ELP
USER MANUAL

version 1.7
Septemper 2003

prepared by
Peter Iványi
Jelle Muylle

Heriot-Watt University
Structural Engineering Computational Technology (SECT) Research Group
Department of Mechanical and Chemical Engineering
Edinburgh, U.K.

Contents

List of Figures

Chapter 1

Introduction

This document contains the user manuals for the different tools bundled in ELP. ELP stands for E_Library Package. It is a series of general purpose finite element analysis and design tools developed by the Structural Engineering Computational Technology (SECT) research group at Heriot-Watt University Edinburgh. All the tools in ELP have in common that they were created to the same standard, the E_Library.

The purpose of this manual is to explain the usage and behaviour of the different components of ELP. It is not the aim of this manual to explain the code or supply developer information.

1.1 ELP programs

The following programs are described in the chapters of this manual:

CUBIC: a structured mesh generator, which is mainly used as a pre-processor for DR because of its capacity to handle typical types of membrane features, such as cables, g-strings and clothes. (Chapter 3)

FEM: a very basic, robust finite element analysis program for 2D meshes. (Chapter 4)

ADAPT: generates finite element errors and mesh parameters for an adaptive remeshing of a 2D mesh. (Chapter 5)

E_PLOT32: viewing and printing of meshes, subdomains and mesh parameters in an easy to use windows interface. (Chapter 6).

MDFTOOLS: a series of basic editing tools to check and modify mesh definition files (Chapter 7).

Chapter 2

E_Library file types

The E-Lib format supports meshes of a single element type and mixed meshes. The E-Lib mesh file format takes the form of a series of keywords followed by one or more data items. For example,

```
NELEMENTS_TRIANG1   120
```

denotes that the mesh contains 120 elements of type **TRIANG1**.

IMPORTANT Most keywords are optional, but those which are declared must be in a strict, predefined order. This order automatically accounts for dependencies between the data.

The format also allows the use of file comments. If the first non-white character on a line is the # character, then all remaining text on that line is ignored. Two or more lines can be commented out by enclosing the lines inside a /* and */. The opening /* must be the first two characters on the first comment line and */ the first two characters on the last comment line. All text on the line following /* and */ is ignored. The /*...*/ comments must not be nested, although they can contain # comments. Blank lines are also permitted.

There are currently seven types of data file – the mesh definition (**.mdf**), geometric definition (NURBS curve) (**.gmf**), materials (**.mat**), domain decomposition (**.dom**), element or nodal mesh parameters (**.mpr**), stresses (**.ste**) and finite element errors files(**.fee**). The mesh definition file contains the main description of the mesh. The other files describe additional properties of the mesh or they represent a state of the mesh.

The following sections describe the keywords and keyword data for the above data files. Each section contains a table which lists the keywords **in the order** they must be declared and defines the associated keyword data. The data type – whether it is an integer (i), real (r) or a character string (s) – is also given, where, for example, 'i 3*r' denotes that the data consist of an integer followed by three reals. Units, where relevant, are enclosed in square brackets [...].

IMPORTANT Keyword data consisting of a single data item must follow the keyword on the same line. For vectors and matrices, each vector component or matrix row must start on a new line and each line must begin with an integer index. Unless otherwise stated, this index is either the vector component number or matrix row index. All keyword data strings must be enclosed in double quotes.

2.1 The Mesh Definition File

The mesh definition file (**.mdf**) contains the main description of the mesh.

At present twelve different elements types are supported – five one dimensional (**LINK1**, **LINK2**, **LINK3**, **LINK4**, **LINK5**), five triangular (**TRIANG1**, **TRIANG2**, **TRIANG3**, **TRIANG4**, **TRIANG5**), two quadrilateral (**QUAD1** and **QUAD3**), one tetrahedral (**TETRAH1**) and one three dimensional block element (**BLOCK1**). These element types are defined in Figure 2.1 which shows the order in which the element nodes must be labelled and in Table 2.1 which gives a short description as well. With the exception of the **TRIANG2** and **QUAD3** elements, individual elements have uniform material properties. **TRIANG2** and **QUAD3** elements are composed of layered composite materials referred to as composite materials of type 1.

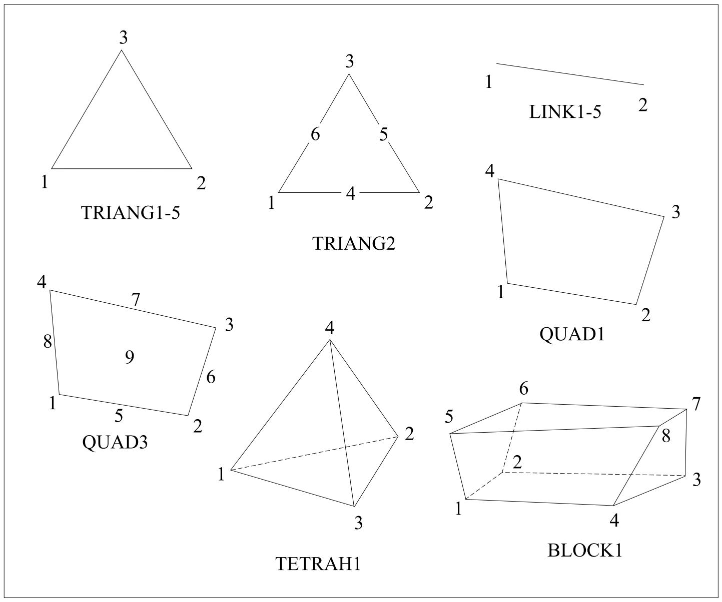


Figure 2.1: The element node orders

An example mesh (Figure 2.2) definition file is shown in the following

```
# An example mesh definition file

TITLE "An example mesh"
NMESHPOINTS        6
NNODES             6
NELEMENTS_TRIANG1  4

MESHPOINT_COORDINATES
  1     0.000     0.000     0.000
  2     3.000     0.000     0.000
  3     6.000     0.000     0.000
  4     0.000     3.000     0.000
  5     3.000     3.000     0.000
  6     6.000     3.000     0.000

NODES_TRIANG1
  1       1         2         4
  2       2         5         4
  3       2         3         5
  4       3         6         5
```

Element Type	Description	Number of Vertices	Number of Nodes
LINK1	Truss	2	2
LINK2	Cable	2	2
LINK3	Fixed tension 1-D link	2	2
LINK4	Fixed force density 1-D link	2	2
LINK5	Geodesic string	2	2
TRIANG1	Constant strain triangular	3	3
TRIANG2	A combined constant strain plane stress and constant moment plate bending simple facet triangular	3	6
TRIANG3	Solid triangular	3	3
TRIANG4	Membrane triangular	3	3
TRIANG5	Constant stress triangular	3	3
QUAD1	Plane stress quadrilateral	4	4
QUAD3	Mindlin plate quadrilateral	4	9
TETRAH1	Tetrahedral	4	4
BLOCK1	Block	8	8

Table 2.1: The element types

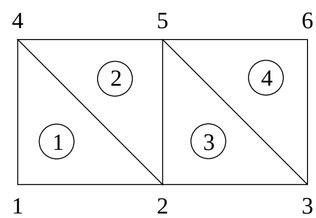


Figure 2.2: An example with TRIANG1 elements

The first keyword **TITLE**, is compulsory and specifies the title of the mesh. The title can be used to annotate output such as the display in a plot program and, as with all string arguments, must be enclosed in double quotes.

A mesh is defined in terms of a set of *mesh-points* which are joined by straight lines to form the edges or surfaces of the elements.

IMPORTANT A distinction is made between the terms *mesh-point*, *vertex* and *finite element node*. A mesh-point is a point in space which may or may not form an element vertex, whereas a finite element node (or simply *node*) is a point on an element used for function interpolation during the finite element analysis. Unlike mesh-points, nodes need not coincide with the element vertices. (See Figure 2.3.)

NMESHPOINTS is the keyword for the number of mesh-points defined in the mesh definition file and **NNODES** defines the total number of finite element nodes in the mesh. The coordinates of the mesh-points follow the keyword **MESHPOINT_COORDS**. All three of the keywords **NMESHPOINTS**, **NNODES** and **MESHPOINT_COORDS** are compulsory. The keyword **NELEMENTS_TRIANG1** act both to declare the type of elements in the mesh and to give the number of the element. If this number is zero then the keyword should be omitted. At least one of the keywords which defines the

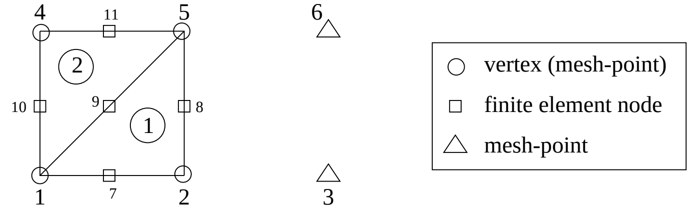


Figure 2.3: An example with TRIANG2 elements

element type must be declared and for each such keyword present there must also be the corresponding keyword like **NODES_TRIANG1** etc. The data for these keywords define the node indices of each element in turn in the order given in Figure 2.1. Element vertex nodes are always labelled before non-vertex nodes and for two dimensional elements, labelling occurs in an anti-clockwise direction.

Another example is presented to demonstrate the difference between *mesh-point*, *vertex* and *finite element node*. The geometric arrangament of the elements can be seen in Figure 2.3 while the generated mesh file is the following:

```
# Example with TRIANG 2 elements

TITLE "An example mesh"
NMESHPOINTS        6
NNODES             10
NELEMENTS_TRIANG2  2

MESHPOINT_COORDINATES
  1    0.000    0.000    0.000
  2    3.000    0.000    0.000
  3    6.000    0.000    0.000
  4    0.000    3.000    0.000
  5    3.000    3.000    0.000
  6    6.000    3.000    0.000

NODES_TRIANG2
  1      1        2        5        7        8        9
  2      1        5        4        9       11       10
```

NOTE *All other keywords are optional* except in cases where the declaration of one keyword implies that one or more further keywords will be declared later. For example, if the number of boundary condition nodes is defined (**NBOUNDARY_CONDITION_NODES**) then so must the nodal boundary conditions (**BOUNDARY_CONDITIONS**), see example below.

```
# Example with extra keywords

TITLE "An example mesh"
NMESHPOINTS                3
NNODES                     3
NELEMENTS_TRIANG1          1
NBOUNDARY_CONDITION_NODES  2

MESHPOINT_COORDINATES
  1    0.000    0.000    0.000
  2    6.000    0.000    0.000
  3    6.000    6.000    0.000

NODES_TRIANG1
  1      1        2        3

BOUNDARY_CONDITIONS
  1   "FIXED" 0.000   "FIXED" 0.000   "FREE"
  2   "FREE"          "FIXED" 0.000   "FREE"
```

An exhaustive list of possible keywords in an **.mdf** file is presented in Table 2.2a,b,c . For the use of keywords not discussed here see the relevant section in the documentation.

