COMPUTATIONAL FLUID DYNAMICS

MEPE11 - Assignment - Group 17

Deflection of Cantilever Beam under Uniform Distributed Load
Finite Element Method (FEM)

Submitted by

111117030 : **Devarasetty Sasi Preetham**
111117036 : **Gudapati Nitish**
111117070 : **R Mukesh Kanna**

Mechanical Engineering

National Institute of Technology
Tiruchirappalli (NIT-T)
Tamil Nadu - INDIA

Contents

List of Figures

1 Problem Statement

The deflection 'u' of a cantilever beam under uniformly distributed load is governed by

$$\frac{d^2u}{dx^2} = -150 + 300x - 150x^2$$

The boundary conditions being:

$$u = \frac{\partial u}{\partial x} = 0 \qquad at \quad x = 0$$

Plot the beam deflection as a function of length. Take the length of beam as 5 units. Discretize the governing equations using FEM and solve using Multigrid Gauss-seidel method.

2 Introduction

Finite Element Analysis or FEA is the simulation of a physical phenomenon using a numerical mathematical technique. Analyzing most of the physical phenomena can be done using partial differential equations, but in complex situations where multiple highly variable equations are needed, Finite Element Analysis becomes useful. Computational fluid dynamics (CFD) is an extension of FEA that, with the help of digital computers, produces quantitative predictions of fluid-flow phenomena based on the conservation laws (conservation of mass, momentum, and energy) governing fluid motion. FEA involves 2 steps:

1. Discretization to derive the equations for the problem.
2. Solving the derived equations using numerical methods.

2.1 Discretization using Finite Element Method

The discretization of a boundary-value problem by the finite element method requires the evaluation of various integrals over the elements into which the region of interest is partitioned. A significant characteristic of any patchwork approximation used in finite element computation is the ease with which the integrals can be evaluated. When the integrals are solved over each element, it finally results in a system of algebraic equations.

2.2 Multi-grid Gauss Seidel Method

Gauss-Seidel iterations produce smooth errors. The error vector e has its high frequencies nearly removed in a few iterations. But low frequencies are reduced very slowly. Convergence requires O(n^2) iterations, which can be unacceptable. The extremely effective multigrid idea is to change to a coarser grid, on which smooth becomes rough and low frequencies act like higher frequencies. On that coarser grid a big piece of the error is removable. We iterate only a few times before changing from fine to coarse and coarse to fine. The remarkable result is that multigrid can solve many sparse and realistic systems to high accuracy in a fixed number of iterations, not growing with n.

3 Numerical Simulation

The investigation of deformation of a cantilever beam subjected to a uniform distributed load is a simple static structural problem. In order to correctly solve the problem the foremost objective is to make valid assumptions.

3.1 Assumptions

1. The beam is being analysed only along its length dimension.
2. The material behaviour is considered to be linear.
3. Forces are being applied slowly and do not change direction with time.
4. Contact does not change with time.
5. The deformation at various locations on the beam is sufficiently small so that the stiffness does not change.

3.2 Boundary conditions

1. The deformation at the fixed end is 0.
2. The rate of change of deformation with respect to x at the fixed end is 0.

3.3 Discretization using FEM

Galerkin's Method:

$$\int_i^j \left(\frac{d^2u}{dx^2} + 150x^2 - 300x + 150\right) * w dx = \int_i^j \left(\frac{d^2u}{dx^2} + 150(x-1)^2\right) * w dx = 0$$

