FINITE ELEMENT ANALYSIS

PROF: P.M. MOHITE

PROJECT 1: 1D BAR

PROJECT 2: 2D BEAM

BY,

J. SIRISHA (ALONE)

231010031

PROJECT- 1

Introduction

Objective: To find the solution of governing equation for the bar where it is constrained at both the ends A, B and traction load ($T(x)$) applied on the bar (1D)

To Develop a generic Finite element Code for the Governing Equation of bar with out using inbuilt functions to yield displacements.

Methodology. [Question 1-a]

Mathematical Formulation

- The governing Equation of the 1-D elastic bar considered is

Case 1: $AE \frac{d^2u}{dx^2} - 2u = T(x)$ where $T(x) = 4x^2 - 2x - 4$ (Quadratic)

$T(x)$ ↓ Traction force or Source term.

Stiffness Variation $AE(x)$;

Case 2: $T(x) = \text{Constant}$

Boundary Condition

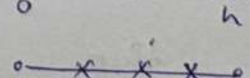
$$\left.\begin{aligned} u &= 0, \quad \text{at } x = 0 \\ u \frac{du}{dx} &= -1, \quad \text{at } x = 1 \end{aligned}\right\} \text{Dirichlet B.C}$$

$$\left.\begin{aligned} u &= 0 \quad \text{at } x = 0 \\ \frac{du}{dx} &= -3 \quad \text{at } x = 1 \end{aligned}\right\} \text{Neumann B.C}$$

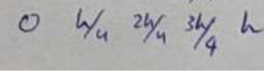
Finite Element Modeling

- Discretization of the domain into Elements.

For element type (Linear) - $e = 2$
DOF = 1



$e = 4$



Element type - Quadratic $e = 2$
O/F = 2

$0 \quad h/4 \quad 2h/4 \quad 3h/4 \quad h$

Trick pattern Found

$$\underbrace{S}_{\text{gives no. of nodes}} = (\underbrace{O/F}_{\text{Order of polynomial}} \times \overset{\text{no. of Elements.}}{e}) + 1$$

for i = 1 : S
location(i) = $(i-1)h/_{s-1}$ } gives the location of node.
end

Shape functions

Lagrange Interpolation Implementation

Consider
Element type - Quadratic [3 nodes]
x_1, x_2, x_3.

Formula of Lagrange ⇒ for shape function at node 1 ⇒ $N_1 \Rightarrow \dfrac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}$

Shape function at node 2 $N_2 = \dfrac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}$

$N_3 = \dfrac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$

By substituting the values of $x_1, x_2, x_3 \rightarrow -1, 0, 1$
we get, $N_1 = \text{phi\_e}(1) = \dfrac{x(x-1)}{2}$

$N_2 = \text{phi\_e}(2) = -(x-1)(x+1)$

$N_3 = \text{phi\_e}(3) = \dfrac{x(x+1)}{2}$

For finding $\left(\dfrac{dN}{dx}\right)$, Trick Pattern found ⇒ Numerator is separately found and denominator as

$\boxed{df = li/den}$ (refer code)

then $\dfrac{dN_1}{dx} = \text{dshap}(1) = x - \dfrac{1}{2}$; $\dfrac{dN_2}{dx} = \text{dshap}(2) = -2x$; $\dfrac{dN_3}{dx} = \text{dshap}(3) = x + \dfrac{1}{2}$

Hierachic shape functions

Element type - Linear $\Rightarrow N_1 \Rightarrow phi(1) = 0.5(1-\xi)$

$N_2 = phi(2) = 0.5(1+\xi)$

Quadratic $\Rightarrow$ $N_1 \rightarrow phi(1) = \dfrac{\xi(\xi-1)}{2}$

$N_2 \Rightarrow phi(2) = \dfrac{-(\xi-1)}{\xi+1}$

$N_3 \Rightarrow phi(3) = \dfrac{\xi(\xi+1)}{2}$

For finding differentiation, directly manually entering.

Linear $\Rightarrow \dfrac{dN_1}{dx} \Rightarrow dphi(1) = -0.5$

$\dfrac{dN_2}{dx} \rightarrow dphi(2) = 0.5;$

Quadratic $\Rightarrow \dfrac{dN_1}{dx} \Rightarrow dphi(1) = \xi - 0.5;$

$\dfrac{dN_2}{dx} \rightarrow dphi(2) = -2\xi;$

$\dfrac{dN_3}{dx} = dphi(3) = \xi + 0.5;$

Implementation in MATLAB

- Using Lagrange interpolation & Hierachic shape functions,

$$I = \int_{-1}^{1} f(\xi)\,d\xi \approx \sum_{i=1}^{N} w_i\, f(\xi_i).$$

So, Using Gauss Quadrature, we get the values of ξ_i & w_i

∴ We manually input the values such as,

n==1 (Linear) ; n=2 (Quadratic)

psi(1)=0;
w(1)=2;

psi(1) = $\sqrt{3}/3$;
psi(2) = $-\sqrt{3}/3$; so, on... till n=5
w(1)=1;
w(2)=1;

Thus phie & dshap are substituted in stiffness matrix and Source term which also involves w(i) & psi(i). We find Global stiffness matrix, source term.

$$\underline{\text{Stiffness term}} \Rightarrow \int_0^h \left(a \frac{dshap(1)}{dx} \frac{dshap(2)}{dx} - c \; phie(1) \cdot phie\_(2) \right) dx$$

$\approx$ Sum. [a * (diff-phie values(1)) * (diff-phie-values(2)) (2/h) − c * phie_values(1) * phie-values(2) * (h/2)]

Similarly for force term.

These steps are same for Lagrange Interpolation and Hierarchic shape method

Results and Discussion

Patch Test Results

① FEM solution and Exact solution of Dirichlet Boundary Conditions and Neumann Boundary Conditions have been plotted

[X , displacements]
length of bar   at each point

② Error in solution of FEM has been plotted (both Lagrange & Hierachic)

Error for FEM solution is calculated as,

$$\boxed{\text{Error} = \text{abs}(\text{FEM\_solution} - \text{displacement vector})}$$

//

Strain Energy Analysis

① Lagrange Interpolation

We know $U = \int_{\Omega} \frac{1}{2} E \epsilon^2 dx$.

$E \rightarrow$ Youngs Modulus.

$\epsilon = \frac{du}{dx}$ → displacement.

ϵ → Strain

For our problem, $u(x) = -2x^2 + x$.

$$\frac{du}{dx} = -4x + 1$$

$$\epsilon^2 \Rightarrow (-4x+1)^2 \Rightarrow \boxed{U_{Exact} = \int_{\Omega} \frac{1}{2} E (-4x+1)^2 + cu^2 \, dx}$$

Using FEM

$$U(x) \approx \sum_{i=1}^{n} N_i(x) \cdot d_i$$

N_i → shape function, d_i → displacement

$$\epsilon \approx \sum_{i=1}^{n} \frac{dN_i}{dx} \cdot d_i$$

$$\epsilon^2 \Rightarrow \left(\sum_{i=1}^{n} \frac{dN_i}{dx} d_i \right)^2$$

$$U_{FEM} = \sum_{i=1}^{N\cdot elems} \int_{(k_i + 1)\text{elements}}^{k_i} \left[\frac{1}{2} E \left(\sum_{i=1}^{n} \frac{dN_i}{dx} d_i \right)^2 + c(d_i)^2 \right] dx$$

Using Gauss Quadrature, local strains = Sum[(dshap .* displacements) + c.di]

$$\boxed{U_{FEM} = \frac{1}{2} * E * (\text{Local strains})^2 * w(i) * h}$$

- Strain Energy is calculated over the range of Elements (1 to 10) and observed the Convergence.

- In Lagrange & Hierarchic shape functions, Only shape functions Calculations are different.

[Quadratic]

Lagrange $\rightarrow$ $N_1(x) \Rightarrow \dfrac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}$ $\quad N_3(x) \Rightarrow \dfrac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$

$N_2(x) \Rightarrow \dfrac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}$

Hierarchic $\rightarrow$ $N_1(x) = 0.5\,\xi(\xi-1)$

$N_2(x) = 1-\xi^2$

$N_3(x) = \frac{1}{2}\xi(\xi+1)$

Error Analysis & Convergence Rate

$$\boxed{\text{Error} = \text{abs}(\text{Strain\_Energy\_FEM} - \text{Strain\_Energy\_Exact})}$$

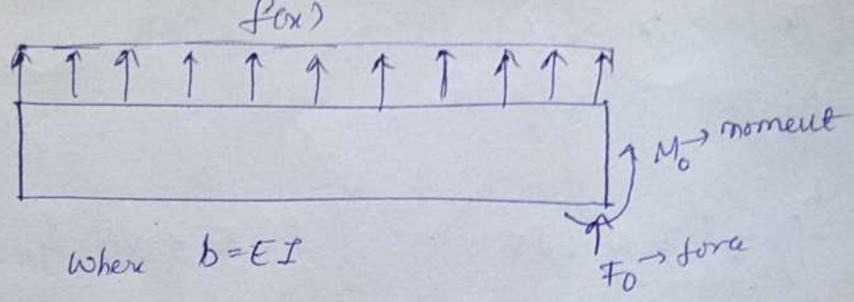
Strain Energy Exact = 1.6608
Strain Energy FEM = 1.2708
} for our problem.

Error of strain Energy at each element from [1 : 10] has been plotted.
Error between FEM & Exact strain Energy.

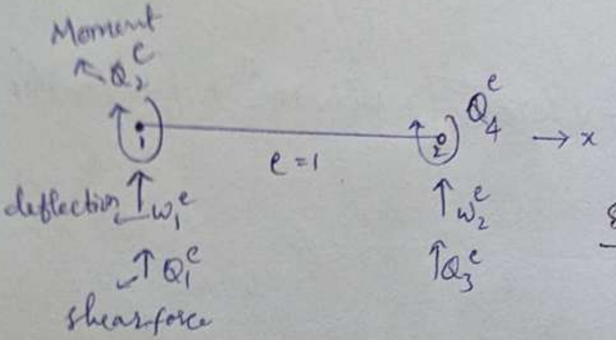
BEAM BENDING PROBLEM

(1) Governing Equation

$$\frac{d^2}{dx^2}\left(b\,\frac{d^2 w}{dx^2}\right) + f(x)$$



Where $b = EI$



Sign convention $w \uparrow$ +ve

shear force $Q_1 \uparrow$ +ve

Moment Q_2 ↻ +ve

(2) Development of a weak form

$$\int_0^{h_e} v \left[\frac{d^2}{dx^2}\left(b\,\frac{d^2 w}{dx^2}\right) - f\right] dx = 0$$

weight function

(Performing Integration by parts) (2 times)

$$\int_0^{h_e} \left[-\frac{dv}{dx}\left[b\,\frac{d^2 w}{dx^2}\right]' - vf\right] dx + \left[v + \frac{d}{dx}\left(b\,\frac{d^2 w}{dx^2}\right)\right]_0^{h_e} = 0.$$

$$\boxed{\int_0^{h_e} \left[\frac{d^2 v}{dx^2}\; b\,\frac{d^2 w}{dx^2} - vf\right] dx + \Big[\underbrace{v\,(b\cdot w'')' - b w'' v'}_{\text{boundary terms}}\Big]_0^{h_e} = 0}$$

Weak form of the governing Equation

Where v determines $\rightarrow w$

$-v'$ determines $-w' \Rightarrow -\frac{dw}{dx}$ slope

} Primary variables replacing with v & v'

- Coefficient of primary variables helps determining secondary variables.

