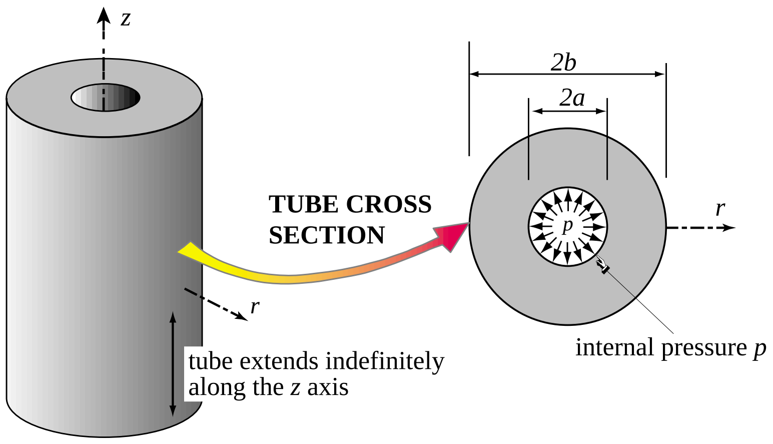

FIGURE 6.2. Benchmark problem: a thick cylinder under internal pressure.

§6.3.1. Benchmark Problem

The problem illustrated in Figure 6.2 is a standard benchmark for FEM discretizations of axisymmetric solids. A thick circular tube of internal radius a and external radius b is subject to internal pressure p. The z axis runs along the hole center. The tube extends indefinitely along the z direction and is considered to be in a plane strain state (meaning that both the axial displacement u_z and axial strain e_{zz} vanish.) The material is isotropic with elastic modulus E and Poisson ratio ν.

The tube is cut with a normal-to-z slice of width d as depicted in Figure 6.3(a). A two element discretization using two 4-node quadrilateral ring elements is constructed as shown in Figure 6.3(b).

The *Mathematica* driver script for this benchmark problem is listed in Figure 6.4. It is explained in detail in following subsections.

§6.3.2. Node Coordinates

The geometry data is specified through `NodeCoordinates`. This is a node-by-node coordinate array configured as a two-dimensional list:

$$\boxed{\texttt{NodeCoordinates = }\{\{r_1,z_1\},\{r_2,z_2\},\ \ldots\ \{r_N,z_N\}\};} \tag{6.1}$$

where N is the number of nodes. Coordinate values should be floating point numbers;[2] use the `N` function to insure that property if necessary. Nodes are numbered consecutively, starting from 1. No gaps in node numbers are permitted.

For the 2-element model of Figure 6.3, with $a = 4$, $b = 10$ and $d = 2$ it is recommended to use a parametric form:

```
a=4; b=10; d=2;
NodeCoordinates=N[{{a,0},{a,d},{(a+b)/2,0},{(a+b)/2,d},{b,0},{b,d}}];
```

[2] The QuadSOR program can also be used to process small discretizations symbolically or in exact arithmetic for special studies. But for most problems the numeric floating-point form is most appropriate as well as efficient, and that is the use emphasized here.

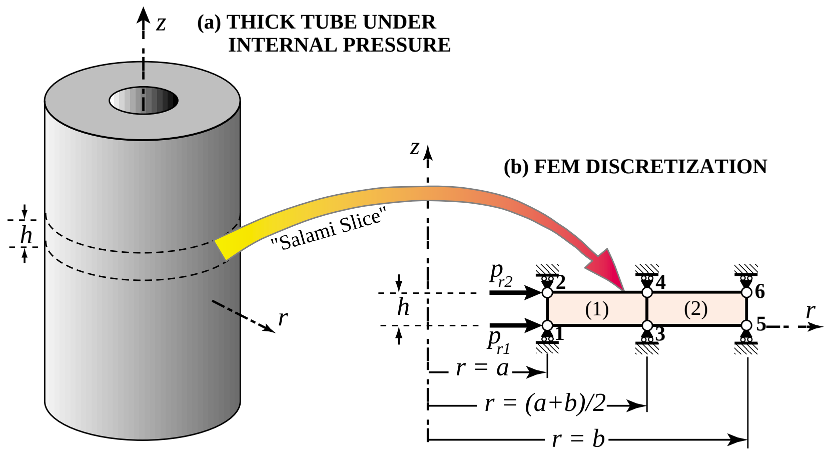

FIGURE 6.3. A slice of width d is cut from the thick tube of Figure 6.1 and used to build a 2-element discretization with 4-node quadrilateral ring elements.

The advantage of parametrizing `NodeCoordinates` is that other dimension values can be rapidly tested without the need of retyping the array each time. Such changes are error prone.

Remark 6.1. For regular discretizations, node coordinates may be constructed through mesh generation. This may be advantageous for systematically exploring mesh refinement. A module provided in `QuadSOR.nb` is `GenQuad4NodeCoordinates`, which builds nodes over a regular discretization of a four-sided region ABCD. This is invoked as

```
NodeCoordinates = GenQuad4NodeCoordinates[MeshCorners,Ner,Nez];
```
(6.2)

Here

`MeshCorners`	Array `{{`r_A, z_A`},{`r_B, z_B`},{`r_C, z_C`},{`r_D, z_D`}}]` of ABCD region corner coordinates, traversed counterclockwise A→B→C→D
`Ner`	number of elements in the radial direction, which is assumed to be A→B
`Nez`	number of elements in the axial direction, which is assumed to be A→D.

