Biomechanics

Concepts and Computation

SOFTWARE MANUAL

Cees Oomens, Boudewijn Deurloo, Marcel Brekelmans

Version 1.0 - September 2008

Eindhoven University of Technology
Department of Biomedical Engineering
Tissue Biomechanics & Engineering

Contents

1
Introduction

In the exercises attached to the Chapters 14 to 18 of the book 'Biomechanics: concepts and computation' extensive use is made of Matlab-scripts, which have to be written by students themselves, as well as a Finite Element Code (also written in Matlab), that can be downloaded from the website: http://www.mate.tue.nl/biomechanicsbook.

Applying this code, requires that the reader has a licence for the use of Matlab, which can be obtained via MathWorks. For this, see the website: http://www.mathworks.com.

The finite element software is called `mlfem_nac`. The present document is a manual for the installation and use of this software.

The general structure of a finite element method (FEM) program is given in Fig. (2.1):

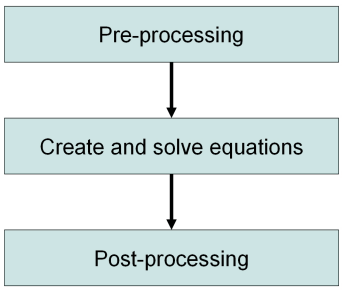

Fig. 1.1. Flow chart of a finite element program

In the pre-processing phase the model is created. This means that the coordinates of `userpoints` are defined. These are used to create curves, surfaces and, in the three-dimensional case, volumes. This defines the geometry of the system that is modelled. Together with information on element types and the required mesh refinement the program uses this information to generate a finite element mesh. After that, boundary conditions, material and possibly additional geometrical properties have to be added.
The information is used by the finite element program to create an element stiffness matrix and a right-hand-side array. Subsequently the set of equations will be solved. The result is stored in the solution array.
In the post-processing phase of the program user required information is derived from the solution array and shown in the form of graphs and/or contour plots.

Chapter 3 describes the installation and implementation of the software. After that the different in- and output arrays will be discussed in Chapter 4. Their meanings will be clarified by different examples. It is important to be familiar with these arrays before starting to activate the program. During the 'solving phase' the program uses element functions to compute the element matrix and right-hand-side column. The meaning of the different element functions and the way they are related will be outlined in Chapter 5.
In Chapters 5, 6 and 7 different 'demo files' are discussed. These files are given as examples on how to use `mlfem_nac` for one-dimensional and two-dimensional problems. Each phase (see Fig. 2.1) will be discussed separately.
Finally different plot functions defined in Matlab will be given in Chapter 8. These may be convenient to interpret the output by for example giving element numbers, node numbers, etc..

2
Installation

Make sure you have a working version of Matlab on your computer. The program `mlfem_nac` should work with any version higher than Matlab 5.1.

Download the program `mlfem_nac` from the website:
http://www.mate.tue.nl/biomechanicsbook.

Unpack and decode the program by means of the program WinZip (see: http://www.winzip.com/index.htm).

After unpacking a root-directory `../mlfem_nac` is created with a number of sub-directories. In Windows Explorer this should look like:

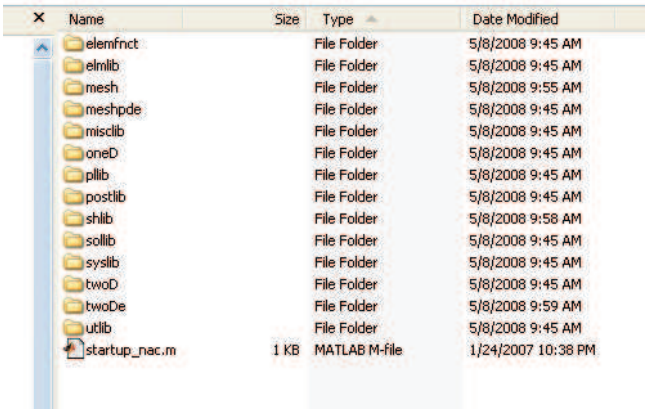

Fig. 2.1. Directory tree of the finite element code

After unpacking the following steps have to be taken:

(1) Open Matlab
(2) Select the correct 'current directory'. In this case that is the pathway to mlfem_nac.
(3) Enter `startup_nac` in the command window. This is a script that sets the paths of Matlab, so all scripts in the different directories can be found by Matlab. This script has to be run before you start to work with mlfem_nac.
(4) The program library includes a number of demo files, which are examples of simulations that can be done with the finite element code. These demo files can be found in the directories `oneD`, `twoD` and `twoDe`. To test whether the software is properly installed one of the demo files can be run. Go to the directory `twod`. Run `demo_cd` by typing this in the Matlab command window. This should result in the following Fig. 3.2.

