A two-level nonlinear beam model using adjoints

Marco Morandini*

Politecnico di Milano
Dipartimento di Scienze e Tecnologie Aerospaziali

Abstract

A two-level nonlinear beam model, with the two levels linked by first- and second-order adjoints, allows to successfully deal with beam problems characterized by complex and multi-material cross-sections. Emphasis is given to the approach used for coupling the two levels within the FEniCS framework, and on how to overcome a few issues encountered during the actual implementation of the coupling procedure.

Kewywords: two-level, beam, Dolfin, adjoint

1 Introduction

Beam models are an extremely useful computational approach that allows to drastically reduce the computational complexity required for the structural analysis of slender solids, even if the overall structural response is nonlinear. Most of the time the cross-section material response is linear elastic, and this allows to characterize the beam cross-section by computing the so-called cross-section stiffness matrix, a linear relationship between resultant and moment resultant of the the normal stress vector and the beam generalized strain measures. Even when the cross-section undergoes plastic deformations it is

*email: marco.morandini@polimi.it
Proceedings of the FEniCS 2021conference -- 22-26 March 2021

often possible to simplify the cross-section analysis by assuming a kinematic model of the cross-section displacement. This may not be the case, however, for completely nonlinear deformation of complex cross sections, even more so if these cross-sections are not made with a homogeneous, isotropic material. In this case it may be necessary to resort to completely three dimensional analysis. Some approaches, based on simple kinematic assumptions for the cross section motion and/or on the assumption of an axial stress state, allow to deal with complex cross-section; among them, the works of Rigobello et al. (2013), Rezaiee-Pajand and Gharaei-Moghaddam (2015) and Chiorean (2017) are worth mentioning. Bilotta and Garcea (2019) proposed an interesting two-level beam analysis approach, but limited to small cross-section strains.

Recently a new coupling scheme between a nonlinear beam model and the completely nonlinear cross-section response analysis procedure of Morandini (2019) was proposed in Morandini (2020). The coupling is based on a mixed variational beam formulation, and depends, in order to link the beam and the cross-section model, on the first- and second-order adjoints of the cross-section complementary strain energy. This paper is based on Morandini (2020), and is aimed at presenting the actual implementation of the two-level analysis procedure, together with its shortcomings, within the Dolfin library (Logg et al., 2012) and the FEniCS framework (Alnæs et al., 2015).

The outline of the paper is as follows. Section 2 details the proposed formulation. Section 3 details Section 4 is dedicated to two selected examples taken from Morandini (2020). The overall run time requirements of the proposed two-level method are very briefly discussed within Section 5. The conclusions of Section 6 close the paper.

2 Formulation

This section briefly details the formulation of the beam model (Section 2.1) and of the local cross-section model (Section 2.2), as well as of the coupling strategy (Sections 2.3 and 2.4). The presentation follows Morandini (2020), and is required in order to understand the implementation description of Section 3.

2.1 Beam model

This section briefly introduces the high-level beam model. The interested reader can check e.g. Pietraszkiewicz and Eremeyev (2009); Cardona and Geradin (1988); Merlini and Morandini (2013) for details about intrinsic beam model formulations, and (Morandini, 2017) for a Dolfin implementation of simple intrinsic beam models. The deformed configuration of a beam is defined by the position $\boldsymbol{x}'(s)$ and by the orientation tensor $\boldsymbol{\alpha}'(s)$ at a point. Tensor $\boldsymbol{\alpha}'$ is an orthogonal unit tensor, and s the arc length in the reference configuration. The corresponding position vector and orientation tensor in the reference configuration are $\boldsymbol{x}$ and $\boldsymbol{\alpha}$, respectively. The beam transmits, ant any point, an internal force $\boldsymbol{T}$ and an internal moment $\boldsymbol{M}$, defined as the resultant and moment resultant of the beam cross-section normal stress vector. The linear and angular strain measures $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\beta}}$, defined as

$$\begin{aligned}\hat{\boldsymbol{\epsilon}} &= \boldsymbol{\alpha}'^T \boldsymbol{x}'_{,s} - \boldsymbol{\alpha}^T \boldsymbol{x}_{,s} \\ \hat{\boldsymbol{\beta}} &= \boldsymbol{\alpha}'^T \mathrm{ax}(\boldsymbol{\alpha}'^T \boldsymbol{\alpha}'_{,s}) - \boldsymbol{\alpha}^T \mathrm{ax}(\boldsymbol{\alpha}^T \boldsymbol{\alpha}_{,s}),\end{aligned} \tag{1}$$

are work-conjugated the back-rotated resultant and moment resultant vectors

$$\begin{aligned}\hat{\boldsymbol{T}} &= \boldsymbol{\alpha}'^T \boldsymbol{T} \\ \hat{\boldsymbol{M}} &= \boldsymbol{\alpha}'^T \boldsymbol{M}.\end{aligned} \tag{2}$$

The one-field principle of virtual work reads

$$\int_l \left(\delta\hat{\boldsymbol{\epsilon}} \hat{\boldsymbol{T}} + \delta\hat{\boldsymbol{\beta}} \hat{\boldsymbol{M}} \right) \mathrm{d}s - \delta L_e = 0, \tag{3}$$

with δL_e the virtual work of the external loads. The force and moment vectors $\hat{\boldsymbol{T}}$ and $\hat{\boldsymbol{M}}$ can be assumed to be function of the generalized strains $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\beta}}$. The first variation of the strain energy per unit of length $w(\hat{\boldsymbol{\epsilon}}, \hat{\boldsymbol{\beta}})$ is $\delta w = \hat{\boldsymbol{T}} \delta\hat{\boldsymbol{\epsilon}} + \hat{\boldsymbol{M}} \delta\hat{\boldsymbol{\beta}}$, with $\hat{\boldsymbol{T}} = w_{,\hat{\boldsymbol{\epsilon}}}$ and $\hat{\boldsymbol{M}} = w_{,\hat{\boldsymbol{\beta}}}$. Its Legendre transform defines the complementary strain energy v, function of $\hat{\boldsymbol{T}}$ and $\hat{\boldsymbol{M}}$,

$$v(\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}) = \hat{\boldsymbol{\epsilon}} \hat{\boldsymbol{T}} + \hat{\boldsymbol{\beta}} \hat{\boldsymbol{M}} - w \tag{4}$$

so that $\delta v = \hat{\boldsymbol{\epsilon}} \delta\hat{\boldsymbol{T}} + \hat{\boldsymbol{\beta}} \delta\hat{\boldsymbol{M}}$, $\hat{\boldsymbol{\epsilon}} = v_{,\hat{\boldsymbol{T}}}$ and $\hat{\boldsymbol{\beta}} = v_{,\hat{\boldsymbol{M}}}$. The Hellinger-Reissner two-field variational principle is

$$\begin{aligned}&\mathcal{H}(\delta\hat{\boldsymbol{\epsilon}}, \delta\hat{\boldsymbol{\beta}}, \delta\hat{\boldsymbol{T}}, \delta\hat{\boldsymbol{M}}, \hat{\epsilon}, \hat{\boldsymbol{\beta}}, \hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}) = \\ &\int_l \left(\delta\hat{\boldsymbol{\epsilon}} \hat{\boldsymbol{T}} + \delta\hat{\boldsymbol{\beta}} \hat{\boldsymbol{M}} + \delta\hat{\boldsymbol{T}} \hat{\epsilon} + \delta\hat{\boldsymbol{M}} \hat{\boldsymbol{\beta}} - \delta v(\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}) \right) \mathrm{d}s - \delta L_e = 0,\end{aligned} \tag{5}$$

where $\boldsymbol{x}'$, $\boldsymbol{\alpha}'$, $\hat{\boldsymbol{T}}$ and $\hat{\boldsymbol{M}}$ are independent unknowns and the linear form $\mathcal{H}$ must be equal to zero for any compatible variation of the test functions $\delta\boldsymbol{x}'$, $\delta\boldsymbol{\alpha}'$, $\delta\hat{\boldsymbol{T}}$ and $\delta\hat{\boldsymbol{M}}$. The strains $\hat{\boldsymbol{\varepsilon}}$ and $\hat{\boldsymbol{\beta}}$, and their variations, are computed from Eq. 1.

