Matlab Codes for 1D FEM Method

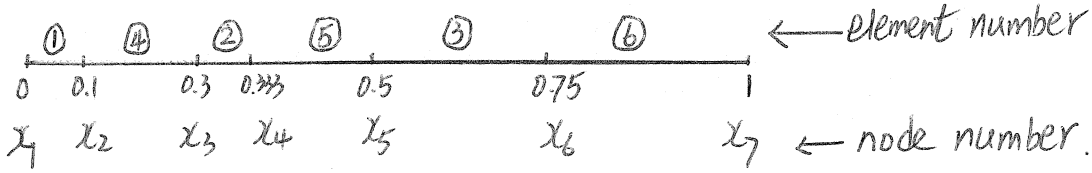
Matlab codes to solve the simple model problem

$$-u''(x) = f(x), \quad a < x < b, \quad u(a) = u_a, \quad u(b) = u_b \tag{1}$$

using the hat functions and assembling the stiffness matrix and the load vector element-by-element.

1 Step 1: Generate the mesh

Suppose that we use the following non-uniform mesh:



So the total number of nodal points is

$$N_d = 7,$$

and the total number of elements is

$$N_e = N_d - 1 = 6.$$

Store the coordinates of all nodal points in a vector of length N_d:

$$\mathbf{Node} = \begin{pmatrix} 0.0 \\ 0.1 \\ 0.3 \\ 0.333 \\ 0.5 \\ 0.75 \\ 1.0 \end{pmatrix}_{N_d \times 1}.$$

Store the numbers of nodes defining elements in a $N_e \times 2$ matrix (a 2D matrix):

$$\mathbf{Elem} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{pmatrix}_{N_e \times 2}.$$

Basically, the first entry of the j-th row of the matrix **Elem** is the number of the left-end node of Element j; similarly, the second entry of the j-th row of the matrix **Elem** is the number of the right-end node of the element.

Remark: Ordering of nodal points matters, but ordering of elements does NOT!

2 Step 2: Calculate local stiffness matrices/load vectors

For each element $E_j, j = 1, 2, \cdots, N_e$, do the following:

(a) Retrieve element information $E_j = [x_L, x_R]$, where

$$x_L = \mathbf{Node}(\mathbf{Elem}(j, 1)), \qquad x_R = \mathbf{Node}(\mathbf{Elem}(j, 2)).$$

(b) In an element $[x_L, x_R]$, there are two non-zero hat functions (Shape functions). One is

$$\phi_1(x) = \frac{x_R - x}{x_R - x_L}. \tag{2}$$

The Matlab code is the file **hat1.m**

```
function y = hat1(x,xL,xR)
% This function evaluates the hat function of the form
y = (xR-x)/(xR-xL);
return
```

The other one is

$$\phi_2(x) = \frac{x - x_L}{x_R - x_L}. \tag{3}$$

The Matlab code is the file **hat2.m**

```
function y = hat2(x,xL,xR)
% This function evaluates the hat function of the form
y = (x-xL)/(xR-xL);
return
```

(c) Calculate the 2×2 local stiffness matrix $\mathbf{A}^{(j)}$:

$$\begin{aligned}
\mathbf{A}^{(j)} &= \begin{bmatrix} \int_{E_j} \phi_1'(x)\phi_1'(x)\mathrm{d}x, & \int_{E_j} \phi_2'(x)\phi_1'(x)\mathrm{d}x \\ \int_{E_j} \phi_1'(x)\phi_2'(x)\mathrm{d}x, & \int_{E_j} \phi_2'(x)\phi_2'(x)\mathrm{d}x \end{bmatrix} \\
&= \begin{bmatrix} \int_{x_L}^{x_R} \phi_1'(x)\phi_1'(x)\mathrm{d}x, & \int_{x_L}^{x_R} \phi_2'(x)\phi_1'(x)\mathrm{d}x \\ \int_{x_L}^{x_R} \phi_1'(x)\phi_2'(x)\mathrm{d}x, & \int_{x_L}^{x_R} \phi_2'(x)\phi_2'(x)\mathrm{d}x \end{bmatrix} \\
&= \begin{bmatrix} 1/h, & -1/h \\ -1/h, & 1/h \end{bmatrix},
\end{aligned}$$