Keyword	Keyword Data	Data Type
TITLE	Mesh title	s
NMESHPOINTS	Number of mesh-points	i
NNODES	Number of nodes	i
NELEMENTS_LINK1	Number of **LINK1** elements	i
NELEMENTS_LINK2	Number of **LINK2** elements	i
NELEMENTS_LINK3	Number of **LINK3** elements	i
NELEMENTS_LINK4	Number of **LINK4** elements	i
NELEMENTS_LINK5	Number of **LINK5** elements	i
NELEMENTS_TRIANG1	Number of **TRIANG1** elements	i
NELEMENTS_TRIANG2	Number of **TRIANG2** elements	i
NELEMENTS_TRIANG3	Number of **TRIANG3** elements	i
NELEMENTS_TRIANG4	Number of **TRIANG4** elements	i
NELEMENTS_TRIANG5	Number of **TRIANG5** elements	i
NELEMENTS_QUAD1	Number of **QUAD1** elements	i
NELEMENTS_QUAD3	Number of **QUAD3** elements	i
NELEMENTS_TETRAH1	Number of **TETRAH1** elements	i
NELEMENTS_BLOCK1	Number of **BLOCK1** elements	i
NBOUNDARY_CONDITION_NODES	Number of boundary condition nodes	i
NLOADED_NODES	Number of loaded nodes	i
NMATERIALS	Total number of materials including those declared in composite materials	i
NCOMP_MATERIALS_TYPE1	Number of type 1 composite materials	i
NNURBS_CURVES	Number of NURBS curves	i
TIMESTEP	The value of the timestep	r
NTIMESTEPS	Number of time-steps This keyword and the pair of keywords **NINTERNAL_TIMESTEPS** and **NEXTERNAL_TIMESTEPS** are mutually exclusive	i
NINTERNAL_TIMESTEPS **NEXTERNAL_TIMESTEPS**	Number internal and external time-steps These data enable time-stepping to be broken down into a series of *NExternalTimesteps* sets of *NinternalTimesteps* time-steps (See **NTIMESTEPS**)	i
DAMPING_FACTOR	Viscous damping factor	r
BETA	Newmark's β integration constant	r
GAMMA	Newmark's γ integration constant	r

Table 2.2: (a) The mesh definition file keywords and keyword data (cont.)

Keyword	Keyword Data	Data Type
NSTRESS_POINTS_LINK1	Number of stress points in a LINK1 element	i
NSTRESS_POINTS_LINK2	Number of stress points in a LINK2 element	i
NSTRESS_POINTS_LINK3	Number of stress points in a LINK3 element	i
NSTRESS_POINTS_LINK4	Number of stress points in a LINK4 element	i
NSTRESS_POINTS_LINK5	Number of stress points in a LINK5 element	i
NSTRESS_POINTS_TRIANG1	Number of stress points in a TRIANG1 element	i
NSTRESS_POINTS_TRIANG2	Number of stress points in a TRIANG2 element	i
NSTRESS_POINTS_TRIANG3	Number of stress points in a TRIANG3 element	i
NSTRESS_POINTS_TRIANG4	Number of stress points in a TRIANG4 element	i
NSTRESS_POINTS_TRIANG5	Number of stress points in a TRIANG5 element	i
NSTRESS_POINTS_QUAD1	Number of stress points in a QUAD1 element	i
NSTRESS_POINTS_QUAD3	Number of stress points in a QUAD3 element	i
NSTRESS_POINTS_TETRAH1	Number of stress points in a TETRAH1 element	i
NSTRESS_POINTS_BLOCK1	Number of stress points in a BLOCK1 element	i
MESHPOINT_COORDS	x-,y- and z-coordinates of the mesh-point	i 3*r
NODES_LINK1	Node indices of each **LINK1** element	i 2*i
NODES_LINK2	Node indices of each **LINK2** element	i 2*i
NODES_LINK3	Node indices of each **LINK3** element	i 2*i
NODES_LINK4	Node indices of each **LINK4** element	i 2*i
NODES_LINK5	Node indices of each **LINK5** element	i 2*i
NODES_TRIANG1	Node indices of each **TRIANG1** element	i 3*i
NODES_TRIANG2	Node indices of each **TRIANG2** element	i 6*i
NODES_TRIANG3	Node indices of each **TRIANG3** element	i 3*i
NODES_TRIANG4	Node indices of each **TRIANG4** element	i 3*i
NODES_TRIANG5	Node indices of each **TRIANG5** element	i 3*i
NODES_QUAD1	Node indices of each **QUAD1** element	i 4*i
NODES_QUAD3	Node indices of each **QUAD3** element	i 9*i
NODES_TETRAH1	Node indices of each **TETRAH1** element	i 4*i
NODES_BLOCK1	Node indices of each **BLOCK1** element	i 8*i
BOUNDARY_CONDITIONS	Nodal boundary conditions For each boundary condition node, the node index followed by, for each degree of freedom, the string **"FREE"** if no boundary conditions are imposed, **"FIXED"** followed by a real displacement [m], or **"SPRING"** followed by a non-negative spring-constant [Nm^{-1}]. The displacement argument specifies an initial displacement of a node. The spring constant enables elastic forces to be modelled.	←

Table 2.2: (b) The mesh definition file keywords and keyword data (cont.)

Keyword	**Keyword Data**	**Data Type**
LOADS	Nodal loads For each loaded node, the node index followed by the applied load [N] for each degree of freedom	i r
MATERIALS_LINK1	Material name of each **LINK1** element	i s
MATERIALS_LINK2	Material name of each **LINK2** element	i s
MATERIALS_LINK3	Material name of each **LINK3** element	i s
MATERIALS_LINK4	Material name of each **LINK4** element	i s
MATERIALS_LINK5	Material name of each **LINK5** element	i s
MATERIALS_TRIANG1	Material name of each **TRIANG1** element	i s
COMP_MATERIALS_TRIANG2	Composite material name of each **TRIANG2** element	i s
MATERIALS_TRIANG3	Material name of each **TRIANG3** element	i s
MATERIALS_TRIANG4	Material name of each **TRIANG4** element	i s
MATERIALS_TRIANG5	Material name of each **TRIANG5** element	i s
MATERIALS_QUAD1	Material name of each **QUAD1** element	i s
COMP_MATERIALS_QUAD3	Composite material name of each **QUAD3** element	i s
MATERIALS_TETRAH1	Material name of each **TETRAH1** element	i s
MATERIALS_BLOCK1	Material name of each **BLOCK1** element	i s
REMESH_DATA	Defines the beginning of NURBS definition	

Table 2.2: (c) The mesh definition file keywords and keyword data

2.2 The Material File

The materials file (**.mat**) contains the properties of the material and composite material declared in the mesh definition file. Any number of materials can be defined and not just those used by the current mesh. This enables a number of different meshes to use a single materials file. The materials and composite materials can be defined in any order.

Each definition of material properties begins with the keyword **MATERIAL** and ends with the keyword **END**. Type 1 composite material properties begin with the keyword **COMP_MATERIAL_TYPE1** and must also end with the keyword **END**. Tables 2.3 and 2.4 define the property keywords.

Keyword	Keyword Data	Data Type
MATERIAL	Material name	s
MAT_TYPE	Material model index	i
DENSITY	Density	r
YMOD	Isotropic Young's modulus This keyword and the x-,y- and z-direction Young's moduli keywords are mutually exclusive	r
YMOD_X	x-direction Young's modulus	r
YMOD_Y	y-direction Young's modulus	r
YMOD_Z	z-direction Young's modulus	r
THICKNESS	Thickness	r
POISSONS_RATIO	Poisson ratio	r
IPARAM*n*	n-th integer property, $n = 1, \ldots, 6$	i
DPARAM*n*	n-th double precision property, $n = 1, \ldots, 20$	r
END		

Table 2.3: The material property keywords and keyword data

Keyword	Keyword Data	Data Type
COMP_MATERIAL_TYPE1	Type 1 composite material name	s
NLAYERS	Number of layers	i
LAYER_MATERIALS	Material name of each layer	i s
LAYER_THICKNESSES	Thickness of each layer	i r
END		

Table 2.4: The type 1 composite material property keywords and keyword data

The structure offers a possibility to introduce new material properties which are not listed among the keywords. For integer values **IPARAM*n*** and for real values **DPARAM*n*** should be used.

IMPORTANT When new material property is defined, do not forget to document that which parameters correspond to which material property.

In the case of the composite material the thicknesses can have any value and their sum is not checked.

An example **.mdf** with materials would be the following.

```
# Example with TRIANG 1 elements and materials

TITLE "An example mesh"
NMESHPOINTS        6
NNODES             6
NELEMENTS_TRIANG1  4
NMATERIALS         2

MESHPOINT_COORDINATES
  1    0.000    0.000    0.000
  2    3.000    0.000    0.000
  3    6.000    0.000    0.000
  4    0.000    3.000    0.000
  5    3.000    3.000    0.000
  6    6.000    3.000    0.000

NODES_TRIANG1
  1      1        2        4
  2      2        5        4
  3      2        3        5
  4      3        6        5

MATERIALS_TRIANG1
  1      "steel"
  2      "concrete"
  3      "concrete"
  4      "steel"
```

While the corresponding **.mat** file looks like this.

```
MATERIAL  "steel"
DENSITY   2000.0
YMOD      210000.0
# Yielding stress
DPARAM1   20
END

MATERIAL  "concrete"
DENSITY   1000.0
YMOD      50000.0
# Yielding stress
DPARAM1   10
END
```

2.3 The Domain Decomposition File

The domain decomposition file (**.dom**) specifies the subdomains into which the elements have been partitioned (Table 2.5). The keywords **NSUBDOMAINS** and **SUBDOMAINS_TRIANG1** etc. specify the number of subdomains to be created and the partition indices for the various element types. The indices must lie between 1 and the number of subdomains inclusive and the number of each element type must be consistent with the number of that type defined in the mesh definition file.

Keyword	Keyword Data	Data Type
NSUBDOMAINS	Number of subdomains	i
SUBDOMAINS_LINK1	Subdomain index of each **LINK1** element	i i
SUBDOMAINS_LINK2	Subdomain index of each **LINK2** element	i i
SUBDOMAINS_LINK3	Subdomain index of each **LINK3** element	i i
SUBDOMAINS_LINK4	Subdomain index of each **LINK4** element	i i
SUBDOMAINS_LINK5	Subdomain index of each **LINK5** element	i i
SUBDOMAINS_TRIANG1	Subdomain index of each **TRIANG1** element	i i
SUBDOMAINS_TRIANG2	Subdomain index of each **TRIANG2** element	i i
SUBDOMAINS_TRIANG3	Subdomain index of each **TRIANG3** element	i i
SUBDOMAINS_TRIANG4	Subdomain index of each **TRIANG4** element	i i
SUBDOMAINS_TRIANG5	Subdomain index of each **TRIANG5** element	i i
SUBDOMAINS_QUAD1	Subdomain index of each **QUAD1** element	i i
SUBDOMAINS_QUAD3	Subdomain index of each **QUAD3** element	i i
SUBDOMAINS_TETRAH1	Subdomain index of each **TETRAH1** element	i i
SUBDOMAINS_BLOCK1	Subdomain index of each **BLOCK1** element	i i

Table 2.5: The domain decomposition file keywords and keyword data

An example domain decomposition file is shown for Figure 2.2

```
# An example domain decomposition file
NSUBDOMAINS  2

SUBDOMAINS_TRIANG1
  1   1
  2   1
  3   2
  4   2
```

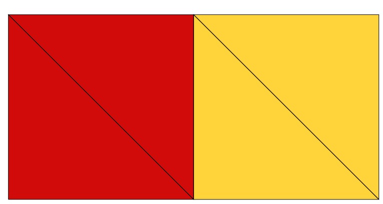


Figure 2.4: An example domain decomposition with TRIANG1 elements

2.4 The Mesh Parameters File

The mesh parameters file (**.mpr**) defines the element or nodal mesh parameters of each element or node. The number of element mesh parameters for each element type must equal the number of elements defined in the mesh definition file. The keywords for the different element types is shown in Table 2.6.