2.2 Cross section model

Following Morandini (2019), the three-dimensional unknown displacement of the beam at a given cross section is locally approximated with a power expansion along the beam axis. To do so, one starts from the left hand side of the principle of virtual work

$$\int_V \delta\boldsymbol{F} : \hat{\boldsymbol{S}} dV = \int_A \delta\hat{\boldsymbol{x}}'(L) \cdot \boldsymbol{f}(L)\,\mathrm{d}A + \int_A \delta\hat{\boldsymbol{x}}'(0) \cdot \boldsymbol{f}(0)\,\mathrm{d}A, \tag{6}$$

where $\boldsymbol{F}$ is the deformation gradient, $\hat{\boldsymbol{S}}$ the first Piola-Kirchhoff stress tensor, and $\boldsymbol{f}$ are external forces per unit of undeformed area, and integrates it by part in the direction of the beam axis, leading to

$$\begin{aligned} &-\int_L\int_A \delta\hat{\boldsymbol{x}}' \otimes \boldsymbol{i}^3 : \hat{\boldsymbol{S}}_{,z}\mathrm{d}A\mathrm{d}z + \int_L\int_A \delta\mathrm{grad}_S(\hat{\boldsymbol{x}}') : \hat{\boldsymbol{S}}\mathrm{d}A\mathrm{d}z + \\ &+\left[\int_A \delta\hat{\boldsymbol{x}}' \cdot \left(\hat{\boldsymbol{S}}\cdot\boldsymbol{n} - \boldsymbol{f}\right)\mathrm{d}A\right]_L + \left[\int_A \delta\hat{\boldsymbol{x}}' \cdot \left(\hat{\boldsymbol{S}}\cdot\boldsymbol{n} - \boldsymbol{f}\right)\mathrm{d}A\right]_0 = 0. \end{aligned} \tag{7}$$

where the cross-section lies onto the x, y, plane, with $\boldsymbol{i}^1$, $\boldsymbol{i}^2$ and $\boldsymbol{i}^3$ the unit vectors along the x, y and z axis, respectively, $\boldsymbol{n}$ if the outward-pointing unit normal (i.e. $\boldsymbol{n} = \boldsymbol{i}^3$ for $z = L$ and $\boldsymbol{n} = -\boldsymbol{i}^3$ for $z = 0$), and the deformation gradient $\boldsymbol{F}$ is decomposed into its in-plane and out-of-plane components as

$$\boldsymbol{F} = \mathrm{grad}_S(\hat{\boldsymbol{x}}') + \hat{\boldsymbol{x}}'_{,z} \otimes \boldsymbol{i}^3, \tag{8}$$

with $\mathrm{grad}_S(\hat{\boldsymbol{x}}') = \hat{\boldsymbol{x}}'_{,x} \otimes \boldsymbol{i}^1 + \hat{\boldsymbol{x}}'_{,y} \otimes \boldsymbol{i}^2$. It is clear from Eq. 7 that the equilibrium along the beam is satisfied if

$$-\int_A \delta\hat{\boldsymbol{x}}' \otimes \boldsymbol{i}^3 : \hat{\boldsymbol{S}}_{,z}\mathrm{d}A + \int_A \delta\mathrm{grad}_S(\hat{\boldsymbol{x}}') : \hat{\boldsymbol{S}}\mathrm{d}A = 0. \tag{9}$$

The displacement $\hat{\boldsymbol{u}} = \hat{\boldsymbol{x}}' - \hat{\boldsymbol{x}}$, with $\hat{\boldsymbol{x}}$ the position vector in the reference configuration, is s approximated around $z = 0$ as

$$\hat{\boldsymbol{u}}(x, y, z) \approx \sum_{i=0}^{N} \frac{1}{i!}\hat{\boldsymbol{u}}_i(x, y) z^i, \tag{10}$$

where the unknown field $\hat{\boldsymbol{u}}_i(x,y)$, is a function of the cross section position only, and is equal to the i-th displacement derivative of the displacement $\hat{\boldsymbol{u}}(x,y,z)$ wrt. z evaluated at $z=0$. Eq. 9, together with its derivatives up to order N, is used to close the nonlinear problem and solve it for the $N+1$ unknown fields $\hat{\boldsymbol{u}}_i$. As an example, the first derivative is equal to

$$\begin{aligned} -\int_A \delta\hat{\boldsymbol{x}}'_{,z} \otimes \boldsymbol{i}^3 : \hat{\boldsymbol{S}}_{,z} \mathrm{d}A - \int_A \delta\hat{\boldsymbol{x}}' \otimes \boldsymbol{i}^3 : \hat{\boldsymbol{S}}_{,zz} \mathrm{d}A + \\ \int_A \delta \left(\mathrm{grad}_S \hat{\boldsymbol{x}}'\right)_{,z} : \hat{\boldsymbol{S}} \mathrm{d}A + \int_A \delta \mathrm{grad}_S \hat{\boldsymbol{x}}' : \hat{\boldsymbol{S}}_{,z} \mathrm{d}A = 0. \end{aligned} \tag{11}$$

Four additional constraints are required as well. The first two impose that the cross- section stress resultant and moment resultant should be equal to the sought values $\hat{\boldsymbol{T}}$ and $\hat{\boldsymbol{M}}$

$$\begin{array}{ll} \int_A \hat{\boldsymbol{S}} \cdot \boldsymbol{i}^3 \mathrm{d}A & = \hat{\boldsymbol{T}}, \\ \int_A \hat{\boldsymbol{x}}' \times \hat{\boldsymbol{S}} \cdot \boldsymbol{i}^3 \mathrm{d}A & = \hat{\boldsymbol{M}}; \end{array} \tag{12}$$

the last two set of equations constraint the average displacement and rotation of the cross section

$$\begin{array}{l} \int_A \hat{\boldsymbol{u}}_0 \mathrm{d}A = \boldsymbol{0}, \\ \int_A \hat{\boldsymbol{x}} \times \hat{\boldsymbol{u}}_0 \mathrm{d}A = \boldsymbol{0}, \end{array} \tag{13}$$

getting rid of rigid-body motion. Eqs. 12 and 13 are imposed by means of four Lagrange multiplier vectors $\boldsymbol{\lambda}_j$, $j \in [1,4]$. The whole set of nonlinear equations will be referred to, in the sequel, as

