FEAP - - A Finite Element Analysis Program

Version 8.4 Programmer Manual

Robert L. Taylor
Department of Civil and Environmental Engineering
University of California at Berkeley
Berkeley, California 94720-1710
E-Mail: rlt@ce.berkeley.edu

Revised June 2014
# Contents

1 Introduction ........................................ 1
   1.1 Setting Program Options ............................ 1
   1.2 Uses of Common and Include Statements ............. 3

2 Data Input and Output ............................... 5
   2.1 Parameters and Expressions .......................... 5
   2.2 Array Outputs .................................... 7

3 Allocating Arrays ................................. 8

4 User Functions ..................................... 15
   4.1 Mesh Input Functions - UMESHn .................... 15
      4.1.1 Command line TX data ........................ 17
   4.2 User Material Models ............................. 18
      4.2.1 The UMATIn Module .......................... 20
      4.2.2 The UMATLn Module .......................... 20
      4.2.3 Auto time step control ........................ 25
   4.3 Mesh Manipulation Functions - UMANIn ............. 27
   4.4 Solution Command Functions - UMACRn ............... 29
   4.5 Plot Command Functions - UPLOTn .................. 29

5 Adding Elements ................................... 32
   5.1 Material property storage .......................... 36
   5.2 Non-linear Transient Solution Forms ................. 37
   5.3 Example: 2-Node Truss Element ...................... 47
      5.3.1 Theory for a Truss ........................... 47
   5.4 Additional Options in Elements ..................... 51
      5.4.1 Task 1 Options ............................... 51
      5.4.2 Task 6 Options ............................... 53
   5.5 Projection of element variables to nodes .......... 54
   5.6 Elements with History Variables ................... 55
      5.6.1 Assigning amount of storage for each element ... 55
      5.6.2 Accessing history data for each element ...... 56
CONTENTS

5.7 Elements with Finite Rotation Parameters .................................. 57
5.7.1 Nodal rotation treatment: \texttt{UROTmm} subprogram .................. 58
5.7.2 Local nodal rotation treatment ............................................. 59
5.8 Energy Computation ............................................................... 60
5.9 A Non-linear Theory for a Truss .............................................. 61

6 Utility routines ........................................................................... 70
6.1 Numerical quadrature routines .................................................. 70
6.1.1 One dimensional quadrature .................................................. 70
6.1.2 Two dimensional quadrature ................................................... 71
6.1.3 Three dimensional quadrature .............................................. 72
6.2 Shape function subprograms ..................................................... 73
6.2.1 Shape functions in one-dimension ........................................... 74
6.2.2 Shape functions in two-dimensions ........................................ 75
6.2.3 Shape functions in three-dimensions ...................................... 78
6.3 Eigenvalues for $3 \times 3$ matrix .................................................. 80
6.4 Plot routines .............................................................................. 81
6.4.1 Mesh plots ............................................................................ 81
6.4.2 Element data plots ............................................................... 83
6.4.3 Other user plots .................................................................... 85

7 Adding a user solver .................................................................... 87
## List of Figures

4.1 Sample **UMESHn** module ........................................... 16  
4.2 Sample **UMESH** module ........................................... 19  
4.3 Sample **UMATI1** module ........................................... 21  
4.4 Sample **UMATLn** module for small deformation ................. 24  
4.5 Sample **UMATLn** module for finite deformation ................. 26  
4.6 Sample **UMANLn** module ........................................... 28  
4.7 Sample **UMACRN** module .......................................... 29  
4.8 Sample **UPLOTn** module ........................................... 30  

5.1 **FEAP** Element Subprogram. Part 1 ................................. 32  
5.1 **FEAP** Element Subprogram. Part 2 ................................. 33  
5.2 **FEAP** Element Common Blocks .................................... 64  
5.3 **FEAP** Element Common Blocks using Includes .................. 65  
5.4 2-Node Truss Element .................................................. 65  
5.5 Element residual for two node truss .................................. 66  
5.6 Truss Tangent Matrix. Part 1 ......................................... 67  
5.7 Truss Tangent Matrix. Part 2 ......................................... 68  
5.8 Element variable projection routine .................................. 69  

6.1 Line type elements in **FEAP** library .............................. 74  
6.2 Triangular surface type elements in **FEAP** library ............... 76  
6.3 Quadrilateral surface type elements in **FEAP** library ............ 77  
6.4 Tetrahedron solid type elements in **FEAP** library ............... 78  
6.5 Brick solid type elements in **FEAP** library ........................ 79
List of Tables

3.1 Mesh Array Names, Numbers and Sizes ........................................... 11
3.2 Solution Array Names, Numbers and Sizes ..................................... 11
3.3 Element Array Names, Numbers and Sizes ...................................... 12
3.4 Element connection array IX use for element e .............................. 12
3.5 Element control array IE use for material number ma ..................... 13
3.6 Element degree of freedom assignment array IEDOF use for material num-
ber ma .................................................................................................. 13

4.1 Color Table for Plots ................................................................. 31

5.1 Arguments of $FEAP$ Element Subprogram. ................................. 34
5.2 Task Options for $FEAP$ Element Subprogram. R = Required; O = 
Optional; E = For eigensolutions ..................................................... 35
5.3 $FEAP$ common block definitions. ................................................ 37
5.4 $FEAP$ common block definitions. ................................................ 38
5.5 Material Parameters ................................................................. 39
5.6 Material Parameters ................................................................. 40
5.7 Material Parameters ................................................................. 41
5.8 Material Parameters ................................................................. 42
5.9 Material Parameters ................................................................. 43
5.10 Tangent Parameters .................................................................... 47
5.11 Element Plot Definition Subprograms .......................................... 53
5.12 Momenta and Energy Assignments .............................................. 60

6.1 Quadrature for triangles ............................................................. 72
6.2 Quadrature for tetrahedra ............................................................. 73
6.3 Color pallet for $FEAP$ plots ........................................................ 84
6.4 Values for control of plots .......................................................... 84
Chapter 1

INTRODUCTION

In this part of the FEAP manual some of the options to extend the capabilities of the program are described. We begin by describing the utilities provided in FEAP for use in data input. Options to add user commands for mesh and command language extensions is then described and finally the method to add an element to the program is described.

1.1 Setting Program Options

The size of problems which may be solved by FEAP depends on the amount of memory available in the computer, as well as, solution options used. Memory for the main arrays used to solve problems is dynamically allocated during the solution. Arrays are allocated and deallocated using a system subprogram PALLOC or, for user developed modules using subprogram UALLOC. Further information on use of these routines is given in Section 3.

The IPR parameter in the feap84.f module controls the specification of the ratio of REAL to INTEGER variables. For typical UNIX and PC systems all real variables should be twice as large as integers and IPR is set to 2. For systems in which INTEGER*8 variables are used (set by compiler option) the IPR parameter is set to 1. Any error in setting this parameter may lead to incorrect behavior of the program, consequently, do not reset the parameter unless a careful assessment of compiler behavior has been made.

Normally FEAP reads each input data line as text data and checks each character for the presence of parameters, expressions, and constants. For very large data sets this parsing of each instruction can consume several seconds of compute time. If all data is normally provided as numerical data, without use of any parameters or expressions,
the input time may be reduced by setting the value of the logical variable COFLG in feap84.f to false. FEAP will automatically switch to parsing mode if any record contains non-numerical data item. It is also possible to use the PARSe and NOPArse commands to set the appropriate mode of data input.

In Windows versions it is sometimes desirable to obtain the input file name from a pop-up menu. This is accomplished by setting the parameter CIFLG to true.

During the input of plot commands FEAP has the option to either set input options automatically (DEFAULT mode) or to read the values or range of contours to plot. The default mode of operation may be assigned in the feap84.f module by setting the variables DEFAULT and PROMPT. Setting DEFAULT to true indicates that all default options are to be set automatically. If DEFAULT is set false, a prompt for contour intervals may be requested by setting PROMPT to true.

FEAP has options to produce encapsulated PostScript output files in either gray scale or in color. The default mode may be established by setting the variable PSCOLR and PSREVS. Setting PSCOLR true indicates the PostScript files will be in color (unless set otherwise by the PLOT COLOR data command. The PSREVS variable reverses the color sequence.

The last parameter which may be set in the feap84.f module is the level for displaying available commands when the HELP command is used while in mesh, solution, or plot mode. FEAP contains a large number of commands which are not commonly used by many users. To control the default number of commands displayed to users the commands have been separated into four levels: (0) Basic; (1) Intermediate; (2) Advanced; and (3) Expert. The level to be displayed when using the HELP command is given may be set in the integer variable HLPLEV. That is, setting:

\[
\text{hlplev} = 1 \quad ! \text{Intermediate}
\]

results in commands up to the intermediate level being displayed. It is possible to raise or lower the level during execution using the command \text{MANUal,,level} where level is the numerical value desired.

When developing program modules it is often desirable to have output of specific quantities available (e.g. tracking the change in some parameters during successive iterations. FEAP provides for a switch to make the outputs active or inactive during an execution. The switch is named debug and placed in

\[
\begin{align*}
\text{integer} & \quad \text{ndebug} \\
\text{logical} & \quad \text{debug} \\
\text{common /} & \quad \text{/debugs/ ndebug,debug}
\end{align*}
\]
The value of the debug is set true by the solution command `DEBUG` and false by the command `DEBUG,OFF`. Thus, placing code fragments into modules as

```fortran
if(debug) then
    write(iow,*) 'LABEL',list ... ! writes to output file
    c and/or
    write(*,*) 'LABEL',list ... ! writes to screen
endif ! debug
```

This device supplements use of available debuggers on the computer.

### 1.2 Uses of Common and Include Statements

`FEAP` contains many `COMMON` statements that are used to pass parameters and small array values between subprograms. For example, access to the debugging parameter `debug` is facilitated through `common /debugs/`. Users may either place the common statement (as well as data typing statements) directly in the routine or may use an include statement. For debugging the statement would be

```fortran
include 'debugs.h'
```

which during compilation would direct the precompiler to load the current common statement from this file. In `FEAP` all include files have the same name as the common with an added extender `.h`. For example, the common file name `comblk.h` is defined as

```fortran
real*8 hr
integer mr
common /comblk/ hr(1024),mr(1024)
```

The arrays `hr(1024)` and `mr(1024)` serve to pass all dynamically allocated arrays between subprograms using a pointer array contained in the common array named `np(*)` [or for user defined arrays in `up(*)`] located in the include file `pointer.h`. \(^1\) See Section 3 for more details on use of pointers. All include files are located in the directories

\(^1\) The values 1024 are necessary to ensure loops on arrays using pointers directly are considered as long.
include and include/integer4 or include/integer8. The subdirectory integer4
defines 32-bit integer parameters and integer8 defines 64-bit integer parameters.

It is highly recommended that users use include files rather than giving equivalent
common statements directly. If later releases of the FEAP program revise contents in
a common block, it will only be necessary to recompile the user routine rather than
change all the common statement definitions. Also, by defining the correct path in
the makefile.in on your compiler it is not necessary to modify any routine when
switching from 32-bit machines to 64-bit machines.
Chapter 2

DATA INPUT AND OUTPUT

FEAP includes utilities to perform input and to output small arrays of data. Users are strongly encouraged to use the input utilities but often may wish to use their own utilities to output data.

2.1 Parameters and Expressions

The subroutines PINPUT and TINPUT are input subprograms used by FEAP to input each data record. They permit the data to be in a free form format with up to 255 characters on each record, as well as to employ expressions, parameters, and numerical representations for each data item. These routines also should be used to input data in any new program module developed. The PINPUT routine returns data to the calling subprogram in a double precision array. The following statements may be included as part of the routine performing the input.

```fortran
subroutine xxx(.....)
   logical errck, pinput
   include 'iofile.h' ! ior,iow,ilg unit numbers
   real*8 td(5)
1   if(ior.lt.0) write(*,3000)
      errck = pinput(td, 5)
      if(errck) go to 1
```

The parameters defined in the include file (common block) are:
ior - input file unit number (if negative, input from keyboard)
iow - output file unit number
ilg - solution log file unit number

If an error occurs during input from the keyboard FEAP returns a value of true for the function and a user may reinput the record if the implied loop shown above is used. For inputs from a file, the program will stop and an error message indicating the type of error occurring and the location in an input file is written to the output file.

The input routines return data in a real*8 array td(*). If any td(i) is to be used as an integer or real*4 quantity, it must be cast to the correct type. That is, the following operations should be used to properly cast the variable type:

```
real*4 t
real*8 td(5)
integer j
logical errck, pinput

errck = pinput (td, 5)

j = nint( td(1)) ! Integer assignment
t = float(td(2)) ! Real*4 assignment
```

PINPUT may be used to input up to 16 individual expressions on one input record (each input record is, however, limited to 255 characters).

The routine TINPUT differs from PINPUT by permitting text data to also be input. It is useful for writing user commands or to input data described by character arrays. The routine is used as

```
logical errck, tinput
integer nt, nn
character text(16)*16
real*8 td(16)

errck = tinput(text,nt,td,nn)
```

The parameter nt specifies the number of text values to input and the nn specifies the number of real data values to input. The value for parameter nt or nn may be zero. Thus the use of

```
errck = tinput(text,0,td,nn)
```
is equivalent to

\[
\text{errck} = \text{pinput}(\text{td}, \text{nn})
\]

Text variables may be converted to numerical (\texttt{REAL*8}) form using the subroutine call

\[
\text{call setval(text, nc, td)}
\]

where \texttt{text} is a string with \texttt{nc} characters and \texttt{td} a \texttt{REAL*8} variable. The text string can contain any parameters, expressions or numerical constants which evaluate to a single value.

\section*{2.2 Array Outputs}

Two subprograms exist to output arrays of integer and real (double precision) data. The routine \texttt{MPRINT} is used to output real data and is accessed by the statement:

\[
\text{call mprint( array, nrow, ncol, ndim, label)}
\]

where \texttt{array} is the name of the array to print, \texttt{nrow} and \texttt{ncol} are the number of rows and columns to output, \texttt{ndim} is the first dimension on the array, and \texttt{label} is a character label which is added to the output. For example the statements:

\[
\text{real*8 aa(8,6)}
\]

\[
\text{call mprint( aa(2,4), 2, 3, 8, 'AA')}
\]

outputs a $2 \times 3$ submatrix from the array \texttt{aa} starting with the entry \texttt{aa(2,4)}. The output entries will be ordered as the terms:

\[
\begin{align*}
\text{aa(2,4)} & \quad \text{aa(2,5)} & \quad \text{aa(2,6)} \\
\text{aa(3,4)} & \quad \text{aa(3,5)} & \quad \text{aa(3,6)}
\end{align*}
\]

The \texttt{MPRINT} routine adds row and column labels as well as the character label.

The routine \texttt{IPRINT} is used to output integer data and is accessed by the statement:

\[
\text{call iprint( array, nrow, ncol, ndim, label)}
\]

where all parameters are identical to those for \texttt{MPRINT} except the array must be of type integer.
Dynamic data allocation is accomplished in *FEAP* by defining addresses in pointers contained in the common block defined in `pointer.h`. This common block contains pointers `np` for standard program arrays and `up` for user defined arrays and has the form:

```
integer num_nps , num_ups
parameter (num_nps = 400 , num_ups = 200)
integer np , up
common /pointer/ np(num_nps) , up(num_ups)
```

Each pointer is an offset relative to the address of a `REAL*8` array `hr(1)` or an `INTEGER` array `mr(1)` defined in a blank common:

```
real*8 hr
integer mr
common /comblk/ hr(1024),mr(1024)
```

which is placed in the file `comblk.h` in the `include/integer4` or `include/integer8` directory. The pointers in the `integer4` subdirectory have 32-bit lengths and should be used for cases where addressing is within 4 GByte range. The pointers in the `integer8` subdirectory have 64-bit lengths and should be used for additional index ranges. *The arrays 'hr' and 'mr' are used to establish addresses only and not to physically store data.* This mechanism permits references to elements in arrays which have positions relative to `hr` or `mr` that may be after or before 1. Thus, *FEAP* must be compiled without strict array bound checking. Size of problems is limited only by the available memory in the computer used.
When using 64-bit pointers users must be careful to always define the address of an array in a calling statement to also be 64-bits in length. For example use of

```plaintext
integer ioff
...
ioff = np(111) + numnp
call submat( hr(ioff), ...)
```

would cause an error since the pointer `ioff` is only 32 bits in length. To avoid this problem it is necessary to either declare `ioff` to be 64-bits long as

```plaintext
integer*8 ioff
```

or use one of the `FEAP` include files `p_int.h` (defining the integer type array `fp(10)`) or `p_point.h` (defining the integer type scalar `point`). A type of correct length is controlled at compile time by which subdirectory is used (i.e., `integer4` or `integer8`).

Using this scheme permits direct reference to either `real*8` or `integer` arrays in program modules without need to pass arrays through arguments of subprograms. A subprogram `PALLOC` controls the allocation of all standard arrays in `FEAP` defined by the `np` pointers and a subprogram `UALLOC` permits users to add allocation for their own arrays defined by the pointers `up`. The basic use of the routines is provided by an instruction

```plaintext
setvar = palloc(number,'NAME',length,precision)
```
or

```plaintext
setvar = ualloc(number,'NAME',length,precision)
```

where `setvar`, `palloc` and `ualloc` are logical types, `number` is an integer number of the array, `NAME` is a 5 character name of the array, `length` is the number of words of storage needed for the array, and `precision` is the type of array to allocate (1 for `integer` and 2 for `real*8` types). Upon initial assignment of any array its values are set to zero. Thus, if the array is to be used only once it need not be set to zero before accumulating additional values. If the array is to be reused or resized (see below) it must be reinitialized prior to accumulating any additional values. Use of these subprograms controls the assignment of memory space for all arrays such that no
conflicts occur between \texttt{hr} and \texttt{mr} referenced arrays. Each routine which makes direct reference to an allocated array using a pointer (e.g., \texttt{hr(np(43))} or \texttt{mr(up(1))}) must contain include files as

\begin{verbatim}
include 'pointer.h'
include 'comblk.h'
\end{verbatim}

As an example for the use of the above allocation scheme consider a case where it is desired to allocate a real (double precision array) with length \texttt{NUMNP} (number of nodes in mesh) and an integer array with length \texttt{NUMEL} (number of elements in mesh). The parameters \texttt{NUMNP} and \texttt{NUMEL} are contained in \texttt{COMMON /CDATA/} and available using the include file \texttt{cdata.h}. The new arrays are defined using the temporary names \texttt{TEMP1} and \texttt{TEMP2} which have numerical locations ‘111’ and ‘112’, respectively.\footnote{See the subprogram \texttt{palloc.f} in the \texttt{program} directory for the names and numbers of existing arrays.} The two arrays are allocated using the statements

\begin{verbatim}
setvar = palloc( 111, 'TEMP1', numnp, 2 )
setvar = palloc( 112, 'TEMP2', numel, 1 )
\end{verbatim}

where the last entry indicates whether the array is \texttt{REAL*8} (2) or \texttt{INTEGER} (1). These arrays are now available in any subprogram by specifying the \texttt{pointer.h} and \texttt{comblk.h} include files and referencing the arrays using their pointers, e.g., in a subroutine call as:

\begin{verbatim}
include 'pointer.h'
include 'comblk.h'
...
call subname ( hr(np(111)) , mr(np(112)) .... )
\end{verbatim}

Note the use of \texttt{hr(*)} and \texttt{mr(*)} for the double precision and integer references, respectively. Also, the use of the pointers avoids a need to include the array reference until it is needed in a computation.

