FEM2D

Two Dimensional Finite Element Analysis
Edition 1.2, for FEM2D Version 3.0
May 2009

Russell Leighton

This is the third edition of the FEM2D documentation.

FEM2D Copying Conditions

Disclaimer

1 Introduction

FEM2D forms the core of a collection of codes developed to perform various analyses on structures using the finite element technique. Specifically, FEM2D is a pre/post-processor used to create two dimensional finite element models and to interpret the results from subsequent solutions. It was developed to handle both "plane" and axisymmetric two dimensional finite element analysis problems. It is a fairly general capability in the sense that a wide variety of two dimensional problems can be modeled by FEM2D. Other codes are included to provide a fairly complete analysis capability. These codes include the solvers (see Appendix A [Lin2D], page 33, and Appendix B [Nln2D], page 35), a node equivalence code (see Appendix D [Equ2D], page 39), and a node order optimizer (see Appendix E [Opt2D], page 41). Of course, the user is free to utilize other utility codes to enhance the operation and functionality of this collection.

The main features of the FEM2D collection are as follows:

- two-dimensional plane or axisymmetric problems
- cartesian or polar coordinate systems
- uses Rexx macro language
- linear and quadratic isoparametric elements available
- no limit on number of nodes, elements, materials, or boundary conditions (only limited by memory)
- no need to specify problem size (determined at run time)
- group (part) definition/manipulation
- simple draw commands for symbols, lines, and labels
- Dirichlet (constraint) or Neumann (flux), point or distributed boundary conditions
- globally or locally referenced material properties
- laminated materials
- definable post-processing functions
- solution and result superposition
- command aliases for customized uses
- linear and nonlinear (geometric) solvers provided
- solvers use a symmetric banded matrix storage format
- solvers can handle coupled analyses
- node order optimizer for bandwidth reduction
- node equivalence code for part "welding"

2 Usage

This chapter serves as an introduction into the usage of FEM2D. Briefly, the philosophy behind the operation of the codes and, specifically, use of the "as distributed" user interface will be discussed.

2.1 Coding Philosophy

When I first set out to write a finite element code for the Amiga I decided from the very beginning on a programming philosophy. I was determined to utilize what I feel is one of the single greatest advantages of a multitasking operating system, that is, the ability to create a complex application from simple building blocks, each having a distinct, unique capability. Depending on the user's needs the application could be built to suit very specific requirements. Additionally, each module (program) can be designed to handle a particular task very efficiently, making best use of available computational resources. This capability, as I have briefly outlined it, was a driver behind the development of many of the commands available under the UNIX operating system. It was felt that small, efficient utilitarian codes could be used together to form more complicated operations as opposed to developing single large applications. In general, UNIX operating system commands allow use of Input/Output redirection and piping to "connect" the smaller utilities to perform complex operations. Once a user becomes accustomed to making use of this ability it can be utilized very effectivily. Of course, on the Amiga this capability also exists, but with the added twist of InterProcess Communication (IPC) afforded by the Amiga's Exec kernel. With the implementation of ARexx as an InterProcess Control Language (allowing the user to control IPC), the Amiga has the ability to perform unique operations unrealized in any other operating system. However, even on the Amiga, this capability is rarely exploited beyond the control of single processes.

Most of the codes that form the FEM2D collection make extensive use of the Amiga's InterProcess Communication (through message ports) which is controlled through Rexx (which is in fact an interpreter running as another process with its own message port for IPC). By utilizing Rexx not only were the above discussed benefits realized but the advanced programming constructs available in the REXX language could be taken advantage of directly by the individual codes. In other words, by relying on Rexx there was no need to implement a macro language into FEM2D (programming constructs like conditional 'IF' blocks or 'DO' loops). For example, FEM2D understands the 'node' command and expects three numerical arguments. However, within an Rexx macro, variables and 'DO' loops may be used to generate 'node' commands. FEM2D "sees" these as a series of 'node' commands with numeric arguments since Rexx handles the variable interpretation and programmatic looping. Hence, some very complex mesh generation capabilities can be built using strictly Rexx commands. Additionally, since several programs can be run simultaneously, each with its own message port, macros can be constructed to perform complete analysis functions tailored to the requirements of the problem being analyzed. A fairly general modeling, analysis, and results interpretation application can be built just from the provided codes and some Rexx macros. In fact several have already been created and will be described in the following section.

2.2 Macros and Syntax

FEM2D accepts Rexx commands passed to its message port (address fem2d). These include all necessary commands to completely define the two dimensional finite element model, material properties, and boundary conditions. It is possible (and often most desirable) to create a model completely from these commands (saved in an Rexx macro file). The model may be saved to a binary model file which can then be read back into FEM2D at a later time or read into and processed by a solver (see Appendix A [Lin2D], page 33, and Appendix B [Nln2D], page 35), post-processor (see Appendix C [Post2D], page 37), or other utility codes. In turn, the results produced by other codes can be saved back into the model file which in turn can be graphically output by FEM2D.

In general the command line syntax for executing FEM2D as a stand-alone program is:

```
-> [run] fem2d [script]
```

where *script* is the name of an input Rexx script to be processed. If no script is specified then the code will wait until it receives a command, presumably from a subsequently executed Rexx command or script.

Commands for FEM2D, placed in an Rexx macro, consist of a command keyword followed by arguments. The arguments may be separated from the command keyword with parentheses and delimited with commas (analogous to a function call) or may be separated (and delimited) with spaces. If parentheses and commas are used then the command line must be surrounded by single quotes or else Rexx will interpret the command as a library function call (rather than a command meant for the host). The use of spaces to delimit the command keyword and arguments is recommended for most commands, however, the use of parentheses and commas is recommended when defining aliases (see Section 3.6 [Utility Commands - alias], page 17). Also any expressions that are meant to be evaluated internally by FEM2D should be surrounded by single quotes (see Section 3.6 [Utility Commands - set], page 17). For example, the lines below show a sequence of Rexx commands for FEM2D.

```
/* this is a simple example of an Rexx macro for FEM2D */

address fem2d

set 'struct 0'
set 'la=a2*a3/((1+a3)*(1-2*a3))'
set 'mu=a2/(2*(1+a3))'
alias 'iso=mat(a1,0,la+2*mu,la+2*mu,la+2*mu,mu,la,la,0.0,la)'
alias 'quad4=elem(a1,a2,4,4,a3,a4,a5,a6)'

reset
problem 'struct'
axisym
iso 1 1.0 0.0
h = 0.25
do i=0 to 4
    do j=1 to 5
        k = i*5 + j
        node k j*h i*h
    end
end
do i=0 to 3
    do j=1 to 4
        k = i*4 + j
        n = i*5 + j
        quad4 k 1 n n+1 n+6 n+5
    end
end
```

Note that the `alias` and `set` commands always require expressions as arguments (to be evaluated internally by `FEM2D`) and, therefore, the arguments must be surrounded by single quotes. Also note that in the `problem` command the argument is also surrounded by single quotes since `'struct'` is an internally defined variable (which is not known to `Rexx`). The general rule is to surround expressions (which can be single variables), which are to be interpreted internally by `FEM2D`, with single quotes. Otherwise, expressions (and variables) will be interpreted by `Rexx` before being sent to `FEM2D`.