When the mesh parameters are defined on a nodal basis the **NODAL_MESHPARAMS** keyword should be specified and after that the mesh parameter for each mesh-point should also be defined.

Keyword	Keyword Data	Data Type
NODAL_MESHPARAMS	Mesh parameters for each node	i r
MESHPARAMS_LINK1	Element mesh parameter of each **LINK1** element	i r
MESHPARAMS_LINK2	Element mesh parameter of each **LINK2** element	i r
MESHPARAMS_LINK3	Element mesh parameter of each **LINK3** element	i r
MESHPARAMS_LINK4	Element mesh parameter of each **LINK4** element	i r
MESHPARAMS_LINK5	Element mesh parameter of each **LINK5** element	i r
MESHPARAMS_TRIANG1	Element mesh parameter of each **TRIANG1** element	i r
MESHPARAMS_TRIANG2	Element mesh parameter of each **TRIANG2** element	i r
MESHPARAMS_TRIANG3	Element mesh parameter of each **TRIANG3** element	i r
MESHPARAMS_TRIANG4	Element mesh parameter of each **TRIANG4** element	i r
MESHPARAMS_TRIANG5	Element mesh parameter of each **TRIANG5** element	i r
MESHPARAMS_QUAD1	Element mesh parameter of each **QUAD1** element	i r
MESHPARAMS_QUAD3	Element mesh parameter of each **QUAD3** element	i r
MESHPARAMS_TETRAH1	Element mesh parameter of each **TETRAH1** element	i r
MESHPARAMS_BLOCK1	Element mesh parameter of each **BLOCK1** element	i r

Table 2.6: The element mesh parameters file keywords and keyword data

An example element mesh parameters file is shown below and in Figure 2.5, which corresponds to the mesh in Figure 2.2.

```
# An example element mesh parameters file

MESHPARAMS_TRIANG1
  1   0.1
  2   0.2
  3   0.2
  4   0.3
```

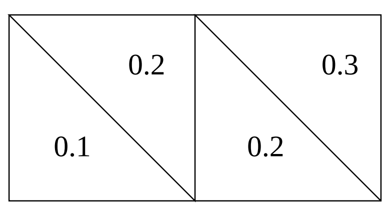


Figure 2.5: An example element mesh parameter definition

An example nodal mesh parameters file is shown below and in in Figure 2.6, which corresponds to the mesh in Figure 2.2. It should be noted that Figure 2.6 is **NOT** equivalent to Figure 2.5. However it is possible to convert nodal mesh parameters to element mesh parameters and vica versa by the tools `MPR2EMP` and `EMP2MPR`, see Sections 7.10 and 7.3.

```
# An example nodal mesh parameters file

NODAL_MESHPARAMS
  1   0.1
  2   0.2
  3   0.3
  4   0.1
  5   0.2
  6   0.3
```

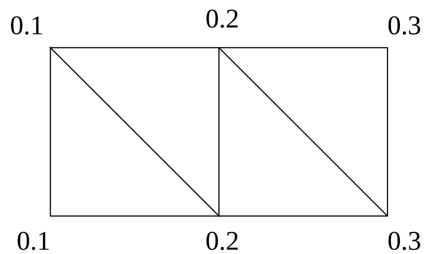


Figure 2.6: An example nodal mesh parameter definition

2.5 The Finite Element Error File

The finite element errors are stored in the finite element error file (**.fee**). One error value must be defined for each element (Table 2.7). Even in the case of **TRIANG2** and **QUAD3** elements which are layered finite elements only one value should be defined for one element.

Keyword	Keyword Data	Data Type
FEERRORS_LINK1	Finite element error of each **LINK1** element	i r
FEERRORS_LINK2	Finite element error of each **LINK2** element	i r
FEERRORS_LINK3	Finite element error of each **LINK3** element	i r
FEERRORS_LINK4	Finite element error of each **LINK4** element	i r
FEERRORS_LINK5	Finite element error of each **LINK5** element	i r
FEERRORS_TRIANG1	Finite element error of each **TRIANG1** element	i r
FEERRORS_TRIANG2	Finite element error of each **TRIANG2** element	i r
FEERRORS_TRIANG3	Finite element error of each **TRIANG3** element	i r
FEERRORS_TRIANG4	Finite element error of each **TRIANG4** element	i r
FEERRORS_TRIANG5	Finite element error of each **TRIANG5** element	i r
FEERRORS_QUAD1	Finite element error of each **QUAD1** element	i r
FEERRORS_QUAD3	Finite element error of each **QUAD3** element	i r
FEERRORS_TETRAH1	Finite element error of each **TETRAH1** element	i r
FEERRORS_BLOCK1	Finite element error of each **BLOCK1** element	i r

Table 2.7: The finite element error file keywords and keyword data

An example finite element error file is shown for Figure 2.2

```
# An example finite element error file

FEERRORS_TRIANG1
  1   0.0021
  2   0.0032
  3   0.0123
  4   0.0001
```

2.6 The Finite Element Stress File

The element stresses are defined in the stress file (**.ste**). The number of stress points for each element must be defined in the **.mdf** file with the corresponding keyword, like **NSTRESS_POINTS_LINK1**, etc. For each stress point a line is defined consisting of 6 components, e.g. normal stresses in the x, y and z directions, the three shear stresses. Other interpretation is possible as well, e.g. the first three components store the principal stresses while the other three components store the angles of the principal directions. Any of the components can be ignored, e.g. in a plane problem only the first three components will be used, as the two normal stresses and a shear stress, or in the case of truss elements only one component is used.

`IMPORTANT` In the case of **TRIANG2** and **QUAD3** elements the stresses are defined at the stress points per layer per element, therefore it is a requirement that the material file is available for these elements. For all other elements the existence of the material file is not necessary to load/write stresses.

Table 2.8 shows the possible keywords.

Keyword	Keyword Data	Data Type
STRESSES_LINK1	6 available components of stresses	i i 6*r
STRESSES_LINK2	6 available components of stresses	i i 6*r
STRESSES_LINK3	6 available components of stresses	i i 6*r
STRESSES_LINK4	6 available components of stresses	i i 6*r
STRESSES_LINK5	6 available components of stresses	i i 6*r
STRESSES_TRIANG1	6 available components of stresses	i i 6*r
STRESSES_TRIANG2	6 available components of stresses	i i i 6*r
STRESSES_TRIANG3	6 available components of stresses	i i 6*r
STRESSES_TRIANG4	6 available components of stresses	i i 6*r
STRESSES_TRIANG5	6 available components of stresses	i i 6*r
STRESSES_QUAD1	6 available components of stresses	i i 6*r
STRESSES_QUAD3	6 available components of stresses	i i i 6*r
STRESSES_TETRAH1	6 available components of stresses	i i 6*r
STRESSES_BLOCK1	6 available components of stresses	i i 6*r

Table 2.8: The stress file keywords and keyword data

An example finite element stress file is shown for Figure 2.2

```
# An example stress file

# Triang1 stresses
# element index
#   Stress point index
#           Szigma x    Szigma y    Szigma z    Tau xy      Tau yz      Tau zx
#_________________________________________________________________________________

STRESSES_TRIANG1
    1  1    1.000e+00   2.000e+00   3.000e+00   4.000e+00   5.000e+00   6.000e+00
    1  2    7.000e+00   8.000e+00   9.000e+00  10.000e+00  11.000e+00  12.000e+00
    2  1   13.000e+00  14.000e+00  15.000e+00  16.000e+00  17.000e+00  18.000e+00
    2  2   19.000e+00  20.000e+00  21.000e+00  22.000e+00  23.000e+00  24.000e+00
    3  1   25.000e+00  26.000e+00  27.000e+00  28.000e+00  29.000e+00  30.000e+00
    3  2   31.000e+00  32.000e+00  33.000e+00  34.000e+00  35.000e+00  36.000e+00
    4  1   37.000e+00  38.000e+00  39.000e+00  40.000e+00  41.000e+00  42.000e+00
    4  2   43.000e+00  44.000e+00  45.000e+00  46.000e+00  47.000e+00  48.000e+00
```

2.7 The Element Geometric Definition File

The element geometric definition file **.gmf** defines the boundary of a finite element problem using Non-Uniform Rational B-Splines (NURBS).

There are two most common nonlinear mathematical forms in geometric modeling for curve and surface representation, one is implicit and another is parametric polynomial forms. The implicit form has the advantage that circles, conics, and primitive quadric surfaces, such as cylinders, spheres and cones can be concisely and precisely represented. A disadvantage of the implicit form is that free-form curves and surfaces, which also important in geometric modeling can not be represented. With parametric polynomials, such as polynomial B-splines, one can represent and manipulate free-form curves and surfaces; but unfortunately, circles, conics and the quadric primitives cannot be represented precisely. Non-uniform Rational B-spline (NURBS) is a geometric modeller that offers the advantages of both forms.

Despite the versatility of NURBS in our implementation only "cubic" NURBS are implemented. Cubic NURBS are defined by two end points and two control points. (All other manipulation of NURBS, such as degree elevation, Bezier Curves conversion, knot-removal and local smoothing or modification is not present in the current implementation.)

The element geometric definition file (**.gmf**) file contains the following keywords:

- NENDPOINTS: Number of end points used for the geometric modelling.
- NNURBS_CURVES: Number of NURBS curves.
- ENDPOINT_COORDINATES: A list of coordinates of end points. These nodes will be referred by their node number in the curve specifications. (Good practice to use the same mesh points here as they are in the mesh. In this case the compatibility between the mesh and the geometric definition can always be ensured.)
- NURBS_CURVE: Defines each NURBS curve.
- DEGREE: The number of freedom for the NURBS curve. At the moment it must always be equal to three!
- CONTROL_POINTS: The four control points for Cubic NURBS curve.
- WEIGHTS: The value of weights for the four control points.

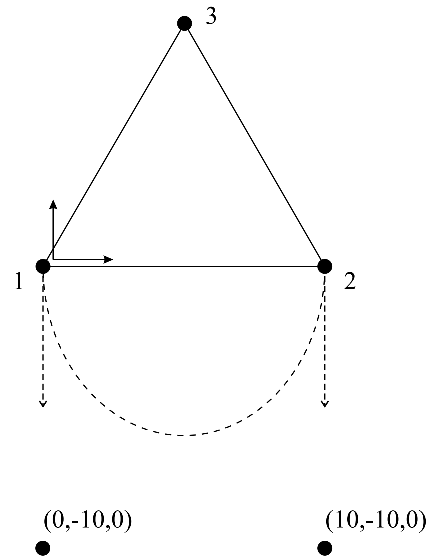
The geometric definition file should only be used together with the remeshing data extensions of the mesh definition file. These extensions are not covered in the main definition of the MDF syntax and can be best explained by the following example.

Figure 2.7a shows one coarse triangle defined by nodes 1,2 and 3. On the side between nodes 1 and 2 a NURBS curve `C1` is defined. The curve is cubic (degree 3) and stretches from node 1 where it has paramete 0.0 to node 2 where it has parameter 1.0. Two control points determine the shape of the curve. They have coordinates (0,-10,0) and (10,-10,0). The weights for these control points are set to 0.5. Figure 2.7b shows how the remeshed triangle may look like when three nodal meshparameters were defined with value δ=0.8.