A short list of the mesh arrays available in \texttt{FEAP} is given in Table 3.1, for solution arrays in Table 3.2, and for element arrays in Table 3.3. The names of all active arrays in any analysis may be obtained using the \texttt{SHOW,DICTionary} solution command.

The array \texttt{IX(nen1,numel)} is used to store basic information for each element in the mesh related to the nodal connections and material data requirements. In addition,
CHAPTER 3. ALLOCATING ARRAYS

<table>
<thead>
<tr>
<th>NAME</th>
<th>Num.</th>
<th>dim 1</th>
<th>dim 2</th>
<th>dim 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANG</td>
<td>45</td>
<td>numnp</td>
<td>-</td>
<td>-</td>
<td>Angle</td>
</tr>
<tr>
<td>D</td>
<td>25</td>
<td>ndd</td>
<td>nummat</td>
<td>-</td>
<td>Material parameters</td>
</tr>
<tr>
<td>F</td>
<td>27</td>
<td>ndf</td>
<td>numnp</td>
<td>2</td>
<td>Force and Displacement</td>
</tr>
<tr>
<td>ID</td>
<td>31</td>
<td>ndf</td>
<td>numnp</td>
<td>2</td>
<td>Equation nos.</td>
</tr>
<tr>
<td>IE</td>
<td>32</td>
<td>nie</td>
<td>nummat</td>
<td>-</td>
<td>Element control, dofs, etc.</td>
</tr>
<tr>
<td>IX</td>
<td>33</td>
<td>nen1</td>
<td>numel</td>
<td>-</td>
<td>Element connections</td>
</tr>
<tr>
<td>T</td>
<td>38</td>
<td>numpn</td>
<td>-</td>
<td>-</td>
<td>Temperature</td>
</tr>
<tr>
<td>U</td>
<td>40</td>
<td>ndf</td>
<td>numpn</td>
<td>3</td>
<td>Solution array</td>
</tr>
<tr>
<td>VEL</td>
<td>42</td>
<td>ndf</td>
<td>numpn</td>
<td>nt</td>
<td>Solution rate array</td>
</tr>
<tr>
<td>X</td>
<td>43</td>
<td>ndm</td>
<td>numpn</td>
<td>-</td>
<td>Coordinates</td>
</tr>
</tbody>
</table>

Table 3.1: Mesh Array Names, Numbers and Sizes

<table>
<thead>
<tr>
<th>NAME</th>
<th>Num.</th>
<th>dim 1</th>
<th>dim 2</th>
<th>dim 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMASn</td>
<td>n+8</td>
<td>compro</td>
<td>-</td>
<td>-</td>
<td>Consistent Mass</td>
</tr>
<tr>
<td>DAMPn</td>
<td>n+16</td>
<td>compro</td>
<td>-</td>
<td>-</td>
<td>Damping</td>
</tr>
<tr>
<td>JPn</td>
<td>n+20</td>
<td>neq</td>
<td>-</td>
<td>-</td>
<td>Profile pointer</td>
</tr>
<tr>
<td>LMASn</td>
<td>n+12</td>
<td>neq</td>
<td>-</td>
<td>-</td>
<td>Lump Mass</td>
</tr>
<tr>
<td>TANGn</td>
<td>n</td>
<td>maxpro</td>
<td>-</td>
<td>-</td>
<td>Symmetric tangent</td>
</tr>
<tr>
<td>UTANn</td>
<td>n+4</td>
<td>maxpro</td>
<td>-</td>
<td>-</td>
<td>Unsymmetric tangent</td>
</tr>
</tbody>
</table>

Table 3.2: Solution Array Names, Numbers and Sizes

arrays IE and IEDOF define additional information required to process each element. Tables 3.4, 3.5 and 3.6 describe the use of individual entries in the arrays IX, IE, and IEDOF, respectively.

The subprograms PALLOC and UALLOC may also be used to destroy a previously defined array. This is achieved when the length of the array is specified as zero (0). For example, to destroy the arrays defined as TEMP1 and TEMP2 the statements

```c
setvar = palloc( 111, 'TEMP1', 0, 2 )
setvar = palloc( 112, 'TEMP2', 0, 1 )
```

are given. Use of these statements results in the pointers np(111) and np(112) being set to zero and the space used by the arrays being released for use by other allocations at a later point in the program.

A call to PALLOC or UALLOC for any previously defined array but with a different non-zero length causes the size of the array to be either increased or decreased.
### Chapter 3. Allocating Arrays

#### Table 3.3: Element Array Names, Numbers and Sizes

<table>
<thead>
<tr>
<th>NAME</th>
<th>Num.</th>
<th>dim 1</th>
<th>dim 2</th>
<th>dim 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGL</td>
<td>46</td>
<td>nen</td>
<td>-</td>
<td>-</td>
<td>Angle</td>
</tr>
<tr>
<td>LD</td>
<td>35</td>
<td>nst</td>
<td>-</td>
<td>-</td>
<td>Assembly nos.</td>
</tr>
<tr>
<td>P</td>
<td>35</td>
<td>nst</td>
<td>-</td>
<td>-</td>
<td>Element vector</td>
</tr>
<tr>
<td>S</td>
<td>36</td>
<td>nst</td>
<td>nst</td>
<td>-</td>
<td>Element matrix</td>
</tr>
<tr>
<td>TL</td>
<td>39</td>
<td>nen</td>
<td>-</td>
<td>-</td>
<td>Temperature</td>
</tr>
<tr>
<td>UL</td>
<td>41</td>
<td>ndf</td>
<td>nen</td>
<td>6</td>
<td>Solution array</td>
</tr>
<tr>
<td>XL</td>
<td>44</td>
<td>ndm</td>
<td>nen</td>
<td>-</td>
<td>Coordinates</td>
</tr>
</tbody>
</table>

#### Table 3.4: Element connection array IX use for element e

<table>
<thead>
<tr>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX( 1 ,e)</td>
<td>Global node 1</td>
</tr>
<tr>
<td>...</td>
<td>to</td>
</tr>
<tr>
<td>IX(nen ,e)</td>
<td>Global node nen</td>
</tr>
<tr>
<td>IX(nen+1,e)</td>
<td>H1 history data pointer</td>
</tr>
<tr>
<td>IX(nen+2,e)</td>
<td>H2 history data pointer</td>
</tr>
<tr>
<td>IX(nen+3,e)</td>
<td>H3 history data pointer</td>
</tr>
<tr>
<td>IX(nen+4,e)</td>
<td>Lagrange multiplier tag</td>
</tr>
<tr>
<td>IX(nen+5,e)</td>
<td>Lagrange multiplier data pointer</td>
</tr>
<tr>
<td>IX(nen+6,e)</td>
<td>Time integrator: 0=implicit; &gt;0=explicit</td>
</tr>
<tr>
<td>IX(nen+7,e)</td>
<td>Element type: FE ≤ 0; IGA &gt; 0</td>
</tr>
<tr>
<td>IX(nen1 ,e)</td>
<td>Element material type number</td>
</tr>
<tr>
<td>IX(nen1-1,e)</td>
<td>Element region number (default = 0); Active region &gt;0; Inactive region &lt;0</td>
</tr>
<tr>
<td>IX(nen1-2,e)</td>
<td>Active/deactive start</td>
</tr>
<tr>
<td>NAME</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IE(1,ma)</td>
<td>Plot shape dimension (0,1,2,3); 0 = no plot, 1 = line; 2 = surface; 3 = solid.</td>
</tr>
<tr>
<td>IE(2,ma)</td>
<td>Rigid material number.</td>
</tr>
<tr>
<td>IE(nie, ma)</td>
<td>Number history variables/element (NH1 and NH2).</td>
</tr>
<tr>
<td>IE(nie-1, ma)</td>
<td>Element material type number (ELMT01 = 1, etc.).</td>
</tr>
<tr>
<td>IE(nie-2, ma)</td>
<td>Element material type identifier (default = ma).</td>
</tr>
<tr>
<td>IE(nie-3, ma)</td>
<td>Offset to NH1/2 history variables (default = 0).</td>
</tr>
<tr>
<td>IE(nie-4, ma)</td>
<td>Offset to NH3 history variables (default = 0).</td>
</tr>
<tr>
<td>IE(nie-5, ma)</td>
<td>Number history variables/element (NH3).</td>
</tr>
<tr>
<td>IE(nie-6, ma)</td>
<td>Finite rotation update number (for PROTxx or UROTxx).</td>
</tr>
<tr>
<td>IE(nie-7, ma)</td>
<td>Get tangent from element if 0; if &gt; 0 numerically differentiate residual to obtain tangent.</td>
</tr>
<tr>
<td>IE(nie-8, ma)</td>
<td>Equation number for element Lagrange multiplier.</td>
</tr>
<tr>
<td>IE(nie-9, ma)</td>
<td>Partition number for element Lagrange multiplier.</td>
</tr>
<tr>
<td>IE(nie-10, ma)</td>
<td>Global equation number.</td>
</tr>
</tbody>
</table>

Table 3.5: Element control array IE use for material number ma

<table>
<thead>
<tr>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEDOF(1,i,ma)</td>
<td>Degree of freedom 1 for node i of material ma.</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>IEDOF(ndf,i,ma)</td>
<td>Degree of freedom ndf for node i of material.</td>
</tr>
</tbody>
</table>

Table 3.6: Element degree of freedom assignment array IEDOF use for material number ma
For user defined arrays specified in UALLOC care should be exercised in selecting the alphanumeric NAME parameter, which is limited to 5 characters, so that conflicts are not created with existing names (use of the SHOW,DICT command is one way to investigate names of arrays used in an analysis) or check the names already contained in the subprogram PALLOC.

The subroutine PGETD also may be used to retrieve internal data arrays by NAME for use in user developed modules. For example, if a development requires the nodal coordinate data the call

\begin{verbatim}
integer xpoint, xlen, xpre
logical flag
... call pgetd ('X ',xpoint,xlen,xpre,flag)
\end{verbatim}

will return the first word address in memory for the coordinates as xpoint, the length of the array as xlen, and the precision of the array as xpre. If the retrieval is successful flag is returned as true, whereas if the array is not found it is false. The precision will be either one (1) or two (2) for INTEGER or double precision (REAL*8) quantities, respectively. Thus, the above coordinate call will return xpre as 2 and xlen will be the product of the space dimension of the mesh and the total number of nodes in the mesh. The first coordinate, x1, may be given as

\begin{verbatim}
x1 = hr(xpoint)
\end{verbatim}

any other coordinates at nodes may also be recovered by a correct positioning in later words of hr. For example y1 is located at hr(xpoint+1). The use of pgetd can lead to errors for situations in which the length of arrays changes during execution, since in these cases the value of the pointer xpoint can change. For such cases a call to pgetd must be made prior to each reference involving xpoint. On the other hand, reference using the pointers defined in arrays NP or UP are adjusted each time an array changes size. However, users must ensure that a calling sequence is not sensitive to a change in pointer. One way pointer changes can still lead to errors is through a program

\begin{verbatim}
call subname ( hr(np(111)), mr(np(112)), ....)
\end{verbatim}

and then change the length of the array number ‘111’ or ‘112’ in the subroutine.
Chapter 4

USER FUNCTIONS

Users may add their own procedures to facilitate additional mesh input features, to perform transformations or manipulations on mesh data, to add new solution commands, or to add new plot capabilities.

4.1 Mesh Input Functions - UMESHn.

To add a mesh input command a subprogram with the name UMESHn, where n has a value between 0 and 9 must be written, compiled, and linked with the program. The basic structure of the routine UMESH1 is:

The parameter TX is a character array which is assigned by the input and UPRT is a logical parameter which is set to false when the NOPRint mesh command is given and to true when the PRINT command is used (default is true). The common block UMAC1 transfers the character variable UCT to assign the name of the command. The default name is MESn where n is the same as the routine name number. Assignment of a unique character name (which must not conflict with names already assigned for mesh input commands) should be used to replace the xxxx shown.

When FEAP begins execution it scans all of the UMESHn routines and replaces the command names mes1, etc., by the user furnished names. Thus, when the command HELP is issued while in interactive MESH mode, the user name will appear in the list instead of the default name (note, FEAP does not always display all available commands. To see all commands issue the command MANUal,3 and then the HELP command).
subroutine umesh1(tx, uprt)

C------[--.--------.--------.-------------------------------]
C Purpose: User defined routine to input mesh data to FEAP
C
C Inputs:
C   tx(*) - Command line input parameter name
C   uprt  - Flag, Output results if true
C
C Outputs:
C   none  - Users responsible for outputs to arrays, etc.
C------[--.--------.--------.-------------------------------]
implicit none

include 'umac1.h' ! Contains UCT variable
character tx(*)*15
logical uprt

C Set name 'mes1' to user defined
if(pcomp(uct,'mes1',4)) then
  uct = 'xxxx' ! Set user defined command name
else
C User execution function statements follow
end if
end

Figure 4.1: Sample UMESHn module
The ability to get array names as shown in Chapter 3 can be used to develop user routines for input of coordinates, element connections, etc. With this facility it is possible to develop an ability to directly input data prepared by other programs which may be in a format which is not compatible with the requirements of standard FEAP mesh commands.

4.1.1 Command line TX data

It is possible to include up to 8 data items on the command line for user functions. All the data is passed to the UMESHn functions by the character array TX(*)*15 and may be used to control actions in the function. If the information is of type character it may be used directly, however, if it is numeric it must be converted within the UMESHn function before any additional input statements are processed. For example if a user input function has the command line:

\texttt{GETData VALUes 35}

is developed in the user function UMESH1 the first argument GETData must match the name assigned to UCT and will also be in TX(1). The second parameter will be in TX(2) and the third in TX(3). To recover the numerical value for the third parameter the statement

\begin{verbatim}
real*8   ctl \\
... \\
call setval(tx(3),15, ctl)
\end{verbatim}

may be used to assign the real value 35.0d0 to ctl. If necessary, the real value for ctl can be cast into an integer using

\begin{verbatim}
   it1 = nint(ctl)
\end{verbatim}

If more than 8 items are desired on the input line it is possible to recover their values from the character string yyy*256 which has been parsed into columns with width 15 characters. Note that the total number of added words must be 15 items or less (this is imposed by the total of 16 items on any FEAP input record). To recover their values the statements
are added to the user function and the items recovered in the else option of the function using the statements:

\[
\begin{align*}
lct(1) &= yyy(16:30) \\
\text{call setval}(yyy(31:45),15, rtl(1))
\end{align*}
\]

would assign \(lct(1)\) values from the second set of 15 characters and \(rtl(1)\) to the third set of 15 characters. In this case \(lct(1) = tx(2)\) and \(rtl(1)\) would have the same value as \(ctl\) above.

If users wish to add more than 10 material models it is possible to use the user function \texttt{UMESH}\ which has the form

### 4.2 User Material Models

Users may add material models to elements by appending subprograms \texttt{UMATIn} and \texttt{UMATLn} (where \(n\) have values from 0 to 9) to the \texttt{FEAP} system. The subprogram \texttt{UMATIn} defines parameters used by the model and the subprogram \texttt{UMATLn} is called by the element for each computation point (i.e., the quadrature point), receives the value of a deformation measure as input and must return the value of stress and tangent moduli as output.

To activate a user material model the input data for the mesh \texttt{MATErial} command must include a statement with \texttt{UCON} as the first field. For example in a solid element the command sequence can be

\[
\begin{align*}
\text{MATErial } ma \\
\text{SOLID} \\
\text{UCONstitutive } xxxx \ v1 \ v2 \ ...
\end{align*}
\]

The role of the \(xxxx\) and \(v1\) data will be described in Section 4.2.1.

It is possible to use standard input parameters defined in Tables 5.5 to 5.8, as well as by preceding the \texttt{UCON} command with a normal input sequence. For example, if isotropic elastic properties are needed they may be included in the input sequence as
logical function umesh(cc,tx,prt)

c-----[--.----+.----+.----+.---------------------------------------]

Purpose: User mesh command interface

c Inputs:
c cc - User command option
c tx(*) - Command line input data
c prt - Output if true

c Outputs:
c none - Data stored by user development

implicit none

logical  prt,pcomp
character cc*4, tx(*)*15

c Match on 'USER': Add as many checks as desired with 'user'

if(pcomp(cc,'xxxx',4)) then  ! Provide name for 'xxxx'
    umesh = .true.  ! Activate command
        ......
    endif(........
endif
end

Figure 4.2: Sample UMESH module
MATerial ma
SOLID
  ELAStic ISOTropic e nu
UCONstitutive xxxx v1 v2 ...

No standard commands should follow the UCON command.

Alternatively, users may input elastic properties as part of their UMATIn module. If the user routine does input additional data records (after the UCON record) and these are terminated by a blank record, a second blank record will be needed to discontinue material data input for this set. In all cases at least one blank record is always needed to terminate the input of standard options for the material set.

4.2.1 The UMATIn Module

A sample module for a user constitutive model is shown in Fig. 4.3. As shown in this figure, the UMATIn module has 5 arguments. The name of the constitutive equation to be described is passed in the first parameter \texttt{type}. The second parameter passes an array (vv(*)) which may be used to define up to 5 parameters for the material model. The example shown above for the UCON includes the \texttt{type} data as \texttt{xxxx} and the array vv(*) values as v1 v2 .... Users may also provide additional input within the UMATIn module using the routines PINPUT or TINPUT described in Sect. 2.1. The values of user parameters must be saved in the array ud(*) (the fifth argument of UMATIn). In the current version there are 150 words of double precision values available by default. Additional values may be allocated by assigning a larger value on the control record (first record after the FEAP title record). Each material model is assigned a user material number to the return parameter umat. This number must be a positive integer. Finally, the number of history parameters to be assigned to each computation (quadrature) point must be returned in the parameter n1. Currently, the parameter n3 may be set but is not available to the user material model. Thus, all history variables must be retained in the n1 list. Use of history variables is described later as part of the UMODEL module.

4.2.2 The UMATLn Module

A sample for the UMATL1 module with arguments defined for small deformation is shown in Fig. 4.4 and for arguments defined for finite deformation in Fig. 4.5. This subprogram will be called by many of the elements included within FEAP if a user model
subroutine umati1(type, vv, d, ud, n1, n3)

c-----[--.----+----.----+----.-------------------------------------]

c Purpose: User material model interface

c Inputs:
  type - Name of constitutive model (character variable)
  vv(*) - Parameters: user parameters from command line
  d(*) - Program material parameter data

Outputs:
  n1 - Number history terms: nh1,nh2
  n3 - Number history terms: nh3
  ud(*) - User material parameters

c-----[--.----+----.----+----.-------------------------------------]

implicit none

include 'iofile.h'

logical pcomp

character type*15

integer n1, n3, nh1, nh2, nh3

real*8 vv(5), d(*), ud(*)


c Specify type of user model

if(pcomp(type,'mat1',4)) then
  type = 'E-1d' ! Specify new name for model
end if


c Input/output user data and save in ud(*) array

else

  c Set values of 'n1' if required
  n1 = ...

  write(iow,*) ' User Constitutive Inputs: E = ', vv(1)

  ud(1) = vv(1) ! Parameter from input on command name
end if

end

Figure 4.3: Sample UMAT1 module
has been specified as part of the MATE mesh data (see previous subsection). The user model will not be called for truss, frame, plate, and shell elements which use resultant models to describe behavior. Also, any form which requires a one-dimensional model will not use a UMATLn module. The module is designed to compute three-dimensional constitutive models in which the stress and strain are stored as 6-component vectors and the tangent moduli as a $6 \times 6$ matrix.

