Hyperelasticity - Material Law and FE-Discretization

Finite Element Method [304.007]

19.05.2022

Contents

1 Hyperelastic Material Law

The simplest hyperelastic material is the Saint-Venant-Kirchhoff material. It generalizes the linear elastic isotropic material (Hooke's law) for nonlinear kinematics. For this material, elastic potential, PK2 (second Piola-Kirchhoff stress tensor) and material elasticity tensor read

$$\Psi(\boldsymbol{E}) = \frac{1}{2}\left(\lambda\,\mathrm{tr}(\boldsymbol{E})^2 + 2\mu\,\mathrm{tr}(\boldsymbol{E}\boldsymbol{E})\right)$$

$$\boldsymbol{S} = \frac{\partial\Psi}{\partial\boldsymbol{E}}(\boldsymbol{E}) = \lambda\,\mathrm{tr}(\boldsymbol{E})\boldsymbol{I} + 2\mu\boldsymbol{E}$$

$$\mathbb{C} = \frac{\partial^2\Psi}{\partial\boldsymbol{E}\partial\boldsymbol{E}}(\boldsymbol{E}) = \lambda\boldsymbol{I}\otimes\boldsymbol{I} + 2\mu\mathbb{I}^{sym} \quad \text{or} \quad \mathbb{C}_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right)\,.$$

Exercise 1 Create (and modify if necessary) classes of the namespace `nsModel.nsMaterial` according to the UML Diagramm from Section 3.1.
Create the script `testStVenantKirchhoff` in order to test the interface (i.e. all public methods) of the class `nsModel.nsMaterial.HyperelasticStVenantKirchhoffMaterial`:

a) Create an object of type `HyperelasticStVenantKirchhoffMaterial` with $E = 210\,000\,\mathrm{N/mm^2}$ and $\nu = 0.3$.

b) Check if the material gives the correct PK2 tensor $\boldsymbol{S} = \boldsymbol{0}$ for the undeformed state $\boldsymbol{F} = \boldsymbol{I}$.

c) Check the material elasticity tensor for the undeformed state $\boldsymbol{F} = \boldsymbol{I}$:

```
>> reshape(C,4,4)

ans =

   1.0e+05 *

    2.8269         0         0    1.2115
         0    0.8077    0.8077         0
         0    0.8077    0.8077         0
    1.2115         0         0    2.8269
```

2 Implementation of Stiffness Matrix and Residuum

The nonlinear function we want to find the root of in continuum mechanics is the virtual work. It depends on the deformation χ (nonlinearly), and on the test function $\boldsymbol{v}$ (linearly). If we have no external forces, it reads

$$\begin{aligned} W(\chi, \boldsymbol{v}) &= \int_\omega \boldsymbol{\sigma}(\chi) : \nabla \boldsymbol{v} \, \mathrm{d}v \\ &= \int_\Omega \boldsymbol{S} : \delta \boldsymbol{E} \, \mathrm{d}V = \int_\Omega \boldsymbol{P} : \delta \boldsymbol{F} \, \mathrm{d}V = \int_\Omega \boldsymbol{F}\boldsymbol{S} : \delta \boldsymbol{F} \, \mathrm{d}V = 0 \,. \end{aligned} \tag{1}$$

Before we can discretize the problem with finite elements, we need to linearize this equation. We use the notation $D\,W(\chi, \boldsymbol{v})[\Delta \boldsymbol{u}] = \Delta W(\chi, \boldsymbol{v})$ for the directional derivative with respect to the displacement increment $\Delta \boldsymbol{u}$. The linearization of the virtual work reads

$$W(\chi + \Delta \boldsymbol{u}, \boldsymbol{v}) \approx W(\chi, \boldsymbol{v}) + \Delta W(\chi, \boldsymbol{v}) \overset{!}{=} 0 \tag{2}$$

with

$$\Delta W(\chi, \boldsymbol{v}) = \underbrace{\int_\Omega \Delta \boldsymbol{F}\boldsymbol{S} : \delta \boldsymbol{F} \, \mathrm{d}V}_{\text{(a)}} + \underbrace{\int_\Omega \boldsymbol{F} \Delta \boldsymbol{S} : \delta \boldsymbol{F} \, \mathrm{d}V}_{\text{(b)}} \,. \tag{3}$$

The first term(a) is called *inital stress component (I.S.C.)*, because the stress tensor of the current state appears in it. The second term (b) is called *constitutive component (C.C.)*, because it contains the material law (constitutive law) via

$$\Delta \boldsymbol{S} = \mathbb{C} : \Delta \boldsymbol{E} \,. \tag{4}$$

As discussed in the learning video of this unit, the discretization of the incremental deformation gradient reads

$$\begin{aligned} \Delta F_{ij} &= \Delta \hat{u}_{il} N_{l,k} J_{kj} \\ &= \Delta \hat{u}_{ml} \delta_{mi} N_{l,k} J_{kj} \,. \end{aligned} \tag{5}$$