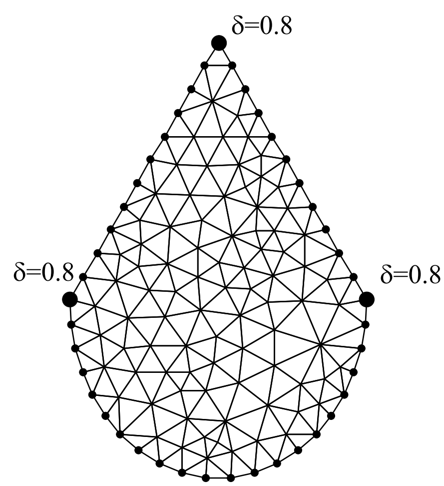
The following three files show how to create the curve definition in the geometric definition file and how to assign it in the mesh definition file.

```
# coarse.mdf : mesh definition file

TITLE "coarse mesh"
NMESHPOINTS     3
NNODES     3
NELEMENTS_TRIANG1   1
NNURBS_CURVES       1

MESHPOINT_COORDINATES
1   0.0     0.0     0.0
2   10.0    0.0     0.0
3   5.0     8.67    0.0

NODES_TRIANG1
1   1     2     3

REMESH_DATA
1   "EXTERIOR"
    NNURBS_CURVES   1
    1 NURBS_CURVE   "C1"
      PARAMETER     0.0
2   "EXTERIOR"
    NNURBS_CURVES   1
    1 NURBS_CURVE   "C1"
    PARAMETER       1.0
3   "EXTERIOR"
```

```
# coarse.gmf : geometric definition file

NENDPOINTS  3

ENDPOINT_COORDINATES
1   0.0     0.0     0.0
2   10.0    0.0     0.0
3   5.0     8.67    0.0

NURBS_CURVE  "C1"
DEGREE  3
CONTROL_POINTS
  1   1
  2   0.0  -10.0  0.0
  3   10.0 -10.0  0.0
  4   2
WEIGHTS
  1   1
  2   0.5
  3   0.5
  4   1
END
```



(a)



(b)

Figure 2.7: An example for geometric definition

Chapter 3

Structured mesh generation: CUBIC

3.1 Status

name of executable(s)	`cubic` | `cubic.exe`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`*.mdf`
own filetypes	`*.gen`

3.2 Syntax

```
Usage: cubic filename [options]
Options:
        -h      Print help, this screen
```

3.3 File format

This chapter describes the file format for the `CUBIC` structured mesh generator program. The extension of the input file must be **.gen**. This file also follows the E_Library conventions with regards to keywords, comments and empty lines.

3.4 Entity definitions

Figure 3.1 shows the conventions for the definition of the different entity types. The origin of the local, element coordinate system is the first node of the geometric entity. The 2D surfaces are defined in an anti-clockwise manner which ensures that the final finite elements will also be defined in a counter-clockwise manner. The first local coordinate direction (ξ) is parallel with the edge defined by the first and the second nodes of the geometric entity. In the case of 2D geometric entities the second local coordinate direction (η) is parallel with the edge determined by the first and the last node of the entity. The first and second nodes determine the first side, the second and third node specifies the second side and so on. The nodes are created and numbered in a row as it is also shown in Figure 3.1 for all 1D and 2D entity types.

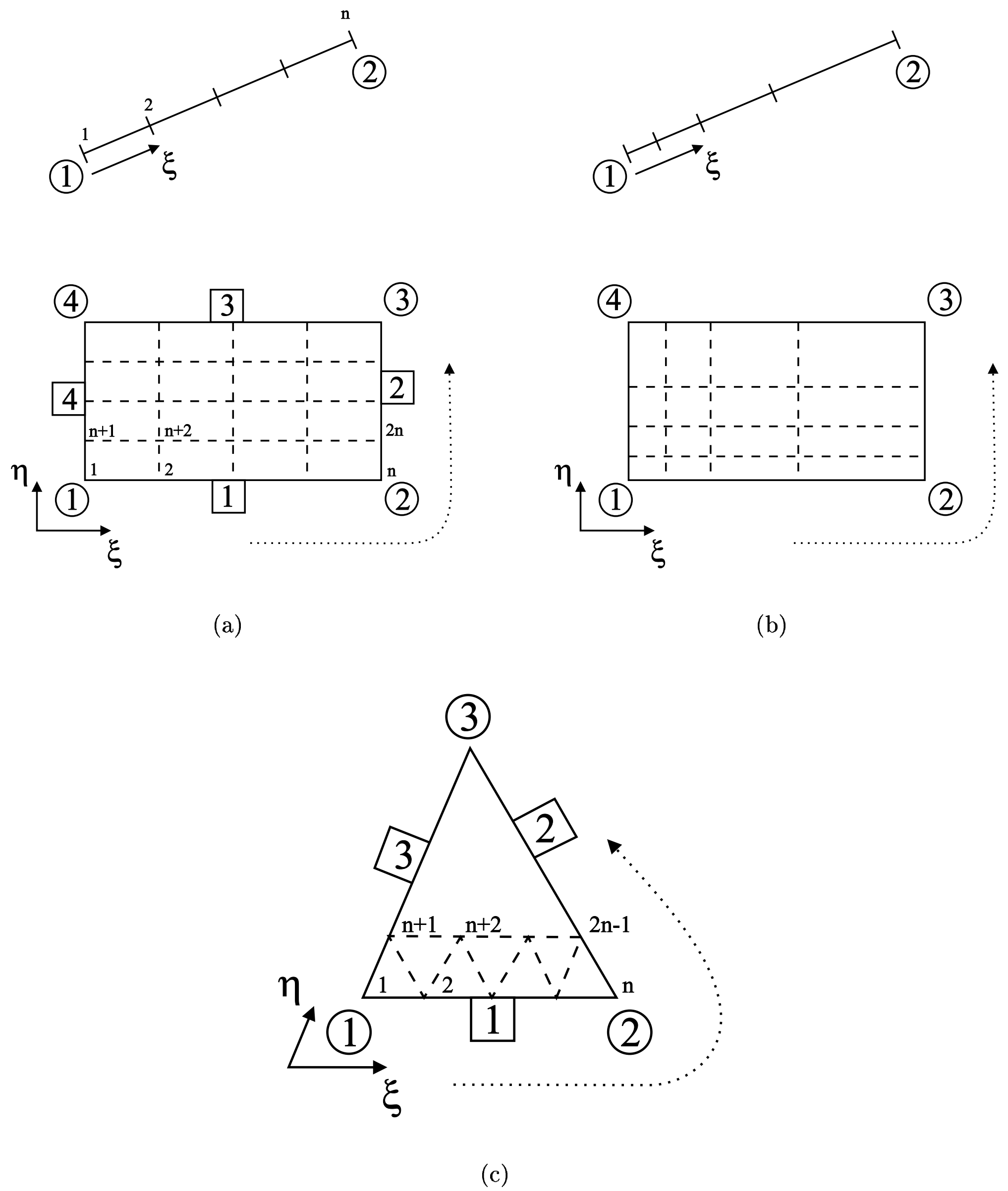


Figure 3.1: Conventions for the entity definition, (a) uniform quadrilateral, (b) nonuniform quadrilateral and (c) triangular entities

3.5 Data file format

The input to the program is a textual data file which describes the geometric model of the structure. The data structure has been designed in such a way that the sequence of mesh definition can be carried out easily at one step. The file starts with a header section, then lists the point entities, the two and the one dimensional entities and finally the boundary conditions. Comment and empty lines can be inserted at any line in the file, because these lines are automatically ignored during the reading of the input file. The comment lines start with a "#" sign. Most of the data is provided in a keyword-data format. The information should be placed into one line, new lines cannot be started in the middle of the data definition.

```
TITLE "this is the descriptive title"
NNODES        i
NSURFACES     j
NLINKS        k
( GEN_MATERIAL  )
( NMATERIALS  l )
( NBC_NODES   m )
( NBC_SIDES   n )
```

Table 3.1: Header section of the geometric data file

3.5.1 Header section

The header section of the input file is defined in Table 3.1 where the `TITLE` keyword defines the title of the project, the `NNODES` keyword defines the number of point entities, the `NSURFACES` keyword defines the number of 2D entities and the `NLINKS` keyword defines the number of 1D entities. Later in the data file the above given number of entities should be specified. The entities are not defined in the order of their dimension, like first points, one dimensional and finally two dimensional elements, because the two dimensional entities are thought to be the main components of the structure, while the one dimensional entities are considered as added components. In spite of this, it is possible to generate a structure containing only one dimensional elements.

The bracketed keywords are optional. The `GEN_MATERIAL` keyword simply specifies that for the final mesh material information should be generated. The next `NMATERIALS` keyword provides the maximum number of user definable material types. If only the first keyword is specified, then all finite elements will have the default material definition, which is assigned to them according to their type. In this case all of the same element types will have the same material. If the second keyword is defined, then extra material definition is allowed by the user. This means that it is possible to assign different material to all finite elements generated from the same geometric entity. The geodesic strings (link type 5 according to the E-lib specification [4]) are exceptions, because their material cannot be overdefined, they will always have the default material definition. The last two keywords `NBC_NODES` and `NBC_SIDES` define the number of boundary condition points and the number of boundary condition sides.

3.5.2 Point entity definition

In this section those points are defined by their three coordinate values, which are used in the later sections to define higher dimension entities. The number of points given in this section must be the same as `NNODES` defined in the header section, but it is not necessary to use all the defined points. Those points, which are not referenced by their index in the later sections are ignored and they will not appear in the final mesh.

The format of this section is shown in Table 3.2, where `x1-coord`, `y1-coord` and `z1-coord` provide the position of the first point in the global coordinate system. The definition of all points are similar.

```
NODE_COORDINATES
1  x1-coord  y1-coord  z1-coord
2  x2-coord  y2-coord  z2-coord
.
.
.
i  xn-coord  yn-coord  zn-coord
```

Table 3.2: Point entity definition in the geometric data file

3.5.3 Surface entity definition

In this section the two dimensional entities, the quadrilateral and the triangular surfaces are defined. They can be defined in any order, but their numbering should be consecutive starting from one and the total number of these entities should be equal to NSURFACES specified in the header section. At the moment the triangular surfaces can be defined only with straight edges, while the quadrilateral surfaces are defined as cubic patches. A special case of the cubic patch definition is when the quadrilateral surface has straight edges. To help the insertion of one dimensional elements into the final mesh (like geodesic strings) those one dimensional elements which are part of the surface are also defined in this section. The format of the surface entity definition is presented in Table 3.3.

```
SURFACE n
  SHAPE         TRIANG | QUAD | 3 | 4
  ELEMENT_TYPE  TRIANG1 | TRIANG3 | TRIANG4 | TRIANG5 |
                [ QUAD1 ]
  DEFINITION    [ LINEAR | PARAMETRIC | POINT ]
  nodes | [ nodes consts ]
  N_DIVISIONS n [ n ]
  ( DIVISION_WEIGHTS
      weights
      weights
      < weights > )
  ( LINK_DIRECTION   1 | 0   1 | 0   < 1 | 0 >
    LINK_DIVISION
      division
      division
      < division >
    LINK_TYPE
      types
      types
      < types > )
  ( MATERIAL n )
  [ DIAGONAL 1 | 2 | SHORTEST | LONGEST ]
END
```

Table 3.3: Surface entity definition in the geometric data file

The bracketed sections are optional, the square brackets ('[',']') surround options which can be defined only for quadrilateral geometric entities and the ('<', '>') brackets contains expressions only for the triangular geometric entities. The vertical line ('|') is written between options, from which at least one should be defined. n represents an integer value. The SHAPE keyword specifies the form of the surface which can be quadrilateral or triangular in shape. A triangular geometric entity is simply defined by its corner nodes after the DEFINITION keyword. On the other hand the quadrilateral geometric entity can be defined in three different ways. In the case of the LINEAR definition the four corner nodes should be supplied, in the case of the POINT 16 nodes should be specified to define a cubic patch and in the case of the PARAMETRIC four corner nodes and the derivatives are required to define the cubic patch. The order of the derivatives for the parametric definition is:

$$X_\xi \; Y_\xi \; Z_\xi \quad X_\eta \; Y_\eta \; Z_\eta \quad X_{\xi\eta} \; Y_{\xi\eta} \; Z_{\xi\eta} \tag{3.1}$$

A node can be a reference to a previously defined point entity or to a generated node of one of the entities. Allowing the later case of point referencing the definition of the surface topology can be simplified. Two examples are shown in Figure 3.2 where the surfaces do not have to be cut up into further pieces to ensure the compatibility between the adjacent surfaces. The format of the *reference point* is either s n x y if it references a node in a two dimensional entity or l n x if it references a node in a one dimensional entity, where n is the index of the entity and x and y define the position of the node in the local coordinate system of the referenced geometric entity (see Figure 3.1). An entity can reference only such an entity in which the mesh generation has been finished. For example, `s 1 1 3` references a node in the first surface entity which is in the first column and in the third row of the to be generated points. The example is shown in a quadrilateral and in a triangular surface in Figure 3.3.

