OpenFOAM® Journal

Volume 5 [Full Papers], Pages 180–210
ISSN: 2753-8168

beamFoam: A cell-centred finite volume solver for nonlinear geometrically-exact beams in OpenFOAM®

Seevani Bali[1,*], Amirhossein Taran[1], Željko Tuković[2], Vikram Pakrashi[1], and Philip Cardiff[1]

Email address: seevani.bali@ucd.ie, amirhossein.taran@ucdconnect.ie, zeljko.tukovic@fsb.unizg.hr, vikram.pakrashi@ucd.ie, philip.cardiff@ucd.ie

[1]School of Mechanical and Materials Engineering, University College Dublin, Ireland

[2]Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia

DOI: https://doi.org/10.51560/ofj.v5.170
Results with version(s): OpenFOAM® v2306
Repository: https://github.com/solids4foam/beamFoam

Abstract. This paper presents a new cell-centred finite volume beam solver, `beamFoam`, for simulating slender structures undergoing large deformations and rotations. The solver is based on the geometrically exact nonlinear Simo–Reissner beam theory and can represent tension, bending, shear, and torsion. Although the underlying model is 1-D, the formulation resolves 3-D kinematics, enabling accurate prediction of spatial deformations and rotations. The paper outlines the governing equations, spatial and temporal discretisation, and code implementation, together with guidelines for simulation setup. The solver's accuracy and efficiency are demonstrated through three benchmark cases. A 3-D cantilever column under finite deformation shows close agreement with a continuum 3-D solver from the `solids4Foam` toolbox, while achieving improved accuracy on coarse meshes and significantly reduced computational cost. Additional quasi-static and dynamic tests illustrate the implementation of boundary conditions, stability of time-integration schemes, and mesh convergence behaviour. These results establish `beamFoam` as an efficient and accurate beam solver within OpenFOAM® and represent a first significant step towards fluid–structure interaction simulations of slender structures.

1. Introduction

The implementation of solid mechanics solvers in a finite volume framework [1–3] integrated within the OpenFOAM® environment has enabled the development of multi-physics simulations with seamless integration between fluid flow and structural deformations [4–6]. A range of solid solvers, starting from one-dimensional (1-D) models to three-dimensional (3-D) continuum formulations, exist in the finite element (FE) community to address various structural deformations. The governing equations for 3-D solid mechanics problems are often reduced to mathematically consistent 1-D beam or rod theories for modelling slender bodies. A structure is considered *slender* if the longitudinal dimension is significantly greater than its cross-sectional area. Slender structures are prevalent in wide-span engineering applications, including (a) Offshore systems (mooring cables [7–9], risers [10]), (b) Biomedical devices (stents and catheters [11, 12]), (c) Environmental flows (vegetation and coastal plant dynamics modelling [13–15]), and, (d) Soft robotics (actuators and tendon-driven mechanisms [16, 17]). Using 1-D beam theories instead of full 3-D continuum mechanics to model slender bodies results in efficient computation without compromising the overall accuracy of the numerical formulation. This work introduces a 1-D beam solver to expand the capabilities of OpenFOAM® software and contribute towards furthering the fluid-structure interaction (FSI) applications in OpenFOAM® community.

Traditional beam formulations like the classic Euler-Bernoulli and Timoshenko theories [18] are limited to linear analysis of straight, isotropic beams undergoing small deformations. The works by [19–21] have presented finite volume implementations of Euler-Bernoulli and shear-deformable Timoshenko beam

* Corresponding author

Received: 31 August 2025, Accepted: 21 November 2025, Published: 22 December 2025

theories for both straight and curved beam configurations. Additional studies have examined beam stability [22–24] and vibrational characteristics [25, 26] using finite volume method on beams. In contrast, nonlinear beam theories, such as, Kirchhoff-Love [27] and Simo-Reissner [28–30], can model large deformations, arbitrary initial curvatures, and anisotropic cross-sections. The Kirchhoff-Love theory neglects shear deformation, while the Simo-Reissner beam includes it — but assumes rigid cross-sections (no warping). Recently, Bali et al. [31, 32] implemented a geometrically exact Simo-Reissner beam solver using finite volume discretisation. Taran et al. [33] extended this work, applying the finite volume-based beam solver to mooring dynamics via OpenFOAM® `sixDOFRigidBodyMotion` library. Currently, OpenFOAM® lacks a general nonlinear 1-D beam solver. While `solids4Foam` toolbox [34] supports 3-D linear and nonlinear solid mechanics with fluid-structure coupling, it does not provide 1-D solvers for slender structures like cables/rods. This paper presents `beamFoam`, the first native 1-D nonlinear beam solver in OpenFOAM® (v2306), building upon the geometrically exact formulation by Bali et al. [31]. The focus here is to document the solver's implementation including pre- and post-processing `utilities` and `functionObjects`, boundary condition treatments, and block-coupled solution algorithm. The goal is to provide a comprehensive, user-friendly reference for the OpenFOAM® community, ensuring straightforward adoption and application of `beamFoam`.

The remainder of the paper is structured as follows. Section 2 introduces the mathematical model underlying the `beamFoam` solver. The spatial and temporal finite volume discretisation is presented in Section 3. Section 4 describes the code implementation, while Section 5 outlines the steps required to set up a test case. Three benchmark simulations demonstrating the solver's capabilities are reported in Section 6, and conclusions with future outlook are given in Section 7.

2. Mathematical model

This section summarises the kinematics and governing equations of the nonlinear Simo-Reissner beam theory used in the `beamFoam` solver. The Simo–Reissner beam is a 1-D theory that represents the beam by its centerline, with governing equations in a single spatial coordinate. Despite this, it captures full 3-D kinematics, allowing each cross-section to translate and rotate freely, including shear deformation, torsion, and large displacements/rotations. A full description of the mathematical model is given in Bali et al. [31]; here, the essential formulation is outlined.

2.1. Beam configurations and kinematics.

The beam's motion is described in three configurations (Fig. 1):

(1) **Reference configuration (A)**: The beam is aligned along the global x-axis, with a local orthonormal triad $\boldsymbol{e}_i, \boldsymbol{e}_1 = (1,0,0)^T, \boldsymbol{e}_2 = (0,1,0)^T, \boldsymbol{e}_3 = (0,0,1)^T$. The vectors $\boldsymbol{e}_2$ and $\boldsymbol{e}_3$ lie in the cross-section plane. The governing equations are formulated using variables measured to this reference state, and hence, it is a total Lagrangian description.

(2) **Initial configuration (B)**: For a beam with an arbitrary orientation or initial curvature, the centroid line $\boldsymbol{r}_0(s)$ together with the initial curvature described by the 3×3 rotation matrix $\boldsymbol{\Lambda}_0(s)$ define its initial configuration. The local bases is given by $\boldsymbol{g}_i^0(s) = \boldsymbol{\Lambda}_0(s)\ \boldsymbol{e}_i$. The parameter $s \in [0, L]$ is the arc-length variable, and L is the total length of the beam. For straight beams aligned with the x-axis, the arc-length derivative $\boldsymbol{r}_0'(s) = \boldsymbol{g}_1^0 \equiv \boldsymbol{e}_1$ and $\boldsymbol{\Lambda}_0(s) = \mathbf{I}_3$ - identity matrix and the reference and initial configuration of the beam coincide.

(3) **Deformed configuration (C)**: From the initial stress-free state, the centroid line deforms to $\boldsymbol{r}(s,t) = \boldsymbol{r}_0(s) + \boldsymbol{w}(s,t)$, where $\boldsymbol{w}(s,t)$ is the displacement vector. The cross-section in the deformed state rotates via $\boldsymbol{g}_i(s,t) = \boldsymbol{\Lambda}(s,t)\ \boldsymbol{g}_i^0(s)$, where $\boldsymbol{\Lambda}(s,t)$ is the relative rotation with respect to the initial configuration. The parameter t represents time. The tangent to the mean deformed line $\boldsymbol{r}'(s)$ no longer aligns with $\boldsymbol{g}_1(s)$, i.e., $\boldsymbol{r}'(s) \neq \boldsymbol{g}_1(s)$, indicating shear deformation. The total rotation of the cross-section relative to the reference state is $\boldsymbol{\Lambda}_t(s,t) = \boldsymbol{\Lambda}(s,t)\ \boldsymbol{\Lambda}_0(s)$.

The pair $(\boldsymbol{r}(s,t), \boldsymbol{\Lambda}_t(s,t))$ fully define the deformed state of the beam.

Finite 3-D rotations used in the Simo-Reissner beam theory belong to the Lie group SO(3) - the set of all proper orthogonal rotation matrices. An important property of 3-D rotations is that the composition of rotations in 3D is multiplicative but non-commutative ($\boldsymbol{\Lambda} = \boldsymbol{\Lambda}_2\boldsymbol{\Lambda}_1 \neq \boldsymbol{\Lambda}_1\boldsymbol{\Lambda}_2$). Therefore, rotation vectors cannot be updated by additive properties ($\boldsymbol{\Lambda} \neq \exp(\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2)$). A rotation vector $\boldsymbol{\theta} \in \mathbb{R}^3$ is mapped to the corresponding rotation tensor $\boldsymbol{\Lambda} \in \mathrm{SO}(3)$ [35] (Fig. 2) via,

$$\boldsymbol{\Lambda}(\boldsymbol{\theta}) = \exp(\widehat{\boldsymbol{\theta}}) = \mathbf{I} \ + \ \frac{\sin\theta}{\theta}\widehat{\boldsymbol{\theta}} \ + \ \frac{1-\cos\theta}{\theta^2}\widehat{\boldsymbol{\theta}}\widehat{\boldsymbol{\theta}} \tag{1}$$

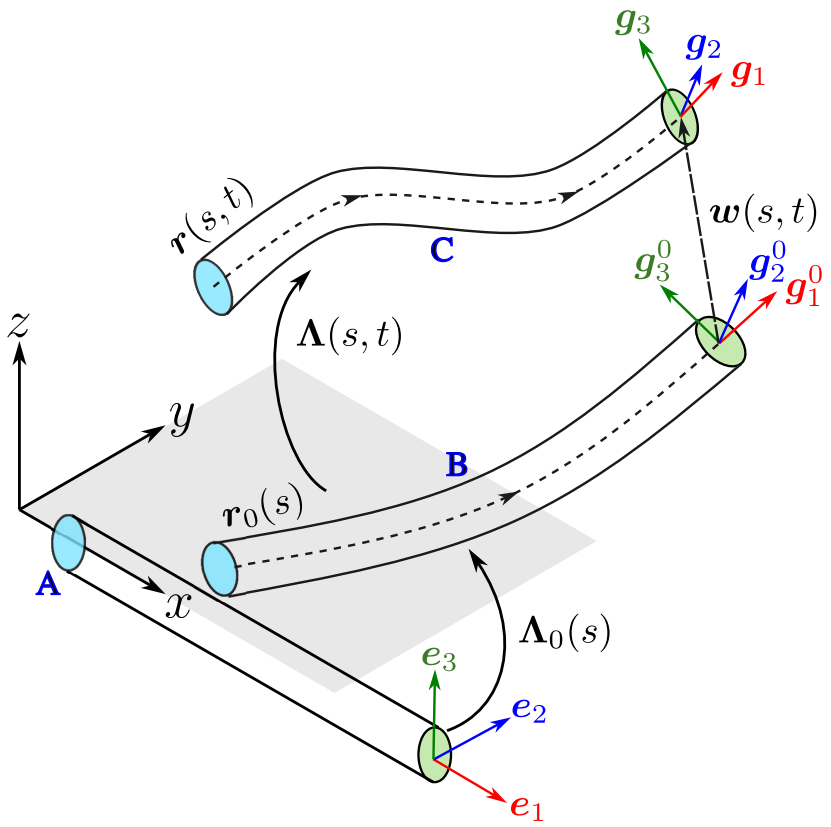

Figure 1. Beam kinematics description - Reference configuration (A), Initial configuration (B), and Deformed configuration (C)

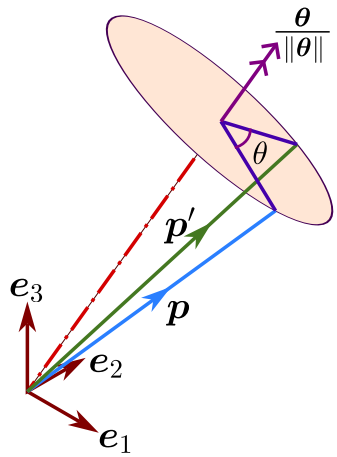

Figure 2. Schematic diagram of the Rodrigues' rotation parametrisation, where a vector $\boldsymbol{p}$ (blue) is rotated to the new position $\boldsymbol{p}'$ (green) by a rotation vector $\boldsymbol{\theta}$ with a magnitude of rotation θ; the relation between rotated vector and the original vector is given via the rotation tensor $\boldsymbol{\Lambda}$, i.e., $\boldsymbol{p}' = \boldsymbol{\Lambda}\ \boldsymbol{p} \equiv \exp(\widehat{\boldsymbol{\theta}})\ \boldsymbol{p}$

Equation 1, known as Rodrigues' formula, provides the nonlinear mapping between a finite rotation vector and the associated rotation matrix used to update beam orientation. The $\widehat{(\cdot)}$ operator in Eqn. 1 and henceforth, denotes a skew-symmetric matrix constructed from the corresponding rotation vector in $\mathbb{R}^3$ as follows,

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \implies \widehat{\boldsymbol{a}} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{2}$$

2.2. **Governing equations and strain measures.** The governing equations of Simo-Reissner beam theory differ fundamentally from 3-D solid mechanics, as the beam is modelled as a 1-D continuum with a deformable space curve $\boldsymbol{r}(s,t)$ and a rigidly rotating cross-section described by the rotation field $\boldsymbol{\Lambda}_t(s,t)$. Consequently, the balance laws are formulated *directly* on cross-section integrated stress resultants, i.e., forces $\boldsymbol{q}$ and moments $\boldsymbol{m}$, rather than being derived from pointwise 3-D stress equilibrium. Through the 3-D beam kinematics, the stress resultants are linearly related to the strain vectors - translational strains, $\boldsymbol{\gamma}$, and rotational strains, $\boldsymbol{\kappa}$ - as,

$$\boldsymbol{q} = \boldsymbol{\Lambda}_t \boldsymbol{C}_q \boldsymbol{\gamma} \tag{3a}$$

$$\boldsymbol{m} = \boldsymbol{\Lambda}_t \boldsymbol{C}_m \boldsymbol{\kappa} \tag{3b}$$

where $\boldsymbol{C}_q = \text{diag}\,[EA, GA_2, GA_3]$ and $\boldsymbol{C}_m = \text{diag}\,[GJ, EI_2, EI_3]$ are constant diagonal constitutive matrices. These entries in $\boldsymbol{C}_q, \boldsymbol{C}_m$ are related to the beam geometry and the hyperelastic material law. E and $G = \frac{E}{2(1+\nu)}$ denote the elastic and shear moduli of the material, respectively, and ν is Poisson's ratio. A_j and I_j $(j = 1, 2, 3)$ are the areas and area moments about the principal axes of inertia, respectively $(J = I_2 + I_3)$. These constitutive relations restrict the Simo-Reissner beam formulation to bi-symmetric cross-sections (where the shear centre coincides with the geometric centroid); adaptations are required for arbitrary cross-section shapes.