The above example shows the command `node k j*h i*h` where `i`, `j`, `k`, and `h` are `Rexx` variables. In the first pass through the loop containing this command the actual command line issued to `FEM2D` is `node 1 0.25 0.0` which results in the creation of node 1 at coordinates (0.25,0.0). For further details on the `Rexx` macro language the interested reader should refer to texts (see Chapter 5 [References], page 29) that specifically address the `Rexx` language. `Rexx` is a complete macro programming language and as such cannot not be described in adequate detail here. The following section, however, describes the commands available within `Rexx` macros for use with `FEM2D`. A detailed example is also included in a later chapter.

It is important to note that all identification numbers are one based. That is all material, node, element and element side identification numbers must start at one or greater.

Most examples shown (as well as most of the commands in the scripts included) use parentheses and commas to delimit commands and arguments. However, this is done only for clarity. In addition, to help clarify commands as well as to demonstrate the flexibility of the codes, many of the examples show the use of the `alias` and `set` commands. It is through the use of these commands that `FEM2D` can be tailored to many different engineering applications. More on this will be discussed later.

2.3 Modeling Considerations

Since the stiffness matrix resulting from the finite element descretization is generally large, sparse and is always symmetric, storage space and computational requirements can be reduced by minimizing the bandwidth of the stiffness matrix. The bandwidth of the stiffness matrix is directly related to the maximum separation in node order in each element. It should be noted that the node ordering, not the assignment of node identification numbers, affects the bandwidth. For rectangular grids the minimum bandwidth can be achieved by defining the nodes in the direction defined by the shorter side. For non-rectangular grids it is not so apparent how to order the grid to achieve the minimum bandwidth. A rule of thumb is to try to keep the difference between the earliest defined node and latest defined node within any element to a minimum. The absolute worst ordered grid would be to have an element that contains both the first defined node and the last defined node.

It is also important to keep local node numbering within all elements ordered in a counterclockwise direction to avoid negative Jacobians. It is also good practice to number this way as a habit in order to maintain model consistency.

A node order optimizer called `Opt2D` is included with this distribution. For poorly ordered grids this optimizer can significantly reduce the bandwidth and therefore reduce memory requirements and execution times. However, for low aspect ratio rectangular grids with well ordered nodes the optimizer can increase the bandwidth. Documentation is provided for `Opt2D`.

2.4 Element Instability

All element configurations exhibit "spurious" modes (instabilities) which can typically be prevented through proper boundary condition specification. These modes can be described as zero energy modes where certain nontrivial solutions produce no energy. An example of a zero energy mode is rigid body motion (translation and/or rotation). Through proper application of displacement (Dirichlet) boundary conditions, rigid body motions can be prevented. However, some element configurations exhibit additional "spurious" modes some of which render the element unusable. Proper selection of element order and number of integration points can prevent these extra modes from occurring, however, over-integration is also undesirable due to the additional (typically unnecessary) computational requirements.

To determine if an element is usable one must solve the eigenvalue problem associated with the particular element configuration (that is find all eigenvalues and associated eigenvectors that satisfy the equation `Ku = cu`, where `K` is the element stiffness matrix and `c` and `u` are the eigenvalues and eigenvectors). The eigenvectors associated with any zero eigenvalues represent a basis defining (spanning) the null space of `K` (i.e. any vectors contained in this space may be represented as a linear combination of the basis vectors defining the space). That is these eigenvectors (and any linear combination of these vectors), if operated on by `K`, result in no reactive force (no energy produced). Applying these zero energy eigenvectors to the nodal coordinates one obtains a visual representation of the zero energy modes.

To illustrate, consider the vector valued displacement problem using the eight node element with a `2x2` integration rule. The number of zero energy modes, for this element, is four (found by subtracting the rank of the element stiffness matrix from its order). These modes correspond to the expected three rigid body motions (two translations and one rotation) plus an extra "spurious" mode. However, this element is still usable since this extra mode is not communicable (adjoining elements provide constraint against this mode). It can also be shown that for scalar valued problems (such as heat conduction problems) this element is over-integrated and exhibits no extra "spurious" modes.

For the same element order, but with a `3x3` integration rule instead, the number of zero energy modes is three which corresponds to the three possible rigid body motions. This element is in fact over-integrated thereby resulting in no extra "spurious" modes but at a cost of doing more calculations (and not necessarily gaining any accuracy).

To further illustrate, consider the vector valued displacement problem using the nine node element with a `2x2` integration rule. The number of zero energy modes, for this element, is six. The extra zero energy modes found for this element are all "spurious" modes some of which are communicable thereby rendering this element unusable. With a `3x3` integration rule this element is over-integrated but is the element of choice since the `2x2` integration rule results in an unusable element.

2.5 Post-Processing Accuracy

The accuracy of the results obtained from the finite element method can depend heavily on where evaluation of the solution is performed. The locations, within a particular element, where the results (based on solution derivatives) are the most accurate have been shown to coincide with the Gaussian quadrature points with the order of the Gaussian rule chosen to be the same as the order of the element. The capability to compute results at these locations is an option in the external post-processor (see Appendix C [Post2D], page 37, and Section 3.6 [Utility Commands - select], page 17) and should be used when the solution derivative is known to be changing rapidly across an element. Unfortunately, the Gaussian quadrature points are typically interior points which do not coincide with the nodal point locations. Therefore, if results must be output for nodal point locations it is suggested that the mesh be refined in areas where the solution derivatives are expected to vary significantly.

3 Command Reference

This chapter is included as a reference to the commands available for use in `Rexx` macros.

3.1 Problem Type

These commands specify the kind of problem to be modeled and the type of assumption used to restrict the problem to two dimensions.

`problem` *kind ndof title* [Command]

`problem` requires an argument specifying the *kind* of problem. The problem *kind* can be set to any desired value. This value is used to associate materials, boundary conditions and solutions (with the *kind* corresponding to the problem *kind*). The second argument (*ndof*) may be used to specify the number of degrees of freedom per node. For instance, in structural problems (in two dimensions) two displacements for each node are the degrees of freedom. In thermal (heat conduction) problems one temperature for each node is solved for. The final argument is used to set the problem *title*.

```
'problem 0 1 "Heat Transfer"'
```

`plane` [Command]

`plane` is used to specify a “plane” assumption (i.e. the problem solution is independent of the out-of-plane direction). There are no arguments to `plane`.

`axisym` [Command]

`axisym` is used to specify an axisymmetric assumption (i.e. the problem solution is independent of the circumferential direction). There are no arguments to `axisym`.

3.2 Materials

The following commands are used for specification of material properties.

`mat` *id type C*[*10*] *ckind f*[*2*] *ref ckind f*[*2*] *ref* [Command]

`mat` requires specification of the material identification number (*id*), material *type*, material coefficients (*C[10]*), and two sets of body force terms consisting of couple problem kind (*ckind*), body force coefficients (*f[2]*), and reference value (*ref*).

