Rakenteiden Mekaniikka (Journal of Structural Mechanics)
Vol. 52, No 3, 2019, pp. 160–176
https://rakenteidenmekaniikka.journal.fi/index
https://doi.org/10.23998/rm.76193

# JuliaFEM beam element implementation

Jukka Aho[1], Ville Jämsä, Teemu Kuivaniemi, Niclas Liljenfeldt and Tero Frondelius

**Summary.** This article describes implementations of beam elements to JuliaFEM. The theory is briefly introduced, and the usage of beam elements is introduced with a usage example that involves a natural frequency calculation of a formula race car frame. The calculation results were compared to results from a commercial program, and their consistency is excellent.

*Key words:* beam element, Euler-Bernoulli, JuliaFEM

*Received 30 October 2019. Accepted 30 April 2019. Published online 12 September 2019.*

## Introduction

This article describes implementations, theory, and usage of beam elements in JuliaFEM. JuliaFEM is an open-source finite element method solver written in the Julia programming language, see Frondelius et al. [1]. Julia itself is a programming language designed especially for numerical computing, including features like just-in-time code compilation and multiple dispatch of generic functions [2].

Basically, beam element can be thought of as a reduction of a full continuum model, which still preserves the most important characteristics of the underlying physical phenomena. In academia, beams have often been described in two dimensions to introduce the basic concepts of finite element method to students. However, since calculation models needed in practical simulation work are usually described in three dimensions, the physics of the beam elements must be also described in three dimensions. JuliaFEM also has more general model reduction techniques which are described in article written by Rapo et al. [3].

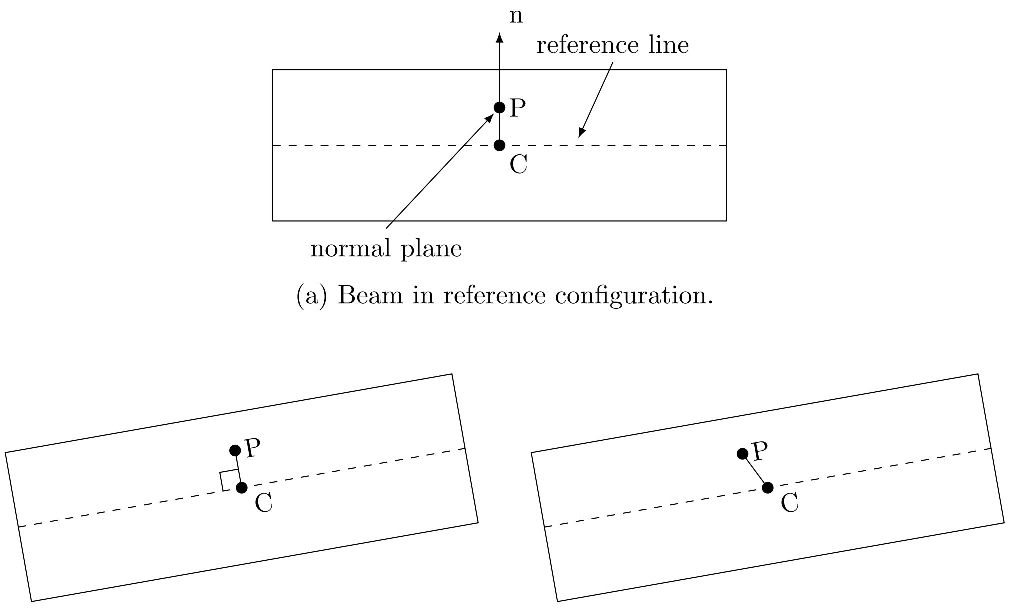
In this article, we introduce a practical example of a natural frequency analysis of a formula race car frame. The obtained results are compared to the corresponding results of the commercial software to validate the implementation. Similar comparison of JuliaFEM natural frequency solver with continuum elements is done by Rapo et al. [4], where the model also had contacts, see Aho [5]. A more complicated use case, where beam elements are used in the optimization loop to find the fuel pipe route that minimizes the given cost function, is presented by Rapo et al. [6]. JuliaFEM also has preliminary support for shells, see Niemi et al. [7]. For a comprehensive history of Wärtsilä's industrial calculations, see

[1]Corresponding author. ahojukka5@gmail.com

Frondelius et al. [8]. Typical applications of beam elements in industrial simulation is to model pipes and bolts in locations where there are no high requirements for the accuracy of the results.

Theory of beams is given only briefly. For a more comprehensive description about the theory of beam, see e.e. Crisfield [9], Belytschko [10], Oñate [11], and Zienkiewich [12]. Beam implementations has been considered in this journal earlier by Paavola and Salolainen [13]. For the very first FEM-based beam element implementations, see articles written by Kapur [14], Archer [15], Thomas et al. [16], Reissner [17] and Cowper [18].

## Theory



(a) Beam in reference configuration.

(b) Euler-Bernoulli beam theory.

(c) Timoshenko beam theory.

Figure 1: The difference between Euler-Bernoulli and Timoshenko beam theories: in Euler-Bernoulli beam theory, the planes normal to the midline are assumed to remain plane and normal, whereas in Timoshenko beam theory the planes normal to the midline are assumed to remain plane, but not necessarily normal.

Two types of theories are widely used in the development of beam elements. The most important kinematic assumption concerns the motion of the normals to the midline of the beam. In Euler-Bernoulli beam theory, the planes normal to the midline are assumed to remain plane and normal, whereas in Timoshenko beam theory the planes normal to the midline are assumed to remain plane, but not necessarily normal see figure 1. Timoshenko beam theory can be seen as an extension to Euler-Bernoulli beam theory, allowing the effect of transverse shear deformation. Timoshenko beam theory relaxes the normality assumption, and the motions of an Euler–Bernoulli beam are a subset of the motions allowed by Timoshenko beam theory [10]. To clarify this, let us study the equations of Euler-Bernoulli beam theory and Timoshenko beam theory in a quasistatic situation. The governing equations of Timoshenko beam are the following coupled system of ordinary differential equations:

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(EI\frac{\mathrm{d}\varphi}{\mathrm{d}x}\right) = q\left(x\right), \tag{1}$$

$$\frac{\mathrm{d}w}{\mathrm{d}x} = \varphi - \frac{1}{\kappa AG}\frac{\mathrm{d}}{\mathrm{d}x}\left(EI\frac{\mathrm{d}\phi}{\mathrm{d}x}\right), \tag{2}$$

where $L$ is the length of the beam, $A$ is the cross section area, $E$ is the elastic modulus, $G$ is the shear modulus, $I$ is the second moment of area, $q\left(x\right)$ is distributed load and $\kappa$ is called Timoshenko shear coefficient. Now, if

$$\frac{EI}{\kappa L^2 AG} \ll 1, \tag{3}$$

the last term of equation (2) can be neglected, thus resulting

$$\frac{\mathrm{d}\varphi}{\mathrm{d}x} = \frac{\mathrm{d}^2 w}{\mathrm{d}x^2}, \tag{4}$$

and substituting this result back to equation (1) reveals the governing equations of Euler-Bernoulli beam theory:

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(EI\frac{\mathrm{d}^2 w}{\mathrm{d}x^2}\right) = q\left(x\right). \tag{5}$$

The main implication of the kinematic assumptions are that when using Euler-Bernoulli beam theory, transverse shear vanishes. For this reason, Euler-Bernoulli beam theory works well only in the case of slender beams, whereas Timoshenko beam theory, also known as shear beam theory, works well also in the case of thick beams.