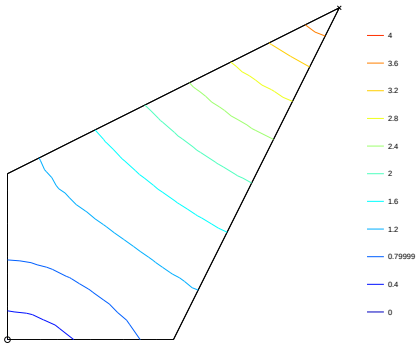

Fig. 2.2. Installation check

When Fig. 3.2 is found after running the file `demo_cd` the installation seems to be correctly performed and the software is ready to use.

3
Some important arrays

In this chapter different arrays, used in the program, are outlined. These arrays are subdivided into three categories:

- The arrays `coord`, `top` and `mat.mat` respectively, which are used to specify the mesh and the material properties,
- the arrays `bndcnd` and `nodfrc`, to define the boundary conditions,
- the arrays `pos` and `dest`, that give information about the location of the degrees of freedom in the global solution array.

The finite element mesh in Fig. 4.1 is used to explain the meaning of the arrays. This mesh consists of one 4-noded quadrilateral element (having element number 1) and two 3-noded triangular elements (having element number 2 and 3). The coordinates of the user points (in this case also the nodes of the mesh) are given.

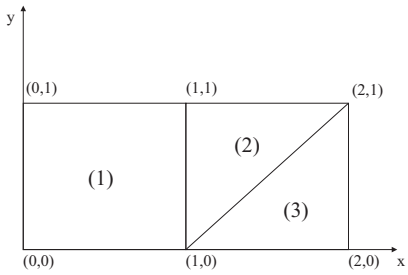

Fig. 3.1. Example of a finite element mesh with coordinates of the nodes

3.1 Mesh specification

In this section the arrays used to define the finite element mesh are outlined. Fig. 4.2shows the same mesh as in Fig. 4.1, but now including the node numbers.

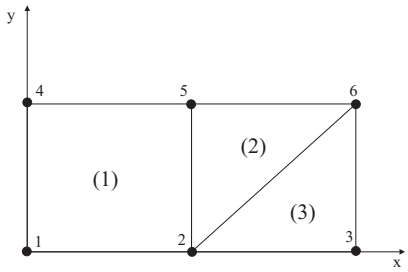

Fig. 3.2. Example of a finite element mesh with node numbers

3.1.1 Coordinates

The array `coord` specifies the coordinates of the different nodes in the mesh. Depending on the number of dimensions only x-, or x- and y-coordinates are given for each node. The coordinates in the i'th row of the coordinate-array belong to the i'th node (`inode`) in the mesh.

In the two-dimensional case the coordinates of a certain node `inode` can be extracted from the array `coord` by the following command:

```
[x y]=coord(inode,:)
```

Example: The array `coord` that belongs to Fig. 4.2 is structured as follows:

```
coord=[0 0
       1 0
       2 0
       0 1
       1 1
       2 1]
```

3.1.2 Topology

The array `top` specifies the topology of each element. The i'th row in this matrix refers to the i'th element of the mesh. The first `maxnodelm` (maximum number of nodes per element) columns are used for the node numbers. The last two column of the array `top` contain identifiers pointing to the structured array `mat` that will be outlined below. The second last column contains a material property identifier (`iimat`) and the last column contains an identifier for the element function that is used to calculate the stiffness matrix (`iitype`).

```
top(ielem,:)=[node1 node2 ... maxnodelm ... iimat iitype]
```

If, in the mesh, elements are used with a different number of nodes per element, the highest number of nodes (`maxnodelm`) determines the number of columns used to specify the nodes per element. If an element has less than `maxnodelm` nodes the remaining columns are filled with zero's.

Example:

```
top=[1 2 5 4 1 1
     2 3 6 0 2 1
     2 6 5 0 3 1]
```

In this case the maximum number of nodes is 4. Note, the zero's in the second and third row (fourth column) belonging to the 3-noded triangular elements. In this example different material property identifiers are used for each element, while the same element function is used to compute the stiffness matrix. Note, that in this example the nodes that define the elements are all ranked anti-clockwise. Most finite element packages require that elements are consistently either numbered clockwise or anti-clockwise.

Extracting the node numbers from a certain element `ielem` from the matrix `top` is done by the command line:

```
nodes=nonzeros(top(ielem,1:maxnodelm))
```

In this command line `maxnodelm` is the largest number of nodes found in one single element. The Matlab function `nonzeros` extracts all numbers from an array that have a value that is not equal to zero.

3.1.3 Material properties and element type

The material properties and the element type are defined using the structured array mat.

```
mat.mat(iimat,:)=[m1 m2 ... mn]
```

Here `mi` denotes the i'th material property. The index `iimat` stands for the material property identifier (see Topology). Depending on the type of element that is used a different number of material properties may be required. How many properties are required and the meaning of each material property, can be found in the next chapter. These properties may actually be anything that is used within the element functions to compute the stiffness matrix. It is necessary to know the way in which the properties are ordered.

mat.types(`iitype`,:)='element function name'

Here `iitype` is an identifier for the element function of a certain group of elements (see Topology). Usually, only one type of element function will be used in a simulation, so `iitype`=1. The different element functions can be found under the directory `elmlib` and will be discussed in more detail in the next Chapter. Possible choices are: '`elm1d`', '`elm1dcd`', '`elcd`', and '`ele`'.