The translational strain vector $\boldsymbol{\gamma}$ and the rotational strain vector $\boldsymbol{\kappa}$ relate to the beam kinematics pair $(\boldsymbol{r}(s,t), \boldsymbol{\Lambda}_t(s,t))$ as given in Eqn. 4. The arguments (s,t) are dropped for brevity and the relation $\boldsymbol{\Lambda}_t = \boldsymbol{\Lambda}\boldsymbol{\Lambda}_0$ is used for manipulation.

$$\boldsymbol{\gamma} = \boldsymbol{\Lambda}_t^\top \boldsymbol{r}' - \boldsymbol{\Lambda}_t^\top \boldsymbol{g}_1 = \boldsymbol{\Lambda}_0^\top \boldsymbol{\Lambda}^\top \boldsymbol{r}' - \boldsymbol{\Lambda}_0^\top \boldsymbol{r}_0' \tag{4a}$$

$$\widehat{\boldsymbol{\kappa}} = \boldsymbol{\Lambda}_t^\top \boldsymbol{\Lambda}_t' - \boldsymbol{\Lambda}_0^\top \boldsymbol{\Lambda}_0' \tag{4b}$$

In Eqn. 4, the strains $\boldsymbol{\gamma}$ and $\boldsymbol{\kappa}$ are defined are in the reference configuration. The $(\cdot)^\top$ operation on $\boldsymbol{\Lambda}_t$ denotes the transpose matrix operation that *pulls-back* the spatial derivative of $\boldsymbol{r}'(s)$ to $\boldsymbol{\gamma}$ in its material or Lagrangian description. However, the stress resultants $\boldsymbol{q}$ and $\boldsymbol{m}$ are vectors in the spatial description; therefore, the operation $\boldsymbol{C}_q\boldsymbol{\gamma}$ in the material/reference frame is *pushed* into the local/spatial frame via $\boldsymbol{\Lambda}_t$ resulting into the spatial force form. Such push-pull operations are common in the Simo-Reissner beam theory [28].

In the calculation of rotational strain $\boldsymbol{\kappa}$ in Eqn. 4b, the arc-length derivative of the rotation matrix is a mathematically involved and nonlinear calculation. Instead, the rotational strain increment $\Delta\boldsymbol{\kappa}$ is updated iteratively with each solution increment (Newton iterations) as [36, 37],

$$\Delta\boldsymbol{\kappa} = \boldsymbol{\Lambda}_t^\top \boldsymbol{T}^\top(\Delta\boldsymbol{\theta})\Delta\boldsymbol{\theta}' \tag{5}$$

where the tangent operator $\boldsymbol{T}(\boldsymbol{\theta})$ is the nonlinear mapping given by ($\theta \equiv \|\boldsymbol{\theta}\|$ and $\boldsymbol{\theta} \otimes \boldsymbol{\theta}$ is the tensor or dyadic product of vectors),

$$\boldsymbol{T}(\boldsymbol{\theta}) = \frac{\sin\theta}{\theta}\mathbf{I} \;+\; \frac{1}{\theta^2}\left(1 - \frac{\sin\theta}{\theta}\right)\boldsymbol{\theta}\otimes\boldsymbol{\theta} \;+\; \frac{1-\cos\theta}{\theta^2}\widehat{\boldsymbol{\theta}} \tag{6}$$

As previously mentioned, the governing equations for 1-D beams are expressed in terms of integrated cross-section resultants $\boldsymbol{q}$ and $\boldsymbol{m}$. Therefore, the integral form of the balance equations are reduced from 3-D volume integrals to 1-D line integrals, and are given by [30, 31],

$$\int_L \boldsymbol{q}'\,\mathrm{d}L \;+\; \int_L \bar{\boldsymbol{q}}\,\mathrm{d}L = \int_L \rho A\,\ddot{\boldsymbol{r}}\,\mathrm{d}L \tag{7}$$

$$\int_L \boldsymbol{m}'\,\mathrm{d}L \;+\; \int_L (\boldsymbol{r}'\times\boldsymbol{q})\,\mathrm{d}L \;+\; \int_L \bar{\boldsymbol{m}}\,\mathrm{d}L = \int_L \left[\boldsymbol{\Lambda}_t\bar{I}_\rho\boldsymbol{\alpha} + \boldsymbol{\Lambda}_t\widehat{\boldsymbol{\omega}}\,(\bar{I}_\rho\,\boldsymbol{\omega})\right]\mathrm{d}L \tag{8}$$

where, $\bar{\boldsymbol{q}}$ and $\bar{\boldsymbol{m}}$ are distributed external forces and torques per unit arc-length respectively, and "$\times$" denotes the cross product between two vectors. In the Eqn. 7, ρ is the density of the material, A is the cross-section area, and $\ddot{\boldsymbol{r}} \equiv \boldsymbol{a}$ is the linear acceleration term where $\ddot{(\cdot)}$ denotes second time-derivative. In the angular momentum equation (Eqn. 8), $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ represent angular velocity and angular acceleration in the reference/material frame respectively; $\bar{I}_\rho = \rho \cdot \text{diag}(\mathrm{J}, \mathrm{I}_2, \mathrm{I}_3)$. The dependence of all the vector quantities on the arc-length parameter s is avoided in the notations here for brevity.

3. Numerical method

The governing equilibrium equations of the Simo–Reissner beam formulation in Section 2 are highly nonlinear due to finite rotation kinematics and nonlinear strain measures. Consequently, fixed-point or segregated algorithms such as SIMPLE or PIMPLE exhibit poor convergence for such highly coupled nonlinear systems. Preliminary tests with a segregated implementation of the beam solver confirmed this behaviour. Instead, a block-coupled approach using the Newton-Raphson method with exact linearisation (Jacobian) is implemented in the current work, which ensures consistent tangent linearisation of internal forces and moments.

3.1. Solution methodology.

The governing equations (Eqns. 7 and 8) can be expressed in the general residual form,

$$\boldsymbol{R}_{n+1}(\boldsymbol{w}, \boldsymbol{\theta}) = \boldsymbol{0} \tag{9}$$

where $\boldsymbol{R}$ is the residual imbalance of the balance laws at time $(n+1)$, written in terms of the primary unknowns: displacement vector $\boldsymbol{w}$ and rotation vector $\boldsymbol{\theta}$. A Newton-Raphson iterative scheme is adopted to solve the system. The residual $\boldsymbol{R}$ is expanded in a first-order Taylor series about the current iteration, neglecting the higher-order terms,

$$\boldsymbol{R}_{n+1}(\boldsymbol{w}, \boldsymbol{\theta}) = \overset{*}{\boldsymbol{R}}(\boldsymbol{w}, \boldsymbol{\theta}) + \overset{*}{\boldsymbol{R}'}(\boldsymbol{w}, \boldsymbol{\theta}) \left[\Delta \boldsymbol{w}, \Delta \boldsymbol{\theta}\right]^\top = 0 \tag{10}$$

The superscript $\overset{*}{(\cdot)} \equiv (\cdot)^{(k)}_{n+1}$ operation denotes evaluation at the previous iteration (k is the iteration counter) and $\boldsymbol{R}' \equiv \boldsymbol{J}$ is the Jacobian. This leads to a linearised system,

$$\overset{*}{\boldsymbol{J}}(\boldsymbol{w}, \boldsymbol{\theta}) \begin{bmatrix} \Delta \boldsymbol{w} \\ \Delta \boldsymbol{\theta} \end{bmatrix} = -\overset{*}{\boldsymbol{R}}(\boldsymbol{w}, \boldsymbol{\theta}) \tag{11}$$

where $\Delta \boldsymbol{w}$ and $\Delta \boldsymbol{\theta}$ are the displacement and rotation corrections, respectively. At the beginning of each time (or load) step $(n+1)$, the initial guess for the Newton-Raphson iteration is taken as the converged solution from the previous step, i.e., $(\cdot)^0_{n+1} = (\cdot)_n$. Subsequent iterations within the same step use the updated displacement and rotation corrections from the previous Newton iteration as the starting initial guess. Therefore, the residuals are small and ensure quadratic convergence. For problems like post-buckling analysis, special globalisation techniques like (line search or arc-length continuation methods) would be required to ensure convergence of Newton-Raphson method.

Thus, the linear system in Eqn. 11 is solved at each iteration to obtain the corrections $\Delta \boldsymbol{w}$ and $\Delta \boldsymbol{\theta}$. The primary unknowns are then updated at $(k+1)$-th iteration as,

$$\boldsymbol{w}^{(k+1)}_{n+1} = \boldsymbol{w}^{(k)}_{n+1} + \Delta \boldsymbol{w} \tag{12a}$$

$$\boldsymbol{\theta}^{(k+1)_{n+1}} = \boldsymbol{\theta}^{(k)}_{n+1} + \boldsymbol{T}(\Delta \boldsymbol{\theta})^\top \Delta \boldsymbol{\theta} \tag{12b}$$

$$\boldsymbol{r}^{(k+1)}_{n+1} = \boldsymbol{r}^{(k)}_{n+1} + \boldsymbol{w}^{(k+1)}_{n+1} \tag{12c}$$

$$\boldsymbol{\Lambda}^{(k+1)}_{t,n+1} = \exp(\widehat{\Delta \boldsymbol{\theta}}) \boldsymbol{\Lambda}^{(k)}_{t,n+1} \tag{12d}$$

This process is repeated until convergence is achieved, i.e. the residual $\boldsymbol{R}$ becomes sufficiently small and/or the increments $\Delta \boldsymbol{w}, \Delta \boldsymbol{\theta}$ fall below prescribed tolerances.

Since the 3-D rotations are not additive in nature, the consistent update is performed by updating the rotation matrix $\boldsymbol{\Lambda}_t$ directly as shown in Eqn. 12d. The exponential map $\exp(\widehat{\Delta \boldsymbol{\theta}})$ in Eqn. 12d is evaluated using the Rodrigues' formula (Eqn. 1). Equation 12b is not employed directly for advancing the solution; it is retained for post-processing and visualisation.

Using the incremental displacement ($\Delta \boldsymbol{w}$) and rotation ($\Delta \boldsymbol{\theta}$) correction vectors, the translational strain $\boldsymbol{\gamma}$ is updated by Eqn. 4a and rotational strain $\boldsymbol{\kappa}$ is evaluated using Eqn. 5. The stress resultants $\boldsymbol{q}$ and $\boldsymbol{m}$ are updated by Eqn. 19 and Eqn. 20 respectively. With subsequent Newton-Raphson iterations, the corrections $\Delta \boldsymbol{w}$ and $\Delta \boldsymbol{\theta}$ approach zero and the linearised force and moment expressions approach the exact expressions in Eqn. 3.

To assemble $\overset{*}{\boldsymbol{J}} \equiv \boldsymbol{J}^{(k)}_{n+1}$ in Eqn. 11, each term in the linear and angular momentum equations (Eqn. 7 and Eqn. 8) must be linearised (details are presented in Appendix A). Since these are two vector equations,

there are six scalar unknowns $[\Delta w_1, \Delta w_2, \Delta w_3, \Delta\theta_1, \Delta\theta_2, \Delta\theta_3]$ per solution point (cell centre), yielding a 6×6 Jacobian given by,

$$\overset{*}{\boldsymbol{J}}(\boldsymbol{w}, \boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{J}_{qw} & \boldsymbol{J}_{q\theta} \\ \boldsymbol{J}_{mw} & \boldsymbol{J}_{m\theta} \end{bmatrix} \tag{13}$$

where $\boldsymbol{J}_{qw}$, $\boldsymbol{J}_{q\theta}$, $\boldsymbol{J}_{mw}$, and $\boldsymbol{J}_{m\theta}$ are 3×3 blocks. $\boldsymbol{J}_{qw}$ and $\boldsymbol{J}_{q\theta}$ are obtained from the linearised linear momentum equation, Eqn. 7, and $\boldsymbol{J}_{mw}$ and $\boldsymbol{J}_{m\theta}$ from the linearised angular momentum equation Eqn. 8. The details of the linearisation are presented in Appendix A.

Remark: For consistency with the numerical implementation, this work adopts a slightly modified notation compared to Bali et al. [31]. While the theoretical framework is unchanged, certain symbols have been renamed to align with coding conventions — for example, the rotation vector $\boldsymbol{\psi}$ is now denoted $\boldsymbol{\theta}$, and the force vector is written as $\boldsymbol{q}$. Minor adjustments to other variables have also been made for clarity in the computational context. Readers are referred to Bali et al. [31] for the full derivation and theoretical details; the equations here are fully equivalent in formulation.

The mathematical model for Simo-Reissner beams is discretised in a fully implicit, coupled manner using the cell-centred finite volume method, resulting in a discrete approximation of the integral formulation. First, the solution domain discretisation is presented followed by discretised form of governing equations. When temporal effects are considered, time is also discretised into finite steps, and the model is solved in a time-marching fashion.

3.2. **Computational domain discretisation.** The beam in its reference configuration is divided into a finite number of uniform segments or control volumes (CVs) as is shown in Fig. 3. A typical computational stencil (Fig. 3) consists of the central CV (cell) of length L_p with computational node P, located at the cell centroid, bounded by internal face f - the common face between cell P and neighbouring cell N. The normal to the internal face f is given by $\boldsymbol{n}_f$.

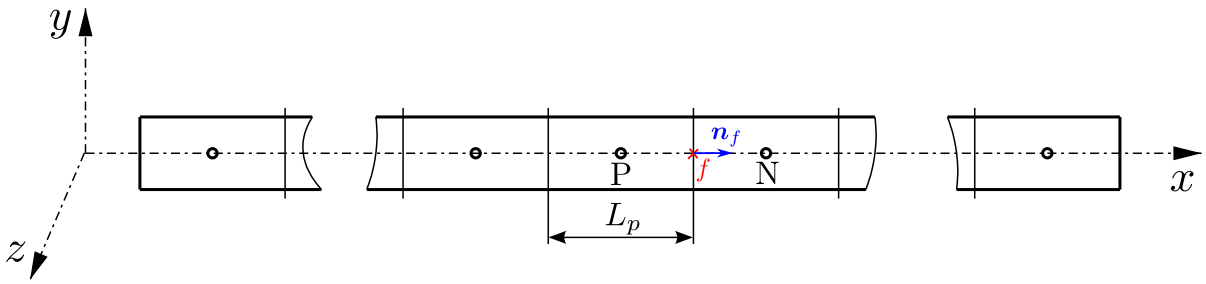

Figure 3. Beam body in reference configuration discretised by a finite set of 1-D CVs (cells).

3.3. **Spatial discretisation.** The integral form of the linear and angular momentum equations (Eqns. 7 and 8) are discretised over the beam computational domain as follows,

$$\int_{\mathrm{L_p}} \boldsymbol{q}' \, \mathrm{d}L = \sum_f \boldsymbol{q}_f \cdot \boldsymbol{n}_f \quad ; \quad \int_{\mathrm{L_p}} \bar{\boldsymbol{q}} \, \mathrm{d}L \approx \bar{\boldsymbol{q}}_{\mathrm{p}} \, \mathrm{L_p} \tag{14a}$$

$$\int_{\mathrm{L_p}} \boldsymbol{m}' \, \mathrm{d}L = \sum_f \boldsymbol{m}_f \cdot \boldsymbol{n}_f \quad ; \quad \int_{\mathrm{L_p}} (\boldsymbol{r}' \times \boldsymbol{q}) \, \mathrm{d}L = \frac{1}{2}\mathrm{L_p} \sum_f (\boldsymbol{r}'_f \times \boldsymbol{q}_f) \quad ; \quad \int_{\mathrm{L_p}} \bar{\boldsymbol{m}} \, \mathrm{d}L \approx \bar{\boldsymbol{m}}_{\mathrm{p}} \, \mathrm{L_p} \tag{14b}$$

where $\boldsymbol{q}_f$ and $\boldsymbol{m}_f$ are the force and moment vectors evaluated at the centre of the faces enclosed by the cell volume P. Since, the beam model is 1-D solver, the stress resultants are balanced along the beam centreline. Therefore, for a typical computational cell P, the faces to the east 'e' and west 'w' of the cell constitute the two internal faces (See Fig. 4).
The terms $\bar{\boldsymbol{q}}$, $\bar{\boldsymbol{m}}$ are assumed to have a linear variation across the CV and are approximated by the mid-point rule. In the moment balance equation, the coupling term - integral of $(\boldsymbol{r}' \times \boldsymbol{q})$ is approximated over the CV using the trapezoidal rule and is evaluated at the faces w and e respectively.