Euler-Bernoulli beam has only single dependent variable whereas the Timoshenko beam has two dependent variables. Because second derivative of velocity appears in the expression for the rate-of-deformation, the velocity field must be at least $C^1$ for Euler-Bernoulli beam. That is, the shape functions of Euler-Bernoulli beam needs higher continuity requirements than Timoshenko beam, where it is enough to have $C^0$ continuity.

According to Belytschko [10], the continuity requirement is the biggest disadvantage of Euler-Bernoulli and Kirchhoff-Love theories, since $C^1$ approximation is difficult to construct in multi-dimensions. In the study of plates and shells, theory corresponding to Euler-Bernoulli beam theory is Kircchoff-Love theory, having similar assumptions, and theory of Timoshenko can be replaced with the theory of Reissner-Mindlin, respectively.

In the derivation of beam element, we assume that beam is straight with the axis of definition in the $x$ direction and the cross-section $A$ in the $yz$ plane. We also assume that the primary stress components are the normal stress $\sigma_x$ and shear stresses $\tau_{xy}$ and $\tau_{xz}$ acting on each cross-section. The rest of the stress components are either neglected or included as boundary loads to the beam.

### *Equilibrium equations*

In the derivation of equilibrium equations, conservation of linear momentum and angular momentum must be considered. First we study the global form of a balance of linear momentum in the cross section $A$ of the beam:

$$\int_A \left(\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \boldsymbol{b}\right)\,\mathrm{d}A = \frac{\mathrm{d}}{\mathrm{d}t}\int_A \rho\dot{\boldsymbol{u}}\,\mathrm{d}A, \tag{6}$$

which, given using index notation, is

$$\int_A \left[\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + b_x\right] \mathrm{d}A = \int_A \rho \ddot{u}\, \mathrm{d}A, \tag{7}$$

$$\int_A \left[\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + b_y\right] \mathrm{d}A = \int_A \rho \ddot{v}\, \mathrm{d}A, \tag{8}$$

$$\int_A \left[\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + b_z\right] \mathrm{d}A = \int_A \rho \ddot{w}\, \mathrm{d}A. \tag{9}$$

The main assumption is that the the cross-section is either constant of varies slowly enough along the $x$ direction so that we can use

$$\int_A \frac{\partial f}{\partial x}\, \mathrm{d}A \approx \frac{\partial}{\partial x}\int_A f\, \mathrm{d}A, \qquad \int_A \frac{\partial f}{\partial y}\, \mathrm{d}A = \oint_S f n_y\, \mathrm{d}S, \qquad \int_A \frac{\partial f}{\partial z}\, \mathrm{d}A = \oint_S f n_z\, \mathrm{d}S, \tag{10}$$

where $S$ is the perimeter of the cross section. Thus we can write

$$\frac{\partial}{\partial x}\int_A \sigma_x\, \mathrm{d}A + \oint_S \tau_{yx} n_y\, \mathrm{d}S + \oint_S \tau_{zx} n_z\, \mathrm{d}S + \int_A b_x\, \mathrm{d}A = \rho A \ddot{u}, \tag{11}$$

$$\frac{\partial}{\partial x}\int_A \tau_{xy}\, \mathrm{d}A + \oint_S \sigma_y n_y\, \mathrm{d}S + \oint_S \tau_{zy} n_z\, \mathrm{d}S + \int_A b_y\, \mathrm{d}A = \rho A \ddot{v}, \tag{12}$$

$$\frac{\partial}{\partial x}\int_A \tau_{xz}\, \mathrm{d}A + \oint_S \tau_{yz} n_y\, \mathrm{d}S + \oint_S \sigma_z n_z\, \mathrm{d}S + \int_A b_z\, \mathrm{d}A = \rho A \ddot{w}. \tag{13}$$

Rearranging terms allows us to write the equilibrium of linear momentum in a compact form

$$\frac{\partial P}{\partial x} + q_x = \rho A \ddot{u}, \qquad \frac{\partial S_y}{\partial x} + q_y = \rho A \ddot{v}, \qquad \frac{\partial S_z}{\partial x} + q_z = \rho A \ddot{w}, \tag{14}$$

where

$$P = \int_A \sigma_x\, \mathrm{d}A, \qquad q_x = \int_A b_x\, \mathrm{d}A + \oint_S (\tau_{yx} n_y + \tau_{zx} n_z)\, \mathrm{d}S, \tag{15}$$

$$S_y = \int_A \tau_{xy}\, \mathrm{d}A, \qquad q_y = \int_A b_y\, \mathrm{d}A + \oint_S (\sigma_y n_y + \tau_{zy} n_z)\, \mathrm{d}S, \tag{16}$$

$$S_z = \int_A \tau_{xz}\, \mathrm{d}A, \qquad q_z = \int_A b_z\, \mathrm{d}A + \oint_S (\tau_{yz} n_y + \sigma_z n_z)\, \mathrm{d}S. \tag{17}$$

Here, $P$ is axial force resultant and $S_y$ and $S_z$ are transverse shear force resultants, and $q_x$, $q_y$ and $q_z$ are loads in $x$, $y$ and $z$ directions, respectively.