**Small deformation models**

For small deformation models the strains are passed to UMATLn in the argument array $\text{eps}(6)$ and stored in the order

$$
\epsilon = \begin{bmatrix}
\epsilon_{11} & \epsilon_{22} & \epsilon_{33} & \gamma_{12} & \gamma_{23} & \gamma_{31}
\end{bmatrix}^T
$$

where $\gamma_{ij} = 2\epsilon_{ij}$ is the engineering shearing strain. Stress and moduli are to be associated with the same ordering and returned in the argument arrays dimensioned as $\text{sig}(6)$ and $\text{dd}(6,6)$, respectively. All real values are in double precision (i.e., REAL*8).

When UMATLn is called the model $n$ will be that which is defined in the module UMATIn. Current values of the strains are, as mentioned above, passed in the array $\text{eps}(6)$ and the trace of the strain in the parameter $\theta$. Thus,

$$
\theta = \epsilon_{ii} = \epsilon_{11} + \epsilon_{22} + \epsilon_{33}.
$$

In addition, if thermal problems are being solved the current value for the temperature is passed as $\text{td}$. All material parameters for the current model are passed in the arrays $\text{d}(\ast)$ and $\text{ud}(\ast)$. The array $\text{d}(\ast)$ contains parameters assigned by standard FEAP commands as described in Tables 5.5 to 5.8 and the array $\text{ud}(\ast)$ contains values as assigned in the user module UMATIn.

Other values for use in writing material models can be obtained from parameters in common blocks. For models which depend on position in the body the values of the reference and current coordinates for the constitutive point are passed in common block elcoor which contains the values in

```plaintext
real*8      xref,xcur
common /elcoor/ xref(3),xcur(3)
```

For models that may need to use an incremental formulation with

$$
\Delta \epsilon = \epsilon_{n+1} - \epsilon_n
$$
the array for strains may be dimensioned as $\text{eps}(9,2)$ where the first 6 entries of $\text{eps}(9,1)$ store the strains at $t_{n+1}$ and the first 6 entries of $\text{eps}(9,2)$ store those at $t_n$. The extra entries are not defined as they are provided only for use in the finite deformation form of the model described next.

**Finite deformation models**

For finite deformation models the deformation gradient is passed to $\text{UMATLn}$ in the argument array $f(3,3,4)$ where $f(3,3,1)$ defines $F_{n+1}$, $f(3,3,2)$ defines $F_n$, $f(3,3,3)$ defines $G_{n+1}$ and $f(3,3,4)$ defines $G_n$. The deformation gradient is stored as

\[
\begin{align*}
    f(i,J,1) &= F_{iJ}(t_{n+1}) \\
    f(i,J,2) &= F_{iJ}(t_n) \\
    f(i,J,3) &= G_{iJ}(t_{n+1}) = F_{iJ}(t_{n+1}) - \delta_{iJ} \quad \text{and} \\
    f(i,J,4) &= G_{iJ}(t_n = F_{iJ}(t_n) - \delta_{iJ})
\end{align*}
\]

where $G_{iJ}$ are displacement gradients. Stress and moduli are to be returned in the argument arrays dimensioned as $\text{sig}(6)$ and $\text{dd}(6,6)$, respectively. Stresses and moduli are returned using Voigt notation where stresses are ordered as

\[
\sigma = \begin{bmatrix}
    \sigma_{11} & \sigma_{22} & \sigma_{33} & \sigma_{12} & \sigma_{23} & \sigma_{31}
\end{bmatrix}^T
\]

with corresponding order for the moduli. All real values are in double precision (i.e., REAL*8).

When $\text{UMATLn}$ is called the model $n$ will be that which is defined in the module $\text{UMATIn}$. Current values of the deformation gradient are, as mentioned above, passed in the array $f(3,3,4)$ and the determinant of the deformation gradient in the parameter $\text{theta}(4)$ where

\[
\theta_1 = \det F_{n+1} \quad \text{and} \quad \theta_2 = \det F_n
\]

In addition

\[
\theta_3 = \det F_{n+1} - 1 \quad \text{and} \quad \theta_4 = \det F_n - 1
\]

If thermal problems are being solved the current value for the temperature is passed as $\text{td}$. All material parameters for the current model are passed in the arrays $\text{d}(\ast)$ and $\text{ud}(\ast)$. The array $\text{d}(\ast)$ contains parameters assigned by standard $\text{FEAP}$ commands as described in Tables 5.5 to 5.8 and the array $\text{ud}(\ast)$ contains values as assigned in the user module $\text{UMATIn}$.

Other values for use in writing material models can be obtained from parameters in common blocks. For models which depend on position in the body the values of the
subroutine umat1(eps, theta, td, d, ud, hn, h1, nh, ii, istrt, sig, dd, isw)

c-----[--.----+----.----+----.-----------------------------------------]

   Purpose: User Constitutive Model

   Input:
   eps(*) - Current strains at point
   theta - Trace of strain at point
   td - Temperature change
   d(*) - Program material parameters
   ud(*) - User material parameters
   hn(nh) - History terms at point: t_n
   h1(nh) - History terms at point: t_n+1
   nh - Number of history terms
   ii - Current point number
   istrt - Start state: 0 = elastic; 1 = last solution
   isw - Solution option from element

   Output:
   sig(6) - Stresses at point.
   dd(6,6) - Current material tangent moduli

   implicit none

   integer nh, ii, istrt, isw, i
   real*8 td
   real*8 eps(*), theta(*), d(*), ud(*), hn(nh), h1(nh), sig(6), dd(6,6)

   c Dummy model: sig = ud(1)*eps

   if(isw.eq.14) the ! Set initial values for history parameters
       ! .... None needed for this model
   c Compute tangent and stress
   else
      do i = 1,6
         dd(i,i) = ud(1)
         sig(i) = ud(1)*eps(i)
      end do
   endif

end

Figure 4.4: Sample UMATLn module for small deformation
reference and current coordinates for the constitutive point are passed in common block `elcoor` which contains the values in

```
real*8 xref, xcur
common /elcoor/ xref(3), xcur(3)
```

Internal variable storage and use

For constitutive equations with additional (internal) variables that evolve in time, users must define entries for the `h1(*)` array. The number of entries available in the array for each evaluation (i.e., each quadrature point) is `nh`. The value for `nh` is defined by the parameter `n1` in module `UMATIn` (see Fig. 4.3). Values from the previous time step are passed back to the module in the array `hn(*)` (which also contains `nh` entries). Users should never modify entries in the `hn(*)` array. Finally, the values of the element operation switch is passed as the parameter `isw` (See Chapter 5 for operations performed during different values of `isw`).

Using the above information users must compute values for the stress and the associated tangent matrix. These are returned to the element in the arrays `sig(6)` and `dd(6,6)`. In addition, updates for any of the history parameters must be assigned in the array `h1(*)` and returned to the element. Values of history variables returned are not used for all values of `isw` (e.g., when reporting or projecting stresses under `isw = 4` and `isw = 8` they are not saved). Values retained in the `h1(*)` array are copied to the `hn(*)` array each time the command statement `TIME` is issued in a solution.

### 4.2.3 Auto time step control

The solution command:

```
AUTO MATERial rvalu(1) rvalu(2) rvalu(3)
```

initiates an attempt to control the solution process by a variable time stepping algorithm based on a user set value in the material constitution. The value to be set is named `rmeas` which is passed between constitution and solution modules in the labeled common

```
real*8 rmeas, rvalu
```
subroutine umatl1(f, detf, td, d, ud, hn, h1, nh, ii, istrt, sig, dd, isw)

c-----[--.----+----.----+----.-------------------------------------]
c Purpose: User Constitutive Model

c Input:
c f(3,3,*) - Deformation gradient (finite deformation)
c detf(*) - Determinant of deformation gradient
c td - Temperature change
c d(*) - Program material parameters
c ud(*) - User material parameters
c hn(nh) - History terms at point: t_n
c h1(nh) - History terms at point: t_{n+1}
c nh - Number of history terms
c ii - Current point number
c istrt - Start state: 0 = elastic; 1 = last solution
c isw - Solution option from element

c Output:

c sig(6) - Stresses at point.
c dd(6,6) - Current material tangent moduli

c-----[--.----+----.----+----.-------------------------------------]
implicit none

integer nh, istrt, ii, isw
real*8 td
real*8 f(3,3,*) detf(*), d(*), ud(*), hn(nh), h1(nh)
real*8 sig(6), dd(6,6)

if(isw.eq.14) then  ! Set any initial values for history
else
  ! Compute model tangent and stress
  ...
endif
end

Figure 4.5: Sample UMATLn module for finite deformation
The three parameters may be used in defining an acceptable value for \( r_{\text{meas}} \). The algorithm coded monitors the solution during a standard iteration process set by, for example:

\[
\text{LOOP,,n} \\
\quad \text{TANG,,1} \\
\text{NEXT}
\]

If during any iteration up to \( n \) the value of \( r_{\text{meas}} \) exceeds a value of 2 (\( r_{\text{meas}} = 0 \) at the start of the loop) a new value of \( \Delta t \) is immediately set to

\[
\Delta t_{\text{new}} = 0.85 \Delta t/r_{\text{meas}}
\]

and the iteration process is started over. On the other hand if convergence occurs during the time step and the value of \( r_{\text{meas}} \) is smaller than 1.25, the time step is adjusted according to

\[
\begin{align*}
\Delta t_{\text{new}} &= 1.50 \Delta t ; \quad r_{\text{meas}} \leq 0.5 \\
\Delta t_{\text{new}} &= 1.25 \Delta t ; \quad 0.5 < r_{\text{meas}} \leq 0.8 \\
\Delta t_{\text{new}} &= \Delta t/r_{\text{meas}} ; \quad 0.8 < r_{\text{meas}}
\end{align*}
\]

Finally, if convergence does not occur within the \( n \) steps, then the time step is reset according to

\[
\begin{align*}
\Delta t_{\text{new}} &= 0.85 \Delta t/r_{\text{meas}} ; \quad 1.25 < r_{\text{meas}} \\
\Delta t_{\text{new}} &= \Delta t/3 ; \quad \text{otherwise.}
\end{align*}
\]

After any of the above adjustments the value of \( r_{\text{meas}} \) is reset to zero (0).

An optimal value of \( r_{\text{meas}} \) is 1.25 – which leaves the step unchanged. The above algorithm was proposed by Weber et al. [1].

### 4.3 Mesh Manipulation Functions - UMAnIn.

The UMAnIn modules, where \( n \) ranges from 0 to 9, may be used to perform transformations or manipulations on previously prescribed data. These commands appear between the mesh input END command and the first INTeractive or BATCh solution command. To add a mesh manipulation command a subprogram with the name UMAnIn,
subroutine umani1

c User defined routine to manipulate mesh data for FEAP

implicit none

include 'umac1.h' ! Contains UCT variable

c Set name 'man1' to user defined

if(pcomp(uct,'man1',4)) then
    uct = 'xxxx' ! Set user defined command name
else

    c User execution function statements follow

    end if

end

Figure 4.6: Sample UMANLn module

where n has a value between 0 and 9 must be written, compiled, and linked with the program. The basic structure of the routine UMANI1 is:

The common block UMAC1 transfers the character variable UCT for the name of the command. The default names are MANn where n is the same as the routine name number. Assignment of a unique character name (which must not conflict with names already assigned for mesh input commands) should be used to replace the xxxx shown.

After FEAP completes the input of mesh data it scans all of the UMANIn routines and replaces the command names man1, etc., by the user furnished names.

The ability to get array names as shown in Chapter 3 can be used to develop user routines for manipulation of the mesh data. For example, if a user has added the specification of information by coordinates it may later be necessary to associate the data with specific node numbers. This can be accomplished using a manipulation command which searches for the node number whose coordinates are closest to the specified location.
subroutine umacr0(lct,ctl)
c  User solution command function
implicit none
include 'umac1.h' ! Contains the variable UCT
character lct*15
real*8  ctl(3)
c  Set command word
if(pcomp(uct,'mac0',4)) then
   uct = 'xxxx'
else
   User command statements are placed here
endif
end

Figure 4.7: Sample UMACRn module

4.4 Solution Command Functions - UMACRn.

In a similar manner, users may add solution commands to the program by adding a routine with the name UMACRn where n ranges from 0 to 9.

The parameters LCT and CTL are used to pass the second word of a solution command and the three parameter values read, respectively. Again the name xxxx should be selected to not conflict with existing solution command names and will appear whenever HELP is issued.

4.5 Plot Command Functions - UPL0Tn.

In a similar manner, users may add new plot commands to the program by adding a routine with the name UPL0Tn where n ranges from 0 to 9.

The parameters CTL(3) are used to pass the three parameter values read, respectively. Again the name xxxx should be selected to not conflict with existing plot command names and will appear whenever HELP is issued.
subroutine uplot0(ctl)
c User plot command function
implicit none
include 'umac1.h' ! Contains the variable UCT
real*8 ctl(3)
c Set command word
if(pcomp(uct,'plt0',4)) then
 uct = 'xxxx'
else

c User plot command statements are placed here
endif
end

Figure 4.8: Sample UPL0Tn module

Two plot utilities are available for placing lines on the screen. These are named DPLOT and PLOTL. The calling form for DPLOT is given as

    call dplot(s1,s2,ipen)

where s1, s2 are screen coordinates ranging from 0 to 1. Similarly, the calling sequence for PLOTL is

    call plotl(x1,x2,x3,ipen)

where x1, x2, x3 are coordinates values of the mesh. The value of ipen ranges from 1 to 3: 1 starts a filled panel; 2 draws a line from the current previous point to the new point; 3 moves to the new point without drawing a line. If a filled panel is started it must be closed by inserting the statement

    call clpan()

Lines are drawn or panels filled in the current color. A color is set using the statement
Table 4.1: Color Table for Plots

<table>
<thead>
<tr>
<th>Number</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>White</td>
</tr>
<tr>
<td>2</td>
<td>Red</td>
</tr>
<tr>
<td>3</td>
<td>Green</td>
</tr>
<tr>
<td>4</td>
<td>Blue</td>
</tr>
<tr>
<td>5</td>
<td>Yellow</td>
</tr>
<tr>
<td>6</td>
<td>Cyan</td>
</tr>
<tr>
<td>7</td>
<td>Magenta</td>
</tr>
<tr>
<td>8</td>
<td>Orange</td>
</tr>
<tr>
<td>9</td>
<td>Coral</td>
</tr>
<tr>
<td>10</td>
<td>Green-Yellow</td>
</tr>
<tr>
<td>11</td>
<td>Wheat</td>
</tr>
<tr>
<td>12</td>
<td>Royal Blue</td>
</tr>
<tr>
<td>13</td>
<td>Purple</td>
</tr>
<tr>
<td>14</td>
<td>Aquamarine</td>
</tr>
<tr>
<td>15</td>
<td>Violet-Red</td>
</tr>
<tr>
<td>16</td>
<td>Dark Slate Blue</td>
</tr>
<tr>
<td>17</td>
<td>Gray</td>
</tr>
<tr>
<td>18</td>
<td>Light Gray</td>
</tr>
</tbody>
</table>

call pppcol(color, switch)

where color is an integer defining the color number and switch should be zero. The color values are given in Table 4.1.
Chapter 5

ADDING ELEMENTS

FEAP permits users to add their own element modules to the program by writing a single subprogram called

    subroutine elmntn(d,ul,xl,ix,tl,s,r,ndf,ndm,nst,isw)

where nn may have values between 01 and 50. Each element subprogram must be added before loading the FEAP library since dummy subprograms are included in the library to avoid unsatisfied externals. The basic structure for an element routine is shown in Figures 5.1 and 5.1.

Information is provided to the element subprogram through data passed as arguments and data passed in common blocks. The data passed as arguments consists of eleven

    subroutine elmntn(d,ul,xl,ix,tl,s,r,ndf,ndm,nst,isw)

    c Prototype FEAP Element Routine: nn = 01 to 50
    implicit none
    c Common blocks: See Figure 5.2.
    integer ndf,ndm,nst,isw
    integer ix(*)
    real*8 d(*),ul(ndf,*),xl(ndm,*),tl(*),s(nst,nst),r(nst)
    if(isw.eq.0 .and. ior.lt.0) then
        c Output element description
        write(*,*) ' Elmt 1: Element description'

Figure 5.1: FEAP Element Subprogram. Part 1.
else if (isw.eq.1) then
  c Input/output of property data after command: 'mate'
  c  d(*) stores information for each material set
  c  Return: nh1 = number of nh1/nh2 words/element
  c  Return: nh3 = number of nh3 words/element

else if (isw.eq.2) then
  c  Check element for errors. Negative jacobian, etc.

else if (isw.eq.3) then
  c  Return: Element coefficient matrix and residual
  c  s(nst,nst) element coefficient matrix
  c  r(ndf,nen) element residual
  c  hr(nh1) history data base: previous time step
  c  hr(nh2) history data base: current time step
  c  hr(nh3) history data base: time independent

else if (isw.eq.4) then
  c  Output element quantities (e.g., stresses)

else if (isw.eq.5) then
  c  Return: Element mass matrix
  c  s(nst,nst) consistent matrix
  c  r(ndf,nen) diagonal matrix

else if (isw.eq.6) then
  c  Compute residual only
  c  r(ndf,nen) element residual

else if (isw.eq.7) then
  c  Return: Surface loading for element
  c  s(nst,nst) coefficient matrix
  c  r(ndf,nst) nodal forces

else if (isw.eq.8) then
  c  Compute stress projections to nodes (diagonal)
  c  p(nen) projection weight: wt(nen)
  c  s(nen,*) projection values: st(nen,*)
  c  (default: project 8 quantities)
endif
end

Figure 5.1: FEAP Element Subprogram. Part 2.

(11) items which are briefly described in Table 5.11.

1Note in Table 5.1 that FEAP transfers the values for most of the solution parameters in array UL(NDF,NEN,*) at time \( t_{n+a} \), where \( a \) denotes a value between 0 and 1. The value of \( a \) is 1 (i.e., values are reported for time \( t_{n+1} \)) unless generalized midpoint integration methods are used. For the present we will assume \( a \) is 1.
FEAP carries out tasks according to the value of the parameter ISW passed as the eleventh parameter of the ELMTnn subprogram. A short description of the task carried out by each value, as currently implemented, is shown in Table 5.2.