The function returns `NodeCoordinates` in the format (6.1).

The generation is done by the standard isoparametric mapping method with equal increments in the zonal ξ and η. For the discretization of Figure 6.3(b) use

```
a=4; b=10; d=2;
MeshCorners=N[{{a,0},{b,0},{b,d},{0,d}}];
NodeCoordinates=GenQuad4NodeCoordinates[MeshCorners,2,1];
```

To generate nodes for a regular mesh of 8-noded quadrilaterals use

```
NodeCoordinates = GenQuad8NodeCoordinates[MeshCorners,Ner,Nez];
```
(6.3)

The arguments have same meaning as in (6.2). Here `Ner` and `Nez` refer to the number of Quad8 elements in the radial and axial direction, respectively. As a result about 3 times more nodes will be generated compared to (6.2) if `Ner` and `Nez` are the same.

§6.3.3. Element Types

```
ClearAll[Em,ν,th,a,b,d,p,numer];
Em=1000.; ν=0; etype="Quad4"; numer=True;
Kfac=1; a=4; b=10; d=2; p=10; aspect=d/(b-a);

(*  Define FEM model *)

NodeCoordinates=N[{{a,0},{a,d},{(a+b)/2,0},{(a+b)/2,d},{b,0},{b,d}}];
ElemNodes={{1,3,4,2},{3,5,6,4}};
numnod=Length[NodeCoordinates]; numele=Length[ElemNodes];
ElemType=Table[etype,{numele}];
ElemMaterial=Table[N[{Em,ν}],{numele}];
FreedomValues=Table[{0,0},{numnod}];
FreedomTags=Table[{0,1},{numnod}];  pfor=N[Kfac*p*a*d];
FreedomValues[[1]]=FreedomValues[[2]]={pfor/2,0};
ElemBodyForces={}; ElemTractionForces={};
DefaultOptions={numer};
PrintRingNodeCoordinates[NodeCoordinates,"Node Coordinates",{2,4}];
PrintRingElementTypeNodes[ElemType,ElemNodes,"Element type & nodes",{}];
PrintRingElementMaterials[ElemMaterial,"Material data",{}];
PrintRingElemBodyForces[ElemBodyForces,"Body forces",{}];
PrintRingElemTracForces[ElemTractionForces,"Traction forces",{}];
PrintRingFreedomActivity[FreedomTags,FreedomValues,"BC data",{}];
Plot2DElementsAndNodes[NodeCoordinates,ElemNodes,aspect,
    "Pressurized thick cylinder mesh",True,True];

(*  Solve problem and print results *)

{NodeDisplacements,NodeForces,NodeStresses}=RingAnalysisDriver[
    NodeCoordinates,ElemType,ElemNodes,
    ElemMaterial,ElemBodyForces,ElemTractionForces,
    FreedomTags,FreedomValues,DefaultOptions];
PrintRingAnalysisSolution[NodeDisplacements,NodeForces,
     NodeStresses,"Computed solution",{6,6}];
{ExactNodeDisplacements,ExactNodeStresses}=
     ExactSolution[NodeCoordinates,{a,b},{Em,ν,ρ},p,
     "PressThickCylinder",numer];
PrintRingNodeDispStresses[ExactNodeDisplacements,
    ExactNodeStresses,"Exact (Lame) solution",{}];

(*  Plot Node Stress Distributions *)

legend={(a+2*b)/3,0.8*d};
ContourPlotStresses[NodeCoordinates,ElemNodes,NodeStresses,
  {True,False,True,False},True,{},legend,aspect];

(* Through-wall plots comparing FEM vs exact solutions *)

pwhat={"ur","σrr","σzz","σθθ"};  If [ν==0,pwhat={"ur","σrr","σθθ"}];
For [ip=1,ip<=Length[pwhat],ip++, what=pwhat[[ip]];
     RadialPlotFEMvsExact[etype,NodeCoordinates,NodeDisplacements,
     NodeStresses, {a,b,h},{Em,ν,ρ},p,2,
     "PressThickCylinder",what,numer]  ];
```

FIGURE 6.4. Driver script for example problem of Figure 6.3.

The element type is a label that specifies the model to be used in the discretization. Those labels are placed in a one-dimensional list:

$$\boxed{\texttt{ElemType = \{type}^{(1)}\texttt{, type}^{(2)}\texttt{, ... type}^{N_e}\texttt{\};}} \tag{6.4}$$

Here $\texttt{type}^e$ is the type descriptor of the e-th element, specified as a character string. The two element types available in the axisymmetric program are

`"Quad4"`	4-node iso-P quadrilateral ring element with 2 x 2 Gauss quadrature
`"Quad8"`	8-node iso-P quadrilateral ring element with 2 x 2 Gauss quadrature. Although rank deficient by one, this usually causes no problems as spurious modes are suppressed by boundary conditions and do not propagate through the mesh.

