Project Numerical Analysis
Summer Semester 2022
Technische Universität Berlin

Pia Callmer, Henry Jacobson, Nicola Sabbadini, Alexander Quinlan

September 22, 2022

1 Introduction

1.1 Description

We use the Finite Element Method to model the bending of elastic beams in both static and dynamic cases. We start with a single cantilevered one-dimensional beam of a certain stiffness and under an applied force, and we use the finite element method to model the static bending of the beam. In a second step, we change the force over time and observe the beam's dynamic behavior using both the Newmark method and the Eigenvalue method. Finally, we extend this single one-dimensional beam to a two-dimensional network of connected beams.

2 Mathematical Model

2.1 1-dimensional case

In the 1-dimensional case the mathematical model of the static bending equation is given as follows

$$(EIw'')''(x) = q(x), \quad x \in \Omega \tag{2.1.1}$$

where the domain $\Omega = [0, L]$ is one-dimensional and L is the length of the beam. Specifically we have

- $w(x)$: height of the neutral axis (bending curve) at x
- $E(x)$: Young's module
- $I(x)$: area moment of inertia
- $q(x)$ load at x

In addition,

- $M(x) = EIw''(x)$: bending moment at x
- $Q(x) = -(EIw'')'(x)$: shear force at x

can be used for setting the boundary conditions, influencing the bending curve.

2.2 2-dimensional case

In the 2-dimensional case each edge is modelled very similar to the 1-dimensional case, where we in local coordinates to the edge have a displacement in both the y-direction and x-direction where the displacement in the x-direction is modelled using the wave-equation

$$\mu\ddot{v} - (EAv')' = f_1. \tag{2.2.1}$$

To get the displacement in global coordinates we simply rotate displacement according to the angle between the edge and the x-axis.

Since we have a network of connected beams, we need some way to determine geometric constraints. Depending on what type each node is, we have different constraints. For all types we have that the end points between connecting beams should always be touching, and the angle between beams should be constant (i.e. stiffness of angle). For fixed nodes one of the end points needs to be fixed in space for all times.

These constraints are mathematically modelled, where $\cdot$ is either the endpoint or startpoint of the edge, depending on the orientation between the two edges, as

- Stiffness of angle: $w_i'(\cdot, t) - w_j'(\cdot, t) = 0$

- Fixed bearing: $v_i(\cdot,t)=0$, $w_i(\cdot,t)=0$, $w_i'(\cdot,t)$.
- Movable bearing in direction $\vec{m}$:

$$\left(\begin{bmatrix}\cos(\pi/2) & -\sin(\pi/2)\\ \sin(\pi/2) & \cos(\pi/2)\end{bmatrix}\vec{m}\right)^T\begin{bmatrix}\cos(\phi_i)v_i(\cdot,t)-\sin(\phi_i)w_i(\cdot,t)\\ \sin(\phi_i)v_i(\cdot,t)+\cos(\phi_i)w_i(\cdot,t)\end{bmatrix}=0 \tag{2.2.2}$$

- Linking condition:

$$\cos(\phi_i)v_i(\cdot,t)-\sin(\phi_i)w_i(\cdot,t)-\cos(\phi_j)v_j(\cdot,t)+\sin(\phi_j)w_j(\cdot,t)=0 \tag{2.2.3}$$
$$\sin(\phi_i)v_i(\cdot,t)+\cos(\phi_i)w_i(\cdot,t)-\sin(\phi_j)v_j(\cdot,t)-\cos(\phi_j)w_j(\cdot,t)=0 \tag{2.2.4}$$

2.3 Dynamic case

In the case of the displacment, force and boundary condition being functions of time we have that

$$\begin{aligned} w &= w(x,t)\\ q &= q(x,t)\\ Q_{0,L} &= Q_{0,L}(t)\\ M_{0,L} &= M_{0,L}(t)\end{aligned}$$

The differential equation that this setup results in is given by the time dependent bending equation,

$$\mu\ddot{w}+(EIw'')''=q \tag{2.3.1}$$

We will look at the simple case of initial condition being the solution to the static case for some $q=q(x)$ and boundary conditions independent of time and the load being set to 0 at $t=0$. This will incur vibrations in the beam/framework.