Assuming Linear profile:

$$u = a_0 + a_1 x$$

$$u_i = a_0 + a_1 x_i \quad , \quad u_j = a_0 + a_1 x_j \tag{1}$$

$$a_1 = \frac{u_j - u_i}{x_j - x_i} \quad , \quad a_0 = \frac{u_i x_j - u_j x_i}{x_j - x_i} \tag{2}$$

$$u = \frac{x_j - x}{x_j - x_i} u_i + \frac{x - x_i}{x_j - x_i} u_j \tag{3}$$

$$u = N_i u_i + N_j u_j \tag{4}$$

$$N_i = 1 \quad at\ node\ i \quad , \quad N_i = 0 \quad at\ node\ j$$

$$u = \begin{bmatrix} N_i & N_j \end{bmatrix} \begin{bmatrix} u_i \\ u_j \end{bmatrix}$$

$$u = [N][U] \quad , \quad w = [N][W] \tag{5}$$

As w is scalar, $$w^T = w$$

$$[w] = \begin{bmatrix} w_i \\ w_j \end{bmatrix}$$
$$w = [W]^T [N]^T$$

Now integrating,

$$[W]^T [N]^T \frac{du}{dx}|_i^j - [W]^T \int_{x_i}^{x_j} [\frac{dN}{dx}]^T [\frac{dN}{dx}][U] dx + [W]^T \int_{x_i}^{x_j} 150(x-1)^2 [N]^T dx = 0 \tag{6}$$

$$Let, \quad x_j - x_i = \Delta x \quad and \quad [W]^T\ is\ arbitrary$$

Term 1:

$$\begin{bmatrix} N_i \\ N_j \end{bmatrix} \frac{du}{dx}|_i^j = \begin{bmatrix} -\frac{du_i}{dx} \\ \\ \frac{du_j}{dx} \end{bmatrix}$$

Term 2:

$$\int_{x_i}^{x_j} \begin{bmatrix} \frac{dN_i}{dx} \\ \frac{dN_j}{dx} \end{bmatrix} \begin{bmatrix} \frac{dN_i}{dx} & \frac{dN_j}{dx} \end{bmatrix} [U]dx = \frac{1}{\Delta x} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} [U]$$

$$(\frac{dN_i}{dx} = \frac{-1}{\Delta x} \quad , \quad \frac{dN_j}{dx} = \frac{1}{\Delta x})$$

Term 3:

$$\int_{x_i}^{x_j} 150(x-1)^2 \begin{bmatrix} N_i \\ N_j \end{bmatrix} dx = 150 \int_{x_i}^{x_j} (x-1)^2 \begin{bmatrix} \frac{x_j - x}{\Delta x} \\ \frac{x - x_i}{\Delta x} \end{bmatrix} dx = \frac{150}{\Delta x} \int_{x_i}^{x_j} \begin{bmatrix} (x_j - x)(x-1)^2 \\ (x - x_i)(x-1)^2 \end{bmatrix}$$

$$= \frac{150}{\Delta x} \int_{x_i}^{x_j} \begin{bmatrix} x^2 x_j - 2x x_j + x_j - x^3 + 2x^2 - x \\ x^3 - 2x^2 + x - x^2 x_i + 2x x_i - x_i \end{bmatrix} dx$$

$$= \frac{150}{\Delta x} \begin{bmatrix} \frac{x^3 x_j}{3} - x^2 x_j + x x^j - \frac{x^4}{4} + \frac{2x^3}{3} - \frac{x^2}{2} \\ \frac{x^4}{4} - \frac{2x^3}{3} + \frac{x^2}{2} - \frac{x^3 x_i}{3} + x^2 x_i - x x^i \end{bmatrix}_i^j$$

$$= \frac{150}{\Delta x} \begin{bmatrix} (\frac{x_j^4 + 3x_i^4}{12}) - (\frac{x_j^3 + 2x_i^3}{3}) + (\frac{x_j^2 + x_i^2}{2}) - x_j(\frac{x_i^3}{3} - x_i^2 + x_i) \\ (\frac{x_i^4 + 3x_j^4}{12}) - (\frac{x_i^3 + 2x_j^3}{3}) + (\frac{x_j^2 + x_i^2}{2}) - x_i(\frac{x_j^3}{3} - x_j^2 + x_j) \end{bmatrix}$$