3.2 Boundary conditions

To find a unique solution for the system of equations, boundary conditions are necessary. There are two different types of boundary conditions (essential and natural). In the following subsections these types of boundary conditions will be discussed.

3.2.1 Essential boundary conditions

Essential boundary conditions (number: `ibnd`) are specified in the array `bndcon`:

```
bndcon(ibnd,:)=[inode idof value]
```

where `inode` denotes the node number, `idof` a degree of freedom and

`value` the value to be assigned to this degree of freedom. The program `mlfem_nac` can only be used for one- and two-dimensional problems. In case of a one-dimensional problem `idof=1`.
In the two-dimensional case it depends on the type of problem.

- For convection diffusion proplems the unknown is a scalar property and in that case `idof`=1.
- For elastic problems `idof=1` for the nodal displacement in x-direction and `idof=2` for the nodal displacement in y-direction.

When the node numbers associated with the essential boundary conditions are known the array `bndcon` can be defined directly with an editor in the input file, or it can be created by means of the function `addbndc`. The latter method will be explained in Chapter 6.

Example:

Suppose that in the example of the mesh in Fig. 4.1 the displacements at node 1 and 4 are suppressed in both the x- (`idof=1`) and y-direction (`idof=2`), while the displacement in the x-direction at node 6 is set to 10^{-3}, then:

```
bndcon=[1 1 0
        1 2 0
        4 1 0
        4 2 0
        6 1 1e-3]
```

3.2.2 Natural boundary conditions

The definition of natural boundary conditions (nodal forces in case of a mechanical problem) is similar to the definition of the essential boundary conditions. The array used is `nodfrc`:

```
nodfrc(ibnd,:)=[inode idof value]
```

where `inode` denotes the node number, `idof` the related degree of freedom number and `value` the value to be assigned to this nodal right-hand side component.

In principle natural boundary conditions have to be prescribed in each direction at each boundary node, where no essential boundary conditions are prescribed. To avoid unnecessary complicated preprocessing, automatically in all directions at all boundary nodes for which no essential boundary conditions are prescribed, the natural boundary condition is taken as zero, unless stated otherwise. In other words, the default value of the natural boundary condition is zero. The user only has to define nonzero boundary conditions if that is necessary for the problem at hand.

Example:
Suppose that for node 3 the nodal force in the y-direction is prescribed, say $3*10^2$, then:
`nodfrc=[3 2 3e2]`

3.3 Location of degrees of freedom in the global solution array

The solution array `sol` for time-independent problems is a single column containing all degrees of freedom of the finite element mesh (in an ordinary two-dimensional elastic problem this means that the number of entries in the column is twice the number of nodes). The numbers attributed to these degrees of freedom can be randomly assigned and are not related to the node numbers. That is why the program uses two arrays (`dest` and `pos`) to keep track of which degrees of freedoms in the array `sol` belong to which element and which node. Both arrays are available for the user after the solution of the problem to extract specific information from the solution column `sol`.

3.3.1 The array pos

The array `pos` contains the position of the degrees of freedom within the global solution array of the nodes of each specific element:

```
pos(ielem,:)=[locdof1 locdof2 ...]
```

Here `ielem` stands for the element number and `locdofi` for the location of the degrees of freedom, belonging to this element, in the global solution array `sol`. The degrees of freedom associated with element `ielem` may be extracted from the solution array by:

```
ii=nonzeros(pos(ielem,:))
uelem=sol(ii)
```

The command `nonzeros` is used because of the fact that not every element has an equal number of degrees of freedom defined. Zero's are used to fill the array. The number of columns of the array is determined by the highest number of degrees of freedom for an element defined.

Example
For example the `pos` array for the problem in Fig. 4.2 might look as:

```
pos = [ 1  2 11 12 3 4 9 10
       11 12  5  6 7 8 0  0
       11 12  7  8 3 4 0  0]
```

Note, that again the rows are completed with zeros for the elements with less degrees of freedom. The above specifiactions pf `pos` would mean that the sol array would contain the following displacement solution:

$$\text{sol} = \begin{bmatrix} u_x^1 \\ u_y^1 \\ u_x^5 \\ u_y^5 \\ u_x^3 \\ u_y^3 \\ u_x^6 \\ u_y^6 \\ u_x^4 \\ u_y^4 \\ u_x^2 \\ u_y^2 \end{bmatrix},$$

where the subscript indicates the direction and the superscript the node number. It is clear that the degrees of freedom are completely mixed.

3.3.2 The array dest

The array `dest` contains the position of the degrees of freedom within the solution array of each specific global node:

```
dest(inode,:)=[locdof1 locdof2]
```

Here `inode` stands for the node number and `locdof1` and `locdof2` for the positions in the solution array `sol` of the degrees of freedom in the x- and y-direction. In the present course this is only the case for two-dimensional linear elastic problems. In this case each node has 2 degrees of freedom. For two-dimensional convection-diffusion problems only one degree of freedom (e.g. a concentration or a temperature) in each node is found and the array `dest` consists of one single column.