where

$$h = x_R - x_L.$$

(d) Calculate the length-2 local load vector $\mathbf{F}^{(j)}$:

$$\mathbf{F}^{(j)} = \begin{pmatrix} \int_{E_j} f(x)\phi_1(x)\mathrm{d}x \\ \int_{E_j} f(x)\phi_2(x)\mathrm{d}x \end{pmatrix} = \begin{pmatrix} \int_{x_L}^{x_R} f(x)\phi_1(x)\mathrm{d}x \\ \int_{x_L}^{x_R} f(x)\phi_2(x)\mathrm{d}x \end{pmatrix}.$$

Here, we use the Simpson rule to evaluate integrals involved. The Simpson rule is

$$\int_a^b g(x)dx \approx \frac{b-a}{6}\left[g(a) + 4g\left(\frac{a+b}{2}\right) + g(b)\right].$$

Therefore, we have for $j = 1, 2$

$$\int_{x_L}^{x_R} f(x)\phi_j(x)dx = \frac{x_R - x_L}{6}\left[f(x_L)\phi_j(x_L) + 4f(x_M)\phi_j(x_M) + f(x_R)\phi_j(x_R)\right],$$

where x_M is the middle point of the element $[x_L, x_R]$, namely, $x_M = \dfrac{x_L + x_R}{2}$.

The Matlab code to evaluate $\int_{x_L}^{x_R} f(x)\phi_1(x)\mathrm{d}x$ is the file **int_hat1_f.m**:

```
function y = int_hat1_f(xL,xR)

% This function evaluate \int_{xL}^{xR} f(x)*\phi(x) dx,
% where \phi(x) = (x-xL)/(xR-xL), using the Simpson rule.

xM = 0.5*(xL+xR);
y  = (xR-xL)/6.0*(f(xL)*hat1(xL,xL,xR) + 4*f(xM)*hat1(xM,xL,xR)...
                  + f(xR)*hat1(xR,xL,xR));
return
```

The Matlab code to evaluate $\int_{x_L}^{x_R} f(x)\phi_2\mathrm{d}x$ is the file **int_hat2_f.m**:

```
function y = int_hat2_f(xL,xR)

% This function evaluate \int_{xL}^{xR} f(x)*\phi(x) dx,
% where \phi(x) = (xR-x)/(xR-xL), using the Simpson rule.

xM = 0.5*(xL+xR);
y  = (xR-xL)/6.0*(f(xL)*hat2(xL,xL,xR) + 4*f(xM)*hat2(xM,xL,xR)...
                  + f(xR)*hat2(xR,xL,xR));
return
```

Note: All local stiffness matrices are stored in a $N_e \times 2 \times 2$ matrix **LA** (a 3D matrix), and all local load vectors are stored in a $N_e \times 2$ matrix **LF**!

3 Assemble global stiffness matrix A and global load vector F

Symbolically, we write

$$\mathbf{A} = \sum_{j=1}^{N_e} \mathbf{A}^{(j)}, \qquad \mathbf{F} = \sum_{j=1}^{N_e} \mathbf{F}^{(j)}.$$

How to actually do it:

(a) Initialize **A** and **F**:

```
A = zeros(ND, ND);
F = zeros(ND, 1);
```

(b) For each element $E_j, j = 1, 2, \cdots, N_e$, do the following:

(1) Retrieve element information: n_L, the nodal number of the left-end point of E_j and n_R, the nodal number of the right-end point of E_j by

$$n_L = \mathbf{Elem}(j, 1), \qquad n_R = \mathbf{Elem}(j, 2).$$

(2) Add $\mathbf{A}^{(j)}(1,1)$ to $\mathbf{A}(n_L, n_L)$;
add $\mathbf{A}^{(j)}(1,2)$ to $\mathbf{A}(n_L, n_R)$;
add $\mathbf{A}^{(j)}(2,1)$ to $\mathbf{A}(n_R, n_L)$;
add $\mathbf{A}^{(j)}(2,2)$ to $\mathbf{A}(n_R, n_R)$;
add $\mathbf{F}^{(j)}(1)$ to $\mathbf{F}(n_L)$; and
add $\mathbf{F}^{(j)}(2)$ to $\mathbf{F}(n_R)$.