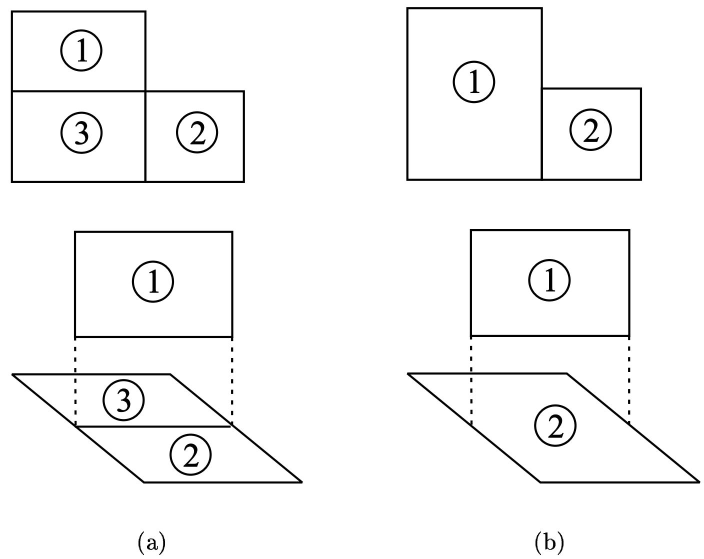


Figure 3.2: Surface definition (a) without and (b) with "referencing points"

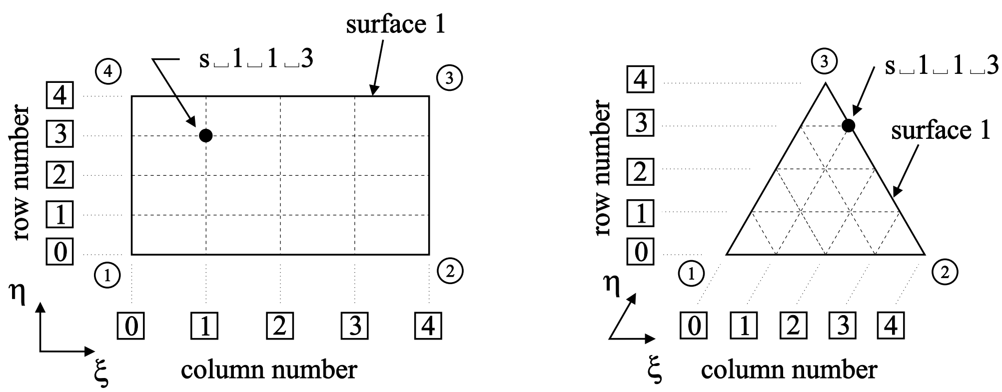


Figure 3.3: Referenced node in the first surface entity

The `ELEMENT_TYPE` keyword specifies the type of the finite element which will be generated in the geometric entity. Quadrilateral elements cannot be generated in a triangular geometric entity. The name of the element types are references to the available element types in the E-lib specification [4] because the generated final finite element mesh will comply the E-lib specification. The `DIVISION` keyword specifies the number of divisions on the sides of the entities. In the case of a triangular entity all sides are divided into equal number of divisions, but if required the divisions can have different lengths on all sides. In the case of the quadrilateral entities the first integer number specifies the division of the first and the third sides, while the second number specifies the division on the second and the fourth sides. If no more data is specified before the `END` keyword, then the geometric entities are divided uniformly and no one dimensional element will be inserted into the surface or to the edge of the surface.

The lines after the optional `DIVISION_WEIGHT` keyword specify the division weights on the sides. The `weights` can be any real numbers except zero. If the weight is zero, the program will use one as a weight. The specified `weights` are normalised. The `weights` are either specified as a series of double numbers or using the syntax [3] of "`n ( d )`" where the `d` real number is considered `n` times after each other. Any combination of the two syntaxes is also acceptable. The `LINK_DIRECTION` keyword specifies whether there are one dimensional elements to insert into the surface. The position of the one ('true') values specify the side which the one dimensional elements will be parallel with (see Figure 3.4). The `LINK_DIVISION` and `LINK_TYPE` keywords specify the position and the type of the one dimensional elements. The available types comply with the E-lib specification [4] which allows five types of one dimensional element at the moment. The positions of the one dimensional elements are given as a series of integer values.

The first integer value in the list specifies the distance from the corresponding side. The distance is expressed in terms of the generated rows of elements. The other values in the list are increments from the previous position. An example is shown in Table 3.4. Finally the material can be defined by the `MATERIAL` keyword. The number `n` specifies that which user defined material should be assigned to all of the generated surface elements in the surface entity. The number cannot be larger than `NMATERIALS` defined in the header section. The final keyword `DIAGONAL` can be used to control the division of quadrilateral elements into triangles in a quadrilateral geometric entity. The default value is `SHORTEST` which defines that the shortest diagonal of the two diagonals will be used in the geometric entity. The other options allow to use the longer, first or second diagonal.

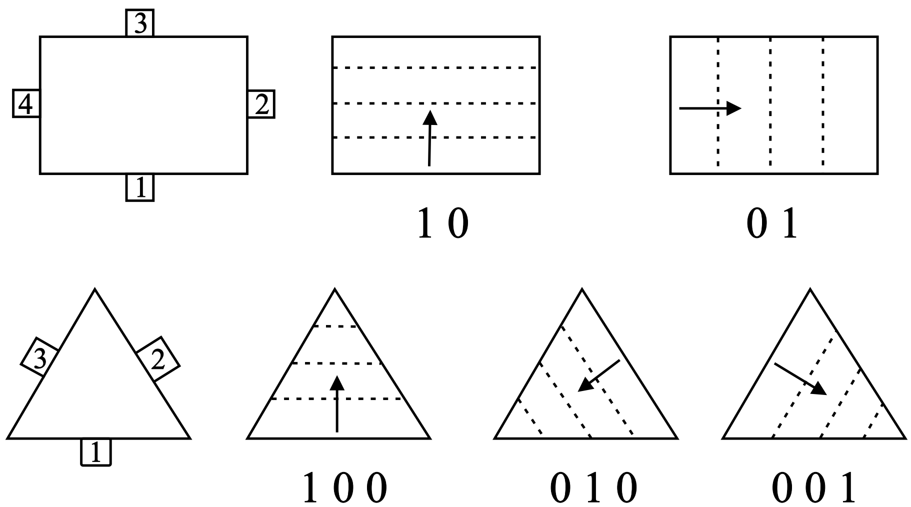


Figure 3.4: `LINK_DIRECTION` values and the corresponding direction of the one dimensional elements, arrows also show the order of the definition of the elements according to `LINK_DIVISION`

3.5.4 Link entity definition

In this section the one dimensional entities are defined. They can be defined in any order, but their numbering should be consecutive starting from one and the total number of these entities should be equal to `NLINKS` specified in the header section. At the moment these elements can be defined only as straight lines. The format of the link entity definition is presented in Table 3.5. Comments about the `nodes`, elementy type, division of the element and user definable material can also be applied for these types of element. The difference is that for the one dimensional elements it is not necessary to define the `DIVISION`.

3.5.5 Boundary condition definition section

The final section contains the boundary condition definitions. Geometrically two types of boundary condition can be defined: point or line (or area for quadrilateral surfaces). In the final mesh format only point boundary conditions exist, according to the E-lib specification [4], therefore all boundary conditions are translated into a point boundary condition. The format of the boundary condition

```
LINK_DIRECTION 1 1
LINK_DIVISION
2 2
0 2 1
LINK_TYPE
3 3
3 5 5
```

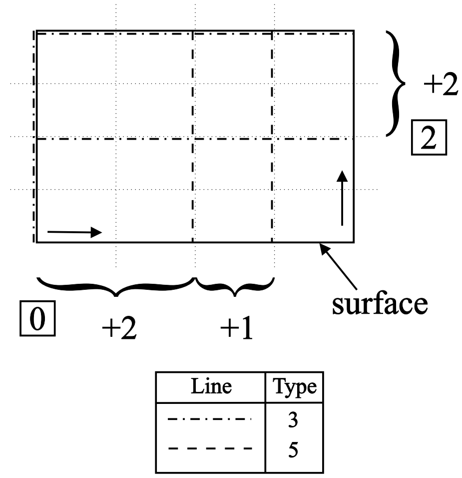


Table 3.4: Example for the definition of one dimensional elements in a surface entity

```
LINK n
  ELEMENT_TYPE  LINK1 | LINK2 | LINK3 | LINK4 | LINK5
  DEFINITION nodes
  ( N_DIVISIONS n )
  ( DIVISION_WEIGHTS
    weights )
  ( MATERIAL n )
END
```

Table 3.5: Link entity definition in the geometric data file

definition section is presented in Table 3.6. First the point boundary conditions are listed. The point boundary conditions can be defined only for the point entities and they can be listed in any order. The only requirement is that the number of the point boundary conditions should be equal to NBC_NODES specified in the header section. The bc_type can be “"FREE"”, “"FIXED" d” or “"SPRING" d” where d is a real number and expresses the support displacement or the spring constant respectively.

```
BOUNDARY_CONDITIONS
point_entity_index_1   bc_type  bc_type  bc_type
point_entity_index_2   bc_type  bc_type  bc_type
.
.
.
point_entity_index_m   bc_type  bc_type  bc_type

ref_pnt_1  ref_pnt_2   bc_type  bc_type  bc_type
ref_pnt_3  ref_pnt_4   bc_type  bc_type  bc_type
.
.
.
ref_pnt_5  ref_pnt_6   bc_type  bc_type  bc_type
```

Table 3.6: Boundary condition definition in the geometric data file

The boundary condition sides are defined by two “reference points” in an entity and they must lie on a straight line. The “reference point” definition is explained in section 3.5.3. A special case is when the entity is a quadrilateral surface and the two reference points mark an area as shown in

Figure 3.5. In this case the same boundary condition will be generated for all points inside the area, including the side and corner points.

