Boundary Condition Application Using Finite Element Method

Paulo Henrique Brito de Souza

December 6, 2023

Contents

1 Introduction

This article will explore the application of the finite element method (FEM) to solve boundary value problems involving beam deflection. The article will compare the FEM solution with the analytical solution and discuss the advantages and limitations of each method. It will be presented an example of a beam fixed at one end and subjected to a concentrated load at the free end, and show how to obtain the deflection distribution using both FEM and analytical solutions.

2 Analytical Solution

The deflection of a beam that is fixed at one end and free at the other, also known as a cantilever beam. Please note that the following formulas are derived based on the assumptions of small deflections and linear elastic material behaviour. For large deflections or materials with non-linear behaviour, a more complex analysis may be required.

2.1 Elastic Curve

Elastic curve calculation for beams is a method to determine the deflection of a beam under a given load. The deflection of a beam affects its strength, stability and serviceability, so it is important to know how much the beam bends under the load. The elastic curve is the shape of the beam after it has deformed due to the load. The calculation involves applying the differential equation of the elastic curve, which relates the bending moment, the flexural rigidity and the curvature of the beam. The solution of the differential equation depends on the boundary conditions and the loading pattern of the beam. In this case, the value of interest will be the maximum deflection throughout the beam denoted by δ.

3 Numerical Solution

3.1 Finite Element Method

The finite element method (FEM) is a numerical technique for solving partial differential equations that arise from various physical and engineering problems. The FEM divides the domain of interest into smaller sub-domains, called finite elements, and approximates the solution by a polynomial function over each element. The FEM can handle complex geometries, boundary conditions, and material properties, and is widely used in fields such as structural mechanics, fluid dynamics, heat transfer, electromagnetic, and more. the polynomial functions used to approximate the results inside an element are called shape functions. They are mathematical expressions that describe the spatial variation of a quantity within an element of a finite element mesh and are used to interpolate the values of the unknowns at the nodes to any point within the element. Shape functions also determine the accuracy and convergence properties of the finite element method, having different forms depending on the type, order and geometry of the element. In this solution will be used a first order polynomial for the shape functions due the simple geometry of the problem

3.2 Matlab Code

This is a MATLAB function named ‘KFTransform‘. It takes three inputs: ‘Neu‘ (Neumann vector), ‘Dir‘ (Dirichlet vector), and ‘K‘ (Global stiffness matrix). The function initializes a zero matrix ‘k‘ of the same size as ‘Dir‘ and ‘K‘. It then enters a loop over the size of ‘Dir‘. If the ‘Dir‘th index of ‘Neu‘ is not zero, it throws an error. Otherwise, it modifies the ‘k‘ and ‘K‘ matrices. Finally, it calculates ‘Neumod‘ by subtracting the product of ‘k"‘ and the second column of ‘Dir‘ from ‘Neu‘, and assigns ‘K‘ to ‘Kmod‘. The function returns ‘Kmod‘ and ‘Neumod‘.

```
function [Kmod, Neumod] = KFTransform(Neu, Dir, K)
    k = zeros(size(Dir, 1), size(K, 2));
    for i = 1:size(Dir, 1)
        if Neu(Dir(i, 1)) ~= 0
            error('Make sure that there are
            not two types of BC at the same node.')
        end
        k(i, :) = K(Dir(i, 1), :);
        K(Dir(i, 1),:) = 0;
        K(:, Dir(i, 1)) = 0;
        K(Dir(i, 1), Dir(i, 1)) = 1;
    end
    Neumod = (Neu - k'*Dir(:, 2));
    Kmod = K;
```

Here’s a brief explanation of what each line does:

- ‘k = zeros(size(Dir, 1), size(K, 2));‘: Initializes ‘k‘ as a zero matrix with the same number of rows as ‘Dir‘ and the same number of columns as ‘K‘.
- ‘for i = 1:size(Dir, 1)‘: Starts a loop that iterates over each row of ‘Dir‘.
- ‘if Neu(Dir(i, 1)) = 0‘: Checks if the ‘Dir(i, 1)‘th element of ‘Neu‘ is not zero.
- ‘error('Make sure that there are not two types of BC at the same node.')‘: If the above condition is true, it throws an error.
- ‘k(i, :) = K(Dir(i, 1), :);‘: Assigns the ‘Dir(i, 1)‘th row of ‘K‘ to the ‘i‘th row of ‘k‘.

- ‘K(Dir(i, 1),:) = 0;‘: Sets the ‘Dir(i, 1)‘th row of ‘K‘ to zero.
- ‘K(:, Dir(i, 1)) = 0;‘: Sets the ‘Dir(i, 1)‘th column of ‘K‘ to zero.
- ‘K(Dir(i, 1), Dir(i, 1)) = 1;‘: Sets the ‘Dir(i, 1)‘th diagonal element of ‘K‘ to one.
- ‘Neumod = (Neu - k’*Dir(:, 2));‘: Calculates ‘Neumod‘ by subtracting the product of the transpose of ‘k‘ and the second column of ‘Dir‘ from ‘Neu‘.
- ‘Kmod = K;‘: Assigns ‘K‘ to ‘Kmod‘.