POLITECNICO
MILANO 1863

A two-level nonlinear beam model using adjoints

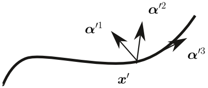
Marco Morandini
marco.morandini@polimi.it

Politecnico di Milano

FEniCS 2021, March 22-26

Beam model

- Domain: line
- Unknowns: $\boldsymbol{x}'$, $\boldsymbol{\alpha}'$
- Needs constitutive law $\{\boldsymbol{T}, \boldsymbol{M}\}(\epsilon, \beta)$



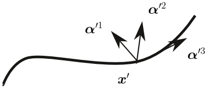
Handling of finite rotations in Dolfin,
proc. FEniCS Conference 2017

https://home.aero.polimi.it/morandini/Downloads/DolfinFiniteRotations/html/

Introduction

Beam model

- Domain: line
- Unknowns: $\boldsymbol{x}'$, $\boldsymbol{\alpha}'$
- Needs constitutive law $\{\boldsymbol{T}, \boldsymbol{M}\}(\epsilon, \beta)$

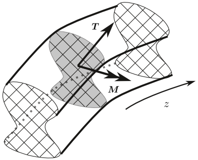


Handling of finite rotations in Dolfin, proc. FEniCS Conference 2017

https://home.aero.polimi.it/morandini/Downloads/DolfinFiniteRotations/html/

Cross-section models

- Domain: cross section (area)
- Unknowns: $\hat{\boldsymbol{u}}_i$
- Known 3D constitutive law
- Function of $\{\boldsymbol{T}, \boldsymbol{M}\}$ (forcing term)



Analysis of beam cross section response accounting for large strains and plasticity, IJSS, 2019.

Two-level model

1. Beam model:
   from $\{\boldsymbol{x}', \boldsymbol{\alpha}'\}$

$$\int_L (\delta\varepsilon \boldsymbol{T} + \delta\beta \boldsymbol{M})\,\mathrm{d}s - \delta L_e = 0$$

Two-level model

1. Beam model:
   from $\{\boldsymbol{x}', \boldsymbol{\alpha}'\}$

$$\int_L (\delta\varepsilon \boldsymbol{T} + \delta\beta \boldsymbol{M})\, \mathrm{d}s - \delta L_e = 0$$

   to $\{\boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{T}, \boldsymbol{M}\}$

Two-level model

1. Beam model:
   from $\{\boldsymbol{x}', \boldsymbol{\alpha}'\}$

$$\int_L (\delta\varepsilon\boldsymbol{T} + \delta\beta\boldsymbol{M})\,\mathrm{d}s - \delta L_e = 0$$

to $\{\boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{T}, \boldsymbol{M}\}$

complementary strain energy

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

Two-level model

1. Beam model:
   from $\{\boldsymbol{x}', \boldsymbol{\alpha}'\}$

$$\int_L (\delta\boldsymbol{\varepsilon}\boldsymbol{T} + \delta\boldsymbol{\beta}\boldsymbol{M})\,\mathrm{d}s - \delta L_e = 0$$

   to $\{\boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{T}, \boldsymbol{M}\}$

   complementary strain energy

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

$$\mathcal{H}(\delta\{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}, \{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}) = \int_l (\delta\boldsymbol{\epsilon}\boldsymbol{T} + \delta\boldsymbol{\beta}\boldsymbol{M} + \delta\boldsymbol{T}\boldsymbol{\epsilon} + \delta\boldsymbol{M}\boldsymbol{\beta} - \delta v)\,\mathrm{d}s - \delta L_e = 0$$

Two-level model

1. Beam model:
   from $\{\boldsymbol{x}', \boldsymbol{\alpha}'\}$

$$\int_L (\delta\varepsilon \boldsymbol{T} + \delta\beta \boldsymbol{M})\,\mathrm{d}s - \delta L_e = 0$$

to $\{\boldsymbol{x}', \boldsymbol{\alpha}', \boldsymbol{T}, \boldsymbol{M}\}$

complementary strain energy

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

$$\mathcal{H}(\delta\{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}, \{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}) = \int_l (\delta\boldsymbol{\epsilon}\boldsymbol{T} + \delta\boldsymbol{\beta}\boldsymbol{M} + \delta\boldsymbol{T}\boldsymbol{\epsilon} + \delta\boldsymbol{M}\boldsymbol{\beta} - \delta v)\,\mathrm{d}s - \delta L_e = 0$$

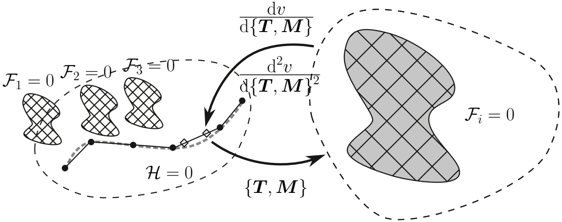
2. Cross section model: $\mathcal{F}(\delta\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i, \{\boldsymbol{T}, \boldsymbol{M}\}) = 0$
   compute