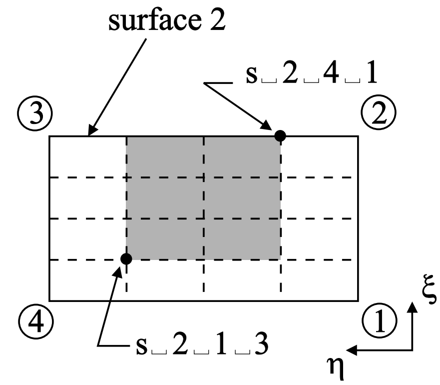


Figure 3.5: Boundary condition generation defined by two reference points for an area, the same boundary conditions will be generated for the points inside and on the edge of the shaded area

First the boundary condition “sides” are generated in the order of their definition. If there are overlapping areas then the boundary conditions of the points in the overlapping area will be redefined and only the later definition will be stored. Finally the boundary condition points are defined and they may redefine the boundary condition for a point again. At every occasion of a redefinition of a boundary condition the program issues a warning message. The order of the generation of the boundary conditions and the possibility of the redefinition of previously defined boundary conditions allows, that at corners of areas or at intersection of boundary condition lines such a boundary condition can be defined which may be a combination of the different crossing boundary conditions.

3.6 Examples for parametric definition

An example is shown in Table 3.7. On the left-hand side the geometric file definition and on the right-hand side the generated mesh is shown. The values of the `PARAMETRIC` definition are determined according to Figure 3.6. In Figure 3.6a the real domain (Ω) is shown. The straight lines determine the quadrilateral surface and the dashed lines are the required shape of the surface. In Figure 3.6b the unit square domain ($\overline{\Omega}$) is presented. The orientation of the real domain and the unit square domain is the same, in this case it is counter-clockwise. To determine the derivative values for the geometric file which are the components of the tangential vectors in the ξ and η directions in the global coordinate system someone has to inspect the orientation of the ξ,η coordinate system at the particular node.

For example at node 1 in the $\overline{\Omega}$ domain the ξ coordinate direction points from node 1 to node 2, which in the real domain Ω is a vertical vector. This is represented in the file as `0 3 0`. The η direction in the $\overline{\Omega}$ domain points from node 1 to node 4 which in the real domain Ω can be determined in a similar way (continuous η vector at node 1 in Figure 3.6). On the other hand in the real domain the side between node 1 and 4 is not a straight line and the η derivative (tangential vector) at node 1 is actually a horizontal vector as shown in Figure 3.6a by a dashed vector. The representation of this vector in the geometric file is `-4.71 0 0`. The length of the vector should be equal to the real side length to ensure equidistant distribution of the points. In this case the quarter length of the perimeter of a circle with a 3 unit radius is ($2r\pi/4 \approx 4.71$). The last three values for node 1 in the geometric file are zero since there is no “twist” defined. The effect of the “twist” is demonstrated in Table 3.14

Another example where the same square surface is defined by the `LINEAR` and the `PARAMETRIC` method is shown in Table 3.8. Some other possible variations of the square are shown in Table 3.9-3.14. In the tables all examples are in 2D except the last one, which is a 3D surface.

```
NODE_COORDINATES
1 6.0 3.0 0.0
2 6.0 6.0 0.0
3 0.0 0.0 0.0
4 3.0 0.0 0.0

SURFACE 1
  SHAPE   QUAD
  ELEMENT_TYPE  TRIANG3

  DEFINITION PARAMETRIC
  1  0 3 0  -4.71  0     0  0 0 0
  2  0 3 0  -9.42  0     0  0 0 0
  3 -3 0 0   0     -9.42 0  0 0 0
  4 -3 0 0   0     -4.71 0  0 0 0
  N_DIVISIONS 10 10

END
```

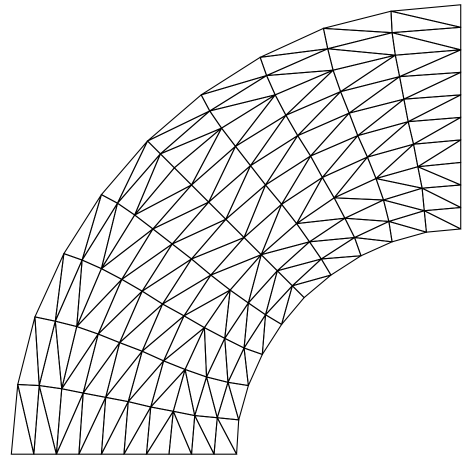
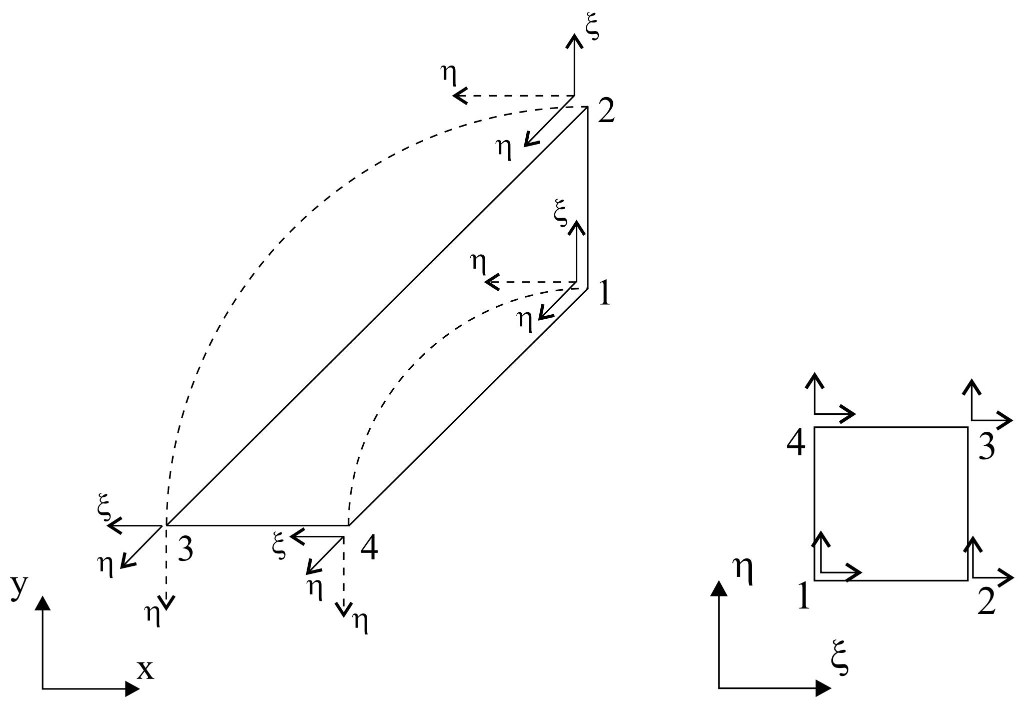


Table 3.7: A complex example to show the geometric file definition and the generated mesh

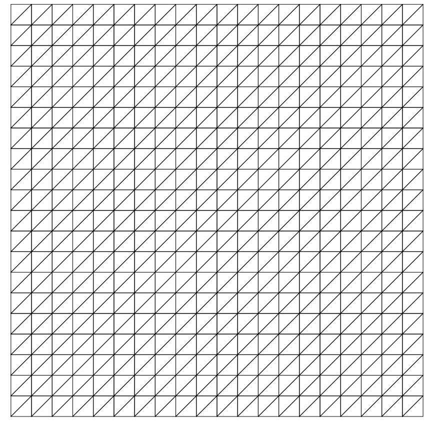


(a)

(b)

Figure 3.6: The real domain Ω (a) and the unit square domain $\overline{\Omega}$ (b) for the complex example

```
DEFINITION LINEAR
1 2 3 4
```



```
DEFINITION PARAMETRIC
1 1 0 0  0 1 0  0 0 0
2 1 0 0  0 1 0  0 0 0
3 1 0 0  0 1 0  0 0 0
4 1 0 0  0 1 0  0 0 0
```

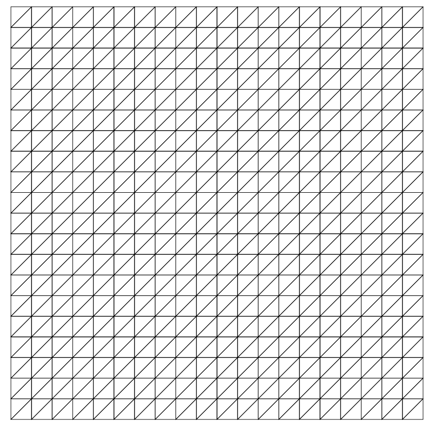


Table 3.8: LINEAR and PARAMETRIC definition of a square

```
DEFINITION PARAMETRIC
1 1 1 0  0 1 0  0 0 0
2 1 0 0  0 1 0  0 0 0
3 1 0 0  0 1 0  0 0 0
4 1 0 0  0 1 0  0 0 0
```

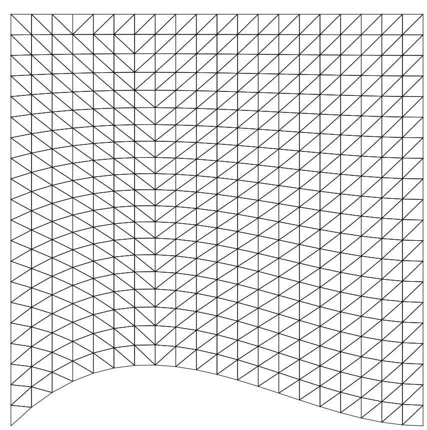


Table 3.9: Example for the definition of cubic surfaces

```
DEFINITION PARAMETRIC
1 1 0 0  1 1 0  0 0 0
2 1 0 0  0 1 0  0 0 0
3 1 0 0  0 1 0  0 0 0
4 1 0 0  0 1 0  0 0 0
```

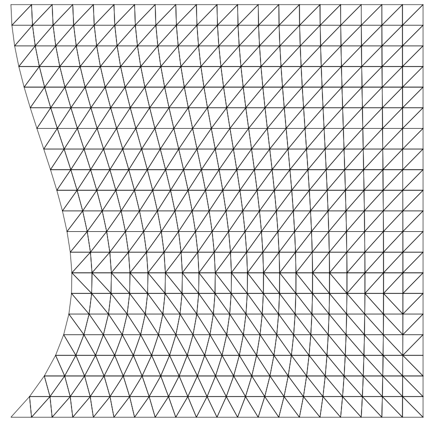


Table 3.10: Example for the definition of cubic surfaces

```
DEFINITION PARAMETRIC
1 1 1 0  1 1 0  0 0 0
2 1 0 0  0 1 0  0 0 0
3 1 0 0  0 1 0  0 0 0
4 1 0 0  0 1 0  0 0 0
```

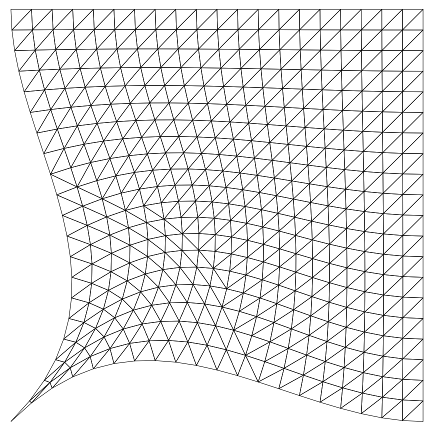


Table 3.11: Example for the definition of cubic surfaces

```
DEFINITION PARAMETRIC
1 1 0 0  0 1 0  10 0 0
2 1 0 0  0 1 0  0 0 0
3 1 0 0  0 1 0  0 0 0
4 1 0 0  0 1 0  0 0 0
```

