Hyperelastic

Andreas Dutzler

Feb 13, 2026

CONTENTS:

Constitutive hyperelastic material formulations for https://github.com/adtzlr/felupe .

This package provides the essential building blocks for constitutive hyperelastic material formulations. This includes material behaviour-independent *Spaces* and *Frameworks* as well as material behaviour-dependent *Models*.

CHAPTER

ONE

TUTORIALS

1.1 Getting Started

Spaces are full or partial deformations on which a given material formulation should be projected to, e.g. to the distortional (part of the deformation) space. Generalized *Total-Lagrange Frameworks* for isotropic hyperelastic material formulations based on the invariants of the right Cauchy-Green deformation tensor and the principal stretches enable a clean coding of isotropic material formulations.

The *Math*-module provides helpers in reduced vector Voigt storage for symmetric three-dimensional second-order tensors along with a matrix storage for (at least minor) symmetric three-dimensional fourth-order tensors.

Material *model formulations* have to be created as classes with methods for the evaluation of the *gradient* (stress) and the *hessian* (elasticity) of the strain energy function. It depends on the *Framework* which derivatives have to be defined, e.g. the derivatives w.r.t. the invariants of the right Cauchy-Green deformation tensor or w.r.t. the principal stretches. An instance of a *Framework* has to be *finalized* by the application on a *Space*.

First, let's import hyperelastic.

```
import hyperelastic
```

1.1.1 Available material formulations

Model Formulation	Framework	Parameters
Third Order Deformation	*Invariants*	$C_{10}, C_{01}, C_{11}, C_{20}, C_{30}$
Torch Autograd Hyperelastic	*Invariants*	$\psi(I_1, I_2, I_3)$
Ogden	*Stretches*	μ, α

An instance of one of these models is then embedded into the corresponding *Framework*, which is further applied onto a *Space*.

```
model = hyperelastic.models.invariants.ThirdOrderDeformation(C10=0.5)
framework = hyperelastic.InvariantsFramework(model)
umat = hyperelastic.DistortionalSpace(framework)
```

Note

Instead of using the implemented models, define your own material model formulation with manual, automatic or symbolic differentiation with the help of your favourite package, e.g. PyTorch, JAX, Tensorflow, TensorTRAX, SymPy, etc.

1.1.2 Invariant-based material formulations

A minimal template for an invariant-based material formulation applied on the distortional space:

```
class MyInvariantsModel:
    def gradient(self, I1, I2, I3, statevars):
        """The gradient as the partial derivative of the strain energy function w.r.t.
        the invariants of the right Cauchy-Green deformation tensor."""

        # user code
        dWdI1 = None
        dWdI2 = None
        dWdI3 = None

        return dWdI1, dWdI2, dWdI3, statevars

    def hessian(self, I1, I2, I3, statevars_old):
        """The hessian as the second partial derivatives of the strain energy function
        w.r.t. the invariants of the right Cauchy-Green deformation tensor."""

        # user code
        d2WdI1I1 = None
        d2WdI2I2 = None
        d2WdI3I3 = None
        d2WdI1I2 = None
        d2WdI2I3 = None
        d2WdI1I3 = None

        return d2WdI1I1, d2WdI2I2, d2WdI3I3, d2WdI1I2, d2WdI2I3, d2WdI1I3

model = MyInvariantsModel()
framework = hyperelastic.InvariantsFramework(model)
umat = hyperelastic.DistortionalSpace(framework)
```

1.1.3 Principal stretch-based material formulations

A minimal template for a principal stretch-based material formulation applied on the distortional space:

```
class MyStretchesModel:
    def gradient(self, , statevars):
        """The gradient as the partial derivative of the strain energy function w.r.t.
        the principal stretches."""

        # user code
        dWd1, dWd2, dWd3 = 0 *

        return [dWd1, dWd2, dWd3], statevars

    def hessian(self, , statevars_old):
        """The hessian as the second partial derivatives of the strain energy function
        w.r.t. the principal stretches."""

        # user code
        d2Wd1d1 = None
```

(continues on next page)