3 Numerical Methods

3.1 Spatial Discretization

To solve the static bending equation with the Finite Element Method we first find the weak formulation of the problem, then apply the Galerkin reduction, insert the boundary conditions and solve the resulting linear equation system.

Our trial and function space V includes piecewise twice differentiable functions $V := C^{2,p}(\Omega) = \{\phi : \Omega \longrightarrow \mathbb{R} | \phi \in C(\Omega), \phi' \in C(\Omega), \phi'' \in C^p(\Omega)\})$, since we need to consider the fourth order derivative of $w(x)$. We begin by integrating the static bending equation 2.1.1 over the domain Ω and multiplying with the test function $\psi(x) \in V$

$$\int_\Omega (EIw'')''(x)\psi(x)dx = \int_\Omega q(x)\psi(x)dx, \quad \forall \psi \in V$$

Applying partial integration twice then yields

$$\begin{aligned}\int_\Omega (EIw'')''\psi dx &= -\int_\Omega (EIw'')'\psi' dx + [(EIw'')'\psi]_\Omega \\ &= \int_\Omega EIw''\psi'' dx + [(EIw'')'\psi]_\Omega - [EIw''\psi']_\Omega \\ &= \int_\Omega EIw''\psi'' dx \underbrace{Q(L)\psi(L) - Q(0)\psi(0) + M(L)\psi(L) - M(0)\psi(0)}_{b(\psi)}\end{aligned}$$

$$\iff \quad \underbrace{\int_\Omega EIw''\psi'' dx}_{a(w,\psi)} = \underbrace{\int_\Omega q\psi dx - b(\psi)}_{F(\psi)}$$

Now we have the weak formulation to to find w, s.t. $a(w,\psi) = F(\psi) \quad \forall \psi \in V$. To reduce the problem to a finite dimension we use the Galerkin substitution of V for V_n, spanned by n basis functions ϕ_i, so that w becomes $w_n(x) = \sum_{k=1}^n \alpha_k \phi_k(x)$ and ψ_j becomes $\phi_j, \forall \quad j \in \{1, ..., n\}$:

$$\sum_k \alpha_k \left(\int_\Omega (EI\phi_k''\phi_j'')dx \right) = \int_\Omega q\phi_j dx + b(\psi) \quad \forall j, \quad j,k \in \{1, ..., n\}$$

Not sure about this part since we have cubic basis functions: The basis functions are cubic on the elements $\Omega_k = [x_k, x_{k+1})$., choosen s.t. $\phi_i(x_j) = \delta_{ij}$ (not sure here). Hence on each element we have

$$\sum_k \alpha_k \int_{\Omega_i} (EI\phi_k''\phi_j'')dx = \sum_k \int_{\Omega_i} q(x)\phi_j dx + b(\psi) \quad \forall j, \quad j,k \in \{1, ..., n\} \cap \{k, k+1, k?\}$$

Each element of the stiffness matrix $\mathbf{S}$ is calculated by

$$s_{jk} = \int_0^L E(x)I(x)\phi_k''\phi_j''.$$

TODO: Gaussian quadrature etc.
In the static case, the resulting system

$$\mathbf{S_e u = q_e}$$

is solved and the solution passed to the cubic basis functions, which are used for the ansatz space V_h. Using piecewise polynomials of degree 3 is necessary, since the static bending equation 2.1.1 is of order 4, s.t. the weak formulation requires a twice differentiable basis function to compute the stiffness matrix. The ansatz space V_h with $h = \frac{L}{n-1}$, $x_i = h(i-1), i = 1, ..,$ and $n \leq 2$ is defined as

$$V_h = \{\phi \in V | \phi_{]x_i, x_{i+1}[}, i = 1, .., n-1\}$$

where each function $\phi_i \in V_h$ is uniquely defined by $u_{2i-1} = \phi(x_i)$ and $u_{2i} = \phi'(x_i)$. As a consequence each function can be determined by the weighted linear combination of the basis functions $\phi_i \in V_h$

$$\phi = \sum_{k=1}^{2n} u_k \phi_k = \sum_{k=1}^{n} (u_{2i-1}\phi_{2i-1} + u_{2i}\phi_{2i}).$$