In the same way than with the linear momentum, we can write the global form of the balance of angular momentum in the cross section of the beam as

$$\int_A \boldsymbol{x} \times (\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \boldsymbol{b})\, \mathrm{d}A = \frac{\mathrm{d}}{\mathrm{d}t}\int_A \rho \boldsymbol{x} \times \dot{\boldsymbol{u}}\, \mathrm{d}A, \tag{18}$$

where $\boldsymbol{x}$ is some arbitrary point in cross section

$$\boldsymbol{x} = \begin{bmatrix} 0 \\ y \\ z \end{bmatrix}, \qquad \dot{\boldsymbol{u}} = \dot{\boldsymbol{\theta}} \times \boldsymbol{x}, \tag{19}$$

and $\dot{\boldsymbol{\theta}}$ is the angular velocity vector. We consider the right hand side first. By using vector identity $\boldsymbol{A} \times (\boldsymbol{B} \times \boldsymbol{C}) = (\boldsymbol{A} \cdot \boldsymbol{C})\, \boldsymbol{B} - (\boldsymbol{A} \cdot \boldsymbol{B})\, \boldsymbol{C}$, we can write the rate-of-change of the angular momentum in a form

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_A \rho \boldsymbol{x} \times \dot{\boldsymbol{\theta}} \times \boldsymbol{x} \, \mathrm{d}A = \frac{\mathrm{d}}{\mathrm{d}t} \int_A \rho \left[ (\boldsymbol{x} \cdot \boldsymbol{x})\, \dot{\boldsymbol{\theta}} - \left( \boldsymbol{x} \cdot \dot{\boldsymbol{\theta}} \right) \boldsymbol{x} \right] \mathrm{d}A, \tag{20}$$

which can be further expanded in Cartesian coordinate system to

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t} \int_A \rho \left[ (\boldsymbol{x} \cdot \boldsymbol{x})\, \dot{\boldsymbol{\theta}} - \left( \boldsymbol{x} \cdot \dot{\boldsymbol{\theta}} \right) \boldsymbol{x} \right] \mathrm{d}A \\
&= \frac{\mathrm{d}}{\mathrm{d}t} \int_A \rho \left( \begin{bmatrix} (y^2+z^2)\, \dot{\theta}_x \\ (y^2+z^2)\, \dot{\theta}_y \\ (y^2+z^2)\, \dot{\theta}_z \end{bmatrix} - \begin{bmatrix} 0 \\ \left( \dot{\theta}_y y + \dot{\theta}_z z \right) y \\ \left( \dot{\theta}_y y + \dot{\theta}_z z \right) z \end{bmatrix} \right) \mathrm{d}A \\
&= \begin{bmatrix} \int_A \rho \, (y^2+z^2) \; \mathrm{d}A \ddot{\theta}_x \\ \int_A \rho z^2 \, \mathrm{d}A \ddot{\theta}_y - \int_A \rho y z \, \mathrm{d}A \ddot{\theta}_z \\ \int_A \rho y^2 \, \mathrm{d}A \ddot{\theta}_z - \int_A \rho y z \, \mathrm{d}A \ddot{\theta}_y \end{bmatrix} = \begin{bmatrix} \rho J \ddot{\theta}_x \\ \rho I_y \ddot{\theta}_y - \rho I_{yz} \ddot{\theta}_z \\ \rho I_z \ddot{\theta}_z - \rho I_{yz} \ddot{\theta}_y \end{bmatrix},
\end{aligned} \tag{21}$$

where the definition of the components of the inertia tensor are

$$I_y = \int_A z^2 \, \mathrm{d}A, \qquad I_z = \int_A y^2 \, \mathrm{d}A, \qquad I_{yz} = \int_A yz \, \mathrm{d}A, \tag{22}$$

$$J = I_y + I_z = \int_A \left( y^2 + z^2 \right) \mathrm{d}A. \tag{23}$$

Writing the equation (18) using index notation and neglecting terms $\sigma_y$, $\sigma_z$and $\tau_{yz}$, (see also Zienkiewich and Taylor [12]), we finally end up to the following:

$$\int_A \left[ y \cdot \left( \frac{\partial \tau_{xz}}{\partial x} + b_z \right) - z \cdot \left( \frac{\partial \tau_{xy}}{\partial x} + b_y \right) \right] \mathrm{d}A = \rho J \ddot{\theta}_x, \tag{24}$$

$$\int_A \left[ z \cdot \left( \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + b_x \right) \right] \mathrm{d}A = I_y \ddot{\theta}_y - I_{yz} \ddot{\theta}_z, \tag{25}$$

$$\int_A \left[ -y \cdot \left( \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + b_x \right) \right] \mathrm{d}A = I_z \ddot{\theta}_z - I_{yz} \ddot{\theta}_y. \tag{26}$$

Rearranging terms allows us to write the equilibrium of angular momentum in a compact form

$$\frac{\partial T}{\partial x} + m_x = \rho J \ddot{\theta}_x, \tag{27}$$

$$\frac{\partial M_y}{\partial x} - S_z + m_y = \rho \left( I_y \ddot{\theta}_y - I_{yz} \ddot{\theta}_z \right), \tag{28}$$

$$\frac{\partial M_z}{\partial x} + S_y + m_z = \rho \left( I_z \ddot{\theta}_z - I_{yz} \ddot{\theta}_y \right), \tag{29}$$

where

$$T = \int_A (\tau_{xz} y - \tau_{xy} z) \, \mathrm{d}A, \qquad m_x = \int_A (y b_z - z b_y) \, \mathrm{d}A, \tag{30}$$

$$M_y = \int_A z \sigma_x \, \mathrm{d}A, \qquad m_y = \int_A z b_x \, \mathrm{d}A + \oint_S z \left( \tau_{yx} n_y + \tau_{zx} n_z \right) \mathrm{d}S, \tag{31}$$

$$M_z = -\int_A y \sigma_x \, \mathrm{d}A, \qquad m_z = -\int_A y b_x \, \mathrm{d}A - \oint_S y \left( \tau_{yx} n_y + \tau_{zx} n_z \right) \mathrm{d}S. \tag{32}$$

*Kinematics*

We assume that the cross-section of beam remains as plane in deformation process. Displacement field is expressed as

$$u_x(x,y,z) = u(x) + z\theta_y(x) - y\theta_z(x), \tag{33}$$
$$u_y(x,y,z) = v(x) - z\theta_x(x), \tag{34}$$
$$u_z(x,y,z) = w(x) + y\theta_x(x), \tag{35}$$

where $u$, $v$, $w$ are displacement of the axis defining the beam and $\theta_x$, $\theta_y$, $\theta_z$ are small rotation angles about the coordinate axes. Strains are calculated from displacement field using small strain theory,

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{36}$$

According to the 3D beam theory, strain components $\varepsilon_y$, $\varepsilon_z$ and $\varepsilon_{yz}$ vanish, $\varepsilon_y = \varepsilon_z = \varepsilon_{yz} = 0$. The nonzero strain components are

$$\varepsilon_x = \frac{\partial u_x(x,y,z)}{\partial x} = \frac{\partial u}{\partial x} + z\frac{\partial \theta_y}{\partial x} - y\frac{\partial \theta_z}{\partial x}, \tag{37}$$
$$\gamma_{xy} = \frac{\partial u_x(x,y,z)}{\partial y} + \frac{\partial u_y(x,y,z)}{\partial x} = \frac{\partial v}{\partial x} - \theta_z - z\frac{\partial \theta_x}{\partial x}, \tag{38}$$
$$\gamma_{xz} = \frac{\partial u_x(x,y,z)}{\partial z} + \frac{\partial u_z(x,y,z)}{\partial x} = \frac{\partial w}{\partial x} + \theta_y + y\frac{\partial \theta_x}{\partial x}. \tag{39}$$