Example

For the example in Sub-section 4.3.1 the `dest` array would look like:

$$\text{dest} = \begin{bmatrix} 1 & 2 \\ 11 & 12 \\ 5 & 6 \\ 9 & 10 \\ 3 & 4 \\ 7 & 8 \end{bmatrix} .$$

If `sol` contains the solution vector, the degrees of freedom associated with global node `inode` may be extracted from the solution vector by:

```
ii=(dest(inode,:)
unode=sol(ii)
```

4
Element functions

The demo-files in the directories `oneD`, `twoD` and `twoDe` show that indeed every demo consists of a pre-processing part (defining the mesh, material properties and the boundary conditions), followed by a call of a script for the solution process and finally a post-processing part. Different scripts are used for different problem types:

(1) The script `fem1d` is used to solve one-dimensional diffusion problems
(2) The script `fem1dcd` is used to solve one-dimensional convection-diffusion problems
(3) The script `femlin_cd` is used to solve two-dimensional convection-diffusion problems
(4) The script `femlin_e` is used to solve two-dimensional linear elastic problems.

These scripts read and interpret the input arrays, calculate the element contributions to the stiffness matrix and right-hand side of each element, assemble the global stiffness matrix, partition the stiffness matrix in accordance with the used boundary conditions and solve the problem. To calculate the element contribution different element functions are used:

`elm1d`:	linear one-dimensional diffusion element
`elm1dcd`	linear one-dimensional convection diffusion element
`elcd`:	two-dimensional convection diffusion element
`ele`:	two-dimensional linear elasticity element

It is worthwhile to have a look at these element files because they illustrate the theory as presented in the book quite clearly.
To each element function `elmfnc` additional functions have to be specified, namely:

- **elmfnc_i** These functions provide all sorts of information such as the number of degrees of freedom for each node, the shape function identifiers to be used in `elmfnc_s`, information about the number of derived quantities at each integration point and nodal points as computed by `elmfnc_d`.
- **elmfnc_s** These functions compute the shape functions and their derivatives locally (per element) based on information of `elmfnc`.

Next to these, two more additional element functions exist:

- **elmfnc_d** When the solution has been determined for a number of problems so-called derived quantities may be computed, such as stress components at the integration points and nodes of each element.
- **elmfnc_a** Only for the element function `elcd` this additional function exists. This function (`elcd_a`) is used to define the convective velocity.

In the next section each element function will be described in detail.

4.1 Description of the available element functions

4.1.1 elm1d

The function `elm1d` calculates the element contribution in case of the one-dimensional diffusion equation:

$$\frac{d}{dx}\left(c\frac{du}{dx}\right) + f = 0;$$

The theory behind this element is described in Chapter 14 of the book: 'Biomechanics: concepts and computation'.

In the structured array `mat.mat` only the entries 1, 3 and 4 have a meaning:

```
mat.mat(1) = c       % the diffusion coefficient
mat.mat(3) = f       % the distributed load or source term
mat.mat(4) = norder  % norder=1 linear element
                     % norder=2 quadratic element
```

4.1.2 elm1dcd

The function `elm1dcd` calculates the element contribution in case of the one-dimensional convection-diffusion equation:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \frac{\partial}{\partial x}\left(c\frac{\partial u}{\partial x}\right) \ .$$

The theory behind this element is described in Chapter 15 of the book: 'Biomechanics: concepts and computation'.

In the structured array `mat.mat` the entries 1 to 3 and 5 have the meaning:

```
mat.mat(1) = c          % the diffusion coefficient
mat.mat(2) = v          % the convective velocity
mat.mat(3) = f          % the distributed load or source term
mat.mat(5) = norder;    % norder=1 linear element
                        % norder=2 quadratic element
```

The function can be used for stationary as well as time-dependent problems (See Chapter 5).

4.1.3 elcd

The function `elcd` calculates the element contribution in case of the two-dimensional convection-diffusion equation:

$$\frac{\partial u}{\partial t} + \vec{v}\cdot\vec{\nabla}u = \vec{\nabla}\cdot\left(c\vec{\nabla}u\right) + f$$

The theory behind this element is described in Chapter 16 of the book: 'Biomechanics: concepts and computation'.

In the structured array `mat.mat` the entries 1 to 4 and 11 have the meaning:

```
mat.mat(1) = c        % diffusion coefficient
mat.mat(2) = fx       % parameter used in elcd_a to
                      % determine fluid velocity
mat.mat(3) = fy       % parameter used in elcd_a to
                      % determine fluid velocity
mat.mat(4) = axi      % Optional parameter. Default value is 0.
                      % axi = 0 : planar flow
                      % axi = 1 : axi-symmetric problem
```

```
mat.mat(11)=norder+2  % norder=1 linear element
                      % norder=2 quadratic element
```

It should be clear that in this case the parameters fx and fy should be used to define a fluid velocity profile. This profile can be defined by means of the function `elcd_a` (also see chapter 6.
The function `elcd` can be used for stationary as well as time-dependent problems (See Chapter 5).

