MATFEM:
USER MANUAL

Ignacio Romero
Universidad Politécnica de Madrid
22 October 2014

Contents

1. **Writing analysis files** **1**
2. **Programming additional finite element models** **5**

`Matfem` *is a* `Matlab` *program designed to help learning about the finite element method and related numerical techniques. The most salient feature of the program is its simplicity. Being programmed in an interpreted language, the user can easily extend it or modify it, and test its effects immediately.*

The code currently solves two dimensional linear and nonlinear quasi-static problems. Regarding structural problems, plane trusses and beams are implemented. Other boundary value problems, such as the heat conduction and the advection-diffusion problem can be solved. Implementing the solution of additional problems is fairly straightforward, as it will be explained in this document.

The code has been used during the last ten years in post-graduate courses at the ETSI Caminos and ETSI Industriales of the Universidad Politécnica de Madrid. Usually, the student would work with an incomplete version of the code, and it would be his/her contribution to add the missing parts, learning at the same time the fundamental techniques of the finite element method (numerical quadrature rules, shape function calculation and their derivatives, computation of the residual vector and tangent matrix, etc).

At the current stage, the program is fairly limited regarding postprocessing. A very limited number of plotting functions are provided. At the same time, the user has access to the complete finite element function and thus he/she can postprocess it at will using some of the many plotting functions that `Matlab` *offers.*

This short manual describes how to write an input file for `Matfem` *as well as some details on how to implement new boundary value problems.*

Madrid, February 2011

1 Writing analysis files

Writing an analysis file for `Matfem` is relatively simple: a single `Matlab` file must be created describing the mesh and the analysis options. The most elemental analysis features are given at the very beginning of the file. Then the mesh is defined through its nodes and elements; the element types are described next and the boundary conditions are given at the end. The file concludes by calling the computational routines that will find the finite element solution.

The following input file describes a simple heat problem on a square. The square is depicted in '`fig-mesh`', in the format created by `Matfem`. The 2×2 square is meshed with four triangular elements with their labels indicated at their centers. The six nodes defined also have labels printed close to them.

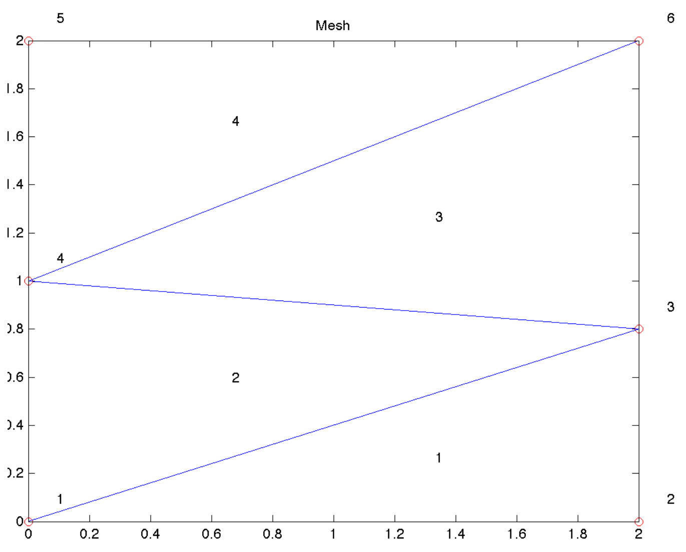

Figure 1: Mesh for heat problem.

The following mesh file defines the problem:

```
% April 1 1998
% last modification nov 2010
% Ignacio Romero

%-------------------------------------------------------
%           Global variable definitions. Do not modify
%
%       Kff: is the tangent matrix for the free dofs
%       U:   the nodal degrees of freedom
%       R:   the residual
```

```
%-------------------------------------------------------
global Kff U R;


%-------------------------------------------------------
%                   Problem properties
%-------------------------------------------------------
nen    = 3;    %nodes per element
ndofn  = 1;    %max dof per node
method = 1;    %solution strategy: 1-linear, 2-newton-raphson



%-------------------------------------------------------
%                    mesh definition
%-------------------------------------------------------
node = [ 0,   0; %1      %global coordinates of all nodes
         2,   0;
         2,   0.8;
         0,   1;
         0,   2;
         2,   2];


element = [1, 1,2,3; %1    %properties, nodei, nodej
           1, 1,3,4;
           1, 4,3,6;
           1, 5,4,6];



%extract information from mesh
% do not write anything in these 4 lines
nn    = size(node, 1);
nel   = size(element,1);
bc    = zeros(nn,ndofn);
force = zeros(nn,ndofn);


%-------------------------------------------------------
%       properties of each type of element
%-------------------------------------------------------
etype = [9,  3.0, 0.0];    % thermal triangle, conductivity, heat source


%-------------------------------------------------------
% boundary conditions
%-------------------------------------------------------

 % first indicate with a 1 if there is a boundary condition
 % '1' is just a label, it does not indicate the value of the fixed field
 % if the field is free, just don‘t write anything
 % bc means bc( nodelabel, doflabel)
  bc(1,1) = 1;  %node, dof=1 if constrained
  bc(2,1) = 1;  %node, dof=1 if constrained
  bc(5,1) = 1;
  bc(6,1) = 1;
 % then set the value. If the dof is constrained by the previous commands,
 % then the force is really the fixed value of the field.

  force(1,1) = 0.0;
  force(2,1) = 0.0;
  force(5,1) = 50.0;
  force(6,1) = 50.0;

 % external forces
 % if the dof is free, then the force is really a force, or a flux
 % force(3,1) = 33.0;


%mesh processing and solution strategy
meshplot(node, element);
sol = solveproblem(method,nen,ndofn,node,element,etype,bc,force);


%solution postprocessing
postprocess(node, element, ndofn,  bc, force, sol);
```