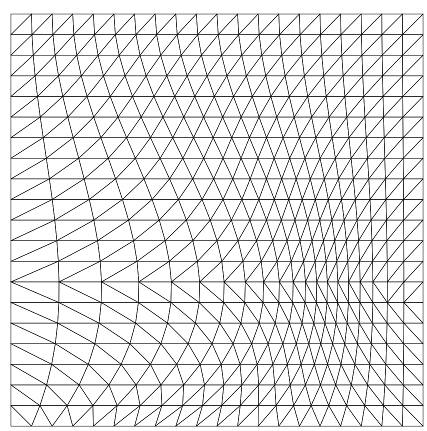


Table 3.12: Example for the definition of cubic surfaces

```
DEFINITION PARAMETRIC
1 1 0 0  0 1 0  10 10 0
2 1 0 0  0 1 0  0 0 0
3 1 0 0  0 1 0  0 0 0
4 1 0 0  0 1 0  0 0 0
```

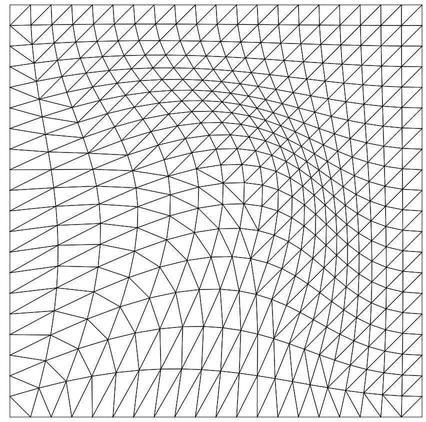


Table 3.13: Example for the definition of cubic surfaces

```
DEFINITION PARAMETRIC
1 1 0 -1 0 1 0  0 0 0
2 1 0 1  0 1 0  0 0 0
3 1 0 1  0 1 0  0 0 0
4 1 0 1  0 1 0  0 0 0
```

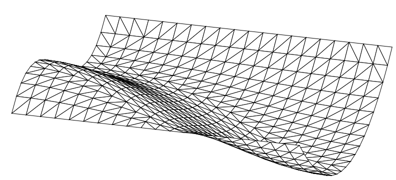


Table 3.14: Example for the definition of cubic surfaces

Chapter 4

2D Finite element analysis: FEM

4.1 Status

name of executable(s)	`fem` | `fem.exe`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.1`
date	`July 2003`
release	`unfinished`
E_lib filetypes	`*.mdf *.mat *.ste`
own filetypes	`-`

4.2 Syntax

```
fem v1.1 SECT Research Group, Heriot-Watt University Edinburgh
usage: fem input output ds
note all filenames are referred to without extensions
   input    : name of the mesh definition and material file
   output   : name of the output mesh definition file
   ds     : displacement scale (e.g. 10)

you might want to renumber the mesh first!
and remove unused nodes!
```

4.3 Overview

`fem` is a very limited 2D finite element program which calculates stresses and displacements for meshes consisting of `TRIANG1` and `QUAD1` elements only. Loads, boundary conditions and material assignments should be specified in the input mesh file. Material information should be given in the material file with the same name as the mesh definition file.

As `TRIANG1` and `QUAD1` are constant stress elements the output stress file will consist of one stress-point per element. As `fem` is a 2D FE package only three stresses will appear in the output stress file: σ_x, σ_y and τ_{xy}.

In order to increase the efficiency of the computation and to reduce the calculation times you should not include any unused nodes in the coordinates array. Renumbering of the nodes to decrease the bandwidth of the stiffness matrix is also advised. The `renumber` program can be used for this purpose. `renumber` is explained in Section 7.11.

4.4 Program limitations

- Only meshes consisting entirely of one of the two supported element types can be analysed.
- From the material file only `YMOD`, `POISSONS_RATIO` and `THICKNESS` will be taken into account.
- The program gives no status reports and might take ages if you have not renumbered your mesh.
- The program will crash if the mesh contains unconnected nodes. These are nodes which are not used in any of the element definitions. Use the `cleanup` program to remove unconnected nodes. `cleanup` is explained in Section 7.2

Chapter 5

Error analysis: ADAPT

5.1 Status

name of executable(s)	`adapt` | `adapt.exe`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.1`
date	`July 2003`
release	`development`
E_lib filetypes	`*.mdf *.mat *.ste *.fee *.mpr`
own filetypes	`-`

5.2 Syntax

```
adapt v1.1 SECT Research Group, Heriot-Watt University Edinburgh
usage: adapt input d mprtype
note all filenames are referred to without extensions
   input    : name of the mesh definition and stress file
   d        : permissible error value
   mprtype  : 1 for element mesh params
            : 2 for nodal mesh params
```

5.3 Overview

`adapt` is small error analysis program for the sort of meshes that can be handled by `fem`. It generates finite element errors on the basis of the stress file by comparing averaged and non averaged nodal stresses.

The finite element element errors are then converted into mesh parameters either nodal or element parameters as specified in the command line. Make sure you choose the right type depending on what remeshing or viewing you want to do next.

It is always advised to check the generated mesh parameters before a new remeshing run is launched. Some mesh parameters might be far to small to be realistic. The `floor` program might be used to set a lower limit to the mesh parameters. `floor` is explained in Section 7.4.

5.4 Program limitations

- Only meshes consisting entirely of one of the two supported element types can be analysed.
- The material file should also be there as material information is required for processing the finite element errors.

- Both material and stress files are expected to have the same name as the mesh definition file.

Chapter 6

Viewing and printing: E_PLOT32

6.1 Status

name of executable(s)	`e_plot32.exe`
platform(s)	`win32`
current version	`v2.52`
date	`July 2003`
release	`complete`
E_lib filetypes	`*.mdf *.mpr *.dom *.fee *.ste`
own filetypes	`-`

6.2 Syntax

```
run e\_plot32.exe from explorer or start menu - run...
or make a shortcut to e\_plot32.exe
```

6.3 Overview

E_plot32 is a native windows 32bit graphical user interface program. It will run on Windows95, Windows98 and Windows NT 4. The program provides a viewer for all sorts of meshes in the E_Library mesh definition format.

The user can view the mesh in the following plot configurations:

- plain mesh plot: just shows the mesh connectivity.
- subdomain plot: colours the different subdomains according to the subdomain information stored in the appropriate subdomain file.
- stress plot: shows a colour representation of all the stresses that are available in the stress file.
- FEerror plot: plots the distributions of the finite element errors as specified in the FEerror file.
- mesh parameter plot: makes a plot of the mesh parameters as stored in the mesh parameter file.

The program can print directly from the menu and also has an export facility to the `*.wmf` format which can be read by many vector graphics packages such as CorelDraw and AutoCAD 2D.

6.4 Help

For a more detailed explanation of each menu option we refer to the online help system within the program. Help is available by selecting the Help menu or by simply pressing **F1** at any time in the program.

6.5 Program limitations

- All files should have the same name as the input `*.mdf` files with the appropriate extensions.
- All filetypes are accepted except `QUAD3` and `BLOCK1`.
- For meshes containing `TRIANG2` elements: only the stresses for `layer 0` are shown in a stress plot.
- When exporting a subdomain plot to `*.wmf` format the subdomain colours are converted to grayscale. This does not happen with the stress, FEerror or mesh parameter plots.
- An export to `*.wmf` might mess up the scaling of the image. Scale it by a factor 100 to obtain normal sizes. However this depends on the graphics package you use.
- E_plot32 will only accept element mesh parameters for a mesh parameter plot.

6.6 Alternative platforms

The viewer technology of E_plot32 was also ported into a Linux and Irix version. These unix versions of the viewer have limited features and are continuously updated. If required obtain the most recent information from the author concerning the status of `eplx` and `qmv`.

Chapter 7

Editing and modifying E_Library files: MDFTOOLS

The MDFTOOLS are a series of checking, editing and conversion tools to help creating and maintaining E_Library files. Some of them are batch files, some are small programs. A good knowledge of scripting or shell languages is advised in the preparation of E_Libary files as there is (so far) no graphical design utility for the E_Libary standard.

7.1 CHKMDF

7.1.1 Status

name of executable(s)	`chkmdf`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`*.mdf`
own filetypes	`-`

7.1.2 Syntax

```
Validity checking of an e_lib mesh definition

Usage: chkmdf filename
   filename : name of mesh definition file
All files should be specified without extension!
```

7.1.3 Overview

`chkmdf` is a tool that checks the validity of a mesh definition file. It displays mesh statistics and performs the following checks:

- Are there any duplicate nodes?
- Are there any invalid elements, i.e. elements in whose definition the same node number appears twice or more?
- Are there any duplicate elements? Here the node order is not taken into account.

7.2 CLEANUP

7.2.1 Status

name of executable(s)	`cleanup` | `cleanup.exe`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf`
own filetypes	`-`

7.2.2 Syntax

```
Remove unconnected nodes from a mesh.

Usage: cleanup input output
   input  : name of input mesh definition file
   output : name of resulting mesh definition file
All files should be specified without extension!
```

7.2.3 Overview

When meshes are adaptively remeshed with `mgn` using the task `REMESH` all the nodes of the input coarse mesh are copied into the resulting file though they might not be used in any of the new triangles. These unconnected nodes cause problems when renumbering the mesh for bandwidth reduction.

`cleanup` removes all unconnected nodes from a mesh definition file. The program only works with meshes containing only `TRIANG1` elements. The topology of the mesh is not altered, only the numbering of points and the point numbers in the element definition. Boundary conditions and loads are also renumbered appropriately.

7.3 EMP2MPR

7.3.1 Status

name of executable(s)	`emp2mpr`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf mpr`
own filetypes	`-`

7.3.2 Syntax

```
Convert element mesh parameters to nodal mesh parameters
by averaging all element mesh parameters connected to a node.

Usage: emp2mpr input output
   input  : name of input mesh file and mesh parameter file
   output : name of output mesh parameter file
All files should be specified without extension!
```

7.3.3 Overview

`emp2mpr` converts an element mesh parameter file into a nodal mesh parameter file. The program requires, as an input, the mesh definition file and the element mesh parameter file, while the output is only a new nodal mesh parameter file. The program only works with meshes containing only `TRIANGLE1` elements.

7.4 FLOOR

7.4.1 Status

name of executable(s)	`floor` | `floor.exe`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`*.mdf *.mpr`
own filetypes	`-`

7.4.2 Syntax

```
Limit the maximum and minimum values of the mesh parameter.

Usage: floor input output
   input  : name of input mesh file and mesh parameter file
   output : name of output mesh parameter file
All files should be specified without extension!
```

7.4.3 Overview

`floor` is a little utility to alter mesh parameter files. Both nodal and element mesh parameter file are accepted. The file is parsed and minimum, maximum and average mesh parameter are printed on the screen. Then the user is asked to specify a new minimum and maximum parameter for this file.

7.5 MAKEDOM

7.5.1 Status

name of executable(s)	`makedom`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf dom`
own filetypes	`-`

7.5.2 Syntax

```
Create a domain decomposition file (*.dom).

Usage: makedom [-v|-h|-n<domain>] mdf_file
        -v - print version information
        -h - print this message
        -n<domain> - set domain number (default 1)
```

7.5.3 Overview

`makedom` creates a domain decomposition file which is compatible with the input mesh definition file. By default it generates a domain decomposition where all elements are in one domain. However it is also possible to specify that all elements should be assigned to a user defined subdomain. For example `makedom -n 2 mesh` will generate a domain decomposition file where all elements belong to subdomain two and the maximum number of subdomains is also equal to two.