The main assumption in Euler-Bernoulli beam theory is that rotations $\theta_y$ and $\theta_z$ coincide with the slopes of the neutral axis:

$$\theta_z = \frac{\partial v}{\partial x}, \qquad \theta_y = -\frac{\partial w}{\partial x}. \tag{40}$$

With this assumption, nonzero strain components are thus

$$\varepsilon_x = \frac{\partial u}{\partial x} - z\frac{\partial^2 w}{\partial x^2} - y\frac{\partial^2 v}{\partial x^2}, \qquad \gamma_{xy} = -z\frac{\partial \theta_x}{\partial x}, \qquad \gamma_{xz} = y\frac{\partial \theta_x}{\partial x}. \tag{41}$$

*Elastic constitutive relations*

Linear constitution relation is assumed:

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\varepsilon}, \tag{42}$$

which is expressed in component form as

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \tau_{xy} \\ \tau_{xz} \end{bmatrix} = \begin{bmatrix} E & 0 & 0 \\ 0 & G_y & 0 \\ 0 & 0 & G_z \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \gamma_{xy} \\ \gamma_{xz} \end{bmatrix} = \boldsymbol{D}\boldsymbol{\varepsilon}. \tag{43}$$

Constitutive matrix $\boldsymbol{D}$ can be deducted from the general 3D constitutive equation.

### *Weak form*

To derive a weak form, we define so-called generalized local strain vector $\hat{\boldsymbol{\varepsilon}}$ and its energetic conjugate, generalized stress vector $\hat{\boldsymbol{\sigma}}$. For a generalized local strain vector, we aim to write the nonzero strain tensor components $\varepsilon_x$, $\gamma_{xy}$ and $\gamma_{xz}$ in terms of first and second order partial derivatives of displacement field. Thus our relation looks like

$$\underbrace{\begin{bmatrix} \varepsilon_x \\ \gamma_{xy} \\ \gamma_{xz} \end{bmatrix}}_{\boldsymbol{\varepsilon}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & -z & -y & 0 \\ 0 & 1 & 0 & 0 & 0 & -z \\ 0 & 0 & 1 & 0 & 0 & y \end{bmatrix}}_{\boldsymbol{S}_1} \underbrace{\begin{bmatrix} \frac{\partial u}{\partial x} \\ 0 \\ 0 \\ \frac{\partial^2 w}{\partial x^2} \\ \frac{\partial^2 v}{\partial x^2} \\ \frac{\partial \theta}{\partial x} \end{bmatrix}}_{\hat{\boldsymbol{\varepsilon}}}. \tag{44}$$

In the same way, we define a generalized stress vector $\hat{\boldsymbol{\sigma}}$:

$$\underbrace{\begin{bmatrix} P \\ S_y \\ S_z \\ M_y \\ M_z \\ T \end{bmatrix}}_{\hat{\boldsymbol{\sigma}}} = \iint_A \begin{bmatrix} \sigma_x \\ \tau_{xy} \\ \tau_{xz} \\ z\sigma_x \\ -y\sigma_x \\ \tau_{xz}y - \tau_{xy}z \end{bmatrix} \mathrm{d}A = \iint_A \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ z & 0 & 0 \\ -y & 0 & 0 \\ 0 & -z & y \end{bmatrix}}_{\boldsymbol{S}_2} \underbrace{\begin{bmatrix} \sigma_x \\ \tau_{xy} \\ \tau_{xz} \end{bmatrix}}_{\boldsymbol{\sigma}} \mathrm{d}A. \tag{45}$$

By examining strain transformation matrix $\boldsymbol{S}_1$ and stress transformation matrix $\boldsymbol{S}_2$, it can be seen that $\boldsymbol{S}_1 = \boldsymbol{S}_2^{\mathrm{T}}$, which has importance in the following step. Internal virtual work can now be written as

$$\begin{aligned} \delta\mathcal{W}_{\text{int}} &= \iiint_V \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}\,\mathrm{d}V = \iiint_V \left(\boldsymbol{S}_1\delta\hat{\boldsymbol{\varepsilon}}\right)^{\mathrm{T}}\boldsymbol{\sigma}\,\mathrm{d}V = \iiint_V \delta\hat{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{\sigma}\,\mathrm{d}V \\ &= \iiint_V \delta\hat{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{S}_2\boldsymbol{\sigma}\,\mathrm{d}V = \int_L \delta\hat{\boldsymbol{\varepsilon}}^{\mathrm{T}}\left(\iint_A \boldsymbol{S}_2\boldsymbol{\sigma}\,\mathrm{d}A\right)\mathrm{d}x = \int_L \delta\hat{\boldsymbol{\varepsilon}}^{\mathrm{T}}\hat{\boldsymbol{\sigma}}\,\mathrm{d}x. \end{aligned} \tag{46}$$

By defining a kinematic matrix $\boldsymbol{B}$ such that $\hat{\boldsymbol{\varepsilon}} = \boldsymbol{B}\boldsymbol{u}$ and $\hat{\boldsymbol{\sigma}} = \hat{\boldsymbol{D}}\boldsymbol{\varepsilon}$, equation 46 can be written in a standard form

$$\delta\mathcal{W}_{\text{int}} = \delta\boldsymbol{u}^{\mathrm{T}} \cdot \int_L \boldsymbol{B}^{\mathrm{T}}\hat{\boldsymbol{D}}\boldsymbol{B}\,\mathrm{d}x\boldsymbol{u}. \tag{47}$$

A more detailed description about generalized quantities is given by Oñate [11].