We explain next of the sections in the input file:

Global variable definition: The program starts with the `global` definition of certain important variables like the residual and the stiffness matrix. This section is not necessary *per se*, but it allows to retrieve these values from the program. Otherwise these variables will remain hidden to the user as the rest of variables employed in the finite element solution.

Problem definition: basic information about the problem to be solved needs to be defined beforehand so that memory can be set aside with the correct dimensions. At the moment, only three variables need to be defined, namely, the number of nodes per element (all the elements must be identical), the maximum number of degrees of freedom per node (in the case of the heat problem this is just 1, the temperature), and the method to be employed in the solution.

Mesh definition: The nodal coordinates must be input first in a matrix denoted always `node`. This matrix must have only two columns (corresponding to the (x, y) coordinates in a Cartesian basis). The nodes defined in this fashion have automatically a label assigned to each of them, corresponding to the row number in the `node` array. After this, the elements must be defined in an array named `element`, using one row per element. The first entry in each row indicates the *element family type*. The subsequent numbers refer to the *ordered* sequence of nodes of the element. The family is described next, and the sequence of nodes must be given counterclockwise. When mixing elements with different number of nodes, use trailing zeros for those elements with less number of nodes.

Element types: An array with name `etype` must be defined holding, in each row, the description of an element type. The first entry in each row selects the element from a list of internal types defined in `Matfem` (see the file `el_info.m`). These are:

Label	Description
1	Frame
2	Frame with rigid joints
3	Linear truss
4	Nonlinear spring
5	Nonlinear frame
6	Nonlinear truss
7	Currently not used
8	Nonlinear frame with fiber section model
9	Thermal element for any number of nodes per element
10	1d thermal element
11	Thermal triangle
12	Advection diffusion equation
13	Thermal quadrilateral
14	Linear elastic for plain strain
15	String

Table 1: Element type labels

After selecting the element label, the element properties should be given in the `etype` matrix. The number of properties depend on the element type and should be checked in the corresponding implementation.

Boundary conditions: Previously to defining the imposed values of the boundary conditions, the matrix `bc` must be modified. For each node and degree of freedom the corresponding value of `bc` must be set to `1` if the degree of freedom is constrained. After this, the matrix `force` might be modified. For each degree of freedom previously indicated to be constrained, the value in `force` is the imposed value (if not explicitly indicated, this value is set to 0). For any degree of freedom not indicted to be constrained, the value in `force` is the actual force (or, more generally, flux) to be introduced as data in the natural boundary conditions. Note that the entries of the matrix `force` have two different meanings, depending on the values in the matrix `bc`.

Launching the analysis: The previous steps completely define the analysis. The user might optionally select to plot the mesh with the `meshplot()` command. The problem is fully solved by calling the `solveproblem()` function. The variable `sol` is the only outcome of the finite element solution although the user might access internal variables by means of global variables as explained at the beginning. Finally, the command `postprocess()` plots the solution as depicted, for example, in '`fig-temperature`'