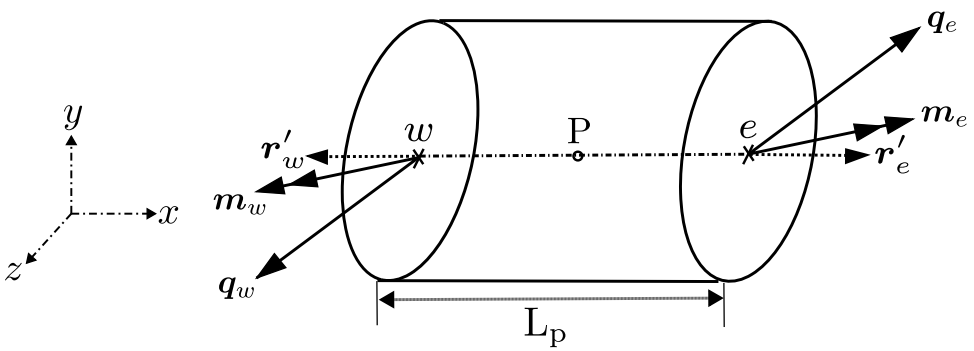

Figure 4. Balance of forces and moments on an isolated CV in the deformed configuration

To evaluate the values of the variables at the face centres, a linear interpolation scheme is adopted, while the gradients are evaluated using central differencing (the orthogonal `snGrad` scheme available in OpenFOAM®).

3.4. **Temporal discretisation.** The inertial terms in the linear and angular momentum equations can be discretised at the computational cell centre P using the mid-point rule as,

$$\int_{\mathrm{L_p}} \rho A \ddot{\boldsymbol{r}} \, \mathrm{d}L = \rho \; A \; \ddot{\boldsymbol{r}}_{\mathrm{p}} \; \mathrm{L_p} \tag{15a}$$

$$\int_{L} \Big[\boldsymbol{\Lambda}_t \bar{I}_\rho \boldsymbol{\alpha} + \boldsymbol{\Lambda}_t \widehat{\boldsymbol{\omega}} \; (\bar{I}_\rho \; \boldsymbol{\omega}) \Big] \, \mathrm{d}L = \boldsymbol{\Lambda}_{t,\mathrm{p}} \Big[\bar{I}_\rho \boldsymbol{\alpha}_{\mathrm{p}} + \widehat{\boldsymbol{\omega}}_{\mathrm{p}} (\bar{I}_\rho \boldsymbol{\omega}_{\mathrm{p}}) \Big] \; \mathrm{L_p} \tag{15b}$$

where $\ddot{\boldsymbol{r}}_{\mathrm{p}}$, $\boldsymbol{\omega}_{\mathrm{p}}$ and $\boldsymbol{\alpha}_{\mathrm{p}}$ are the linear acceleration, angular velocity and angular acceleration evaluated at the cell centre. Temporal discretisation is performed using either a first-order accurate Euler scheme or the second-order accurate Newmark-β scheme, the latter being a standard choice for integrating second-order equations widely used in solving structural dynamics problems.

For a time integration scheme, two important aspects must be considered: (i) stability and (ii) accuracy. The Euler and Newmark-β schemes implemented in the beam solver are both implicit, i.e., they evaluate the temporal terms using quantities at the advanced time level. For linear governing equations, such implicit schemes are unconditionally stable. However, the nonlinear nature of the beam equations means that unconditional stability cannot be strictly guaranteed.

Unlike explicit schemes commonly used in CFD or wave-propagation problems, implicit formulations do not impose an explicit Courant number constraint; hence, an adaptive time-step control based on a CFL criterion is not applicable. Instead, the optimal time-step size is chosen by estimating the dominant natural frequency of the beam for the specified boundary conditions and selecting Δt so that at least $20 - 50$ points are resolved per vibration period (see numerical example 6.1 for more details).

The **stability** of this chosen time step and the time integration scheme is verified a posteriori using an energy balance check: for a conservative system, the numerical discrete total energy (details in Appendix B) should remain constant (or monotonically decrease if numerical damping is present). The accuracy of the time integration is assessed separately by performing time-step convergence studies, where simulations are repeated with successively smaller Δt until the computed response (e.g., displacements or internal moments) converges within a specified tolerance. The numerical examples in Section 6 compare the accuracy of the two implemented time schemes by plotting the total energy evolution over time.

3.5. **Boundary conditions.** For the primary variables $\boldsymbol{w}$ and $\boldsymbol{\theta}$, two types of boundary conditions can be applied,

- **Dirichlet boundaries** - The value of $\boldsymbol{w}$ and $\boldsymbol{\theta}$ are directly imposed at the boundary face centres in the discretised equations.
- **Neumann boundaries** - The specified force $\boldsymbol{q}$ and moments $\boldsymbol{m}$ are prescribed at the boundary, and the corresponding displacements ($\Delta\boldsymbol{w}$) and rotations ($\Delta\boldsymbol{\theta}$) are obtained by extrapolation from the interior solution using the linearised force and moment relations.

In both Dirichlet and Neumann conditions, cell-face derivatives at the boundary are evaluated using the neighbouring cell-centre values, and the updated boundary increments ($\Delta\boldsymbol{w}$, $\Delta\boldsymbol{\theta}$) are used in the Newton–Raphson iterations to update the displacement and rotation fields. The available boundary conditions for the `beamFoam` solver are discussed in Section 5.3.

3.6. **Discretised system of equations.** The final form of the discretised equilibrium equations for a typical computational node P read as follows,

$$\overset{*}{\boldsymbol{J}}_{\mathrm{P}} \begin{bmatrix} (\Delta\boldsymbol{w})_{\mathrm{P}} \\ (\Delta\boldsymbol{\theta})_{\mathrm{P}} \end{bmatrix} + \sum_{\mathrm{N}} \overset{*}{\boldsymbol{J}}_{\mathrm{N}} \begin{bmatrix} (\Delta\boldsymbol{w})_{\mathrm{N}} \\ (\Delta\boldsymbol{\theta})_{\mathrm{N}} \end{bmatrix} = \begin{bmatrix} (\overset{*}{\boldsymbol{R}}_{\boldsymbol{w}})_{\mathrm{P}} \\ (\overset{*}{\boldsymbol{R}}_{\boldsymbol{\theta}})_{\mathrm{P}} \end{bmatrix} \tag{16}$$

where $\overset{*}{\boldsymbol{J}}_{\mathrm{P}}$ is the 6×6 Jacobian matrix containing the contributions of node P while $\overset{*}{\boldsymbol{J}}_{\mathrm{N}}$ contains the neighbouring cell contributions. The right-hand side of the Eqn. 16 is the source vector containing the explicit discretised terms and boundary condition contributions.

The linearised Eqn. 16 are assembled for all CVs forming a system of equations given by,

$$\left[\overset{*}{\boldsymbol{J}}\right]\left[\Delta\boldsymbol{x}\right] = \left[\overset{*}{\boldsymbol{R}}\right] \tag{17}$$

resulting in $6M \times 6M$ sparse matrix $\left[\overset{*}{\boldsymbol{J}}\right]$ evaluated in the previous Newton iteration k at time t_{n+1}, where M is the total number of CVs. The coefficients in $\overset{*}{\boldsymbol{J}}_{\mathrm{P}}$ constitute the diagonal of $\left[\overset{*}{\boldsymbol{J}}\right]$ whereas matrix $\overset{*}{\boldsymbol{J}}_{\mathrm{N}}$ contributes to its off-diagonal terms. The solution vector $\left[\Delta\boldsymbol{x}\right]$ contains the primary unknowns $\Delta\boldsymbol{w}$ and $\Delta\boldsymbol{\theta}$. The final system of linearised algebraic equations, obtained by assembling Eqn. 16 for all control volumes in the mesh is solved using the Eigen solver [38].

For each time increment, the coupled equations are solved iteratively using a Newton–Raphson procedure until a user-defined convergence tolerance is satisfied. Convergence is assessed by monitoring both the Euclidean norm of the solution increment ($|\Delta\boldsymbol{x}|$) and the residual of the linear system ($|\overset{*}{\boldsymbol{R}}|$) (see the end paragraph of Section 4 for more details).

4. Implementation of the beam solver

The `beamFoam` repository is organised into the `applications`, `src`, and `tutorials` directories as outlined in Fig. 5.

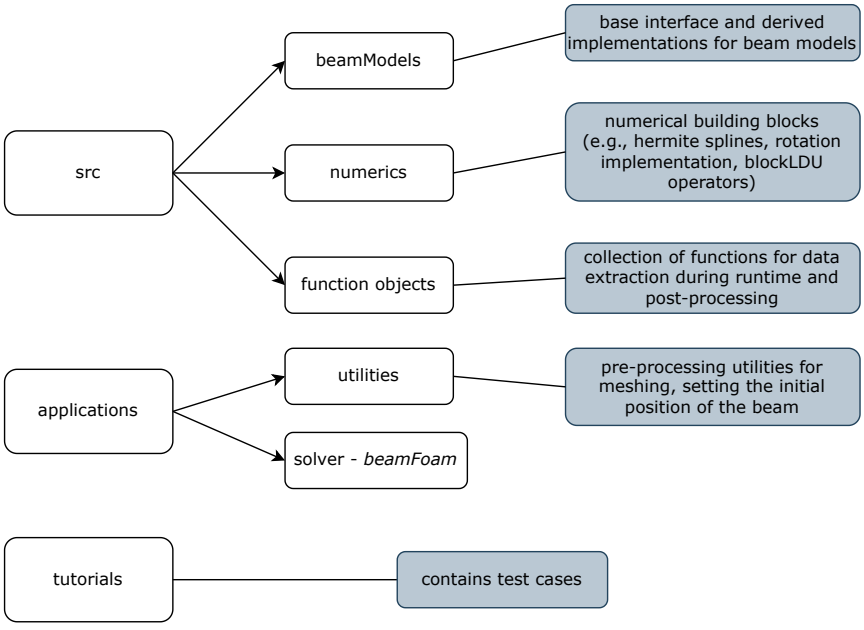

Figure 5. Structure of `beamFoam` solver

The application `beamFoam` drives each simulation; the `src` tree contains the library implementation and related components (e.g., numerics, functionObjects); and test cases reside under the `tutorial` directory. Within `src`, the `beamModels` subfolder provides a base interface and derived implementations; OpenFOAM's run-time selection is used to choose the active beam model at run time. In the current code base, the available model is `coupledTotalLagNewtonRaphsonBeam`, which implements the finite-volume formulation of geometrically exact Simo–Reissner beams. To make the solver extensible and to decouple physics from geometry and boundary data, the implementation adopts a modular, object-oriented hierarchy as depicted in Fig. 6.

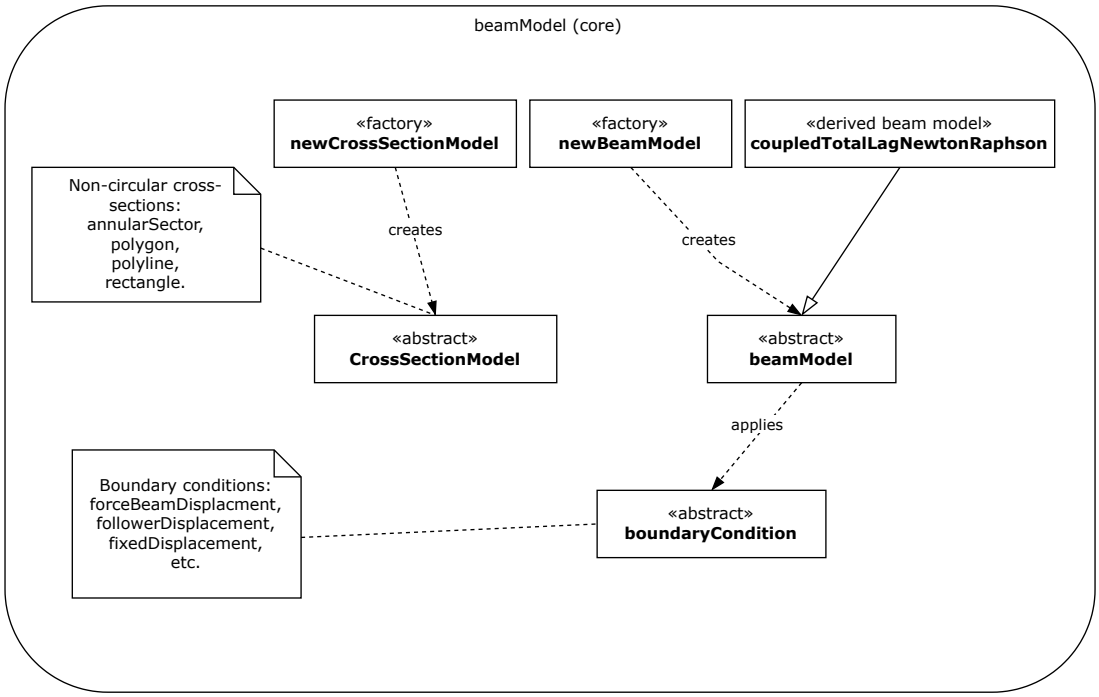

Figure 6. `beamModel` hierarchy with cross-sections and boundary condition

`beamModel` is an abstract base that defines the solver interface used by `beamFoam.C`. All beam solvers inherit from this base and provide model-specific behaviour, and the active class is selected at run-time via the run-time selection tables. `coupledTotalLagNewtonRaphsonBeam` is a derived implementation that implements the total Lagrangian Newton-Raphson algorithm for the Simo-Reissner formulation within the finite-volume framework. Cross-section geometry is supplied by an abstract cross-section interface (`crossSectionModel`), and the active section is created by the `newCrossSectionModel` factory from its dictionary entry. Boundary data are provided by fvPatchField-based classes (boundaryCondition) registered in the run-time selection tables. This modular organisation allows Dirichlet or Neumann boundary conditions and time-series inputs to be interchanged without altering the solver core. As a result of this design, new beam algorithms, cross-sections, or boundary conditions can be added independently and selected via dictionaries, while `beamFoam.C` remains unchanged.

Within the applications directory, the `beamFoam` solver (`beamFoam.C`) is the main solver for the beam formulation (see Listing 1). At run time, it constructs a polymorphic pointer to the base class and advances the solution by calling three abstract hooks: `evolve()`, `updateTotalFields()`, and `writeFields()`. In Listing 1, `beam().evolve()` is the primary function call that assembles the discretised governing equations and solves them using a Newton-Raphson iterative procedure. The object `beam()` takes the solver to the appropriate beam model type in the source and executes the `evolve()` function of the chosen beamModel class. The finite volume formulation implemented for Simo-Reissner beams is a total Lagrangian approach, therefore `beam().updateTotalFields()` computes the cumulative total displacements and stores them in the `pointW` field, which is used to visualise mesh deformation. In the end, `writeFields()` writes outputs to disk by creating time directories that store the solution fields (e.g., displacements and rotations) for each step.

```
#include "beamModel.H"
int main()
{
    Foam::autoPtr<Foam::beamModel> beam =
        Foam::beamModel::New(runTime, Foam::dynamicFvMesh::defaultRegion);
    while (runTime.loop())
    {
        beam().evolve();
        beam().updateTotalFields();
        beam().writeFields();
    }
}
```

Listing 1. Coding implementation of `beamFoam.C` application file

Listing 2 illustrates the main stages of the `evolve()` member function of `beamFoam.C` file. In Step 1, the fields `d`, `u`, `l`, and `source` are created and initialised to zero, matching the beam mesh information. his step sets up the containers required for assembling the block-coupled system (Eqn. 17), which are populated in the subsequent steps.

```
scalar coupledTotalLagNewtonRaphson::evolve
{
    do
    {
        // Step 1 - Initialise l, d, u, source
        Field<scalarSquareMatrix> d
        (
            mesh().nCells(), scalarSquareMatrix(6, 0.0)
        );
        // Repeat above code for u and l initialisation
        Field<scalarRectangularMatrix> source
        (
            mesh().nCells(), scalarRectangularMatrix(6, 1, 0.0)
        );
        // Step 2 - Update the matrix coefficients to populate Jacobian
        updateEqnCoefficients();
        // Step 3 - Assembling the Jacobian matrix
        assembleMatrixCoefficients(d, l, u, source);
        // Step 4 - Add inertia contributions
        if (d2dt2SchemeName_ != steadyState)
        {
            // Consistent fvSchemes check for Euler and Newmark
            // followed by addition of inertia terms
        }
        // Step 5 - Solving the equations
        BlockEigenSolverOF eigenSolver(d, l, u, own, nei);
        // Step 6 - Updating the solution fields
        updateSolutionVariables();
    }
    while
    {
        // Convergence criteria
        !checkConvergence(...)
    }
}
```

Listing 2. Coding details of the `evolve()` member function in the `beamFoam.C` file

The block-coupled system for the translational ($\boldsymbol{w}$) and rotational ($\boldsymbol{\theta}$) degrees of freedom is assembled in the LDU (lower–diagonal–upper) format, storing only non-zero coefficients, thereby reducing memory usage and improving computational efficiency for sparse matrices. The diagonal (`d`), upper (`u`), and lower

(`l`) coefficient fields each contain 6×6 sub-matrices that capture the coupling between translational and rotational components. The `source` vector is defined as a 6×1 block for each beam cell, representing the right-hand side $\overset{*}{\boldsymbol{R}}$ of the system. It is initialised to zero in Step 1 and subsequently populated with the explicit terms from the linearised force and moment equilibrium equations together with external loads and boundary conditions.