$(bw'')' \rightarrow$ Coefficient of v (represents shear-force)

$bw'' \rightarrow$ coefficient of v' (represents moment).

So, we have four boundary conditions

v, $-v'$, shear force and moments.

w , $-dw/dx$ } Called generalized displacements

$(bw'')'$, (bw'') } Called generalised forces.

$\bar{x} = x - x_{1e \to node}$

By Sub in Weak form,

$$\underbrace{\int_0^{h_e} v'' \cdot b \cdot w'' \, d\bar{x}}_{\text{Bilinear term } B(v,w)} = \int_0^{h_e} f v \, d\bar{x} + v(0) Q_1^e + \left[-v'(0) Q_2^e\right] + v(h_e) Q_3^e + \left[-v'(h_e) Q_4^e\right]$$

$$Q_1^e = (bw'')'\Big|_0 \qquad Q_3^e = -(bw'')'\Big|_{h_e}$$

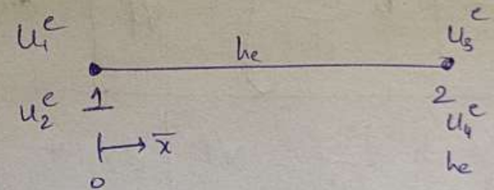
$$Q_2^e = (bw'')\Big|_0 \qquad Q_4^e = -(bw'')\Big|_{h_e}$$

$$I^e(w) = \int_0^{h_e} \frac{1}{2} b (w'')^2 \, dx - \int_0^{h_e} w f \, dx - w(0) Q_1^e - \left[-w'(0)\right] Q_2^e - w(h_e) Q_3^e - \left[-w'(h_e)\right] Q_4^e$$

(3) <u>Interpolating functions</u>.

u_1, u_3 - Translational.

$u_2, u_4 \to$ Rotational.



Four D.O.F for 1 element

$w(0) = w_1 = U_1^e \to$ generalized notation for DOF

$-w'(0) = \theta_1 = U_2^e$ } 2 DOF at node 1

slope

$w(h_e) = w_2 = U_3^e$

$w'(h_e) = \theta_2 = U_4^e$ } 2 DOF at node 2

$$w^e(x) = \sum_{j=1} U_j^e \underbrace{\phi_j^e(x)}_{\text{Approximate function to be found}} \Rightarrow U_1^e \phi_1^e(\bar{x}) + U_2^e \phi_2^e(\bar{x}) + U_3^e \phi_3^e(\bar{x}) + U_4^e (\phi_4^e)(\bar{x})$$

$w^e(0) = U_1$, $-w^{e\prime}(0) = U_2$; $w^e(h_e) = U_3$; $w^{e\prime}(h_e) = U_4$

$$\begin{bmatrix} K_{11}^e & K_{12}^e & K_{13}^e & K_{14}^e \\ K_{21}^e & K_{22}^e & K_{23}^e & K_{24}^e \\ K_{31}^e & K_{32}^e & K_{33}^e & K_{34}^e \\ K_{41}^e & K_{42}^e & K_{43}^e & K_{44}^e \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{Bmatrix} = \begin{Bmatrix} f_1^e + Q_1^e \\ f_2^e + Q_2^e \\ f_3^e + Q_3^e \\ f_4^e + Q_4^e \end{Bmatrix}$$

$U_1 \to w$ at 0; $U_2 \to w'$ at 0; $U_3 \to w$ at h_e; $U_4 \to w'$ at h_e

f^e: due to distributed load; Q^e: due to Point load or moments

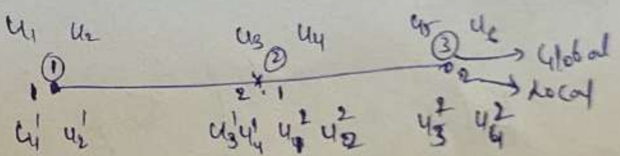
After substitution,

$$[K] = \frac{2b}{h^3} \begin{bmatrix} 6 & -3h & -6 & -3h \\ -3h & 2h^2 & 3h & h^2 \\ -6 & 3h & 6 & 3h \\ -3h & h^2 & 3h & 2h^2 \end{bmatrix} \quad ; \{f\} = \frac{fh}{12} \begin{Bmatrix} 6 \\ -h \\ 6 \\ -h \end{Bmatrix}$$

due to symmetry, 1, 3 are same, so as 2 & 4 row.

④ Assembly

$$\left.\begin{aligned} U_3^1 &= u_1^2 = U_3 \\ U_4^1 &= u_2^2 = U_4 \end{aligned}\right\} \text{symmetry.}$$



Replacing, $u_1^1 \to U_1$; $u_2^1 \to U_2$; $u_3^2 \to U_5$; $U_4^2 = U_6$

Force balance $\quad f_3^1 + f_1^2 = f_3$; $f_4^1 + f_2^2 = f_4$

$Q_3^1 + Q_1^2 = Q_3$; $Q_4^1 + Q_2^2 = Q_4$

For Element 1,

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 & K_{14}^1 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 & K_{24}^1 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 & K_{34}^1 \\ K_{41}^1 & K_{42}^1 & K_{43}^1 & K_{44}^1 \end{bmatrix} \begin{Bmatrix} u_1^1 \\ u_2^1 \\ u_3^1 \\ u_4^1 \end{Bmatrix} = \begin{Bmatrix} f_1^1 \\ f_2^1 \\ f_3^1 \\ f_4^1 \end{Bmatrix} + \begin{Bmatrix} Q_1^1 \\ Q_2^1 \\ Q_3^1 \\ Q_4^1 \end{Bmatrix}$$

$u_1^1 \to U_1$; $u_2^1 \to U_2$

Add eq^n 3 & 5

Add eq^n 4 & 6

For element 2

$$\begin{bmatrix} K_{11}^2 & K_{12}^2 & K_{13}^2 & K_{14}^2 \\ K_{21}^2 & K_{22}^2 & K_{23}^2 & K_{24}^2 \\ K_{31}^2 & K_{32}^2 & K_{33}^2 & K_{34}^2 \\ K_{41}^2 & K_{42}^2 & K_{43}^2 & K_{44}^2 \end{bmatrix} \begin{Bmatrix} u_1^2 \\ u_2^2 \\ u_3^2 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} f_1^2 \\ f_2^2 \\ f_3^2 \\ f_4^2 \end{Bmatrix} + \begin{Bmatrix} Q_1^2 \\ Q_2^2 \\ Q_3^2 \\ Q_4^2 \end{Bmatrix}$$

$$
\begin{bmatrix}
k_{11}^1 & k_{12}^1 & k_{13}^1 & k_{14}^1 & 0 & 0 \\
k_{21}^1 & k_{22}^1 & k_{23}^1 & k_{24}^1 & 0 & 0 \\
k_{31}^1 & k_{32}^1 & k_{33}^1 + k_{11}^2 & k_{34}^1 + k_{12}^2 & k_{13}^2 & k_{14}^2 \\
k_{41}^1 & k_{42}^1 & k_{43}^1 + k_{21}^2 & k_{44}^1 + k_{22}^2 & k_{23}^2 & k_{24}^2 \\
0 & 0 & k_{31}^2 & k_{32}^2 & k_{33}^2 & k_{34}^2 \\
0 & 0 & k_{41}^2 & k_{42}^2 & k_{43}^2 & k_{44}^2
\end{bmatrix}
\begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{Bmatrix}
=
\begin{Bmatrix} f_1 \\ f_2 \\ f_3 \to f_3^1 + f_1^2 \\ f_4 \to f_4^1 + f_2^2 \\ f_5 \\ f_6 \end{Bmatrix}
+
\begin{Bmatrix} Q_1 \\ Q_2 \\ Q_3 \to Q_3^1 + Q_1^2 \\ Q_4 \to Q_4^1 + Q_2^2 \\ Q_5 \\ Q_6 \end{Bmatrix}
$$

Assembled matrix is also symmetric.

⑤ Applying Boundary Conditions

In our case,

At $x = 0$

$$w = w' = 0$$

At $x = L$

$$Q_5 = F_0 \ ; \ Q_6 = -M_0$$

$$Q_3^1 + Q_1^2 = 0$$

$$Q_4^1 + Q_2^2 = 0$$

$$U_1 = U_2 = 0 .$$

∴ we get

$$
\begin{Bmatrix} U_3 \\ U_4 \\ U_5 \\ U_6 \end{Bmatrix}
= \frac{h}{6EI}
\begin{Bmatrix}
5h^2 F_0 + 3h M_0 + \frac{17}{4} f_0 h^3 \\
-9h F_0 - 6M_0 - 7 f_0 h^2 \\
16h^2 F_0 + 12h M_0 + 12 f_0 h^3 \\
-12h F_0 - 12 M_0 - 8 f_0 h^2
\end{Bmatrix}
$$

$$
\therefore \ w(x) = \begin{cases} U_3 \phi_3^1 + U_4 \phi_4^1 \ , & 0 \le x \le h \\ U_3 \phi_1^2 + U_4 \phi_2^2 + U_5 \phi_3^2 + U_6 \phi_4^2 \ , & h \le x \le 2h \end{cases}
$$

$$M = EI \frac{d^2 w}{dx^2} = EI \sum_{j=1}^{4} U_j^I \frac{d^2 \phi_j^I}{dx^2}$$

Bending stress →

$$\sigma_x = \mp \frac{MH}{2I} = \mp \frac{EH}{2} \sum_{j=1}^{4} U_j^I \frac{d^2 \phi_j^I}{dx^2}$$

$H \to$ Height of the beam

Bending Moment and Shear Force Calculation

B.M $\Rightarrow$ $\boxed{M = - EI \dfrac{d^2w}{dx^2}}$

$\dfrac{d^2w}{dx^2}$ is solved using Central difference.

i.e, $\dfrac{d^2w}{dx^2} \Rightarrow$ displacements $\left(2(n+1)\right)$ - 2*displacements $(2n)$