Adding all terms: Term 1 + Term 2 + Term 3

$$\begin{bmatrix} \frac{-du_i}{dx} \\ \frac{du_j}{dx} \end{bmatrix} - \frac{1}{\Delta x} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} [U] + \frac{150}{\Delta x} \begin{bmatrix} (\frac{x_j^4 + 3x_i^4}{12}) - (\frac{x_j^3 + 2x_i^3}{3}) + (\frac{x_j^2 + x_i^2}{2}) - x_j(\frac{x_i^3}{3} - x_i^2 + x_i) \\ (\frac{x_i^4 + 3x_j^4}{12}) - (\frac{x_i^3 + 2x_j^3}{3}) + (\frac{x_j^2 + x_i^2}{2}) - x_i(\frac{x_j^3}{3} - x_j^2 + x_j) \end{bmatrix} = 0$$

KX = F (for an element with length Δx) :

$$\frac{1}{\Delta x} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ u_j \end{bmatrix} = \begin{bmatrix} \frac{-du_i}{dx} \\ \frac{du_j}{dx} \end{bmatrix} + \frac{150}{\Delta x} \begin{bmatrix} (\frac{x_j^4 + 3x_i^4}{12}) - (\frac{x_j^3 + 2x_i^3}{3}) + (\frac{x_j^2 + x_i^2}{2}) - x_j(\frac{x_i^3}{3} - x_i^2 + x_i) \\ (\frac{x_i^4 + 3x_j^4}{12}) - (\frac{x_i^3 + 2x_j^3}{3}) + (\frac{x_j^2 + x_i^2}{2}) - x_i(\frac{x_j^3}{3} - x_j^2 + x_j) \end{bmatrix}$$

3.4 Working algorithm

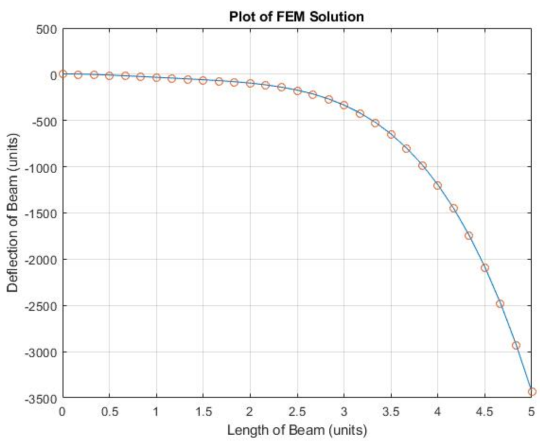
The basic algorithm for solution of the given problem is as under:

1. Define the input conditions: material and length of the beam which are used to arrive at the governing differential equation and boundary conditions.
2. Define the number of elements.
3. Discretize the equation and find the overall stiffness matrix to form the global KX=F equation.
4. Solve the set of equations using multigrid Gauss-Seidel method to generate deformation along the length of the beam.
5. Iterate until the convergence criterion is met.

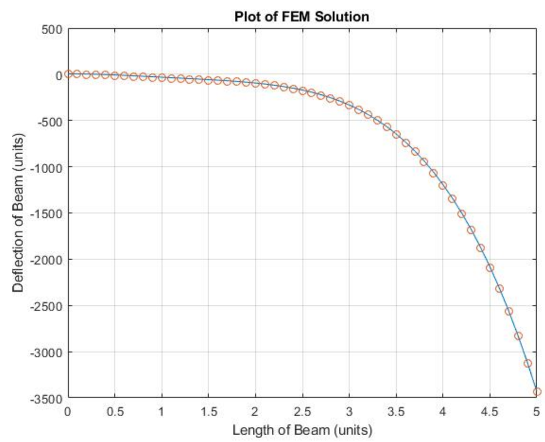
The program has been developed in MATLAB adhering to the above stated algorithm. The source code is presented in Appendix A, and can also be accessed from the GitHub link: https://github.com/gnitish18/FEM_Multigrid