To use the basic features available in FEAP it is necessary to program tasks labeled as R shown above. If elements have local variables that need to be retained between subsequent time steps *history variables* may be defined as described in Section 5.6. In this case it is necessary to code task 12 if special transformations of the variables are required (otherwise merely return with no changes) and if any of the parameters have non-zero initial values task 14 is used to set these values (zero values are set by default). Finally, if special plotting options are desired it may be necessary to program task 20 (note that contours for element variables such as stress, strain, etc. are developed from task 8).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| d(*)      | Element data parameters  
(Moduli, body loads, etc.) |
| ul(ndf,nen,j) | Element nodal solution parameters  
nen is number of nodes on an element (max)  
j = 1: Displacement $u_{n+a}^{(k)}$  
j = 2: Increment $u_{n+a}^{(k)} - u_n$  
j = 3: Increment $u_{n+1}^{(k)} - u_{n+1}^{(k-1)}$  
j = 4: Rate $v_{n+a}^{(k)}$  
j = 5: Rate $a_{n+a}^{(k)}$  
j = 6: Rate $v_n$ |
| xl(ndm,nen) | Element nodal reference coordinates |
| ix(nen) | Element global node numbers |
| tl(nen) | Element nodal temperature values |
| s(nst,nst) | Element matrix (e.g., stiffness, mass) |
| r(ndf,nen) | Element vector (e.g., residual, mass)  
may also be used as r(nst) |
| ndf | Number unknowns (max) per node |
| ndm | Space dimension of mesh |
| nst | Size of element arrays S and R  
N.B. Normally nst = ndf*nen |
| isw | Task parameter to control computation  
See prototype element in Figure 5.1 |

Table 5.1: Arguments of FEAP Element Subprogram.
It is not necessary to implement all other tasks in an element, however, for those
tasks that are not implemented it is important that the element routine not perform
any calculations. Thus if the form of the branch is programmed as an **IF-THEN-ELSE**
construct as shown in Fig. 5.1 then the **ELSE** should not carry out any operations
**unless all options for ISW are programmed.** Similarly if the element is programmed
using a **SELECT-CASE** form as

<table>
<thead>
<tr>
<th>isw-Task</th>
<th>Type</th>
<th>Description</th>
<th>Access Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>O</td>
<td>Output label</td>
<td>SHOW,ELEM</td>
</tr>
<tr>
<td>1</td>
<td>R</td>
<td>Input d(*) parameters</td>
<td>Mesh:MATE,n</td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>Check elements</td>
<td>Soln:CHECK</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>Compute tangent/residual Store in S/r</td>
<td>Soln:TANG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UTAN</td>
</tr>
<tr>
<td>4</td>
<td>O</td>
<td>Output element variables</td>
<td>Soln:STRE</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>Compute cons/lump mass Store in S/r</td>
<td>Soln:MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MASS,LUMP</td>
</tr>
<tr>
<td>6</td>
<td>R</td>
<td>Compute residual</td>
<td>Soln:FORM,REAC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Plot:REAC</td>
</tr>
<tr>
<td>7</td>
<td>O</td>
<td>Surface load/tangents</td>
<td>Mesh:SLOAd</td>
</tr>
<tr>
<td>8</td>
<td>O</td>
<td>Nodal projections</td>
<td>Soln:STRE,NODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Plot:STRE,PSTR</td>
</tr>
<tr>
<td>9</td>
<td>O</td>
<td>Damping</td>
<td>Soln:DAMP</td>
</tr>
<tr>
<td>10</td>
<td>O</td>
<td>Augmented Lagrangian update</td>
<td>Soln:AUGM</td>
</tr>
<tr>
<td>11</td>
<td>O</td>
<td>Error estimator</td>
<td>Soln:ERRO</td>
</tr>
<tr>
<td>12</td>
<td>R</td>
<td>History update For special history treatments else return</td>
<td>Soln:TIME</td>
</tr>
<tr>
<td>13</td>
<td>O</td>
<td>Energy/momentum</td>
<td>Soln:TPLO,ENER</td>
</tr>
<tr>
<td>14</td>
<td>R</td>
<td>Initialize history</td>
<td>BATCh,INTEr</td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>Body force</td>
<td>Mesh:BODY</td>
</tr>
<tr>
<td>16</td>
<td>O</td>
<td>J integrals</td>
<td>Soln:JINT</td>
</tr>
<tr>
<td>17</td>
<td>O</td>
<td>Set after activation</td>
<td>Soln:ACTI</td>
</tr>
<tr>
<td>18</td>
<td>O</td>
<td>Set after deactivation</td>
<td>Soln:DEAC</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>NOT AVAILABLE: used in modal/base Uses isw = 5 in element</td>
<td>BASE</td>
</tr>
<tr>
<td>20</td>
<td>O</td>
<td>Element plotting</td>
<td>Plot:PELM</td>
</tr>
<tr>
<td>23</td>
<td>O</td>
<td>Compute element loads only</td>
<td>ARCL</td>
</tr>
<tr>
<td>25</td>
<td>O</td>
<td>Zienkiewicz-Zhu projection</td>
<td>Soln:ZZHU</td>
</tr>
<tr>
<td>26</td>
<td>R</td>
<td>Used to compute mesh boundary</td>
<td>Called by default.</td>
</tr>
</tbody>
</table>

Table 5.2: Task Options for FEAP Element Subprogram. R = Required; O = Optional; E = For eigensolutions
select case (isw)
  case(1)
    c Input material parameters
    ...
  case default
    ...
end select

the CASE DEFAULT should not perform any operations unless all options are programmed. Finally, if the form

    go to (1,2,..... ), isw
    return

is used the RETURN statement should *always be included* as shown. This prevents any unexpected execution of a statement that appears after the GO TO.

Some of the options for additional data passed through common blocks is shown in Figure 5.2 with each variable defined in Table 5.3. Also, in Figure 5.3 the reference to common blocks using include statements is shown. In the prototype routine the number of nodes on an element (nen) which is used to dimension ul is passed in the labeled common /cdata/. Additional discussion is given below on use of some of the other data passed through the common blocks.

### 5.1 Material property storage

The material parameters to be stored in the array D with pointer np(25) may be input using the subprogram **INMATE**. This subroutine is accessed by the statement:

```
CALL INMATE(D,TDOF, NEV, TYPE)
```

where D is the array storing the material parameters; TDOF is returned as the parameter to access temperature; NEV is the number of element history variables to allocate to NH1; and TYPE is the element type. This routine inputs the commands as described in the user manual and stores the data for each material set into the D array elements as described in the following tables.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
</table>
| o head   | Page eject option  
|          | Title record |
| numnp    | Number of mesh nodes |
| numel    | Number of mesh elements |
| nummat   | Number of material sets |
| nen      | Maximum nodes/element |
| neq      | Number active equations |
| ipr      | Real variable precision |
| nstep    | Total number of time steps |
| niter    | Number of iterations current step |
| naugm    | Number of augments current step |
| titer    | Total iterations |
| taubm    | Total augments |
| iaugm    | Augmenting counter |
| iform    | Number residuals in line search |
| dm       | Element proportional load |
| n        | Current element number |
| ma       | Current element material set |
| mct      | Print counter |
| iel      | User element number |
| nel      | Number nodes on current element |
| tt       | Element stress values for TPLOt |
| bpr      | Principal stretch |
| ctan     | Element multipliers |
| ut       | Element user values for TPLOt |

Table 5.3: FEAP common block definitions.

### 5.2 Non-linear Transient Solution Forms

Before describing the steps in developing an element we summarize first the basic structure of the algorithms employed by FEAP to solve problems. Each problem to be solved using an ELMTnn routine is established in a standard finite element form as described in standard references (e.g., *The Finite Element Method*, 4th ed., by O.C. Zienkiewicz and R.L. Taylor, McGraw-Hill, London, 1989 (vol 1), 1991 (vol 2)). Here it is assumed this step leads to a set of non-linear ordinary differential equations expressed in terms of nodal displacements, velocities, and accelerations given by $u_i(t)$, $\dot{u}_i(t)$, and $\ddot{u}_i(t)$, respectively. We denote the differential equation for node-$i$ as the
### Variable Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>nh1</td>
<td>Pointer to $t_n$ history data</td>
</tr>
<tr>
<td>nh2</td>
<td>Pointer to $t_{n+1}$ history data</td>
</tr>
<tr>
<td>nh3</td>
<td>Pointer to element history</td>
</tr>
<tr>
<td>ior</td>
<td>Current input logical unit</td>
</tr>
<tr>
<td>iow</td>
<td>Current output logical unit</td>
</tr>
<tr>
<td>nph</td>
<td>Pointer to global projection arrays</td>
</tr>
<tr>
<td>ner</td>
<td>Pointer to global error indicator</td>
</tr>
<tr>
<td>erav</td>
<td>Element error value</td>
</tr>
<tr>
<td>j-int</td>
<td>J integral values</td>
</tr>
<tr>
<td>ndf</td>
<td>Maximum dof/node</td>
</tr>
<tr>
<td>ndm</td>
<td>Mesh space dimension</td>
</tr>
<tr>
<td>nen1</td>
<td>Dimension 1 on IX array</td>
</tr>
<tr>
<td>nst</td>
<td>Size of element matrix</td>
</tr>
<tr>
<td>nneq</td>
<td>Total dof in problem</td>
</tr>
<tr>
<td>ttim</td>
<td>Current time</td>
</tr>
<tr>
<td>dt</td>
<td>Current time increment</td>
</tr>
<tr>
<td>ci</td>
<td>Integration parameters</td>
</tr>
<tr>
<td>hr</td>
<td>Real array data</td>
</tr>
<tr>
<td>mr</td>
<td>Integer array data</td>
</tr>
</tbody>
</table>

Table 5.4: FEAP common block definitions.

The residual equation:

\[
R_i(u_i(t), \dot{u}_i(t), \ddot{u}_i(t), t) = 0.
\]

To solve for the nodal displacements, velocities and accelerations it is necessary to introduce an algorithm to integrate the nodal quantities in time, specify a constitutive relation, and develop an algorithm to solve a (possibly) non-linear problem.

In FEAP, the integration method for nodal quantities is taken as a one step algorithm with each quantity defined only at discrete times $t_n$. Accordingly, we have displacements $u_i(t_n)$ with velocities and accelerations denoted as

\[
\dot{u}_i(t_n) \approx v_i(t_n)
\]

and

\[
\ddot{u}_i(t_n) \approx a_i(t_n)
\]

A typical example for an integration algorithm for these discrete quantities is Newmark’s method where

\[
u_i(t_{n+1}) = u_i(t_n) + \Delta t v_i(t_n) + \Delta t^2 [\frac{1}{2} - \beta] a_i(t_n) + \beta a_i(t_{n+1})]
\]
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$E$</td>
<td>Young’s modulus</td>
</tr>
<tr>
<td>2</td>
<td>$\nu$</td>
<td>Poisson ratio</td>
</tr>
<tr>
<td>3</td>
<td>$\alpha$</td>
<td>Thermal expansion coefficient</td>
</tr>
<tr>
<td>4</td>
<td>$\rho$</td>
<td>Mass density</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>Quadrature order for arrays</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>Quadrature order for outputs</td>
</tr>
<tr>
<td>7</td>
<td>$a$</td>
<td>Mass interpolation ($a = 0$: Diagonal; $a = 1$: Consistent)</td>
</tr>
<tr>
<td>8</td>
<td>$q$</td>
<td>Loading intensity (plates/shells)</td>
</tr>
<tr>
<td>9</td>
<td>$T_0$</td>
<td>Stress free reference temperature</td>
</tr>
<tr>
<td>10</td>
<td>$\kappa$</td>
<td>Shear factor (plates/shells/beams)</td>
</tr>
<tr>
<td>11</td>
<td>$b_1$</td>
<td>Body force/volume in 1-directions</td>
</tr>
<tr>
<td>12</td>
<td>$b_2$</td>
<td>Body force/volume in 2-directions</td>
</tr>
<tr>
<td>13</td>
<td>$b_3$</td>
<td>Body force/volume in 3-directions</td>
</tr>
<tr>
<td>14</td>
<td>$h$</td>
<td>Thickness (plates/shells)</td>
</tr>
<tr>
<td>15</td>
<td>nh1</td>
<td>History variable counter</td>
</tr>
<tr>
<td>16</td>
<td>stype</td>
<td>Two dimensional type: 1 - plane stress; 2 - plane strain; 3 - axisymmetric$^2$</td>
</tr>
<tr>
<td>17</td>
<td>etype</td>
<td>Element formulation: 1 - displ; 2 - mixed; 3 - enhanced</td>
</tr>
<tr>
<td>18</td>
<td>dtype</td>
<td>Deformation type: $&lt;$: finite; $&gt;$/small</td>
</tr>
<tr>
<td>19</td>
<td>tdof</td>
<td>Thermal degree-of-freedom</td>
</tr>
<tr>
<td>20</td>
<td>imat</td>
<td>Non-linear elastic material type</td>
</tr>
<tr>
<td>21</td>
<td>$d_{11}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>22</td>
<td>$d_{22}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>23</td>
<td>$d_{33}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>24</td>
<td>$d_{12}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>25</td>
<td>$d_{23}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>26</td>
<td>$d_{31}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>27</td>
<td>$g_{12}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>28</td>
<td>$g_{23}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>29</td>
<td>$g_{31}$</td>
<td>Material moduli</td>
</tr>
<tr>
<td>30</td>
<td>htype</td>
<td>Heat flag</td>
</tr>
</tbody>
</table>

Table 5.5: Material Parameters
## Chapter 5. Adding Elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>$\psi$</td>
<td>Orthotropic angle $x_1$ principal axis 1</td>
</tr>
<tr>
<td>32</td>
<td>$A$</td>
<td>Area cross section (beam/truss)</td>
</tr>
<tr>
<td>33</td>
<td>$I_{11}$</td>
<td>Inertia cross section (beam/truss)</td>
</tr>
<tr>
<td>34</td>
<td>$I_{22}$</td>
<td>Inertia cross section (beam/truss)</td>
</tr>
<tr>
<td>35</td>
<td>$I_{12}$</td>
<td>Inertia cross section (beam/truss)</td>
</tr>
<tr>
<td>36</td>
<td>$J$</td>
<td>Polar inertia cross section (beam/truss)</td>
</tr>
<tr>
<td>37</td>
<td>$\kappa_1$</td>
<td>Shear factor plate</td>
</tr>
<tr>
<td>38</td>
<td>$\kappa_2$</td>
<td>Shear factor plate</td>
</tr>
<tr>
<td>39</td>
<td>-</td>
<td>Non-linear flag (beam/truss)</td>
</tr>
<tr>
<td>40</td>
<td>-</td>
<td>Inelastic material model type</td>
</tr>
<tr>
<td>41</td>
<td>$Y_0$</td>
<td>Initial yield stress (Mises)</td>
</tr>
<tr>
<td>42</td>
<td>$Y_\infty$</td>
<td>Final yield stress (Mises)</td>
</tr>
<tr>
<td>43</td>
<td>$\beta$</td>
<td>Exponential hardening rate</td>
</tr>
<tr>
<td>44</td>
<td>$H_{iso}$</td>
<td>Isotropic hardening modulus (linear)</td>
</tr>
<tr>
<td>45</td>
<td>$H_{kin}$</td>
<td>Kinematic hardening modulus (linear)</td>
</tr>
<tr>
<td>46</td>
<td>-</td>
<td>Yield flag</td>
</tr>
<tr>
<td>47</td>
<td>$\beta_1$</td>
<td>Orthotropic thermal stress</td>
</tr>
<tr>
<td>48</td>
<td>$\beta_2$</td>
<td>Orthotropic thermal stress</td>
</tr>
<tr>
<td>49</td>
<td>$\beta_3$</td>
<td>Orthotropic thermal stress</td>
</tr>
<tr>
<td>50</td>
<td>-</td>
<td>Error estimator parameter</td>
</tr>
<tr>
<td>51</td>
<td>$\nu_1$</td>
<td>Viscoelastic shear parameter</td>
</tr>
<tr>
<td>52</td>
<td>$\tau_1$</td>
<td>Viscoelastic relaxation time</td>
</tr>
<tr>
<td>53</td>
<td>$\nu_2$</td>
<td>Viscoelastic shear parameter</td>
</tr>
<tr>
<td>54</td>
<td>$\tau_2$</td>
<td>Viscoelastic relaxation time</td>
</tr>
<tr>
<td>55</td>
<td>$\nu_3$</td>
<td>Viscoelastic shear parameter</td>
</tr>
<tr>
<td>56</td>
<td>$\tau_3$</td>
<td>Viscoelastic relaxation time</td>
</tr>
<tr>
<td>57</td>
<td>nvis</td>
<td>Number of viscoelastic terms (1-3)</td>
</tr>
<tr>
<td>58</td>
<td>-</td>
<td>Damage limit</td>
</tr>
<tr>
<td>59</td>
<td>-</td>
<td>Damage rate</td>
</tr>
<tr>
<td>60</td>
<td>$k$</td>
<td>Penalty parameter</td>
</tr>
</tbody>
</table>

Table 5.6: Material Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>$K_1$</td>
<td>Fourier thermal conductivity</td>
</tr>
<tr>
<td>62</td>
<td>$K_2$</td>
<td>Fourier thermal conductivity</td>
</tr>
<tr>
<td>63</td>
<td>$K_3$</td>
<td>Fourier thermal conductivity</td>
</tr>
<tr>
<td>64</td>
<td>$c$</td>
<td>Fourier specific heat</td>
</tr>
<tr>
<td>65</td>
<td>$\omega$</td>
<td>Angular velocity</td>
</tr>
<tr>
<td>66</td>
<td>$Q$</td>
<td>Body heat</td>
</tr>
<tr>
<td>67</td>
<td>-</td>
<td>Heat constitution added indicator</td>
</tr>
<tr>
<td>68</td>
<td>-</td>
<td>Follower loading indicator</td>
</tr>
<tr>
<td>69</td>
<td>-</td>
<td>Rotational mass factor</td>
</tr>
<tr>
<td>70</td>
<td>-</td>
<td>Damping factor</td>
</tr>
<tr>
<td>71</td>
<td>$g_1$</td>
<td>Ground acceleration factor</td>
</tr>
<tr>
<td>72</td>
<td>$g_2$</td>
<td>Ground acceleration factor</td>
</tr>
<tr>
<td>73</td>
<td>$g_3$</td>
<td>Ground acceleration factor</td>
</tr>
<tr>
<td>74</td>
<td>$p_1$</td>
<td>Ground acceleration proportional load number</td>
</tr>
<tr>
<td>75</td>
<td>$p_2$</td>
<td>Ground acceleration proportional load number</td>
</tr>
<tr>
<td>76</td>
<td>$p_3$</td>
<td>Ground acceleration proportional load number</td>
</tr>
<tr>
<td>77</td>
<td>$a_0$</td>
<td>Rayleigh damping mass ratio</td>
</tr>
<tr>
<td>78</td>
<td>$a_1$</td>
<td>Rayleigh damping stiffness ratio</td>
</tr>
<tr>
<td>79</td>
<td>-</td>
<td>Plate/Shell/Rod shear activation flag</td>
</tr>
<tr>
<td>80</td>
<td>Method: Type 1</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>Method: Type 2</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>-</td>
<td>Truss/Rod quadrature number</td>
</tr>
<tr>
<td>83</td>
<td>-</td>
<td>Axial loading value</td>
</tr>
<tr>
<td>84</td>
<td>-</td>
<td>Constitutive start indicator</td>
</tr>
<tr>
<td>85</td>
<td>-</td>
<td>Polar angle indicator</td>
</tr>
<tr>
<td>86</td>
<td>-</td>
<td>Polar angle coord_1</td>
</tr>
<tr>
<td>87</td>
<td>-</td>
<td>Polar angle coord_2</td>
</tr>
<tr>
<td>88</td>
<td>-</td>
<td>Polar angle coord_3</td>
</tr>
<tr>
<td>89</td>
<td>-</td>
<td>Constitution transient type</td>
</tr>
<tr>
<td>90</td>
<td>$d_{31}$</td>
<td>Plane stress recovery</td>
</tr>
<tr>
<td>91</td>
<td>$d_{32}$</td>
<td>Plane stress recovery</td>
</tr>
<tr>
<td>92</td>
<td>$\alpha_3$</td>
<td>Plane stress recovery</td>
</tr>
</tbody>
</table>