+ displacements $(2(n+1)$

Similarly for shear force,

$$\frac{dw}{dx} = \frac{\text{displacements}(2(i+1)) - \text{displacements}(2(i-1))}{2^* el}$$

$$V = -EI \frac{dw}{dx} / el \Rightarrow \boxed{V = \frac{dM}{dx}}$$

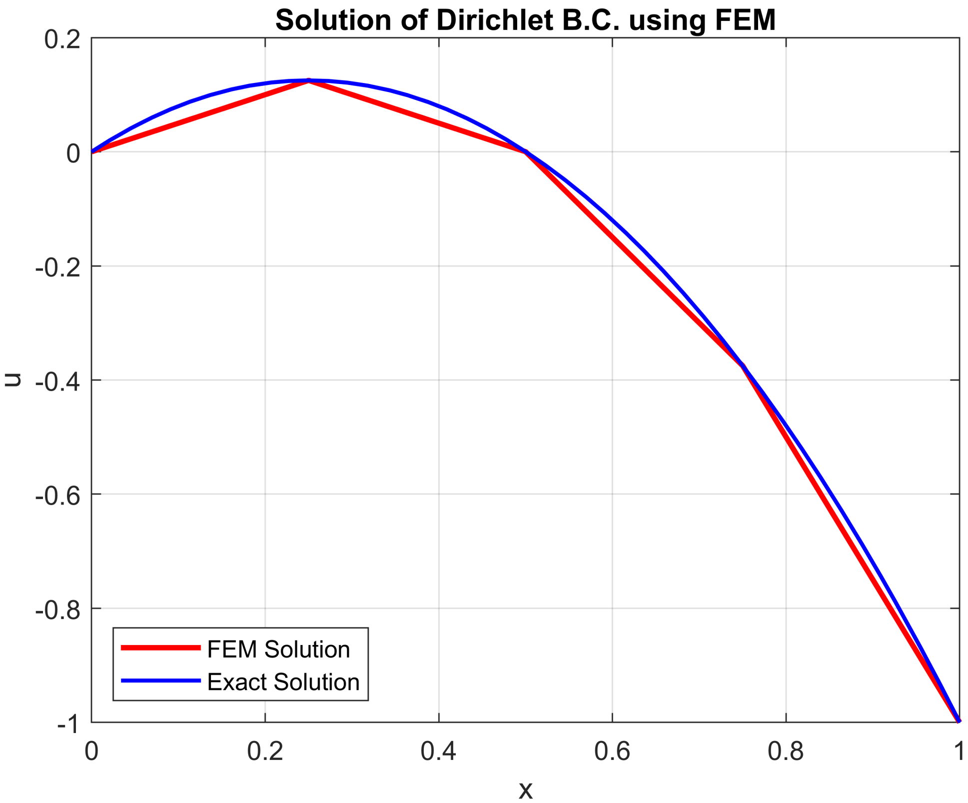
```
clc;
clear;
syms x;

% Input parameters
OIF = 2; % order of polynomial
e = 2; % number of elemnets
n = OIF + 1; % number of nodes
s = (OIF * e) + 1; % Global matrix size pattern

% governing_eqn = du^2/dx^2-2*u = T(x)
T = 4*x^2 - 2*x - 4;  % Source term
T_x = sym(zeros(s, 1));

exact_sol_T = -2*x^2 + x;


% Assign numerical values
a_num = 1;
c_num = -2;
x0 = 0;
xn = 1;
h = (xn - x0) / e;

GP = n; % Number of Guass points

[w, psi] = Guass(GP);
phi_e_values = zeros(GP, n);
diff_phi_e_values = zeros(GP, n);

for i = 1:GP
    psi_values = psi(i);
    [n, phi_e, dshap] = ShapeFunctions(OIF, psi_values);

    phi_e_values(i, :) = phi_e';
    diff_phi_e_values(i, :) = dshap';
end

% w
% psi
%
phi_e_values;
diff_phi_e_values;

% storing KNM and CNM values
K_Local = zeros(n, n);
C_Local = zeros(n, n);

% Assemble local matrices using Gauss quadrature
for i = 1:n
    for j = 1:n
        K_Local(i, j) = 0;
```

```
        C_Local(i, j) = 0;
        for k = 1:GP
            % Define symbolic expressions for F_xi and CNM_integrand
            F_xi = a_num * diff_phi_e_values(k, i) * diff_phi_e_values(k, j)*(2/h) - c_num * ph
            CNM_integrand = -phi_e_values(k, i) * phi_e_values(k, j)*(h/2);

            % Evaluate the expressions with numerical values
            F_xi_eval = double(F_xi);
            CNM_integrand_eval = double(CNM_integrand);

            % Compute the contribution using Gauss quadrature
            F_xi_contribution = F_xi_eval * w(k);
            CNM_contribution = CNM_integrand_eval * w(k);

            % Update the local stiffness matrix and CNM matrix
            K_Local(i, j) = K_Local(i, j) + F_xi_contribution;
            C_Local(i, j) = C_Local(i, j) + CNM_contribution;
        end
    end
end

% K_Local
% C_Local

% Calculation of global stiffness matrix
global_stiffness_size = s;
s = (OIF * e) + 1;

K_global = zeros(global_stiffness_size);
C_global = zeros(global_stiffness_size);

% Loop through each element
for element = 1:n-1:s-1
    % Assemble local stiffness matrix into global stiffness matrix
    start_index = element;
    end_index = element+(n-1);
    K_global(start_index:end_index, start_index:end_index) = K_global(start_index:end_index, st
    C_global(start_index:end_index, start_index:end_index) = C_global(start_index:end_index, st
end

% K_global
% C_global

X = (x0:h/OIF:xn)'; % % Defining the nodal points along the domain

for i = 1:s
    T_x(i) = subs(T, x, X(i));
end


% Dirichlet Boundary Conditions
% Boundary condition 1
C = C_global * T_x;
du_dx = -3;
```

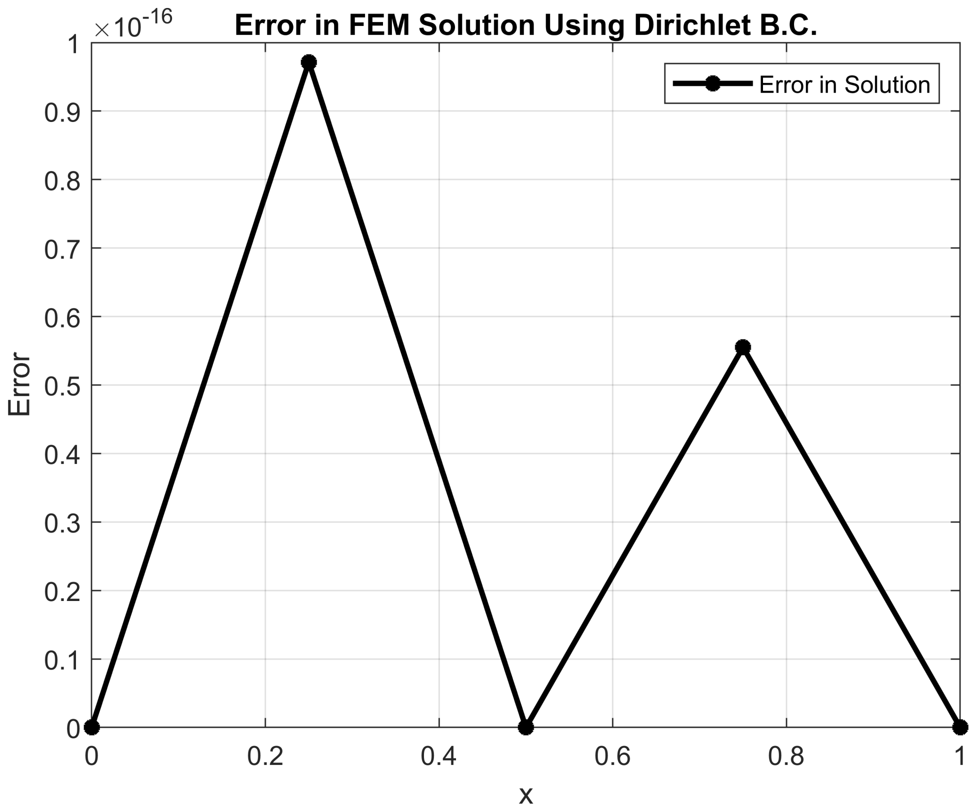
```
K_global_Dirichlet = K_global;
g_x_Dirichlet = sym(zeros(s, 1));
GNM_Dirichlet = a_num * g_x_Dirichlet * du_dx;
RHS_Dirichlet = GNM_Dirichlet + C;

u0 = x0;
un = -xn;
K_global_Dirichlet(1, :) = [1, zeros(1, s-1)];
K_global_Dirichlet(end, :) = [zeros(1, s-1), 1];
RHS_Dirichlet(1) = u0; % RHS of equation
RHS_Dirichlet(end) = un; % RHS of equation


% finding displacements using dirichet boundar conditions

u_D_bc = inv(K_global_Dirichlet)*  RHS_Dirichlet;
displacement_vector = double(u_D_bc)
```

```
displacement_vector = 5×1
         0
    0.1250
         0
   -0.3750
   -1.0000
```