4 Numerical Results

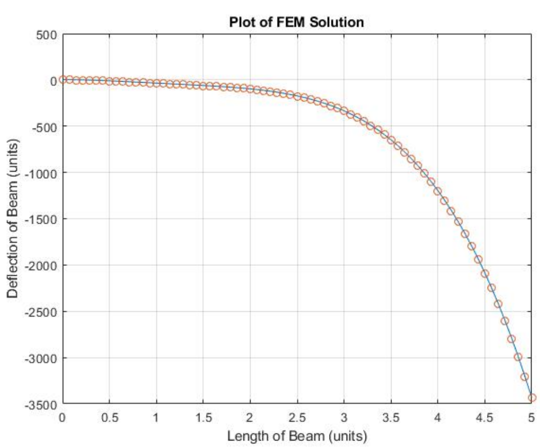
The present section is devoted to the presentation of numerical results of the problem. In order to check the efficacy of the numerical model developed the first task is to perform the grid independence check. The grid independence check is carried out by varying the number of grids. In the present case the above problem has been solved employing 30 grids, 50 grids, 70 grids and 100 grids, as shown in Figure 1. Overlapping deformation profiles were obtained, as shown in Figure 3. This proves that the profile is not changing with grid size above 30 elements. When the simulation is run for 20 grids, the deformation (Figure 2) plot is found to be slightly deviating from the actual results as shown in Figure 5.