Some minor variations are possible:

`"Quad4.1"`	4-node iso-P quadrilateral ring element with 1 x 1 Gauss quadrature. This element is strongly rank deficient and should be avoided except in some research studies.
`"Quad8.3"`	8-node iso-P quadrilateral ring element with 3 x 3 Gauss quadrature. Although this is rank sufficient, its performance is generally inferior to that of the reduced integrated 8-node quadrilateral, especially in near-incompressible situations.

For the discretization of Figure 6.3(b), use

```
ElemType=Table["Quad4",{2}];
```

which is the same as `ElemType={"Quad4","Quad4"};`. A parametrized form of the above is

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

in which `numele` is a variable that contains the number of elements. This can be extracted, for example, as `numele=Length[ElemNodes]`, where `ElemNodes` is described below, if `ElemNodes` is defined before `ElemType`, as is often the case. The advantage of the foregoing parametrized form is that the statement does not have to be modified if the number of element changes, or if a mesh generation utility is used.

§6.3.4. Element Connectivity

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

$$\boxed{\texttt{ElemNodes = \{\{enl}^{(1)}\texttt{\}, \{enl}^{(2)}\texttt{\}, ... \{enl}^{N_e}\texttt{\}\};}} \tag{6.5}$$

where $\texttt{enl}^e$ denotes the lists of nodes of the e^{th} element given by global node numbers, and N_e is the total number of elements.

Element boundaries must be traversed counterclockwise (CCW) starting at any corner. Numbering elements with midnodes, such as the 8-node quadrilateral, requires some care: first the corners are listed CCW, followed by the midpoints (first midpoint is the one that follows the first corner when going CCW). When elements have an interior node, as in the 9-node biquadratic quadrilateral (not implemented in `QuadSOR`) that node goes last.

For the discretization of Figure 6.3(b):

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

[3] Some FEM programs call this the "topology data," which sounds more impressive.

Remark 6.2. This data structure may also be generated if the mesh is topologically regular. A companion module of `GenQuad4NodeCoordinates` that generates `ElemNodes` for a 4-node quadrilateral mesh is `GenerateQuad4ElemNodes`. This is invoked as

$$\boxed{\texttt{ElemNodes = GenQuad4ElemNodes[Ner,Nez]}} \tag{6.6}$$

in which the two arguments must agree with the `Ner` and `Nez` used in `GenQuad4NodeCoordinates`; see (6.2). For the discretization of Figure 6.3(b), use

```
ElemNodes = GenQuad4ElemNodes[2,1];
```

Module `GenQuad8ElemNodes` may be used to generate a regular mesh of 8-node quadrilaterals. It is invoked as

$$\boxed{\texttt{ElemNodes = GenQuad8ElemNodes[Ner,Nez]}} \tag{6.7}$$

in which the two arguments must agree with the `Ner` and `Nez` used in `GenQuad8NodeCoordinates`; see (6.3).

§6.3.5. Material Properties

Array `ElemMaterial` lists the elastic modulus E and the Poisson's ratio ν for each element:

$$\boxed{\texttt{ElemMaterial = }\{\{E^{(1)},\nu^{(1)}\},\ \{E^{(2)},\nu^{(2)}\},\ \ldots\ \{E^{N_e},\nu^{N_e}\}\}} \tag{6.8}$$

In the frequent case in which all elements have the same E and ν, this list can be easily generated by a `Table` command. For the discretization of Figure 6.3(b), assuming $E = 1000$ and $\nu = 0$:

```
Em=1000; ν=0; ElemMaterial = Table[N[{Em,ν}],{numele}]
```

in which `numele=Length[ElemNodes]` is the number of elements (2 for this mesh). Again it is advantageous to declare the numerical values separately for ease of modification.

Remark 6.3. In its present version, `QuadSOR` only accepts isotropic materials. The element modules, however, have been written for arbitrary anisotropic material. The limitation resides in the `ElemMaterial` data structure. Extending the input data to accept more general materials would not be too difficult.

§6.3.6. Element Body Forces

Array `ElemBodyForces` lists the value of the body force field element by element, specified as forces per unit volume. This list can have two forms. The simplest form gives, for each element, the value of the body force radial and axial components b_r and b_z at the quadrilateral center (the center coordinates are the average of the node coordinates):

$$\boxed{\texttt{ElemBodyForces = }\{\{\texttt{bz}_0^{(1)},\texttt{br}_0^{(1)}\},\ \{\texttt{bz}_0^{(2)},\texttt{br}_0^{(2)}\},\ \ldots\ \{\texttt{br}_0^{N_e},\texttt{bz}_0^{N_e}\}\}} \tag{6.9}$$

For the problem of Figure 6.2, the body forces are zero, so one could just say

```
ElemBodyForces = {{0,0},{0,0}}
```

or in parametrized form:

```
ElemBodyForces = Table[{0,0},{numele}
```

in which `numele = Length[ElemNodes]` is the nunber of elements. An even more compact form is to set `ElemBodyForces` to the empty list:

```
ElemBodyForces = { };
```

This has the advantage that the consistent node force computations are skipped.