The material *type* currently has two possible meaningful values. If the value is set to zero (0) then the material properties are referenced to the global coordinate system. If the value is set to one (1) then the material properties are referenced to the local element coordinate system (see Section 3.3 [Geometry], page 14). The latter choice is useful for defining non-isotropic material properties whose principal orientation varies with element orientation (for example, a filament wound composite bottle) and is especially useful when used in conjunction with the `lam` command (see Section 3.2 [Materials - lam], page 13).

The couple problem kind may be specified in order to couple the solution from a previous analysis with the body force coefficients used in the current analysis. If no coupling is desired then these values should be set to the current problem kind (see Section 3.1 [Problem Type - problem], page 13). The body force coefficients are used to define properties associated with body forces. The reference value can be used to specify a reference temperature, species concentration, etc. A reference temperature, for example, could be used to determine a change in temperature due to heat conduction thereby coupling the solution obtained from a heat transfer analysis with a structural analysis.

The use of these values is dependent on the solver (see Section A.2 [Lin2D - Model Requirements], page 33) used to calculate a solution. Therefore, to determine what is required for input the documentation for the specific solver must be consulted.

```
'set la=a2*a3/((1+a3)*(1-2*a3))'
'set mu=a2/(2*(1+a3))'
'alias iso=mat(a1,0,la+2*mu,la+2*mu,la+2*mu,mu,la,la,0.0,la)'
'iso(1,10.e6,0.32)'
```

`lam` *id mid frac mang* [Command]

`lam` requires specification of the laminate identification number (*id*), reference material identification (*mid*), fraction of laminate (*frac*), and lamina material angle (*mang*).

This command is used to define a laminate by building up laminae. Each lamina is defined by a single instance of `lam`. Subsequent `lam` commands issued with the same laminate identification number will add laminae to any current laminate sequence. The reference material identification number must refer to an existing material. It is assumed that the reference material is at most a transversely isotropic material (the material properties are invariant only in the 2-3 plane). This also implies that the primary (1) direction is the principal material direction (the stiffer or fiber direction). The *frac* argument is used to specify the fraction that this lamina takes in the overall thickness of the laminate. The total sum of all the *frac* arguments for every lamina defined in the laminate should equal 1.0. The lamina material angle (*mang*) is used to define the fiber orientation (in the 1-3 material plane). This material angle may be defined as a constant value or as a function specified by name (previously defined by the `set` command, see Section 3.6 [Utility Commands - set], page 17). This capability to specify a function for the fiber angle can be used to effectively model filament wound composite pressure vessels with domes where the fiber angle varies with position along the dome.

```
'set fang1=asin(1.7365/r0)'
'set fang2=-fang1'
'lam 2 0 8 1/2 fang1'
'lam 2 0 8 1/2 fang2'
'lam 7 0 8 1/3  20.0'
'lam 7 0 8 1/3 -20.0'
'lam 7 0 8 1/3 90.0'
```

3.3 Geometry

`node` *id x y* [Command]

`node` will create a node located at a specified coordinate. The arguments are the node identification number (*id*) and the coordinate pair (*x*, *y*). The coordinate pair is interpreted depending on the coordinate setting (see Section 3.6 [Utility Commands - coord], page 17). If the coordinate is not given then the node will be placed at (0,0). If the identification number is not given (or is zero) the next largest node number will be used.

```
'node(1,0.5,1.0)'
```

`elem` *id mat nnodes nint nodes*[*9*] [Command]

`elem` is used to create elements based on isoparametric shape functions. The required arguments are the element identification number (*id*), material identification number (*mat*, referring to an existing material), number of nodes (*nnodes*), number of integration points (*nint*) and the node numbers (*nodes[9]*, also referring to existing nodes). The following are possible element specifications.

nnodes	nint	order	shape	integration rule	
3	1	linear	tri	1 point	
6	1	quadratic	tri	1 point	
6	3	quadratic	tri	3 point	
4	1	linear	quad	1 point	
4	4	linear	quad	2x2 point	
8	1	quadratic	quad	1 point	(not recommended)
8	4	quadratic	quad	2x2 point	
8	9	quadratic	quad	3x3 point	
9	1	quadratic	quad	1 point	(not recommended)
9	4	quadratic	quad	2x2 point	
9	9	quadratic	quad	3x3 point	

If the material number is not given then material one will be used. If the identification number is not given (or is zero) the next largest element number will be used. It should be pointed out that some combinations of element order and integration order result in element instabilities whose effect can be highly dependent on the problem being solved (see Section 2.4 [Element Instability], page 10, for more information).

```
'alias quad4=elem(a1,a2,4,4,a3,a4,a5,a6)'
'quad4(1,1,1,2,3,4)'
```

3.4 Boundary Conditions

The following commands allow specification of the boundary conditions. A sufficient specification of the boundary conditions is necessary to result in a solvable problem. If during the solve phase of the analysis an error occurs it is most likely due to insufficient or improper application of boundary conditions.

`pbc` *node type dir val* [Command]

`pbc` specifies boundary conditions applied at nodes. This boundary condition may be either a Dirichlet or flux type boundary condition. The first argument is the identification number for the node at which the boundary condition is applied (*node*). The second argument is used to specify the *type* of boundary condition. A Dirichlet boundary condition is *type* 1. A flux boundary condition is *type* 2. The next argument is used to specify the global direction (*dir*) for the boundary condition. For scalar boundary conditions use 1. For vector boundary conditions use 1 or 2 to specify the global direction (1 for `x` or `r`, 2 for `y` or `z`). The last argument is the boundary condition value (*val*). If not specified then the boundary condition will be removed (if it exists). The defined boundary condition will be associated with the currently set problem kind (see Section 3.1 [Problem Type - problem], page 13).

```
'alias dispx=pbc(a1,1,1,a2)'
'dispx(5,0.0)'
'alias forcr=pbc(a1,2,1,a2)'
'forcr(10,100.0)'
'alias temp=pbc(a1,1,1,a2)'
'temp(2,77)'
```

`dbc` *elem type dir side val*[*3*] [Command]

`dbc` specifies boundary conditions applied along element sides. This boundary condition may be either Dirichlet or flux *type* boundary conditions. The first argument is the identification number for the element (*elem*). The second argument is used to specify the *type* of boundary condition. A Dirichlet boundary condition is *type* 1. A flux boundary condition is *type* 2. The next argument is used to specify the local direction *dir* for the boundary condition. For scalar boundary conditions use 1. For vector boundary conditions use 1 or 2 to specify the local direction (1 for normal to element side, 2 for tangent to element side). The next argument is the element *side* number. The succeeding arguments specify the value of the boundary condition at each nodal location (*val[3]*). If one value is specified the boundary condition value is assumed to be constant across the element side. If two or more values are given the value will vary either linearly or quadraticly depending on the order of the element. If no values are specified then the boundary condition will be removed (if it exists). The defined boundary condition will be associated with the currently set problem kind (see Section 3.1 [Problem Type - problem], page 13).

```
'alias press=dbc(a1,2,1,a2,a3)'
'press(20,1,-1000.0)'
'alias slope=dbc(a1,1,1,a2,a3)'
'slope(12,3,0.0)'
```

3.5 Results Processing

These commands are included to evaluate results as specific locations as well as to select result functions for processing through an external post-processor (see Appendix C [Post2D], page 37).