### *Discretization and finite element approximation*

For finite element approximation, translations and rotations must be approximated by appropriate basis functions. The simplest 3D Euler-Bernoulli beam has two nodes. Linear $C^0$ continuous interpolation can be used to approximate the axial displacement $u_0$ and for the twist rotation $\theta_x$. A cubic Hermite $C^1$ continuous approximation must be used for $v_c$ and $w_c$. First we consider the $C^0$ continuous linear interpolation for axial displacement and twisting. Starting with a linear polynomial

$$p_1(\xi) = \alpha_0 + \alpha_1\xi = \begin{bmatrix} 1 & \xi \end{bmatrix}\begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}, \tag{48}$$

to find shape function $N_1(\xi)$, there exists coefficients $\alpha_0, \alpha_1$ such that

$$p_1(-1) = 1 \quad \wedge \quad p_1(1) = 0. \tag{49}$$

Substituting conditions (49) to equation (48), we get the following system of equations

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{50}$$

and solution for coefficients $\boldsymbol{\alpha}$ is

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \tag{51}$$

so the first linear shape function is

$$N_1(\xi) = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & \xi \end{bmatrix} = \frac{1}{2}(1 - \xi). \tag{52}$$

Using the same approach for $N_2(\xi)$: there exists coefficients $\alpha_0$ and $\alpha_1$, such that $p_1(-1) = 0$ and $p_1(1) = 1$, which results

$$N_2(\xi) = \frac{1}{2}(1 + \xi). \tag{53}$$

Therefore, linear interpolations for axial displacement and twisting are

$$u(\xi) = N_1(\xi) u_1 + N_2(\xi) u_2, \tag{54}$$

$$\theta_x(\xi) = N_1(\xi) \theta_{x_1} + N_2(\xi) \theta_{x_2}. \tag{55}$$

To construct $C^1$ continuous interpolation polynomials, similar approach is used. Because now the function and its first derivatives needs to be continuous, four conditions needs to be matched, thus the polynomial is cubic:

$$p_3(\xi) = \alpha_0 + \alpha_1 \xi + \alpha_2 \xi^2 + \alpha_3 \xi^3, \tag{56}$$

and its derivative is

$$p_3'(\xi) = \alpha_1 + 2\alpha_2 \xi + 3\alpha_3 \xi^2. \tag{57}$$

It's important to notice, that the $C^1$ continuity must be fulfilled in physical coordinates and not in dimensionless coordinates. For that reason, derivatives must be taken with respect to potentially curvilinear coordinate of beam center line $s$, not $\xi$, using chain rule:

$$\frac{\mathrm{d}p_3(\xi)}{\mathrm{d}s} = \frac{\mathrm{d}p_3(\xi)}{\mathrm{d}\xi} \frac{\mathrm{d}\xi}{\mathrm{d}s}. \tag{58}$$

From here we can calculate, by knowing that $\mathrm{d}s = \sqrt{\mathrm{d}x^2 + \mathrm{d}y^2 + \mathrm{d}z^2}$,

$$\mathrm{d}s = \frac{\mathrm{d}s}{\mathrm{d}\xi} \mathrm{d}\xi = \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}\xi}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}\xi}\right)^2 + \left(\frac{\mathrm{d}z}{\mathrm{d}\xi}\right)^2} \mathrm{d}\xi = \left\| \frac{\partial \boldsymbol{x}}{\partial \xi} \right\| \mathrm{d}\xi, \tag{59}$$

so apparently

$$\frac{\mathrm{d}p_3(\xi)}{\mathrm{d}s} = \frac{\mathrm{d}p_3(\xi)}{\mathrm{d}\xi} J(\xi)^{-1}, \tag{60}$$

where

$$J(\xi) = \left\| \frac{\partial \boldsymbol{x}(\xi)}{\partial \xi} \right\| \tag{61}$$

is Jacobian determinant. Especially, in the case of isoparametric straight beam,

$$\boldsymbol{x}(\xi) = \begin{bmatrix} x(\xi) \\ y(\xi) \\ z(\xi) \end{bmatrix} = N_1(\xi) \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + N_2(\xi) \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}, \tag{62}$$

thus

$$\left\| \frac{\partial \boldsymbol{x}(\xi)}{\partial \xi} \right\| = \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}\xi}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}\xi}\right)^2 + \left(\frac{\mathrm{d}z}{\mathrm{d}\xi}\right)^2} = \frac{1}{2} \left\| \boldsymbol{x}_2 - \boldsymbol{x}_1 \right\| = \frac{\ell^{(e)}}{2}, \tag{63}$$

where $\ell^{(e)}$ is the length of the element $e$. Polynomial and its derivatives needs to be evaluated in the endpoints of the dimensionless coordinate system $\xi_1 = -1$, $\xi_2 = 1$, yielding

$$p_3(\xi_1) = \alpha_0 - \alpha_1 + \alpha_2 - \alpha_3, \tag{64}$$

$$\left. \frac{\mathrm{d}p_3(\xi)}{\mathrm{d}s} \right|_{\xi=\xi_1} = J^{-1}(\xi_1)(\alpha_1 - 2\alpha_2 + 3\alpha_3), \tag{65}$$

$$p_3(\xi_2) = \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3, \tag{66}$$

$$\left. \frac{\mathrm{d}p_3(\xi)}{\mathrm{d}s} \right|_{\xi=\xi_2} = J^{-1}(\xi_2)(\alpha_1 + 2\alpha_2 + 3\alpha_3). \tag{67}$$

Now, shape functions can be found in the same manner than with linear interpolation. Shape function $M_1(\xi)$ can be found by setting the first equation 64 to equal one, and rest of the equations to zero, that is,

$$\alpha_0 - \alpha_1 + \alpha_2 - \alpha_3 = 1, \tag{68}$$

$$J^{-1}(\alpha_1 - 2\alpha_2 + 3\alpha_3) = 0, \tag{69}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 0, \tag{70}$$

$$J^{-1}(\alpha_1 + 2\alpha_2 + 3\alpha_3) = 0. \tag{71}$$

In the same way, proceeding each shape function at time, we find the functions:

$$M_1(\xi) = \frac{1}{4}\left(2 - 3\xi + \xi^3\right), \qquad L_1(\xi) = J\frac{1}{4}\left(\ \ 1 - \xi - \xi^2 + \xi^3\right), \tag{72}$$

$$M_2(\xi) = \frac{1}{4}\left(2 + 3\xi - \xi^3\right), \qquad L_2(\xi) = J\frac{1}{4}\left(-1 - \xi + \xi^2 + \xi^3\right). \tag{73}$$

To calculate the derivatives of shape functions with respect to physical coordinates, chain rule is needed, see again equation 60. The first derivatives are

$$\frac{M_1(\xi)}{\mathrm{d}s} = J^{-1}\frac{3}{4}\left(-1 + \xi^2\right), \qquad \frac{L_1(\xi)}{\mathrm{d}s} = \frac{1}{4}\left(-1 - 2\xi + 3\xi^2\right), \tag{74}$$

$$\frac{M_2(\xi)}{\mathrm{d}s} = J^{-1}\frac{3}{4}\left(\ \ 1 - \xi^2\right), \qquad \frac{L_2(\xi)}{\mathrm{d}s} = \frac{1}{4}\left(-1 + 2\xi + 3\xi^3\right), \tag{75}$$

and the second derivatives are

$$\frac{M_1^2\left(\xi\right)}{\mathrm{d}s^2} = J^{-2}\frac{6}{4}\left(\ \ \xi\right), \qquad \frac{L_1^2\left(\xi\right)}{\mathrm{d}s^2} = J^{-1}\frac{1}{4}\left(-2+6\xi\right), \tag{76}$$

$$\frac{M_2^2\left(\xi\right)}{\mathrm{d}s^2} = J^{-2}\frac{6}{4}\left(-\xi\right), \qquad \frac{L_2^2\left(\xi\right)}{\mathrm{d}s^2} = J^{-1}\frac{1}{4}\left(\ \ 2+6\xi\right). \tag{77}$$

