
MAE4700/5700

Finite Element Analysis for Mechanical and Aerospace Design

Cornell University, Fall 2009

Nicholas Zabararas

**Materials Process Design and Control Laboratory
Sibley School of Mechanical and Aerospace Engineering
101 Rhodes Hall
Cornell University
Ithaca, NY 14853-3801**

Problem definition

- The problem considered in the MATLAB software 1DBVP solves the following BVP:
 - Compute $u(x)$ in $[a,b]$ such that with appropriate boundary conditions, the following equation holds:

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u(x) = f(x), x \in [a,b]$$

- The **particular example provided** solves the following BVP:

$$-\frac{d^2u}{dx^2} + u(x) = (16\pi^2 + 1)\sin 4\pi x, x \in [0,1]$$

$$u(0) = 0$$

$$\frac{du}{dx}(1) = 4\pi \cos 4\pi$$

The analytical solution

is given as: $u(x) = \sin 4\pi x,$

$$\frac{du}{dx} = 4\pi \cos 4\pi x$$

Problem data: Functions p, q, f and $u_{ex}(x)$

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u(x) = f(x), x \in [a, b]$$

Function qq.m

```
function value = qq ( x )  
value = 1.0;
```

Function pp.m

```
function value = pp ( x )  
value = 1.0;
```

Function ff.m

```
function value = ff ( x )  
value = (16*pi^2 + 1)*sin(4*pi*x);
```

Function exact.m

```
function [u du] = exact ( x )  
u = sin ( 4 * pi * x );  
du = 4*pi*cos( 4 * pi * x );
```

This introduces the analytical solution $u(x)$ and its derivative du/dx ONLY if they are known.

Introducing the geometry and FE discretization

Function InputGrid.m

```
include_variables;

xl = 0.0;      % Introduce here the left location of the 1D domain
xr = 1.0;      % Introduce here the right location of the 1D domain

ElementType = 1; % Introduce element type
                % Here linear elements are used (two nodes per element)
                % If quadratic (3 nodes per element), enter 2, etc.

nel = 50;      % number of total elements that you want in your grid

MeshGenerator(xl,xr); % This function will generate the desirable finite element
                    % grid based on your input information (see next slide)
```

Mesh generator: MeshGenerator.m

```
if (ElementType==1)           % This part is for linear elements only!
    nno = nel+1;               % number of total nodes
    nen = 2;                   % number of nodes on each element

    Elems = zeros(nen,nel);    % initialize

    Nodes = linspace(xl,xr,nno); % set the coordinates of each global node
                                % (change this for different type of elements)

    for e = 1:nel               % loop over all elements,
        Elems(1,e) = e;         % set the global node number for each
        Elems(2,e) = e+1;       % local node of the (linear) element
    end
end

if (ElementType==2)
    ..... % This is left for you to program....
end

if (ElementType==3)
    .....
```

Other input data: InputData.m

```
ngp = ElementType+1; % number of Gauss points. The quadrature rule will integrate
% exactly all polynomials up to (2*ngp-1). You need to
% choose different number according to the element
% type and functions in your BVP.
```

```
plot_mesh = 'no'; % Change this information as appropriate
plot_node = 'no'; % You really don't need this in 1D ...
is_exact = 'yes'; % Do you have an exact solution?
```

% Define boundary conditions appropriate for your example.

```
BoundaryCondition(1).type = 1; % left boundary has an essential bound. condition
```

```
BoundaryCondition(1).value = 0;
```

```
BoundaryCondition(1).glbID = 1; % Global node 1 is the left boundary
```

```
BoundaryCondition(2).type = 2; % right boundary is a natural BC
```

```
BoundaryCondition(2).value = 4*pi*cos(4*pi);
```

```
BoundaryCondition(2).glbID = nno; % Global node nno is the right boundary
```

.....