To give a less trivial example, assume that for the mesh of Figure 6.3(b), $b_r = 6r$ (linear in r and $b_z = -3$ (constant). Then

```
ElemBodyForces = {{33,-3},{51,-3}}
```

Where do values 33 and 51 come from? The center of element (1) is at $r_0 = (4+7+7+4)/4 = 5.5$ and so $b_{r0}^{(1)} = 6 \times 5.5 = 33$, whereas for element (2), $b_{r0}^{(2)} = 6 \times 8.5 = 51$.

The second form of `ElemBodyForces` lists, also element by element, the value of b_r and b_z at each element node, in one to one correspondence with the element node lists provided in `ElemNodes`. This is more easily visualized for the example mesh of Figure 6.3(b). Assuming again the force distribution $b_r = 6r$ and $b_z = -3$,

```
ElemBodyForces = {{{24,-3},{42,-3},{42,-3},{24,-3}},
                  {{42,-3},{60,-3},{60,-3},{42,-3}}};
```

Remark 6.4. If body forces are functions of the position coordinates $\{r, z\}$, (for example, in the case of a centrifugal force), it might be convenient to calculate the entries of `ElemBodyForces` in terms of the data in `ElemNodes` and `NodeCoordinates` instead of entering numerical values directly. This requires some programming but it is less error prone than hand computation. For example, suppose that $b_r = 3r - 2z$ and $b_z = 6r + 4z$ over a mesh of 4-noded quadrilaterals. Using the first `ElemBodyForces` format:

```
ElemBodyForces = Table[{0,0},{numele}];
For [e=1,e<=numele,e++, {n1,n2,n3,n4}=ElemNodes[[e]];
     {r0,z0}=(NodeCoordinates[[n1]]+NodeCoordinates[[n2]]+
              NodeCoordinates[[n3]]+Nodecoordinates[[n4]])/4;
     ElemBodyForces[[e]]={3*r0-2*z0,6*r0+4*z0}];
```

§6.3.7. Element Traction Forces

Array `ElemTractionForces` specifies values of surface tractions, such as pressures, that are to be converted to consistent node forces. This conversion capability is not yet implemented in `QuadSOR`. Consequently this array must be set to the empty list:

```
ElemTractionForces = { };
```

§6.3.8. Freedom Tags

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

$$\boxed{\texttt{FreedomTags=\{\{tag}_{r1}\texttt{,tag}_{z1}\texttt{\},\{tag}_{r2}\texttt{,tag}_{z2}\texttt{\}, ... \{tag}_{rN}\texttt{,tag}_{zN}\texttt{\}\}}} \tag{6.10}$$

The tag value is 0 if the force is specified and 1 if the displacement is specified. When there are a lot of nodes and comparatively few displacement BCs, a quick way to build this list is to start from all zeros, and then insert the boundary conditions appropriately.

For the discretization of Figure 6.3(b), all nodes happen to have the same displacement boundary condition: free to move along the radial direction while unable to move axially. The tag for each node is `{0,1}`, and one `Table` statement suffices:

```
FreedomTags=Table[{0,1},{6}];
```

Instead of entering the number of nodes (6) directly, however, parametrization is recommended:

```
FreedomTags=Table[{0,1},{numnod}];
```

where `numnod` is the number of nodes as extracted from the geometry definition; that is, `numnod = Length[NodeCoordinates]`.

§6.3.9. Freedom Values

Array `FreedomValues` has the same node by node configuration as that of `FreedomTags`:

$$\boxed{\texttt{FreedomValues=\{\{val}_{r1}\texttt{, val}_{z1}\texttt{\},\{val}_{r2}\texttt{, val}_{z2}\texttt{\}, ... \{val}_{rN}\texttt{, val}_{zN}\texttt{\}\}}} \tag{6.11}$$

This array lists the specified values of the applied node force component if the corresponding freedom tag is zero, and of the prescribed displacement component if the tag is one. Often most of the list entries are zero.

For the mesh of Figure 6.3(b), only two force values are nonzero: the radial forces on nodes 1 and 2 due to the internal pressure p. It is convenient to let the script calculate the node force values from the data: $f_{r1} = f_{r2} = \frac{1}{2} p\, a\, d = \frac{1}{2} 10 \times 4 \times 2 = 40$:

```
a=4; d=2; p=10;  pfor=p*a*d;
numnod = Length[NodeCoordinates];
FreedomValues = Table[{0,0},{numnod}];
FreedomValues[[1]] = FreedomValues[[2]]={pfor/2,0};
```