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

on the cross-section $\mathcal{F} = 0$ with $\{\boldsymbol{T}, \boldsymbol{M}\}$ from beam model

Two-level model

Global model: $\mathcal{H}(\delta\{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}, \{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\})$

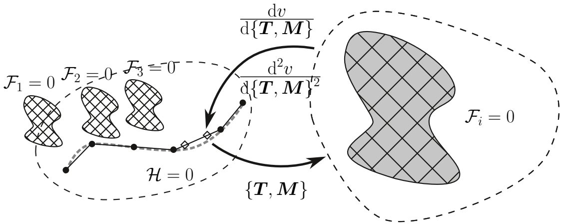


$\delta v = \delta\{\boldsymbol{T}, \boldsymbol{M}\} \cdot \frac{\mathrm{d}v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}}$

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

Two-level model

Global model: $\mathcal{H}(\delta\{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}, \{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\})$



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

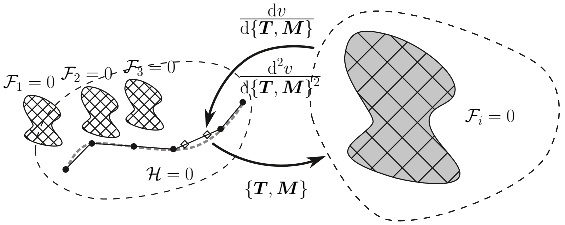
$\partial\delta v = \delta\{\boldsymbol{T}, \boldsymbol{M}\} \cdot \frac{\mathrm{d}^2 v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}^2} \cdot \partial\{\boldsymbol{T}, \boldsymbol{M}\}$

Cross section: $\mathcal{F}(\delta\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i, \{\boldsymbol{T}, \boldsymbol{M}\}) = 0$

- $\frac{\mathrm{d}v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}}$: first order adjoint

Two-level model

Global model: $\mathcal{H}(\delta\{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\}, \{\boldsymbol{\epsilon}, \boldsymbol{\beta}, \boldsymbol{T}, \boldsymbol{M}\})$



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

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

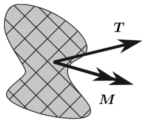
Cross section: $\mathcal{F}(\delta\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_i, \{\boldsymbol{T}, \boldsymbol{M}\}) = 0$

- $\frac{\mathrm{d}v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}}$: first order adjoint
- $\frac{\mathrm{d}^2 v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}^2}$: second order adjoint

Implementation

Cross section:

```
class BeamSection():
    # solve the $\mathcal{F} = 0$
    def solve(self, force, moment):
    ....
    # compute the derivative $\frac{dv}{d\{T,M\}}$
    def delta_v(self, force, moment):
    ....
    # compute the derivative $\frac{d^2v}{d^2\{T,M\}}$
    def de_delta_v(self, force, moment):
    ....
```



where

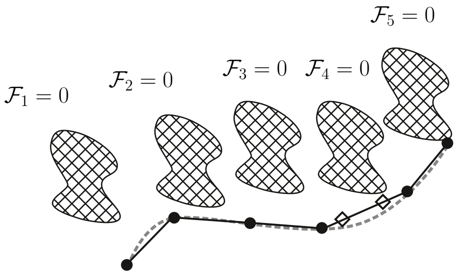
`solve` find the solution of $\mathcal{F} = 0$;

`delta_v` compute the first derivative $\frac{\mathrm{d}v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}}$

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

Implementation

Global beam model $\mathcal{H} = 0$:
store array `beams_sec` of `BeamSections`,
one for each cell



Implementation

Global beam model $\mathcal{H} = 0$, array `beams_sec`

Global beam model $\mathcal{H} = 0$, array `beams_sec`

```
class delta_v_expression(UserExpression):

    # evaluate dv/d{T,M}
    def eval_cell(self, value, x, ufc_cell):

        ...

        value = beams_sec[ufc_cell.index].delta_v(self.Tc, self.Mc)
```

Global beam model $\mathcal{H} = 0$, class `delta_v_expression`

$$\int_l (\delta\epsilon \boldsymbol{T} + \delta\beta \boldsymbol{M} + \delta \boldsymbol{T}\epsilon + \delta \boldsymbol{M}\beta - \delta v)\,\mathrm{d}s - \delta L_e$$

```
# δT and δM
test_FM = as_vector([v_F[0], v_F[1], ...])

# dv/d{T,M}
delta_v_expr = delta_v_expression(u, element = AZ2_EL)
# δv = δ{T, M} · dv/d{T,M}
delta_v = inner(test_FM, delta_v_expr)

# H
functional = ...  - \
         delta_v * dx - \
         ....
```

Linearization?