The basis functions ϕ_i with $i = 2, .., n-1$ are defined as follows

$$\phi_{2i-1}(x) = \begin{cases} \bar{\phi_3}(\frac{x-x_{i-1}}{h}) & x \in [x_{i-1}, x_i] \\ \bar{\phi_1}(\frac{x-x_i}{h}) & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \quad \phi_{2i}(x) = \begin{cases} h\bar{\phi_4}(\frac{x-x_{i-1}}{h}) & x \in [x_{i-1}, x_i] \\ h\bar{\phi_2}(\frac{x-x_i}{h}) & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise.} \end{cases}$$

At the boundary elements, we get

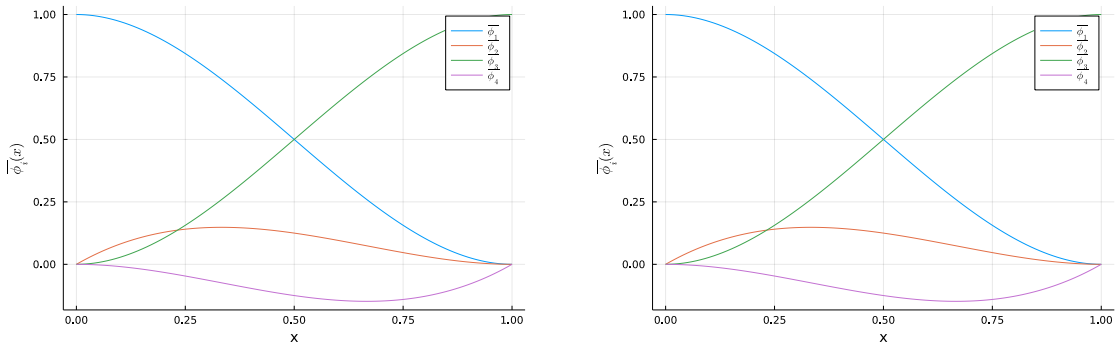
$$\phi_1(x) = \begin{cases} \bar{\phi_1}(\frac{x}{h}) & x \in [0,h] \\ 0 & \text{otherwise} \end{cases} \quad \phi_2(x) = \begin{cases} h\bar{\phi_2}(\frac{x}{h}) & x \in [0,h] \\ 0 & \text{otherwise} \end{cases}$$

and

$$\phi_{2n-1}(x) = \begin{cases} \bar{\phi_3}(\frac{x-x_{n-1}}{h}) & x \in [x_{n-1},L] \\ 0 & \text{otherwise} \end{cases} \quad \phi_{2n}(x) = \begin{cases} h\bar{\phi_4}(\frac{x-x_{n-1}}{h}) & x \in [x_{n-1},L] \\ 0 & \text{otherwise.} \end{cases}$$

The four form functions $\bar{\phi_i}, i = 1,..,4$ used for the basis functions are polynomial functions of degree 3 and determined by

$$\bar{\phi_1}(\xi) = 1 - 3\xi^2 + 2\xi^3 \tag{3.1.1}$$
$$\bar{\phi_2}(\xi) = \xi(\xi-1)^2 \tag{3.1.2}$$
$$\bar{\phi_3}(\xi) = 3\xi^2 - 2\xi^3 \tag{3.1.3}$$
$$\bar{\phi_4}(\xi) = \xi^2(\xi-1). \tag{3.1.4}$$
$$\tag{3.1.5}$$



(a) Form functions

(b) Basis functions, TODO

Figure 1: Piecewise Cubic form- and basisfunctions of the ansatz space V_h.

3.2 Time Discretization

Spatial discretization for the dynamic case is very similar to the static case. Doing what we did in the static case, with multiplying by a test function and integrating, we get

$$\int_0^L \mu\ddot{w}\phi + \int_0^L EIw''\phi'' = \int_0^L q\phi + b \tag{3.2.1}$$

where b represents the boundary conditions. Yet again we do a Galerkin Ansatz, with time dependent basis functions of course, so that we in addition to the stiffness matrix from before, get the mass matrix

$$M = \begin{bmatrix} m_{11} & \dots & m_{1N} \\ \vdots & & \vdots \\ m_{N1} & \dots m_{NN} & \end{bmatrix}, \qquad m_{jk} = \int_0^L \mu\phi_j\phi_k dx \tag{3.2.2}$$

which is a part of the system (TODO: How to get here?)

$$\underbrace{\begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix}}_{M_e} \ddot{\boldsymbol{x}} + \underbrace{\begin{bmatrix} S & C \\ C^T & 0 \end{bmatrix}}_{S_e} \boldsymbol{x} = \boldsymbol{f} \tag{3.2.3}$$

Here $\boldsymbol{x}$ is a vector of both displacements and constrained forces, and $\boldsymbol{f}$ is the vector of forces acting on the system.