$$u(0) = 0$$

$$\frac{du}{dx}(1) = 4\pi \cos 4\pi$$

Mixed boundary condition can also be considered of the form:

$$\frac{du}{dx} = \underbrace{\alpha}_{val(1)} u + \underbrace{\beta}_{val(2)}$$

Preprocessor.m

**% Preprocessing introduces input data, creating and discretizing the domain
% into finite elements (nodes, elements, etc.)**

```
InputGrid;           % Reads geometric information (problem dependent)
                    % and generates the mesh (nodes, elements, connectivities, ..)

neq = nno;           % number of equations

f = zeros(neq,1);   % initialize nodal force vector
d = zeros(neq,1);   % initialize nodal solution vector

I = zeros(nen*neq,1); % initialize the triplet for assembling the sparse
J = zeros(nen*neq,1); % matrix. We assume that the max number of
X = zeros(nen*neq,1); % nonzero entries in nen*neq

ntriplets = 0;

InputData;          % Here you need to add your problem dependent input
                    % data (Gauss quadrature, essential, natural or mixed
                    % boundary conditions, etc.)

PlotGrid;
```

Element basis functions and their derivatives

```
function [n,dndxi] = FiniteElement_1D(xi); % Note that they are defined at a given
                                         % Gauss point in natural coordinates xi
```

```
if ( ElementType == 1 )                % linear element
```

```
    n = [(1/2)*(1 - xi), (1/2)*(xi + 1)]; % Calculate the basis functions (row vector)
                                         % at the given integration point
```

```
    dndxi=[-1/2, 1/2 ];                % Calculate the derivative of the basis functions
                                         % at the given integration point (row vector)
```

```
end
```

```
if ( ElementType == 2 )                % quadratic element
```

```
    .....                               % You need to program this part
```

```
end
```

$$\frac{dN_1}{d\xi} = -\frac{1}{2}$$

$$\frac{dN_2}{d\xi} = \frac{1}{2}$$

$$N_1 = \frac{1}{2}(1 - \xi)$$

$$N_2 = \frac{1}{2}(1 + \xi)$$

Gauss integration: Gauss.m

```
function [gp,w] = gauss(ngp)
if (ngp == 1)    % 1 Gauss point
    gp = 0;
    w = 2;

elseif (ngp == 2) % 2 Gauss points
    gp(1) = - 0.577350269189625764509148780502;
    gp(2) =  0.577350269189625764509148780502;

    w(1) = 1.0;
    w(2) = 1.0;

elseif (ngp ==3) % 3 Gauss points

    gp(1) = - 0.774596669241483377035853079956;
    gp(2) =  0.0;
    gp(3) =  0.774596669241483377035853079956;

    w(1) = 5.0 / 9.0;
    w(2) = 8.0 / 9.0;
    w(3) = 5.0 / 9.0;

    ....
```

$$\int_{-1}^1 g(\xi) d\xi = \sum_1^{N_{\text{int points}}} g(\xi_i) W_i$$

$$\left. \begin{array}{l} \text{gp}(1) \\ \text{gp}(2) \end{array} \right\} \xi_i, i = 1, 2, \dots, N_{\text{int points}}$$

$$\left. \begin{array}{l} w(1) \\ w(2) \end{array} \right\} W_i, i = 1, 2, \dots, N_{\text{int points}}$$

Stiffness and load calculations at a given Gauss point ξ_i

Struct defined **at each Gauss point**

function [Ke,fe] = integrands(e, felm, Ke, fe);

.....

$$B = \text{felm.dN}; \left[\begin{array}{ccc} \frac{dN_1}{dx} & \frac{dN_2}{dx} & \dots \end{array} \right] \quad [N_1 \quad N_2 \quad \dots]$$

$$\text{felm.det JxW} = \frac{dx}{d\xi}(\xi_i)W_i$$

$$Ke = Ke + (\text{pp}(\text{felm.x}) * B' * B + \text{qq}(\text{felm.x}) * \text{felm.N}' * \text{felm.N}) * \text{felm.detJxW};$$

$$fe = fe + \text{ff}(\text{felm.x}) * \text{felm.N}' * \text{felm.detJxW};$$

.....

$$-\frac{d}{dx} \left(p(x) \frac{du}{dx} \right) + q(x)u(x) = f(x), x \in [a,b] \Rightarrow \sum_e \int_{\Omega^e} \left[p(x) \frac{du}{dx} \frac{dw}{dx} + q(x)u(x) \right] dx = \sum_e \int_{\Omega^e} f(x)w(x)dx$$

$$\left. \begin{array}{l} \frac{dw}{dx} = w^{eT} B^{eT} \\ \frac{du}{dx} = B^e u^e \end{array} \right\} \rightarrow \sum_e w^{eT} \int_{\Omega^e} \underbrace{\left[B^{eT} p(x) B^e + q(x) N^{eT} N^e \right]}_{K^e} dx u^e = \sum_e w^{eT} \underbrace{\int_{\Omega^e} N^{eT} f(x) dx}_{f^e}$$