4 Impose boundary conditions: $u(a) = u_a, u(b) = u_b$

Before the imposition of the BCs, the system is

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,N_d} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,N_d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N_d-1,1} & a_{N_d-1,2} & a_{N_d-1,3} & \cdots & a_{N_d-1,N_d} \\ a_{N_d,1} & a_{N_d,2} & a_{N_d,3} & \cdots & a_{N_d,N_d} \end{bmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N_d-1} \\ U_{N_d} \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_{N_d-1} \\ F_{N_d} \end{pmatrix}.$$

The boundary conditions basically require that

$$U_1 = u_a, \qquad U_{N_d} = u_b,$$

which can be achieved by modifying the stiffness matrix and the load vector as follows:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N_d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N_d-1,1} & a_{N_d-1,2} & a_{N_d-1,3} & \cdots & a_{N_d-1,N_d} \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N_d-1} \\ U_{N_d} \end{pmatrix} = \begin{pmatrix} u_a \\ F_2 \\ \vdots \\ F_{N_d-1} \\ u_b \end{pmatrix}.$$

5 Solve the system AU = F by a direct method

In Matlab, you only need one line to do so (backslash!):

```
U=A\F;
```

6 Post-processing: plotting, error analysis, etc

See samples.

7 Main routine

```
%     Program EBE_FEM_POISSON_1D.m

      function U = EBE_FEM_POISSON_1D(ND,NE,Node,Elem,ua,ub)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
%               Matlab code for 1D FEM for                              %
%                                                                       %
%                -u''=f(x), a<=x<=b, u(a)=ua, u(b)=ub                   %
%                                                                       %
%     Input:  ND - Number of total nodal points                         %
%             NE - Number of total elements                             %
%             Node - Nodal points                                       %
%             Elem - Element information                                %
%             ua - Dirichlet BC at x=a                                  %
%             ub - Dirichlet BC at x=b                                  %
%     Output: U - FEM solution at nodal points                          %
%                                                                       %
%     Function needed: f(x)                                             %
%                                                                       %
%     Matlab functions used:                                            %
%                                                                       %
%     hat1(x,xL,xR): hat function in [xL,xR] that is 1                  %
%     at xL, and 0 at xR.                                               %
%                                                                       %
%     hat2(x,xL,xR): hat function in [xL,xR] that is 0                  %
%     at xL, and 1 at xR.                                               %
%                                                                       %
%     int_hat1_f(xL,xR): Contribution to the load vector                %
%     from hat1                                                         %
%                                                                       %
%     int_hat2_f(xL,xR): Contribution to the load vector                %
%     from hat 2                                                        %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      format long;

%Step 2: Calculate local stiffness matrices/load vectors

      LA = zeros(NE,2,2);   % initialize a 3D matrix "LA"
      LF = zeros(NE,2);

      for j = 1:NE
         xL = Node(Elem(j,1));
         xR = Node(Elem(j,2));
         h  = xR - xL;

         LA(j,1,1) = 1.0/h;
         LA(j,1,2) =-1.0/h;
         LA(j,2,1) =-1.0/h;
         LA(j,2,2) = 1.0/h;

         LF(j,1) = int_hat1_f(xL,xR);
         LF(j,2) = int_hat2_f(xL,xR);
      end
```

```
%Step 3: Assemble global stiffness matrix A and global load vector F

      A = zeros(ND,ND);
      F = zeros(ND,1);

      for j = 1:NE
         nL = Elem(j,1);
         nR = Elem(j,2);

         A(nL,nL) = A(nL,nL) + LA(j,1,1);
         A(nL,nR) = A(nL,nR) + LA(j,1,2);
         A(nR,nL) = A(nR,nL) + LA(j,2,1);
         A(nR,nR) = A(nR,nR) + LA(j,2,2);

         F(nL) = F(nL) + LF(j,1);
         F(nR) = F(nR) + LF(j,2);
      end


%Step 4: Impose boundary conditions

      for j = 1:ND
          A(1,j)  = 0.0;
          A(ND,j) = 0.0;
      end
      A(1,1)   = 1.0;
      A(ND,ND) = 1.0;

      F(1)  = ua;
      F(ND) = ub;


%Step 5: Solve the equation by a direct method

      U = A\F;

      return
```