Since we use the same discretization for shape functions and test functions, the discretization of the virtual deformation gradient reads

$$\delta F_{ij} \coloneqq v_{i,j} \quad \rightarrow \quad \delta_{mi} N_{l,k} J_{kj} \,. \tag{6}$$

Note that the discretization of the test functions introduces two new free indices, in this case m and l. If this seems odd to you, remember the discretization of the test functions in the one-dimensional case of a bar under axial load. There, the test functions are discretized by adding the free index j:

$$\begin{aligned} \int_0^L EAN^i_{,x} v_{,x} \,\mathrm{d}x \ \hat{u}_i \quad &\rightarrow \quad \int_0^L EAN^i_{,x}(x) N^j_{,x} \,\mathrm{d}x \hat{u}_i \\ v \quad &\rightarrow \quad N^j \end{aligned} \tag{7}$$

This new index j was responsible for creating a system of equations ($j = 1 \ldots n_{\text{nodes}}$) out of one single equation (the weak form without discretization). In the same sense, the indices m and l in Equation (6) are responsible for creating our system of equations. Since we operate in the two- or three-dimensional space, we need two indices instead of one - the first index is responsible for the dimensions ($m = 1 \ldots n_{\text{dimensions}}$), the second one for the nodes of our elements ($l = 1 \ldots n_{\text{nodes}}$).
Plugging in the discretizations (5) and (6) into Equation (3), we obtain the tangent- or stiffness matrix:

$$\begin{aligned} A &= A^{\text{I.S.C.}} + A^{\text{C.C.}} \\ A^{\text{I.S.C.}}_{ijkl} &= \int_{\Omega^{\text{ref}}} \delta_{ik} N_{l,m} J_{mn} S_{no} N_{j,p} J_{po} \det(J^{-1}) \,\mathrm{d}V \\ A^{\text{C.C.}}_{ijkl} &= \int_{\Omega^{\text{ref}}} \mathbb{C}_{mnop} F_{im} N_{l,q} J_{qo} F_{kp} N_{j,r} J_{rn} \det(J^{-1}) \,\mathrm{d}V \end{aligned} \tag{8}$$

Now it's clear why we care about the names of the indices - we need to sum up the two components of the stiffness matrix. This only works if they have the same indices. You find details of this derivation in the learing video in the TeachCenter.
We still need to calculate the residuum, i.e. the function we want to find the root of. In our case, this is the virtual work from Equation (1). Plugging in the discretazion of the virtual displacement gradient from Equation (6), we obtain the residuum vector:

$$\begin{aligned} W &= \int_{\Omega} \boldsymbol{F}\boldsymbol{S} : \delta \boldsymbol{F} \,\mathrm{d}V \\ F^{\text{int}}_{ij} &= \int_{\Omega^{\text{ref}}} F_{ik} S_{kl} N_{j,m} J_{ml} \det(J^{-1}) \,\mathrm{d}V \end{aligned} \tag{9}$$

This term is called *internal force* F^{int}, because it's the residuum of the internal virtual work and has the dimension of a force.
If we considered external forces (surface or body forces), we would have to consider them in the force vector *and* in the stiffness matrix. We will see that in unit 6, when we consider pressure boundary conditions.
As in the linear case, we use the Matlab intern function `reshape` to transform the stiffness matrix into a two-dimensional array A_{ij} and the residuum vector into an one dimensional array F_i^{int}. The linear system to be solved in each step of the Newton-Raphson method then reads

$$\begin{aligned} D\,W[\Delta\boldsymbol{u}] &= -W \\ \implies A_{ij}\Delta\hat{u}_j &= -F_i^{\text{int}}\,. \end{aligned} \tag{10}$$

In the next unit, we will implement the Newton-Raphson method and the Dirichlet boundary conditions.

Exercise 2 You find the methods of the nonlinear element implementation (`nsAnalyzer.nsImplementation.NonlinearElementImpl`) in the UML diagram in Section 3.2.

a) Derive $A_{ijkl}^{\text{C.C.}}$ in Equation (8) and F_{ij}^{int} in Equation (9). Keep in mind that there are multiple correct solutions for $A_{ijkl}^{\text{C.C.}}$ due to the three symmetries of the elasticity tensor.

b) You find the class `nsAnalyzer.nsImplementation.NonlinearElementImpl` in the TeachCenter. Copy it to the correct location of your program folder `soofeam`. Complete the methods `constitutiveComponentIntegrator` and `internalForcesIntegrator`.