Here the circumferential span factor K_{fac}, introduced in Chapter 11, is assumed to be 1. If this factor, called `Kfac` in the `QuadSOR` program, is not one, the foregoing computation of the total pressure force on face 1–2 should be changed to `pfor = Kfac*p*a*d`. In fact this is the way implemented in the driver script of Figure 6.4; note that `Kfac` is set to 1 at the script top.

```
Node Coordinates
node      r-coor     z-coor
   1      4.0000     0.0000
   2      4.0000     2.0000
   3      7.0000     0.0000
   4      7.0000     2.0000
   5     10.0000     0.0000
   6     10.0000     2.0000

Element type & nodes
elem     type       node-list
   1    Quad4    {1, 3, 4, 2}
   2    Quad4    {3, 5, 6, 4}

Material data
elem    material: {E,ν}
   1        {1000., 0.}
   2        {1000., 0.}

No body forces specified
No traction forces specified

BC data
node    r-tag    z-tag   r-value   z-value
   1        0        1     40.00      0.00
   2        0        1     40.00      0.00
   3        0        1      0.00      0.00
   4        0        1      0.00      0.00
   5        0        1      0.00      0.00
   6        0        1      0.00      0.00
```

FIGURE 6.5. Model definition data structures printed by the script of Figure 6.4.

§6.3.10. Element Processing Options

Array `DefaultOptions` declares certain behavioral options for the processing phase. The most important is `numer`. This is a logical flag that specifies that computations are to be carried out in floating-point arithmetic if `True`, which significantly speeds up processing. Setting `numer` to `False` specifies exact or symbolic arithmetic, which is only advisable for certain research studies that carry free parameters along. Accordingly the recommended setting is

$$\boxed{\texttt{numer=True;\ \ DefaultOptions=\{numer\};}} \tag{6.12}$$

Although more default options may be given following `numer`, it is not recommended to play with those.

§6.3.11. Printing Model Definition Data

The model definition data structures may be printed by the statements shown in the driver script of Figure 6.4, which are

```
PrintRingNodeCoordinates[NodeCoordinates,"Node Coordinates",{2,4}];
PrintRingElementTypeNodes[ElemType,ElemNodes,"Element type & nodes",{}];
PrintRingElementMaterials[ElemMaterial,"Material data",{}];
PrintRingElemBodyForces[ElemBodyForces,"Body forces",{}];
PrintRingElemTracForces[ElemTractionForces,"Traction forces",{}];
PrintRingFreedomActivity[FreedomTags,FreedomValues,"BC data",{}];
```

The resulting printed output is shown in Figure 6.5. Such statements could be commented out to reduce output volume once the model definition is considered bug free.

§6.3.12. Mesh Display

A mesh plot may be produced by the statement:

$$\boxed{\texttt{Plot2DElementsAndNodes[NodeCoordinates, ElemNodes, aspect, title, True, True];}} \tag{6.13}$$

Here `aspect` is the plot frame aspect ratio (z dimension over r dimension), `title` is a character string specifying a plot title, and the last two `True` argument values ask that node labels and element labels, respectively, be displayed.

The mesh plot produced by

```
Plot2DElementsAndNodes[NodeCoordinates,ElemNodes,aspect,
    "Pressurized thick cylinder mesh",True,True];
```

that appears in the driver script of Figure 6.4 is shown in Figure 6.6.

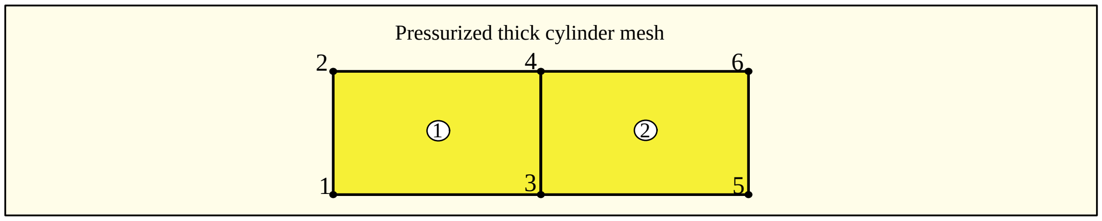

FIGURE 6.6. Mesh plot produced by the script of Figure 6.4.

§6.4. Processing

The static solution is carried out by calling the analysis driver module `RingAnalysisDriver`. The function call is

```
{NodeDisplacements,NodeForces,NodeStresses}=RingAnalysisDriver[
   NodeCoordinates,ElemType,ElemNodes,
   ElemMaterial,ElemBodyForces,ElemTractionForces,
   FreedomTags,FreedomValues,DefaultOptions];
```

The arguments of this call: `NodeCoordinates`, `ElemType`, `ElemNodes`, `ElemMaterial`, `ElemBodyForces`, `ElemTractionForces`, `FreedomTags`, `FreedomValues` and `DefaultOptions`, have been described in the foregoing section.

Assuming no processing errors occur, `RingAnalysisDriver` returns the following arrays:

`NodeDisplacements`	The computed node displacements, arranged as `{{ur1,uz1},{ur2,uz2}, ... {urN,uzN}}`.
`NodeForces`	The computed node forces, including recovered reactions, arranged as `{{fr1,fz1},{fr2,fz2}, ... {frN,fzN}}`.
`NodeStresses`	Recovered stresses at node locations, arranged as `{{`σ`rr1,`σ`zz1,`$\sigma\theta\theta$`1,`σ`rz1}, ... {`σ`rrN,`σ`zzN,`$\sigma\theta\theta$`N,`σ`rzN}}`.