8 Define $f(x)$

The Matlab code to define $f(x)$ is the file **f.m**

```
function y = f(x)
y = x^3;
return
```

9 Test example

We test the Matlab code with

$$f(x) = x^3, \quad a = 0, \quad b = 1, \quad u_a = u_b = 0.$$

The exact solution is

$$u(x) = -\frac{x^5}{20} + \frac{x}{20}.$$

First we tested it using the afore-mentioned non-uniform grid and the drive code is **driver1A.m**

```
%     Program driver1A.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                           %
%     Driver code for the problem:                          %
%              -u'' = f(x)    0 < x < 1                     %
%              u(0) = 0                                     %
%              u(1) = 0                                     %
%     where f(x) = x^3.                                     %
%                                                           %
%     In this case, the exact solution is                   %
%              u(x) = -x^5/20+x/20                          %
%                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      clear all            % clear variables and functions from memory
      close all            % close figures

      format long;         % double-precision


%Step 1: Generate the mesh (manually)

      ND = 7;              % number of total nodes
      NE = ND-1;           % number of total elements

      % Save all node coordinates in the vector "Node"
      Node = zeros(ND,1);  % initialize "Node"
      Node(1) = 0.0;
      Node(2) = 0.1;
      Node(3) = 0.3;
      Node(4) = 0.333;
      Node(5) = 0.5;
      Node(6) = 0.75;
      Node(7) = 1.0;

      % Save all element information in the 2D matrix "Elem"
      Elem = zeros(NE,2);  % initialize "Elem"
      Elem(1,1) = 1;
      Elem(1,2) = 2;
      Elem(2,1) = 3;
      Elem(2,2) = 4;
      Elem(3,1) = 5;
      Elem(3,2) = 6;
      Elem(4,1) = 2;
      Elem(4,2) = 3;
      Elem(5,1) = 4;
      Elem(5,2) = 5;
      Elem(6,1) = 6;
      Elem(6,2) = 7;

      % Specify Dirichlet boundary conditions:
      ua = 0.0;
      ub = 0.0;
```

```
%Steps 2-5: FEM analysis (call EBE_FEM_POISSON_1D)

      uAppr = zeros(ND,1);
      uAppr = EBE_FEM_POISSON_1D(ND,NE,Node,Elem,ua,ub);


%Step 6: Post-processing, say, plot the numerical solution

      figure(1)
      plot(Node, uAppr, 'r-o', 'LineWidth', 2);
      h_legend=legend('Numerical solution',2);
      set(h_legend,'FontName','Times', 'FontSize',18, 'FontAngle','normal');
      xlabel('{\sl{x}}', 'FontName','Times','FontSize', 18);
      ylabel('{\sl{u}}', 'FontName','Times','FontSize', 18);
      hold off

%
%-------------------------------------------------------------
%
% Step 7: Error analysis (if exact solution is available)

      x = 0:0.01:1.0;      % create vector: x=(0,0.01,0.02,0.03,.....,1)
      K = length(x);       % find length of vector "x"
      for i = 1:K
         uExact(i) = -x(i)^5/20.0+x(i)/20.0;
      end

      for j = 1:ND
         utmp = -Node(j)^5/20.0+Node(j)/20.0
         error(j) = uAppr(j)-utmp;
      end

      figure(3)
      plot(Node, uAppr, 'r-.o', 'LineWidth', 2);
      hold on
      plot(x, uExact, 'b-', 'LineWidth', 2);
      h_legend=legend('Numerical solution','Exact solution',2);
      set(h_legend,'FontName','Times', 'FontSize',18, 'FontAngle','normal');
      xlabel('{\sl{x}}', 'FontName','Times','FontSize', 18);
      ylabel('{\sl{u}}', 'FontName','Times','FontSize', 18);
      hold off

      figure(5)
      plot(Node, error, 'r-o', 'LineWidth', 2);
      xlabel('{\sl{x}}', 'FontName','Times','FontSize', 18);
      ylabel('{\sl{Error}}', 'FontName','Times','FontSize', 18);
      hold off
```