$$\mathcal{F}(\delta\hat{\boldsymbol{u}}_i, \delta\boldsymbol{\lambda}_j, \hat{\boldsymbol{u}}_i, \boldsymbol{\lambda}_j, \{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}) = 0 \tag{14}$$

where $\hat{\boldsymbol{u}}_i$ and $\boldsymbol{\lambda}_j$ are the unknowns, $i \in [1,N]$, $j \in [1,4]$, and $\delta\hat{\boldsymbol{u}}_i$ and $\delta\boldsymbol{\lambda}_j$ are the test functions; vectors $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$ are the sought cross-section internal actions, and act as independent forcing parameters. The form $\mathcal{F}$ is linear both with respect to the test functions and to the forcing parameters $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$. Standard finite elements are chosen for approximating $\hat{\boldsymbol{u}}_i$; the test function are defined by the same expansion adopted for $\hat{\boldsymbol{u}}$, $\delta\hat{\boldsymbol{u}} = \sum_{i=0}^{N} \frac{1}{i!} \delta\hat{\boldsymbol{u}}_i(x,y) z^i$, and $\delta\hat{\boldsymbol{u}}_i(x,y)$ is approximated with the same finite elements chosen for $\hat{\boldsymbol{u}}_i$.

2.3 Cross-section complementary strain energy derivatives

In order to couple the cross-section model of Section 2.2 with the beam model of Section 2.1 one needs to compute the derivative, with respect to

the internal actions $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$, of the complementary strain energy per unit of beam length v,

$$v = \int_A \boldsymbol{S} : \boldsymbol{\epsilon} - \psi(\boldsymbol{\epsilon}, \boldsymbol{\chi}) \mathrm{d}A, \tag{15}$$

where $\psi(\boldsymbol{\epsilon}, \boldsymbol{\chi})$ is the material internal energy per unit of reference volume at constant temperature, $\boldsymbol{\epsilon} = \frac{1}{2}(\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I})$ is the Green-Lagrange strain tensor, $\boldsymbol{S}$ is the second Piola-Kirchhoff stress tensor, and $\boldsymbol{\chi}$ represents any internal hidden variable that could be needed in order to describe the material response. The Second Piola-Kirchhoff stress tensor $\boldsymbol{S}$ can be computed as $\boldsymbol{S} = \psi_{,\boldsymbol{\epsilon}}$. The functional v is constrained by the equilibrium equations $\mathcal{F}(\delta\boldsymbol{u}, \boldsymbol{u}, \{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}) = 0$, where $\boldsymbol{u} = \{\hat{\boldsymbol{u}}_i, \boldsymbol{\lambda}_i\}$ accounts both for the unknown cross-section displacement $\hat{\boldsymbol{u}}_i$ and the Lagrange multipliers $\boldsymbol{\lambda}_i$. The first and second order adjoint equations allows to compute the derivatives of v with respect to the forcing parameters $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$, see e.g. Hinze et al. (2008). The actual implementation of these adjoint equations mimic that of of dolfin-adjoint library, see Farrell et al. (2013) and Mitusch et al. (2019), but has been re-written in order to leverage the fact that some terms are known to be null, while others are linear. The first derivative of v are computed by resorting to the adjoint variables $\boldsymbol{\lambda}_A$, defined in such a way that

$$\boldsymbol{\lambda}_A^T \mathcal{F}_{,\boldsymbol{u}} = v_{,\boldsymbol{u}}. \tag{16}$$

The first derivative of v with respect to $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$ can thus be computed as

$$\frac{\mathrm{d}v}{\mathrm{d}\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}} = -\boldsymbol{\lambda}_A^T \mathcal{F}_{,\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}} \tag{17}$$

where the expression for $\mathcal{F}_{,\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}}$ is trivial since $\mathcal{F}$ is linear with respect to $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$. The second derivative of v with respect to to the k-th component of $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$,can be computed by deriving Eq. 17:

$$\left(\frac{\mathrm{d}^2 v}{\mathrm{d}\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}^2}\right)_{(k,:)} = -\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}_{(k)}} \mathcal{F}_{,\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}} \tag{18}$$

where $\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}_{(k)}}$ is the derivative of $\boldsymbol{\lambda}_A^T$ with respect to the k-th component of $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$ and we are making use of the fact that the second derivative $\mathrm{d}(\mathcal{F}_{,\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}})/\mathrm{d}\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}_{(k)}$ is null because $\mathcal{F}$ is linear with respect to $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{M}}\}$

and $\mathcal{F}_{,\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}}$ is not function of $\boldsymbol{u}$, cfr. Eq. 12. The derivative $\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}_{(k)}}$ can be computed by deriving Eq. 16

$$\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}_{(k)}}\mathcal{F}_{,\boldsymbol{u}} + \boldsymbol{\lambda}_A^T(\mathcal{F}_{,\boldsymbol{uu}}\boldsymbol{u}_{,k}) = v_{,\boldsymbol{uu}}\boldsymbol{u}_{,k} \tag{19}$$

where again we have made use of the fact that $\mathcal{F}_{,\boldsymbol{u}k} = 0$. the derivative of $\boldsymbol{u}$ with respect to the k-th component of $\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$, $\boldsymbol{u}_{,k}$, can be computed by solving six linear equations:

$$\mathcal{F}_{,\boldsymbol{u}}\boldsymbol{u}_{,\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}} = -\mathcal{F}_{,\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}}. \tag{20}$$

Since $\boldsymbol{u}_{,k}$ and $\boldsymbol{\lambda}_A$ can be computed independently Eq. 19 is nothing but a linear system of equations with $\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}_{(k)}}$ as unknown:

$$\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}_{(k)}}\mathcal{F}_{,\boldsymbol{u}} = v_{,\boldsymbol{uu}}\boldsymbol{u}_{,k} - \boldsymbol{\lambda}_A^T(\mathcal{F}_{,\boldsymbol{uu}}\boldsymbol{u}_{,k}) \tag{21}$$

After solving Eq. 21 for $\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}_{(k)}}$ the k-th row of $\frac{\mathrm{d}^2 v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2}$ is readily given by Eq. 18. The computation of $\frac{\mathrm{d}^2 v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2}$ requires the solution of twelve linear systems: for each row of $\frac{\mathrm{d}^2 v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2}$ one needs to solve two linear systems: the first for computing $\boldsymbol{u}_{,k}$ and the second for computing $\frac{\mathrm{d}\boldsymbol{\lambda}_A^T}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}_{(k)}}$. The linear solution required for computing the first order adjoint variables $\boldsymbol{\lambda}_A$ usually brings no additional cost, since $\boldsymbol{\lambda}_A$ is already required in order to compute the first order derivative of v.