Last but not least we need to test the class. In the TeachCenter you find the script `testNonlinearElementImpl.m`. It creates a quad element and calculates stiffness matrix and residuum vector.
Check if your implementation leads to the following results:

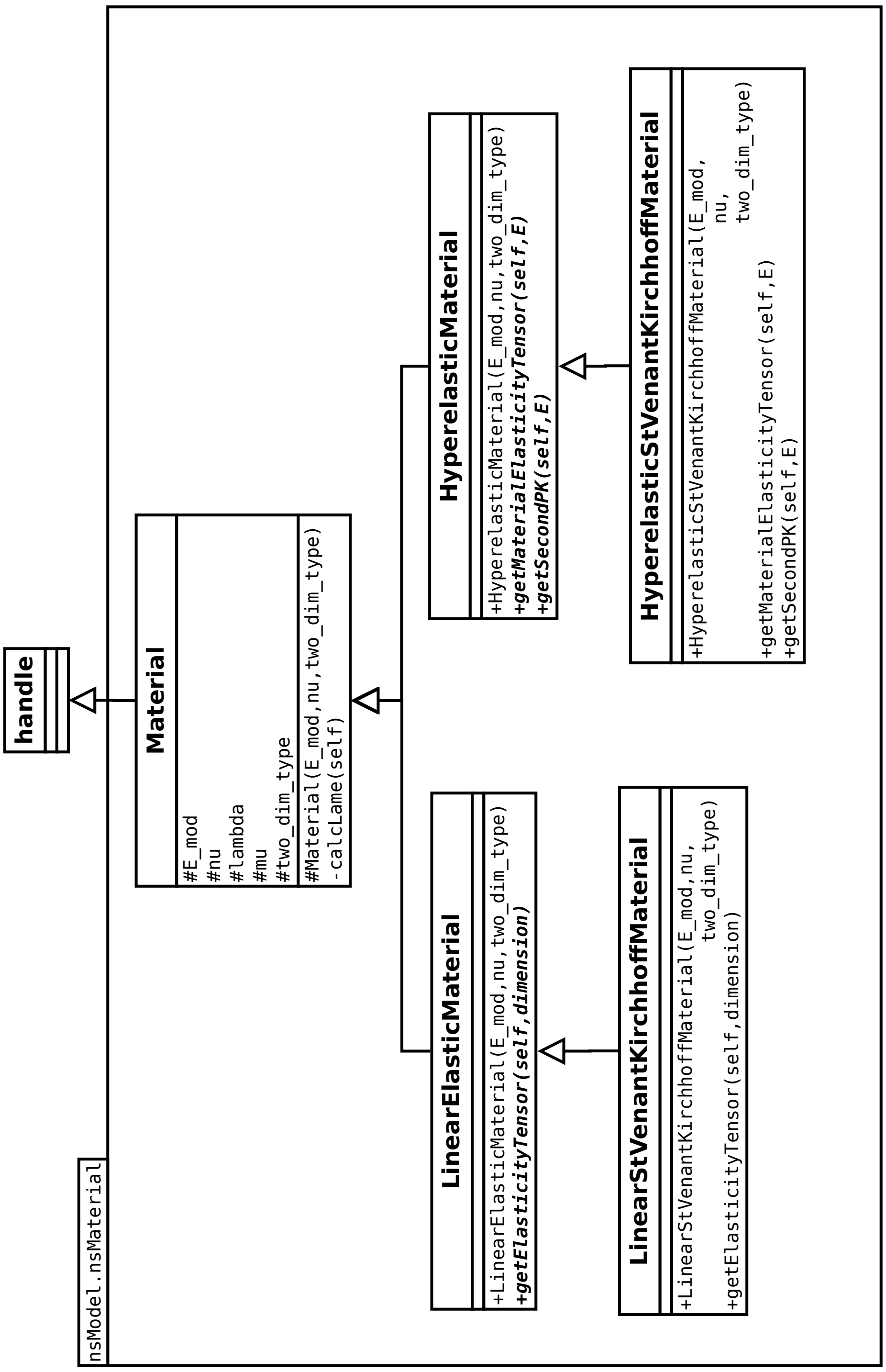
```
A_ISC =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0