Then we tested it using the uniform Cartesian grid with $N_d = 7$ and $N_d = 20$, respectively, and the drive code is similar to driver1A.m. The only change we need is to replace the corresponding segment of code in driver1A.m by the following.

```
%Step 1: Generate the mesh (uniform mesh)

      ND = 20;                % number of total nodes
      NE = ND-1;              % number of total elements

      % Save all node coordinates in the vector "Node"
      Node = zeros(ND,1);  % initialize "Node"
      h = 1.0/NE;
      for j = 1:ND
         Node(j) = (j-1)*h;
      end

      % Save all element information in the 2D matrix "Elem"
      Elem = zeros(NE,2);  % initialize "Elem"
      for j = 1:NE
         Elem(j,1) = j;
         Elem(j,2) = j+1;
      end
```

10 Results

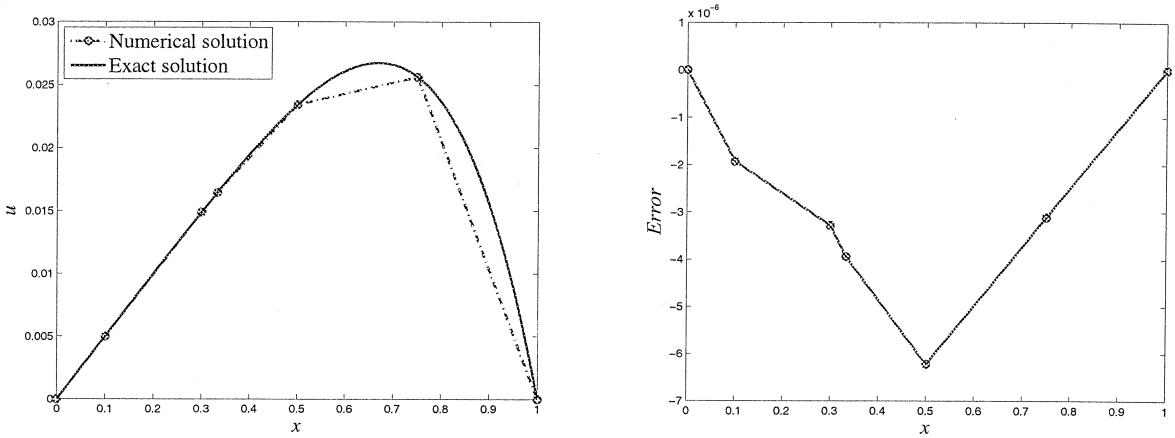


Figure 1: FEM simulation results using the non-uniform grid. (a) Solution plots; (b) Error plot at nodal points.

Note: For this model problem: $-u''=f(x)$, the FEM solution should be exact at nodal points if all integrals are evaluated exactly. However, in sample codes, we use Simpson's Rule to evaluate $\int_{x_L}^{x_R} f(x)\phi_j(x)\,dx$. For this example, $f(x)=x^3$ so $f(x)\phi_j(x)$ has a degree of 4, whose integral evaluted by Simpson's Rule can not be exact! That is why we still have 10^{-6} error at nodal points.