Assemble process: Local to global DOF mapping

```
function AssembleGlobalMatrix(e,Ae,be);
for i = 1 : nen           % local row index
    glbi = Elems(i,e);    % global row index
    if ( be(i)~=0 )
        f(glbi) = f(glbi) + be(i); % assemble load
    end
    for j = 1 : nen       % local column index
        glbj = Elems(j,e); % global column index
        if ( Ae(i,j) ~= 0) % set up the list of triplets
            ntriplets = ntriplets + 1;
            len = length(X); % Manually increase the size of the I, J, X arrays if
                               % needed. All the values of these arrays between
            if ( ntriplets > len) % len and 2*len are set to zero and the length of
                I(2*len) = 0; J(2*len) = 0; X(2*len) = 0;
            end % these arrays is increased to 2*len
            I(ntriplets) = glbi; J(ntriplets) = glbj;
            X(ntriplets) = Ae(i,j); } ← Stiffness assembly
        end
    end
end
end
```

Assemble.m

```

ldof = nen*ndof;
[gp,w] = gauss(ngp);
for elmID = 1 : nel
    Ke = zeros(ldof);
    fe = zeros(ldof,1);
    glb = Elems(:,elmID);
    coord = Nodes(glb)';
    for i = 1:length(gp)
        [n,dndxi] = FiniteElement_1D(gp(i));
        felm.N = n;
        felm.x = n*coord;
        dxdxi = dndxi*coord;
        Jacc = dxdxi;
        felm.detJxW = Jacc * w(i);
        felm.dN = dndxi/Jacc;
        [Ke,fe] = integrands(elmID, felm, Ke, fe);
    end

    clear felm;
    AssembleGlobalMatrix(elmID,Ke,fe);
end

```

Nodal coordinates of element e

Loop over each element

$[N_1 \quad N_2 \quad \dots]$

Loop over each Gauss point

$felm.det JxW = \frac{dx}{d\xi}(\xi_i)W_i$

Note Jacobian changes in general changes from Gauss point to Gauss point

$x = [N^e] \{x^e\} \Rightarrow x(\xi_i)$

$\begin{bmatrix} \frac{dN_1}{dx} & \frac{dN_2}{dx} & \dots \end{bmatrix}$

felm is a struct containing information at a Gauss point

% Constructs the global stiffness matrix in sparse format using I, J, X

$K = \text{sparse}(I(1:ntriplets), J(1:ntriplets), X(1:ntriplets), neq, neq);$

% Assemble global matrix

Application of boundary conditions: ApplyBC.m

```
debc = []; ebcVals = [];           % Initialize debc, ebcVals
for i = 1 : 2                       % There are two boundary nodes for 1D BVPs
    if (BoundaryCondition(i).type == 1) % essential boundary condition
        debc = [debc BoundaryCondition(i).glbID];
        ebcVals = [ebcVals; BoundaryCondition(i).value];
    end
    if (BoundaryCondition(i).type == 2) % natural boundary condition
        glbID = BoundaryCondition(i).glbID;
        val = BoundaryCondition(i).value;

        if (i==1)
            val = -val;           % You need to change the sign in the flux on the 1st node
        end

        f(glbID) = f(glbID) + val; % Modify global force vector for nodes with natural BC
    end
end
....
```

Application of boundary conditions: ApplyBC.m

```

if (BoundaryCondition(i).type == 3)    % mixed boundary condition
    glbID = BoundaryCondition(i).glbID;

    val = BoundaryCondition(i).value; % for this boundary condition,
                                       % there are two values.

    x = Nodes(index);

    if ( i==1)
        val = -val;                    % left boundary
    end

    K(glbID,glbID) = K(glbID,glbID) - pp(x)*val(1); % modify the stiffness matrix
    f(glbID) = f(glbID) + pp(x)*val(2);           % modify the load vector
end
end