3.3 System Solution

3.4 Eigenvalue method to solve the analytic bending equation

A different approach to solve the dynamic bending equation is to use the eigenvalue method. In this case we want to compute the solution as a superposition of standing waves, also called eigenmodes. The eigenfunctions, i.e. eigenvectors, then respresent the shape of the waves and the wave frequency is defined by the eigenfrequency. From the Fourier analysis of the initial conditions we can get the amplitude of the waves. When we use the eigenvalue method we assume no external forces or momentums and just homogeneous boundary conditions.

3.4.1 Eigenvalue Analytic Solution

To solve the dynamic bending equation of a uniform unloaded Bernoulli beam

$$\mu\ddot{w}(x,t) + EIw^{(4)}(x,t) = 0 \quad w : [0,L] \times \mathbb{R} \longrightarrow \mathbb{R}f \tag{3.4.1}$$

where $\mu, E, I, L > 0$, analytically we are looking for a solution of the form of standing waves

$$w(x,t) = \sigma(t)w(x) \tag{3.4.2}$$

, where $\sigma(t) = r\sin(\omega t - \phi)$ with $r, \phi \in \mathbf{R}$ and $\omega > 0$. Here, $w(x)$ represents the shape of the wave and $\sigma(t)$ the oscillating amplitude. If we insert (3.4.2) into the bending equation we get

$$-\mu\omega^2\sigma(t)w(x) + EI\sigma(t)w^{(4)}(x) = 0$$

, where $\ddot{\sigma}(t) + \omega^2\sigma(t) = 0$ is used to substitute $\ddot{\sigma}(t)$. This equation originates from the reformulation of $\sigma(t)$ to $\sigma(t) = \alpha\cos(\omega t) + \frac{\beta}{\omega}\sin(\omega t)$, which is the solution to $\ddot{\sigma}(t) + \omega^2\sigma(t) = 0$. The division by $\sigma(t)$ and rearranging yields

$$w^{(4)}(x) = \underbrace{\left(\frac{\omega\mu}{EI}\right)^{1/4}}_{=:\kappa^4} w(x) \tag{3.4.3}$$

This can be interpreted as an eigenvalue equation for $\frac{d^4}{dx^4}$, where κ^4 is the eigenvalue and $w(x)$ the eigenfunction. The general solution of (3.4.3) has the form

$$w(x) = A\cosh(\kappa x) + B\sinh(\kappa x) + C\cos(\kappa x) + D\sin(\kappa x), \quad A, B, C, D \in \mathbb{R}. \tag{3.4.4}$$

For the parameters we get $\kappa > 0$ and the frequency

$$\omega = \frac{EI}{\mu}\kappa^4.$$

By imposing boundary conditions and differentiating (3.4.4), we can solve for the unknowns A, B, C, D.

As use-case we look at the unloaded cantilever beam, which implies the following boundary conditions

$$w(0) = w'(0) = w''(L) = w'''(L).$$

The cantilever is fixed on the left hand side and no bending moment or load are assumed at the free right side of the beam. Inserting those boundary conditions and many other steps (TODO) consequently leads to

$$w(x,t) = \sum_{j=1}^{\infty} \left(\langle w_j, w_0 \rangle \cos(\omega_j t) + \frac{\langle w_j, \dot{w_0} \rangle}{\omega_j} \sin(\omega_j t) \right) w_j(t)$$