`select` *function location* [Command]

Given a solution calculated by a solver (see Appendix A [Lin2D], page 33, and Appendix B [Nln2D], page 35) `select` will select functions to be subsequently used to post-process the solution (see Appendix C [Post2D], page 37). Any previously defined *function* (see Section 3.6 [Utility Commands - set], page 17) may be given in the first argument. The second argument indicates where the solution will be evaluated in each element (*location*). If `gauss` is specified then the solution will be processed at the Gaussian quadrature points. If `node` is specified then processing will occur at the node locations. Finally, if `ave` is specified then the solution will be processed at the nodes and averaged across element boundaries (unless it is a material boundary). If this argument is not given then `gauss` is assumed. More accurate stress/strain results (for that matter any result based on solution derivatives) can be expected using the `gauss` option. No results are actually calculated until the model is processed through the post-processor.

```
'set flux=-c1*(x1*x1+y1*y1)^0.5'
'select(flux,ave)'
```

`eval` *expr elem np xi yi* [Command]

Given a solution calculated by a solver (see Appendix A [Lin2D], page 33, and Appendix B [Nln2D], page 35) `eval` is used to evaluate the solution at specified locations. Any valid expression (see Section 3.6 [Utility Commands - set], page 17) may be given in the first argument (*expr*). The second argument, if given, specifies the element (*elem*) by order number (greater than or equal to zero) or by identification number (less than zero). The third argument (*np*) indicates either the local node number (greater than zero), the Gaussian quadrature point (less than zero) or if given as zero the next two arguments are used to specify the local coordinates (see Section 3.3 [Geometry], page 14). If the element number (and all subsequent arguments) are not given then the expression will be evaluated as a stand-alone expression (no solution values will be valid). In this way `eval` may be used as an online calculator (though somewhat limited). The computed value for the specified location and the expression given is returned in RESULT (assuming `options results` is used in the macro).

```
'eval(sig1,101,0,0.0,0.0)'
```

3.6 Utility Commands

These commands are included for additional capability.

`save` *file* [Command]

The complete model, problem type, material properties, geometry, boundary conditions, solutions, results, functions and aliases may be saved to a file using `save`. The file format used is an Interchange File Format (IFF) type developed specifically for this code. It is a binary file format and therefore cannot be edited conventionally. The only argument to this command is the name of the *file* to store the data.

```
'save(ram:model.dat)'
```

`read` *file* [Command]

The file produced by the `save` command (see Section 3.6 [Utility Commands - save], page 17) can be read in by `read`. The only argument to this command is the name of the *file* containing the data.

```
'read(results.dat)'
```

`coord` *system xorg yorg* [Command]

`coord` sets the coordinate *system* to either the cartesian or polar coordinate systems. It also allows specification of the coordinate system origin (*xorg*, *yorg*). `coord` affects all commands that accept or return coordinate data. It should be noted that even if a model is not inherently cylindrical many circular features can be modeled more effectively using the polar coordinate system (with an origin offset). The coordinate system may be changed at any point during the modeling process with no effect on previously issued commands.

```
'coord(polar,5,5)'
```

`reset` *flag function* [Command]

`reset` will selectively clear (deallocate) memory. If no arguments are given then all memory will be deallocated except for functions and aliases. This should always be done before starting a new problem. If `all` is specified for the *flag* then all memory is deallocated. Specifying `solutions`, `groups`, `functions` or `aliases` for the *flag* will deallocate the associated sections of memory. If `results` is specified for the *flag* then the next argument (if given) indicates the *function* name of the result to deallocate. If this argument is not given then all results are deallocated.

`check` [Command]

`check` will perform a check of model integrity. It will check for undefined nodes, elements and materials over elements and boundary conditions. These checks do not insure that the resulting system will be solvable, nor does it insure that element mapping is invertible (a necessary condition). However, if errors occur during formation of the global stiffness matrix that is generally an indication that the mesh is malformed, therefore, check the model for elements that are misshapen or misnumbered. Also, if errors occur in the solver check the boundary conditions for completeness (do not allow rigid body motion or rotation). Model statistics and memory requirements will also be output with this command. There are no arguments to `check`.

`info` [Command]

`info` gives information on the overall model and optionally on materials, nodes, elements, boundary conditions, solutions, groups and window settings. If no argument is given then the number of nodes, elements, groups and solutions is returned in the `Rexx` variable *RESULT* (assuming `options results` is used).

`info` "*mat*" *n* [Command]

If the first argument is `mat` then information is returned for the material specified by the second argument. A material is referenced by its identification number and by the currently specified problem kind (see Section 3.1 [Problem Type - problem], page 13). The data returned is the material order number, material id, type, kind, material coefficients (10), and two sets of body force terms consisting of the couple problem kind, body force coefficients (2), and reference value.

`info` "*node*" *n* [Command]

If the first argument is `node` then information is returned for the node specified by the second argument. A node can be referred to by its order number or assigned identification number (the number you originally assigned). In this way information can be obtained without knowledge of the original assigned numbering. If the order number is used then the argument must be positive (or zero since order numbers are zero based). Otherwise, negative numbers are interpreted as identification numbers (indicated by the absolute value). If *max* is specified for the second argument then information will be returned for the node with the maximum identification number.

If *last* is specified then information is returned for the node last defined. The data returned is the order number, node id, and the node x,y coordinates (referenced to the coordinate origin). If the polar coordinate system is set (see Section 3.6 [Utility Commands - coord], page 17) then the coordinate data returned will be in polar form (a radius and angle, referenced to the coordinate system origin).

`info` "*elem*" *n* [Command]

If the first argument is `elem` then information is returned for the element specified by the second argument. An element can be referred to by its order number or assigned identification number (the number you originally assigned, as above). If *max* is specified for the second argument then information will be returned for the element with the maximum identification number. If *last* is specified then information is returned for the element last defined. The data returned is the order number, element id, material id, number of nodes, and the node order numbers (not the node ids).

`info` "*pbc*" *dof n* [Command]

If the first argument is `pbc` then information is returned for the point boundary condition on the node specified by the second argument (see above) for the degree of freedom specified. The data returned is the node order number, node id, boundary condition type, kind, degree of freedom, and the boundary condition value. If no boundary conditions (associated with the current problem kind) exist for the node then only the node order number and node identification number are returned.

`info` "*dbc*" *dof n side* [Command]

If the first argument is `dbc` then information is returned for the distributed boundary condition on the element, side, and degree of freedom specified. The data returned is the element order number, element id, element side, boundary condition type, kind, degree of freedom, and the boundary condition values (one for each node along the element side). If no boundary conditions (associated with the current problem kind) exist for the element side then only the element order number, identification number, and side are returned.