4.1.4 ele

The function `ele` calculates the element contribution in case of a two-dimensional linear elastic problem:

$$\vec{\nabla} \cdot \boldsymbol{\sigma} + \vec{f} = \vec{0} \ ,$$

with the isotropic Hooke's law describing the material behaviour. The theory behind this element is described in Chapter 18 of the book: 'Biomechanics: concepts and computation'.

In the structured array `mat.mat` the entries 1 to 3 and 11 have the meaning:

```
mat.mat(1) = E        % the Young's modulus
mat.mat(2) = nu       % Poisson's ration
mat.mat(3) = axi      % axi = 0 : plane strain
                      % axi = 1 : axi-symmetric problem
                      % axi = 2 : plane stress
mat.mat(11)= ieltop   % parameter defines element topology
```

The parameter `ieltop` requires some explanation. In the case of a linear elastic element it is possible to make a choice between quadrilateral elements or triangles. This is done by means of the parameter `itype` that has to be specified before the routine `crmesh` is called. In case of a quadrilateral `itype`=1, in case of a triangular element `itype`=20. Beside that, a choice can be made between elements with a linear interpolation (norder=1) of the displacements or an element with quadratic interpolation (norder=2). This leads to the following available choices for `ieltop`:

ieltop = 1 : linear triangle
ieltop = 2 : quadratic triangle
ieltop = 3 : bi-linear quadrilateral element
ieltop = 4 : quadratic quadrilateral element

5
The demo files in the directory OneD

Chapter 14 and 15 of the book 'Biomechanics: Concepts and Computation' are devoted to one-dimensional diffusion and convection-diffusion problems. To determine approximate solutions the scripts `fem1d` and `fem1dcd` are used respectively, which can be found in the directory `oneD`. In the present chapter the demo files that make use of these scripts will be explained. These demo files can also be found in the directory `OneD`. As a typical example the demo file `demo_fem1dcd` is explained here.

The pre-processing starts by defining the nodes and the mesh. For one-dimensional problems a sophisticated mesh generation program is not necessary. The arrays `coord` and `top` can be filled in a straightforward way.

```
% demo of fem1dcd
% this code solves the 1D convection diffusion problem
%
% du/dt + a du/dx = d/dx(c du/dx) + f

clear all        % clears memory
close all        % closes all figures

%%%%%%%%%%%%%%%%%%%% DEFINE THE MESH %%%%%%%%%%%%%%%%%%%%%%

xmin  =  0;           % domain = [xmin xmax]
xmax  =  1;           %
nelem = 10;           % number of elements
norder = 1;           % element order
```

```
% coordinates

dx=(xmax-xmin)/nelem  ;  % the distance between the nodes
coord=(xmin:dx:xmax)' ;  % the column with nodal coordinates

% element topology

top=[(1:nelem)' (2:nelem+1)' ones(nelem,2)];

%%%% CONTROLPARAMETERS FOR TIME INTEGRATION  %%%%%%%%%%%%%%

istat   =     2;        % 1: steady state problem
                        % 2: unsteady problem
dt      =  0.01;        % magnitude of the time step
ntime   =    10;        % number of time steps
theta   =   0.5;        % theta parameter of theta-time
                        % integration scheme
```

Before the mesh is generated the minimum and maximum value of x (`xmin` and `xmax`) in the domain, the number of elements (`nelem`) and the order of the elements (`norder`) have to be specified. After that, the mesh is generated by calculating the distance between the nodes (`dx`), the corresponding nodal coordinates (`coord`) and the element topology array (`top`).

The flag `istat` defines, whether the problem is time dependent or not. If `istat=1` it is not time dependent and if `istat=2` it is. Furthermore `dt` is the time step for each time increment and `ntime` the total number of time steps. The variable `theta` stands for θ used in the θ-scheme.

```
%%%%%%%%%% DEFINE THE MATERIAL PROPERTIES %%%%%%%%%%

mat.mat(1)=1;          % c: diffusion coefficient
mat.mat(2)=1;          % v: convective velocity
mat.mat(3)=0;          % f: source term
mat.mat(4)=norder;     % norder: element order

mat.types=['elm1dcd']; % element type
```

In this part the different material properties are specified as well as the element type. As discussed earlier (see (4.1.3) it is necessary to know where the different properties are specified in the structured array `mat.mat`. Furthermore the used element function is defined in the array `mat.types`. In this case `elm1dcd` is used as element function. This element can only be used to solve one-dimensional convection-diffusion problems. See Chapter 5.

```
%%%%%%%%%% DEFINE THE BOUNDARY CONDITIONS %%%%%%%%%%
% essential boundary conditions

bndcon=[1 1 0
        nelem+1 1 1];

% natural boundary conditions

nodfrc=[];
```