The interpolations with bending shapes in $v\left(\xi\right)$ and $w\left(\xi\right)$ are then

$$v\left(\xi\right) = M_1\left(\xi\right)v_1 + L_1\left(\xi\right)\theta_{z_1} + M_2\left(\xi\right)v_2 + L_2\left(\xi\right)\theta_{z_2}, \tag{78}$$

$$w\left(\xi\right) = M_1\left(\xi\right)w_1 - L_1\left(\xi\right)\theta_{y_1} + M_2\left(\xi\right)w_2 - L_2\left(\xi\right)\theta_{y_2}. \tag{79}$$

The displacement interpolation can be written conveniently using matrix notation as

$$\boldsymbol{u}\left(\xi\right) = \begin{bmatrix} u \\ v \\ w \\ \theta \end{bmatrix} = \sum_{i=1}^{2} \begin{bmatrix} N_i & 0 & 0 & 0 & 0 & 0 \\ 0 & M_i & 0 & 0 & 0 & L_i \\ 0 & 0 & M_i & 0 & -L_i & 0 \\ 0 & 0 & 0 & N_i & 0 & 0 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ \theta_{x_i} \\ \theta_{y_i} \\ \theta_{z_i} \end{bmatrix} = \sum_{i=1}^{2} \boldsymbol{N}_i\boldsymbol{a}_i. \tag{80}$$

Generalized local strain is

$$\begin{aligned} \hat{\boldsymbol{\epsilon}} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial^2 w}{\partial x^2} \\ \frac{\partial^2 v}{\partial x^2} \\ \frac{\partial \theta_x}{\partial x} \end{bmatrix} &= \sum_{i=1}^{2} \begin{bmatrix} \frac{\partial N_i}{\partial x} u \\ \frac{\partial^2 M_i}{\partial x^2} w_i - \frac{\partial^2 L_i}{\partial x^2}\theta_{y_i} \\ \frac{\partial^2 M_i}{\partial x^2} v_i + \frac{\partial L_i}{\partial x^2}\theta_{z_i} \\ \frac{\partial N_i}{\partial x}\theta_{x_i} \end{bmatrix} \\ &= \sum_{i=1}^{2} \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\partial^2 M_i}{\partial x^2} & 0 & -\frac{\partial^2 L_i}{\partial x^2} & 0 \\ 0 & \frac{\partial^2 M_i}{\partial x^2} & 0 & 0 & 0 & \frac{\partial^2 L_i}{\partial x^2} \\ 0 & 0 & 0 & \frac{\partial N_i}{\partial x} & 0 & 0 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ \theta_{x_i} \\ \theta_{y_i} \\ \theta_{z_i} \end{bmatrix} \\ &= \sum_{i=1}^{2} \boldsymbol{B}_i\boldsymbol{a}_i \end{aligned} \tag{81}$$

Stiffness matrix is

$$\hat{\boldsymbol{K}}_{ij}^{(e)} = \int_{l^{(e)}} \boldsymbol{B}_i^{\mathrm{T}}\boldsymbol{D}\boldsymbol{B}_j\,\mathrm{d}x. \tag{82}$$

Force vector is

$$\hat{\boldsymbol{f}}_i^{(e)} = \int_{l^{(e)}} \boldsymbol{N}_i\boldsymbol{t}\,\mathrm{d}x. \tag{83}$$

Finally, matrices needs to be transformed to global coordinate systems with transformation

$$\boldsymbol{K}_{ij}^{(e)} = \boldsymbol{T}_i^{\mathrm{T}}\hat{\boldsymbol{K}}_{ij}^{(e)}\boldsymbol{T}_j \tag{84}$$

$$\boldsymbol{f}_i^{(e)} = \boldsymbol{T}_i^{\mathrm{T}}\hat{\boldsymbol{f}}_i^{(e)} \tag{85}$$

$$\boldsymbol{T} = \left[\begin{array}{ccc} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \end{array}\right], \tag{86}$$

$$\boldsymbol{e}_1 = \frac{\boldsymbol{x}_1 - \boldsymbol{x}_2}{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|}, \tag{87}$$

where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the coordinate vectors of the element end nodes. Unit vectors $\boldsymbol{e}_2$ and $\boldsymbol{e}_3$ are defined along the principal directions $\boldsymbol{y}$ and $\boldsymbol{z}$ at each node, respectively.

### Usage example

JuliaFEM is used by running a JuliaFEM input file, which is a normal Julia script including all details of the analysis. In this usage example, an input file is presented and explained row by row. The example is a natural frequency calculation of a three-dimensional frame structure using beam elements. Only the frame geometry, including node and element sets, is imported from the Abaqus input file. All other necessary details are defined in the JuliaFEM input. The orientation of beam elements is a key factor when using beam elements. The orientations are defined by normal and tangent directions. The tangent is automatically calculated from coordinates, but the normal must be defined to the JuliaFEM input by the user.

The example structure is a formula race car frame from Formula Student Oulu [19]. The Formula student Oulu race car is presented in Figure 2. The geometry of the example

Figure 2: Picture of the Formula Student Oulu race car

model is the same as that of the actual race car used by the team, but the cross-section and material values are simplified. All pipes are considered as steel pipes having diameter of 12 mm and thickness of 2.5 mm. The example frame model is presented in Figure 3. For pipes, the moment of inertia is the same in all directions. In such situations, functions and loops can be conveniently used in JuliaFEM inputs. When using packages in Julia, it must always be stated which packages will be used. FEMBeam is a package itself but it is included in JuliaFEM; thus, we only need to write using JuliaFEM.

```julia
using JuliaFEM
```

The first thing needed for this calculation is the mesh. The beam frame mesh in this case includes an Abaqus input file that we want to import to JuliaFEM. Mesh is imported using abaqus_read_mesh()-function from AbaqusReader, which is also a package included in JuliaFEM. Before reading the mesh, a file path to the location of the mesh file must be defined. All basic information about the mesh is imported to JuliaFEM, including node sets and element sets. The mesh is imported to variable mesh, which comprises a