(continued from previous page)

```
        d2Wd2d2 = None
        d2Wd3d3 = None
        d2Wd1d2 = None
        d2Wd2d3 = None
        d2Wd1d3 = None

        return d2Wd1d1, d2Wd2d2, d2Wd3d3, d2Wd1d2, d2Wd2d3, d2Wd1d3


model = MyStretchesModel()
framework = hyperelastic.StretchesFramework(model)
umat = hyperelastic.DistortionalSpace(framework)
```

1.1.4 Lab

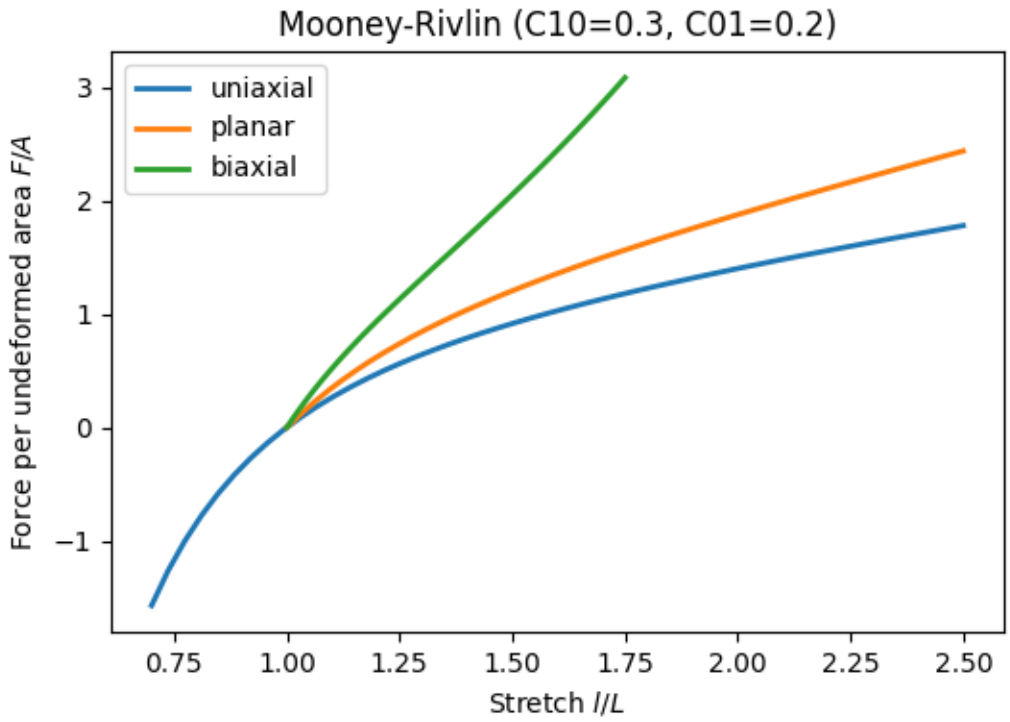
In the `Lab`, `Simulations` on homogeneous load cases provide a visualization of the material response behaviour.

```
import numpy as np
import hyperelastic

stretch = np.linspace(0.7, 2.5, 181)
parameters = {"C10": 0.3, "C01": 0.2}

def material(C10, C01):
    tod = hyperelastic.models.invariants.ThirdOrderDeformation(C10=C10, C01=C01)
    framework = hyperelastic.InvariantsFramework(tod)
    return hyperelastic.DeformationSpace(framework)

ux = hyperelastic.lab.Simulation(
    loadcase=hyperelastic.lab.Uniaxial(label="uniaxial"),
    stretch=np.linspace(0.7, 2.5),
    material=material,
    labels=parameters.keys(),
    parameters=parameters.values(),
)

ps = hyperelastic.lab.Simulation(
    loadcase=hyperelastic.lab.Planar(label="planar"),
    stretch=np.linspace(1.0, 2.5),
    material=material,
    labels=parameters.keys(),
    parameters=parameters.values(),
)

bx = hyperelastic.lab.Simulation(
    loadcase=hyperelastic.lab.Biaxial(label="biaxial"),
    stretch=np.linspace(1.0, 1.75),
    material=material,
    labels=parameters.keys(),
    parameters=parameters.values(),
)
```