In this part the essential (`bndcon`) and natural (`nodfrc`) boundary conditions are specified. In this case at $x = 0$ a value $u = 0$ is prescribed and at $x = 1$ a value of $u = 1$ is prescribed. So we have two essential boundary conditions and no natural boundary conditions. For the meaning of the arrays see Section 4.2.

```
%%%% SOLVE THE PROBLEM USING FEM1DCD %%%%

fem1dcd
```

With the information given earlier (mesh, material properties and boundary conditions) the script `fem1dcd` is used to solve the problem. The script `fem1dcd` produces different output arrays which can be used for postprocessing.
In case of time dependent problems also an initial condition has to be given. This can be done in the script `fem1dcd`. The following lines can be found in this script:

```
% determine whether this program is time dependent or not
```

```
% and define the initial condition

if istat==1, % steady solution
   sol=zeros(ndof,1);
elseif istat==2, % unsteady solution
   sol=zeros(ndof,ntime);
   % sol(:,1) contains the initial condition
end
```

In case of an instationary problem the solution array `sol` is a matrix with dimensions: `ndof` $\times$ `ntim`. This means that each column represents a time step. All entries in the column represent nodal values of the unknown u at a certain time. In the case presented here, all initial values of the solution array `sol` are set to zero. This is not really necessary, but in this way the memory is already reserved for the total matrix `sol`. When the program is running it starts with the initial values at $t = 0$ and in a recursive manner replaces the zero values at higher time steps with the correctly calculated values.
Running the demo-file as presented in the present chapter should lead to the result given in Fig. 5.1.

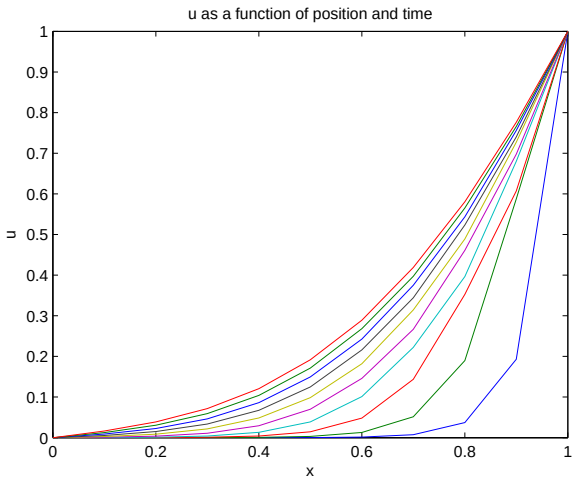

Fig. 5.1. Result of demo file demo_fem1dcd

6

The demo files in the directory TwoD

For two-dimensional convection-diffusion problems the script `femlin_cd` is used, that can be found in the directory `twoD`. The general structure of the script for pre- and post-processing is the same as in the one-dimensional case, but there are some differences. The script will be explained by means of the demo `demo_cd`.

Let us first look at the procedure to create a finite element mesh. The mesh definition starts by defining user points (corner points) of the domain for which a mesh has to be defined. This is done by filling a matrix called `points` with x- and y-coordinates. The i'th row specifies the x- and y-coordinates of user point i.

```
%%%%%%%%%%      DEFINE THE MESH          %%%%%%%%%%

% Define user points of the domain

points=[ 0 0
         1 0
         2 2
         0 1];
```

After that these points are connected by defining curves. This is done by filling the matrix `curves`. The i'th row defines curve i. A row in the matrix has to be defined as follows:

```
curves(i) = [point1 point2 nel dummy1 bias]
```

The first two entries are the numbers of the points that are used, `nel` defines the number of elements the mesh generator has apply along the curve. The fourth number is a dummy parameter that is not used. The

fifth number is a so-called bias parameter that is used, if a mesh has to be refined because high gradients of the unknown are expected. By definition:

$$\text{bias} = \frac{\text{length of last element}}{\text{length of first element}}$$

If this parameter is 1 all elements have equal lengths along the current curve. If bias > 1 the last element is larger than the first element, if bias < 1 the last element is shorter than the first element. In demo_cd are defined as follows:

```
% definition of curves

n=5;    % number of elements in the x-direction
m=5;    % number of elements in the y-direction

 curves=[ 1 2 n 1 1
          2 3 m 1 1
          3 4 n 1 1
          4 1 m 1 1];
```

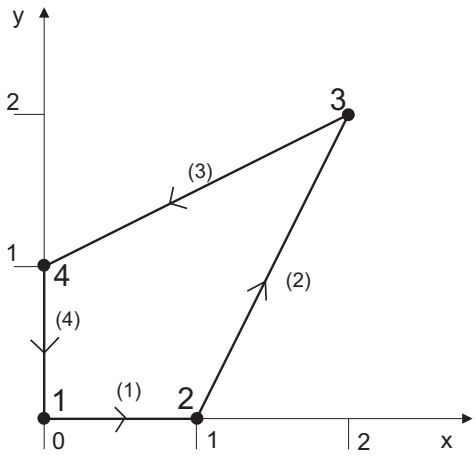

Fig. 6.1. Definition of userpoints and curves in demo_cd

Fig. 6.1 illustrates the consequences of the selections that were made so far. Two items are important to keep in mind at this point:

- It should be clear that the user points in this figure are not node numbers, but points to define a domain that is used at a later stage to create a mesh.
- Moreover, to the curves a direction has been attributed (depicted in the figure) which is defined by the first user point and the second user point given in the rows of the matrix `curves`. This direction is important when surface areas are defined in the next step.