3.4.2 Eigenvalue Numerical Solution

The DAE of the dynamic bending equation has the form

$$\underbrace{\begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix}}_{M_e} \begin{bmatrix} \ddot{w} \\ \ddot{\mu} \end{bmatrix} + \underbrace{\begin{bmatrix} S & C \\ C^T & 0 \end{bmatrix}}_{S_e} \begin{bmatrix} w \\ \mu \end{bmatrix} = 0, \tag{3.4.5}$$

where we assume that $M, S \in \mathbb{R}^{N \times N}$ is symmetric positive definite and $S_e \in \mathbb{R}^{(N+K)\times(N+K)}$ is invertible.
Similar to the analytical approach, we want to find solutions of the form

$$\begin{bmatrix} w(t) \\ \mu(t) \end{bmatrix} = \sigma(t) \begin{bmatrix} w_0 \\ \mu_0 \end{bmatrix}, \quad \sigma(t) \in \mathbb{R}, \begin{bmatrix} w_0 \\ \mu_0 \end{bmatrix} \in \mathbb{R}^{N+K} \backslash \{0\}, \quad \sigma \equiv 0 \tag{3.4.6}$$

Now we define $A := S_e^{-1} M_e$.
Proposition 1?

4 Implementation

We wrote our program in the Julia programming language, due to its design as a high-performance programming language designed for scientific computing. We also made a simple graphical interface (GUI) in Python that we use to generate interesting frameworks. We divided our code into the following 8 modules:

- Beam1D.jl: Underlying solver for the single 1-dimensional case. Given a set of boundary conditions, this creates the system of matrices to solve, and has methods to solve this system in various ways.
- single_static.jl: Sets up the problem for the static single-beam case, and calls Beam1D.jl to solve it.
- single_dynamic_newmark.jl: Sets up the problem for the dynamic single-beam case, and calls Beam1D.jl to solve it using the Newmark method.
- single_dynamic_eigen.jl: Sets up the problem for the dynamic single-beam case, and calls Beam1D.jl to solve it using the eigenvalue method.
- construct_framework.py: GUI to create *.ne* files, which we load into our framework solvers.
- visualize_framework.py: GUI to create *.ne* files, which we load into our framework solvers.
- visualize_framework.jl: Takes one of the generated .ne files to visualize it with each node colored corresponding to the type of the node.
- Beam2D.jl: Underlying solver for a framework of beams. Given a framework *.ne* file, it sets up the system of matrices to solve.
- framework_static.jl: Sets up the problem for the static framework, and calls Beam2D.jl to solve it.

- framework_dynamic.jl: Sets up the problem for the dynamic framework, and calls Beam2D.jl to solve it. Can use either the Newmark or the Eigenvalue method.

4.1 Beam1D

Our *Beam1D.jl* file takes a set of problem parameters (boundary conditions, the grid for our solution, and functions describing the beam's stiffness and the force placed on the beam), and it calculates the system of the mass matrix M_e, the stiffness matrix S_e, and the constraint matrix q_e. It generates these by the finite element method, using cubic basis functions specified above and a 3 point Gaussian quadrature.

4.2 Beam2D

The *Beam2D.jl* file instead takes in a .ne (node-edge) file that specifies the problem structure using coordinates of nodes and node connections, as well as the type of each node (Free/Fixed/Force/Movable). The file format is

```
NODES
<x-coord> <y-coord>
<x-coord> <y-coord>
...
EDGES
<node 1> <node 2> [optional: params=[<fnc>,<fnc>,<fnc>,<fnc>],gp=x]
<node 1> <node 2>
...
TYPES
1 <FIXED/FREE/FORCE/MOVABLE>
2 <FIXED/FREE/FORCE/MOVABLE>
3 <FIXED/FREE/FORCE/MOVABLE>
...
PARAMETERS
E <fnc>
I <fnc>
A <fnc>
mu <fnc>
```

where the optional `params` defines specific material properties for that edge, and the optional `gp` defines a specific amount of gridpoints at that edge. A short example of this can be found in the appendix.

Using a file in this format the function `Problem` creates a new instance of the struct. This struct has three members `nodes`, `edges`, and the shape of the problem. The `nodes` and `edges` members are vectors of the structs `Node` and `Edge`, which holds information about the specific edge and node.

4.2.1 Creating constraint matrix

Before creating the constraints matrix we need to check how many columns it needs to have, i.e. the number of constraints. That is dependent on the types of nodes and how many connecting edges each one has. So we therefore loop through the nodes, and for `FIXED` nodes we have three constraints per connecting edge, for `MOVABLE` nodes we have one bearing constraint for the 'first' edge and then three constraints for the remaining connecting edges. Lastly, for nodes of type `FORCE` and `FREE` we have three conditions per *pair* of connecting edges, so we add those as well.