§6.5. Postprocessing

Postprocessing mean activities undertaken on return from the analysis driver. They include printout of node displacements, node forces and node stresses, as well as contour plots of stress distributions.

```
Computed solution
node    r-disp    z-disp   sigma-rr  sigma-zz  sigma-θθ  sigma-xy   r-force   z-force
   1    0.0532    0.0000    -4.5536    0.0000   12.3907    0.0000   40.0000    0.0000
   2    0.0532    0.0000    -4.5536    0.0000   12.3907    0.0000   40.0000    0.0000
   3    0.0395    0.0000    -2.6527    0.0000    5.3211    0.0000    0.0000    0.0000
   4    0.0395    0.0000    -2.6527    0.0000    5.3211    0.0000    0.0000    0.0000
   5    0.0373    0.0000    -0.7518    0.0000    3.6338    0.0000    0.0000    0.0000
   6    0.0373    0.0000    -0.7518    0.0000    3.6338    0.0000    0.0000    0.0000

Exact (Lame) solution
node    r-disp    z-disp   sigma-rr  sigma-zz  sigma-θθ  sigma-xy
   1    0.0552    0.0000   -10.0000    0.0000   13.8095    0.0000
   2    0.0552    0.0000   -10.0000    0.0000   13.8095    0.0000
   3    0.0405    0.0000    -1.9825    0.0000    5.7920    0.0000
   4    0.0405    0.0000    -1.9825    0.0000    5.7920    0.0000
   5    0.0381    0.0000     0.0000    0.0000    3.8095    0.0000
   6    0.0381    0.0000     0.0000    0.0000    3.8095    0.0000
```

FIGURE 6.7. Computed solution results produced by script of Figure 6.4.

§6.5.1. Printing Analysis Results

The displacements, forces and stresses return by the analysis driver may be printed by saying

```
PrintRingAnalysisSolution[NodeDisplacements,NodeForces,
    NodeStresses,"Computed solution",{}];
```

Here `"Computed solution"` may be replaced with another title string. The last argument may be used to control the number of places printed after the decimal point. See Figure 6.7 for an example.

If an analytical solution is available as in the case of the pressurized thick cylinder, it may be evaluated and printed at this point to facilitate comparison by inspection. Cells 11–13 of the posted notebook provide a template on how to do that.

§6.5.2. Stress Field Contour Plots

To do contour plots of stress fields you may use the call to module `ContourPlotStresses`, which is invoked as

```
ContourPlotStresses[NodeCoordinates,ElemNodes,NodeStresses,
                   whichones,showrange,plotopt,legend,aspect];
```
(6.14)

The arguments beyond `NodeCoordinates`, `ElemNodes` and `NodeStresses` are:

`whichones`	A 4-logical-flag list that specifies which stress components are to be plotted. Better understood by example: `whichones={True,False,True,False}` requests that σ_{rr} and $\sigma_{\theta\theta}$, which are entries 1 and 3 of the stress vector, be plotted.
`showrange`	A logical flag. If `True`, a line giving the stress plot range and the increment between contour lines is printed before the plot.
`plotopt`	A list of six logical flags passed to the plotter for things such as "draw node locations" and so on. If set to { }, the `ContourPlotStresses` default options are assumed; this is recommended unless one wants to play with the appearance.
`legend`	If given as an empty list, that is, `legend={ }`, no legend table is produced. If `legend={rleg,zleg}` a contourline legend box will appear drawn with the upper left corner at that position. The implementation of this feature is presently crude; thus `legend={ }`, which skips the legend display, is recommended unless you want to play with settings.
`aspect`	The axial-to-radial (z dimension over r dimension) plot aspect ratio.

See Figure 6.8 for the plots produced by the driver script using

```
legend={(a+2*b)/3,0.8*d};
ContourPlotStresses[NodeCoordinates,ElemNodes,NodeStresses,
      {True,False,True,False},True,{},legend,aspect];
```

here `aspect=d/(b-a)` is defined at script start, since that value is used also for the mesh plot.

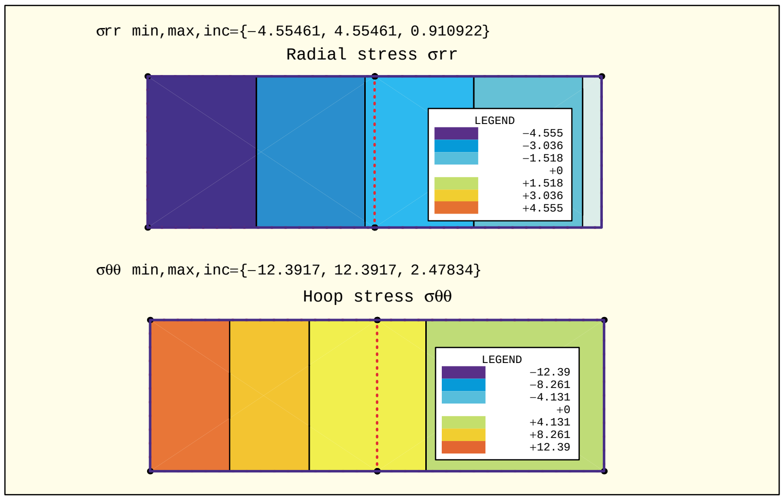

FIGURE 6.8. Stress contour plots results produced by script of Figure 6.4.