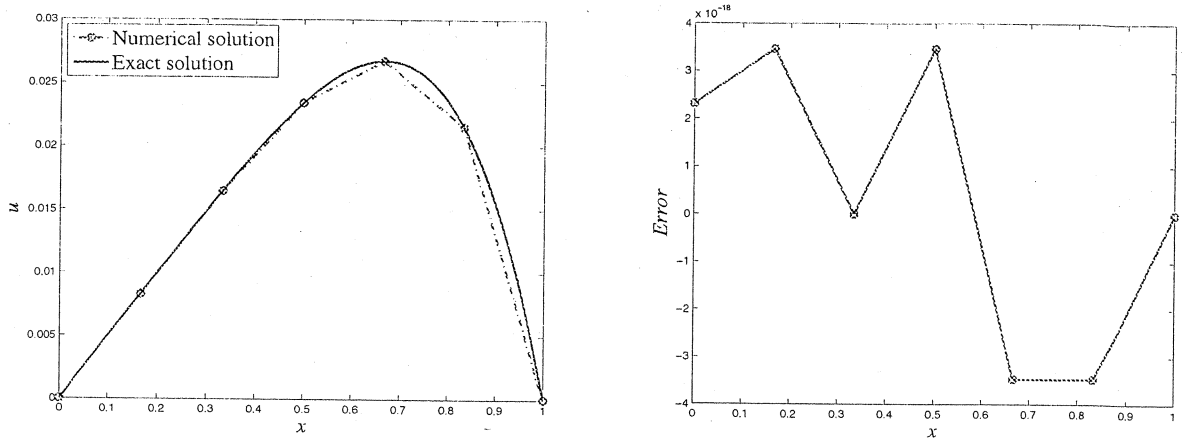


Figure 2: FEM simulation results using the uniform grid with $N_d = 7$. (a) Solution plots; (b) Error plot at nodal points.

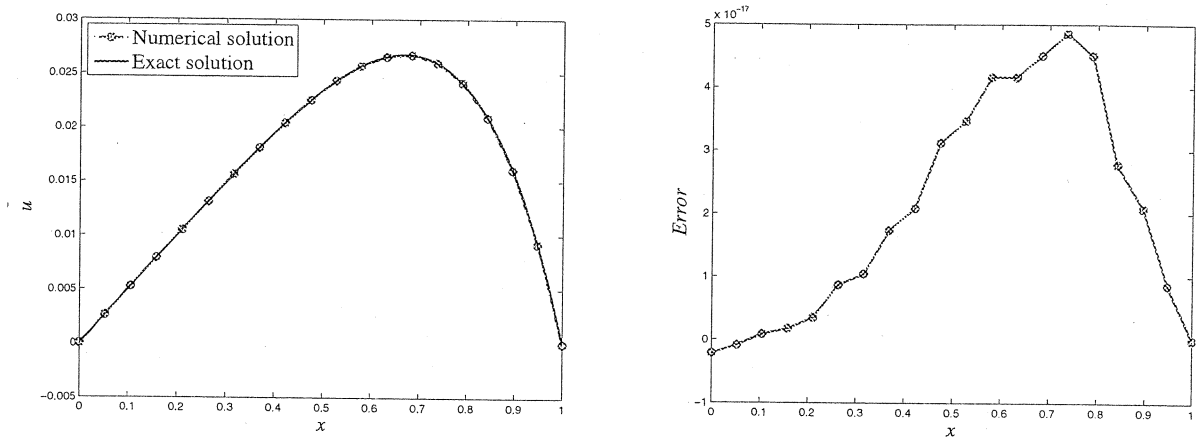


Figure 3: FEM simulation results using the uniform grid with $N_d = 20$. (a) Solution plots; (b) Error plot at nodal points.

Note: For Figure 2(b), why FEM solution appears to be exact at nodal points even though $\int_{x_L}^{x_R} f \phi_i(x)\, dx$ are still evaluated by Simpson's rule?
My guess is that, using uniform grids, there happens to have some "error cancellation"!