For creating the constraint matrix we loop through all the nodes again and depending on the type of the node we add the specific values to the matrix in the correct place using the procedure described

in section 2. Specifically we make sure to use the correct side of the edge connected to the node under inspection, depending on the orientation of the node.

At the same time as the construction of the constraint matrix we also construct the constraint forces vector, of which the values are gotten from the nodes which are of type `FORCE`.

When initializing the constraints for a node between two edges we need to keep track of if the edge start or end at the node, so that our indexing in the matrix will be correct. To know if it an edge starts or ends at a node we have a member of the `Node` class called `number`, so that we can compare the first node of the edge with a node to see if they are the same.

To get the specific orientation for an edge at a node we keep track of the index in the problem matrix for where the edge has it's first element, so to then add the constraints to the constraint matrix from that index if the edge 'starts' at the node or the last index, which would be first index plus the number of elements defined in the edge minus one to get the last index.

In the source code there are comments explaining most of the functions we use to set up the problem.

5 Results

5.1 Analytical Solution

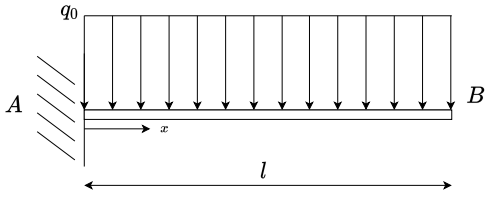
In this section we compare the analytical results of several load cases to our numerical results. For all cases a constant Youngs module $E(x) = E$ and a constant area moment or inertia $I(x) = I$ is assumed. The numerical result is computed with 20 grid points and the length of the beam is $l = 1$. Fixing the beam on the left side leads to the boundary conditions

$$w(0) = 0, \quad w'(x) = 0. \tag{5.1.1}$$

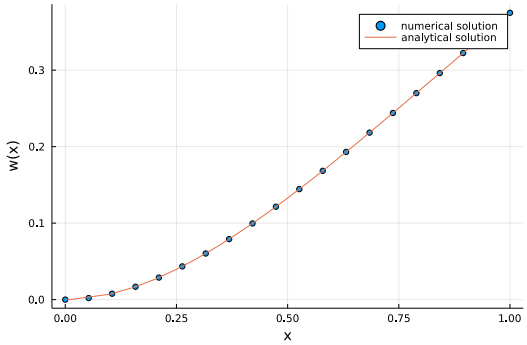
The first case is a simple constant load case, depicted in figure 2a, where we choose $q_0 = 3$. By integrating the static bending equation 2.1.1 four times and using the boundary conditions 5.1.1 and $Q(l) = 0$ and $M(l) = 0$ to find the integration constants we get the bending curve

$$EIw(x) = \frac{q_0 l^4}{24}\left(6\xi^2 - 4\xi^3 + \xi^4\right)$$

where $\xi = \frac{x}{l}$. Looking at the bending curve $w(x)$ in figure 2b, the analytical result coincides with the numerical results and gives an L_2 error of TODO.



(a) Example visualisation



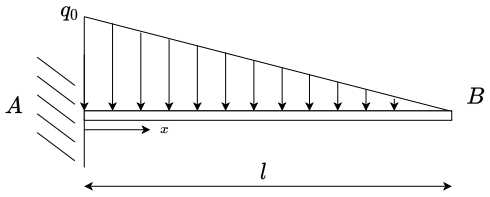
(b) Numerical result and the analytical result

Figure 2: Constant load with $q_0 = 3$ and the boundary conditions $w(0) = 0$, $w'(0) = 0, Q(L) = 0$ and $M(l) = 0$.