```

$$BC: \frac{du}{dx} = \underbrace{\alpha}_{val(1)} u + \underbrace{\beta}_{val(2)} \Rightarrow$$

Contribution to
the rhs of the
weak form:

$$p \frac{du}{dx} w \Big|_a^b = p(\alpha u + \beta) w \Big|_{x=b} - p(\alpha u + \beta) w \Big|_{x=a}$$

Nodal solution: NodalSoln.m

```
function [d, rf] = NodalSoln(K, R, debc, ebcVals)
```

```
% Computes the nodal solution and fluxes  
% (at points with essential BCs)
```

```
dof = length(R);
```

```
df = setdiff(1:dof, debc); % Separate the degrees-of-freedom to those we know  
% (essential BCs) and those we want to compute
```

```
Kf = K(df, df);
```

```
Rf = R(df) - K(df, debc)*ebcVals;
```

$$\begin{bmatrix} K_E & K_{EF} \\ K_{EF}^T & K_F \end{bmatrix} \begin{Bmatrix} \bar{d}_E \\ d_F \end{Bmatrix} = \begin{Bmatrix} f_E + r_E \\ f_F \end{Bmatrix}$$

$$K_F d_F = f_F - K_{EF}^T \bar{d}_E$$

```
dfVals = Kf\Rf;
```

```
d = zeros(dof,1);  
d(debc) = ebcVals;  
d(df) = dfVals;
```

```
% Restore the solution vector
```

$$r_E = K_E \bar{d}_E + K_{EF} d_F - f_E$$

```
rf = K(debc,:)*d - R(debc); % Calculate the reaction vector
```

Error estimation: ErrorAnalysis.m (here the L_2 norm is shown)

```

function ErrorAnalysis;
L2_Norm = 0.0;
L2_Norm_Normalize = 0;
[gp,w] = gauss(ngp);
for elmID = 1 : nel
    glb = Elems(:,elmID);
    coord = Nodes(glb)';
    de = d(glb);
    for i = 1:length(gp)
        [n,dndxi] = FiniteElement_1D(gp(i));
        x = n*coord;
        dx = n*dndxi;
        dxdxi = dndxi*coord;
        Jacc = dxdxi;
        detJxW = Jacc * w(i);
        dN = dndxi/Jacc;
        [u du] = exact(x);
        uh = n*de;
        duh = dN*de;
        L2_Norm = L2_Norm + (u - uh)^2*detJxW;;
        L2_Norm_Normalize = L2_Norm_Normalize + u^2*detJxW;
    end
end
L2_Norm = sqrt(L2_Norm) / sqrt (L2_Norm_Normalize);
    
```

$$\frac{\|u^h(x) - u^{exact}(x)\|_{L_2}}{\|u^{exact}(x)\|_{L_2}} = \frac{\left(\int_a^b (u^h(x) - u^{exact}(x))^2 dx \right)^{1/2}}{\left(\int_a^b (u^{exact}(x))^2 dx \right)^{1/2}}$$

$$\int_a^b (u^h(x) - u^{exact}(x))^2 dx = \sum_e \int_{\Omega^e} (u^h(x) - u^{exact}(x))^2 dx =$$

$$\sum_e \int_{\Omega^e} \sum_{i=1}^{N_{int}} (u^h(x(\xi_i)) - u^{exact}(x(\xi_i)))^2 \frac{dx}{d\xi} \Big|_{\xi=\xi_i} W_i$$

$$\frac{dx}{d\xi} \Big|_{\xi=\xi_i} W_i$$

det JxW

$$u^h(x(\xi_i)) = \begin{bmatrix} N_1^e & N_2^e & \dots \end{bmatrix} \{d^e\}$$

$$\frac{dx}{d\xi} \Big|_{\xi=\xi_i}$$

Main program: Main.m

```
preprocessor;           % Preprocessing

Assemble;              % Loop over all the elements and
                      % assemble the global sparse format of the
                      % stiffness matrix matrix

ApplyBC;               % Apply different boundary conditions

[d,r] = NodalSoln(K ,f, debc, ebcVals); % Solution of linear systems of equations

postprocessor;         % Postprocessor
```