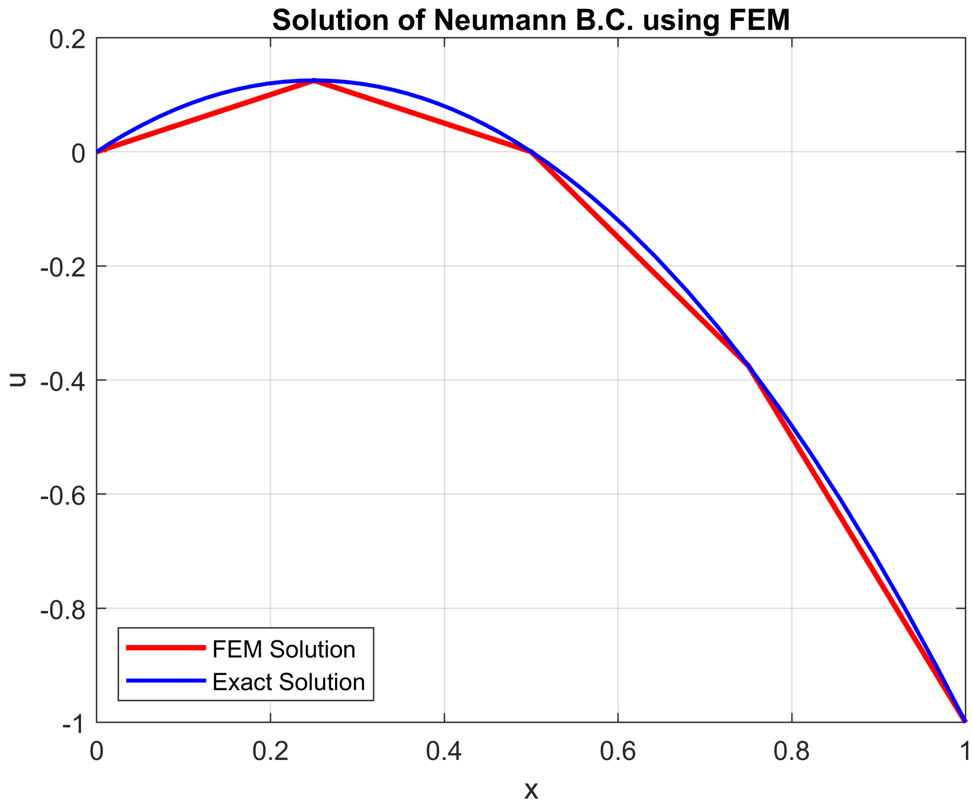
```
exact_values = double(subs(exact_sol_T, x, X)); % Evaluate exact solution at node positions
errors = abs(exact_values - displacement_vector); % Calculate absolute error at each node

% Plot the FEM Solution and the actual solution
figure;
plot(X, displacement_vector, "r", "LineWidth", 2, "DisplayName", "FEM Solution");
hold on;
fplot(exact_sol_T, [0 1], "b", "LineWidth", 1.5, "DisplayName", "Exact Solution");
xlabel("x");
ylabel("u");
title("Solution of Dirichlet B.C. using FEM");
legend('Location', 'southwest');
grid on;
```




```
figure;
plot(X, errors, 'k-*', 'LineWidth', 2, 'DisplayName', 'Error in Solution');
xlabel('x');
ylabel('Error');
title('Error in FEM Solution Using Dirichlet B.C.');
legend show;
grid on;
```




```
% Neumann boundary Conditions
K_global_neumann = K_global;
g_x_neumann = sym(zeros(s, 1));
g_x_neumann(end)=1;

GNM_neumann= a_num * g_x_neumann * du_dx;
RHS_neumann = GNM_neumann+C;

% Boundary Conditions 2

K_global_neumann(1,:)= [1,zeros(1, s-1)];
RHS_neumann(1) = u0;
u_neumann = inv(K_global_neumann)* RHS_neumann;
displacement_vector_n = double(u_neumann);

%Plot the FEM Solution and the actual solution
figure;
plot(X, displacement_vector_n, "r", "LineWidth", 2, "DisplayName", "FEM Solution");
hold on;
fplot(x,exact_sol_T, [0 1], "b", "LineWidth", 1.5, "DisplayName", "Exact Solution");
xlabel("x");
ylabel("u");
title("Solution of Neumann B.C. using FEM");
legend('Location', 'southwest');
grid on;
```




```
function [w, psi] = Guass(p_bar)
    psi = zeros(p_bar, 1);
    w = zeros(p_bar, 1);

    if p_bar == 1
        psi(1) = 0; % for p_bar = 1
        w(1) = 2;
    elseif p_bar == 2
        psi(1) = sqrt(3)/3;
        psi(2) = -sqrt(3)/3;
        w(1) = 1;
        w(2) = 1;
    elseif p_bar == 3
        psi(1) = sqrt(15)/5;
        psi(2) = -sqrt(15)/5;
        psi(3) = 0;
        w(1) = 5/9;
        w(2) = 5/9;
        w(3) = 8/9;
    elseif p_bar == 4
        psi(1) = sqrt((15 - 2*sqrt(30))/35);
        psi(2) = -sqrt((15 - 2*sqrt(30))/35);
        psi(3) = (sqrt((14*sqrt(30))/5 + 21))/7;
        psi(4) = -(sqrt((14*sqrt(30))/5 + 21))/7; % for p_bar = 7
```

```
        w(1) = 1/2 + sqrt(30)/36;
        w(2) = 1/2 + sqrt(30)/36;
        w(3) = 1/2 - sqrt(30)/36;
        w(4) = 1/2 - sqrt(30)/36;
    elseif p_bar == 5
        psi(1) = 0.906179845938664;
        psi(2) = -0.906179845938664;
        psi(3) = 0.538469310105683;
        psi(4) = -0.538469310105683;
        psi(5) = 0;
        w(1) = 0.236926885056189;
        w(2) = 0.236926885056189;
        w(3) = 0.478628670499366;
        w(4) = 0.478628670499366;
        w(5) = 0.568888888888889;
    end
end

function [n, phi_e, dshap] = ShapeFunctions(OIF, x_loc)
    syms x;
    n = OIF + 1;
    x_local = sym(zeros(1, n));

    for i = 1:n
        x_local(i) = -1 + (i-1)*(2/OIF); % Calculate the elements of the vector
    end

    x_s = sym('x', [1, n]);
    N = sym(zeros(size(x_s)));
    df = sym(zeros(size(x_s))); % Initialize df

    for j = 1:n
        Li = 0;
        den = 1;
        product_term = 1;
        for i = 1:n
            if j ~= i
                term = 1; % Initialize product term for index j
                for k = 1:n
                    if k ~= i && k ~= j
                        term = term * (x - x_s(k));
                    end
                end
                % Add the product term for index j to Lagrange polynomial for index i
                Li = Li + term;
            end
%% creating pattern for differentiation
            if i ~= j
                product_term = product_term * (x - x_s(i)) / (x_s(j) - x_s(i));
                den = den * (x_s(j) - x_s(i));
            end
        end

        N(j) = product_term;
```

```
        df(j) = Li / den;
    end

    phi_e = subs(N, x_s, x_local);
    dshap = subs(df, x_s, x_local);

    % Evaluate the shape functions and their derivatives at the Gauss points
    xi_loc = x_loc;
    phi_e = double(subs(phi_e, x, xi_loc));
    dshap = double(subs(dshap, x, xi_loc));
end
```

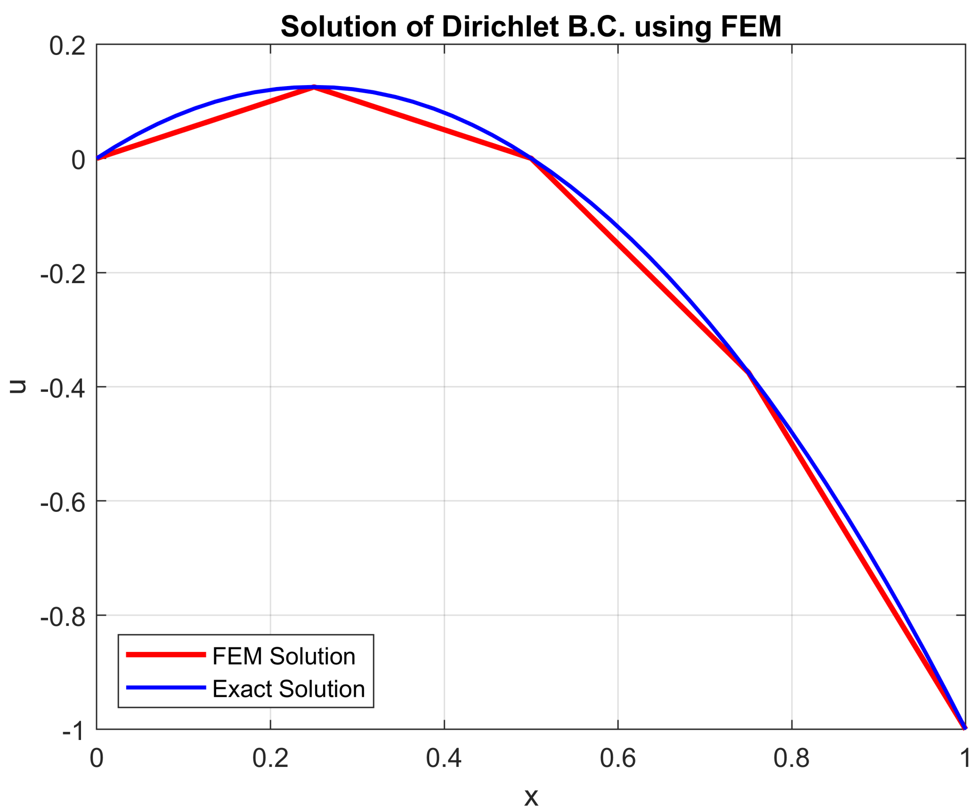
```
clc;
clear;
syms x;

% Input parameters
OIF = 2; % order of polynomial
e = 2; % number of elements
n = OIF + 1; % number of nodes
s = (OIF * e) + 1; % Global matrix size

T = 4*x^2 - 2*x - 4;  % Source term
T_x = sym(zeros(s, 1));

exact_sol_T = -2*x^2 + x;



% Domain definitions
a_num = 1;
c_num = -2;
x0 = 0;
xn = 1;
h = (xn - x0) / e;

% Number of Gauss points
GP = n;

% Initialize matrices for values and derivatives
phi_e_values = zeros(GP, n);
diff_phi_e_values = zeros(GP, n);

% Calculate Gauss points and weights manually
[w, psi] = Gauss(GP);

% Define hierarchical shape functions and their derivatives
for i = 1:GP
    psi_values = psi(i);
    [phi_e, dphi_e] = HierarchicalShapeFunctions(n, psi_values);
    phi_e_values(i, :) = phi_e;
    diff_phi_e_values(i, :) = dphi_e;
end



% Initialize local matrices
K_Local = zeros(n, n);
C_Local = zeros(n, n);

% Compute local matrices using numerical integration
for i = 1:n
    for j = 1:n
        for k = 1:GP
            xi = psi(k);
```

```
            F_xi = a_num * diff_phi_e_values(k, i) * diff_phi_e_values(k, j) * (2/h) - c_num *
            CNM_integrand = -phi_e_values(k, i) * phi_e_values(k, j) * (h/2);

            % Integrate using Gaussian quadrature weights
            K_Local(i, j) = K_Local(i, j) + F_xi * w(k);
            C_Local(i, j) = C_Local(i, j) + CNM_integrand * w(k);
        end
    end
end

% Calculation of global stiffness matrix
global_stiffness_size = s;
K_global = zeros(global_stiffness_size);
C_global = zeros(global_stiffness_size);

% Loop through each element
for element = 1:e
    start_index = 1 + (element-1)*OIF;
    end_index = start_index + n - 1;
    K_global(start_index:end_index, start_index:end_index) = K_global(start_index:end_index, st
    C_global(start_index:end_index, start_index:end_index) = C_global(start_index:end_index, st
end

% Define the nodal points along the domain
X = (x0:h/OIF:xn)';

% Define the source term and calculate its values at nodes
T_x = subs(T, x, X);

exact_values = double(subs(exact_sol_T, x, X)); % Evaluate exact solution at node positions


% Dirichlet Boundary Conditions
C = C_global * T_x;
du_dx = -3;

K_global_Dirichlet = K_global;
g_x_Dirichlet = sym(zeros(s, 1));
GNM_Dirichlet = a_num * g_x_Dirichlet * du_dx;
RHS_Dirichlet = GNM_Dirichlet + C;

u0 = x0;
un = -xn;
K_global_Dirichlet(1, :) = [1, zeros(1, s-1)];
K_global_Dirichlet(end, :) = [zeros(1, s-1), 1];
RHS_Dirichlet(1) = u0;
RHS_Dirichlet(end) = un;

% Solving for displacements using Dirichlet boundary conditions
u_D_bc = inv(K_global_Dirichlet) * RHS_Dirichlet;
displacement_vector = double(u_D_bc)
```