2.4 The two-level scheme

The beam model of Section 2.1 is coupled to the cross-section model of Section 2.2. The global model makes use of linear continuous interpolating functions for the deformed position $\boldsymbol{x}'$ and the rotation vector $\boldsymbol{\phi}$, defined in such a way that $\boldsymbol{\alpha}' = \exp(\boldsymbol{\phi}\times)\boldsymbol{\alpha}$. The unknown internal force and moments $\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$ are approximated with constant piece wise discontinuous functions. Each finite element of the global model has associated a mesh of the cross section, as in Figure 1, together with suitable approximating functions for all the $\hat{\boldsymbol{u}}_i$. The cross-section local model receives as forcing parameters the

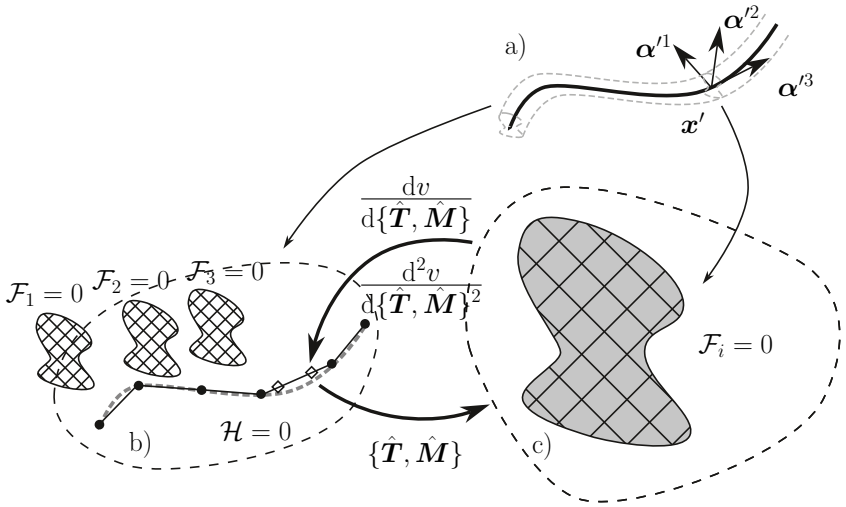

Figure 1: Interaction between the global beam and local cross-section models.

values of the internal force and moment vectors, $\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$, and solves the local nonlinear problem $\mathcal{F} = 0$. After that, it can compute the values of $\mathrm{d}v/\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$ and $\mathrm{d}^2v/\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2$ that are required for evaluating $\mathcal{H}$ and its linearization, respectively: the beam model, after the cross section model has computed $\mathrm{d}v/\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$ and $\mathrm{d}^2v/\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2$ can readily evaluate

$$\delta v = \delta\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\} \cdot \mathrm{d}v/\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$$

and

$$\partial\delta v = \delta\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\} \cdot \mathrm{d}^2v/\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2 \cdot \partial\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\},$$

where $\partial\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$ are the trial functions. This data exchange needs to be performed at each Newton iteration of the global model. As a consequence, the inner model keeps solving the nonlinear problem $\mathcal{F} = 0$, whenever the value of $\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}$ change, before evaluating the first and second derivatives of v that are required by the global model while iterating toward $\mathcal{H} = 0$.

3 Implementation

This section details the actual implementation strategy. The code is written using the Python Dolfin wrappers. Section 3.1 showcases the actual implementation, Section 3.2 explains why it was necessary to introduce a limited

form of caching, and Section 3.3 discusses a significant bottleneck related to the current handling strategy of the just-in-time UFL forms compilation.

3.1 Implementation: the good part

Thanks to the flexibility of Dolfin's Python wrappers a basic implementation of the above coupling scheme turns out to be relatively straightforward. A single local cross-section is wrapped within an instance of a `BeamSection` class:

```
class BeamSection():
    # initialization
    def __init__(self):
        ....
    # solve the cross section variational problem
    def solve(self, force, moment):
        # solve Eq. 14
        ....
    # compute the first derivative of the
    # complementary strain energy with respect to
    # the internal force and moment
    def delta_v(self, force, moment):
        # need first to reach convergence
        self.solve(force, moment)
        # compute the derivative dv/d{T,M} from Eqs. 16 and 17
        ....
    # compute the second derivative of the
    # complementary strain energy with respect to
    # the internal force and moment
    def de_delta_v(self, force, moment):
        # first go to convergence and compute λ_A
        self.delta_v(force, moment)
        for k in range(6):
            # compute u_{,k} from Eq. 20
                ....
            # compute d(λ_A^T)/d{T,M}_(k) from Eq. 21
                ....
            # compute (d^2 v / d{T,M}^2)_(k,:) from Eq. 18
                ....
```

where the relevant methods are

`solve` to find the solution of $\mathcal{F} = 0$;

`delta_c` to compute the first derivative $\frac{\mathrm{d}v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}}$;

`de_delta_v` to compute the second derivative $\frac{\mathrm{d}^2 v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2}$.

The whole set of cross-section instances, one for each cell of the global mesh, is stored inside a vector, `beam_sect`. At the global level, the `BeamSection`'s methods `delta_v` and `de_delta_v` are wrapped leveraging Dolfin's `UserExpression` class. The first derivative reads

```
class delta_v_expression(UserExpression):
    # the constructor takes a reference to
    # the global problem unknown vector, u
    def __init__(self, u, *arg, **kwargs):
        UserExpression.__init__(self, *arg, **kwargs)
        self.u = u
    # define proper value_shape
    def value_shape(self):
         return (6,)
    # evaluate dv/d{T,M}
    def eval_cell(self, value, x, ufc_cell):
        # compute the global unknown values taken
        # by this element at x
        # and store them into self.uvalues
        self.u.eval_cell(self.uvalues, x, ufc_cell)
        # extract the force and moment resultant from self.uvalues
        self.Tc.assign(Constant((self.uvalues[7], ...)))
        self.Mc.assign(Constant((self.uvalues[10], ...)))
        # actually compute dv/d{T,M} for the cross-section
        # beams_sec[ufc_cell.index]
        value = beams_sec[ufc_cell.index].delta_v(self.Tc, self.Mc).vector()
```

with the second derivative closely matching the first:

```
class de_delta_v_expression(UserExpression):
    # the constructor takes a reference to
    # the global problem unknown vector, u
    def __init__(self, u, *arg, **kwargs):
        UserExpression.__init__(self, *arg, **kwargs)
        self.u = u
    # define proper value_shape
    def value_shape(self):
         return (6, 6)
    # ... and value_rank
    def value_rank(self):
```

```
        return 2
    # evaluate d²v/d{T̂,M̂}²
    def eval_cell(self, value, x, ufc_cell):
        # compute the global unknown values taken
        # by this element at x
        # and store them into self.uvalues
        self.u.eval_cell(self.uvalues, x, ufc_cell)
        # extract the force and moment resultant from self.uvalues
        # and store them int self.Tc, self.Mc
        self.Tc.assign(Constant((self.uvalues[7], ... )))
        self.Mc.assign(Constant((self.uvalues[10], ...)))
        # actually compute dv/d{T̂,M̂} for the cross-section
        # beams_sec[ufc_cell.index] ...
        (delta_v, de_delta_v) = beams_sec[ufc_cell.index].de_delta_v(self.Tc, self.Mc)
        # ... and store it in the vector used to return the result
        for i in range(6):
             for j in range(6)
                  value[i * 6 + j] = de_delta_v[i].vector()[j]
```