```

```

A_CC =

   1.0e+05 *

    1.4481    0.3059   −0.5303    0.0245   −0.4120   −0.4283   −0.5058    0.0979
    0.3059    1.9478   −0.1774    0.4385   −0.4283   −0.7078    0.2998   −1.6786
   −0.5303   −0.1774    1.1422   −0.6119    0.2713    0.2080   −0.8832    0.5813
    0.0245    0.4385   −0.6119    1.1524    0.0061   −0.7485    0.5813   −0.8424
   −0.4120   −0.4283    0.2713    0.0061    0.6710    0.3977   −0.5303    0.0245
   −0.4283   −0.7078    0.2080   −0.7485    0.3977    1.0178   −0.1774    0.4385
   −0.5058    0.2998   −0.8832    0.5813   −0.5303   −0.1774    1.9193   −0.7037
    0.0979   −1.6786    0.5813   −0.8424    0.0245    0.4385   −0.7037    2.0825


F_int =

     0
     0
     0
     0
     0
     0
     0
     0
```

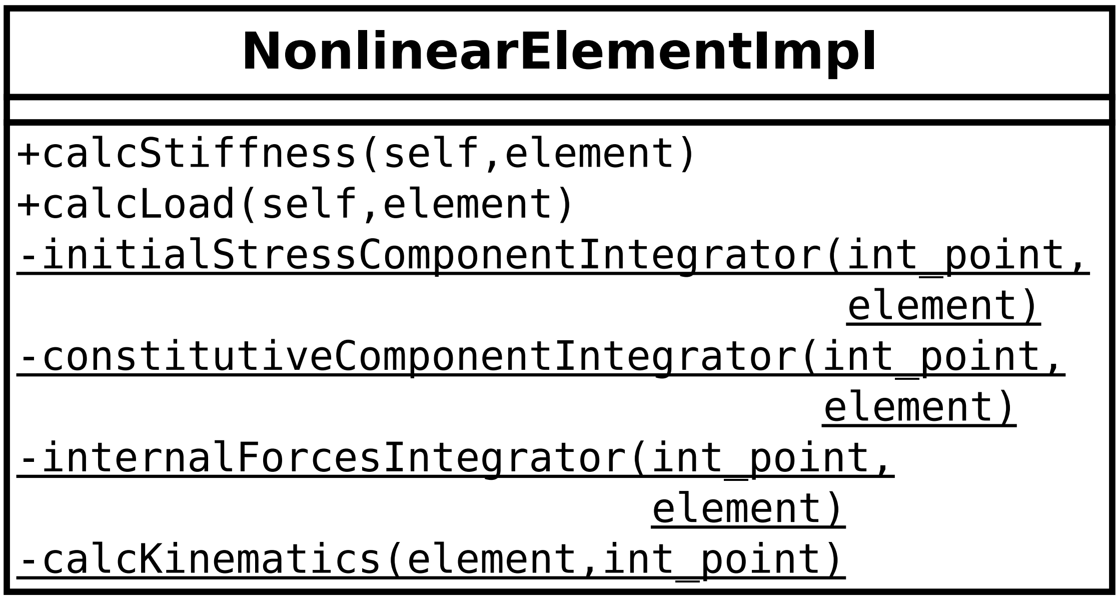
Use the debugger to view $A^{\text{I.S.C.}}$ and $A^{\text{C.C.}}$ seperately.
Why are $A^{\text{I.S.C.}}$ and F^{int} zero (or almost zero, depending on your computer and Matlab version)?

3 Diagrams

3.1 UML diagram for nsModel.nsMaterial


nsModel.nsMaterial
handle
Material
#E_mod
#nu
#lambda
#mu
#two_dim_type
#Material(E_mod,nu,two_dim_type)
-calcLame(self)
LinearElasticMaterial
+LinearElasticMaterial(E_mod,nu,two_dim_type)
+getElasticityTensor(self,dimension)
HyperelasticMaterial
+HyperelasticMaterial(E_mod,nu,two_dim_type)
+getMaterialElasticityTensor(self,E)
+getSecondPK(self,E)
LinearStVenantKirchhoffMaterial
+LinearStVenantKirchhoffMaterial(E_mod,nu, two_dim_type)
+getElasticityTensor(self,dimension)
HyperelasticStVenantKirchhoffMaterial
+HyperelasticStVenantKirchhoffMaterial(E_mod, nu, two_dim_type)
+getMaterialElasticityTensor(self,E)
+getSecondPK(self,E)

3.2 UML diagram for nsAnalyzer.nsImplementation.NonlinearElementImpl



```
NonlinearElementImpl

+calcStiffness(self,element)
+calcLoad(self,element)
-initialStressComponentIntegrator(int_point,
                                  element)
-constitutiveComponentIntegrator(int_point,
                                 element)
-internalForcesIntegrator(int_point,
                          element)
-calcKinematics(element,int_point)
```