(continues on next page)

(continued from previous page)

```
fig, ax = ux.plot_stress_stretch(lw=2)
fig, ax = ps.plot_stress_stretch(ax=ax, lw=2)
fig, ax = bx.plot_stress_stretch(ax=ax, lw=2)

ax.legend()
ax.set_title(rf"Mooney-Rivlin (C10={parameters['C10']}, C01={parameters['C01']})")
```

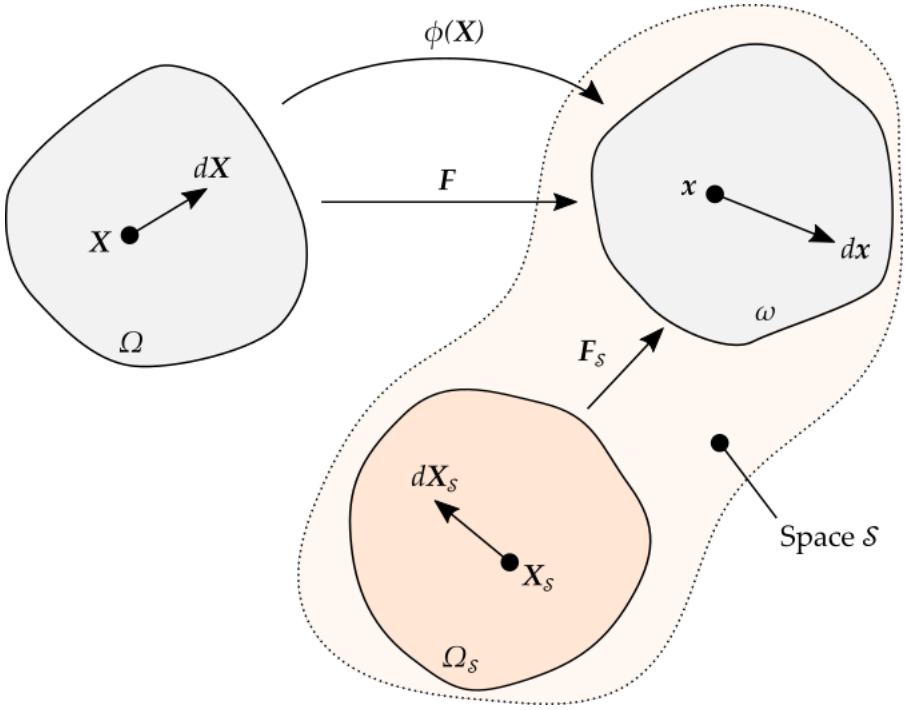


CHAPTER

TWO

SPACES

Spaces are full or partial deformations on which a given material formulation may be projected to, e.g. to the distortional (part of the deformation) space.



class hyperelastic.spaces.**Deformation**(*material*, *parallel=False*, *finalize=True*, *force=None*, *area=0*)

The deformation space.

This class takes a Total-Lagrange material formulation and applies it on the deformation space.

$$\psi = \psi(\boldsymbol{C}(\boldsymbol{F})) \tag{2.1}$$

The gradient of the strain energy function is carried out w.r.t. the Green Lagrange strain tensor. Hence, the work-conjugate stress tensor here refers to the second Piola-Kirchhoff stress tensor.

$$\boldsymbol{S} = \frac{\partial \psi}{\partial \frac{1}{2}\boldsymbol{C}} \tag{2.2}$$

The hessian of the strain energy function is carried out w.r.t. the Green-Lagrange strain tensor. Hence, the work-conjugate elasticity tensor here refers to the fourth-order Total-Lagrangian elasticity tensor.

$$\mathbb{C} = \frac{\partial^2 \psi}{\partial \frac{1}{2}\boldsymbol{C}\, \frac{1}{2}\boldsymbol{C}} \tag{2.3}$$