Table 5.7: Material Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>sref</td>
<td>Shear center type</td>
</tr>
<tr>
<td>94</td>
<td>$y_1$</td>
<td>Shear center coordinate</td>
</tr>
<tr>
<td>95</td>
<td>$y_2$</td>
<td>Shear center coordinate</td>
</tr>
<tr>
<td>96</td>
<td>lref</td>
<td>Reference vector type</td>
</tr>
<tr>
<td>97</td>
<td>$n_1$</td>
<td>Reference vector parameter</td>
</tr>
<tr>
<td>98</td>
<td>$n_2$</td>
<td>Reference vector parameter</td>
</tr>
<tr>
<td>99</td>
<td>$n_3$</td>
<td>Reference vector parameter</td>
</tr>
<tr>
<td>100</td>
<td>-</td>
<td>Cross section shape type: 1 = rectangles; 2 = tube; 3 = Wide flange; 4 = Channel; 5 = Angle; 5 = Circle</td>
</tr>
<tr>
<td>101-126</td>
<td>-</td>
<td>Shape data</td>
</tr>
<tr>
<td>127</td>
<td>-</td>
<td>Surface convection ($h$)</td>
</tr>
<tr>
<td>128</td>
<td>-</td>
<td>Free-stream temperature ($T_\infty$)</td>
</tr>
<tr>
<td>129</td>
<td>-</td>
<td>Reference absolute temperature</td>
</tr>
<tr>
<td>130</td>
<td>nseg</td>
<td>Number of hardening segments</td>
</tr>
<tr>
<td>131-148</td>
<td>-</td>
<td>Segment data sets $e_pY_{iso}H_{kin}$</td>
</tr>
<tr>
<td>149</td>
<td>-</td>
<td>Total variables on frame section</td>
</tr>
<tr>
<td>150</td>
<td>-</td>
<td>Piezoelectric flag</td>
</tr>
<tr>
<td>151-159</td>
<td>-</td>
<td>Piezoelectric data</td>
</tr>
<tr>
<td>160</td>
<td>-</td>
<td>Initial stress flag</td>
</tr>
<tr>
<td>161-166</td>
<td>$\sigma_{ij}$</td>
<td>Initial stresses (constant)</td>
</tr>
<tr>
<td>167</td>
<td>-</td>
<td>Tension/compression only indicator</td>
</tr>
<tr>
<td>170</td>
<td>C</td>
<td>Fung pseudo elastic model modulus</td>
</tr>
<tr>
<td>171</td>
<td>$a_1$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>172</td>
<td>$a_2$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>173</td>
<td>$a_3$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>174</td>
<td>$a_4$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>175</td>
<td>$a_5$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>176</td>
<td>$a_6$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>177</td>
<td>$a_7$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>178</td>
<td>$a_8$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>179</td>
<td>$a_9$</td>
<td>Fung model energy parameter</td>
</tr>
<tr>
<td>180-181</td>
<td>-</td>
<td>Viscoplastic rate parameters</td>
</tr>
<tr>
<td>182</td>
<td>-</td>
<td>Nodal quadrature parameters</td>
</tr>
<tr>
<td>183</td>
<td>$\beta_m$</td>
<td>$M_L - M_C$ mass scaling factor</td>
</tr>
<tr>
<td>184</td>
<td>c</td>
<td>Estimate on maximum wave speed</td>
</tr>
<tr>
<td>185</td>
<td>-</td>
<td>Augmentation switch: &lt;on/off&gt;</td>
</tr>
<tr>
<td>186</td>
<td>-</td>
<td>Augmentation explicit indicator</td>
</tr>
<tr>
<td>187</td>
<td>-</td>
<td>Implicit = 0; Explicit = 1 element integration</td>
</tr>
</tbody>
</table>

Table 5.8: Material Parameters
and

\[ v_i(t_{n+1}) = v_i(t_n) + \Delta t \left[ (1 - \gamma) a_i(t_n) + \gamma a_i(t_{n+1}) \right] \]

with \( u, v, \) and \( a \) being the set of displacements, velocities, and accelerations at node-\( i \), respectively.

A Newton method is commonly adopted to solve a non-linear (or linear) problem. To implement a Newton method it is necessary to linearize the residual equation. For FEAP, the Newton equation may be written as

\[ R_i^{(k+1)} = R_i^{(k)} + \frac{\partial R_i}{\partial \alpha_j} \bigg|^{(k)} d\alpha_j^{(k)} = 0 \]

where \( \alpha_j \) is one of the variables at time \( t_{n+1} \) (e.g., \( u_j(t_{n+1}) \)). We define

\[ S_{ij}^{(k)} = -\frac{\partial R_i}{\partial \alpha_j} \bigg|^{(k)} \]

and solve

\[ S_{ij}^{(k)} d\alpha_j^{(k)} = R_i^{(k)} \]

The solution is updated using

\[ \alpha_j^{(k+1)} = \alpha_j^{(k)} + d\alpha_j^{(k)} \]
In the above \((k)\) is the iteration number for the Newton algorithm. To start the solution for each step, FEAP sets

\[
\alpha_j^{(0)}(t_{n+1}) = \alpha_j(t_n)
\]

where a quantity without the \((k)\) superscript represents a converged value. For a linear problem, Newton’s method converges in one iteration. Computing the residual after one iteration must yield a zero value to within the roundoff of the computer used. For non-linear problems, a properly implemented Newton’s method must exhibit a quadratic asymptotic rate of convergence. Failure of the above performance for linear and non-linear cases implies a programming error in an implementation or lack of a consistently linearized algorithm (i.e., \(S_{ij}\) is not an exact derivative of the residual).

In a non-linear problem, Newmark’s method may be parameterized in terms of increments of displacement, velocity, or acceleration. From the Newmark formulas, the relations

\[
du_i = \beta \Delta t^2 da_i
\]

and

\[
dv_i = \gamma \Delta t da_i
\]

define the relationships between the increments. Note that only scalar multipliers involving \(\beta, \gamma, \) and \(\Delta t\) are involved between the different measures.

The tangent matrix for the transient problem using Newmark’s method may be expressed in terms of the incremental displacement, velocity, or acceleration. As an example, consider the case where the solution is parameterized in terms of increments of the displacements (i.e., \(\alpha_j\) is the displacement vector \(u_j\)). For this case, the tangent matrix is (we do not show dependence on the iteration \((k)\) for simplicity of notation)

\[
S_{ij} du_j = -\frac{\partial R_i}{\partial u_j} du_j - \frac{\partial R_i}{\partial v_j} \frac{\partial v_k}{\partial u_j} du_j - \frac{\partial R_i}{\partial a_k} \frac{\partial a_k}{\partial u_j} du_j.
\]

Note that from the Newmark formulas

\[
\frac{\partial a_k}{\partial u_j} = \frac{1}{\beta \Delta t^2} \delta_{kj}; \quad \frac{\partial v_k}{\partial u_j} = \frac{\partial v_k}{\partial a_l} \frac{\partial a_l}{\partial u_j} = \frac{\gamma}{\beta \Delta t} \delta_{kj}
\]

in which \(\delta_{kj}\) is the Kronnecker delta identity matrix for the \(k,j\) nodal pair. From the residual we observe that

\[
K_{ij} = -\frac{\partial R_i}{\partial u_j}; \quad C_{ij} = -\frac{\partial R_i}{\partial v_j}; \quad M_{ij} = -\frac{\partial R_i}{\partial a_j}
\]

define the tangent stiffness, damping, and mass, respectively. Thus, for the Newmark algorithm the total tangent matrix in terms of the incremental displacements is

\[
S_{ij} = K_{ij} + \frac{\gamma}{\beta \Delta t} C_{ij} + \frac{1}{\beta \Delta t^2} M_{ij}.
\]
CHAPTER 5. ADDING ELEMENTS

For other choices of increments, the tangent may be written in the general form

\[ S_{ij} = c_1 K_{ij} + c_2 C_{ij} + c_3 M_{ij} \]

where the \( c_i \) are scalar quantities involving the integration parameters of the method selected and \( \Delta t \). Thus, any one step integrator may be considered and will affect only the specification of the constants in the tangent.

In FEAP the element tangent matrix, \( S_{ij} \), is stored as a two dimensional array which is dimensioned as \( s(nst,nst) \), where \( nst \) is the product of \( ndf \) and \( nen \), with \( ndf \) the maximum number of degree-of-freedoms at any node in the problem and \( nen \) the maximum number of nodes on any element. The ordering of the unknowns into \( nst \) must be carefully aligned in order for FEAP to properly assemble each element matrix into the global tangent. The ordering is such that sub-matrices are defined for each node attached to the element. Thus

\[
S = \begin{bmatrix}
S_{11} & S_{12} & S_{13} & \cdots \\
S_{21} & S_{22} & S_{23} & \cdots \\
S_{31} & S_{32} & S_{33} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\]

where \( S_{ij} \) is the sub-matrix for nodal pairs \( i,j \). Each of the sub-matrices is a square matrix of the size of the maximum number of degree-of-freedoms in the problem which is passed to the subprogram as \( ndf \). Thus,

\[
S_{ij} = \begin{bmatrix}
S^{ij}_{11} & S^{ij}_{12} & S^{ij}_{13} & \cdots \\
S^{ij}_{21} & S^{ij}_{22} & S^{ij}_{23} & \cdots \\
S^{ij}_{31} & S^{ij}_{32} & S^{ij}_{33} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\]

in which \( S^{ij}_{ab} \) is an array coefficient for nodal pair \( i,j \) for the degree-of-freedom pair \( a,b \).

In FEAP, the element residual may be stored as one dimensional array which is dimensioned \( r(nst) \) with entries stored in the same order as the rows of the element tangent matrix or as a two dimensional array which is dimensioned as \( r(ndf,nen) \). The one dimensional form of the residual is given as

\[
R = \begin{bmatrix}
R_1 \\
R_2 \\
R_3 \\
\vdots
\end{bmatrix}
\]
where the entries in each submatrix are given as

\[
R_i = \begin{bmatrix}
R_{i1} \\
R_{i2} \\
R_{i3} \\
\vdots \\
R_{in_{df}}
\end{bmatrix}.
\]

The two dimensional form places the entries \( R_i \) as columns. Accordingly,

\[
\mathbf{R} = \begin{bmatrix} R_1 & R_2 & R_3 & \cdots \end{bmatrix}.
\]

The two forms for defining the residual \( \mathbf{r} \) are equivalent based on the Fortran ordering of information into double subscript arrays.

If \( n_{df} \) is larger than needed for the element and residual the unused positions need not be defined (the tangent array \( \mathbf{s} \) and the residual \( \mathbf{r} \) are set to zero before each element routine is called).

The arrays \( xl(i,j) \), \( ul(i,j,1) \), \( ul(i,j,4) \), and \( ul(i,j,5) \) (described in Table 5.1) are used to obtain the nodal coordinates, displacements, velocities, and accelerations, respectively. When \( FEAP \) solves a problem without transient loading (e.g., inertial loading as mass times acceleration) the velocities and accelerations are set to zero prior to calling the element subroutine. Consequently, in programming the steps to compute the residual \( \mathbf{r} \) the inertia terms have no effect for static or quasi-static problems and may be included (generally there are very few additional operations involved to add these terms). The programming of the tangent array, however, must distinguish between cases in which transient (e.g., inertial) loads are present and those in which they are omitted. The different cases are implemented in \( FEAP \) by making appropriate assignments to the \( c_i \) parameters. To facilitate the programming of the tangent array returned in \( \mathbf{s} \) for the various cases, a parameter array \( \text{ctan}(3) \) is passed to the subprogram in labeled common \( \text{eltran} \). When the task parameter \( \text{isw} \) is 3, the values in the \( \text{ctan} \) array are interpreted according to Table 5.10.

Thus, in solid mechanics applications the tangent matrix is defined in an element routine as

\[
\mathbf{S} = \text{ctan}(1) \mathbf{K} + \text{ctan}(2) \mathbf{C} + \text{ctan}(3) \mathbf{M}
\]

where \( \mathbf{K} \) is the stiffness matrix, \( \mathbf{C} \) is the damping matrix, and \( \mathbf{M} \) is the mass matrix. For non-linear applications these matrices normally are computed with respect to the current values of the available solution parameters. The values provided in the \( \text{ctan} \) array are set by \( FEAP \) according to the active transient solution option. For a static
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctan(1)</td>
<td>$c_1$: Multiplier of $s$ matrix for $u_l(i,j,1)$ terms (e.g., stiffness matrix multiplier)</td>
</tr>
<tr>
<td>ctan(2)</td>
<td>$c_2$: Multiplier of $s$ matrix for $u_l(i,j,4)$ terms (e.g., damping matrix multiplier)</td>
</tr>
<tr>
<td>ctan(3)</td>
<td>$c_3$: Multiplier of $s$ matrix for $u_l(i,j,5)$ terms (e.g., mass matrix multiplier)</td>
</tr>
</tbody>
</table>

Table 5.10: Tangent Parameters

option both $ctan(2)$ and $ctan(3)$ are zero. For options integrating first order differential equations in time only $ctan(3)$ will be zero. For options integrating second order differential equations in time all the parameters are non-zero.

### 5.3 Example: 2-Node Truss Element

An element routine carries out tasks according to the value assigned to the parameter $isw$ as indicated in Table 5.2 To describe basic steps to program the various tasks defined by $isw$, we consider next the problem of a 2-node, linear elastic truss element for small deformation applications. The element is described in sufficient generality to permit solution of both two and three dimensional truss problems.

#### 5.3.1 Theory for a Truss

The governing equations for a typical truss member element, shown in Figure 5.4, are the balance of momentum equation:

$$\frac{\partial (A\sigma_{ss})}{\partial s} + A b_s = \rho A \ddot{u}_s$$

the strain-displacement equation for small deformations:

$$\epsilon_{ss} = \frac{\partial u_s}{\partial s}$$

and a constitutive equation. For example, considering a linear elastic material the constitutive equation may be written as
\[ \sigma_{ss} = E \epsilon_{ss} . \]

Boundary and initial conditions must also be specified to obtain a well posed problem; however, our emphasis here is the derivation of the element arrays associated with the above differential equations. In the above:

- \( s \) is the coordinate along the truss member axis,
- \( b_s \) is a loading in direction \( s \) per unit length,
- \( A \) is the truss cross-section area,
- \( \rho \) is the mass density per unit volume,
- \( u_s \) is a displacement in direction \( s \),
- \( \dot{v}_s \) is an acceleration in direction \( s \) \((v = \dot{u})\),
- \( \epsilon_{ss} \) is a strain along the truss member axis, and
- \( \sigma_{ss} \) is the stress on a truss cross section.

The equations may also be deduced from the variational equation

\[ \delta \Pi = \int_L \delta \epsilon_{ss} \sigma_{ss} A \, ds + \sum_{i=1}^{d} \int_L \delta u_i \rho A \dot{v}_i \, ds - \sum_{i=1}^{d} \int_L \delta u_i b_i \, ds + \delta \Pi_{ext} \]

where \( \delta \Pi_{ext} \) contains the boundary and loading terms not associated with an element. Where, in addition to previously defined quantities, we define:

- \( d \) is the spatial dimension of the truss \((1, 2, \text{or } 3)\),
- \( x_i \) are the Cartesian coordinates in the \( d \) directions,
- \( L \) is the length of the truss member,
- \( \delta u_i \) is a virtual displacement in direction \( x_i \),
- \( \dot{v}_i \) is an acceleration in direction \( x_i \) \((v = \dot{u})\),
- \( b_i \) is a loading in direction \( x_i \) per unit length, and
- \( \delta \epsilon_{ss} \) is a virtual strain along the truss axis.
For a straight truss member the displacement along the axis, \( u_s \) may be expressed in terms of the components in the directions \( x_i \) as

\[
    u_s = \mathbf{1} \cdot \mathbf{u}(s, t) = \sum_{i=1}^{d} l_i u_i(s, t)
\]

where \( t \) is time, \( \mathbf{u} \) is the displacement vector with components \( u_i \), \( \mathbf{1} \) is a unit vector along the axis of the member with direction cosines \( l_i \) defined by

\[
    l_i = \frac{\partial x_i}{\partial s} = \frac{x_{i2} - x_{i1}}{L}
\]

\[
    L^2 = \sum_{i=1}^{d} (x_{i2} - x_{i1})^2
\]

and \( x_{i1}, x_{i2} \) are the coordinates of nodes 1 and 2, respectively. The displacement components are interpolated on the 2-node truss member as

\[
    u_i(s, t) = (1 - \xi) u_{i1}(t) + \xi u_{i2}(t) ; \quad \xi = \frac{s}{L}
\]

in which \( u_{i1}, u_{i2} \) are the displacements at nodes 1 and 2. The virtual displacements are obtained from the above by replacing \( u_i \) by \( \delta u_i \), etc. The truss strain is

\[
    \epsilon_{ss} = \frac{\partial u_s}{\partial s} = \sum_{i=1}^{d} l_i \frac{\partial u_i}{\partial s}.
\]

Using the interpolations for the displacement components yields

\[
    \epsilon_{ss} = \frac{1}{L^2} \sum_{i=1}^{d} \Delta x_i \Delta u_i
\]

where

\[
    \Delta x_i = x_{i2} - x_{i1} = l_i L
\]

and

\[
    \Delta u_i = u_{i2} - u_{i1}.
\]

Thus, in matrix form the strain is

\[
    \epsilon_{ss} = \frac{1}{L^2} \sum_{i=1}^{d} \begin{bmatrix} -\Delta x_i & \Delta x_i \end{bmatrix} \begin{bmatrix} u_{i1} \\ u_{i2} \end{bmatrix}
\]
Using the above displacement interpolations, the variational equation for the truss may be expressed in matrix form as

\[ \delta \Pi = [\delta u_{i1} \quad \delta u_{i2}] \left\{ \int_1 \frac{1}{L^2} \left[ \frac{-\Delta x_i}{\Delta x_i} \right] \sigma_{ss} A ds + \int_L \left[ 1 - \xi \right] \rho A \left[ 1 - \xi \right] ds \begin{bmatrix} \ddot{u}_{i1} \\ \ddot{u}_{i2} \end{bmatrix} \right. \]

\[ \left. - \int_L \left[ 1 - \xi \right] b_i ds \right\} + \delta \Pi_{ext}. \]

FEAP constructs the finite element arrays from the element residuals which are obtained from the negative of the terms multiplying the nodal displacements. Accordingly,

\[ \mathbf{R}_i = \begin{bmatrix} R_{i1} \\ R_{i2} \end{bmatrix} = \int_L \left[ 1 - \xi \right] b_i ds \]

\[ - \int_L \frac{1}{L^2} \left[ \frac{-\Delta x_i}{\Delta x_i} \right] \sigma_{ss} A ds - \int_L \left[ 1 - \xi \right] \rho A \left[ 1 - \xi \right] ds \begin{bmatrix} \ddot{u}_{i1} \\ \ddot{u}_{i2} \end{bmatrix} \]

is the residual for the i-coordinate direction. For constant properties and loading over an element length (note that for this case the stress will also be constant since strains are constant on the element), the above may be integrated to yield

\[ \mathbf{R}_i = \begin{bmatrix} R_{i1} \\ R_{i2} \end{bmatrix} = \frac{1}{2} b_i L \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{\sigma_{ss} A}{L} \left[ -\Delta x_i \Delta x_i \right] - \frac{\rho A L}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} \ddot{u}_{i1} \\ \ddot{u}_{i2} \end{bmatrix}, \quad (5.1) \]

For the present we assume the material model is a linear elastic in which the stress is related to strain through

\[ \sigma_{ss} = E \epsilon_{ss} \]

where \( E \) is the Young’s modulus.