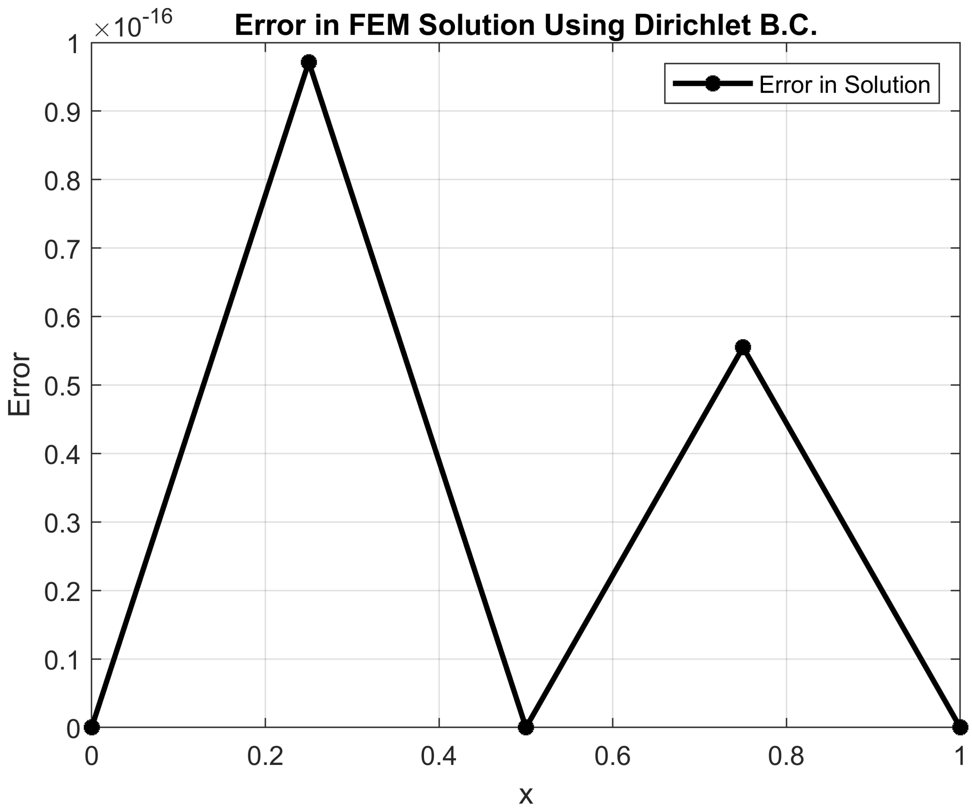
```
displacement_vector = 5×1
         0
```

```
    0.1250
         0
   -0.3750
   -1.0000
```

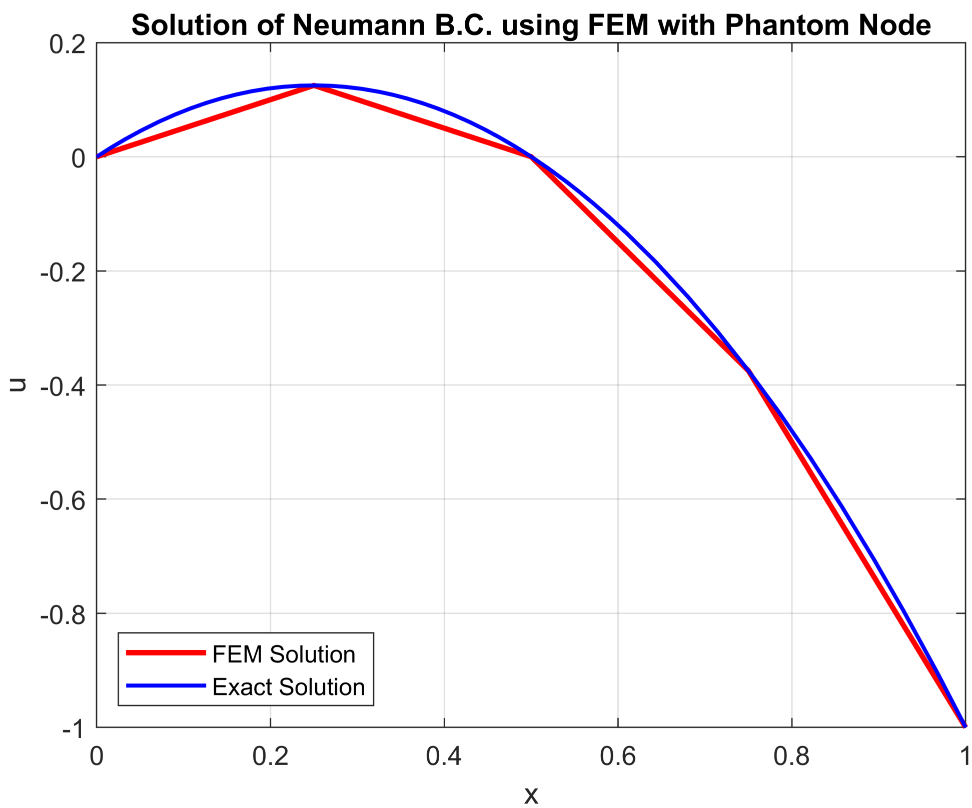
```
errors = abs(exact_values - displacement_vector); % Calculate absolute error at each node
% Plot the FEM Solution and the actual solution
figure;
plot(X, displacement_vector, "r", "LineWidth", 2, "DisplayName", "FEM Solution");
hold on;
fplot(exact_sol_T, [0 1], "b", "LineWidth", 1.5, "DisplayName", "Exact Solution");
xlabel("x");
ylabel("u");
title("Solution of Dirichlet B.C. using FEM");
legend('Location', 'southwest');
grid on;
```




```
figure;
plot(X, errors, 'k-*', 'LineWidth', 2, 'DisplayName', 'Error in Solution');
xlabel('x');
ylabel('Error');
title('Error in FEM Solution Using Dirichlet B.C.');
legend show;
grid on;
```




```
% Neumann boundary Conditions
K_global_neumann = K_global;
g_x_neumann = sym(zeros(s, 1));
g_x_neumann(end) = 1;

GNM_neumann = a_num * g_x_neumann * du_dx;
RHS_neumann = GNM_neumann + C;

K_global_neumann(1, :) = [1, zeros(1, s-1)];
RHS_neumann(1) = u0;
u_neumann = inv(K_global_neumann) * RHS_neumann;
displacement_vector_n = double(u_neumann);

% Plot the FEM Solution for Neumann boundary conditions
figure;
plot(X, displacement_vector_n, "r", "LineWidth", 2, "DisplayName", "FEM Solution");
hold on;
fplot(x, exact_sol_T, [0 1], "b", "LineWidth", 1.5, "DisplayName", "Exact Solution");
xlabel("x");
ylabel("u");
title("Solution of Neumann B.C. using FEM with Phantom Node");
legend('Location', 'southwest');
grid on;
```




```
function [phi, dphi] = HierarchicalShapeFunctions(n, xi)
    phi = zeros(1, n);
    dphi = zeros(1, n);
    if n == 2  % Linear case
        phi(1) = 0.5 * (1 - xi);
        phi(2) = 0.5 * (1 + xi);
        dphi(1) = -0.5;
        dphi(2) = 0.5;
    elseif n == 3  % Quadratic case
        phi(1) = xi * (xi - 1) / 2;
        phi(2) = -(xi - 1) * (xi + 1);
        phi(3) = xi * (xi + 1) / 2;
        dphi(1) = xi - 0.5;
        dphi(2) = -2 * xi;
        dphi(3) = xi + 0.5;
    end
end

function [w, psi] = Gauss(p_bar)
    psi = zeros(p_bar, 1);
    w = zeros(p_bar, 1);

    if p_bar == 1
        psi(1) = 0; % for p_bar = 1
        w(1) = 2;
```

```
    elseif p_bar == 2
        psi(1) = sqrt(3)/3;
        psi(2) = -sqrt(3)/3;
        w(1) = 1;
        w(2) = 1;
    elseif p_bar == 3
        psi(1) = sqrt(15)/5;
        psi(2) = -sqrt(15)/5;
        psi(3) = 0;
        w(1) = 5/9;
        w(2) = 5/9;
        w(3) = 8/9;
    elseif p_bar == 4
        psi(1) = sqrt((15 - 2*sqrt(30))/35);
        psi(2) = -sqrt((15 - 2*sqrt(30))/35);
        psi(3) = (sqrt((14*sqrt(30))/5 + 21))/7;
        psi(4) = -(sqrt((14*sqrt(30))/5 + 21))/7; % for p_bar = 7
        w(1) = 1/2 + sqrt(30)/36;
        w(2) = 1/2 + sqrt(30)/36;
        w(3) = 1/2 - sqrt(30)/36;
        w(4) = 1/2 - sqrt(30)/36;
    elseif p_bar == 5
        psi(1) = 0.906179845938664;
        psi(2) = -0.906179845938664;
        psi(3) = 0.538469310105683;
        psi(4) = -0.538469310105683;
        psi(5) = 0;
        w(1) = 0.236926885056189;
        w(2) = 0.236926885056189;
        w(3) = 0.478628670499366;
        w(4) = 0.478628670499366;
        w(5) = 0.568888888888889;
    end
end
```

```
clc;
clear;
syms x;

% Constants and parameters
OIF = 2; % Order of polynomial
E = 1; % Young's modulus
x0 = 0; % Start of the domain
xn = 1; % End of the domain
c_num = -2;

% Source term and exact solution
T = 4*x^2 - 2*x - 4;
exact_sol_T = -2*x^2 + x;
du_dx_exact = diff(exact_sol_T, x); % Derivative of exact solution
strain_exact = du_dx_exact^2 + c_num * exact_sol_T^2; % Strain for exact solution

% Strain energy for the exact solution
strain_energy_exact = double(int(1/2 * E * strain_exact, x, x0, xn));

% Range of elements
element_range = 1:10;
strain_energies_fem = zeros(size(element_range));
strain_energies_exact = repmat(strain_energy_exact, size(element_range));
errors = zeros(size(element_range)); % Store error values

% Initialize the arrays outside the loop
[w, psi] = gaussQuad(OIF + 1); % Assuming maximum order is related to OIF

for e = element_range
    n = OIF + 1;
    h = (xn - x0) / e; % Length of each element
    strain_energy_fem = 0;

    for element = 1:e
        local_X = linspace(x0 + (element-1)*h, x0 + element*h, n);
        local_T_x = double(subs(T, x, local_X));
        local_displacements = double(subs(exact_sol_T, x, local_X));

        for k = 1:length(w)
            [phi_e, dshap] = ShapeFunctions(OIF, psi(k));
            local_strain = dshap .* local_displacements' + c_num .* local_displacements'; % Co
            strain_energy_fem = strain_energy_fem + 1/2 * E * sum(local_strain.^2) * w(k) * h;
        end
    end

    % Store the computed FEM strain energy for this element count
    strain_energies_fem(e) = strain_energy_fem;
    errors(e) = abs((strain_energy_fem - strain_energy_exact) / strain_energy_exact); % Normali
end

% Display and plot results
disp('Exact Strain Energy:');
```

```
Exact Strain Energy:
```

```
disp(strain_energies_exact);
```

```
    1.0333    1.0333    1.0333    1.0333    1.0333    1.0333    1.0333    1.0333    1.0333    1.0333
```

```
disp('FEM Strain Energies:');
```

```
FEM Strain Energies:
```

```
disp(strain_energies_fem);
```

```
    3.4667    1.8792    1.4979    1.3581    1.2922    1.2561    1.2343    1.2200    1.2103    1.2033
```