In Step 2 of Listing 2, the function `updateEqnCoefficients()` updates the coefficient fields of the governing equations based on the current kinematic state of the beam. These updated quantities provide the entries for the block matrix and residual used in the assembly step.

The function `assembleMatrixCoefficients()` fills the block-coupled system with the terms obtained in the previous step (Step 3 of Listing 2). For each internal face, the owner–neighbour addressing is used to distribute contributions to the diagonal (`d`), upper (`u`), and lower (`l`) blocks. The explicit load terms are also inserted into the `source` vector as 6×1 blocks. Through this process, the local stiffness and coupling terms are consistently mapped into the global block matrix, while the right-hand side is built from the explicit forces and moments. In effect, this stage translates the updated coefficient fields into the discrete LDU system that will be solved in subsequent steps. A condensed version of this procedure is illustrated in Listing 3, where the assembly call is shown in its simplified form.

```
void coupledTotalLagNewtonRaphsonBeam::assembleMatrixCoefficients
(
    Field<scalarSquareMatrix>& d,
    Field<scalarSquareMatrix>& l,
    Field<scalarSquareMatrix>& u,
    Field<scalarRectangularMatrix>& source
)
{
    forAll(u, faceI)
    {
        // for each component (i,j) in the 6x6 block:
        u[faceI](i,j) += ... ;
        d[own[faceI]](i,j) += ... ;
        l[faceI](i,j) += ... ;

        // for each entry in the 6x1 residual block:
        source[nei[faceI]](i,0) -= ... ;
    }
}
```

Listing 3. Implementation details of the sub-member function `assembleMatrixCoefficients` in `beam().evolve()` function

Once the complete LDU structure and source vector are populated, they form the block-coupled 6×6 system for each beam segment as,

$$\underbrace{\begin{bmatrix} \boldsymbol{D}_0 & \boldsymbol{U}_0 & & \\ \boldsymbol{L}_0 & \boldsymbol{D}_1 & \boldsymbol{U}_1 & \\ & \boldsymbol{L}_1 & \boldsymbol{D}_2 & \ddots \\ & & \ddots & \ddots \end{bmatrix}}_{\text{LDU structure}} \underbrace{\begin{bmatrix} \Delta\boldsymbol{x}_0 \\ \Delta\boldsymbol{x}_1 \\ \Delta\boldsymbol{x}_3 \\ \vdots \end{bmatrix}}_{\text{corrections}} = \underbrace{\begin{bmatrix} \overset{*}{\boldsymbol{R}}_0 \\ \overset{*}{\boldsymbol{R}}_1 \\ \vdots \end{bmatrix}}_{\text{source vector}}$$

Here, $\boldsymbol{D}_i, \boldsymbol{L}_i, \boldsymbol{U}_i \in \mathbb{R}^{6\times6}$ represent the diagonal, lower, and upper coupling blocks for beam segment i, while $\Delta\boldsymbol{x}_i$ and $\overset{*}{\boldsymbol{R}}_i$ are the corresponding 6×1 vectors. The vector of unknowns is written as $\Delta\boldsymbol{x}_i = (\Delta w_1, \Delta w_2, \Delta w_3, \Delta\theta_1, \Delta\theta_2, \Delta\theta_3)^\top$, which collects the translational and rotational increments, with $\overset{*}{\boldsymbol{R}}_i$ denoting the associated source block.

The LDU coefficient fields are then handed to the `BlockEigenSolverOF` class, which takes `d`, `l`, and `u` together with the owner–neighbour addressing and converts them into an Eigen-compatible sparse matrix (Step 5 in Listing 2). Triplet storage (`Eigen::Triplet` is used, with each 6×6 block mapped to its correct position according to the addressing lists. The `source` vector is flattened into a global column vector

following the same ordering. An initial residual is computed as the norm of $\overset{*}{\boldsymbol{J}}\Delta\boldsymbol{x} - \overset{*}{\boldsymbol{R}}$ ($\overset{*}{\boldsymbol{J}}$ is the LDU (Jacobian) matrix), providing a measure of the equilibrium error before the solve.

Depending on the selected configuration, the system is solved using one of several Eigen backends, with `SparseLU` as the default choice. The solver analyses the sparsity pattern, performs LU factorisation, and executes a single direct solve. After the solution vector is obtained, it is mapped back into the OpenFOAM fields, restoring the 6×1 block format for each beam cell. Then, the solution variables and the derived quantities (Eqn. 12) are updated in the function `updateSolutionVariables` (Step 6 in Listing 2) of the code.

At present, the `evolve()` member function operates in a single processor. However, all field variables use OpenFOAM's parallel-aware data structures (`volField`, `surfaceField`), so domain decomposition via `decomposePar` is inherently supported. The computational cost of the beam solver is low (typically on the order of seconds for all benchmark cases); hence, a serial implementation was found sufficient for the present study. For future multi-physics simulations, such as fluid-beam interaction problems where the fluid domain dominates computational expense, the beam solver can efficiently be executed in serial on the master processor while the fluid solver runs in parallel.

OpenFOAM's built-in iterative solvers (e.g., PBiCG, PCG, GAMG) are designed for weakly coupled scalar or tensor systems common in fluid and thermal analyses. In contrast, the Simo-Reissner beam equations produce a strongly coupled, non-symmetric block system. As OpenFOAM.org and ESI versions lack a native block-matrix solver, these iterative schemes perform poorly for such systems, motivating the use of the Eigen SparseLU direct solver in the present implementation.

The outer Newton-Raphson loop iterates until the convergence criteria are met. Three convergence criteria are checked within each time (or loading) step,

- **Residual norm** $\|\overset{*}{\boldsymbol{R}}\|$ of the equilibrium equations:
  - Absolute norm - the convergence criteria is met when $\|\overset{*}{\boldsymbol{R}}\|$ falls below an absolute tolerance `absoluteTol`, taken here as 1×10^{-30}. This criterion serves as a safeguard in cases where the relative criterion is ineffective, such as when the initial solution already nearly satisfies the governing equations.
  - Relative norm - the convergence criteria is met when $\|\overset{*}{\boldsymbol{R}}\|$ falls below `residualTol` $\times \|\overset{*}{\boldsymbol{R}}_0\|$, where $\|\overset{*}{\boldsymbol{R}}_0\|$ is the initial residual norm computed as the norm of $\overset{*}{\boldsymbol{J}}\Delta\boldsymbol{x} - \overset{*}{\boldsymbol{R}}$. The default value of `residualTol` is set to 1×10^{-8}.
- **Solution correction norm** $\|\Delta\boldsymbol{x}\|$- the convergence criteria is met when the change in the solution between successive iterations falls below `solutionTol` $\times \|\Delta\boldsymbol{x}\|$. the default value of `solutionTol` is set to 1×10^{-10}.

5. Steps to simulate a test case using the `beamFoam` solver

The numerical simulation for beams in `beamFoam` typically involves three main stages:

- **Pre-processing**:
  - **Geometry and mesh**: A custom meshing utility, `createBeamMesh`, is used to generate a `polyMesh` directory tailored to the beam geometry and compatible with the `OpenFOAM` environment (Section 5.1). A 3-D representation of the cross-section is included solely to enable realistic visualisation in ParaView for post-processing purposes.
  - **Setting the initial configuration of the beam**: If the simulation involves a straight beam whose centroid line is not aligned with the global x-axis, a secondary utility, `setInitialPositionBeam`, must be executed to correctly initialise the undeformed centreline and the associated rotation matrix of the beam cross-section (Section 5.2).
  - **Physical properties**: Following the standard OpenFOAM® input structure, the material and geometric properties of the beam are specified in `constant/beamProperties` (Section 5.4).
  - **Initial and boundary conditions**: The primary variables in the `beamFoam` solver are displacements `W` and rotation vectors `Theta`, and the initial and boundary conditions for these fields are specified in the `0` directory. (Section 5.3).

- **Solver execution**: The application used to solve for beam deformation is `beamFoam`. The `system` directory (Section 5.4) contains standard control files as used in other OpenFOAM® cases (e.g., `controlDict`, `fvSchemes`, `fvSolution`).

- **Post-processing**: Simulation results are visualised in typical OpenFOAM fashion, e.g., using ParaView. Various `functionObjects` (Section 5.5) are available to extract quantities such as displacements, internal forces, moments at specific patches, and system energy. Numerical diagnostics such as convergence information from the beam linearisation loop can also be recorded.

The `Allrun` file of a typical beam test case is shown in Listing 4.

```
# Generates beam mesh aligned with global x-axis and creates the polyMesh/ folder
runApplication createBeamMesh

# Secondary utility to set the initial configuration of beam
runApplication setInitialPositionBeam -cellZone beam_0 \
    -translate "(x y z)" -rotateAngle "((x y z) angleInDegrees)"

# Executes the beamFoam application
runApplication beamFoam
```

Listing 4. Code excerpt for `Allrun` in a typical beam test case

5.1. **The `createBeamMesh` utility for geometry and mesh creation.** The utility `createBeamMesh` is a custom mesh generation tool for the `beamFoam` solver which creates a `polyMesh/` directory with necessary mesh information compatible with OpenFOAM® environment. The 3-D cross-section is also defined for visualisation in post-processing.

The utility reads user-defined parameters from the dictionary `beamProperties` located in the `constant/` directory. Each beam is specified as a dictionary entry under the `beams` list as shown in Listing 5.

```
beams
{
    beam_0
    {
        crossSectionModel   circle; // Cross-section type
        circleCrossSectionModelDict // Sub-dictionary
        {
                radius    0.01; // Cross-section specific parameters
        }
        length    1.0;  // Beam length
        nSegments    100; // Number of cells across length
        // Additional local parameters like Youngs' modulus and shear modulus
        // E    200e9;   // Reads as scalar
        // G    100e9;
    }
        ... // Add more beams here
}
```

Listing 5. Code excerpt for defining beam properties to create beam mesh

The utility `createBeamMesh` reads beam parameters from the `beamProperties` file inside `constant` directory and generates polygonal faces for one or more beams. For each beam, the cross-sectional geometry is defined using a parametric model through the `crossSectionModel` argument. The list of available cross-sections for the solver is provided in Tab. 1.
Each `crossSectionModel` has a sub-dictionary where the cross-section parameters (e.g., radius of circle) can be specified. The cross-section is extruded along the beam's mean line (x-axis) to construct a 3-D point cloud. Faces are then assembled to form the different patches for beam geometry and cell connectivity of each face is defined. Boundary patches are created - either as `left`, `right`, and `beam`

Type of cross-section	Parameters to be specified
`circle`	`radius` - radius of circle
`rectangle`	width - `b`, height - `h`
`annularSector`	inner radius - `innerRadius`, outer radius - `outerRadius`

Table 1. Supported cross-section types for the `beamFoam` solver

for single-beam cases, or individually named patches like `left\_k` and `right\_k` for each beam with a shared `beam` patch in multi-beam cases. Additionally, a `cellZone` file is generated to distinguish each beam, and the final mesh is written to OpenFOAM's `polyMesh` format.

The output of `createBeamMesh` includes:

- A structured `polyMesh` with `points`, `faces`, `owner`/ `neighbour` arrays.
- A `boundary` file with appropriate patch definitions.
- `pointZones` and `cellZones` for each beam.

5.2. **Define initial configuration of beams.** To move the beam(s) from the reference position or to set the initial curvature of beam(s) correctly, additional utilities must be used. Each utility code is associated with a command and a set of arguments that needs to provided as input by the user in the `Allrun` script to set the initial configuration of beam(s). These utilities move the circumferential points of the straight beam to the desired location and set the initial mean centroid line, $\boldsymbol{r}_0(s)$, and the tangents of the beam centreline, $\boldsymbol{r}_0'(s) \equiv \boldsymbol{g}_1^0(s)$. They also set the initial orientation of the cross-section faces via the rotation matrix $\boldsymbol{\Lambda}_0(s)$. For visualisation of the initial configuration of the beam(s) in the post-processing step, the information of the displacement field in the initial (stress-free) configuration is stored in the `pointW` field. The available utility to set the initial configuration of the beam(s) is,

`setInitialPositionBeam` - this command allows the user to translate the beam by a desired translation vector and rotate the beam via the Rodrigues' rotation formula (Eqn. 1), where the user must specify the axis of rotation and magnitude of rotation angle in degrees. The `setInitialPositionBeam` command in the `Allrun` script looks as shown in Listing 6.

```
setInitialPositionBeam -cellZone beam_0 -translate "(x y z)" \
    -rotateAngle "((a b c) angleInDegrees)"
```

Listing 6. Code excerpt for `setInitialPosition` command

Here, the `cellZone` argument is used to grab the mesh information of a particular beam and apply the translation via `-translate` and rotation via `-rotateAngle` to the mesh of that beam.

Running the command `setInitialPositionBeam` creates several fields in the `0` directory that define the beam's initial configuration relative to the global x-axis. These include the displacement fields (`refW`, `refWf`), the tangent to the centreline (`refTangent`), and the rotation matrices for the cross-section (`reflambda`, `refLambdaf`). The solver then uses these fields as inputs.

5.3. **Specifying the `0` directory.** For the `beamFoam` solver, the mandatory input variables are displacements (`W`) and rotation (`Theta`) vector fields. Optionally, externally applied distributed forces ($\texttt{q} \equiv \bar{\boldsymbol{q}}$ in Eqn. 14) and moments ($\texttt{m} \equiv \bar{\boldsymbol{m}}$ in Eqn. 14) vectors can also be specified for a test case.

For the `beamFoam` solver, several Dirichlet and Neumann boundary conditions for `W` and `Theta` fields are developed. A list of available boundary conditions is given in Tab. 2. Numerical test cases in Section 6 show the usage of most boundary conditions developed for the solver.

5.4. **Specifying the `constant` and `system` directory.** The `constant` directory contains details of the beam's physical properties in `beamProperties` file and the information related to mesh stored in the `polyMesh` subdirectory. Listing 7 shows the typical details required in `beamProperties` file.

Keyword	Type	Description
`fixedValue`	Dirichlet	displacement/rotation BC
`fixedDisplacement`	Dirichlet	time-varying displacement BC
`fixedRotation`	Dirichlet	time-varying rotation BC
`forceBeamDisplacementNR`	Neumann	non-circulatory force BC
`followerForceBeamDisplacementNR`	Neumann	circulatory force BC
`momentBeamRotationNR`	Neumann	non-circulatory moment BC

Table 2. Supported boundary conditions (BCs) available for the `beamFoam` solver

```
// runTime selection of beam model
beamModel coupledTotalLagNewtonRaphsonBeam;
// List of beams
beams
(
    // List of beams, cross-section type and parameters,
    // and variables local to each beam like E, G, rho,
    // length, and nSegments

);
// Variables specific to selected beam solver
coupledTotalLagNewtonRaphsonBeamCoeffs
{
    // Global variables common to all beams
    E E [1 -1 -2 0 0 0 0] 200e9; // Young's modulus
    G G [1 -1 -2 0 0 0 0] 100e9; // Shear Modulus
    rho rho [1 -3 0 0 0 0 0] 1; // Density

    // Maximum allowed outer (Newton) iterations
    nCorrectors 200;

    // Convergence tolerance for solution correction norm and residual norm
    solutionTol 1e-10;
    residualTol 1e-8;
}
```

Listing 7. `beamProperties` file

In addition to this, the `constant` directory also contains the input text files for time-varying boundary conditions that are specified for `W` and `Theta` field.

The `system` directory contains the standard `controlDict` settings, and the valid time-integration schemes implemented in the `beamFoam` solver can be specified in `fvSchemes` file (Tab. 3).