At this point a surface has to be defined by connecting curves, that were defined earlier. This is done with the following Matlab script:

```
definition of subareas

subarea=[1 2 3 4 1];
```

In this case the curves 1, 2, 3 and 4 specify the boundaries of subarea 1. The 5th entry in the row defines an element group. If more subareas have to be defined and subareas have different material properties this last entry allows the definition of different element groups.
As mentioned earlier the direction of the curves is important. They should either all be in anti-clockwise direction or all in clock-wise direction.

Example
Assume that the third curve was defined by means of the line: `4 3 n 1 1`
In that case the curve does not match the anti-clockwise direction as is shown in Fig. 6.1. The definition of the subarea should in that case be done with the statement:

```
subarea=[1 2 -3 4 1]
```

Finally, the element order has to be defined and the function `crmesh` has to be called to create the finite element mesh:

```
norder=2;   % element order 1: bi-linear elements
            %               2: bi-quadratic elements

% create the mesh
[top,coord,usercurves]=crmesh(curves,subarea,points,norder);
```

The above procedure leads to the mesh that is shown in Fig. 6.2

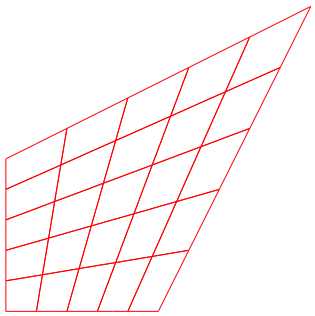

Fig. 6.2. Finite element mesh

The user is allowed to choose between solving a stationary problem or solving an instationary time dependent problem. The choice is made by means of a 'flag' `istat`. In the case `istat=1` a stationary problem is solved. In case `istat=2` the problem is time dependent. In that case the number of time steps (`ntime`), the magnitude of the time step (`dt`) and the value of `theta` used in the θ-scheme have to be specified.

```
%%%%%    CONTROL PARAMETERS FOR TIME-INTEGRATION       %%%%%
istat=1;    % 1: steady state solution, 2: unsteady solution

if istat==2,
    ntime   =   10;     % Number of time-steps
    dt      =   0.1;    % Magnitude of the time-step
    theta   =   0.5;    % Value of theta used in the
                        % theta-scheme
end
```

The only real material property in a convection diffusion problem is the diffusion constant c. This parameter has to be defined in `mat.mat(1)`.

In a convection-diffusion problem the fluid velocity profile is required. This fluid profile can be a function of the x- and y-coordinates, implying that in each integration point of each element of the simulation the convective velocity has to be calculated. This can be done by means of the user defined function `elcd_a`. This function can also be found in directory `twod`. The heading of the function is:

```
function [ax,ay]=elcd_a(x,y,fx,fy);
```

The input parameters are the x- and y-coordinate of the integration point of an element, for which the function is called, and the constant parameters `fx` and `fy`, to be defined by the parameters `mat.mat(2)` and `mat.mat(3)`. The output parameters are the convective velocities ax and ay, in x- and y-direction respectively.
So the part of the input file defining the material properties may look like:

```
%%%%%%%%%% DEFINE THE MATERIAL PROPERTIES %%%%%%%%%%
mat.mat(1)=1; % c : diffusion coefficient
mat.mat(2)=0; % fx: parameter used in elcd_a
mat.mat(3)=0; % fy: parameter used in elcd_a

mat.mat(11)=norder+2;

mat.types='elcd';   % convection-diffusion element
```

In this case the convective velocity is zero.

The boundary conditions are also a bit more complicated than in the one-dimensional case, where only values at nodal points can be prescribed. Boundary conditions in two-dimensional problems are usually prescribed along a curve. This can be done my means of the function `addbndc`. The function heading is defined as:

```
function bndcon = addbndc(bndcon,coord,top,mat
                          ,usercurves,crv,dof,string,iplot);
```

The input parameter `bndcon` is used to reserve memory. The parameters `coord, top, mat, usercurves` are necessary information about the mesh. The parameters `crv, dof, string` and `iplot` have to be supplied by the user:

- `crv` is an array that defines the curves for which a specific boundary condition will be prescribed.
- `dof` For convection-diffusion problems dof=1.
- `string` defines the value of the boundary value to be prescribed. This can either be a numerical value that is input as a string (in Matlab

format `'0'` or `'1e-3'` for example), or it can be a function of x and/or y (for example: `'x^2'` or `'y.^2'`)
- `iplot` = 1 when the user would like a plot of the mesh including boundary conditions