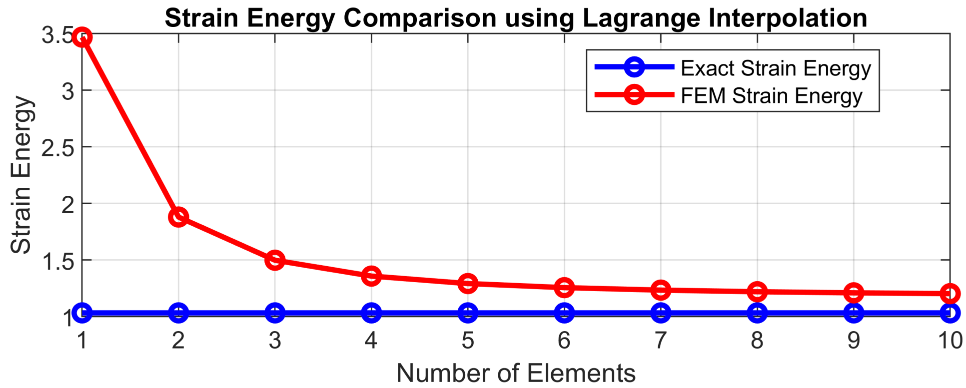
```
disp('Normalized Errors:');
```

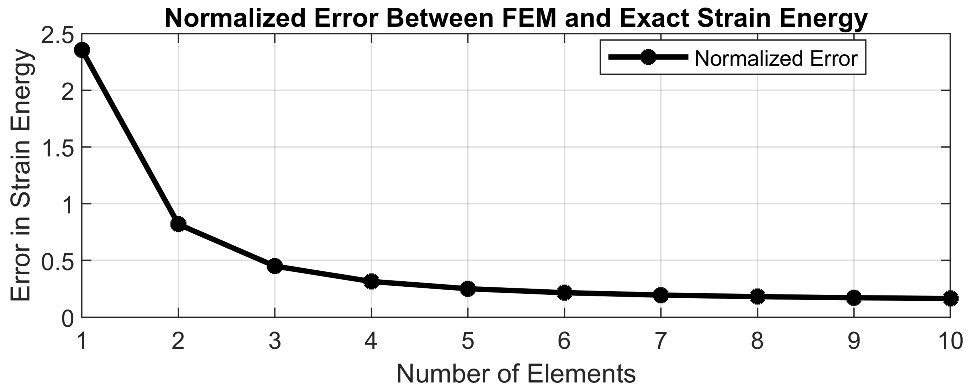
```
Normalized Errors:
```

```
disp(errors);
```

```
    2.3548    0.8185    0.4496    0.3143    0.2505    0.2156    0.1944    0.1807    0.1712    0.1644
```

```

% Plotting the results
figure;
subplot(2, 1, 1);
plot(element_range, strain_energies_exact, 'b-o', 'LineWidth', 2, 'DisplayName', 'Exact Strain
hold on;
plot(element_range, strain_energies_fem, 'r-o', 'LineWidth', 2, 'DisplayName', 'FEM Strain Ener
xlabel('Number of Elements');
ylabel('Strain Energy');
title('Strain Energy Comparison using Lagrange Interpolation');
legend('Location', 'best');
grid on;

subplot(2, 1, 2);
plot(element_range, errors, 'k-*', 'LineWidth', 2, 'DisplayName', 'Normalized Error');
xlabel('Number of Elements');
ylabel('Error in Strain Energy');
title('Normalized Error Between FEM and Exact Strain Energy');
legend('Location', 'best');
grid on;
```






```
function [w, psi] = gaussQuad(p_bar)

    psi = zeros(p_bar, 1);
    w = zeros(p_bar, 1);
   if p_bar == 1
        psi(1) = 0;
        w(1) = 2;
    elseif p_bar == 2
        psi(1) = sqrt(3)/3;
        psi(2) = -sqrt(3)/3;
        w(1) = 1;
        w(2) = 1;
    elseif p_bar == 3
        psi(1) = sqrt(15)/5;
        psi(2) = -sqrt(15)/5;
        psi(3) = 0;
        w(1) = 5/9;
        w(2) = 5/9;
        w(3) = 8/9;
   end
end

function [n, phi_e, dshap] = ShapeFunctions(OIF, x_loc)
```

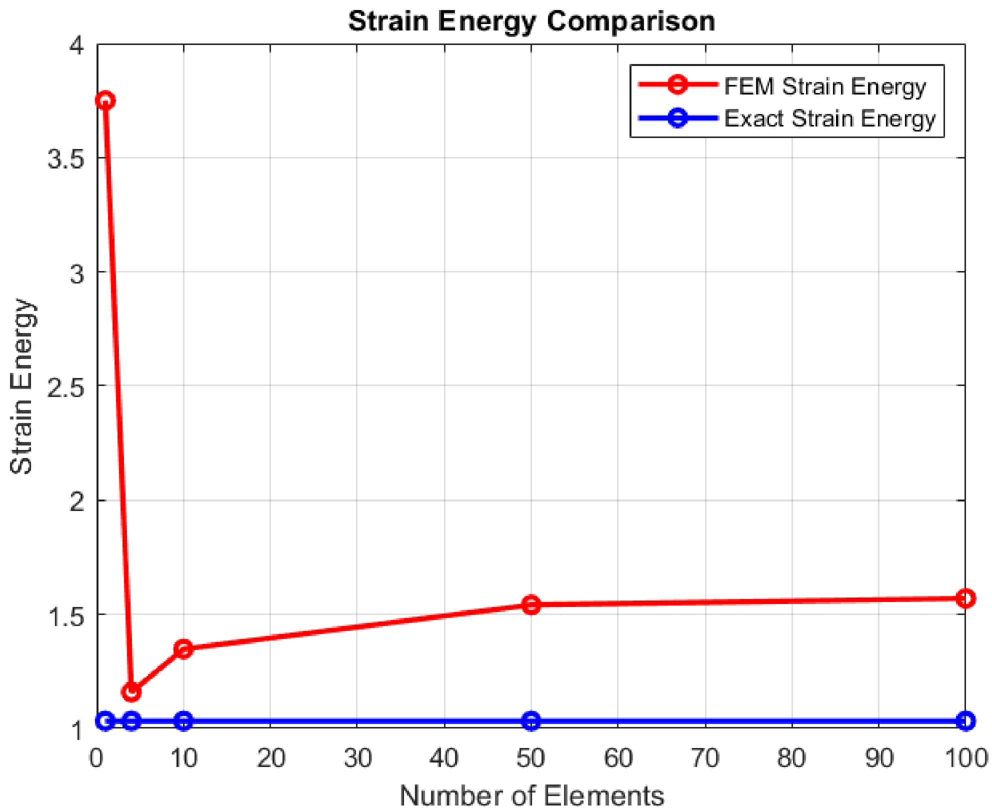
```
    syms x;
    n = OIF + 1;
    x_local = sym(zeros(1, n));

    for i = 1:n
        x_local(i) = -1 + (i-1)*(2/OIF); % Calculate the elements of the vector
    end

    x_s = sym('x', [1, n]);
    N = sym(zeros(size(x_s)));
    df = sym(zeros(size(x_s))); % Initialize df

    for j = 1:n
        Li = 0;
        den = 1;
        product_term = 1;
        for i = 1:n
            if j ~= i
                term = 1; % Initialize product term for index j
                for k = 1:n
                    if k ~= i && k ~= j
                        term = term * (x - x_s(k));
                    end
                end
                % Add the product term for index j to Lagrange polynomial for index i
                Li = Li + term;
            end
          %% creating pattern for differentiation
            if i ~= j
                product_term = product_term * (x - x_s(i)) / (x_s(j) - x_s(i));
                den = den * (x_s(j) - x_s(i));
            end
        end

        N(j) = product_term;
        df(j) = Li / den;
    end

    phi_e = subs(N, x_s, x_local);
    dshap = subs(df, x_s, x_local);

    % Evaluate the shape functions and their derivatives at the Gauss points
    xi_loc = x_loc;
    phi_e = double(subs(phi_e, x, xi_loc))';
    dshap = double(subs(dshap, x, xi_loc))';
end
```

```
clc;
clear;
syms x;

% Constants and parameters
OIF = 2; % Order of polynomial
E = 1; % Young's modulus
x0 = 0; % Start of the domain
xn = 1; % End of the domain
c_num = -2;

% Source term and exact solution
T = 4*x^2 - 2*x - 4;
exact_sol_T = -2*x^2 + x;   % Exact solution for displacement
du_dx_exact = diff(exact_sol_T, x); % Derivative of exact solution
strain_exact = du_dx_exact^2 + c_num * exact_sol_T^2; % Strain for exact solution

% Strain energy for the exact solution
strain_energy_exact = double(int(1/2 * E * strain_exact, x, x0, xn))
```