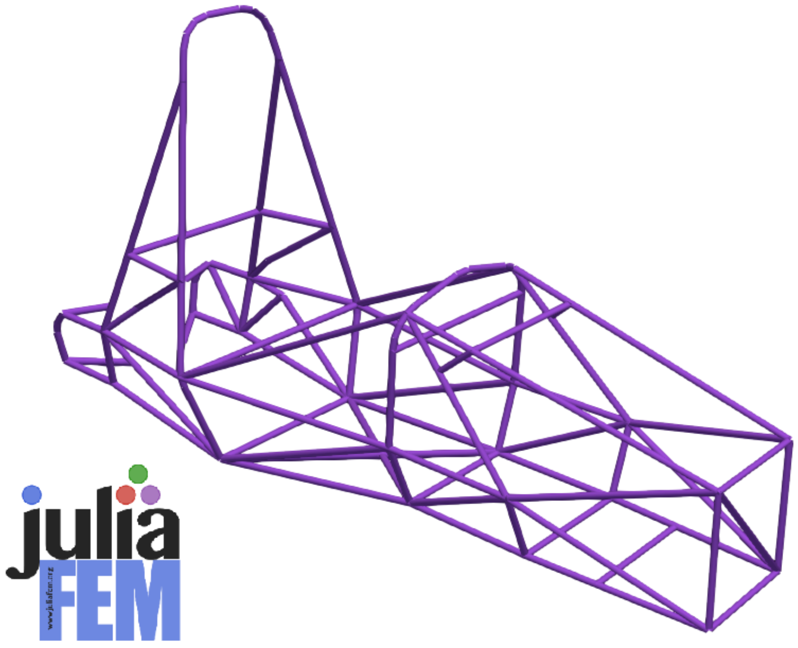


Figure 3: Frame model of the Formula Student Oulu race car.

number of dictionaries. These dictionaries can be called by functions, for example, to create elements or boundary conditions to the problem.

```
datadir = Pkg.dir("JuliaFEM", "examples", "formula_frame")
mesh = abaqus_read_mesh(joinpath(datadir, "formula_frame.inp"))
```

The elements for the problem are created using create_elements() -function. With inputs mesh and "FRAME", the element set called "FRAME" from the mesh is used to create beam elements. The material and cross-section values are defined using the update!() -function. In this case, all beam elements have the same values and they can be defined all at once.

```
beam_elements = create_elements(mesh, "FRAME")
update!(beam_elements, "youngs modulus", 210.0e3)
update!(beam_elements, "shear modulus", 80.77e3)
update!(beam_elements, "density", 7.800e-9)
update!(beam_elements, "cross-section area", 176.715)
update!(beam_elements, "moment of inertia 1", 11320.778)
update!(beam_elements, "moment of inertia 2", 11320.778)
update!(beam_elements, "polar moment of inertia", 22641.556)
```

The update!() -function is also used to define normals for the beam elements. As previously mentioned, the moment of inertia is the same in every direction when the cross-section is circular. This means that the normals can be chosen freely as long as they form an orthogonal basis where one direction is the tangent of the beam. This for-loop finds a suitable normal direction vector for every beam element. In the term k, the function indmax() finds the index of the maximum element in a collection.

```
for element in beam_elements
    X1, X2 = element("geometry", 0.0)
    t = (X2-X1)/norm(X2-X1)
    I = eye(3)
    k = indmax([norm(cross(t, I[:,k])) for k in 1:3])
    n = cross(t, I[:,k])/norm(cross(t, I[:,k]))
    update!(element, "normal", n)
end
%
```

In this usage example, four nodes around the front wheel and six nodes around the rear wheel are fixed. The boundary conditions in JuliaFEM are handled by creating the boundary condition elements which in this case are one node Poi1-typed elements. The geometry and the wanted boundary conditions must be defined using update()-function. In this case, all three displacements and rotations are fixed to zero. The for-loop is used to save some space and keep the syntax cleaner.

```
bc_elements = [Element(Poi1, [j]) for j in mesh.node_sets[:Fixed]]
update!(bc_elements, "geometry", mesh.nodes)
for i=1:3
    update!(bc_elements, "fixed displacement $i", 0.0)
    update!(bc_elements, "fixed rotation $i", 0.0)
end
```

In JuliaFEM, all different types of elements will be added to a problem that will be solved. The problem is created using Problem()-function; in this case, the problem type is Beam with the name ”frame” and the third value given for the function is the number of degrees which in this case is six. The beam elements and boundary condition elements are added to the problem using the add_elements!()-function.

```
frame = Problem(Beam, "frame", 6)
add_elements!(frame, beam_elements)
add_elements!(frame, bc_elements)
```

To calculate the natural frequencies for the beam frame, only one step is needed, which is modal analysis. The step is created using the Analysis()-function. The number of calculated eigenvalues can be set by defining step.properties.nev. Problems are added to the step using add_problems!()-function. Results can be written to a Xdmf file [20], which is a file format that can be processed with, for example, Paraview. The file is created using the Xdmf-function and in this example, it is created on the same folder with the Abaqus input file.

```
step = Analysis(Modal)
step.properties.nev = 5
add_problems!(step, [frame])
xdmf = Xdmf(joinpath(datadir, "f_frame_results"); overwrite=true)
```

```julia
add_results_writer!(step, xdmf)
run!(step)
close(xdmf.hdf)
```

After analysis, the eigenvalues are stored in step properties step.properties.eigvals and the eigenfrequencies are simply calculated by taking the square root of the eigenvalues and dividing it by two times pi. The wanted output can be printed with the println()-function. round()-function can be used to show only the required number of decimals.

```julia
eigvals = step.properties.eigvals
freqs = sqrt(eigvals)/(2*pi)

println("Five lowest natural frequencies [Hz]:")
println(round.(freqs, 2))
```

### Results

The outputs of the println-functions presented above are as follows.

```
Five lowest natural frequencies [Hz]:
[54.13, 73.89, 113.07, 144.58, 163.49]
```

Moreover, the eigenmodes were animated using Paraview and analyzed. With given boundary conditions, the eigenmodes of the usage example were just as predicted. The biggest displacement occurs on top of the main hoop of the race car. The first eigenmode is presented in Figure 4.

Natural frequencies were also calculated using a commercial program to obtain comparison results. The same input file was used with same material and cross-section values. The only difference in the input was the defining of the beam orientations, which was done by hand with the Abaqus CAE. Defining all beam orientations by hand is inconvenient and leads to some inaccuracy.

The five lowest natural frequencies from both JuliaFEM and commercial program results were compared. The differences in frequencies are under 1 Hz, which is caused by the difference in defining beam orientations. Moreover, the eigenmode animations from both JuliaFEM and the commercial program were compared. Both programs showed the same eigenmodes.