For each different specification of a boundary condition the function `addbndc` has to be called once. An example of boundary conditions, referring to the mesh in Fig. 6.2 could be:

```
%%%%%%%%%% DEFINE THE BOUNDARY CONDITIONS %%%%%%%%%%

% inflow boundary
bndcon=[];
crv=1;
dof=1;
iplot=0;%=1: plots the mesh and boundary conditions; =0: no-plot
string='x';
bndcon=addbndc(bndcon,coord,top,mat,usercurves,crv,dof,string,iplot);

% outflow boundary
crv=3;
dof=1;
iplot=0;%=1: plots the mesh and boundary conditions; =0: no-plot
string='y.^2';
bndcon=addbndc(bndcon,coord,top,mat,usercurves,crv,dof,string,iplot);


% no natural boundary conditions
nodfrc=[];
```

With the information given earlier (mesh, material properties and boundary conditions) the script `femlin_cd` is used to solve the problem. `femlin_cd` supplies different output arrays which can be used for postprocessing. Some convenient plot functions are given in Chapter 8.

7
The demo files in the directory TwoDe

The demo files in the twoDe directory solve two-dimensional elastic problems. These programs are structured in the same way as `demo_cd` (see Section 6) except for some small differences. These differences are described in this chapter.

The variable `itype` defines the geometry of the element and is used in the function `crmesh`. Two types can be chosen. If `itype=1` quadrilateral elements are used and if `itype=20` regular triangular elements.

The material properties have to be given in these two-dimensional elastic problems. These properties are Young's Modulus (`E`), Poisson's ratio (`nu`). Additionally a parameter called `axi` has to be specified, which defines whether a problem can be characterized as 'plane strain' (`axi=0`), 'axi-symmetric' (`axi=1`) or 'plane stress' (`axi=2`).
Parameter `mat.mat(11)` defines the element topology (See: Section: 5.1.4).

An example on how this could be implemented in the demo files could be:

```
%%%%%%%%%% DEFINE THE MATERIAL PROPERTIES %%%%%%%%%%

E    =  1;    %   Young's modulus
nu   =  0.3;  %   Poisson's ratio
axi  =  2;    %  =0 plane strain
              %  =1 axi symmetric
              %  =2 plane stress
```

```
mat.mat=[E nu axi];

if itype<10,
   mat.mat(11)=norder+2;
else
   mat.mat(11)=itype-20+norder;
end

mat.types='ele';  % linear elastic element
```

8

Plot functions in the mlfem_nac toolbox

In the directory `pllib` several plot functions can be found. This chapter gives an overview of the different functions is given. The same information is given, when the help function of Matlab is used.

plbound

```
plots boundary of 2d mesh,
  h = plbound(coord,top,mat);
  h = plbound(coord,top,mat,colorstyle);
  input:
    coord             : nodal coordinates
    top               : element topology
    mat               : material parameters
    colorstyle ('y-') : style and color i.e. 'y-' (optional)
```

pldisp

```
plots original and deformed mesh
  h = pldisp(sol,coord,top,dest)
  h = pldisp(sol,coord,top,dest,mat,scale,original,idim)
  input:
    sol          : displacements, may be either in solution type form or as
                   nodal displacements
    coord        : nodal coordinates
    top          : element topology
    dest         : destination array
    mat          : material parameters (optional)
    scale        : scale factor (optional)
```

```
    original     : choice parameter, indicating whether the original mesh
                   must be drawn fully (2), boundary only (1) or not (0)
                   if (original==-1), only deformed boundary is shown
                   (optional)
    idim         : plot dimension choice parametes
                   idim=2: 2 d plot
                   idim=3: 3 d plot
                   (optional)
```

pldofnum

```
 plots the numbers of the dofs in the current meshplot
  h = pldofnum(coord,dest);
  input:
    coord        : nodal coordinates
    dest         : destination array
```

plelem

```
 plots elements in an existing mesh
  h = plelem(coord,top,mat)
  h = plelem(coord,top,mat,color,style)
  input:
    coord        : nodal coordinates
    top          : element topology
    mat          : material parameters (optional)
    color        : color of the nodes i.e. 'y' (optional)
    style        : style of the nodes i.e. 'o' (optional)
```

plelnum

```
 add element numbers of the elements represented by top
  h = plelnum(coord,top);
  input:
    coord        : nodal coordinates
    top          : element topology
```

plmesh

```
 plots mesh, essential boundary conditions, nodal forces
  h = plmesh(coord,top,mat)
  h = plmesh(coord,top,mat,bndcon,nodfrc)
  input:
    coord        : nodal coordinates
    top          : element topology
    mat          : material properties
    bndcon       : definition of essential boundary conditions (optional)
    nodfrc       : definition of nodal forces (optional)
```

plnodes

```
plots nodes
  h = plnodes(coord)
  h = plnodes(coord,color,style);
  input:
    coord        : nodal coordinates
    color        : color of the nodes i.e. 'y' (optional)
    style        : style of the nodes i.e. 'o' (optional)
```

plnodnum

```
 plots the numbers of the nodes in the current meshplot
  h = plnodnum(coord);
  input:
    coord        : nodal coordinates
```