Time-integration scheme	keyword for `ddtSchemes` and `d2dt2Schemes`
steady-state	`steadyState`
first-order Euler	`Euler`
second-order Newmark-β	`Newmark`

Table 3. Supported `fvSchemes` for the `beamFoam` solver

5.5. **Post-processing visualisation and utilities.** The numerical results generated by the beam solver can be visualised in standard OpenFOAM format, e.g., the open-source software ParaView is used [39] for this work. Since `beamFoam` employs a total Lagrangian formulation, the beam mesh does not physically deform. Therefore, to display the deformed configuration in ParaView, the *WarpByVector* filter must be applied using the vector field `pointW`, which stores the displacement of the beam mesh at all time steps. This allows the deformed beam shape to be viewed at any selected time step. Other output fields, such as the displacement field `W`, can also be visualised along the beam length with adjustable colour maps.

To extract the simulation data like displacements, and forces after simulation, several `functionObjects` as presented in Tab. 4 are available. An example implementation of the `functionObjects` in `system/controlDict` of a typical test case is provided in Listing 8.

Keyword	Description
`beamDisplacements`	Extracts x, y, z displacement components and magnitude at the specified end patches
`beamForcesMoments`	Extracts x, y, z forces and moment components at the specified end patches
`beamConvergenceData`	Reports the number of Newton-Raphson iterations to solve the nonlinear beam equations at each time step
`beamEnergyData`	Calculates and reports numerical potential, kinetic and total energy components at each time step

Table 4. Supported `functionObjects` for the `beamFoam` solver

```
functions
{
   beamDisplacements1
   {
       type    beamDisplacements;
       historyPatch    right;
   }
   beamConvergenceData
   {
       type    beamConvergenceData;
   }
   beamForcesMoments1
   {
       type    beamForcesMoments;
       historyPatch    left;
   }
   beamEnergyData
   {
       type    beamEnergyData;
   }
}
```

Listing 8. Code excerpt for `functionObjects` call in `system/controlDict`

After the simulation of a case is finished, the primary variables and other derived quantities are written to the output disk (Tab. 5).

6. Numerical Cases

Three benchmark cases are investigated in this section:

(1) 3-D vibration of a rectangular cantilever column — a large deformation benchmark where `beamFoam` results are compared against the `solids4Foam` toolbox and the commercial software ABAQUS, demonstrating both accuracy and computational efficiency.
(2) In-plane cantilever under a circulatory follower end force — a quasi-static 2-D test case with a special boundary condition, where reference results from literature are compared with `beamFoam`, along with a mesh convergence study.
(3) 3-D free vibration of a flexible beam with free ends — a dynamic case where the beam is subjected to initial forces and moments, used to assess the stability of the time-integration schemes and energy conservation over long simulations.

All simulations are executed in serial on an Apple M1 Pro (8-core CPU, 16 GB RAM) running macOS Ventura 13.5.2. The solver is compiled using AppleClang 14.0.0 (ARM64) with OpenFOAM-v2306. For all the three test cases, the default convergence tolerances for residuals and solution increments are used, i.e., `residualTol` $= 1 \times 10^{-8}$ and `solutionTol` $= 1 \times 10^{-10}$.

Name	Description	Dimensions
`W`	Displacement - `volVectorField`	m
`Theta`	Rotation - `volVectorField`	dimensionless
`Q`	Resultant Force - `surfaceVectorField`	N
`M`	Resultant moment - `surfaceVectorField`	Nm
`q`	Distributed forces per unit length - `volVectorField`	N/m
`m`	Distributed moments per unit length - `volVectorField`	N
`Gamma`	Translation strains - `surfaceVectorField`	dimensionless
`K`	Rotational strains - `surfaceVectorField`	m^{-1}
`pointW`	Total deformation of the beam from reference configuration stored for visualisation - `volVectorField`	m

Table 5. Output variables written to the disk after running the `beamFoam` solver

6.1. **3-D vibration of a dynamic cantilever.** The first test case is replicated from the recent article by Cardiff et al. [40]. It employs a neo-Hookean hyperelastic solid solver from the `solids4Foam` toolbox [34] and uses a Jacobian free Newton Krylov (JFNK) approach to solve the governing equations. This solver is a fully 3-D formulation that resolves all three displacement components in each cell, with the geometry discretised in three spatial dimensions. The same test case is provided in the tutorials repository of the beam solver under the name `3DdynamicCantilever`. It consists of a dynamic cantilever column with a square cross-section of side 0.2 m and a length of 2 m (see Listing 9 for the cross-section excerpt). A sudden constant force, $\boldsymbol{q} = (2000, 2000, 0)$ N, is applied to the top surface of the cantilever from $t = 0$ to $t = 1$ s, inducing large deformations and finite strains in the column.

```
beams
(
    beam_0
    {
            crossSectionModel rectangle;

            rectangleCrossSectionModelDict
        {
            b    0.2;
            h    0.2;
            nx    1;
            ny    1;
        }

        length    2;
        nSegments    20;
        E    15.293e6;
        G    5.882e6;
    }
);
```

Listing 9. 3D Dynamic Cantilever - The properties and settings are defined in the `constant/beamProperties`

The bottom (`left`) patch is clamped, i.e., both displacement `W` and rotation `Theta` are set to `fixedValue - value uniform (0 0 0)`. The top (`right`) patch is a free edge where the specified force is applied but the moments at the free edge are zero (Listing 10).

The material properties assumed are $E = 15.293$ MPa, $\nu = 0.3$ and density $\rho = 1000$ kg/m^3. For the cantilever, the dominant natural frequency is the first bending mode, $\omega_n = 1.875^2\sqrt{\frac{EI}{\rho AL^4}} = 6.28$ s^{-1}. Therefore the time-period $T_p = \frac{2\pi}{\omega_n} \approx 1$ s. Therefore, Δt should be chosen such that atleast $20-50$ points

are resolved for the vibration time-period of 1 s to achieve accurate results, i.e., $\Delta t \leq \frac{T_p}{50} \leq 0.02$ s. To be consistent with Cardiff et al. [40] results, a `deltaT`= 1 ms and a second-order accurate Nemmark-β time scheme is employed. The Newmark scheme is defined in `fvSchemes` as shown in Listing 11.

```
// W boundary condition
right // top edge
{
    type forceBeamDisplacementNR;

    forceSeries
    {
        "fileName|file"        "$FOAM_CASE/constant/timeVsForce";
        outOfBounds     clamp;
    }
}
// timeVsForce file in constant folder
(
    ( 0 (2000 2000 0) )
    ( 1 (2000 2000 0 ) )
)
// Theta boundary condition
right // top edge
{
    type momentBeamRotationNR;

    momentSeries
    {
        "fileName|file"        "$FOAM_CASE/constant/timeVsMoment";
        outOfBounds     clamp;
    }
}
// timeVsMoment file in constant folder
(
    ( 0 (0 0 0) )
    ( 1 (0 0 0 ) )
)
```

Listing 10. 3D Dynamic Cantilever - boundary conditions

```
ddtScheme
{
        ddt(W)  Newmark;
}
d2dt2Scheme
{
        d2dt2(W)          Newmark;
        newmarkBeta(W)    0.25; // Default value
        newmarkGamma(W)   0.5; // Default value
}
```

Listing 11. 3D Dynamic Cantilever - Setting up of Newmark-β time scheme in `fvSchemes`

To run this test case, the mesh of the beam is first created using `createBeamMesh` utility which generates a straight beam with cross-section normal parallel global x-axis. To rotate the beam into vertical position, `setInitialPositionBeam` utility is executed as shown in Listing 12 followed by the `beamFoam` application.

```
setInitialPositionBeam -cellZone beam_0 -translate "(0 0 0)" \
    -rotateAngle "((0 -1 0) 90)"
```

Listing 12. 3D Dynamic Cantilever - `setInitialPositionBeam` command

The deformed configuration of the beam at six time steps is shown in Fig. 7 for the 20 mesh segments, where the upper surface is seen to drop below the horizontal ground level at $t = 0.3$ s.

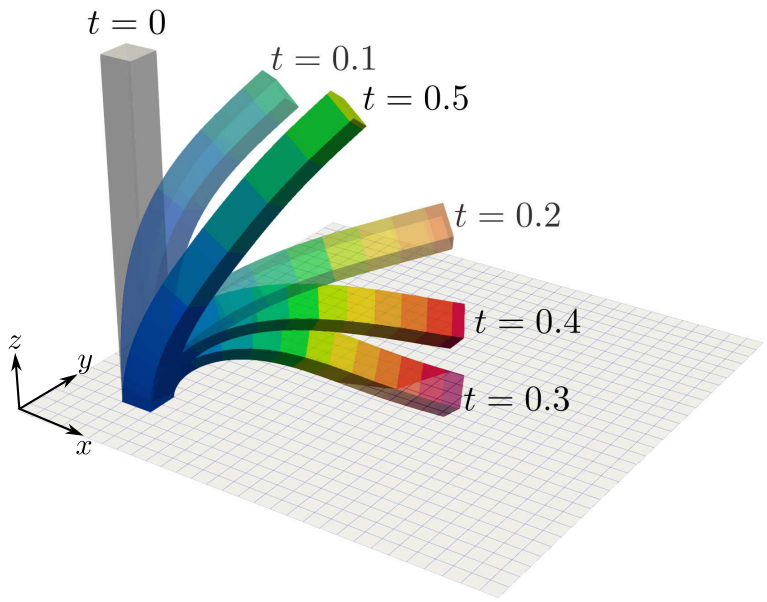

Figure 7. 3D dynamic cantilever: deformed shapes at different times

Figure 8 shows the displacement magnitude at the face centre of the top (`right`) patch versus time for three mesh resolutions - (5, 10, and 20 mesh segments) using the Newmark-β time scheme with a timestep of $\Delta t = 1$ ms. An average of 4 outer (Newton) iterations are required for the residuals to converge. Results from `beamFoam` are compared against the finest-mesh (138240 total cells) solution obtained with the `solids4Foam` toolbox [40] and reference results from finite element software ABAQUS (C3D8 elements and 138240 mesh).

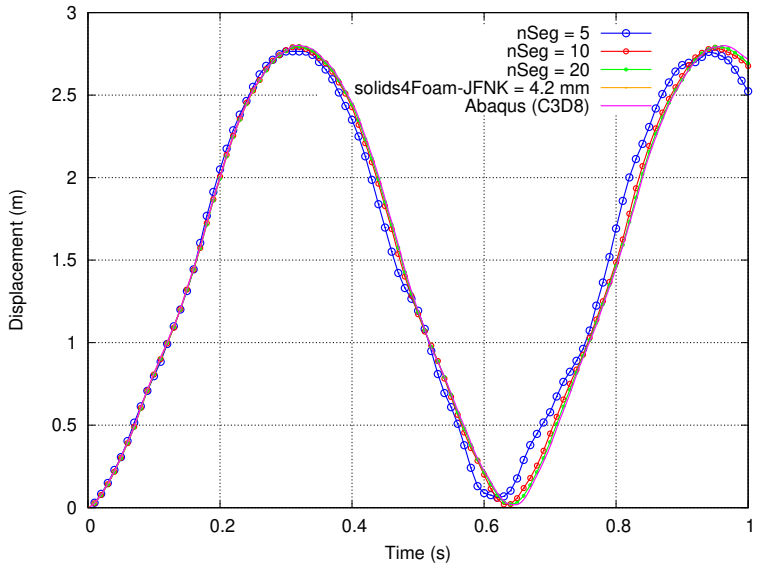

Figure 8. 3D dynamic cantilever: Comparison of displacement magnitude at the centre of top (`right`) edge with `solids4Foam` and ABAQUS results for varying mesh discretisation (nSeg shows the number of segments) using Newmark method, $\Delta t = 1$ ms

As evident in Fig. 8, the displacement predictions of the coarsest mesh (5 cells) in the beam cells provide predictions which are close to the reference results. The results converge to the `solids4Foam` and

ABAQUS solution for the finest mesh of 20 mesh segments. The comparison of execution times of the beam solver against other solvers is presented in Tab. 6. The time taken by beam solver is observed to be orders of magnitude faster than `solids4Foam` and ABAQUS.

Keyword	Execution Time (s)
beam - 5 segments	0.89
beam - 10 segments	1.01
beam - 20 segments	1.26
`solids4Foam`-JNFK solver	12156
ABAQUS	9240

Table 6. Comparison of execution times of `beamFoam` against `solids4Foam` and ABAQUS results for varying mesh discretisation using Newmark method, $\Delta t = 1$ ms

Both Euler and Newmark-β time integration schemes implemented in the `beamFoam` solver are implicit, and therefore there is no Courant number criterion to check the stability of the time-step Δt. However, coarser time-steps can lead to less accurate results. For this test case, the effect of time step on the results is investigated by fixing the mesh size to 20 segments and varying the Δt for both Newmark and Euler time-integration schemes.

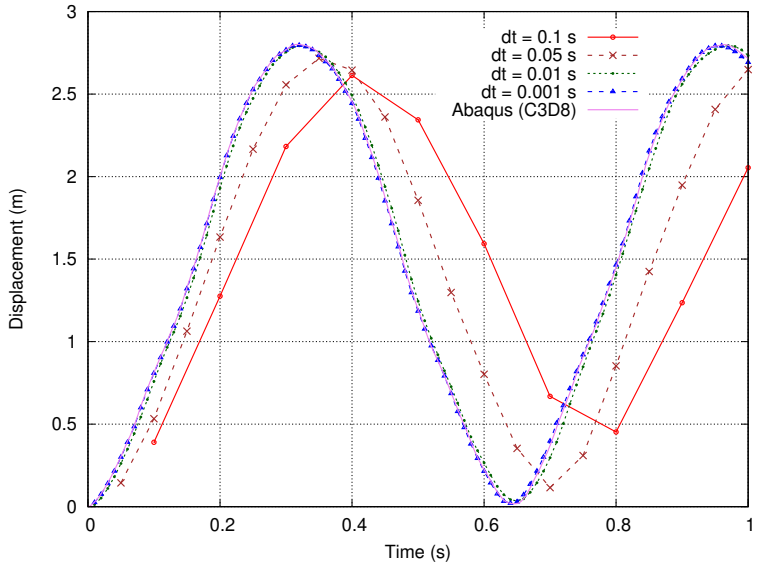

Figure 9. 3D dynamic cantilever: Comparison of displacement magnitude for a fixed mesh size of 20 segments but a varying Δt using Newmark time integration method

Figure 9 shows the displacement graphs for Newmark-β time scheme for $\Delta t = 0.1, 0.05, 0.01$, and 0.001 s. Since the optimal time step size as per the first bending mode is $\Delta t \leq 0.02$ s, the results corresponding to $\Delta t = 0.1, 0.05$ s are less accurate (but stable) and have both damping and phase errors. The results converge to the reference solution of ABAQUS for $\Delta t = 0.01$, and 0.001 s. In contrast, Fig. 10 shows the displacement variation with time for the Euler time scheme for $\Delta t = 0.1, 0.05, 0.01, 0.001$ and 0.0001 s.

One clear difference between the two schemes is that the first-order accurate Euler method causes large numerical dissipation for the coarser time steps, but the predictions converge to the reference solution as the time step is reduced, e.g., $\Delta t = 0.1$ ms. When using the Newmark scheme (Fig. 9), the overall oscillatory motion of the cantilever is better captured for a given time step size, and a time-step $\Delta t = 0.01$ s is visibly indistinguishable from the reference results.