Based on a linear elastic material, the term in the residual involving \( \sigma_{ss} \) may be written as

\[ \frac{\sigma_{ss} A}{L} \left[ -\Delta x_i \Delta x_i \right] = \frac{E A}{L^3} \left[ -\Delta x_i \Delta x_i \right] \sum_{j=1}^{d} \left[ -\Delta x_j \Delta x_j \right] \begin{bmatrix} u_{j1} \\ u_{j2} \end{bmatrix}. \]

For the linear elastic material, a stiffness matrix may be expressed as

\[ \mathbf{K}_{ij} = \frac{E A}{L^3} \left[ -\Delta x_i \Delta x_i \right] \left[ -\Delta x_j \Delta x_j \right] = \begin{bmatrix} k_{ij} & -k_{ij} \\ -k_{ij} & k_{ij} \end{bmatrix} \]

where

\[ k_{ij} = \frac{E A}{L^3} \Delta x_i \Delta x_j. \]
The residual may now be written using a stiffness and mass matrix as
\[
\mathbf{R}_i = \begin{bmatrix} R_{i1} \\ R_{i2} \end{bmatrix} = \frac{1}{2} b_i L \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \sum_{j=1}^{d} \begin{bmatrix} k_{ij} & -k_{ij} \\ -k_{ij} & k_{ij} \end{bmatrix} \begin{bmatrix} u_{j1} \\ u_{j2} \end{bmatrix} - \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} \ddot{u}_{i1} \\ \ddot{u}_{i2} \end{bmatrix}
\]
(5.2)
with
\[
m_{11} = m_{22} = \frac{\rho A L}{3}; \quad m_{12} = m_{21} = \frac{\rho A L}{6}.
\]
For non-linear material behavior the residual must be computed using Equation 5.1 with the stress replaced by the value computed from the constitutive equation.

The integration method for nodal quantities is taken as Newmark’s method described in Section 5.2. The residual and tangent matrix for a Newton type method are now available and may be inserted into \( \mathbf{R} \) and \( \mathbf{S} \) after noting that for the truss that the damping matrix \( \mathbf{C} \) is zero. The residual may be programmed directly from Equation 5.1 and an implementation using the two dimensional form \( r(\text{ndf, nen}) \) is shown in Figure 5.5.

Similarly, using the results from Section 5.2, the tangent matrix for the truss may be programmed as indicated in Figures 5.6 and 5.7.

5.4 Additional Options in Elements

FEAP permits some additional options to be included within element tasks.

5.4.1 Task 1 Options

Setting active equations

Often it is necessary to use several element types to perform an analysis. For example it may be necessary to use both truss and frame (bending resistant) elements to perform an analysis. As developed in Section 5.3, the truss element has one degree-of-freedom for each spatial dimension, whereas, the frame element must have additional unknowns to represent the bending behavior. For nodes connected only to truss elements it is not necessary to have the additional degrees-of-freedoms active and a user would be required to specify restraint conditions for these nodes and degrees-of-freedom. By inserting the following lines of code into the truss element routine for the \( \text{iSW} = 1 \) task \( \text{FEAP} \) will automatically eliminate the degrees-of-freedom with values greater than \( \text{ndm} \) (the spatial dimension of the mesh).
This avoids the need to specify appropriate fixed boundary conditions for the unused values.

Instead, if one wishes to set the active degrees-of-freedom at each individual node of an element it is necessary to dimension the array as \( \text{ix} (n \text{df}, \ast) \). In this form the first column corresponds to the global pattern described above and columns 2 to \( \text{n} \text{en} + 1 \) are associated with the local element nodes 1 to \( \text{n} \text{en} \). The element degrees-of-freedom are then assigned to each node individually by assigning a 1 for an active degree-of-freedom or 0 for an inactive one. Note when using this option: Do not make changes to the first column of the \( \text{ix} \) array.

**Example: 3-node element with 3-dof/node**

Consider a problem with three degrees-of-freedom and three nodes on each element. It is desired to have degrees-of-freedom 1 and 3 active on node 2 and degree-of-freedom 2 active on nodes 1 and 3. This is accomplished by setting the \( \text{ix} \) array values as:

\[
\begin{align*}
\text{ix}(1,2) & = 0 \quad \text{! For node 1} \\
\text{ix}(2,2) & = 1 \\
\text{ix}(3,2) & = 0 \\
\text{ix}(1,3) & = 1 \quad \text{! For node 2} \\
\text{ix}(2,3) & = 0 \\
\text{ix}(3,3) & = 1 \\
\text{ix}(1,4) & = 0 \quad \text{! For node 3} \\
\text{ix}(2,4) & = 1 \\
\text{ix}(3,4) & = 0.
\end{align*}
\]

Note that for \( \text{isw} = 1 \) the \( \text{ix} \) parameter is not used to pass the nodal connection array but is used to return the list of unused degrees-of-freedom.

**Setting element plot sequence**

Utility routines are also provided to provide the necessary list of nodes needed to properly draw the mesh for each element type during plot outputs. The names of the routines are listed in Table 5.11 and each routine is called as
CHAPTER 5. ADDING ELEMENTS

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLTLN2</td>
<td>2-node line element</td>
</tr>
<tr>
<td>PLTLN3</td>
<td>3-node line element</td>
</tr>
<tr>
<td>PLTRI3</td>
<td>3-node triangular element</td>
</tr>
<tr>
<td>PLTRI6</td>
<td>6-node triangular element</td>
</tr>
<tr>
<td>PLQU4</td>
<td>4-node quadrilateral element</td>
</tr>
<tr>
<td>PLQU8</td>
<td>8 or 9-node quadrilateral element</td>
</tr>
<tr>
<td>PLTQ16</td>
<td>16-node quadrilateral element</td>
</tr>
<tr>
<td>PLTET4</td>
<td>4-node tetrahedron element</td>
</tr>
<tr>
<td>PLTET10</td>
<td>4-node tetrahedron element</td>
</tr>
<tr>
<td>PLBRK8</td>
<td>8-node brick element</td>
</tr>
<tr>
<td>PLBK27</td>
<td>27-node brick element</td>
</tr>
<tr>
<td>PLBKPR</td>
<td>64-node brick element</td>
</tr>
</tbody>
</table>

Table 5.11: Element Plot Definition Subprograms

call plname (iel)

where iel is an integer parameter defined in common eldata. If no call to a sub-program is included each element is assumed to be a 4 to 9 node quadrilateral and a default drawing order will be assigned. Users may construct their own drawing order also by following the steps employed in one of the routines defined in Table 5.11.

5.4.2 Task 6 Options

The TPL0t solution command includes an option to save specific element quantities (e.g., stress, strain, etc.). This option is implemented for user elements by including the common

```fortran
real*8       tt
common /elplot/ tt(1000)
```

and then inserting the statement

```fortran
  tt(i) = value
```

at an appropriate location in the isw = 3 task.

For example if it were desired to save the force and strain in the truss element the statements
tt(1) = EA*eps ! Element axial force
tt(2) = eps ! Element axial strain

could be placed anywhere after the stress and strain are defined. These values would be output by using a solution command sequence such as

```
batch
tplot
end
stress,nn,1 ! saves force for element nn
stress,nn,2 ! saves strain for element nn
show ! writes tplot items to output file
```

## 5.5 Projection of element variables to nodes

The STRESS,NODE solution command and the PLOT,STRESS command require a projection of element variables to nodes. A continuous stress field is assumed to obtain the nodal values. Accordingly,

\[ \sigma = N_\alpha \sigma_\alpha \]

where \( \sigma \) is any value which is to be projected to nodes (e.g., a stress or strain), \( N_\alpha \) are shape functions for the element type considered, and \( \sigma_\alpha \) nodal values of the projected quantity. The projection routine uses a diagonal weight matrix to project the values. For simple elements the matrix is computed by a procedure identical to mass lumping. For example,

\[ M_{aa} = \int_\Omega N_\alpha d\Omega \]

defines a ‘row sum’ form of projection\[^2\,3\]. Using the above results in the set of equations and a least square fit with the finite element values \( \hat{\sigma} \) gives the equation set

\[ M_{aa} \sigma_\alpha = \int_\Omega N_\alpha \hat{\sigma} d\Omega . \]

This defines nodal values for projected quantities. Since the coefficient matrix is diagonal the solution to the set of equations for each component is trivial. The actual solution is performed automatically by FEAP.

To permit each element to project its own quantities it is necessary to add the projection operations for each element under ISW = 8.\[^3\] These are performed locally for

\[^3\]An implementation of the Zienkiewicz-Zhu projection method is implemented using ISW = 24.
each element similar to all other operations. Figure 5.8 shows a simple routine for two-dimensional elements with 4-stress components begin projected. When multiple element types are used in an analysis users must be careful to project like quantities to common values of the $ST(nen,\star)$ array so as to get correct results. Also, when results are displayed it is necessary to plot results by material type to obtain correct indications of stress discontinuities at material interfaces.

## 5.6 Elements with History Variables

*FEAP* provides options for each element to manage variables which must be saved during the solution. These are history variables and are separated into three groups: (a) Variables associated with the last converged solution time $t_n$; (b) Variables associated with the current solution time $t_{n+1}$; and variables which are not associated to any particular time. All history variables are associated with the allocation name $H$ which has a pointer value 49. Users are not permitted direct access to the data stored as $H$ (of course, it is possible to access from $hr(np(49))$ but this should not normally be attempted!). Before calling the element routine for each element, *FEAP* transfers the required history variable to a local storage for each type. Users may then access the history data for each element and if necessary update values and return them *FEAP*. Only for specific actions will the local history data be transferred back to the appropriate $H$ locations. The element history data associated with $t_n$ starts at the memory address of the pointer for $NH_1$ using the double precision dummy array $HR$ in blank common; similarly data for $t_{n+1}$ starts at the memory address of the pointer for $NH_2$, and that not associated with a time at $NH_3$. The three pointers are passed to each element routine in the labeled common

```fortran
integer nh1,nh2,nh3
common /hdata/ nh1,nh2,nh3
```

Parameters of type $NH_3$ may not be used in material model routines $UMATLn$.

### 5.6.1 Assigning amount of storage for each element

The specification for the amount of history information to be associated with each material set is controlled in the $isw = 1$ task of an element routine. For each material type specified within the element routine a value for the length of the $NH_1$ and the $NH_3$ data must be provided (the amount of $NH_2$ data will be the same as for $NH_1$). This is
accomplished by setting the variables nh1 and nh2 in common hdata (see above) to
the required values. That is, the statements required are:

```fortran
if(isw .eq. 1) then
    nh1 = 6
    nh3 = 10
```

reserves 6 words of NH1 and NH2 data and 10 words of NH3 data for each element with
the current material number. Care should be taken to minimize the number of history
variables since, for very large problems, the memory requirements can become large,
thus reducing the size of problem that FEAP can solve.

### 5.6.2 Accessing history data for each element

As noted above the data for each element is contained in arrays whose first word is
located at hr(nh1), hr(nh2) (where nh1 and nh2 are pointers) for \( t_n, t_{n+1} \), respectively;
and at hr(nh3) for that not associated with time (note that there are values for each
only if non-zero values are assigned to nh1 and/or nh3 during the isw = 1 task. Any
other allocated data follows immediately after each first word It is a users responsibility
to manage what is retained in each variable type; however, the order of placing the \( t_n \)
and \( t_{n+1} \) data into the NH1 and NH2 arrays should be identical. There are no provisions
to store integer history variables separately from double precision quantities. It is
necessary to cast the integer data as double precision and move to the history location.
For example, using the statement

```fortran
hr(nh3+5) = dble(ivarbl)
```

saves the value for the integer variable ivarbl in the sixth word of the NH3 element
history array. At a subsequent iteration for this element the value of the integer would
be recovered as

```fortran
ivarbl = int(hr(nh3+5))
```

While this wastes storage for integer variables, experience indicates there is little need
to save many integer quantities and, thus, it was not deemed necessary to provide for
integer history variables separately.
Although users may define new values for any of the \texttt{hr(nh1)}, \texttt{hr(nh2)}, or \texttt{hr(nh3)} types, the new quantities will be returned to the \texttt{H} history for the element only for \texttt{isw} tasks where residuals are being formed for a solution step (i.e., solution command \texttt{FORM}, \texttt{TANG,,1}, or \texttt{UTAN,,1} and for history reinitialization during a time update (i.e., solution command, \texttt{TIME}). These access the task options \texttt{isw} equal to 3 or 6 and 14, respectively.

If a user adds a new option for which it is desired to save the history variables, it is necessary to set the variables \texttt{hf1gu} and \texttt{hf3flgu} to true as required, if no update is wanted the variables should be set to false. These parameters are located in

\begin{verbatim}
  logical       hf1gu,hf3flgu
  common /hdatam/ hf1gu,hf3flgu
\end{verbatim}

### 5.7 Elements with Finite Rotation Parameters

When considering structural elements that undergo large displacements it is usually necessary to treat the rotation parameters for large angle changes. The nodal parameters for this case are a rotation vector $\theta$ and the finite rotations are given as an orthogonal matrix $\Lambda$.

\[
\Lambda_{n+1} = \exp[\hat{\theta}] \Lambda_n
\]

in which $\hat{\theta}$ denotes a skew matrix given as

\[
\hat{\theta} = \begin{bmatrix}
0 & \theta_3 & -\theta_2 \\
-\theta_3 & 0 & \theta_1 \\
\theta_2 & -\theta_1 & 0
\end{bmatrix}
\]

The actual method used to update the rotations and their increments must be specified when writing the element module \texttt{ELMTnn} and is performed by a user subprogram named \texttt{UROTmn} where \texttt{mm} is a number between 01 and 10. To specify which routine is to be used, it is necessary to include the statement

\[
\texttt{rotyp = mm}
\]

in the section of \texttt{ELMTnn} where \texttt{isw = 1}. This parameter is located in the common \texttt{erotasi} which has the structure:

\begin{verbatim}
  real*8       xln
  real*8       rots ,rvel ,racc ,thkl
\end{verbatim}
The other entries in the common are arrays that return values for each element to treat the rotation values and rates. We shall return to their description after describing the treatment of the global nodal data for rotations.

### 5.7.1 Nodal rotation treatment: UROTmm subprogram

The nodal rotation data is stored in the array \(xlg\) which is dimensioned as

\[xlg(9,6,numnp)\]

For node \(ng\), the entries in \(xlg\) are stored as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLG(1:9,1,ng)</td>
<td>I</td>
<td>Rotation matrix (\Lambda_n) at time (t_n) (Alternatively, entries 1 to 4 may be used to store a quaternion).</td>
</tr>
<tr>
<td>XLG(1:9,2,ng)</td>
<td>O</td>
<td>Rotation matrix (\Lambda_{n_a}) at time (t_{n_a})</td>
</tr>
<tr>
<td>XLG(1:9,3,ng)</td>
<td>O</td>
<td>Rotation matrix (\Lambda_{n+1}) at time (t_{n+1})</td>
</tr>
<tr>
<td>XLG(1:3,4,ng)</td>
<td>O</td>
<td>Rotation increment angle (\Delta\theta)</td>
</tr>
<tr>
<td>XLG(4:6,4,ng)</td>
<td>I</td>
<td>Rotation rate (\omega_n) at time (t_n)</td>
</tr>
<tr>
<td>XLG(7:9,4,ng)</td>
<td>I</td>
<td>Rotation acceleration (\alpha_n) at time (t_n)</td>
</tr>
<tr>
<td>XLG(1:3,5,ng)</td>
<td>O</td>
<td>Rotation angle (\theta)</td>
</tr>
<tr>
<td>XLG(4:6,5,ng)</td>
<td>O</td>
<td>Rotation rate (\omega_{n+a}) at time (t_{n+a})</td>
</tr>
<tr>
<td>XLG(7:9,5,ng)</td>
<td>O</td>
<td>Rotation acceleration (\alpha_{n+a}) at time (t_{n+a})</td>
</tr>
<tr>
<td>XLG(1:9,6,ng)</td>
<td>-</td>
<td>Rotation matrix (\Lambda_0) at time (t_0)</td>
</tr>
</tbody>
</table>

While storage is provided for the \(3 \times 3\) rotation matrices the representation may also be specified in terms of quaternions for which only 4 components are necessary. In this case the 9 entries may be divided into two 4 entry quantities if required. Indeed, the space may be used in anyway necessary provided, no conflict in the way each time value is associated to the data. Note that sufficient storage is available to define integration methods for which the rotation is defined at an intermediate time \(t_{n+a}\).

For a typical node \(n\) in the mesh the location of the entries in the \(xlg\) array are obtained from
\[ \text{ng} = \text{mropt}(n, 2) \]

and the routine \text{UROTmm} is called as:

\[
\text{call urotmm}(xlg(1,1,\text{ng}), xlg(1,2,\text{ng}), xlg(1,3,\text{ng}), \]
\& \quad xlg(1,4,\text{ng}), xlg(1,5,\text{ng}), \]
\& \quad xlg(4,4,\text{ng}), xlg(4,5,\text{ng}), \]
\& \quad xlg(7,4,\text{ng}), xlg(7,5,\text{ng}), \text{du}, \text{tsw})
\]

where \text{du}(1:3) are the solution increments for rotation from the solver and \text{tsw} is the time update switch which is set according to

\[
\text{tsw} = 1: \text{Initialize for new time step} \\
\text{tsw} = 2: \text{Update within a time step} \\
\text{tsw} = 3: \text{Back up to beginning of time step}
\]

The entry \( u(1) \) is the location for the first entry in the rotation vector \( \theta \).