`info` "*soln*" *dof n* [Command]

If the first argument is `soln` then a solution value is returned for the degree of freedom and node specified. The data returned is the node order number, node id, and solution value (associated with the current problem kind).

`info` "*result*" *name* [Command]

If the first argument is `result` then information on results is returned. The second argument (if specified) is the function name of the result. The data returned is the result kind, number of degrees of freedom, location flag, the minimum and maximum computed result values. If the function name is not specified then a list of result function names is returned. If any names are prepended with an `*` then that result has computed values otherwise the result function is only selected (ready for post-processing, see Appendix C [Post2D], page 37).

`info` "*functions*" [Command]

If the first argument is `function` then a list of defined function names is returned.

`info` "*group*" *name* [Command]

If the first argument is `group` then information on groups is returned. The second argument (if specified) is the name of the group. The data returned is the number of nodes contained in the group. If the group name is not specified then a list of group names is returned.

`info` "*window*" [Command]

If the first argument is `window` then data is returned containing the window maximum and minimum coordinates in the form of xmin, xmax, ymin, ymax (as specified in the `window` command).

`version` [Command]

`version` returns the version and the compilation date and time for `FEM2D`.

`stop` [Command]

`stop` will signals the code to exit thereby removing the code from memory. Any subsequently issued macros that reference this program will fail unless it is restarted. There are no arguments to `stop`.

`set` *function expr* [Command]

`set` is used to define functions used to post-process solution data and to store intermediate data for exchange between macros. Pre-defined variable names may be used in the function expression (*expr*) as well as any other defined *function* names. Material coefficients may be referenced by using the pre-defined variables, *c1* through *c10* (the global material properties), which refer to the coefficients as input in the `mat` command and *l1* through *l10* (the local lamina material properties) which refer to transformed coefficients referenced to the local element orientation. The latter variables will only be defined for laminates (defined by the `lam` command) and will vary depending on location within the laminate (the material angle used to transform the properties is obtained from the lamina at the specified location). Also, element and material angles may be used through the predefined variables *m1* and *m2*. The variable *m1* will contain the element principal orientation and *m2* will contain the lamina material angle (in degrees). Both of these angles will vary with location and are most useful for transforming results to coincide with these natural orientations (for example, they could be used to calculate the fiber stress in a composite laminate). Body force components may be accessed through the pre-defined variables, *f1* through *f4* (see Section 3.2 [Materials - mat], page 13). Additionally, the variables *p1* and *p2* are defined as the local parametric coordinates of the element and range from 0.0 to 1.0 (note that these coordinates may be used to interpolate within an element and only make sense when a function is evaluated within an element, see Section 3.5 [Results Processing - eval], page 16). The pre-defined variables for `plane` (see Section 3.1 [Problem Type - plane], page 13) problems are:

```
u# - solution value
x# - solution derivative with respect to 'x' or 'x' coordinate
y# - solution derivative with respect to 'y' or 'y' coordinate
```

The pre-defined variables for `axisym` (see Section 3.1 [Problem Type - axisym], page 13) problems are:

```
u# - solution value
```

```
r# - solution derivative with respect to 'r' or 'r' coordinate
z# - solution derivative with respect to 'z' or 'z' coordinate
t# - solution derivative with respect to 't'
```

Where the # symbol is replaced by the desired degree of freedom or is specified as zero (0) if coordinate values are desired.

The function expression (*expr*) may be any valid algebraic expression which can contain any combination of operators, pre-defined variables and other defined functions. The expressions may also contain parenthesized sub-expressions, nested to any level. The allowed operators are `+`, `-`, `*`, `/`, and `^` (real exponent). Also built-in functions `abs`, `sin`, `cos`, `tan`, `asin`, `acos`, and `atan` (which takes two arguments separated by a comma) are available as well as the variable `pi`. Note that all angle arguments for all functions and the return values from the `asin`, `acos`, and `atan` functions are in degrees.

The *function* name may be given as the first argument or as the left hand side of the expression separated by an `=`. The latter option should be used for functions while the previous option should be used for defining constants. Also, no parentheses should be used to delimit the arguments because of possible conflicts with parentheses used in functions.

```
'set tau12=(0.25*(sigr-sigz)^2+taurz^2)^0.5'
'set sigf=cos(m2)*cos(m2)*sigz+sin(m2)*sin(m2)*sigt'
'set material 2'
```

`get` *function* [Command]

`get` is used to retrieve a *function* or alias expression defined by the `set` (see Section 3.6 [Utility Commands - set], page 17) and `alias` (see Section 3.6 [Utility Commands - alias], page 17) commands. The argument is the *function* or alias name. The *function* or alias expression is returned in the `Rexx` variable *RESULT* (assuming that `options results` was specified in the macro).

```
'get(material)'
```

`group` [Command]

The `group` commands allow for manipulation of groups of nodes and associated elements. In general, groups are composed of nodes only, however, elements and boundary conditions associated with nodes contained in a group will be affected by group manipulation. Group names may be any length, but only the first nine characters are significant.

`group` *name* "*add*" *n1 n2* [Command]

The `add` option is used for group creation. If the group specified (*name*) does not exist then a new group is created otherwise the existing group will be added to. The two subsequent arguments specify the nodes (by identification number) to be added as a sequence of nodes starting with node *n1* and ending with node *n2*. If *n2* is given as zero then all nodes starting at *n1* up to the last node in the model will be added.

`group` *name* "*rem*" *n1 n2* [Command]

The `rem` option is used to remove nodes from an existing group (*name*). This command is not destructive (no nodes are deleted). The two subsequent arguments specify the nodes (by identification number) to be removed as a sequence of nodes starting with node *n1* and ending with node *n2*. If *n2* is given as zero then all nodes starting at *n1* up to the last node in the model will be removed.

`group` *name* `"`*tran*`"` *x1 y1 x2 y2* [Command]

The `tran` option is used to translate an existing group (*name*). The subsequent arguments specify the starting coordinate (*x1*, *y1*) and ending coordinate (*x1*, *y1*) for the translation. These coordinates are specified in either cartesian or polar form depending on the setting given in the `coord` command (see Section 3.6 [Utility Commands - coord], page 17). It should be noted that the meaning of "translation" is different depending on the coordinate form. In cartesian coordinates a translation is defined as a movement in the global `x` and `y` directions. In polar coordinates a translation is defined as a movement in the radial and angular directions. This effectively will result in rotation, translation and scaling in the most general use. This feature can be utilized with great effectiveness if used properly.

`group` *name* `"`*del*`"` [Command]

The `del` option is used to delete nodes contained in an existing group (*name*). This command is destructive (so beware). All nodes and associated elements and boundary conditions will be deleted from the model.

`group` *name* `"`*save*`"` *file* [Command]

The `save` option is used to save a group (*name*) to an IFF binary file (see Section 3.6 [Utility Commands - save], page 17). Only nodes and associated elements (elements whose nodes are all contained in the group) are saved.