Having at hand these two `UserExpression` classes the implementation of the global model Eq. 5 is straightforward:

```
# bring  together the resultant and moment resultant test functions δT̂ and δM̂
merged_V = as_vector([v_T[0], v_T[1], v_T[2], v_M[0], v_M[1], v_M[2]])
# build an instance of the dv/d{T̂,M̂} UserExpression
delta_v_expr = delta_v_expression(u, element = AZ2_EL)
# δv = δ{T̂,M̂} · dv/d{T̂,M̂}
delta_v = inner(merged_V, delta_v_expr)
# derivatives of linear and angular strain measures
delta_epsilon = derivative(epsilon, u, v)
delta_beta = derivative(beta, u, v)
# define the linear form
functional = inner(delta_epsilon, u_T) * dx + \
        inner(delta_beta, u_M) * dx + \
        inner(vu_T, epsilon) * dx + \
        inner(vu_M, beta) * dx - \
        delta_v * dx - \
        .... # add forcing terms here
```

where `epsilon` and `beta` are defined from Eq. 1, `delta_epsilon and delta_beta` are the corresponding test functions, `v` are the problem test functions, `u` the vector of unknowns, `v_T, v_M, u_T and u_M` the portions of `v` and `u` related to the the internal force $\boldsymbol{T}$ and moment $\boldsymbol{M}$ resultants, respectively,

and `merged_V` wraps the test functions of both the internal force and moment resultants, $\boldsymbol{T}$ and $\boldsymbol{M}$. The linearization is straightforward as well

```
# bring  together the resultant and moment resultant trial functions ∂T̂ and ∂M̂
merged_T = as_vector([d_T[0], d_T[1], d_T[2], d_M[0], d_M[1], d_M[2]])
# build an instance of the d²v/d{T̂,M̂}² UserExpression
de_delta_v_expr = de_delta_v_expression(u, delta_v_expr, element = TANG2_EL)
# ∂δv = δ{T̂,M̂} · d²v/d{T̂,M̂}² · ∂{T̂,M̂}
de_delta_v = inner(merged_V, dot(de_delta_v_expr, merged_T))
# define the bilinear form
J = derivative(functional, u, du) - de_delta_v * dx
```

where `du` are the problem trial functions, `d_T and d_M` the portions of `du` related to the the internal force $\boldsymbol{T}$ and moment $\boldsymbol{M}$ resultants, respectively, and `merged_T` wraps the trial functions of both the internal force and moment resultants, $\boldsymbol{T}$ and $\boldsymbol{M}$. Since UFL has no clues about the dependency of the first variation of the complementary strain energy `delta_v` on the problem unknowns, one needs to add the `de_delta_v` term by hand instead of relying on UFL's symbolic differentiation.

3.2 Implementation: the not-so-good part

The basic implementation of Section 3.1 can be significantly sped up by a proper caching of results. Since the global beam model has more than one point of integration for each element, the assembly loop will call the lower cross-section model methods more than once for each element. Since however the internal actions are constant within an element, it is advisable to cache the last computed result, and avoid recomputing it if the element does not change and the value of the internal actions are unchanged as well. To keep the code simple only the last computed value is cached. Furthermore, some additional saving can be achieved by adding a caching layer also at the cross-section level. For example, if the code has already computed the first derivative of the complementary strain energy `delta_v`, there is no need to compute it again when the second derivative method `de_delta_v` is called. For this reason the code tracks the computed values everywhere, and recompute them only if needed. As an example, the actual implementation of `delta_v` becomes

```
def delta_v(self, force, moment):
```

```
# if the values of force and moment are different
# from the previous ones
if (not(
    numpy.array_equal(force.values(), self.previous_Tc) and
    numpy.array_equal(moment.values(), self.previous_Mc) and
    numpy.array_equal(force.values(), self.previous_Tc_dv) and
    numpy.array_equal(moment.values(), self.previous_Mc_dv)))
    # store the current values into the ‘‘previous one’’ vectors
    self.FORCE_IA.assign(force)
    self.MOMENT_IA.assign(moment)
    self.previous_Tc_dv[:] = self.FORCE_IA.values()
    self.previous_Mc_dv[:] = self.MOMENT_IA.values()
    # and perform the actual computations
    self.solve(force, moment)
    ....
```

and this kind of caching, although relatively benign and not too invasive, is scattered all around, both for the solution of $\mathcal{F} = 0$ and for the computation of $\frac{\mathrm{d}v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}}$ and $\frac{\mathrm{d}^2v}{\mathrm{d}\{\hat{\boldsymbol{T}},\hat{\boldsymbol{M}}\}^2}$. This significantly speeds up the solution procedure.

3.3 Implementation: the weird detail

The Dolfin Python wrapper was crucial for a fast and almost frictionless implementation of the coupling procedure. However, after everything was in place, it soon turned out that the startup time of each and every simulation was unnecessarily long, exceeding, for some specific test cases, the actual computational time and literally wasting hours of computational time. This is due to a critical difference between the how the C++ Dolfin library and its Python wrapper deals with the code generation.

When directly dealing with the C++ library the user is supposed to run the form compiler, generate the C++ code that computes the forms, and then use the C++ compiler with these automatically-generated code. Since the file defining the form is well–separated from the user-written code that drives the computations, the user is in control, and can run the form compiler only when the forms do actually change.

For the Dolfin wrappers, however, things are slightly different: since the form definition is often mixed within the code that drives the solution, the Python wrapper tries to be be “smart”: it computes a unique signature of the form at hand and, if the signature if new, it first runs the form compiler, then the C++ compiler and finally stores the resulting files into an on-disk

cache, identified by that particular, unique signature. If the cache already holds the code, instead, it reuses the previously compiled files.

This caching strategy is extremely handy and user-friendly. This convenience, however, brings an overhead, since the code needs to compute this unique signature. The overhead is luckily very small for the vast majority of users and use-cases, but can be significant for some of the forms at required for this applications.

The problem is exacerbated even more for the application at hand because there can be tens or hundreds of identical forms, one for for every low-level cross-section problem: there is a cross-section problem for each beam element. Since problems with complex hyperelastic material constitutive laws turned out to spend up to twenty minutes only for computing the signature of a single form, repeating these computations over and over for tens or hundreds cross-section problems turned out to be unbearable.

A possible solution could be to define the Form once for all, and change on the fly the mesh and the Functions that should be used to perform the computation; unfortunately the Dolfin Form wrapper does not make easy to do this. The alternative of defining a unique cross-section problem and copying the data there would not only complicate the code, but also prevent the possibility of dealing with different cross-section, and thus different meshes.

It's clear that the real solution to this would be to ditch the Python implementation of the coupling procedure, and re-write it in C++. An acceptable stop-gap solution turned out to override the Form signature computation, replacing it with the hash of the Python file defining the form, and forcing a rebuild whenever there were some changes, in a different file, that I knew could modify the form and would not be spotted by this file hash.

4 Examples

The correctness of the proposed coupling scheme is confirmed by the fact that quadratic convergence is achieved both for the local cross-section problem and the global beam problem. As an example, Table 1 shows, for two different time steps, the residual norm history of the global problem of the PVC-copper cross section shown in Sec. 4.2.

Sections 4.1 and 4.2 show two examples selected among those reported in Morandini (2020).

Table 1: Global beam model convergence.