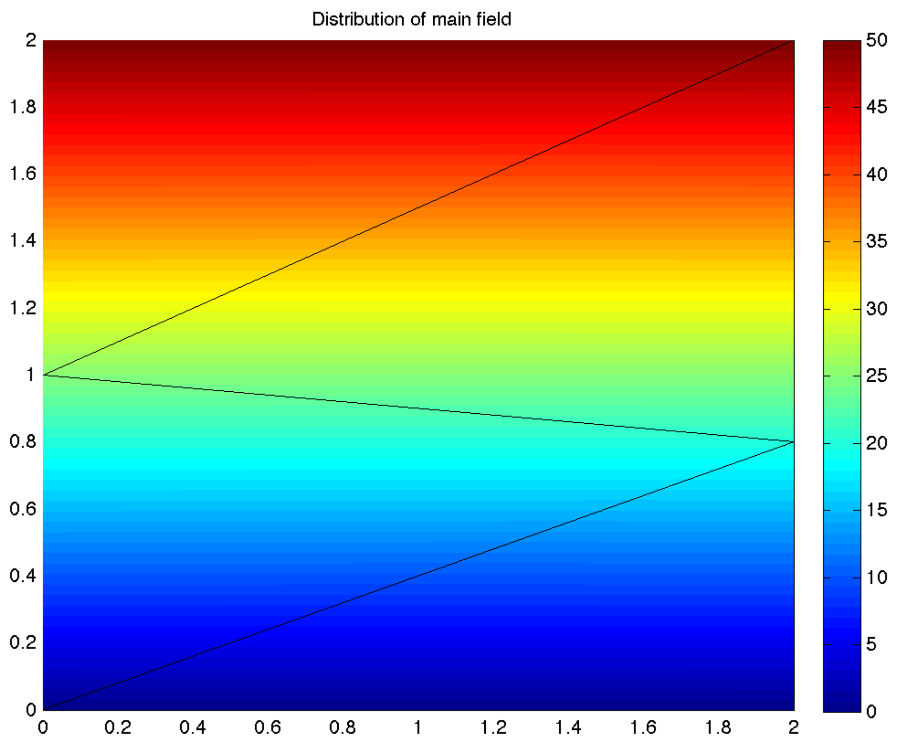

Figure 2: Temperature distribution for the heat problem.

2 Programming additional finite element models

The addition of a new finite element type in `Matfem` requires one modification in the file `el_info.m` and writing a new file that calculates the residual and tangent.

The file `el_info.m` serves as dictionary of the programmed element types. For each type, it declares its label (see '`tab-etypes`'), and it extracts the element degrees of freedom from the global list. This step is standard for all element types and the precise syntax can be deduced by looking at existing ones.

More importantly, for every new type a complete file must be written that indicates the element computations associated with the residual and tangent computation, as well as other minor tasks. For example, for the nonlinear spring the main computational function is as follows:

```
% EL_BARNG.M      2D geometrically nonlinear truss element
%
% etype:          3, ea
% Arguments:      task:   1 for stiffness calculation
%                         2 for initial load
%                         3 for internal displacements and forces
%                         4 for element internal force
%                 Uel:    nodal global displacement
%                 Ucel:   " in the last converged load step
%                 xy: nodal coordinates. Nodes are in rows
%                 elnum:  element number
%                 eldata: element material data
%
%   ret:   returned value...either a matrix or a vector,
%          depending on the task

function ret = el_barng(task,Uel,Ucel, xy, elnum, eldata)


%element initial data
EA     = eldata(2);                  %axial stiffness
N0     = eldata(3);                  %initial prestress
L0     = norm(xy(2,:)-xy(1,:));      %length at undeformed configuration

%geometic relations
node1 = xy(1,:) + Uel(1:2)';         %node1 at n+1 (k+1)
node2 = xy(2,:) + Uel(3:4)';         %node2 at n+1 (k+1)
L     = norm(node2-node1);           %length at current position
str   = L/L0-1;
N     = EA*str + N0;

% unit vector from node 2 to node 1
u = ( node2 - node1 ) / L;


%--------------------------------------------------------------------
%                   stiffness matrix
%--------------------------------------------------------------------
if task == 1     % computation of stiffness matrix

   kel = EA * u'*u + N/L * ( eye(2) - u'*u );
   ret = zeros(4,4);
   ret(1:2,1:2) =  kel;
   ret(1:2,3:4) = -kel;
   ret(3:4,1:2) = -kel;
   ret(3:4,3:4) =  kel;
```

```
%-------------------------------------------------------------------
%                initial force vector
%    (only valid for vertical dist. load in horizontal members
%-------------------------------------------------------------------
elseif task == 2
   ret = [0 0 0 0];


%------------------------------------------------------------------
%                  internal deformations and forces
%------------------------------------------------------------------
elseif task == 3
   i   = elnum;
   u   = a*Uel;
   S   = k*u;
   ret = [u' S']';


%--------------------------------------------------------------------
%                   element internal force
%--------------------------------------------------------------------
elseif task == 4,
  ret = zeros(4,1);
  ret(1:2) = N * (-u);
  ret(3:4) = N *   u;


%--------------------------------------------------------------------
%                   element energy
%--------------------------------------------------------------------
elseif task == 4,
  ret = 0.5 * EA * str * str;


end
```

The element information defined in the `etype` input array of the input file is received through the argument `eldata`. From this array the user can read information such as material constants, distributed forces, etc. The array `xy` contains the nodal reference coordinates. The matrix `Uel` collects the degrees of freedom of all the nodes in the element.

The switch flag `task` indicates what the function must compute. Irrespective of this task, the computed result must be written in the variable `ret` (which can be a vector or a matrix) which is the only result returned to the finite element program.

The main tasks are `isw=1, 3` and `5`. For the first task, the stiffness matrix must be calculated. For the second one, the internal force vector of the element must be written in the vector `ret` for the current state of the degrees of freedom given in `Uel`. Task `isw=5` corresponds to the computation of the energy functional which is minimized in elliptic problems.

Other tasks are rarely employed and are not described in this manual.