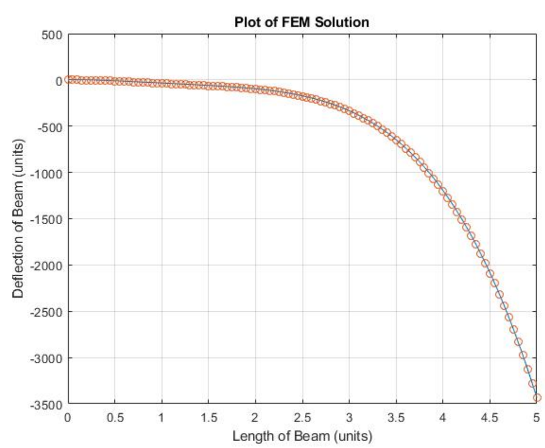
((a)) 30 elements



((b)) 50 elements



((c)) 70 elements



((d)) 100 elements

Figure 1: FEM solution with various grid sizes

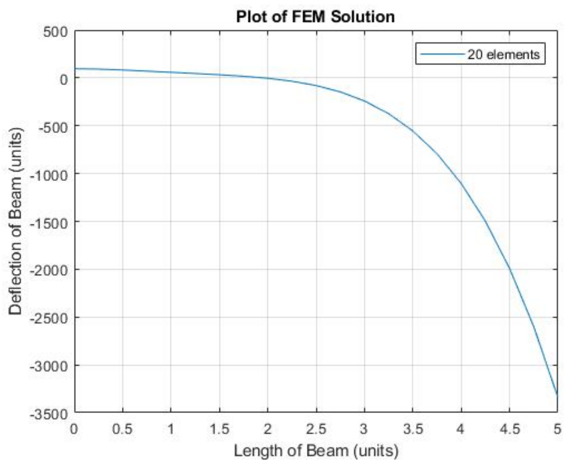


Figure 2: FEM solution for 20 elements

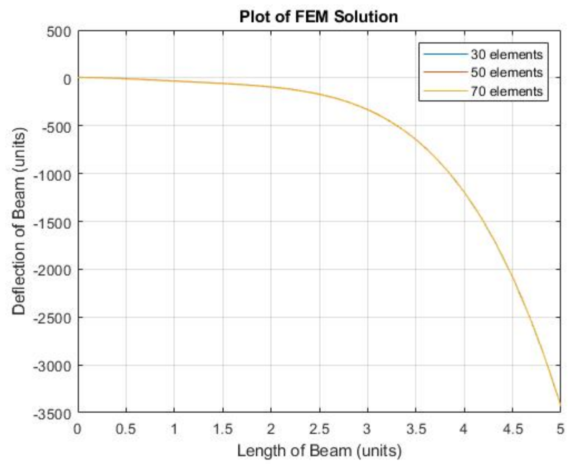


Figure 3: Grid independence check

5 Validation

The developed numerical model is validated by comparing the numerical results to the analytical solution of the governing differential equation, as shown in Figure 4. The obtained experimental and numerical deformation profiles are also shown for comparison. It is evident that the analytical data is in close agreement with the predicted numerical profile after grid independence is achieved, as shown in Figure 6.

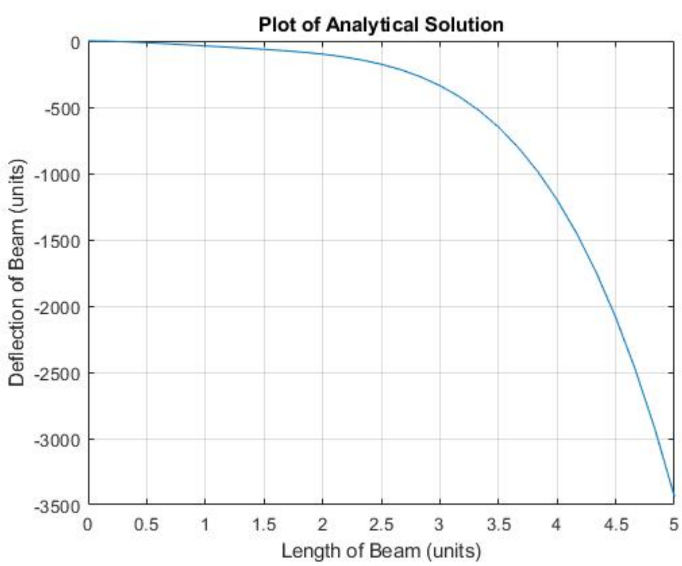


Figure 4: Analytical Solution

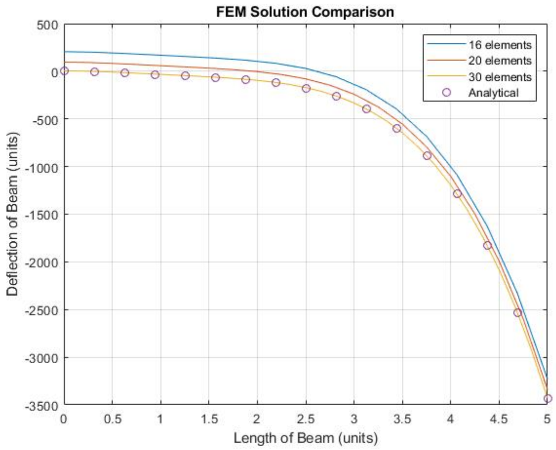


Figure 5: FEM solution for grid < 30

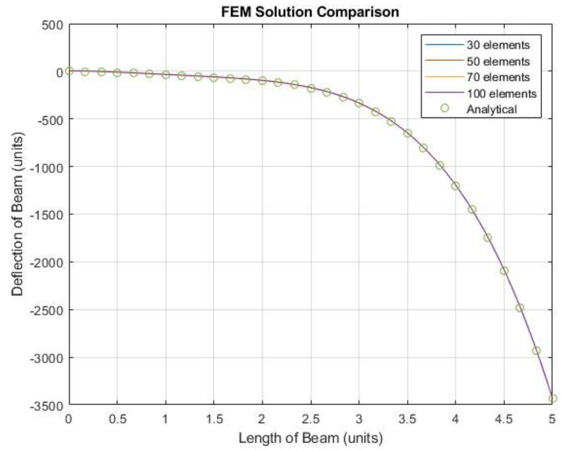


Figure 6: FEM solution for grid > 30

6 Conclusion

A suitable numerical model for performing structural analysis of cantilever beam under uniformly distributed load has been presented. The effect of variation in grid size was checked and grid independence was established. The model has been validated by comparing the numerical results with the analytical solution. The physical aspects of the problem have been discussed in Appendix B.