7.6 MAKEMAT

7.6.1 Status

name of executable(s)	`makemat`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf mat`
own filetypes	`-`

7.6.2 Syntax

```
Creates material information and an example material file

Usage: makemat input output
   input  : name of input mesh file
   output : name of output mesh and material file
All files should be specified without extension!
```

7.6.3 Overview

`makemat` creates material information for a mesh definition file and it also generates an example material file (**.mat**). The program only accepts meshes which contain a single element type. Moreover the acceptable element types are TRIANG1, TRIANG3 and LINK1.

7.7 MAKEMPR

7.7.1 Status

name of executable(s)	`makempr`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf mpr`
own filetypes	-

7.7.2 Syntax

```
Creates mesh parameter file (element or nodal)

Usage: makempr [-v|-h|-n<meshpram>|-e<meshparam>|-a] mdf_file
        -v - print version information
        -h - print this message
        -n<meshparam> - set value of nodal mesh parameter
        -e<meshparam> - set value of element mesh parameter
        -a - automatic calculation of nodal mesh parameter (default)
```

7.7.3 Overview

`makempr` can create a nodal or element mesh parameter file. When the mesh parameter is explicitly specified on the command line all mesh parameters in the file will be the specified value. In the case of the `-a` option the program determines the minimum edge length at each point and this minimum edge length will become the nodal mesh parameter at each point.

7.8 MDF2K

7.8.1 Status

name of executable(s)	`mdf2k`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`development`
E_lib filetypes	`mdf`
own filetypes	`-`

7.8.2 Syntax

```
Converts and MDF file to an LS-DYNA keyword file
describing only the geomtery of the mesh.

Usage: mdf2k file(.mdf) n > out.k
   n : the numbering of the nodes and elements starts at n*10000+1
```

7.8.3 Overview

The `mdf2k` program converts a mesh geometry to an LS-DYNA keyword file. The created LS-DYNA file only describes the geometry, no other data is created. The numbering of the nodes and elements starts at $n * 10000 + 1$.

7.9 MDFMERGE

7.9.1 Status

name of executable(s)	`mdfmerge`
platform(s)	`irix` | `linux`
current version	`v1.0`
date	`July 2003`
release	`development`
E_lib filetypes	`mdf`
own filetypes	`-`

7.9.2 Syntax

```
Merges two mdf files.
Only merges geometry and domain decomposition,
but no loads, bc, etc.

Usage: mdfmerge mesh1 mesh2 out [d]
   mesh1: first mesh definition file
   mesh2: second mesh definition file
   out:   output mesh definition file
   d :    when not zero it also merges decomposition data
```

7.9.3 Overview

The `mdfmerge` program merges the geometry of two mesh definition files. When the optional argument is specified and it is not zero then the program also merges the domain decomposition data in the mesh. However the program does not merge loads, boundary conditions, materials, etc.

7.10 MPR2EMP

7.10.1 Status

name of executable(s)	`mpr2emp`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf mpr`
own filetypes	`-`

7.10.2 Syntax

```
Convert nodal mesh parameters to element mesh parameters
by averaging all nodal mesh parameters of an element.

Usage: mpr2emp input output
   input  : name of input mesh file and mesh parameter file
   output : name of output mesh parameter file
All files should be specified without extension!
```

7.10.3 Overview

`mpr2emp` converts a nodal mesh parameter file into an element mesh parameter file. The program requires, as an input, the mesh definition file and the nodal mesh parameter file, while the output is only a new element mesh parameter file. The program only works with meshes containing only `TRIANGLE1` elements.

7.11 RENUMBER

7.11.1 Status

name of executable(s)	`renumber` | `renumber.exe`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`*.mdf`
own filetypes	-

7.11.2 Syntax

```
Renumbers the mesh nodes to reduce the matrix bandwidth.

Usage: renumber input output
   input  : name of the input mesh definition file
   output : name of the resulting mesh definition file

All files should be specified without extension!
```

7.11.3 Overview

`renumber` is a small program that changes the node indices of a mesh in order to minimize the bandwidth of the stiffness matrix when this one is build in any finite element package. It changes the node numbers accordingly in loads and boundary condition nodes.

7.11.4 Program limitations

- `renumber` works only for single element type `TRIANG1` or `QUAD1` meshes.
- Only the information within the mesh definition file is altered. Subdomain, stress, finite element error or mesh parameter information is left untouched.
- The program will crash if the mesh contains unconnected nodes. These are nodes which are not used in any of the element definitions. Use the MDFTOOL `cleanup` to remove unconnected nodes. `cleanup` is explained in Section 7.2

7.12 REV

7.12.1 Status

name of executable(s)	`rev`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`development`
E_lib filetypes	`mdf`
own filetypes	`-`

7.12.2 Syntax

```
Revolves quad elements around an axis to create blocks.
The mesh must be in the x-y pane.

Usage: rev mesh n [-o angle] [-x -y]
   -o angle: Open revolution by angle [degree]
   -x : Revolution around x axis
   -y : Revolution around y axis

Defaults are rotation around x axis and rotation by 360 degrees
to form a closed torus
```

7.12.3 Overview

The `rev` program revolves a mesh containing only QUAD1 elements around the x or the y axis to form block (hexahedral) elements. The program is capable to generate a closed torus or by using the `-o` option only a part of a torus. This later case is called open revolution. By default the program revolves the mesh around the x axis, but specifying the `-y` option it will revolve the mesh around the y axis. For example, the command `rev test 10 -o 90 -y` will generate the mesh as shown in Figure 7.1. The output of the program is written into the `rev.mdf` file.

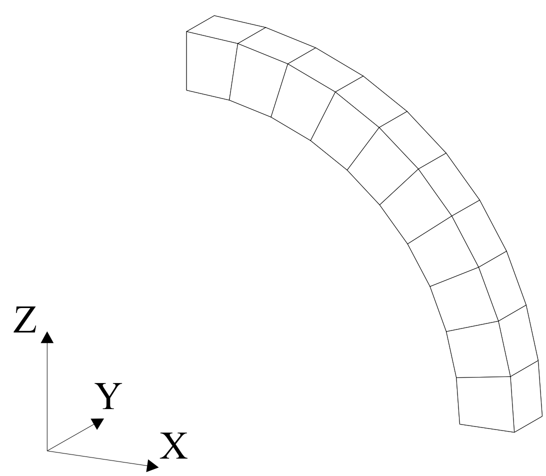


Figure 7.1: Result of `rev test 10 -o 90 -y`

7.13 SPHERE

7.13.1 Status

name of executable(s)	`sphere`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`development`
E_lib filetypes	`mdf`
own filetypes	`-`

7.13.2 Syntax

```
Generate a surface mesh of a sphere.

Usage: sphere n [-q|-t]
  n = number of subdivisions per side
 -q = generate QUAD1 elements
 -t = generate TRIANG1 elements
```

7.13.3 Overview

`sphere` generates a surface mesh of a sphere. The mesh may contain triangle or quad elements. The program initially generates a cube whose points are then displaced onto the surface of a sphere. The number of divisions specifies the number of divisions of an edge of the cube. The name of the output mesh is `triangs.mdf` when triangle elements are generated and `quads.mdf` when quad elements are generated.

7.14 VSPH

7.14.1 Status

name of executable(s)	`vsph`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`development`
E_lib filetypes	`mdf`
own filetypes	-

7.14.2 Syntax

```
Generate a volume mesh of a sphere.

Usage: vsph n
  n = number of subdivisions per side
```

7.14.3 Overview

`vsph` generates a volume mesh of a sphere. The output mesh will contain hexahedral elements. The program initially generates a cube whose points are then displaced to form a sphere. The number of divisions specifies the number of divisions of an edge of the cube. The name of the output mesh is `vsph.mdf`.

7.15 TRF

7.15.1 Status

name of executable(s)	`trf`
platform(s)	`irix` | `linux` | `win32 command line`
current version	`v1.0`
date	`July 2003`
release	`stable`
E_lib filetypes	`mdf`
own filetypes	`trf`

7.15.2 Syntax

```
Usage: trf  mesh transformation [-N]
   mesh           : name of the mesh definition file
   transformation : name of the transformations file
    -N            : also transform the geometric model
All files should be specified without extension!
```

7.15.3 Overview

`trf` is a small tool that allows you to perform geometrical operations on the coordinates array of a mesh. Mesh connectivity will not be touched. The output of the program is `mesh.trf.mdf` where `mesh` is the input filename.

The geometrical operations that `trf` can handle are:

- move: move every meshpoint over a fixed vector in space
- scale: scale all meshpoints in the three coordinate directions
- rotate: rotate meshpoints round the coordinate axes

7.15.4 Syntax of transformation file

The transformations file (***.trf**) has the following syntax:

```
NTRFS   n               /* the number of transformations defined below */

MOVE    dx dx dy        /* move operation over vector (dx,dy,dz) */
SCALE   sx sy sz        /* scale operation with scale factors si */
ROTATE  ax ay az        /* rotate operation around angles ai {-360,360] */
..
```

Moreover you can carry out as many operations as you want but some are equivalent:

```
MOVE    dx dy dz        ==      MOVE    dx 0  0
                                MOVE    0  dy 0
                                MOVE    0  0  dz

SCALE   sx sy sz        ==      SCALE   sx 1  1
                                SCALE   1  sy 1
                                SCALE   1  1  sz

ROTATE  ax ay az        ==      ROTATE  ax 0  0
                                ROTATE  0  ay 0
                                ROTATE  0  0  az
```

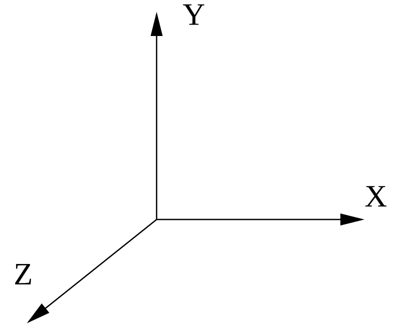


Figure 7.2: Coordinate system

Note that unlike moving or scaling the order of the rotations carried out does matter! So when you issue a `ROTATE ax ay az` where none of `ax`, `ay` and `az` are equal to zero, the rotation around the X axis will be carried out first, then the Y rotation and finally the Z rotation.

7.15.5 Coordinate system

The coordinate system used is shown in Figure 7.2. The X-Y plane corresponds to the screen and the Z-axis is pointing towards you. The rotation angles are positive when they appear to be counter-clockwise hen you look into the oriented axis. For example: `ROTATE 0 90 0` would transform the Z axis into the X axis.

Bibliography

[1] Chew, P.L., "Guaranteed-Quality Delaunay Triangulations", Technical Report TR-89-983, Dept. of Computer Science, Cornell University, 1989.

[2] Muylle, J., Iványi, P., Topping, B.H.V., "A new point creation scheme for uniform Delaunay triangulations", *Engineering Computations, International Journal for Computer Aided Engineering and Software*, 19(6): 707–735, 2002.

[3] PDA Engineering, PATRAN Division, 2975 Redhill Avenue, Costa Mesa, California, USA, *PATRAN Plus User Manual*, October 1990.

[4] SECT Research Group, Heriot-Watt University, *E-lib User's Guide*, 1999.