	Residual norm	
Iteration	$t = 0.2$	$t = 0.4$
0	0.5	0.5
1	0.00317	0.04223
2	0.00939	0.00108
3	0.004111	3.17E-06
4	0.000060	–

4.1 Bimetallic beam

A straight beam of length 10 mm has the bimetallic 1×1 mm cross-section of Fig. 2. It is clamped at one end, and loaded by a transverse force F=1 N at the other. The cross-section is made with two different materials. Both materials are isotropic and elasto-plastic. The constitutive law is based on an additive decomposition of the Green-Lagrange strain tensor, $\boldsymbol{S} = \mathbb{E} : (\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_p)$, where $\boldsymbol{\epsilon}_p$ is the plastic deformation tensor; a standard Von-Mises yield function f with isotropic hardening

$$f = \sqrt{\frac{3}{2}\boldsymbol{s} : \boldsymbol{s}} - (S_0 + K) = 0 \tag{22}$$

is assumed, with $\boldsymbol{s} = \boldsymbol{S} - \frac{1}{3}\boldsymbol{S} : \boldsymbol{I}$, S_0+K the equivalent yield stress. The internal energy is

$$nd\psi(\boldsymbol{\epsilon}, \boldsymbol{\chi}) = \frac{1}{2}(\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_p) : \mathbb{E} : (\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_p) + \frac{1}{2}H\boldsymbol{\epsilon}_p^{eff}\boldsymbol{\epsilon}_p^{eff}$$

and as associated flow rule is assumed.

The constitutive law is defined by the material elastic modulus E, the elasto-plastic tangent modulus E_t, the Poisson coefficient ν and the yield stress S_0, so that the hardening parameter is $H = E_t/(1 - E_t/E)$. The elastic modulus, Poisson coefficient and elasto-plastic tangent modulus are equal for both materials, E=1200 MPa, $\nu = 0.3$ and $E_t = 360\,\text{MPa}$; the two material yield stresses do differ, and are equal to $S_0 = 2.4$ MPa and $S_0 = 12$ MPa for materials Mat. 1 and Mat. 2, respectively.

Figure 3 compares the displacement of the beam model loaded point with that of a three dimensional simulation. Figures 4 and 5 plot the root section equivalent plastic strain and the norm of the normal stress vector,

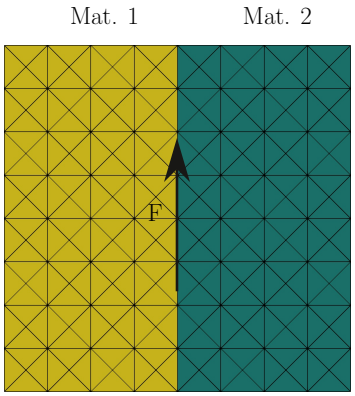

Figure 2: Bimetallic beam cross-section (from Morandini, 2020).

respectively. As expected, the left half of the beam, made with Mat. 1, undergoes a significant plastic deformation, with the normal stress vector limited to values that are smaller than those of the right half of the beam, that is made with Mat. 2.

4.2 Complex cross-section

A circular composite PVC wire of length $l = 10$ mm is discretized with 10 beam elements. It has a circular cross-section with a radius $R = 0.5$ mm and 19 smaller copper wires, each of radius $r = 0.08$ mm, as shown in Fig. 6. The PVC is assumed to be elastic, while the copper is elasto-plastic. The material properties are reported in Table 2. The beam, clamped at one end, is subject to a concentrated shear force $F = 5$ N at the other extremity. The load is increased linearly from $t = 0$ to $t = 1$, and them brought to 0 for $t = 2$. A three dimensional mesh with well-shaped constant stress tetrahedron would require about 450 thousand nodes, for about 1.5 million unknowns, and is out of reach on the desktop computer used for these computations.

Figure 7 plots the deformed configuration taken by the beam when the load reaches it maximum at $t = 1$ and after unloading the structure, at $t = 2$. The corresponding beam tip displacement components are reported in Fig. 8, where x is along the beam axis and the load is applied in the z direction. Figures 9 plot the deformed root cross-section, with the colors representing the cross-section out of plane displacement for the fully loaded ($t = 1$, left) and unloaded ($t = 2$, right) configurations. Figure 10 plots,

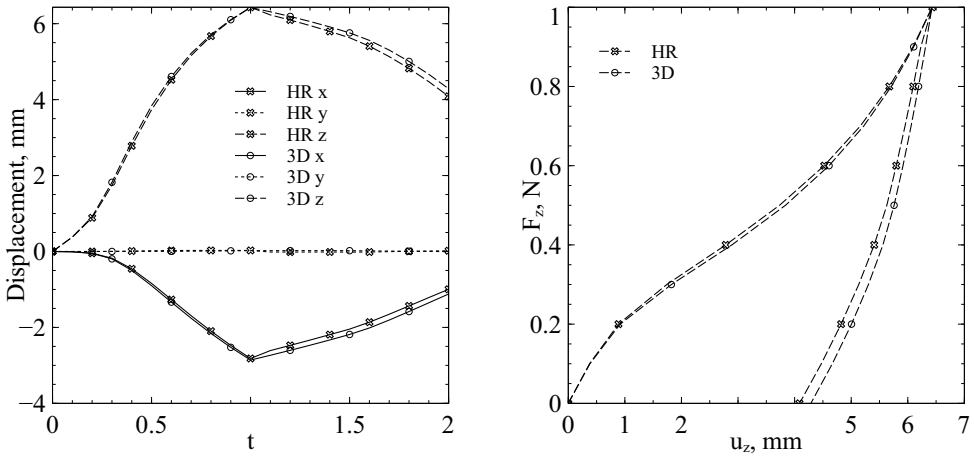

Figure 3: Dual material elasto-plastic beam: loaded point displacement components as a function of time (left) and corresponding load-displacement curve (right) (from Morandini, 2020).

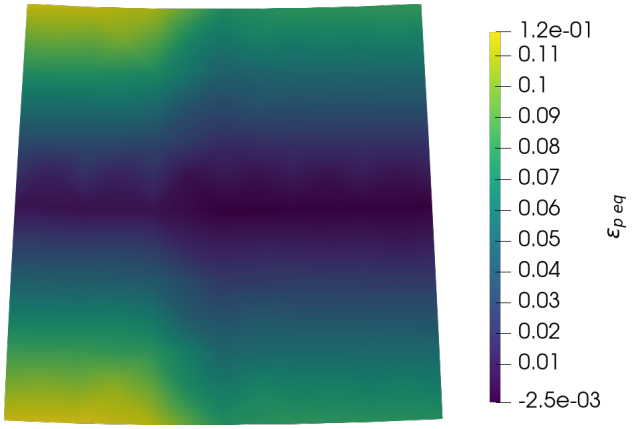

Figure 4: Dual material elasto-plastic beam: equivalent plastic deformation at $t = 1$ (from Morandini, 2020).

Table 2: Composite wire material properties.