### 5.7.2 Local nodal rotation treatment

When each element that is associated with nodal rotation parameters is called the rotation data is transferred to local storage in a manner similar to treatment of translations. The local data is passed to each element using the common \text{erotas} defined above. The entries in the local arrays are extracted from the global array according to:

\[
xln(1:9,\text{nl},1) = xlg(1:9,1,\text{ng}) \\
xln(1:9,\text{nl},2) = xlg(1:9,2,\text{ng}) \\
xln(1:9,\text{nl},3) = xlg(1:9,3,\text{ng}) \\
xln(1:9,\text{nl},4) = xlg(1:9,6,\text{ng}) \\
\text{rots}(1:3,\text{nl},1) = xlg(1:3,4,\text{ng}) \\
\text{rots}(1:3,\text{nl},2) = xlg(1:3,5,\text{ng}) \\
\text{rvel}(1:3,\text{nl},1) = xlg(4:6,4,\text{ng}) \\
\text{rvel}(1:3,\text{nl},2) = xlg(4:6,5,\text{ng}) \\
\text{racc}(1:3,\text{nl},1) = xlg(7:9,4,\text{ng}) \\
\text{racc}(1:3,\text{nl},2) = xlg(7:9,5,\text{ng})
\]

where \text{nl} is a local node number between 1 and 9 (the maximum provided in the current \text{erotas} and \text{ng} is the global node number associated with each local number.
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPL(1) - EPL(3)</td>
<td>Linear momenta</td>
</tr>
<tr>
<td>EPL(4) - EPL(6)</td>
<td>Angular momenta</td>
</tr>
<tr>
<td>EPL(7)</td>
<td>Kinetic energy</td>
</tr>
<tr>
<td>EPL(8)</td>
<td>Stored energy</td>
</tr>
<tr>
<td>EPL(9)</td>
<td>Work by external loads</td>
</tr>
<tr>
<td>EPL(10)</td>
<td>Total energy</td>
</tr>
</tbody>
</table>

Table 5.12: Momenta and Energy Assignments

Using the above data structure one can program the updates in any manner that does not conflict with the time treatment. The only interface to FEAP is through how the increment $du(4:6, n)$ is defined.

### 5.8 Energy Computation

FEAP elements provide an option to accumulate the total momenta and energy during the solution process. The values are accumulated in the array EPL(20) when the switch parameter $isw$ is 13 and written to a file named $Pxxxx.ene$ (where $xxxx$ is extracted from the problem input filename) whenever the solution command TIME is used. The array EPL(2) is in the common block named ptdat6 which has the structure:

```plaintext
real*8 epl
integer iepl, neplts
common /ptdat6/ epl(20), iepl(2,200), neplts
```

For problems in solid mechanics the linear momenta are stored as follows:

The linear momenta are computed as:

$$ p = \int_{\Omega} \rho \mathbf{v} \, d\Omega $$

the angular momenta as:

$$ \pi = \int_{\Omega} (\mathbf{I} \omega + \mathbf{x} \times p) \, d\Omega $$

the kinetic energy

$$ K = \int_{\Omega} \rho \mathbf{v} \cdot \mathbf{v} \, d\Omega $$
the stored energy as

\[ U = \int_{\Omega} W(C) \, d\Omega \]

and the work by external loads as

\[ V = \int_{\Gamma} (x - X) \cdot F_{\text{ext}} \, d\Gamma. \]

The value of the displacement and velocity at the current time \( t_{n+1} \) are passed in \( u_l(i,j,1) \) and \( u_l(i,j,4) \), respectively. Note that this is true no matter which time integration algorithm is specified.

5.9 A Non-linear Theory for a Truss

A simple non-linear theory for a two or three dimensional truss which may undergo large displacements for which the strains remain small may be developed by defining the axial strain approximation in each member as

\[ \epsilon_{ss} = \frac{\partial u_s}{\partial s} + \frac{1}{2} \sum_{j=1}^{d-1} \left( \frac{\partial u_{nj}}{\partial s} \right)^2 \]

where \( u_{nj} \) is a displacement component normal to the axis of the member. The virtual strain from a linearization of the strain is given as

\[ \delta\epsilon_{ss} = \frac{\partial \delta u_s}{\partial s} + \sum_{j=1}^{d-1} \left( \frac{\partial \delta u_{nj}}{\partial s} \right) \left( \frac{\partial u_{nj}}{\partial s} \right). \]

An algorithm to define the two orthogonal unit vectors which are normal to the member may be constructed by taking

\[ \mathbf{v} = \mathbf{e}_k \]

where \( k \) is a direction for which a minimum value of the direction cosine \( l_i \) exists (for a 2-dimensional problem defined in the \( x_1, x_2 \) plane \( \mathbf{v} \) may be taken as \( \mathbf{e}_3 \)). Now,

\[ \mathbf{n}_1 = \frac{\mathbf{v} \times \mathbf{l}}{|\mathbf{v} \times \mathbf{l}|} \]

and

\[ \mathbf{n}_2 = \mathbf{1} \times \mathbf{n}_1. \]
Using these vectors the two normal components of the displacement are given by

\[ u_{nj}(s, t) = n_j \cdot u(s, t) = \sum_{i=1}^{d} n_{ji} u_i(s, t) \]

and the derivative by

\[ \frac{\partial u_{nj}}{\partial s} = \sum_{i=1}^{d} n_{ji} \frac{\partial u_i}{\partial s} . \]

Collecting terms and combining with previously defined quantities the virtual strain may be written as

\[ \delta \epsilon_{ss} = \frac{\partial \delta u}{\partial s} \cdot [g] \]

where

\[ g = 1 + \sum_{j=1}^{d-1} \frac{\partial u_{nj}}{\partial s} n_j . \]

After differentiation of the displacement field the discrete form of the virtual strain is given by

\[ \delta \epsilon_{ss} = \frac{1}{L} \left[ \delta u_1 \quad \delta u_2 \right] \cdot \left[ -g \quad g \right] . \]

Substituting the above virtual strain expression into the weak form gives the modified residual expression

\[ R_i = \frac{1}{2} b_i L \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] - \sigma_{ss} A \left[ \begin{array}{c} -g_i \\ g_i \end{array} \right] - \rho A L \frac{2}{6} \left[ \begin{array}{c} 1 \\ 2 \end{array} \right] \left[ \begin{array}{c} \ddot{u}_{i1} \\ \ddot{u}_{i2} \end{array} \right] . \quad (5.3) \]

The tangent tensor is obtained by linearizing the residual as shown previously. The only part which is different is the term with \( \sigma_{ss} \). Noting that

\[ d\epsilon_{ss} = [g] \cdot \frac{\partial d\mathbf{u}}{\partial s} \]

and

\[ d\delta \epsilon_{ss} = \frac{\partial \delta \mathbf{u}}{\partial s} \cdot \left( \mathbf{n}_1 \otimes \mathbf{n}_1 + \mathbf{n}_2 \otimes \mathbf{n}_2 \right) \cdot \frac{\partial d\mathbf{u}}{\partial s} . \]

If the \( \mathbf{n}_i \) are constructed as column vectors then the tensor product becomes a matrix defined as

\[ \mathbf{G} = \mathbf{n}_1 \otimes \mathbf{n}_1 + \mathbf{n}_2 \otimes \mathbf{n}_2 = \mathbf{n}_1 \mathbf{n}_1^T + \mathbf{n}_2 \mathbf{n}_2^T . \]

With these definitions, the tangent matrix for the non-linear problem is given as

\[ \mathbf{K}_{ij} = \frac{EA}{L} \begin{bmatrix} -g_i & -g_j \\ g_i & g_j \end{bmatrix} + \frac{\sigma_{ss} A}{L^2} \begin{bmatrix} G_{ij} & -G_{ij} \\ -G_{ij} & G_{ij} \end{bmatrix} . \]
Notice that for the linear problem

\[ g_i = \frac{\Delta x_i}{L} \]

thus, the only difference between the linear and non-linear problem is the definition of \( \epsilon_{ss} \) in terms of displacements, the modification for geometric effects for the \( g_i \) and the second term on the tangent matrix which is sometimes called the geometric stiffness part.
CHAPTER 5. ADDING ELEMENTS

Figure 5.2: FEAP Element Common Blocks.
include 'bdata.h'
include 'cdata.h'
include 'counts.h'
include 'eldata.h'
include 'elplot.h'
include 'eltran.h'
include 'hdata.h'
include 'iofile.h'
include 'prstrs.h'
include 'tdata.h'
include 'pointer.h'
include 'comblk.h'

Figure 5.3: FEAP Element Common Blocks using Includes.

Figure 5.4: 2-Node Truss Element
if(isw.eq.3 .or. isw.eq.6) then

  c Compute element length

  L2= 0.0d0
  do i = 1,ndm
    L2 = L2 + (xl(i,2) - xl(i,1))**2
  end do
  L = sqrt(L2)

  c Compute strain-displacement matrix

  Lr = 1.d0/L2
  eps = 0.0d0
  do i = 1,ndm
    bb(i,1) = -(xl(i,2) - xl(i,1))*Lr
    bb(i,2) = -bb(i,1)
    eps = eps + bb(i,2)*(ul(i,2,1) - ul(i,1,1))
  end do

  c Compute mass terms

  cmd = rhoA*L/3.0d0
  cmo = cmd*0.5d0

  c Form body/inertia force vector (dm = prop. ld.)

  sigA = EA*eps*L
  body = 0.5d0*L*dm
  do i = 1,ndm
    r(i,1) = body*d(6+i) - bb(i,1)*sigA
    & - cmd*ul(i,1,5) - cmo*ul(i,2,5)
    r(i,2) = body*d(6+i) - bb(i,2)*sigA
    & - cmo*ul(i,1,5) - cmd*ul(i,2,5)
  end do

Figure 5.5: Element residual for two node truss
if(isw.eq.3) then
  c      Compute element length
  L2= 0.0d0
  do i = 1,ndm
    L2 = L2 + (xl(i,2) - xl(i,1))**2
  end do
  L = sqrt(L2)
  c      Form stiffness multiplier
  dd = ctan(1)*EA*L
  c      Compute strain-displacement matrix
  Lr = 1.d0/L2
  do i = 1,ndm
    bb(i,1) = -(xl(i,2) - xl(i,1))*Lr
    bb(i,2) = -bb(i,1)
    db(i,1) = dd*bb(i,1)
    db(i,2) = -db(i,1)
  end do
Figure 5.6: Truss Tangent Matrix. Part 1
c Compute stiffness terms (N.B. ndm < or = ndf)

i1 = 0
do ii = 1,2
  j1 = 0
  do jj = 1,2
    do i = 1,ndm
      do j = 1,ndm
        s(i+i1,j+j1) = db(i,ii)*bb(j,jj)
      end do
    end do
    j1 = j1 + ndf
  end do
i1 = i1 + ndf
end do

c Compute mass terms and correct for inertial effects

cmd = ctan(3)*rhoA*L/3.0d0
cmo = cmd*0.5d0
do i = 1,ndm
  j = i + ndf
  s(i,i) = s(i,i) + cmd
  s(i,j) = s(i,j) + cmo
  s(j,i) = s(j,i) + cmo
  s(j,j) = s(j,j) + cmd
end do
endif

Figure 5.7: Truss Tangent Matrix. Part 2
subroutine slcn2d(sig,shp,xsj,sg,lint,nel,nes, p,s)

c-----[----+----+----+----.---------------------------------------]

     Purpose: Project element variables to nodes

     Inputs:
     sig(nes,*) - Stresses at quadrature points
     shp(nel,*) - Shape functions at quadrature points
     xsj(*) - Volume element at quadrature points
     sg(3,*) - Gauss points (1,2) and weights (3)
     lint - Number of quadrature points
     nel - Number nodes on element
     nes - Dimension of stress array

     Outputs:
     p(nen) - Weights for 'lumped' projection
     s(nen,*) - Integral of variables

     implicit none

     include 'cdata.h' ! Contains 'nen'
     include 'strnum.h' ! Contains 'iste'

     integer i,lint,nel,nes
     real*8 xg,p(*),s(nen,*),xsj(*),sig(nes,*),shp(nel,*),sg(3,9)

     do l = 1,lint
     do i = 1,nel
       xg = shp(i,l)*xsj(l)
       p(i) = p(i) + xg
       s(i,1) = s(i,1) + sig(1,l)*xg
       s(i,2) = s(i,2) + sig(2,l)*xg
       s(i,3) = s(i,3) + sig(3,l)*xg
       s(i,4) = s(i,4) + sig(4,l)*xg
     end do ! i
     end do ! l
     iste = 4 ! Returns number projections

     end

Figure 5.8: Element variable projection routine
Chapter 6

UTILITY ROUTINES

The FEAP system includes several subprograms that can assist developers in writing new modules. In the next sections we describe some of the routines which perform numerical integration, compute shape functions and their derivatives, etc.

6.1 Numerical quadrature routines

Details on quadrature formula types and the layout and location of points and weights may be found in standard references. Here only the description of subroutine calls is included together with the available options on number of points.

6.1.1 One dimensional quadrature

Line integrals may be evaluated using Gaussian quadrature in which the approximation to an integral is given as

\[ \int_{-1}^{+1} f(\xi) \, d\xi \approx \sum_{l=1}^{L} f(\xi_l) W_l \]  

(6.1)

where \( \xi_l \) are quadrature points and \( W_l \) are the weights to be applied at each point. The weights satisfy the condition.

\[ \sum_{l=1}^{L} W_l = 2 . \]  

(6.2)
CHAPTER 6. UTILITY ROUTINES

The Gauss-Legendre formula has points $|\xi|_{l}$ which are all less than unity. The subprogram call

\[
\text{CALL INT1D ( L , SW )}
\]

in which \( L \) is assigned an integer value between 1 and 5 returns the points and weights are returned in the two dimensional array \( \text{SW}(2,\ast) \) of type \( \text{REAL*8} \): Points in \( \text{SW}(1,\ast) \) and weights in \( \text{SW}(2,\ast) \). The Gauss-Legendre formula integrates exactly polynomials up to order \( 2L - 1 \).

The Gauss-Lobato formula has two of its points at \(-1\) and \(1\) with the remainder in the interior of the interval. A routine to perform quadrature is obtain by using the call

\[
\text{CALL INT1DL ( L , SW )}
\]

in which \( L \) is assigned an integer value between 1 and 6. The values of the points and weights are returned in the two dimensional array \( \text{SW} \): Points in \( \text{SW}(1,\ast) \) and weights in \( \text{SW}(2,\ast) \).

6.1.2 Two dimensional quadrature

Two dimensional quadrature on quadrilateral domains may be performed by repeated one-dimensional integration. The two dimensional integrations are approximated by

\[
\int_{-1}^{1} \int_{-1}^{1} f(\xi,\eta) \, d\xi \, d\eta \approx \sum_{l=1}^{L} f(\xi_{l},\eta_{l}) W_{l} \quad (6.3)
\]

where \( L \) is the total of all quadrature points. A routine to compute \( n \times n \) order Gauss-Legendre quadrature is obtained by the call

\[
\text{CALL INT2D ( L , LINT , SW )}
\]

where \( L \) is assigned to the number of points in each direction, \( \text{LINT} \) is returned as the total number of points and \( \text{SW}(3,\ast) \) is an array containing the points and weights according to: \( \text{SW}(1,1) \) contains values of the points \( \xi_{l} \); \( \text{SW}(2,1) \) contains values of the points \( \eta_{l} \); and \( \text{SW}(3,1) \) contains values of the weights \( W_{l} \).

Two dimensional quadrature on triangles may be performed using the subprograms call
CALL TINT2D ( L , LINT, SW )

where \( L \) is a type indicator, \( LINT \) returns the number of points, and \( SW(4,*\) is an array which returns three area coordinates and the quadrature weight: \( SW(1,1) \) returns the area coordinate \( L_{1\ell} \) (as defined in [2, 3]); \( SW(2,1) \) returns the area coordinate \( L_{2\ell} \); \( SW(3,1) \) returns the area coordinate \( L_{3\ell} \); \( SW(4,1) \) returns the weight \( W_{\ell} \); Table 6.1 describes the admissible types, number and location of quadrature points.

### 6.1.3 Three dimensional quadrature

Three dimensional quadrature on brick domains may be performed by repeated one-dimensional integration. The three dimensional integrations are approximated by

\[
\int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta, \zeta) \, d\xi \, d\eta \, d\zeta \approx \sum_{l=1}^{L} f(\xi_l, \eta_l, \zeta_l) W_l \tag{6.4}
\]

where \( L \) is the total of all quadrature points. A routine to compute \( n \times n \times n \) order Gauss-Legendre quadrature is obtained by the call

CALL INT3D ( L , LINT, SW )

where \( L \) is assigned to the number of points in each direction, \( LINT \) is returned as the total number of points and \( SW(4,\ast) \) is an array containing the points and weights according to: \( SW(1,1) \) contains values of the points \( \xi_l \); \( SW(2,1) \) contains values of the
### 6.2 Shape function subprograms

Finite element approximations commonly use shape function subprograms to perform computations of the functions and their derivatives at preselected points (often the quadrature points). \textit{FEAP} includes options to obtain the shape functions for some low order elements (linear and quadratic order) in one and two dimensions and linear shape functions for three dimensions. In addition a cubic Hermitian interpolation routine is available. The calling arguments for routines is summarized below.
6.2.1 Shape functions in one-dimension

The shape functions for one-dimensional elements, as shown in Fig. 6.1, may be computed using the shape function routines described below.

![Diagram of 2-Node Element and 3-Node Element](image)

Figure 6.1: Line type elements in FEAP library

Lagrangian interpolation in one-dimensional isoparametric forms may be obtained using the call

```fortran
CALL SHP1D ( S , XL , SHP, NDM, NEL, XJAC )
```

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Natural coordinate $\xi$</td>
</tr>
<tr>
<td>XL(NDM,*)</td>
<td>Nodal coordinates for element</td>
</tr>
<tr>
<td>NDM</td>
<td>Spatial dimension of mesh</td>
</tr>
<tr>
<td>NEL</td>
<td>Number element nodes (2 or 3)</td>
</tr>
<tr>
<td>SHP(2,NEL)</td>
<td>Shape function and derivative</td>
</tr>
<tr>
<td>XJAC</td>
<td>Jacobian transformation</td>
</tr>
</tbody>
</table>

The shape functions are evaluated as: $\text{SHP}(1,i)$ shape function derivative along the axis of the element and $\text{SHP}(2,i)$ the shape function $N_i$. In calculations integrals are represented as

$$
\int_{L} f(N_i, N_{i,s}) \, ds = \int_{-1}^{1} f[N_i(\xi), N_{i,s}(\xi)] XJAC(\xi) \, d\xi \tag{6.5}
$$

and quadrature may be used for evaluation.

Calculation of natural coordinate derivatives only may be obtained with the call
CALL SHAP1DN(S, SHP, NEL)

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Natural coordinate ξ</td>
</tr>
<tr>
<td>SHP(2,NEL)</td>
<td>Shape function and derivative</td>
</tr>
<tr>
<td>NEL</td>
<td>Number element nodes (2 or 3)</td>
</tr>
</tbody>
</table>

where SHP(1,i) contains $N_{i,\xi}$ and SHP(2,i) the shape function $N_i$.

Cubic Hermitian interpolation (e.g., for use in straight linear beam elements) given by

$$w = N_1^w \bar{w}_1 + N_2^w \bar{w}_2 + N_1^\theta \bar{\theta}_1 + N_2^\theta \bar{\theta}_2$$

(6.6)
is obtained using the call

CALL SHP1DH(S, LEN, SHPW, SHPT)

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Natural coordinate ξ</td>
</tr>
<tr>
<td>LEN</td>
<td>Length of the element (2-node)</td>
</tr>
<tr>
<td>SHPW(4,2)</td>
<td>Shape functions for $w_i$</td>
</tr>
<tr>
<td>SHPT(4,2)</td>
<td>Shape functions for $\theta_i$</td>
</tr>
</tbody>
</table>

The arrays are evaluated as follows:

1. SHPW(1,i), SHPT(1,i) are first derivatives (e.g. $N_{i,x}$);
2. SHPW(2,i), SHPT(2,i) are second derivatives (e.g. $N_{i,xx}$);
3. SHPW(3,i), SHPT(3,i) are third derivatives (e.g. $N_{i,xxx}$); and
4. SHPW(4,i), SHPT(4,i) are shape functions (e.g. $N_i$).