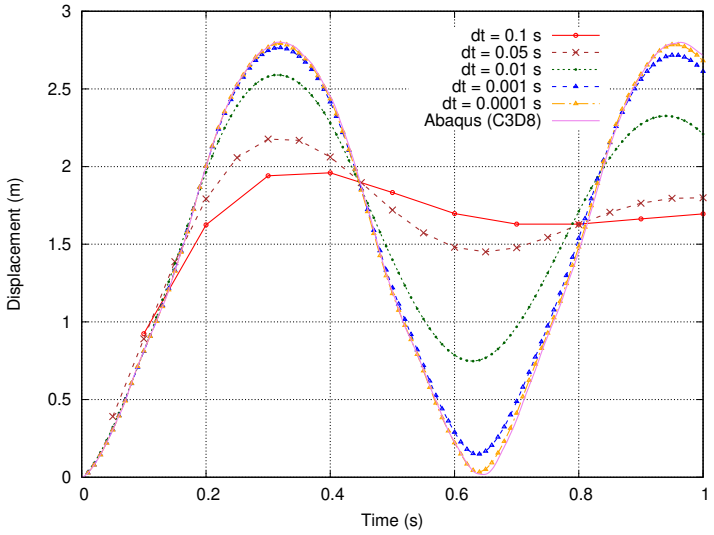

Figure 10. 3D dynamic cantilever: Comparison of displacement magnitude for a fixed mesh size of 20 segments but a varying Δt using backwards (implicit) Euler time integration method

6.2. **In-plane quasi-static cantilever subjected to end follower force.** The second test case is an in-plane quasi-static benchmark originally studied by Argyris et al. [41], and later by Simo and Vu-Quoc [29]. The test case is provided in the tutorials repository under the name `cantileverFollowerForce`. A straight cantilever beam of length $L = 100$ m, clamped at the left end, is subjected at the free end to a circulatory following force of magnitude $\|\boldsymbol{q}\| = 134$ kN. A circulatory follower load is a non-conservative force whose direction follows the instantaneous deformation of the structure, resulting in a configuration-dependent loading that cannot be derived from a potential and may lead to divergence or flutter instabilities. Listing 13 illustrates the boundary condition for `W` field at the free (`right`) end, using the `followerForceBeamDisplacementNR` boundary condition. Listing 14 shows the Newton-Raphson solver settings defined in `constant/beamProperties` file.

```
// W boundary condition
right
{
        type followerForceBeamDisplacementNR;

    followerForceSeries
    {
        "fileName|file"  "$FOAM_CASE/constant/timeVsFollowerForce";
         outOfBounds clamp;
    }
}

// timeVsFollowerForce in constant/beamProperties
(
    ( 0 (0 0 0))
    ( 1 (0 0 1.34e5))
)
```

Listing 13. 2D cantilever subjected to follower force - follower force boundary condition for `W` field

The mechanical properties are taken from the literature, with flexural rigidity $EI = 3.5 \times 10^7$ N/m^2 and shear rigidity $GA = 1.61538 \times 10^8$ N/m^2. A circular cross-section with radius $R \approx 0.658$ m is selected to satisfy these values. This numerical example is a steady state case, and therefore, there is no physical time step involved. Instead, the external load is applied incrementally, called load increments or pseudo-time steps, and the static equilibrium position is solved for each increment. The total load of 134 kN is

applied in the $z-$direction in 1000 increments (pseudo $\Delta t = 0.001$ s), and the corresponding deformed configuration of the beam at different load steps are shown in Fig. 11. For the `Theta` field, the left end is clamped - `fixedValue uniform (0 0 0)`, and at the right end a time-varying zero `momentSeries` is provided with `momentBeamRotationNR` boundary condition (similar to Test case 6.1).

```
beamModel coupledTotalLagNewtonRaphsonBeam;
// Initialising beam
..
// Variables specific to solver
coupledTotalLagNewtonRaphsonBeamCoeffs
{
    // Newton-Raphson loop maximum allowed iterations
    nCorrectors 200;

    // Convergence tolerance for solution correction norm and residual norm
    solutionTol 1e-10;
    residualTol 1e-8;
}
```

Listing 14. 2D cantilever subjected to follower force - flags specific to Newton-Raphson iterations

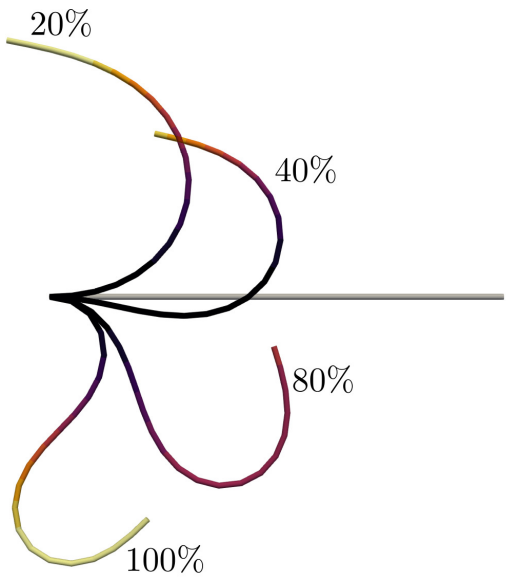

Figure 11. In-plane cantilever beam subjected to follower end load: different stages of deformation expressed as % of total load 134 kN

Figures 12 and 13 show the variation of the in-plane displacement components w_x and w_z at the free end of the beam under the applied load for discretisations of 5, 10, and 20 segments. Reference values are taken from Simo et al. [29]. The `beamFoam` results are in good agreement with the reference solution, and the predictions approach the benchmark as the mesh is refined. For the specified residual and solution tolerances, convergence is typically achieved in 4–6 Newton–Raphson iterations per time step.

A mesh convergence study is carried out to assess the spatial discretisation error of the cantilever tip displacement at the free end. The percentage relative error is evaluated at the final time step and is defined as,

$$\% \text{ relative error} = \left| \frac{\xi^{\text{num}} - \xi^{\text{ref}}}{\xi^{\text{ref}}} \right| \times 100\% \tag{18}$$

where ξ^{num} denotes the numerical value and ξ^{ref} is the reference result from Simo et al. [29].

Figure 14 shows the error in w_x and w_z for discretisations of 10, 20, 40, and 80 CVs. A second order rate of convergence is observed for this special case involving a non-conservative circulatory follower force.

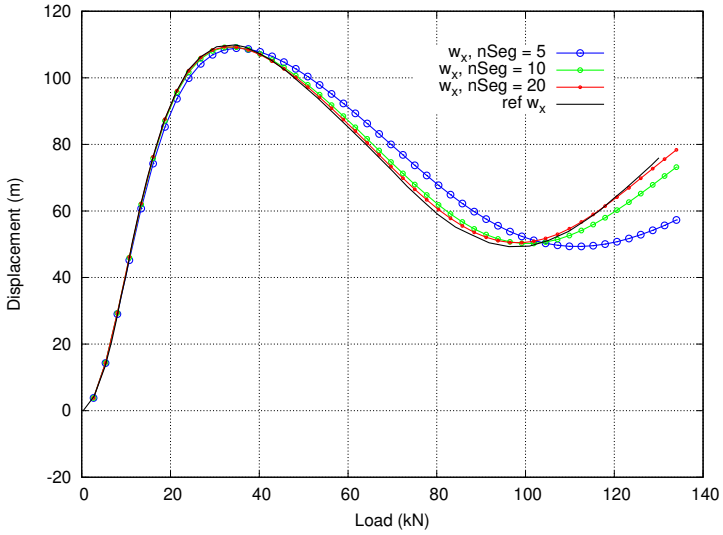

Figure 12. In-plane cantilever beam subjected to follower end load: in-plane displacement w_x versus the magnitude of the applied load

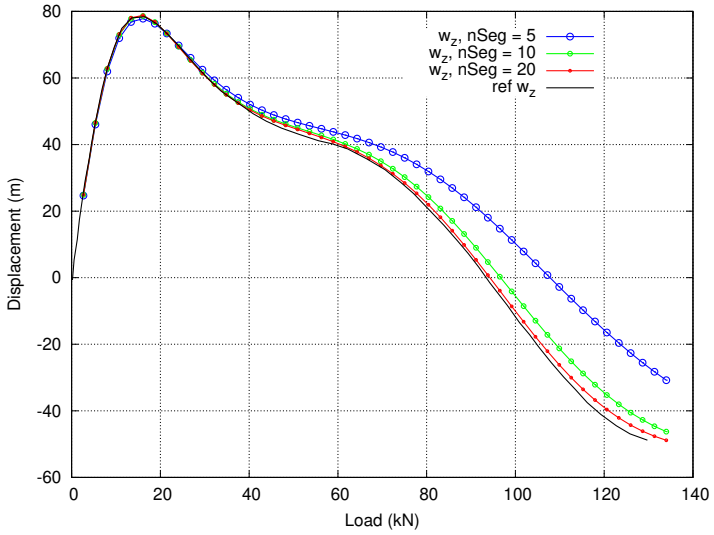

Figure 13. In-plane cantilever beam subjected to follower end load: in-plane displacement w_z versus the magnitude of the applied load

6.3. **Free flight motion of a flexible beam.** The final test case is a 3-D dynamic beam motion test case investigated by several authors [30,35,42]. The motion exhibited by this benchmark problem is commonly referred to as the *kayak-rowing* pattern. It is characterised by a strongly coupled 3-D response in which the free ends of the beam trace paddle-like trajectories due to the nonlinear interaction of bending in two planes with oscillatory forward motion. The test case is provided in the tutorials repository under the name `freeFlexibleBeam`.

An initially straight but inclined beam in the $xy-$plane of length $L = 10$ m is oriented in the undeformed configuration (Fig. 15). To set this configuration, the following command of `setInitialPositionBeam` is used.

```
setInitialPositionBeam -cellZone beam_0 -translate "(0 0 0)" \
    -rotateAngle "((0 0 -1) 53.130102)"
```

Listing 15. Free flight motion of a flexible beam - `setInitialPositionBeam` command

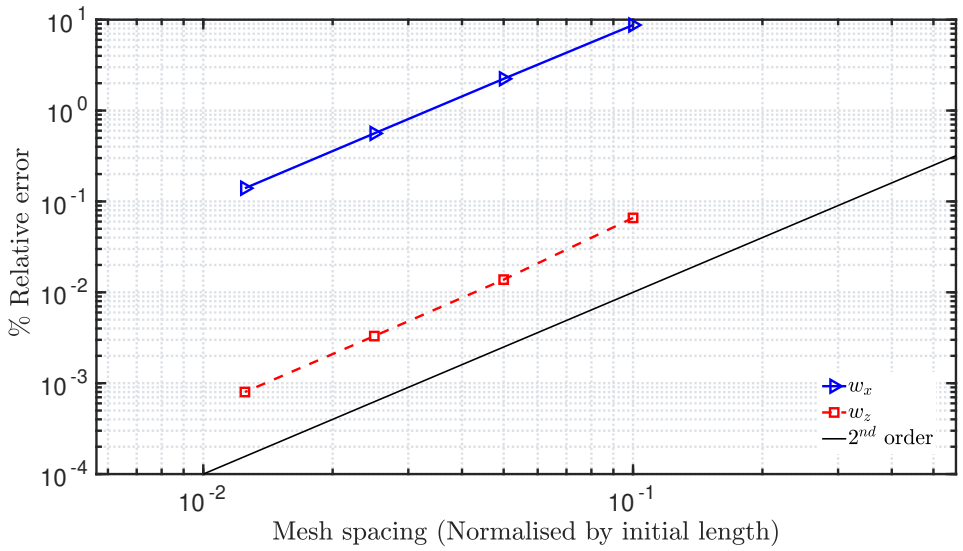

Figure 14. Cantilever beam subjected to follower end load: mesh convergence of the displacement components for the applied load of 134 kN at the right end

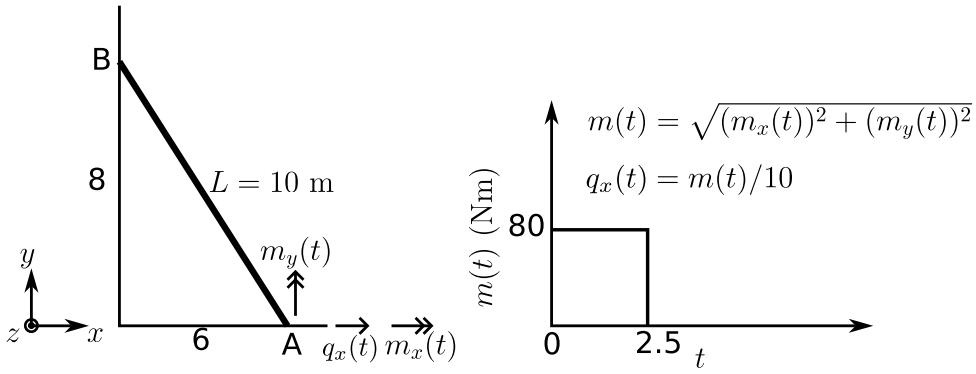

Figure 15. Free flight motion of a flexible beam: Schematic diagram of the beam in its undeformed configuration (left) and the load vs time graph (right)

The mechanical properties are adopted from Simo et al. [30]: $EA = GA = 1 \times 10^4$ N, $EI = 500$ Nm, $A\rho = 1$ kg/m, and $I_2\rho = I_3\rho = 10$ kgm. A circular beam cross-section is selected with radius $R = 0.447214$ m, and material density of $\rho = 1.59155$ kg/m^3. To match the prescribed rotary inertia $I_3\rho$, an artificial rotary inertia scaling factor $\tilde{k} = 200.39427$ is introduced, consistent with values used in the literature.

The two end patches of the beam (denoted as A and B in Fig. 15) are set as free edges. This implies that both ends of the beam are prescribed with Neumann boundary conditions. At point A (`right` patch), a combination of moments about the $x-$ and $y-$axes, $\boldsymbol{m}(t) \equiv (m_x(t), m_y(t), 0)$ such that the magnitude of the total moment $m(t) = 80$ Nm, and a force $\boldsymbol{q}(t) \equiv (q_x(t), 0, 0)$, where $q_x(t) = 8$ N are provided from $t = 0$ to $t = 2.5$ s of the motion. After that, the external loads are instantaneously removed, and the dynamic simulation is run in the free vibration state until $t = 40$ s. Listings 16 and 17 show the boundary conditions for `W` and `Theta` fields respectively. The corresponding time-varying force and moment series for specifying the Neumann boundary condition for `W` and `Theta` are given in Listings 18 and 19.

```
// W boundary condition
left // Point B
{
        type forceBeamDisplacementNR;
        forceSeries
    {
        "fileName|file"        "$FOAM_CASE/constant/timeVsForceLeft";
        outOfBounds     clamp;
    }

}
right // Point A
{
    type forceBeamDisplacementNR;

    forceSeries
    {
        "fileName|file"        "$FOAM_CASE/constant/timeVsForceRight";
        outOfBounds     clamp;
    }
}
```

Listing 16. Free flight motion of a flexible beam - boundary conditions for `W` field

```
// Theta boundary condition
left // Point B
{
        type momentBeamRotationNR;
        momentSeries
    {
        "fileName|file"        "$FOAM_CASE/constant/timeVsMomentLeft";
        outOfBounds     clamp;
    }

}

right // Point A
{
    type momentBeamRotationNR;

    momentSeries
    {
        "fileName|file"        "$FOAM_CASE/constant/timeVsMomentRight";
        outOfBounds     clamp;
    }
}
```

Listing 17. Free flight motion of a flexible beam - boundary conditions for `Theta` field

```
// timeVsForceLeft (Point B) in constant/beamProperties
(
    ( 0 ( 0 0 0 ) )
    ( 40 ( 0 0 0 ) )
)
// timeVsForceRight (Point A) in constant/beamProperties
(
    ( 0 ( 8 0 0 ) )
    ( 0.1 (8 0 0 ))
    ( 2.5 (8 0 0 ) )
    ( 2.51 (0 0 0 ) )
    ( 40 (0 0 0 ) )
)
```

Listing 18. Time varying force series in `constant/beamproperties`

```
// timeVsMomentLeft (Point B) in constant/beamProperties
(
    ( 0 ( 0 0 0 ) )
    ( 40 ( 0 0 0 ) )
)

// timeVsMomentRight (Point A) in constant/beamProperties
(
    ( 0 ( 56.568542495 56.568542495 0 ) )
    ( 0.1 ( 56.568542495 56.568542495 0 ))
    ( 2.5 ( 56.568542495 56.568542495 0 ) )
    ( 2.51 (0 0 0) )
    ( 40 (0 0 0 ) )
)
```

Listing 19. Time varying moment series in `constant/beamproperties`

A mesh discretisation of 10 beam CVs and a uniform time step $\Delta t = 0.01$ s along with Newmark-β time scheme are chosen for the motion. The deformed position of the flexible beam is presented in Fig. 16.