Given a Total-Lagrange material formulation, for the variation and linearization of the virtual work of internal forces, the output quantities have to be transformed: The second Piola-Kirchhoff stress tensor is converted into the deformation gradient work-conjugate first Piola-Kirchhoff stress tensor, along with its fourth-order elasticity tensor. Also, the so-called geometric tangent stiffness component (initial stress matrix) is added to the fourth-order elasticity tensor.

$$\begin{aligned} \delta W_{int} &= -\int_V \boldsymbol{P} : \delta \boldsymbol{F} \, dV \\ \Delta\delta W_{int} &= -\int_V \delta \boldsymbol{F} : \mathbb{A} : \Delta \boldsymbol{F} \, dV \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{F}\boldsymbol{S} \\ \mathbb{A}_{iJkL} &= F_{iI} F_{kK} \mathbb{C}_{IJKL} + \delta_{ik} S_{JL} \end{aligned} \tag{2.5}$$

gradient(*x*)

The gradient as the partial derivative of the strain energy function w.r.t. the deformation gradient.

hessian(*x*)

The hessian as the second partial derivative of the strain energy function w.r.t. the deformation gradient.

piola(*F*, *S*, *detF=None*, *C4=None*, *invC=None*)

Convert the Total-Lagrange stress or elasticity tensor to the chosen configurations for the differential force and area vectors by applying a Piola-transformation.

class hyperelastic.spaces.**Dilatational**(*material*, *parallel=False*, *finalize=True*, *force=None*, *area=0*)

gradient(*x*)

hessian(*x*)

piola(*F*, *S*, *detF=None*, *C4=None*, *invC=None*)

Convert the Total-Lagrange stress or elasticity tensor to the chosen configurations for the differential force and area vectors by applying a Piola-transformation.

class hyperelastic.spaces.**Distortional**(*material*, *parallel=False*, *finalize=True*, *force=None*, *area=0*)

The distortional (part of the deformation) space is a partial deformation with constant volume. For a given deformation map $\boldsymbol{x}(\boldsymbol{X})$ and its deformation gradient $\boldsymbol{F}$, the distortional part of the deformation gradient $\hat{\boldsymbol{F}}$ is obtained by a multiplicative (consecutive) split into a volume-changing (dilatational) and a constant-volume (distortional) part of the deformation gradient. Due to the fact that the dilatational part is proportional to the unit tensor, the order of these partial deformations is not unique.

$$\boldsymbol{F} = \overset{\circ}{\boldsymbol{F}}\hat{\boldsymbol{F}} = \hat{\boldsymbol{F}}\overset{\circ}{\boldsymbol{F}} \tag{2.6}$$

This class takes a Total-Lagrange material formulation and applies it only on the distortional space.

$$\hat{\psi} = \psi(\hat{\boldsymbol{C}}(\boldsymbol{F})) \tag{2.7}$$

The distortional (unimodular) part of the right Cauchy-Green deformation tensor is evaluated by the help of its third invariant (the determinant). The determinant of a distortional (an unimodular) tensor equals to one.

$$\hat{\boldsymbol{C}} = I_3^{-1/3}\boldsymbol{C} \tag{2.8}$$

The gradient of the strain energy function is carried out w.r.t. the Green Lagrange strain tensor. Hence, the work-conjugate stress tensor used in this space projection refers to the second Piola-Kirchhoff stress tensor.

$$\boldsymbol{S}' = \frac{\partial \hat{\psi}}{\partial \frac{1}{2}\boldsymbol{C}} \tag{2.9}$$

The distortional space projection leads to a **physically** deviatoric second Piola-Kirchhoff stress tensor, evaluated by the application of the chain rule.

$$\hat{\boldsymbol{S}} = \frac{\partial \hat{\psi}}{\partial \frac{1}{2}\hat{\boldsymbol{C}}} \tag{2.10}$$

The (**phyiscally**) deviatoric projection is obtained by the partial derivative of the distortional part of the right Cauchy-Green deformation tensor w.r.t. the right Cauchy-Green deformation tensor.

$$\frac{\partial \hat{\boldsymbol{C}}}{\partial \boldsymbol{C}} = \frac{\partial I_3^{-1/3}\boldsymbol{C}}{\partial \boldsymbol{C}} = I_3^{-1/3}\left(\mathbf{1} \odot \mathbf{1} - \frac{1}{3}\boldsymbol{C} \otimes \boldsymbol{C}^{-1}\right) \tag{2.11}$$