	Copper	PVC
E	117 GPa	4.1 GPa
ν	0.3	0.41
S_0	70 MPa	/
E_t	2.34 GPa	/

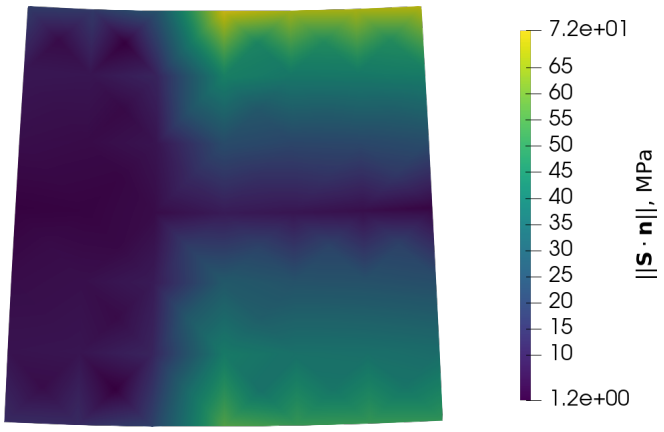

Figure 5: Dual material elasto-plastic beam: norm of the normal stress vector at $t = 1$ (from Morandini, 2020).

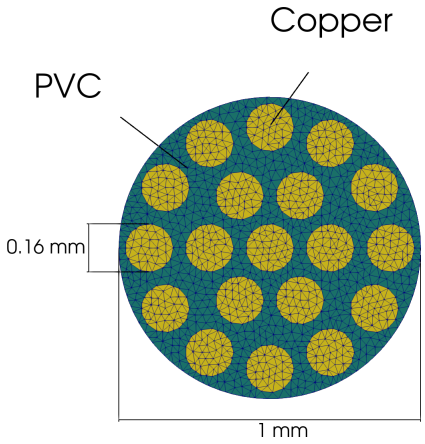

Figure 6: Mesh of the composite wire (from Morandini, 2020).

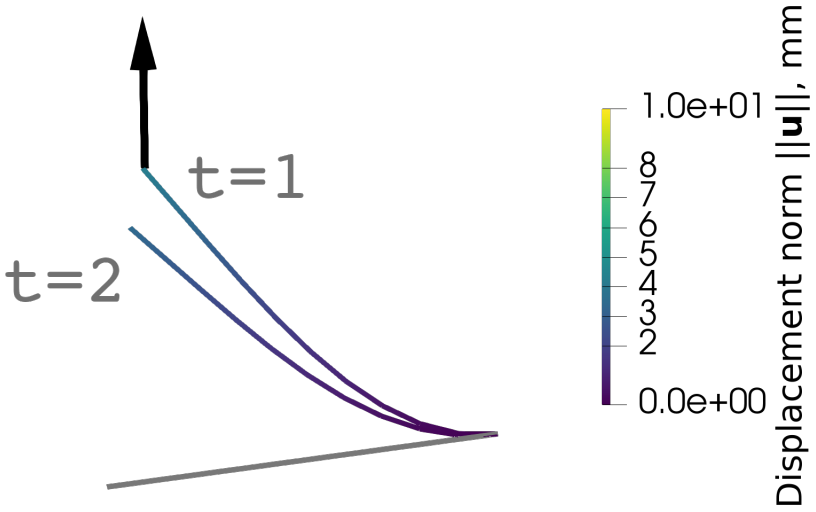

Figure 7: Elasto-plastic wire beam deformed configurations (from Morandini, 2020).

instead, the equivalent plastic strain $\boldsymbol{\epsilon}_{p\,eq}$ of the copper wires; some wires undergo a significant plastic deformation.

5 Run time

As already noted in Morandini (2020), the proposed approach turns out to be faster than a fully three dimensional analysis only if the spatial resolution required on the cross section is relatively high with respect the the resolution along the axis. This is because the need to solve a nonlinear cross-section problem each and every time the residual or the Jacobian matrix of the beam problem need to be assembled really reduces the possible gains of the two-level procedure. Because of this the run time turns to be higher than that of a completely three dimensional simulation if the cross-section mesh is small. The higher the number of elements in the section, the greater the computational savings with respect to a three dimensional analysis. As an example, Table 3 compares the run time required by a simple elastic problem, a bent beam, with ten beam element, for different cross-section meshes.

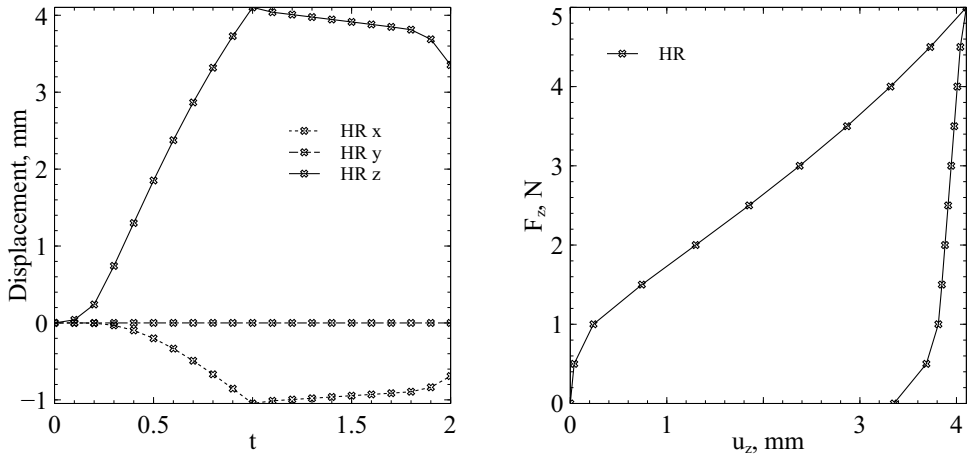

Figure 8: Elasto-plastic wire beam: loaded point displacement components as a function of time (left) and corresponding load-displacement curve (right) (from Morandini, 2020).

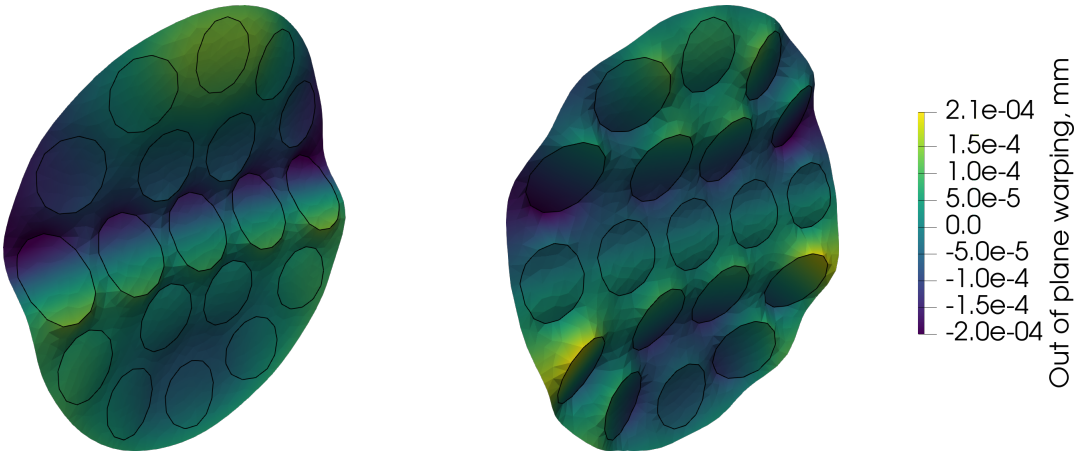

Figure 9: Composite wire out of plane warping, root element cross section: maximum load (left, $t = 1$) and final unloaded state (right, $t = 2$); deformation scale factor: 300 (from Morandini, 2020).