### Conclusions

The implementation of beam elements in JuliaFEM was introduced with an usage example. The usage example model is a complex 3D beam frame with a high number of beam elements being oriented in almost every possible direction. The calculation results were compared to results from commercial FEM program. The results support each other, and it can therefore be stated that our implemented beam element works accurately.

One of JuliaFEM's biggest strengths is that JuliaFEM inputs are normal Julia scripts wherein the user can use functions and loops to get the task done. This strength is

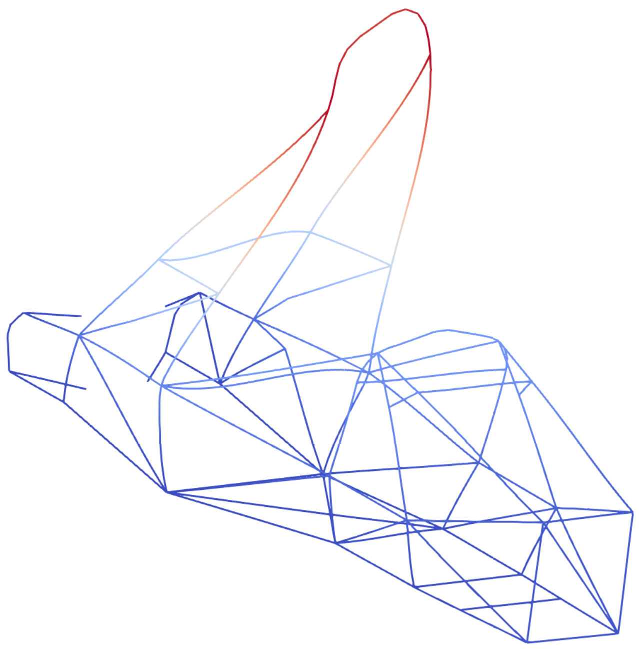


Figure 4: The first eigenmode of the structure.

emphasized in calculations like the usage example presented herein where functions and for-loops can be used. Defining the beam element pipe cross-section orientations is just one good example of using scripting in JuliaFEM inputs. Using these types of functionalities may require some programming skills but the programming language used is simple and easy to learn even for first timers. One reason these type of usage examples are made is because studying these usage examples is an efficient way to learn the basics.

## References

[1] Tero Frondelius and Jukka Aho. JuliaFEM —open source solver for both industrial and academia usage. *Rakenteiden Mekaniikka*, 50(3):229–233, 2017. open access. URL: https://doi.org/10.23998/rm.64224.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. URL: https://doi.org/10.1137/141000671.

[3] Marja Rapo, Jukka Aho, Hannu Koivurova, and Tero Frondelius. Implementing model reduction to the JuliaFEM platform. *Rakenteiden Mekaniikka*, 51(1):36–54, 2018. open access. URL: https://doi.org/10.23998/rm.69026.

[4] Marja Rapo, Jukka Aho, and Tero Frondelius. Natural frequency calculations with JuliaFEM. *Rakenteiden Mekaniikka*, 50(3):300–303, 2017. open access. URL: https://doi.org/10.23998/rm.65040.

[5] Jukka Aho. *Mortar methods for computational contact mechanics in JuliaFEM*. Master's thesis, University of Oulu, 2018.

[6] Marja Rapo, Joona Vaara, Jukka Aho, Teemu Kuivaniemi, Niclas Liljenfeldt, Antti Vuohijoki, and Tero Frondelius. Pipe routing optimization to avoid vibration prob-

lems by using juliaFEM. *Rakenteiden Mekaniikka.* open access. ACCEPTED MANUSCRIPT.

[7] Jukka Aho, Tero Frondelius, and Antti H. Niemi. Implementation of shell elements to juliaFEM project. In *Proceedings of 31st Nordic Seminar on Computational Mechanics — NSCM31*, 2018. URL: http://congress.cimne.com/NSCM-31/admin/Files/FileAbstract/a407.pdf.

[8] Tero Frondelius, Hannu Tienhaara, and Mauri Haataja. History of structural analysis & dynamics of Wärtsilä medium speed engines. *Rakenteiden Mekaniikka*, 51(2):1–31, 2018. open access. URL: https://doi.org/10.23998/rm.69735.

[9] Michael A Crisfield. *Non-linear finite element analysis of solids and structures*, volume 1. Wiley New York, 1991.

[10] Ted Belytschko, Wing Kam Liu, Brian Moran, and Khalil Elkhodary. *Nonlinear Finite Elements For Continua And Structures.* John Wiley & Sons, 2014.

[11] Eugenio Oñate. *Structural analysis with the finite element method. Linear statics: volume 2: beams, plates and shells.* Springer Science & Business Media, 2013.

[12] Olek C Zienkiewicz and Robert L Taylor. *The Finite Element Method for Solid and Structural Mechanics.* 6th edition.

[13] Juha Paavola and Eero-Matti Salonen. Yksinkertaisten rakennemallien tasapainoyhtälöt - osa I suorat ja kaarevat tasosauvat. *Rakenteiden mekanikka*, 28(4):29–43, 1995.

[14] Kanwar K. Kapur. Vibrations of a Timoshenko Beam, Using Finite-Element Approach. *The Journal of the Acoustical Society of America*, 40(5):1058–1063, nov 1966. doi:10.1121/1.1910188.

[15] J. S. Archer. Consistent matrix formulations for structural analysis using finite-element techniques. *AIAA Journal*, 3(10):1910–1918, 1965. doi:10.2514/3.3279.

[16] D.L. Thomas, J.M. Wilson, and R.R. Wilson. Timoshenko beam finite elements. *Journal of Sound and Vibration*, 31(3):315–330, 1973. doi:10.1016/S0022-460X(73)80276-7.

[17] Eric Reissner. On one-dimensional large-displacement finite-strain beam theory. *Studies in applied mathematics*, 52(2):87–95, 1973.

[18] G. R. Cowper. The shear coefficient in Timoshenko's beam theory. *Journal of applied mechanics*, 33(2):335–340, 1966.

[19] Formula student oulu, 2018. URL: https://fsoulu.wordpress.com/.

[20] Eric R Mark. Enhancements to the extensible data model and format (XDMF). In *DoD High Performance Computing Modernization Program Users Group Conference, 2007*, pages 322–327. IEEE, 2007.

Jukka Aho
ahojukka5@gmail.com

Ville Jämsä

Teemu Kuivaniemi, Niclas Liljenfeldt, Tero Frondelius
Wärtsilä
Järvikatu 2-4
65100 Vaasa
firstname.lastname@wartsila.com

Tero Frondelius
Oulu University
Pentti Kaiteran katu 1
90014 Oulu
firstname.lastname@oulu.fi