A MATLAB Source-Code

A.1 Main Program

```
%% Program to determine the deflection of a Cantilever-Beam under Uniform
    Distributed Load using Finite Element Method (FEM) and solved using
    Multigrid Gauss-Seidel method
%%

clear;
close all;
clc;

% Get the number of elements
disp("Grid independency is achieved at 30 elements");
disp("However, feel free to test with any number of elements");
m = input("Enter the number of elements: ");

% If the number of elements is odd, makes it even
% to satisfy multigrid transformation matrix sizes
m = m + mod(m,2);

fprintf("Initializing... ");
n     = m+1;                    % Initialize number of nodes
l     = 5;                      % Define length of the beam
h     = l/m;                    % Length of each element
k     = 1/h*[1, -1; -1, 1];     % Individual stiffness matrix
Ah    = zeros(n,n);             % Deflection matrix
F     = zeros(n,1);             % Force matrix

% Define overall stiffness matrix
for i = 1:m
    Ah(i,i)     = Ah(i,i)     + k(1,1);
    Ah(i,i+1)   = Ah(i,i+1)   + k(1,2);
    Ah(i+1,i)   = Ah(i+1,i)   + k(2,1);
    Ah(i+1,i+1) = Ah(i+1,i+1) + k(2,2);
end

% Force matrix (F) in A*X = F
for i = 1:m
    % Initialize the nodal locations
    xi = i*h;
    xj = xi-h;

    % Initialize Common term
    t = (xi^4-xj^4)/12 - (xi^3-xj^3)/3 + (xi^2-xj^2)/2;
    % Initialize the terms
    T1 = -xj^3/3 + xj^2 - xj + 1/h*t;
    T2 = xi^3/3 - xi^2 + xi - 1/h*t;

```

```
    % Update the Force Matrix
    F(i)   = F(i)   + 150*T1;
    F(i+1) = F(i+1) + 150*T2;
end

% Add last term of force matrix - slope
F(n) = F(n) + (-150*l + 150*l^2 - 50*l^3);
fprintf(" Done");
pause(0.3)
fprintf(repmat('\b', 1, 21));

% Solve the equation using Multigrid Gauss-Seidel
U = Multigrid_TwoGrid(Ah, F);

% Display the results and analysis
Results(U);
```

A.2 Functions

A.2.1 Multigrid Algorithm

```
%% Function to solve the matrix equation using Multigrid Algorithm implementing two-grid
%   Input Parameters:
%   * Ah - Coefficient Fine Grid Matrix
%   * F  - Resultant Force Matrix
%   Output Parameters:
%   * U  - Solved Deflection Matrix
%%

function U = Multigrid_TwoGrid(Ah, F)

    fprintf("Running Multigrid...");
    % Find the size of the fine stiffness matrix
    [n,~]    = size(Ah);

    u(1:n,1) = 0;       % Initialization of Initial Deflection matrix
    U(1:n,1) = 0;       % Define Final Deflection matrix
    U_0      = 0;       % Boundary condition
    v1       = 1;       % Number of iterations for Gauss-Seidal
    fl       = 1;       % Flag to ensure first iteration

    % Define Projection Operator matrix
    I = zeros(n,floor(n/2));

    fprintf("\nIterating Gauss-Seidel with optimum iterations...");
    % Loop to find the optimum number of iterations
    % for application of initial Gauss-Seidel relaxation
    % Error is U_0 - U(1), which is minimized
```

```
    while (U_0 - U(1) > 0 || fl == 1)

        fl = 0;          % Disable flag
        v1 = v1 + 1;     % Increment the number of iterations

        % Applying Gauss-Seidel relaxation method with v1 iterations
        U  = Gauss_Seidel(Ah, F, u, 0, v1);

        % Initialize the Projection Operator matrix as tri-diagonal
        for i = 1:floor(n/2)
            for j = 2*i-1:2*i+1
                if mod(j,2) == 0
                    I(j,i) = 2;
                else
                    I(j,i) = 1;
                end
            end
        end

        R   = 0.5*I';   % Initialize Restriction Matrix
        rh  = F - Ah*U; % Compute Residue
        r2h = R*rh;     % Project Residue from fine grid to coarse

        % Project Stiffness matrix from fine grid to coarse
        A2h = Project_FineToCoarse(Ah);

        e2h = A2h\r2h;  % Solve to find the error
        eh  = I*e2h;    % Interpolate error from coarse grid to fine
        U  = U - eh;    % Update the solution

        % Applying Gauss-Seidel relaxation method until convergence
        % with updated solution of initial deflection matrix
        U   = Gauss_Seidel(Ah, F, U, 1, 1);

    end

    fprintf(" Done");
    pause(0.3)
    fprintf(repmat('\b', 1, 55));
    fprintf(" Done");
    pause(0.3)
    fprintf(repmat('\b', 1, 25));

end
```