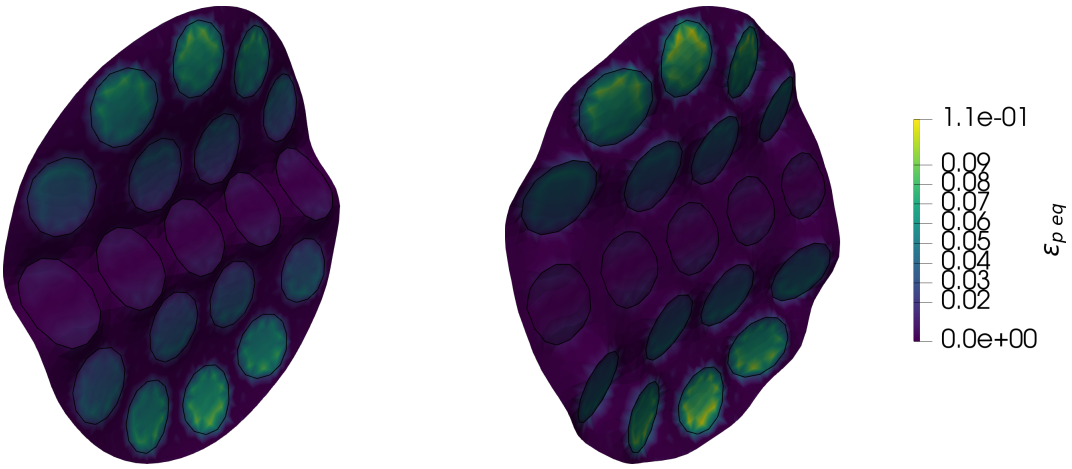

Figure 10: Composite wire equivalent plastic strain, root element cross section: maximum load (left, $t = 1$) and final unloaded state (right, $t = 2$); deformation scale factor: 300 (from Morandini, 2020).

Table 3: Timing comparison between the proposed approach and a fully three-dimensional solution (data from Morandini, 2020).

Section 10×10			Section 20×20			Section 40×40		
Beam	3D	Ratio	Beam	3D	Ratio	Beam	3D	Ratio
1033 s	926 s	1.12	4112 s	5810 s	0.71	29540 s	51598 s	0.57

6 Conclusions

The proposed two-level approach is not competitive with classic kinematic approaches whenever it is reasonable to assume an axial stress state and it is possible to rephrase the constitutive law for it. Such standard beam finite elements, however, fail to deal with hyperelastic constitutive laws, and are limited to relatively simple, homogeneous cross-sections. Thus, the proposed approach can be one of the few viable alternatives for very specific beam problems. Furthermore, the present approach guarantees that the beam response is hyperelastic if the cross-section material is hyperelastic, a correctness property not shared, to the author's knowledge, with different approaches.

The basic implementation of this somewhat complex coupling scheme and of the related complex linear and bilinear forms turned out to be relatively straightforward. This is, without any doubt, thanks to the flexibility of the FEniCS framework and to the ease of use of the Dolfin Python wrappers. However, the optimization of the code and its use for the simulation of complex problems highlighted a few pain points, the more relevant being how the JIT compilation of the UFL Forms is handled and on how these forms are wrapped within Dolfin.

References

Alnæs, M.S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N., 2015. The fenics project version 1.5. Archive of Numerical Software 3. doi:10.11588/ans.2015.100.20553.

Bilotta, A., Garcea, G., 2019. A two-level computational approach for the elasto-plastic analysis of framed structures with composite cross-sections. Composite Structures 209, 192–205. URL: `http://www.sciencedirect.com/science/article/pii/S0263822318321998`, doi:10.1016/j.compstruct.2018.10.056.

Cardona, A., Geradin, M., 1988. A beam finite element non-linear theory with finite rotations. Int. J. Numer. Meth. Engng 26, 2403–2438.

Chiorean, C.G., 2017. Second-order flexibility-based model

for nonlinear inelastic analysis of 3d semi-rigid steel frameworks. Engineering Structures 136, 547–579. URL: http://www.sciencedirect.com/science/article/pii/S0141029617301992, doi:10.1016/j.engstruct.2017.01.040.

Farrell, P., Ham, D., Funke, S., Rognes, M., 2013. Automated Derivation of the Adjoint of High-Level Transient Finite Element Programs. SIAM Journal on Scientific Computing 35, C369–C393. URL: https://epubs.siam.org/doi/10.1137/120873558, doi:10.1137/120873558.

Hinze, M., Pinnau, R., Ulbrich, M., Ulbrich, S., 2008. Optimization with PDE Constraints. 2009 edition ed., Springer, Dordrecht.

Logg, A., Wells, G.N., Hake, J., 2012. DOLFIN: a C++/Python Finite Element Library. Springer. chapter 10.

Merlini, T., Morandini, M., 2013. On successive differentiations of the rotation tensor: An application to nonlinear beam elements. Journal of Mechanics of Materials and Structures 8, 305–340. URL: https://msp.org/jomms/2013/8-5/p03.xhtml, doi:10.2140/jomms.2013.8.305.

Mitusch, S., Funke, S., Dokken, J., 2019. dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake. Journal of Open Source Software 4, 1292. URL: https://joss.theoj.org/papers/10.21105/joss.01292, doi:10.21105/joss.01292.

Morandini, M., 2017. Handling of finite rotations in Dolfin, in: Proceedings of the FEniCS Conference 2017, Jack S. Hale, University of Luxembourg, Luxembourg. URL: http://dx.doi.org/10.6084/m9.figshare.5086369, doi:10.6084/m9.figshare.5086369.

Morandini, M., 2019. Analysis of beam cross section response accounting for large strains and plasticity. International Journal of Solids and Structures 176-177, 150–172. URL: http://www.sciencedirect.com/science/article/pii/S0020768319302471, doi:10.1016/j.ijsolstr.2019.05.014.

Morandini, M., 2020. A two-level nonlinear beam analysis method. International Journal of Solids and Structures 203, 224–235. URL: http://www.sciencedirect.com/science/article/pii/S0020768320303024, doi:10.1016/j.ijsolstr.2020.08.003.

Pietraszkiewicz, W., Eremeyev, V.A., 2009. On natural strain measures of the non-linear micropolar continuum. Int. J. Solids Struct. 46, 774–787.

Rezaiee-Pajand, M., Gharaei-Moghaddam, N., 2015. Analysis of 3d Timoshenko frames having geometrical and material nonlinearities. International Journal of Mechanical Sciences 94-95, 140–155. URL: http://www.sciencedirect.com/science/article/pii/S0020740315000685, doi:10.1016/j.ijmecsci.2015.02.014.

Rigobello, R., Breves Coda, H., Munaiar Neto, J., 2013. Inelastic analysis of steel frames with a solid-like finite element. Journal of Constructional Steel Research 86, 140–152. URL: http://www.sciencedirect.com/science/article/pii/S0143974X13001016, doi:10.1016/j.jcsr.2013.03.023.