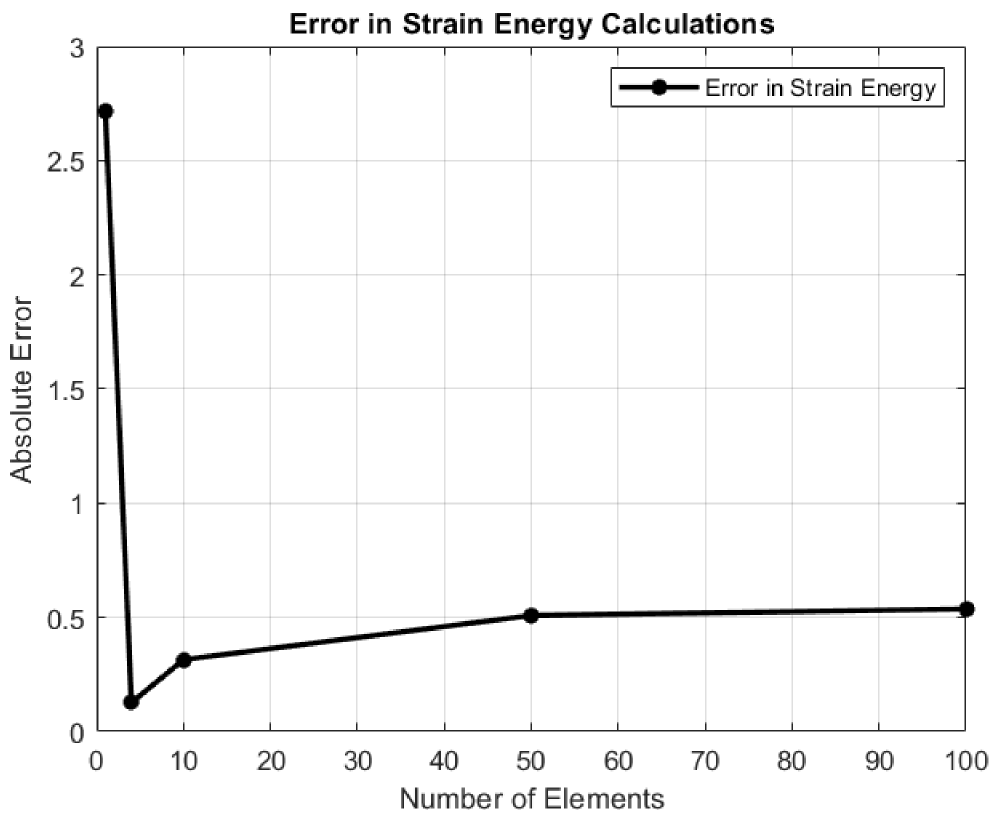
```
strain_energy_exact = 1.0333
```

```

% Element range specified
element_ranges = [1, 4, 10, 50, 100];
strain_energies_fem = zeros(size(element_ranges));
errors = zeros(size(element_ranges));

for idx = 1:length(element_ranges)
    e = element_ranges(idx);
    n = OIF + 1;
    h = (xn - x0) / e; % Length of each element

    % Gauss quadrature process
    [w, psi] = gaussQuad(n); % Using number of nodes to determine Gauss points

    % Initialize variables for integration
    strain_energy_fem = 0;

    for element = 1:e
        local_X = linspace(x0 + (element-1)*h, x0 + element*h, n);
        local_T_x = double(subs(T, x, local_X));
        local_displacements = double(subs(exact_sol_T, x, local_X));

        % Local strain calculations
        for k = 1:length(w)
            [phi_e, dphi_e] = shapeFunctions(OIF, psi(k));
            local_strain = dphi_e * local_displacements' + c_num * local_displacements'; % Corr
            strain_energy_fem = strain_energy_fem + 1/2 * E * sum(local_strain.^2) * w(k) * h;
        end
    end

    % Store results
    strain_energies_fem(idx) = strain_energy_fem;
```

```
    errors(idx) = abs(strain_energy_fem - strain_energy_exact);
end

% Plotting results
figure;
plot(element_ranges, strain_energies_fem, 'r-o', 'LineWidth', 2, 'DisplayName', 'FEM Strain Ene
hold on;
plot(element_ranges, repmat(strain_energy_exact, 1, length(element_ranges)), 'b-o', 'LineWidth

xlabel('Number of Elements');
ylabel('Strain Energy');
title('Strain Energy Comparison');
legend('show');
grid on;
```




```
% Additional plotting for error comparison
figure;
plot(element_ranges, errors, 'k-*', 'LineWidth', 2, 'DisplayName', 'Error in Strain Energy');
xlabel('Number of Elements');
ylabel('Absolute Error');
title('Error in Strain Energy Calculations');
legend('show');
grid on;
```




```
function [w, psi] = gaussQuad(numPoints)
    % Returns the weights and positions for Gauss quadrature
    w = zeros(numPoints, 1);
    psi = zeros(numPoints, 1);
    if numPoints == 1
        psi(1) = 0;
        w(1) = 2;
    elseif numPoints == 2
        psi = [-sqrt(1/3), sqrt(1/3)];
        w = [1, 1];
    elseif numPoints == 3
        psi = [-sqrt(3/5), 0, sqrt(3/5)];
        w = [5/9, 8/9, 5/9];
    end
end

function [phi_e, dphi_e] = shapeFunctions(OIF, psi_val)
    % Shape functions and derivatives for Quadratic elements
    if OIF == 2
        phi_e = [0.5*psi_val*(psi_val-1), 1-psi_val^2, 0.5*psi_val*(psi_val+1)];
        dphi_e = [psi_val - 0.5, -2*psi_val, psi_val + 0.5];
    else
```

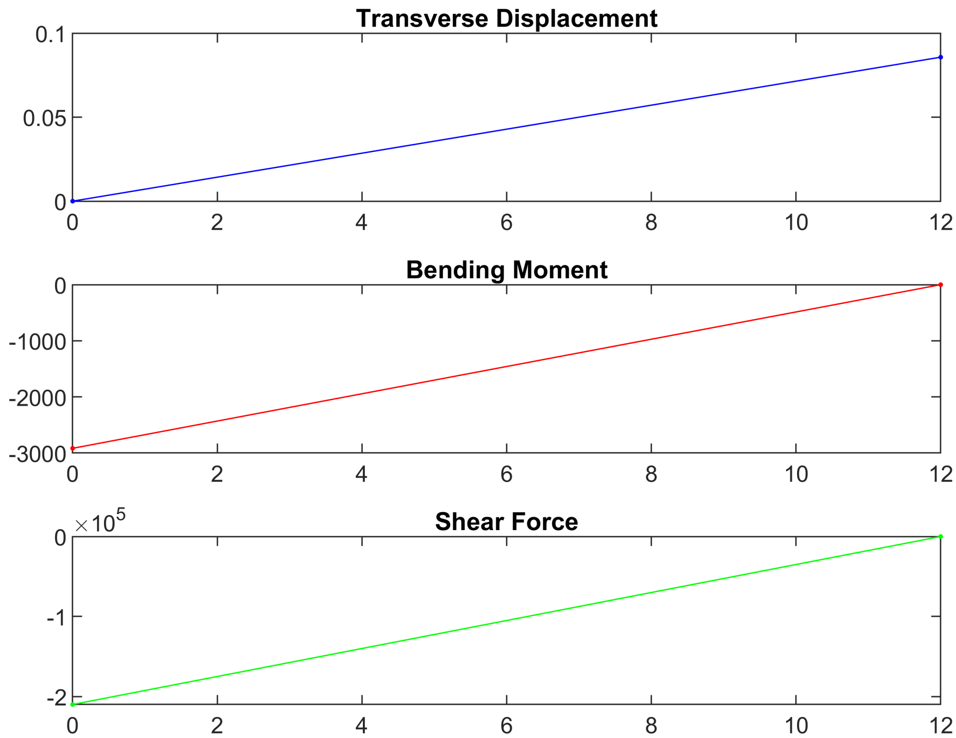
```
        error('Shape functions for given order not implemented.');
    end
end
```

```
clc;
clear;


n = 4; % Number of elements
E = 210e9; % Modulus of Elasticity in Pa
I = 2e-4; % Moment of Inertia
beam_length = 12; % Length of the beam
H = 0.5; % Height of the beam in meters, added for stress calculations

for element_count = [1, 4, 10, 50, 100]  % Different element range
    n = element_count;
    el = beam_length / n; % Length of each element

    % Local stiffness matrix for one element
    ele_K = E * I / el^3 * [12    6*el  -12    6*el;
                             6*el  4*el^2 -6*el  2*el^2;
                            -12   -6*el    12   -6*el;
                             6*el  2*el^2 -6*el  4*el^2];

    % Global stiffness matrix
    K_global = zeros(2*(n+1), 2*(n+1));

    % Assemble the global stiffness matrix
    for element = 1:n
        start_index = 2*element - 1;
        end_index = start_index + 3;
        K_global(start_index:end_index, start_index:end_index) = ...
            K_global(start_index:end_index, start_index:end_index) + ele_K;
    end

% Load vectors initialization
F = zeros(2*(n+1), 1); % Initialize distributed load vector
Q = zeros(2*(n+1), 1); % Initialize point load and moment vector

% Applying distributed loads
for element = 1:n
    start_index = 2 * element - 1;
    end_index = start_index + 3;
    F_local = 1000 * el * [1/2; el/12; 1/2; -el/12];  % 1000 N/m as an example distributed load
    F(start_index:start_index+3) = F(start_index:start_index+3) + F_local;
end

% Initialize parameters for loads
%P_center = 5000;  % 5000 N point load at the center
P_end = 1500;     % 1500 N point load at the end
M_end = 2000;     % 2000 Nm moment at the end

% Apply point load at the center of the beam
center_node_index = n / 2 + 1;  % Correct index for an even number of elements
%Q(2 * center_node_index - 1) = P_center;  % Apply to the vertical displacement DOF

% Apply point load and moment at the end of the beam
```