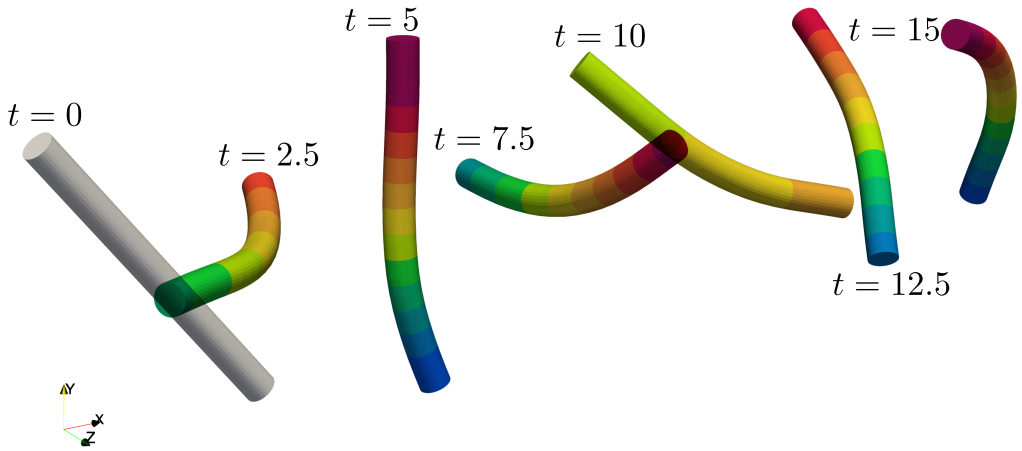

Figure 16. Free flight motion of a flexible beam: Deformed configurations of the beam in the perspective view from time $t = 0$ to $t = 15$ s

The displacement response versus time of the end point A of the beam is shown in Fig. 17. A negligible difference in the deformation pattern is observed when the beam mesh is refined. Therefore, the displacement graph for only 10 CVs is reported here. It is evident from the displacement graph that with the evolution of time, the beam moves forward along the global $x-$direction, and the endpoint A oscillates in the $y-$ and $z-$ axes, thus creating a kayak-rowing pattern as it evolves in space. The residuals converge in an average of 3 iterations per time step.

To verify the accuracy of the dynamic motion of the beam, the energy plots are investigated. Since the applied loads are removed after $t = 2.5$ s and the beam enters a free vibration state, the energy is expected to be conserved after 2.5 s without any numerical damping. The energy data can be extracted for post-processing using the `beamEnergyData` functionObject defined in the `controlDict` as follows,

```
functions
{
        beamEnergyData
        {
                type    beamEnergyData;
        }
}
```

Listing 20. Free flight motion of a flexible beam - energy extraction `functionObject` in `system/controlDict`

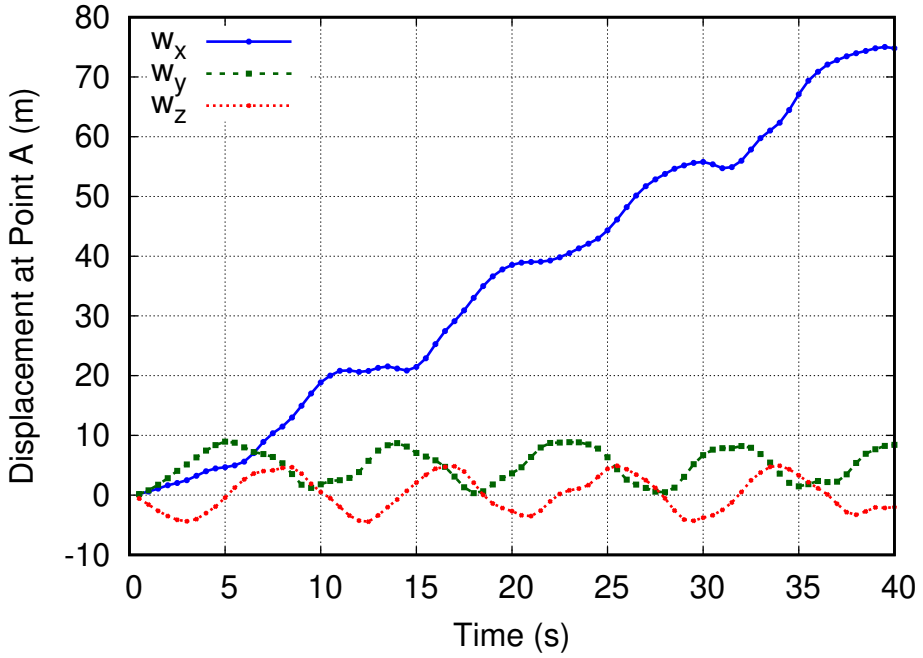

Figure 17. Free flight motion of a flexible beam: Displacement versus time graph at the end point A for 10 CVs and $\Delta t = 0.01$ s using Newmark-β time scheme

A comparison of the variation in the internal (E_{int}), kinetic (E_{kin}) and total energy (E_{tot}) plots using the implicit 1st order backward Euler scheme and the implicit 2nd order Newmark-beta time integration method adopted here is presented in Fig. 18. As expected, for a time step $\Delta t = 0.01$ s, the implicit 1st order Euler method results in numerical damping of the system resulting in the decay of energy, whereas using the implicit 2nd order Newmark-beta time-integration scheme, the energy is conserved for both $\Delta t = 0.1$ s and $\Delta t = 0.01$ s for a coarse mesh discretisation of 10 CVs. A much smaller time step is required for the Euler integration scheme to ensure that numerical damping is small.

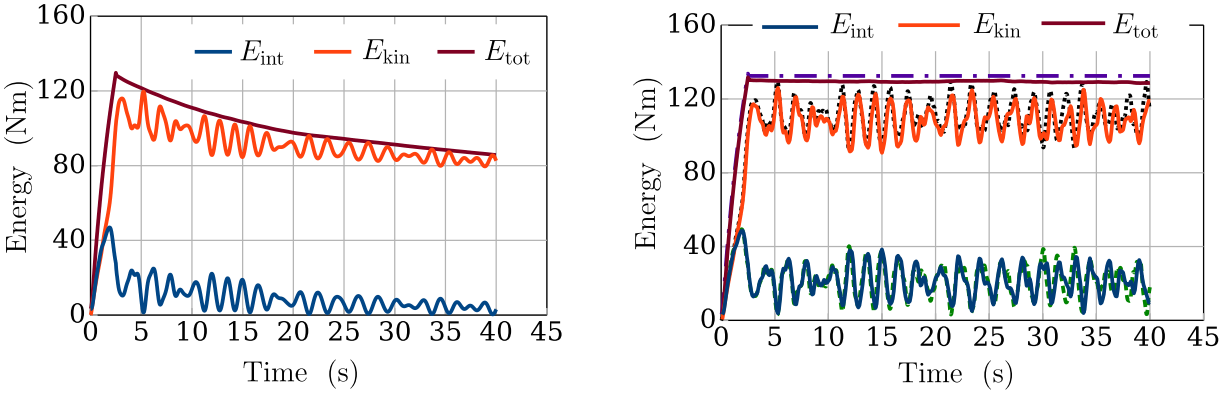

Figure 18. Free flight motion of a flexible beam: Energy plots of the beam (a) Implicit Euler (10 CVs, $\Delta t = 0.01$ s) (b) Newmark-beta integration 10 CVs, $\Delta t = 0.1$ s (solid line), $\Delta t = 0.01$ s (dotted lines)

To observe the energy stability of Newmark-β during long-duration simulation, the same test case is run for a total time of $t = 1000$ s, and the variation of total energy for $\Delta t = 0.1$ s and $\Delta t = 0.01$ s is presented in Fig. 19. For $\Delta t = 0.1$ s, the energy is observed to be stable with slight discrepancies, whereas, for $\Delta t = 0.01$ s, the energy is stable and constant with negligible oscillations. Overall, the energy is observed to be stable and non-dissipative for this long-duration simulation.

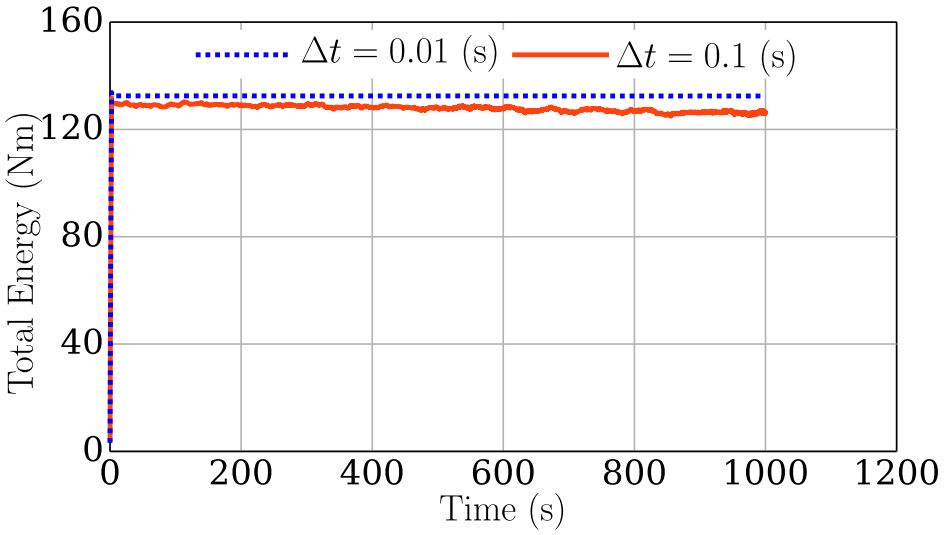

Figure 19. Free flight motion of a flexible beam: The total energy plot for a long duration simulation of $t = 1000$ s for $\Delta t = 0.1$ s and $\Delta t = 0.01$ s using Newmark-β time-integration scheme

7. Conclusions and Outlook

In this work, a new cell-centred finite volume beam solver, `beamFoam`, has been developed within the OpenFOAM framework for simulating slender structures undergoing large displacements and rotations. The solver is based on the nonlinear, geometrically exact Simo–Reissner beam theory and can capture axial, bending, shear, and torsional responses. Key implementation aspects — including spatial and temporal discretisation, boundary condition treatment, and the Newton–Raphson iterative procedure — have been detailed to support reproducibility and future development.

The solver performance was demonstrated through three benchmark problems: (i) large 3D deformation of a cantilever beam, (ii) quasi-static 2D motion of a cantilever under circulatory follower loading, and (iii) long-term free vibration of a 3D flexible beam exhibiting the well-known *kayak-rowing* pattern. In all cases, `beamFoam` showed excellent agreement with reference results from the literature, while maintaining high accuracy even on coarse meshes and achieving a significantly reduced computational cost. For instance, in the 3D cantilever test case (Tab. 6, Case 6.1), execution time was reduced from 12,156 s in `solid4Foam` to just 1.26 s with `beamFoam`. The implemented second-order Newmark-β scheme was found to be stable and accurate for dynamic problems (Fig. 18 of Test Case 6.3), while mesh convergence studies confirmed second-order spatial accuracy (Fig. 14, Test Case 6.2).

Overall, this work establishes `beamFoam` as a robust and efficient beam solver within the OpenFOAM ecosystem. Its demonstrated accuracy on coarse meshes and computational efficiency make it a strong candidate for future extensions, including multi-beam contact modelling [32] and coupling with fluid solvers for fully resolved fluid–structure interaction [33, 43] of slender structures.

Acknowledgements

The authors would like to acknowledge Research Ireland, NexSys project 21/SPP/3756 for funding. Vikram Pakrashi would like to acknowledge SEAI RDD/966 FlowDyn and Research Ireland MaREI RC2302-2 projects. Philip Cardiff gratefully acknowledges financial support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No. 101088740), and from I-Form, funded by Research Ireland (formerly SFI) Grant Number 21/RC/10295 P2, co-funded under European Regional Development Fund and by I-Form industry partners. Additionally, the authors wish to acknowledge the DJEI/DES/SFI/HEA Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support (`www.ichec.ie`), and part of this work has been carried out using the UCD ResearchIT Sonic cluster which was funded by UCD IT Services and the UCD Research Office.

Author Contributions: Conceptualisation, S.B., A.T., Z.T., and P.C.; methodology, S.B., Z.T. and P.C.; software, S.B., A.T., Z.T., and P.C.; validation, S.B.; formal analysis, S.B.; investigation, S.B. and A.T.; resources, V.P. and P.C.; data curation, S.B., A.T., and P.C; writing—original draft preparation, S.B. and A.T.; writing—review and editing, V.P. and P.C.; visualisation, S.B. and A.T.; supervision, P.C., Z.T., and V.P.; project administration, P.C. and V.P.; funding acquisition, V.P. and P.C. All authors have read and agreed to the published version of the manuscript.

Appendix A. Linearisation of balance equations

The description of linearising the integral governing beam equations (Eqns. 7 and 8) is described here. First, the integral equations are converted into discretised algebraic equations using the spatial cell-centred finite volume method (Eqn. 14) and temporal discretisation of the inertia terms (Eqn. 15). The final discretised form for a computational node is given in Eqn. 16. After the spatial discretisation of integral equations (Eqn. 14), the spatial forces and moments ($\boldsymbol{q}_f$ and $\boldsymbol{m}_f$) have to be evaluated at face centres of the cell. Therefore, the nonlinear forces and moments (Eqns. 3a and 3b) have to be linearised about a current equilibrium approximation to construct the Jacobian matrix $\overset{*}{\boldsymbol{J}}$ in Eqn. 11 and form the residual $\boldsymbol{R}_{n+1}(\boldsymbol{w},\boldsymbol{\theta})$ (Eqn. 10). This linearisation enables the iterative refinement of internal forces and moments until convergence is achieved.

To apply the Newton–Raphson scheme, the internal forces $\boldsymbol{q}$ and moments $\boldsymbol{m}$ are linearised about their explicit values computed from the strain and curvature fields obtained in the previous iteration (Eqns. 4a and 5). The linearised forms of $(k+1)$-th Newton iteration at time t_{n+1} are given by,

$$\boldsymbol{q}_{n+1}^{(k+1)} = \overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_q \overset{*}{\boldsymbol{\gamma}} - \widehat{\overset{*}{\boldsymbol{q}}}\ \Delta\boldsymbol{\theta} + \left(\overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_q (\overset{*}{\boldsymbol{\Lambda}}_t)^\top\right)\left[\Delta\boldsymbol{w}' + \widehat{(\overset{*}{\boldsymbol{r}})'}(\Delta\boldsymbol{\theta})\right] \tag{19}$$

$$\boldsymbol{m}_{n+1}^{(k+1)} = \overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_m \overset{*}{\boldsymbol{\kappa}} - \widehat{\overset{*}{\boldsymbol{m}}}\ \Delta\boldsymbol{\theta} + (\overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_m (\overset{*}{\boldsymbol{\Lambda}}_t)^\top)\ \Delta\boldsymbol{\theta}' \tag{20}$$

In these expressions, the superscript $\overset{*}{(\cdot)} \equiv (\cdot)_{n+1}^{(k)}$ denotes known quantities from the previous iteration k. The terms $\overset{*}{\boldsymbol{q}}$ and $\overset{*}{\boldsymbol{m}}$ denote values of internal forces and moments at the k-th iteration evaluated using the *linearised* equations (Eqns. 19 and 20). The correction vectors $\Delta\boldsymbol{w}$ and $\Delta\boldsymbol{\theta}$ represent the incremental corrections in displacement and rotation vectors, respectively. As the Newton–Raphson iterations progress, these increment vectors approach zero, and consequently, the linearised internal forces and moments converge to the exact nonlinear values defined in Eqns. 3a–3b. The $\widehat{(\cdot)}$ operator in Eqns. 19 and 20 denotes a skew-symmetric matrix constructed from the corresponding rotation vector in $\mathbb{R}^3$ as presented in Eqn. 2.