This partial derivative is used to perform the distortional space projection of the second Piola-Kirchhoff stress tensor. Instead of asserting the determinant-scaling to the fourth-order projection tensor, this factor is combined with the second Piola-Kirchhoff stress tensor in the distortional space. Hence, the stress tensor in the distortional space, scaled by $I_3^{-1/3}$, is introduced as a new (frequently re-used) variable, denoted by an overset bar.

$$\begin{aligned} \boldsymbol{S}' &= \mathbb{P} : \bar{\boldsymbol{S}} \\ \bar{\boldsymbol{S}} &= I_3^{-1/3}\hat{\boldsymbol{S}} \\ \mathbb{P} &= \mathbf{1} \odot \mathbf{1} - \frac{1}{3}\boldsymbol{C}^{-1} \otimes \boldsymbol{C} \end{aligned} \tag{2.12}$$

The evaluation of the double-dot product for the distortional space projection leads to the mathematical deviator of the product between the scaled distortional space stress tensor and the right Cauchy-Green deformation tensor, right multiplied by the inverse of the right Cauchy-Green deformation tensor.

$$\boldsymbol{S}' = \bar{\boldsymbol{S}} - \frac{\bar{\boldsymbol{S}} : \boldsymbol{C}}{3}\boldsymbol{C}^{-1} = \mathrm{dev}(\bar{\boldsymbol{S}}\boldsymbol{C})\boldsymbol{C}^{-1} \tag{2.13}$$

The hessian of the strain energy function is carried out w.r.t. the Green-Lagrange strain tensor. Hence, the work-conjugate elasticity tensor used in this space projection refers to the fourth-order Total-Lagrangian elasticity tensor.

$$\mathbb{C}' = \frac{\partial^2 \hat{\psi}}{\partial \frac{1}{2}\boldsymbol{C}\, \frac{1}{2}\boldsymbol{C}} \tag{2.14}$$

The evaluation of this second partial derivative leads to the elasticity tensor of the distortional space projection. The remaining determinant scaling terms of the projection tensor are included in the determinant-modified fourth-order elasticity tensor, denoted with an overset bar.

$$\mathbb{C}' = \mathbb{P} : \bar{\mathbb{C}} : \mathbb{P}^T + \frac{2}{3}\left(\left(\bar{\boldsymbol{S}} : \boldsymbol{C}\right)\boldsymbol{C}^{-1} \odot \boldsymbol{C}^{-1} - \bar{\boldsymbol{S}} \otimes \boldsymbol{C}^{-1} - \boldsymbol{C}^{-1} \otimes \bar{\boldsymbol{S}} + \frac{1}{3}\left(\bar{\boldsymbol{S}} : \boldsymbol{C}\right)\boldsymbol{C}^{-1} \otimes \boldsymbol{C}^{-1}\right) \tag{2.15}$$

$$\bar{\mathbb{C}} = I_3^{-2/3}\hat{\mathbb{C}} \tag{2.16}$$

For the variation and linearization of the virtual work of internal forces, the output quantities have to be transformed: The second Piola-Kirchhoff stress tensor is converted into the deformation gradient work-conjugate first Piola-Kirchhoff stress tensor, along with its fourth-order elasticity tensor. Also, the so-called geometric tangent stiffness component (initial stress matrix) is added to the fourth-order elasticity tensor.

$$\begin{aligned} \delta W_{int} &= -\int_V \boldsymbol{P} : \delta \boldsymbol{F}\, dV \\ \Delta\delta W_{int} &= -\int_V \delta \boldsymbol{F} : \mathbb{A} : \Delta \boldsymbol{F}\, dV \end{aligned} \tag{2.17}$$

where

$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{F}\boldsymbol{S} \\ \mathbb{A}_{iJkL} &= F_{iI}F_{kK}\mathbb{C}_{IJKL} + \delta_{ik}S_{JL} \end{aligned} \tag{2.18}$$

gradient(*x*)

The gradient as the partial derivative of the strain energy function w.r.t. the deformation gradient.

hessian(*x*)

The hessian as the second partial derivative of the strain energy function w.r.t. the deformation gradient.

piola(*F*, *S*, *detF=None*, *C4=None*, *invC=None*)

Convert the Total-Lagrange stress or elasticity tensor to the chosen configurations for the differential force and area vectors by applying a Piola-transformation.