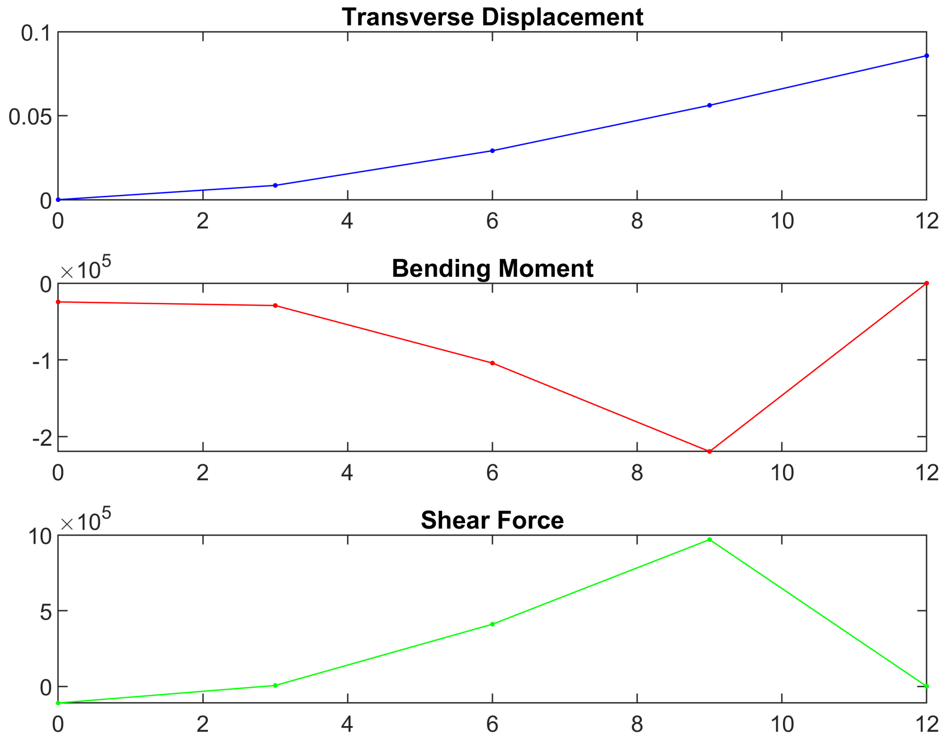
```
Q(end-1) = P_end;  % Point load at the end
Q(end) = M_end;    % Moment at the end


% Total load vector
Total_Load = F + Q;

% Apply boundary conditions
% For a fixed support at node 1 (no displacement or rotation)
K_global(1,:) = 0;
K_global(:,1) = 0;
K_global(1,1) = 1;
Total_Load(1) = 0;

K_global(2,:) = 0;
K_global(:,2) = 0;
K_global(2,2) = 1;
Total_Load(2) = 0;

    % Solve for displacements
    displacements = K_global \ Total_Load;

    displacements

    % Post-processing for shear forces and bending moments
    M = zeros(n+1, 1);
    V = zeros(n+1, 1);
    for i = 1:n
        % Two nodes per element
        start_index = i;
        end_index = i + 1;

        % Displacement derivatives (finite differences)
        dwdx = (displacements(2*end_index) - displacements(2*start_index-1)) / 2*el;
        d2wdx2 = (displacements(2*end_index) - 2*displacements(2*start_index) + displacements(2

        % Bending moment and shear force
        M(start_index) = -E * I * d2wdx2; % bending moment
        V(start_index) = -E * I * (dwdx / el);  % shearforces
    end

    % Bending stress calculation
    sigma = M * (H / 2) / I;

    % Output results
    x = linspace(0, beam_length, n+1);
    figure;
    subplot(3,1,1);
    plot(x, displacements(1:2:end), 'b.-');  % Plotting vertical displacements
    title('Transverse Displacement');

    subplot(3,1,2);
    plot(x, M, 'r.-');
    title('Bending Moment');
```

```
    subplot(3,1,3);
    plot(x, V, 'g.-');
    title('Shear Force');

    fprintf('Analysis for %d elements completed.\n', n);
end
```

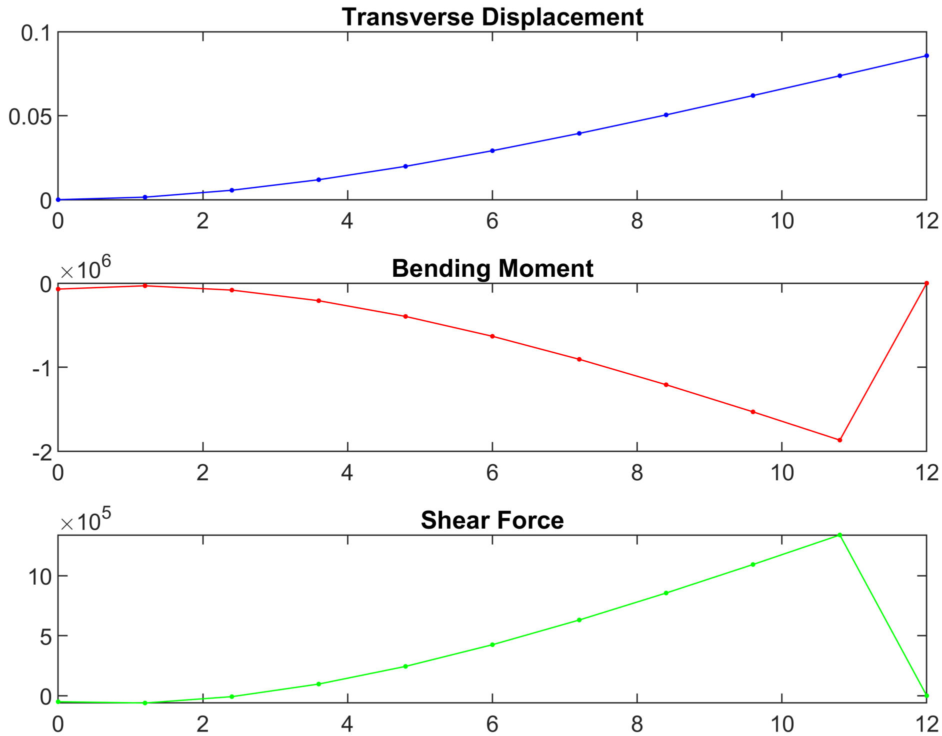
```
displacements = 4×1
         0
         0
    0.0857
    0.0100
```



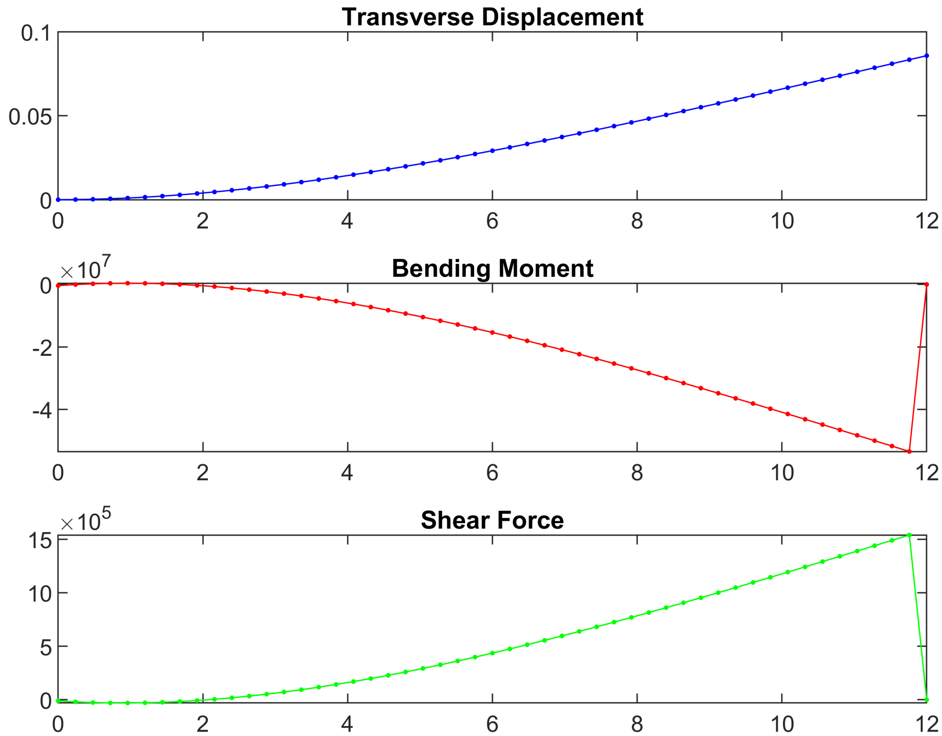

```
Analysis for 1 elements completed.
displacements = 10×1
         0
         0
    0.0085
    0.0052
    0.0291
    0.0082
    0.0562
    0.0096
    0.0857
    0.0100
```



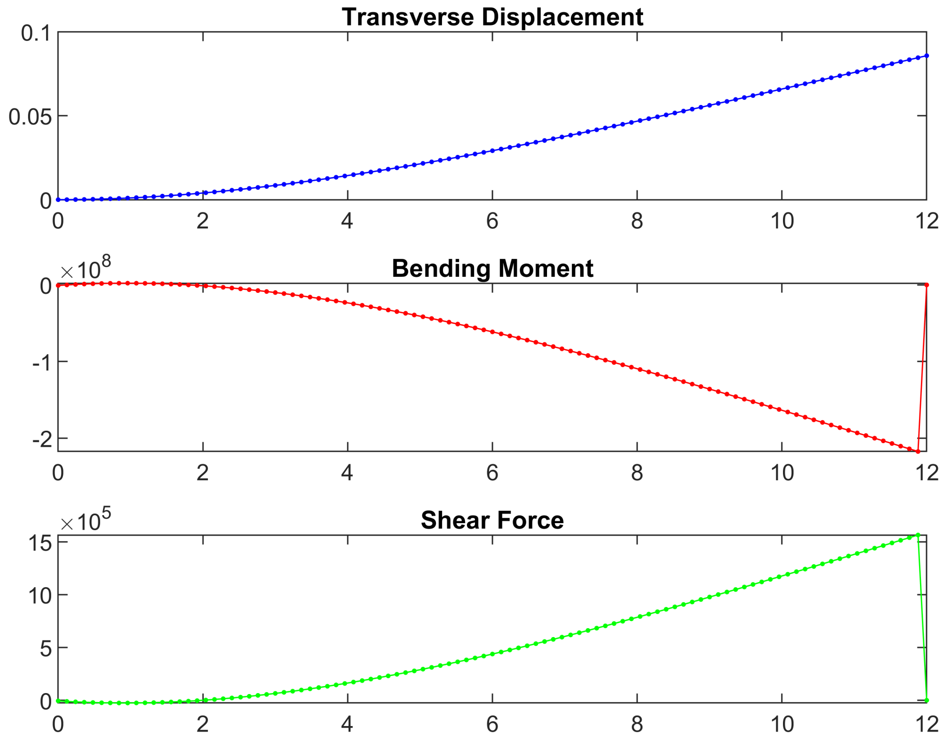

```
Analysis for 4 elements completed.
displacements = 22×1
         0
         0
    0.0015
    0.0024
    0.0056
    0.0044
    0.0119
    0.0060
    0.0198
    0.0073
      ⋮
```




```
Analysis for 10 elements completed.
displacements = 102×1
         0
         0
    0.0001
    0.0005
    0.0002
    0.0010
    0.0005
    0.0015
    0.0010
    0.0020
      ⋮
```




```
Analysis for 50 elements completed.
displacements = 202×1
         0
         0
    0.0000
    0.0003
    0.0001
    0.0005
    0.0001
    0.0008
    0.0002
    0.0010
      ⋮
```




```
Analysis for 100 elements completed.
```