§6.5.3. Exact vs. FEM Radial Plots

The driver script produces 3 plots comparing FEM resultsversus the exact solution for radial displacements u_r as well as radial and hoop stresses $\{\sigma_{rr}, \sigma_{\theta\theta}\}$.[4] The plots are produced by

```
pwhat={"ur","srr","szz","sθ"}; If [ν==0,pwhat={"ur","srr","sθ"}];
For [ip=1,ip<=Length[pwhat],ip++, what=pwhat[[ip]];
      RadialPlotFEMvsExact[etype,NodeCoordinates,NodeDisplacements,
      NodeStresses, {a,b,h},{Em,ν,r},p,2,
      "PressThickCylinder",what,numer] ];
```

The three plots produced by this loop are collected in Figure 6.9.

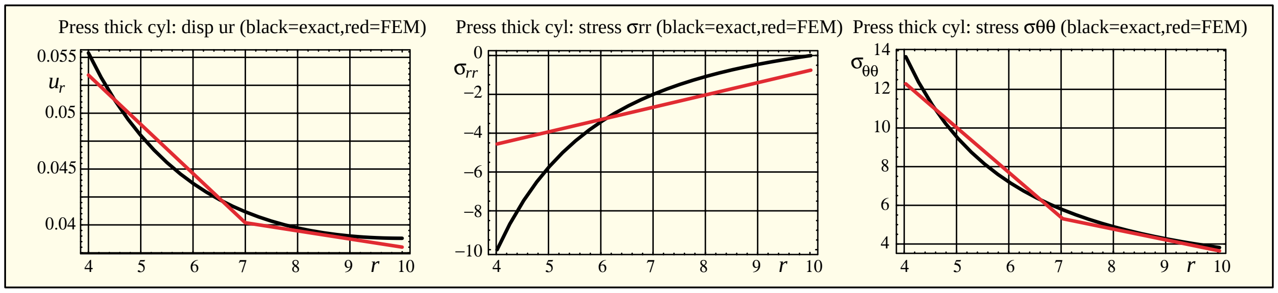

FIGURE 6.9. Comparison of exact vs. FEM results produced by driver script of 6.4.

[4] Note that 2 components that are identically zero: σ_{zz} and σ_{rz}, are skipped. Component σ_{zz} would be nonzero if $\nu > 0$.

Homework Exercises for Chapter 6

A Complete Axisymmetric FEM Program

EXERCISE 6.1 [C:20] This exercise deals with the thick-tube benchmark example discussed in §13.2–5. The script for this exercise is in Cell 12 of the Notebook `Quad4SOR.nb` supplied with this Chapter.

(a) Repeat the 4-element analysis using $\nu = 0.45$ and $\nu = 0.499$. Describe what happens to the accuracy of displacements and stresses when compared with the exact solution.

(b) Can a more refined mesh fix the problems noted in (a)? To check, run 16 elements along the radial direction and report if things have improved.