6.2.2 Shape functions in two-dimensions

The shape functions for two dimensional triangular elements, as shown in Fig. 6.2, and quadrilateral elements, as shown in Fig. 6.3, may be computed using the shape function routines described below.
Two-dimensional $C_0$ isoparametric interpolation on quadrilaterals of linear, quadratic
and cubic order may be obtained using the subprogram call

\[
\text{CALL SHP2D ( SS, XL, SHP, XJAC, NDM, NEL, IX, FLG )}
\]

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS(2)</td>
<td>Natural coordinates $\xi$, $\eta$</td>
</tr>
<tr>
<td>XL(NDM, NEL)</td>
<td>Element coordinates in local order</td>
</tr>
<tr>
<td>NDM</td>
<td>Spatial dimension mesh (2 or 3)</td>
</tr>
<tr>
<td>NEL</td>
<td>Number nodes on element (4-9, 12, 16)</td>
</tr>
<tr>
<td>IX(NEL)</td>
<td>Element global node numbers</td>
</tr>
<tr>
<td>FLG</td>
<td>Return $\xi - \eta$ derivatives if true or $x - y$ derivatives if false</td>
</tr>
<tr>
<td>SHP(3, NEL)</td>
<td>Shape derivatives and derivatives</td>
</tr>
<tr>
<td>XJAC</td>
<td>Jacobian transformation from $x - y$ to $\xi - \eta$.</td>
</tr>
</tbody>
</table>

The array SHP stores the values in the order: SHP(1,i) derivative with respect to $\xi$ or $x$; SHP(2,i) derivative with respect to $\eta$ or $y$; SHP(3,i) shape function.

Two-dimensional $C_0$ isoparametric interpolation on triangles of linear, quadratic and cubic order may be obtained using the subprogram call

\[
\text{CALL TRISHP ( SS, XL, NDM, IORD, XJAC, SHP )}
\]

where

![3-Node Simplex and 6-Node Element](image)

Figure 6.2: Triangular surface type elements in FEAP library
Figure 6.3: Quadrilateral surface type elements in FEAP library

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS(3)</td>
<td>Area coordinates $L_1, L_2, L_3$</td>
</tr>
<tr>
<td>XL(NDM,*)</td>
<td>Element coordinates in local order</td>
</tr>
<tr>
<td>NDM</td>
<td>Spatial dimension mesh (2 or 3)</td>
</tr>
<tr>
<td>IORD</td>
<td>Order of interpolation (1= 3-node, 2 = 6-node, 3 = 7-node, 4 = 6-node + 3 bubble, 10 = 10-node cubic)</td>
</tr>
<tr>
<td>XJAC</td>
<td>Jacobian transformation from $x - y$ to $\xi - \eta$</td>
</tr>
<tr>
<td>SHP(3,NEL)</td>
<td>Shape functions and derivatives</td>
</tr>
</tbody>
</table>
The array \texttt{SHP} stores the values in the order: \texttt{SHP(1,i)} derivative with respect to \(\xi\) or \(x\); \texttt{SHP(2,i)} derivative with respect to \(\eta\) or \(y\); \texttt{SHP(3,i)} shape function. The parameter \texttt{IORD} defines the order of interpolation. If it is 1 simple 3-node triangles with linear interpolation is returned; if 2 quadratic interpolation; if 3 the interpolation is generated plus a cubic bubble in the seventh function. Giving the \texttt{IORD} parameter as a negative returns hierarchical form for mid side nodes.

### 6.2.3 Shape functions in three-dimensions

The shape functions for three dimensional tetrahedral elements, as shown in Fig. 6.4, and brick elements, as shown in Fig. 6.5, may be computed using the shape function routines described below.

Three-dimensional \(C_0\) isoparametric interpolation on bricks of linear order (i.e., 8-node elements) may be obtained using the subprogram call

```
CALL SHP3D ( SS, XJAC, SHP, XL, NDM, NEL )
```

where

![Diagram](image_url)

**Figure 6.4**: Tetrahedron solid type elements in \textit{FEAP} library
The array $\text{SHP}$ stores the values in the order: $\text{SHP}(1,i)$ derivative with respect to $x$; $\text{SHP}(2,i)$ derivative with respect to $y$; $\text{SHP}(3,i)$ derivative with respect to $z$; $\text{SHP}(4,i)$ shape function.

Three-dimensional $C_0$ isoparametric interpolation on tetrahedra of linear order (i.e., 4-node elements) may be obtained using the subprogram call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS(3)</td>
<td>Natural coordinates $\xi, \eta, \zeta$</td>
</tr>
<tr>
<td>XL(NDM,8)</td>
<td>Element coordinates in local order</td>
</tr>
<tr>
<td>NDM</td>
<td>Spatial dimension mesh (2 or 3)</td>
</tr>
<tr>
<td>NEL</td>
<td>Number nodes on element: 8 = linear brick; 20 = serendipity quadratic; 27 = Lagrangian quadratic; 64 = Lagrangian cubic</td>
</tr>
<tr>
<td>SHP(4,8)</td>
<td>Shape functions and derivatives</td>
</tr>
<tr>
<td>XJAC</td>
<td>Jacobian transformation from $xyz$ to $\xi\eta\zeta$.</td>
</tr>
</tbody>
</table>
CALL TETSHP ( SS, XL, NDM, NEL, XJAC, SHP )

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS(4)</td>
<td>Volume coordinates $L_1, L_2, L_3, L_4$</td>
</tr>
<tr>
<td>XL(NDM,4)</td>
<td>Element coordinates in local order</td>
</tr>
<tr>
<td>NDM</td>
<td>Spatial dimension mesh (3)</td>
</tr>
<tr>
<td>NEL</td>
<td>Number of nodes on element (4, 10, 11, 14, 15)</td>
</tr>
<tr>
<td>XJAC</td>
<td>Jacobian transformation from $xyz$ to $\xi\eta\zeta$</td>
</tr>
<tr>
<td>SHP(4,4)</td>
<td>Shape functions and derivatives</td>
</tr>
</tbody>
</table>

The array SHP stores the values in the same order as for the brick element.

### 6.3 Eigenvalues for $3 \times 3$ matrix

Three dimensional problems often require the solution of a $3 \times 3$ eigenproblem to generate principal values and directions. *FEAP* includes a special routine to calculate the values and vectors for symmetric arrays. The routine is used by a call to the subprogram as

```
CALL EIG3 ( V, D, ROT )
```

On call to the routine $V(3,3)$ is a *REAL*8 array containing the symmetric array to be diagonalized. On return the eigenvalues are contained in $D(3)$ and the vectors for each value in the columns of the $V$ array. A Jacobi method is used with $ROT$ an integer parameter returning the number of rotations to diagonalize. The routine is quite efficient compared to any attempt to compute vectors after closed form solution of the cubic for roots.

In addition to the general eigensolution above *FEAP* includes options to compute principal values of a symmetric second order tensor for two and three dimensional problems. In two dimensional use, the call to

```
CALL PSTR2D ( SIG, PV )
```

is used where $SIG(4)$ stores stresses in the order $\sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}$ and returns principal values and directions in $PV(3)$ in the order $\sigma_1, \sigma_2,$ and $\theta$, where the angle is in degrees between $x$ and the 1-axis. This routine does not use $SIG(3)$. 
In three dimensions the principal values are obtained using the call

```
CALL PSTR3D ( SIG, PV )
```

where SIG(6) stores stresses in the order $\sigma_{11}$, $\sigma_{22}$, $\sigma_{33}$, $\sigma_{12}$, $\sigma_{23}$, $\sigma_{31}$, and returns principal values in PV(3) in the order $\sigma_1$, $\sigma_2$, $\sigma_3$. Roots are ordered from most positive to most negative.

### 6.4 Plot routines

Several options exist in the FEAP system to create graphical plots for data and results.

#### 6.4.1 Mesh plots

FEAP has plot capabilities to represent some standard element shapes. For continuum elements where the shape of the element is identical to the space dimension of the mesh (i.e., $\text{NDM}$) it is not necessary to provide any extra descriptions. However, if the dimension of the element topology is different from the mesh dimension it is necessary to add the include statement

```
include 'eldata.h'
```

and the statement

```
pstyp = pdim
```

within the $\text{ISW} = 1$ part of the element routine. Here $\text{pdim}$ is the dimension of the element topology. For example, in a three dimensional shell problem where $\text{NDM} = 3$ and the element topology is two dimensional the statement is given as

```
pstyp = 2
```

Provided the nodal numbering of an element is as described in the FEAP User manual (i.e., numbered with vertex nodes first, followed by mid-side nodes, then face nodes and finally internal nodes) the program can use the actual number of nodes on the element to draw each element.
Failure to include a `pstyp` statement will usually result in unpredictable plots of the mesh and contour values.

The known types of plots are:

1. **Point element** with one node obtained by call

   ```
   CALL PLPT1 ( IEL )
   ```

2. **Line element** with two nodes obtained by call

   ```
   CALL PLTN2 ( IEL )
   ```

   and for three node elements

   ```
   CALL PLTN3 ( IEL )
   ```

3. **Triangular element** with 3-nodes obtained by call

   ```
   CALL PLTRI3 ( IEL )
   ```

   and for 6-nodes obtained by call

   ```
   CALL PLTRI6 ( IEL )
   ```

4. **Quadrilateral element** with 4-nodes obtained by call

   ```
   CALL PLQUD4 ( IEL )
   ```

   for 8- or 9-node elements the plot call is

   ```
   CALL PLQUD8 ( IEL )
   ```

   and for 12- or 16-node quadrilaterals the call is

   ```
   CALL PLTQ16 ( IEL )
   ```

5. **Tetrahedral element** with 4-nodes obtained by call

   ```
   CALL PLTET4 ( IEL )
   ```
and for 10-node tetrahedra the call is

\begin{verbatim}
CALL PLTET10( IEL )
\end{verbatim}

6. Brick element with 8-nodes obtained by call

\begin{verbatim}
CALL PLBRK8 ( IEL )
\end{verbatim}

and for 20- or 27-node bricks the call is

\begin{verbatim}
CALL PLBRK27( IEL )
\end{verbatim}

Using these and internal extraction of element surfaces the program is able to make some hidden surface plots in three dimensions.

### 6.4.2 Element data plots

Users may construct plots within their elements (i.e., an \texttt{ELMTnn}) and access using the plot command:

\begin{verbatim}
PLOT,PELE,v1,v2,v3
\end{verbatim}

In interactive mode in the plot environment it is only necessary to enter

\begin{verbatim}
PELE,v1,v2,v3
\end{verbatim}

The values entered in \texttt{v1,v2,v3} are optional and are passed to the element through a common block as

\begin{verbatim}
REAL*8 ELPLT
COMMON /ELPDAT/ ELPLT(3)
\end{verbatim}

The \texttt{PELE} option calls each element with the switch parameter \texttt{ISW = 20}. Users merely code whatever option they wish to include within their element module.

The standard color table is available through use of the subroutine call

\begin{verbatim}
CALL PPPCOL(ICOL, 0)
\end{verbatim}
in which \texttt{ICOL} designates the color to be assigned according to Table 6.3. An exception occurs for PostScript outputs where black and white are switched (since the background then is assumed to be white).

A straight line segment may be drawn to the screen in the current color between the coordinates \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) using the commands

\begin{verbatim}
CALL PLOTL(X1,Y1,Z1, 3)
CALL PLOTL(X1,Y1,Z1, 2)
\end{verbatim}

Here the basic command is

\begin{verbatim}
CALL PLOTL(X1,Y1,Z1, IP)
\end{verbatim}

where the three cartesian coordinates relate to mesh coordinates (not screen values) and \texttt{IP} is a parameter defined according to Table 6.4.

The perimeter of a panel is drawn with standard line drawing commands starting with

\begin{verbatim}
CALL PLOTL(X1,Y1,Z1, 1)
\end{verbatim}
and continuing with a sequence of draw commands

    CALL PLOTL(Xi,Yi,Zi, 2)

(however, no lines appear on the screen) and the fill of each panel is completed by the statement

    CALL CLPAN

It should be noted that all plots within FEAP are performed in three dimensions. For two dimensional problems no $z_i$ coordinates are available in the $XL(NDM,NEN)$ array and, hence, it is necessary to assign zero values for the $z_i$ coordinates before calling a plot subprogram. If a perspective view has been requested a full use of a $x_i, y_i, z_i$ specification is made. In this case a user may wish to pass the value of some solution variable as the $z_i$ value (scaled so that it will make sense relative to the $x_i, y_i$ coordinate values). Similarly, if deformed plots are being performed it is necessary to add (scaled) displacements to the coordinates. The current value of the scaling parameter (i.e., variable $CS$) is available in labeled common $PVIEW$. In this case one can add the statements (assuming here that the displacements correspond to the coordinate directions)

    DO NE = 1,NEL
       DO I = 1,NDM
          XP(I,NE) = XL(I,NE) + CS*UL(I,NE)
       END DO ! I
    END DO ! NE

(NEL is the number of connected nodes to each element and is passed through labeled common $ELDATA$) before performing any deformed plots and then plot the appropriate values of $XP$. Indeed, this may always be performed as the value of $CS$ will be zero for an undeformed plot.

### 6.4.3 Other user plots

It is also possible for users to prepare plot outputs unrelated to elements. The plot command

    PLOT UPLOt v1 v2 v3
CHAPTER 6. UTILITY ROUTINES

initiates a call to the subroutine UPL0T which has the basic structure

```fortran
SUBROUTINE UPL0T(CT)
 IMPLICIT NONE
 REAL*8 CT(3)
...
END
```

The argument CT contains the values for the three parameters v1, v2, v3. The default color is white. Direct plots in screen coordinates [lower left at (0,0); upper right at (1,1)] may be given using the statement

```fortran
CALL DPL0T(XS, YS, IP)
```

where XS, YS are between zero (0) and one (1) and IP is interpreted according to Table 6.4. Panels are closed using

```fortran
CALL CLPAN
```

and colors treated according to values specified in calls to PPC0L.
Chapter 7

Adding a user solver

ADDING USER SOLVERS

There are several public domain linear equation solution routines available at various internet locations. Examples are SuperLU, umfpack, xxx to name three. To access any of these solvers it is necessary to add user modules named umacr1.f and usolve.f to FEAP. The module umacr1.f (x ranges between 0 and 9) has the basic form

```
subroutine umacr1(lct,ctl,prt)
               include 'setups.h' ! for parameter 'solver'
               include 'umac1.h'   ! for parameter 'uct'

               logical   prt
               character lct*15
               real*8     ctl(3)

               if(pcomp(uct,'mac1',4)) then
                   uct = 'name'   ! Set name of command for solver
               else
                   if(pcomp(lct,'off',3)) then
                       solver = .true.  ! Sets flag for FEAP solvers
                       ... any other statements needed
                   else
                       solver = .false. ! Sets flag for user solver
                       ... any other statements needed
                   endif
               endif

               end
```

and the module usolve.f
subroutine usolve(flags,b)

c-----[--.--+++.-----+.-----------------]----------------------
c Purpose: Solver interface for SuperLU

c Inputs:
c   flags(1) - Allocation and/or initialization phase
   flags(2) - Perform factorization for direct solutions
   flags(3) - Coefficient array unsymmetric
   flags(4) - Solve equations
   flags(5) - Purge storage of pointers

c b(*) - RHS vector

c Outputs:
c   flags(5) - True if error occurs (for factor/solve only)

implicit none
logical flags(*)
real*8 b(*)

  Presolve setups
  if(flags(1)) then
    ...

  Solution steps for assembled equations
  else

  Factor equations
  if(flags(2)) then
    ...
  endif

  Perform solve
  if(flags(4)) then
    ...
  endif

  Purge storage in 'factor'
  if(flags(5)) then
    ...
  endif
endif
end
Bibliography


Index

Adding elements, 32
   ELMTnn subprogram, 32
Arguments on subprogram ELMTnn, 34
Array allocation, 8
   PALLOC subprogram, 9
   Removing arrays, 10
   UALLOC subprogram, 9
Color in plots, 30
Common block definitions, 38
Common statements, 3

Data input
   Converting text to real, 7
   PINPUT, 5
   TINPUT, 6

Data output
   Integer arrays, 7
   Real arrays, 7

Debug output, 2

Eigenvalues for $3 \times 3$ matrix, 80
Element array names, 12
Element connection array, 12
Element control array, 13
Element degree of freedom array, 13
Element energy computation, 60
Element history variables, 55
   Accessing for each element, 56

Element options
   Projection to nodes, 54
   Task 1 (ISW=1), 51
   Task 6 (ISW=6), 53

   TPLOt definitions, 53

   Element plots
      User, 83

   Element task definitions, 35

   Finite rotation, 57

   Include statements, 3

   Locating arrays
      PGETD, 14

   Material models, 18
      Auto time step control, 25
      History variables, 25
      Internal variables, 25
      UCON use, 18
      UMATIn subprogram, 20
      UMATLn subprogram, 20

   Material property variables, 36

   Mesh array names, 11

   Mesh input
      UMESHn subprogram, 15

   Mesh manipulation
      UMANIn subprogram, 27

   Mesh plots, 81
      Brick element, 83
      Line element, 82
      Point element, 82
      Quadrilateral element, 82
      Tetrahedral element, 82
      Triangular element, 82

   Newton solution
Element residual array storage, 45
Element tangent array storage, 45
Non-linear problems, 43
Tangent array definition, 44
Numerical integration, 70
  One dimension, 70
  Three dimension, 72
  Two dimension, 71
Partial list of element common block variables, 64
Plot command
  UPlot subprogram, 85
  UPlotN subprogram, 29
Plot utility function
  Fills, 84
  Lines, 84
Plot utility functions
  Color, 30
  Fills, 29
  Lines, 29
Precision
  Integer, 1
  Real, 1
Principal stress
  Three dimension, 80
  Two dimension, 80
Problem size, 1
Quadrature, 70
  One dimension, 70
  Three dimension, 72
  Two dimension, 71
Setting colors, 30
Setting options
  Help level, 2
  Parsing statements, 1
  Plot prompts, 2
  PostScript, 2
Shape functions, 73
  Brick isoparametric, 78
  One dimensional cubic Hermitian, 75
  One dimensional isoparametric, 74
  One dimensional natural derivative, 74
  Quadrilateral isoparametric, 75
  Tetrahedral isoparametric, 79
  Three dimensional isoparametric, 78
  Triangular isoparametric, 76
  Two dimensional isoparametric, 75
Solution array names, 11
Solution command
  UMACRn subprogram, 29
Table of colors, 31
Transient solution
  CTAN - definition, 47
  Newton method, 43
  Non-linear, 37
Truss element
  Newmark method, 51
  Non-linear theory, 61
  Tangent modulus, 62
  Theory, 47
  Variational equation, 49
  Weak form, 49