Same approach but with $\frac{\mathrm{d}^2 v}{\mathrm{d}\{\boldsymbol{T},\boldsymbol{M}\}^2}$

Does it work?

Yes

Does it work?

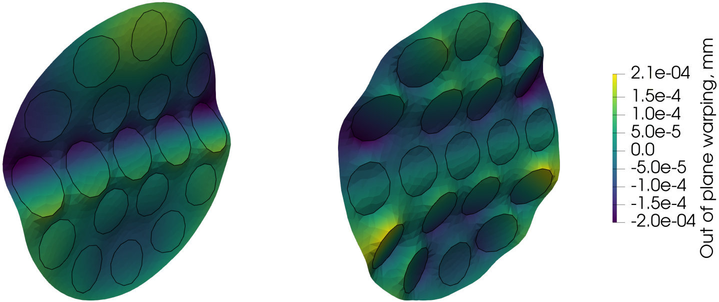
Yes

- quadratic convergence

Does it work?

Yes

- quadratic convergence
- nice results, also with elasto-plastic & hyperelastic materials



Morandini M, A two-level nonlinear beam analysis method, IJSS, 2020

Issue #1: caching

- $\{\boldsymbol{T}, \boldsymbol{M}\}$ piece-wise constant discontinuous (DG0)

Issue #1: caching

- $\{\boldsymbol{T}, \boldsymbol{M}\}$ piece-wise constant discontinuous (DG0)
- repeated calls, for the same cell, with same $\{\boldsymbol{T}, \boldsymbol{M}\}$

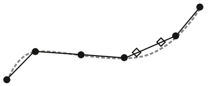
Issue #1: caching

- $\{\boldsymbol{T}, \boldsymbol{M}\}$ piece-wise constant discontinuous (DG0)
- repeated calls, for the same cell, with same $\{\boldsymbol{T}, \boldsymbol{M}\}$
- (limited) caching $\rightarrow$ significant speedup

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

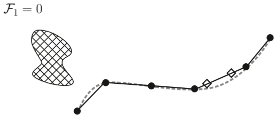
Early in the morning



Issue #2: a tale of coffee mugs

Python Dolfin wrapper

...

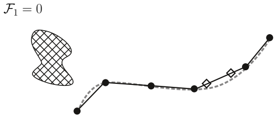


... 20 min ...

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

...

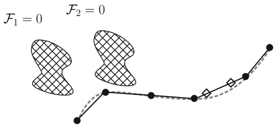


... 20 min ...

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

...

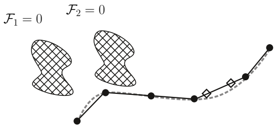


... 20 min ...

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

...

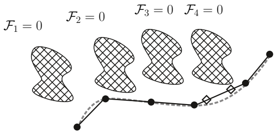


... 20 min ...

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

...

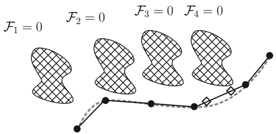


... n x 20 min ...

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

... late at night

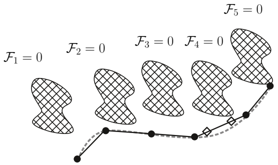


... n x 20 min ...

Issue #2: a tale of coffee mugs

Python Dolfin wrapper

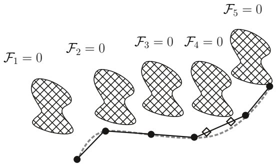
... late at night



Issue #2: a tale of coffee mugs

Python Dolfin wrapper

... late at night



Issue #2: a tale of coffee mugs

Why?

Issue #2: a tale of coffee mugs

Why?

- dijitso caches FFC code generation & compilation

Issue #2: a tale of coffee mugs

Why?

- dijitso caches FFC code generation & compilation
- caching based of Form “signature”

Issue #2: a tale of coffee mugs

Why?

- dijitso caches FFC code generation & compilation
- caching based of Form "signature"
- Form.signature() really expensive

Issue #2: a tale of coffee mugs

Why?

- dijitso caches FFC code generation & compilation
- caching based of Form “signature”
- Form.signature() really expensive
- Form.signature() gets recomputed!

Issue #2: a tale of coffee mugs

Why?

- dijitso caches FFC code generation & compilation
- caching based of Form "signature"
- Form.signature() really expensive
- Form.signature() gets recomputed!

Stop-gap solution:

Issue #2: a tale of coffee mugs

Why?

- dijitso caches FFC code generation & compilation
- caching based of Form "signature"
- Form.signature() really expensive
- Form.signature() gets recomputed!

Stop-gap solution:

replace Form.signature()

hash(.py source file) + form name

Questions/comments?