A.2.2 Gauss-Seidel Algorithm

```
%% Function to solve the matrix equation A*x = B using iterative Gauss-Seidel
    relaxation method
%   Input Parameters:
%   * A  - Coefficient Matrix
```

```
%   * B  - Resultant Matrix
%   * X  - Variable Matrix
%   * fl - Flag variable for termination condition
%   * v  - Number of iterations
%   Output Parameters:
%   * X  - Solved Variable Matrix
%%

function X = Gauss_Seidel(A, B, X, fl, v)
    % Find the size of the matrix
    [n,~]     = size(A);

    % Iterate Gauss-Seidal for v iterations when flag is disabled
    if ~fl

        for k = 1:v
            % Store previous iteration values
            x_old = X;
            for i = 1:n
                sigma = 0;
                for j = 1:i-1
                    sigma = sigma + A(i,j)*X(j);
                end
                for j = i+1:n
                    sigma = sigma + A(i,j)*x_old(j);
                end
                X(i) = (1/A(i,i))*(B(i) - sigma);
            end

        end

    % Iterate Gauss-Seidal until convergence when flag is enabled
    else
        % Initial error
        normval = 1;

        % Iterate until error is greater than tolerance
        while normval>0
            % Store previous iteration values
            x_old = X;

            for i = 1:n
                sigma = 0;
                for j = 1:i-1
                    sigma = sigma + A(i,j)*X(j);
                end
                for j = i+1:n
                    sigma = sigma + A(i,j)*x_old(j);
                end
                X(i) = (1/A(i,i))*(B(i) - sigma);
            end
```

```
            % Update error and find its norm
            normval = norm(X-x_old);
        end

    end

end
```

A.2.3 Projection from Fine Grid to Coarse Grid

```
%% Function to project a given Fine-Grid matrix to Coarse-Grid
%   Input Parameters:
%   * A - Fine Grid Matrix
%   Output Paramerter:
%   * B - Coarse Grid Matrix
%%

function B = Project_FineToCoarse(A)
    % Find the size of the fine stiffness matrix
    [n,~] = size(A);
    % Define the Projection Operator matrix
    I(1:n,1:floor(n/2)) = 0;

    % Initialize the Projection Operator matrix as tri-diagonal
    for i = 1:floor(n/2)
        for j = 2*i-1:2*i+1
            if mod(j,2) == 0
                I(j,i) = 2;
            else
                I(j,i) = 1;
            end
        end
    end

    R = 0.5*(I');   % Initialize Restriction matrix
    B = R*A*I;      % Project the fine stiffness matrix to coarse

end
```