`group` *name* `"`*read*`"` *file* [Command]

The `read` option is used to read in a group from an IFF binary file (see Section 3.6 [Utility Commands - read], page 17). If the named group (*name*) exists then the nodes (and elements) read in will be added to the group. If the group does not exist a new group will be created. The node (and element) identification numbers will start after the maximum identification number in the current model (in other words, no nodes or elements will be overwritten).

`group` *name* `"`*off*`"` [Command]

The `off` option is used to remove a group temporarily. If a group is "turned off" most subsequent element operations will ignore elements contained in this group. Additionally, results will not be calculated by the post-processor (see Appendix C [Post2D], page 37) for any of the associated elements.

`group` *name* `"`*on*`"` [Command]

The `on` option is used to return a group to the processing list. Subsequent element operations will now effect this group. Additionally, post-processing will result in values being calculated for the associated elements. Note that if results were calculated previously with the group "turned off" then no results are available for the affected group even if the group is "turned on". In this case, the result function would have to be reselected (see Section 3.5 [Results Processing - select], page 16) and the model post-processed (see Appendix C [Post2D], page 37).

`alias` *name expr* [Command]

`alias` is used to create aliases for built-in commands. In this manner `FEM2D` may be customized to many different applications. Aliases are defined through the use of the pre-defined variables *a1* through *a#*. These pre-defined variables contain the argument entries of the alias command. For example, the alias `iso` shown below will have its arguments placed in the pre-defined variables *a1* through *a3*. These arguments are then used in the alias definition as well as in functions used in the alias definition. In this case, an isotropic material definition command was created from the more complex `mat` command. In actual use the `iso` alias command might look like the following.

```
'set la=a2*a3/((1+a3)*(1-2*a3))'
'set mu=a2/(2*(1+a3))'
'alias iso=mat(a1,0,0,la+2*mu,la+2*mu,la+2*mu,mu,la,la,0.0,la)'
'iso(1,10.e6,0.3)'
```

Note that functions may be used in aliases with no restrictions (although some uses may not make any sense). Aliases, however, may not be used in other aliases or functions. Also note that functions containing the pre-defined variables *a1* through *a#* only make sense when used with aliases.

`why` [Command]

`why` will return the last error message associated with an error condition in the `Rexx` variable *RESULT* (assuming `options results` is used in the macro). If an error occurs the `Rexx` variable *RC* will contain a severity level. Refer to references on the `Rexx` language for details on error handling (see Chapter 5 [References], page 29). Upon receiving an error condition this command can be used to help determine what error actually occurred.

3.7 Graphic Commands

The following commands comprise the graphical functions builtin to `FEM2D`.

`plot` [Command]

`plot "`*model*`"` [Command]

`plot` will plot the nodes and elements of the model. If no arguments are specified then the model will be plotted.

`plot "`*deform*`"` *scale* [Command]

If `deform` is specified for the first argument then displacements (actually, any solution with two degrees of freedom will be treated as displacements by this command, so beware) are added to the nodal coordinates. Optionally, a third argument specifies a deformation *scale* which by default is 1 (giving actually deformation).

```
'plot(deform,2.0)'
```

`plot "`*bcs*`"` [Command]

If `bcs` is specified for the first argument then all boundary conditions are plotted.

```
'plot(bcs)'
```

`plot "`*bound*`"` [Command]

If `bound` is used then free boundaries (unconnected element edges) will be highlighted. The last option is useful for determining if any nodes need to be equivalenced (see Appendix D [Equ2D], page 39).

```
'plot(bound)'
```

`plot "`*result*`"` *fun comb rmin rmax* [Command]

If `result` is specified for the first argument then contour lines will be plotted with a color dependent on the results (given by the function name, *fun*) calculated by the post-processor (see Appendix C [Post2D], page 37). Additional arguments may be given to indicate how to combine the results within an element and to give the range of result values. Result combination (*comb*) may be specified as `max`, `min`, `ave` or an integer. These options will give either the maximum, minimum or average value of the results calculated within each element respectively. If an integer value (> 0) is given for *comb* then each element is subdivided into equal regions each of which are contoured according to the interpolated result at the center of the subregion. Since result interpolation is used in this option results must have been processed at the nodes (see Section 3.5 [Results Processing - select], page 16). If this argument is not specified then `max` is assumed. Finally, a range of values may be given by specifying the minimum (*rmin*) and maximum (*rmax*) result values. If not given then the range is set to include the the full range of values calculated by the post-processor.

```
'plot(result,temp,ave,0.0,100.0)'
'plot(result,sigd,4)'
```

`plot "`*vector*`"` *fun1 fun2 subd rmin rmax* [Command]

If `vector` is specified for the first argument then results will be plotted within each element. The first result function specified (*fun1*) is assumed to be a magnitude with the second (*fun2*) assumed to define a direction. The combination of these two results are plotted as vectors with corresponding magnitude (indicated by color) and direction. The specified result functions must have been previously processed (see Appendix C [Post2D], page 37). A direction result is assumed to have units of degrees (such as would be returned by the `atan` builtin function, see Section 3.6 [Utility Commands - set], page 17). If an element subdivision (*subd*) is given and the results were processed at the nodes (see Section 3.5 [Results Processing - select], page 16) then each element is subdivided into equal regions each of which will have a vector plotted at the centroid corresponding to the interpolated result at that location. If results were processed at the Gaussian quadrature points then vectors will only be plotted at those points within each element (and this argument will be ignored. If this argument is not specified then 0 is assumed. Finally, a range of values may be given by specifying the minimum (*rmin*) and maximum (*rmax*) result values. If not given then the range is set to include the the full range of values processed.

```
'set flux=-c1*(x1*x1+y1*y1)^0.5'
'set fdir=atan(y1,x1)'
        .
        .
        .
--- run Post2D ---
        .
        .
        .
'plot(vector,flux,fdir,2)'
```

`fill` [Command]
`fill "`*model*`"` [Command]

`fill` will plot the elements of the model and fill them with a color dependent on the material assigned to the element or results computed. If no arguments are specified then the model will be plotted.

`fill "`*deform*`"` *scale* [Command]

If `deform` is specified for the first argument then displacements (actually, any solution with two degrees of freedom will be treated as displacements by this command, so beware) are added to the nodal coordinates. Optionally, an additional argument specifies a deformation *scale* which by default is 1 (giving actually deformation).

```
'fill(deform,2.0)'
```

`fill "`*result*`"` *fun comb rmin rmax* [Command]

If `result` is specified for the first argument then the elements will be filled with a color dependent on the results (given by the function name, *fun*) calculated by the post-processor (see Appendix C [Post2D], page 37). Additional arguments may be given to indicate how to combine the results within an element and to give the range of result values. Result combination (*comb*) may be specified as `max`, `min`, `ave` or an integer. These options will give either the maximum, minimum or average value of the results calculated within each element respectively. If an integer value (> 0) is given for *comb* then each element is subdivided into equal regions each of which are filled according to the interpolated result at the center of the subregion. Since result interpolation is used in this option results must have been processed at the nodes (see Section 3.5 [Results Processing - select], page 16). If this argument is not specified then `max` is assumed. Finally, a range of values may be given by specifying the minimum (*rmin*) and maximum (*rmax*) result values. If not given then the range is set to include the the full range of values calculated by the post-processor.

```
'fill(result,temp,ave,0.0,100.0)'
'fill(result,sigd,4)'
```

`window` *xmin xmax ymin ymax* [Command]

`window` is used to specify world coordinates for the window. The arguments are the minimum x (*xmin*), maximum x (*xmax*), minimum y (*ymin*) and maximum y (*ymax*) coordinates in that order.

```
'window(-10.0,20.0,0.0,30.0)'
```

`find` [Command]

`find` will reset the window to include the full model. The model will be replotted. There are no arguments to `find`.