Additionally, the nonlinear coupling term $(\boldsymbol{r}' \times \boldsymbol{q})$ in the angular momentum equation (Eqn. 8) is also linearised as,

$$(\boldsymbol{r}' \times \boldsymbol{q})_{n+1}^{(k+1)} = \widehat{(\overset{*}{\boldsymbol{r}})'}\overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_q \overset{*}{\boldsymbol{\gamma}} + \left[\widehat{(\overset{*}{\boldsymbol{r}})'}\left(\overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_q (\overset{*}{\boldsymbol{\Lambda}}_t)^\top\right) - \widehat{\overset{*}{\boldsymbol{q}}}\right]\Delta\boldsymbol{w}' + \widehat{(\overset{*}{\boldsymbol{r}})'}\left[\overset{*}{\boldsymbol{\Lambda}}_t \boldsymbol{C}_q (\overset{*}{\boldsymbol{\Lambda}}_t)^\top \widehat{(\overset{*}{\boldsymbol{r}})'} - \widehat{\overset{*}{\boldsymbol{q}}}\right]\Delta\boldsymbol{\theta} \tag{21}$$

The inertial force, $\boldsymbol{q}_\rho \equiv \rho A\ \ddot{\boldsymbol{r}}$ and inertial moment $\boldsymbol{m}_\rho \equiv \left[\boldsymbol{\Lambda}_t \bar{I}_\rho \boldsymbol{\alpha} + \boldsymbol{\Lambda}_t \widehat{\boldsymbol{\omega}}\ (\bar{I}_\rho\ \boldsymbol{\omega})\right]$ terms in right-hand side of the balance equations (Eqns. 7 and 8), also have to be linearised about the values calculated in the previous iteration to apply the iterative Newton-Raphson scheme. For a time step t_{n+1} and $(k+1)-$th iteration, the linearised form of the inertial forces $\boldsymbol{q}_\rho$ and inertial moments $\boldsymbol{m}_\rho$ is given by,

$$\mathrm{L}[\boldsymbol{q}_\rho] = \ \overset{*}{\boldsymbol{q}}_\rho + \rho A \Delta\ddot{\boldsymbol{r}} \tag{22a}$$

$$\begin{aligned}\mathrm{L}[\boldsymbol{m}_\rho] = \ & \overset{*}{\boldsymbol{m}}_\rho + \Delta\boldsymbol{\Lambda}\left(\bar{I}_\rho \overset{*}{\boldsymbol{\alpha}}\ + \widehat{\overset{*}{\boldsymbol{\omega}}}(\bar{I}_\rho \overset{*}{\boldsymbol{\omega}})\right) \\ & + \overset{*}{\boldsymbol{\Lambda}} \bar{I}_\rho \Delta\boldsymbol{\alpha} + \overset{*}{\boldsymbol{\Lambda}} \Delta\widehat{\boldsymbol{\omega}}(\bar{I}_\rho \overset{*}{\boldsymbol{\omega}}) + \overset{*}{\boldsymbol{\Lambda}}\overset{*}{\widehat{\boldsymbol{\omega}}}(\bar{I}_\rho \Delta\boldsymbol{\omega})\end{aligned} \tag{22b}$$

where $\overset{*}{(\cdot)} \equiv (\cdot)_{n+1}^{(k)}$ represents the explicit values of the quantities evaluated in the previous $k-$th iteration of $(n+1)$th time step. The implicit correction terms $(\Delta(\cdot))$ of the linear and angular accelerations and the angular velocities have to be expressed in terms of the primary unknowns, i.e., the incremental displacement $(\Delta\boldsymbol{w})$ and incremental rotation $(\Delta\boldsymbol{\theta})$ vectors using appropriate time integration scheme.

The linearised expressions (Eqns. 19-22a) are substituted in the discretised form of integral equations (Eqns. 14 and 15). The coefficients of $\Delta\boldsymbol{w}$ and $\Delta\boldsymbol{\theta}$ are then populated to form the 6×6 Jacobian matrix (Eqn. 13).

Appendix B. Total energy in beams

In a conservative dynamic system without damping, the sum of kinetic and potential energies is invariant once external loads are removed. During load application, the total energy increases due to the work done by the external forces; after the load is removed, the system vibrates freely and the total energy remains constant.

For Simo–Reissner beams, the total energy is the sum of internal and kinetic contributions. The internal (strain) energy (E_{int}) is obtained by integrating the hyper-elastic, length-specific energy along the beam,

$$E_{\text{int}} = \int_L \tilde{\Pi}_{\text{int}} \, \mathrm{d}L = \int_L \left(\frac{1}{2} \boldsymbol{\gamma}^\top \boldsymbol{C}_q \boldsymbol{\gamma} + \frac{1}{2} \boldsymbol{\kappa}^\top \boldsymbol{C}_m \boldsymbol{\kappa} \right) \mathrm{d}L \tag{23a}$$

$$\approx \sum_f \left(\frac{1}{2} \boldsymbol{\gamma}_f^\top \boldsymbol{C}_q \boldsymbol{\gamma}_f + \frac{1}{2} \boldsymbol{\kappa}_f^\top \boldsymbol{C}_m \boldsymbol{\kappa}_f \right) L_f \tag{23b}$$

where the $(\cdot)_f$ denotes the internal strains ($\boldsymbol{\gamma}$) and curvatures ($\boldsymbol{\kappa}$) at face centres, and L_f is the distance between two adjacent cell-centres surrounding the face f. For a uniform mesh, $L_f \equiv \mathrm{L_p}$ and the value of L_f at the boundaries is $L_f = 0.5 \ \mathrm{L_p}$.

The kinetic energy (E_{kin}) is computed from the length-specific kinetic energy over the beam length, which is related to the linear velocity $\boldsymbol{v} \equiv \dot{\boldsymbol{r}}$ and (material) angular velocity $\boldsymbol{\omega}$ as,

$$E_{\text{kin}} = \int_L \tilde{\Pi}_{\text{kin}} \mathrm{d}L = \int_L \left(\frac{1}{2} \rho \boldsymbol{v}^\top A \boldsymbol{v} + \frac{1}{2} \boldsymbol{\omega}^\top \bar{I}_\rho \boldsymbol{\omega} \right) \, \mathrm{d}L \tag{24a}$$

$$\approx \sum_{\tilde{p}} \left(\frac{1}{2} \rho \boldsymbol{v}_{\tilde{p}}^\top A \boldsymbol{v}_{\tilde{p}} + \frac{1}{2} \boldsymbol{\omega}_{\tilde{p}}^\top \bar{I}_\rho \boldsymbol{\omega}_{\tilde{p}} \right) L_{\tilde{p}} \tag{24b}$$

where $(\cdot)_{\tilde{p}}$ denotes the cell-centre values, and $L_{\tilde{p}}$ is the length of beam cell ($L_{\tilde{p}} \equiv \mathrm{L_p}$ for a uniform mesh). Spatial and temporal discretisation introduce errors in E_{int} and E_{kin}, which decrease with mesh and time-step refinement. The total energy of the beam is given by $E_{\text{tot}} = E_{\text{int}} + E_{\text{kin}}$.

References

[1] I. Demirdžić and D. Martinović, "Finite volume method for thermo-elasto-plastic stress analysis," *Computer methods in applied mechanics and engineering*, vol. 109, no. 3-4, pp. 331–349, 1993.

[2] P. Cardiff, A. Karač, P. De Jaeger, H. Jasak, J. Nagy, A. Ivanković, and Ž. Tuković, "An open-source finite volume toolbox for solid mechanics and fluid-solid interaction simulations," *arXiv preprint arXiv:1808.10736*, 2018.

[3] P. Cardiff and I. Demirdžić, "Thirty years of the finite volume method for solid mechanics," *Archives of Computational Methods in Engineering*, pp. 1–60, 2021.

[4] P. Cardiff, A. Karač, and A. Ivanković, "A large strain finite volume method for orthotropic bodies with general material orientations," *Computer Methods in Applied Mechanics and Engineering*, vol. 268, pp. 318–335, 2014.

[5] P. Cardiff, Ž. Tuković, H. Jasak, and A. Ivanković, "A block-coupled finite volume methodology for linear elasticity and unstructured meshes," *Computers & structures*, vol. 175, pp. 100–122, 2016.

[6] Ž. Tuković, A. Karač, P. Cardiff, H. Jasak, and A. Ivanković, "Openfoam finite volume solver for fluid-solid interaction," *Transactions of FAMENA*, vol. 42, no. 3, pp. 1–31, 2018.

[7] H. Chen and M. Hall, "Cfd simulation of floating body motion with mooring dynamics: Coupling moordyn with openfoam," *Applied Ocean Research*, vol. 124, p. 103210, 2022.

[8] J. Palm, C. Eskilsson, G. M. Paredes, and L. Bergdahl, "Coupled mooring analysis for floating wave energy converters using cfd: Formulation and validation," *International Journal of Marine Energy*, vol. 16, pp. 83–99, 2016.

[9] E. Marino, M. Gkantou, A. Malekjafarian, S. Bali, C. Baniotopoulos, J. van Beeck, R. P. Borg, N. Bruschi, P. Cardiff, E. Chatzi *et al.*, "Offshore renewable energies: A review towards floating modular energy islands—monitoring, loads, modelling and control," *Ocean engineering*, vol. 313, p. 119251, 2024.

[10] A. G. Neto, C. A. Martins, and P. M. Pimenta, "Static analysis of offshore risers with a geometrically-exact 3d beam model subjected to unilateral contact," *Computational Mechanics*, vol. 53, no. 1, pp. 125–145, 2014.

[11] S. Čanić and J. Tambača, "Cardiovascular stents as pde nets: 1d vs. 3d," *Ima journal of applied mathematics*, vol. 77, no. 6, pp. 748–770, 2012.

[12] P. Zunino, J. Tambača, E. Cutrì, S. Čanić, L. Formaggia, and F. Migliavacca, "Integrated stent models based on dimension reduction: review and future perspectives," *Annals of biomedical engineering*, vol. 44, no. 2, pp. 604–617, 2016.

[13] N. G. Jacobsen, W. Bakker, W. S. Uijttewaal, and R. Uittenbogaard, "Experimental investigation of the wave-induced motion of and force distribution along a flexible stem," *Journal of Fluid Mechanics*, vol. 880, pp. 1036–1069, 2019.
[14] M. Maza, J. L. Lara, and I. J. Losada, "Tsunami wave interaction with mangrove forests: A 3-d numerical approach," *Coastal Engineering*, vol. 98, pp. 33–54, 2015.
[15] M. Wang, E. Avital, T. Korakianitis, J. Williams, and K. Ai, "A numerical study on the influence of curvature ratio and vegetation density on a partially vegetated u-bend channel flow," *Advances in Water Resources*, vol. 148, p. 103843, 2021.
[16] H. B. Gilbert, "On the mathematical modeling of slender biomedical continuum robots," *Frontiers in Robotics and AI*, vol. 8, p. 732643, 2021.
[17] C. Armanini, F. Boyer, A. T. Mathew, C. Duriez, and F. Renda, "Soft robots modeling: A structured overview," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1728–1748, 2023.
[18] S. Timoshenko, "On the correction for shear of the differential equation for transverse vibrations of prismatic bars." *Phil Mag Ser*, vol. 41, pp. 744–764, 1921.
[19] N. Fallah and M. Ebrahimnejad, "Finite volume analysis of adaptive beams with piezoelectric sensors and actuators," *Applied Mathematical Modelling*, vol. 38, no. 2, pp. 722–737, 2014.
[20] N. Fallah and A. Ghanbari, "A displacement finite volume formulation for the static and dynamic analysis of shear deformable circular curved beams," *Scientia Iranica*, vol. 25, no. 3, pp. 999–1014, 2018.
[21] N. Fallah and F. Hatami, "A displacement formulation based on finite volume method for analysis of timoshenko beam," in *Proceedings of the 7th international conference on civil engineering, Tehran, Iran*, 2006.
[22] N. Fallah and F. Hatami, "Extension of the finite volume method for instability analysis of columns with shear effects," in *Proceedings of the eighth international conference on computational structures technology, Stirlingshire, Scotland. Civil-Comp Press. Paper*, vol. 192, 2006.
[23] S. Isić, V. Doleček, and I. Karabegović, "Numerical and experimental analysis of postbuckling behaviour of prismatic beam under displacement dependent loading," in *Proceedings of First Serbian Congress on Theoretical and Applied Mechanics.-Kopaonik, Serbia*, 2007, pp. 331–338.
[24] S. Isić, V. Doleček, and I. Karabegović, "A comparison between finite element and finite volume methods on the problem of stability of timoshenko beam," in *The 12th international conference on problems of material engineering, Jasna, Slovakia*, 2007.
[25] N. Fallah, "Finite volume method for determining the natural characteristics of structures," *Journal of Engineering Science and Technology*, vol. 8, no. 1, pp. 93–106, 2013.
[26] L.-L. Jing, P.-J. Ming, W.-P. Zhang, L.-R. Fu, and Y.-P. Cao, "Static and free vibration analysis of functionally graded beams by combination timoshenko theory and finite volume method," *Composite structures*, vol. 138, pp. 192–213, 2016.
[27] A. Love, *A treatise on the mathematical theory of elasticity*. University press, 1927.
[28] J. Simo, "A finite strain beam formulation. the three-dimensional dynamic problem. part i," *Computer Methods in Applied Mechanics and Engineering*, vol. 49, no. 1, pp. 55–70, 1985.
[29] J. Simo and L. Vu-Quoc, "A three-dimensional finite-strain rod model. part ii: Computational aspects," *Computer methods in applied mechanics and engineering*, vol. 58, no. 1, pp. 79–116, 1986.
[30] J. Simo and L. Vu-Quoc, "On the dynamics in space of rods undergoing large motions—a geometrically exact approach," *Computer methods in applied mechanics and engineering*, vol. 66, no. 2, pp. 125–161, 1988.
[31] S. Bali, Ž. Tuković, P. Cardiff, A. Ivanković, and V. Pakrashi, "A cell-centered finite volume formulation of geometrically exact simo–reissner beams with arbitrary initial curvatures," *International journal for numerical methods in engineering*, vol. 123, no. 17, pp. 3950–3973, 2022.
[32] S. Bali, Ž. Tuković, P. Cardiff, A. Ivanković, and V. Pakrashi, "A finite volume adaptation of beam-to-beam contact interactions implemented for geometrically exact simo–reissner beams," *Computational mechanics*, vol. 75, no. 1, pp. 237–263, 2025.
[33] A. Taran, S. Bali, Željko Tuković, V. Pakrashi, and P. Cardiff, "A finite volume simo–reissner beam method for moored floating body dynamics," *Applied Ocean Research*, vol. 165, p. 104845, 2025.
[34] P. Cardiff, I. Batisti *et al.*, "solids4foam: A toolbox for performing solid mechanics and fluid-solid interaction simulations in openfoam," *Journal of Open Source Software*, vol. 10, no. 108, p. 7407, 2025.
[35] G. Jelenić and M. Crisfield, "Interpolation of rotational variables in nonlinear dynamics of 3d beams," *International Journal for Numerical Methods in Engineering*, vol. 43, no. 7, pp. 1193–1222, 1998.
[36] A. Ibrahimbegovic, "On the choice of finite rotation parameters," *Computer Methods in Applied Mechanics and Engineering*, vol. 149, no. 1-4, pp. 49–71, 1997.
[37] G. Jelenić and M. Crisfield, "Geometrically exact 3d beam theory: implementation of a strain-invariant finite element for statics and dynamics," *Computer methods in applied mechanics and engineering*, vol. 171, no. 1-2, pp. 141–171, 1999.
[38] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.
[39] K. Inc. and Paraview., "Paraview user's guide." in *URL http://www.paraview.org*, 2023.
[40] P. Cardiff, D. Armfield, Ž. Tuković, and I. Batistić, "A jacobian-free newton-krylov method for cell-centred finite volume solid mechanics," *arXiv preprint arXiv:2502.17217*, 2025.
[41] J. Argyris and S. Symeonidis, "Nonlinear finite element analysis of elastic systems under nonconservative loading-natural formulation. part i. quasistatic problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 26, no. 1, pp. 75–123, 1981.
[42] F. Boyer and D. Primault, "Finite element of slender beams in finite transformations: a geometrically exact approach," *International Journal for Numerical Methods in Engineering*, vol. 59, no. 5, pp. 669–702, 2004.
[43] C. McAlister, A. Taran, M. Campell, S. Bali, M. Karimirad, V. Pakrashi, and P. Cardiff, "Validation of a finite volume simo-reissner beam method for moored floating solar platforms," in *The 20th OpenFOAM Workshop (OFW20)*, 2025.