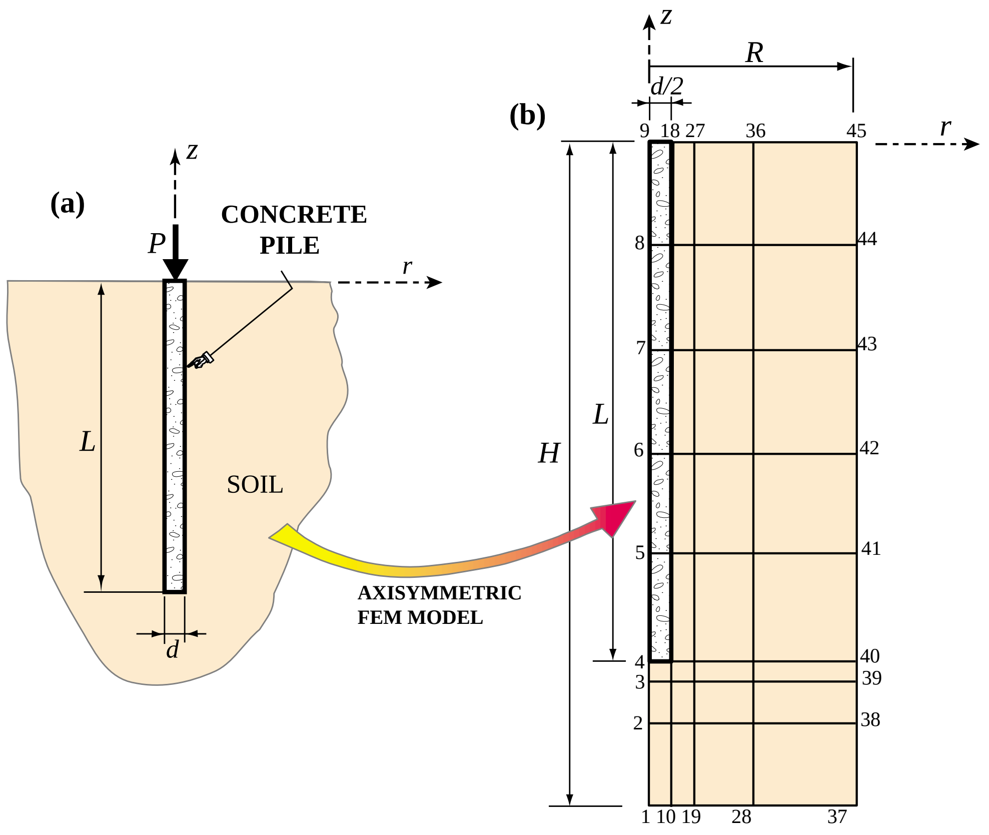

FIGURE E6.1. Pile embebded in soft soil.

EXERCISE 6.2 [C:20] A concrete pile embedded in a soil half-space, as defined in Figure E6.1 This problem is solved using a very coarse mesh in Cell 17 of the Notebook `Quad4SOR.nb` supplied with this Chapter. Repeat the analysis using a more refined mesh, like that suggested in the figure. Module `GenerateGradedRingNodeCoordinates` may be used to generate a graded regular mesh.

The total force applied to pile is $P = 500000$ lbs (this may be assumed to be uniformly distributed on the top pile surface, or placed as a point load). Other data: pile modulus $E_P = 300,000$ psi, soil modulus $E_S = E_P/50$, Poisson's ratio for pile $\nu_P = 0.1$, Poisson's ratio for soil $\nu_S = 0.40$, pile diameter $d = 10$ in,

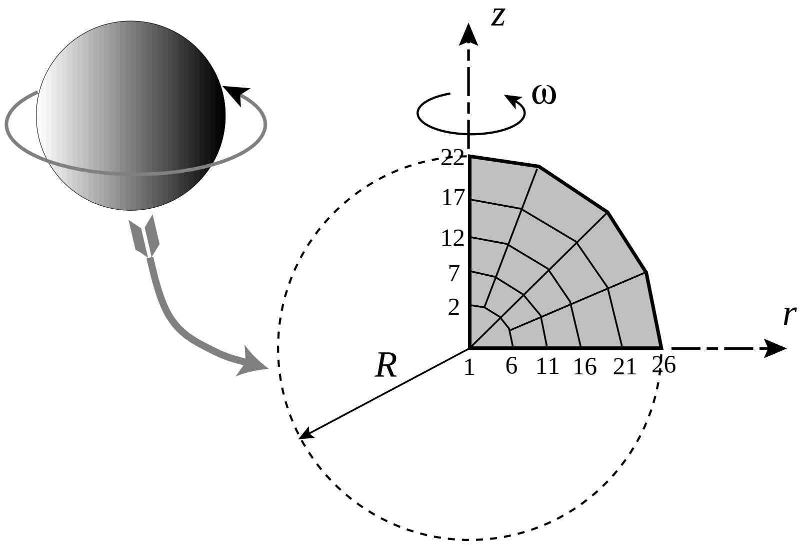

FIGURE E6.2. Our spinning planet.

pile length $L = 250$ in, and mesh z-length $H = 1.2L$. Truncate the mesh at $R = 80$ in. Neglect all body forces.

Nodes on the "soil truncation boundary" should be fixed. Points on the z axis $r = 0$ should be on vertical rollers except for 1.

The most interesting numerical results are: (i) the top and bottom vertical displacement of the pile, (ii) the reaction forces at the bottom of the pile, and (iii) the normal stress σ_{zz} in the pile.

EXERCISE 6.3 [C:20] The spinning Mother Earth. Model one quadrant of the planet cross section with an axisymmetric finite element mesh as sketched in Figure E6.2 (note that all elements are 4-node quadrilaterals). The problem is solved in Cell 18 of the Notebook `Quad4SOR.nb` supplied with this Chapter, using a very coarse mesh of only 4 elements.

Use the Kg-force/m/sec unit system for this problem. The Earth spins with angular velocity $\omega = 2\pi$ rad/24hrs $= (2\pi/86400)$ sec^{-1} about the z axis. The planet radius is R = 6370 Km = 6.37 10^6m. For E take 1/3 of the rigidity of steel, or E = 7 10^5 Kg/cm^2 = 7 10^9 Kg/m^2 as Love (Theory of Elasticity) recommends; Poisson's ratio $\nu = 0.3$ (this is my own guess), and mass density $\rho = 5.52$ times the water density. [Watch out for units: the centrifugal body force $\rho\omega^2 r$ should come up in Kg/m^3.] All gravitational field effects (self weight) are ignored.

(a) Get the equatorial "bulge" and the polar "flatening" in Km, and the maximum stress in MPa.

(b) Where do the maximum normal stresses occur?