`zoom` *factor* [Command]

`zoom` will increase or decrease the window size dependent on the specified *factor*. A value of one will cause no change. Values less than one will shrink the window (zoom in) whereas values greater than one will expand the window (zoom out).

```
'zoom(0.5)'
```

`move` *xfract yfract* [Command]

`move` is used to move the window along the x and y directions. The arguments specify the x and y move in terms of window view fraction (*xfract*, *yfract*). Values of zero cause no change for that direction. A value of one will cause a shift of a full window view width or height (depending on the direction specified) in the positive direction (right or up).

```
'move(-.5,0.0)'
```

`clear` [Command]

`clear` will clear the window view. There are no arguments to `clear`.

`show` [Command]
`show "`*node*`"` *n flag* [Command]
`show "`*elem*`"` *n flag* [Command]
`show "`*group*`"` *name* [Command]

`show` will plot (in complementary mode) the requested item(s). If *flag* is set to `1` then the corresponding node or element identification number (*n*) will be plotted next to the item. If the `group` option is specified all nodes in the group (specified by *name*) will be plotted.

```
'show(group,part)'
```

`hard` *file* [Command]

`hard` initiates and terminates hardcopy output. When first issued a *file* will be opened for writing hardcopy output. Subsequent plot (see Section 3.7 [Graphic Commands - plot], page 23) and fill (see Section 3.7 [Graphic Commands - fill], page 23) commands issued will result in hardcopy output. Color Postscript commands are used as the output format and should result in a complete Postscript file readable by any Postscript capable software. If `hard` is issued again, hardcopy output will be closed allowing for multiple hardcopy invocations in any session.

```
'hard(ram:model.ps)'
```

`color` *id R G B* [Command]

`color` allows for the specification of the output color spectrum. The first argument (*id*) is the identification number of the color to be changed (1-12 inclusive). The subsequent arguments (*R*, *G*, *B*) are integers between the values 0 and 15 (inclusive).

```
'color(4,10,5,8)'
```

4 Example Problem

The following macro is included as a simple example to illustrate the use of an `Rexx` macro to generate a model. The advantages of using an `Rexx` macro are ease of debugging and the use of parametric variation to name a couple. This macro can be executed as any `Rexx` macro and will produce a model file ready to be solved by `Lin2D`. `FEM2D` is automatically invoked, by the macro, if it is not already running.

```
/* simple beam */
options results

address fem2d

h = 0.5
L = 10.0
E = 10.0e6
nu = 0.3000

reset
problem 'struct'
plane
iso 1 E nu
do i=0 to 20
    xx = L*i/20
    node i+1 xx 0.0
    node i+31 xx h
end
do i=0 to 19
    quad4 i+1 1 i+1 i+2 i+32 i+31
end
dispx 1 0.0
dispy 1 0.0
dispx 31 0.0
dispy 31 0.0
forcy 51 (-100.0)
check
say result
save 'beam.dat'
exit
```

5 References

1. Hawes, William S., *Rexx User's Reference Manual*
2. Becker, Eric B., Carey, Graham F., Oden, J. Tinsley, *Finite Elements, An Introduction*, Volume 1, Prentice-Hall Inc., 1981
3. Zienkiewicz, O.C., Taylor, R.L., *The Finite Element Method*, Volumes 1 & 2, Fourth Edition, McGraw-Hill Book Company, 1991
4. Schildt, H., *C, The Complete Reference*, Second Edition, Osborne McGraw-Hill, 1990

6 Acknowledgments

I would like to acknowledge Dr. Eric Becker and Dr. Linda Hayes, of the Aerospace Engineering and Engineering Mechanics Department of The University of Texas at Austin, for passing on the knowledge of the finite element method directly and indirectly through course work and research assistance. It is this knowledge that made possible the development of `Lin2D` and `Nln2D`. In fact, both codes started out as course term projects directed by Dr. Hayes and Dr. Becker.

Appendix A Lin2D

A.1 Introduction to Lin2D

`Lin2D` is a general linear solver developed to handle "plane" and axisymmetric two dimensional finite element analysis problems. It is a fairly general capability in the sense that a wide variety of two dimensional problems can be handled by `Lin2D` limited only by computational resources. Specific attention was given to efficient memory and resource management, however, the minimum recommended amount of memory needed to solve moderate problems, using `Lin2D`, is 6 megabytes (although simple problems can be run with considerably less). A hard disk is also recommended.

It is important to note that any engineer, planning to use a finite element code for actual application, understand fully the concepts and implementation of the finite element method. Therefore, the prospective user should consider an in-depth study of the method and ideally some training. This document should not be considered appropriate for training users in the use of the finite element method.

The main features of `Lin2D` are as follows:

- runs as a background task
- two-dimensional plane or axisymmetric problems
- linear and quadratic elements available
- no limit on number of nodes, elements, materials, or boundary conditions (only limited by memory)
- no need to specify problem size (determined at run time)
- choice of Dirichlet (constraint) or Neumann (flux), point or distributed boundary conditions
- symmetric banded matrix storage format

In general the command line syntax for executing `Lin2D`:

```
        -> [run] Lin2D [file]
```

where *file* is the name of an input file to be processed.

A.2 Model Requirements

The constitutive law implemented in `Lin2D` (relating stress and strain in structural problems, for example) consists of a linear matrix operator with up to ten independent constants. These constants are input in the `mat` command in `FEM2D` as the material coefficients. For structural problems the components of stress and strain, relevant to plane or axisymmetric problems, are the three normal components plus the in-plane shear component. This requires that the material matrix operator be a four by four matrix. This results in sixteen constants but, due to elastic symmetry, only ten of these constants are independent. The material coefficients allow for specification of materials ranging from isotropic (properties invariant to material direction) to monoclinic (properties symmetric about one plane). The coefficients are specified in diagonal order and the matrix is symmetric. In other words, if the coefficients are arranged in matrix form they would appear as follows.

```
C0  C4  C7  C9
C4  C1  C5  C8
C7  C5  C2  C6
C9  C8  C6  C3
```

Body forces are also input with the `mat` command. `Lin2D` treats the two sets of body forces differently. The first set of body force terms are for body forces arising from internal energy. For example, thermal strain in a material develops as a result of a change in temperature in the material which can be treated as a body force due to a change in internal energy. In this case the body force (thermal strain) is computed through the following equation:

```
eT[] = f1[]*(Ti - Tref)
```

where `f1[]` is the first set of body force coefficients (thermal coefficients of expansion, one for each geometric degree of freedom), `Tref` is the reference value (temperature), and `Ti` is the computed value (temperature found from a previous heat transfer analysis).