A.2.4 Results and Analysis

```
%% Function to plot the Analytical solution and compare it with the FEM
    solution
%   Input Parameters:
%   * U_fem - FEM solution of Deflection Matrix
%%

function Results(U_fem)

    fprintf("Displaying Results...")
```

```
    % Find the size of the fine stiffness matrix
    [n,~] = size(U_fem);

    % Generate points along the length of beam
    X = linspace(0,5,n);

    figure(1);      % Initialize the plot name
    plot(X, U_fem); % Plot the deflection of the beam vs its nodal location
    title('Plot of FEM Solution')
    ylabel('Deflection of Beam (units)')
    xlabel('Length of Beam (units)')
    grid on

    % Analaytical solution of deflection
    U_ana = -75*X.^2 + 50*X.^3 - 12.5*X.^4;

    figure(2);      % Initialize the plot name
    plot(X, U_ana); % Plot the deflection of the beam vs its nadal location
    title('Plot of Analytical Solution')
    ylabel('Deflection of Beam (units)')
    xlabel('Length of Beam (units)')
    grid on

    figure(3);      % Initialize the plot name
    plot(X, U_fem); % Plot the deflection of the beam vs its nadal location
    title('Analytical and FEM Comparison')
    ylabel('Deflection of Beam (units)')
    xlabel('Length of Beam (units)')
    hold on         % Multiple plots on same graph
    scatter(X, U_ana);
    legend('FEM','Analytical')
    hold off
    grid on

    fprintf("Done");
    pause(0.3)
    fprintf(repmat('\b', 1, 26));

    % Display the animation of Beam Deflection
    Animate(U_fem, U_ana, X);

end
```

A.2.5 Animation

```
%% Function to display the animation of Beam Deflection
%   Input Parameters:
%   * U_fem - FEM solution deflection matrix
%   * U_ana - Analytical solution deflrction matrix
%   * X     - Nodal locations along beam length
```

```
%%

function Animate(U_fem, U_ana, X)

    % Define number of frames
    framemax = 90;
    M = moviein(framemax);

    % Set the position of the animation window
    set(gcf,'Position',[100 100 640 480]);

    for i = 1:framemax

        % Interpolate the beam deflections framewise
        u_fem = U_fem*i/framemax;
        u_ana = U_ana*i/framemax;

        % Plot the deflection alog the length of the beam
        figure(4)
        plot(X,u_fem,'k','LineWidth',5);
        hold on
        scatter(X,u_ana,'y','LineWidth',1.5);
        axis([0 5 -4000 1500])
        title('Deflection of Cantilever Beam','fontsize',18)
        ylabel('Deflection of Beam (units)')
        xlabel('Length of Beam (units)')
        legend('FEM','Analytical')
        legend('boxoff')
        hold off

        % Add frames to list
        M(:,i) = getframe(gcf);
        if i == framemax
            pause(1.5)
        end

    end

    % Reset graphics properties
    clf reset
    axis off

    % Close animation
    close(figure(4))

end
```

B Physical aspects of the problem

The general equation for a cantilever beam under uniformly distributed load has been compared with the analytical solution of the given governing differential equation to obtain a ratio between the load applied and beam parameters.

Analytical solution,

$$u = -75x^2 + 50x^3 - 12.5x^4$$

General equation of Cantilever beam under uniformly distributed load,

$$u = -\frac{Wx^2}{24EI}(6L^2 - 4Lx + x^2)$$

Comparing the coefficients of both equations gives the following,

$$\frac{W}{EI} = 300 \qquad , \qquad L = 1$$

The stress and factor of safety at each node can be computed and their corresponding graphs can be plotted as a function of length by considering the material of the beam.

References

[1] *Gilbert Strang, Multigrid Methods.*
`https://math.mit.edu/classes/18.086/2006/am63.pdf`

[2] Gustaf Söderlind, Numerical Analysis, Lund University. *Introduction to Multigrid Methods, Chapter 8: Elements of Mutigrid Methods*

[3] Gunes Senel, The Ohio State University. *Multigrid Method*