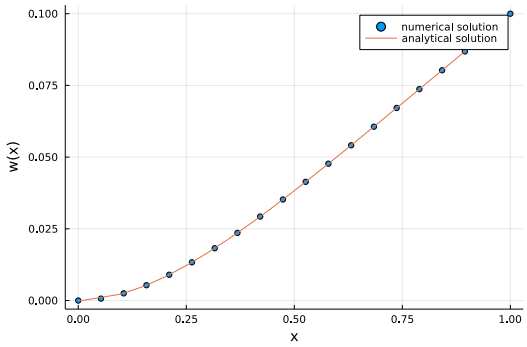
The second case is displayed in figure 3a, where the load is decreasing with $q(x) = 3 - 3x$ and the same boundary conditions as in the first constant load example. The analytic result gives

$$EIw(x) = \frac{q_0 l^4}{120}\left(10\xi^2 - 10\xi^3 + 5\xi^4 - \xi^5\right)$$

and is displayed in figure 3b together with the numerical result.



(a) Example visualisation



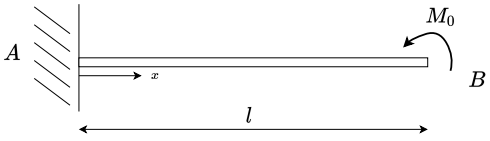
(b) Numerical result and the analytical result

Figure 3: Decreasing load with $q(x) = 3 - 3x$ and the boundary conditions $w(0) = 0$, $w'(0) = 0, Q(L) = 0$ and $M(l) = 0$.

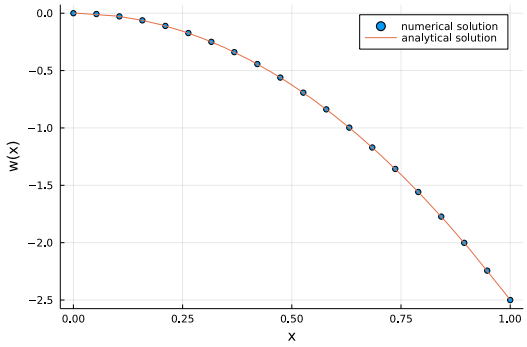
The third case has no load, but a momentum $M_0 = 5$ at the end of the beam. Here, the boundary conditions 5.1.1 together with $Q(l) = 0$ and $M(l) = M_0$ results in the analytical solution

$$EIw(x) = \frac{M_0 l^2}{2}\xi^2.$$

The numerical result next to the analytical one is plotted in figure 4b and has the same result at the grid points with an L_2 error of $e_{L_2} = 7 \cdot 10^{-11}$.



(a) Example visualisation



(b) Numerical result and the analytical result

Figure 4: Momentum at the end of the beam and the boundary conditions $w(0) = 0$, $w'(0) = 0, Q(L) = 0$ and $M(l) = 5$.

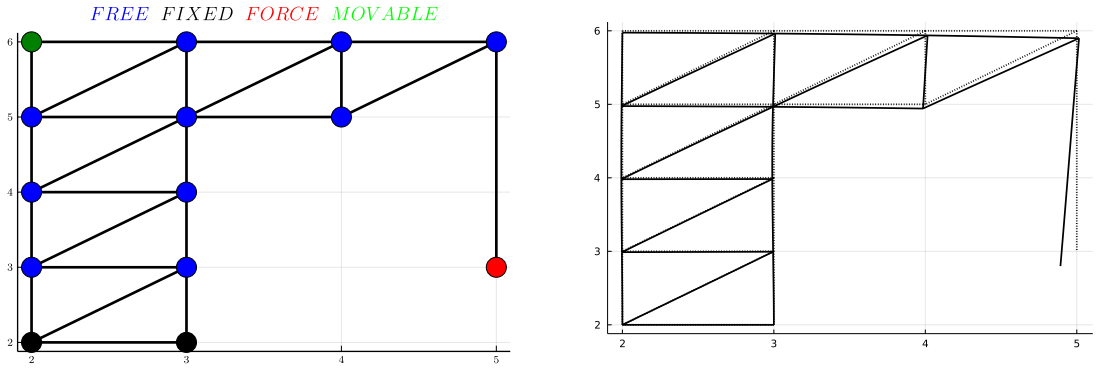


Figure 5: Simulation of building crane

5.2 Numerical solution

5.2.1 1-dimensional case

5.2.2 2-dimensional case

Pictured in 5 is a framework of beams supposed to depict a building crane with a rope attached to it at the end, where the rope has a smaller value of Young's module to simulate a bendier beam.

5.3 Validation

5.4 Parameter Study

6 Conclusion