The second set of body force terms are for body forces arising from potential energy such as conservative force fields (gravity or acceleration being an example). Furthermore, for axisymmetric problems the radial component of this conservative body force is multiplied by the radial location. This, for example, allows for a radial force component that depends on the radial acceleration arising from the steady state rotation of an axisymmetric body. In this case the body forces are computed through the following equations:

```
F[r] = f2[r]*r^2*density
F[a] = f2[a]*r*density
```

where `f2[r]` is the first (radial) body force coefficient in the second set (representing the angular velocity), `f2[a]` is the second (axial) coefficient (which represents an axial acceleration), `r` is the radial location, and `density` may be either a constant (input by specifying a negative reference value) or a computed value (obtained from a previous diffusion analysis and setting the reference value to zero).

`formkf` forms the stiffness matrix and load vector. The stiffness matrix and load vector are assembled from the individual element contributions which results in a linear system of equations to be subsequently solved. The only argument to this command is a print flag (*iprint*). A value of 1 indicates that the stiffness matrix and load vector will be printed to the standard output. This is not recommended for problems resulting in large systems. A value of 0 will result in no matrix/vector output (the default setting).

`solve` will attempt to solve the system assembled using the `formkf` command. Simple checks of the model are performed at various stages, but this does not insure that the system can be solved. The user must pay special attention to correctness of model geometry and boundary conditions. If the system is not solvable it is most probably due to errors in the geometry and/or boundary conditions. The only argument to this command is a print flag (*iprint*). A value of 1 indicates that the solution vector will be printed to the standard output. This is not recommended for large problems. A value of 0 will result in no solution output (the default setting).

Appendix B Nln2D

This solver is equivalent to `Lin2D` except that it is restricted – results are only valid for – structural analysis problems where large displacements (and/or rotations) are expected, but small strain theory still applies. A good example would be the large deflection of a thin cantilever beam. The functional difference between `Nln2D` and `Lin2D` is the solution process. The first iteration through `formkf` and `solve` will result in a linear solution (equivalent to the solution given by `Lin2D`). Subsequent executions of `formkf` and `solve` will produce results that represent a lower total energy state (the internal energy approaches the external energy of the system). After “enough” iterations have been performed the total energy should be zero (that is an equilibrium state should have been reached). The command `formkf` returns the maximum (absolute value) nodal residual force in the system. This value can be checked against a tolerance value (typically around 0.01, but depends on loading conditions) as a stopping criteria. Also, the `solve` command returns the maximum (absolute value) change in nodal displacement. This value may also be checked against a tolerance value (typically around 0.0001) as a stopping criteria. The following is an example script showing a solution procedure.

```
/* example script for nln2d */
options results

address nln2d 'read(ram:temp.dat)'

do i=0 to 20
    address nln2d 'formkf'
    address nln2d 'solve'
    parse var result du
    if du < 1e-4 then leave
end

address nln2d 'save(ram:temp.dat)'
exit
```

This code can be more effectively used when run in conjunction with `FEM2D`. For instance, intermediate results can be viewed graphically thereby given a visual indication of solution convergence. It is even possible (and often desirable) to assemble an animation consisting of these intermediate results. This animation would show, very effectively, the convergence process.

Also, convergence of the solution may require incremental loading which can be accomplished with the aid of `FEM2D`. This would be done by applying a fraction of the appropriate load(s), performing an iterative solution with `Nln2D` until convergence, then increasing the load fraction using `FEM2D` and repeating the solution procedure. This whole process is repeated until the full load is reached.

Currently, only a linear elastic stress-strain material law is implemented thereby restricting problems to those where small strains are expected. However, future additional material constitutive laws will be implemented thereby extending the code applicability to more general nonlinear problems.

Appendix C Post2D

`Post2D` is a general two dimensional finite element solution post-processor. In general the command line syntax for executing `Post2D`:

```
-> [run] Post2D [file]
```

where *file* is the name of an input file to be processed.

Given a solution calculated by a solver (see Appendix A [Lin2D], page 33, and Appendix B [Nln2D], page 35) `Post2D` will post-process the solution for all selected (see Section 3.5 [Results Processing - select], page 16) functions (see Section 3.6 [Utility Commands - set], page 17) over the complete model.

Appendix D Equ2D

Given a two-dimensional finite element mesh (stored in the IFF format currently used by `FEM2D`) `Equ2D` will equivalence nodes that fall within a specified tolerance apart. Either an inclusion or exclusion node group may be given to restrict the equivalence operation.

The equivalence operation is typically used as a modeling tool to "weld" separate parts together or to close gaps.

The command line syntax for `Equ2D` is:

```
-> [run] equ2d [-igroup] [-xgroup] [-ttol] <input> [output]
```

where *input* is the input data file containing the finite element data (in the form produced by the save command in `FEM2D`). This input data file is a required argument. The output data file, if specified, will be the name of the file that will contain the equivalenced finite element data. If not specified then the data will be written back to the input file (thereby overwriting the previous data). If the `-i` option is used then *group* specifies the group containing the nodes to be included in the equivalence search. Likewise, if the `-x` option is used then *group* specifies the group containing nodes to be excluded from the equivalence search. If the `-t` option is given then *tol* specifies the tolerance to be used to determine equivalence (the default is 1.0e-5).

Appendix E Opt2D

`Opt2D` is a node order optimization program for arbitrary two dimensional finite element meshes. The algorithm used to optimize the mesh is fairly simplistic but may result in a smaller bandwidth for some problems. For simple rectangular meshes, if the node ordering is consistent, then `Opt2D` will probably give a node ordering with a bandwidth that is larger than the original. However, for irregular meshes, very high aspect ratio meshes, or inconsistently ordered meshes, `Opt2D` can reduce the bandwidth significantly. A side effect of `Opt2D` is that all unreferenced nodes will be removed resulting in a further reduction in model size and memory requirements.

The command line syntax for `Opt2D` is:

```
-> [run] opt2d [-nelement] <input> [output]
```

where *input* is the input data file containing the finite element data (in the form produced by the save command in `FEM2D`). This input data file is a required argument. The output data file, if specified, will be the name of the file that will contain the optimized finite element data. If not specified then the optimized data will be written back to the input file (thereby overwriting the previous data). If the `-n` option is used then *element* specifies the identification number of the element where the optimization search is to begin. Proper selection of the starting element can dramatically improve the resulting optimization.

Appendix F Rem2D

`Rem2D` is a simple code to remove all unreferenced nodes from a model resulting in a reduction in model size and memory requirements without a reordering of the node numbers.

The command line syntax for `Rem2D` is:

```
-> [run] rem2d <input> [output]
```

where *input* is the input data file containing the finite element data (in the form produced by the save command in `FEM2D`). This input data file is a required argument. The output data file, if specified, will be the name of the file that will contain the optimized finite element data. If not specified then the optimized data will be written back to the input file (thereby overwriting the previous data).

Concept Index

Command Index

Table of Contents