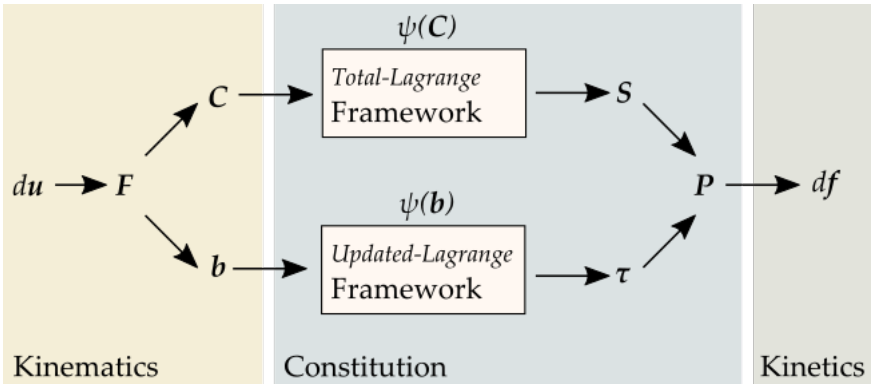
CHAPTER

THREE

FRAMEWORKS

The (topologic) deformation of a solid body is given by a displacement field. The deformation gradient, the right and left Cauchy-Green deformation tensors as well as related strain tensors are displacement- or deformation-derived kinematic quantities. The kinetics, i.e. the force vector acting on a solid body, is evaluated by the stress tensor and the appropriate area normal vector. The constitutive material model is formulated within a framework which links work-conjugate quantities of stress and strain. The stress tensor of the framework has to be transformed to be consistent with the force and area normal vectors. Here, the first Piola-Kirchhoff stress tensor is used to evaluate the force vector in the deformed configuration by the area normal vector of the undeformed configuration.

Generalized *Total-Lagrange* `Frameworks` for isotropic hyperelastic material formulations based on the invariants of the right Cauchy-Green deformation tensor and the principal stretches enable a clean coding of isotropic material formulations.



class hyperelastic.frameworks.**GeneralizedInvariants**(*material*, *fun*, *nstatevars=0*, *parallel=False*, ***kwargs*)

Generalized-invariants isotropic hyperelastic material formulation based on the principal stretches.

$$\psi = \psi\left(I_1\left(E_1, E_2, E_3\right), I_2\left(E_1, E_2, E_3\right), I_3\left(E_1, E_2, E_3\right)\right) \tag{3.1}$$

The three principal invariants

$$\begin{aligned} J_1 &= E_1 + E_2 + E_3 \\ J_2 &= E_1E_2 + E_2E_3 + E_1E_3 \\ J_3 &= E_1E_2E_3 \end{aligned} \tag{3.2}$$

are formulated on a one-dimensional strain-stretch relation.

$$\begin{aligned} E_\alpha &= f(\lambda_\alpha) \\ E'_\alpha &= f'(\lambda_\alpha) = \frac{\partial f(\lambda_\alpha)}{\partial \lambda_\alpha} \\ E''_\alpha &= f''(\lambda_\alpha) = \frac{\partial^2 f(\lambda_\alpha)}{\partial \lambda_\alpha \, \partial \lambda_\alpha} \end{aligned} \tag{3.3}$$

Depending on the strain-stretch relation, the invariants contain deformation- independent values.

$$\begin{aligned} J_{1,0} &= J_1(E_\alpha(\lambda_\alpha = 1)) \\ J_{2,0} &= J_2(E_\alpha(\lambda_\alpha = 1)) \\ J_{3,0} &= J_3(E_\alpha(\lambda_\alpha = 1)) \end{aligned} \tag{3.4}$$

The deformation-dependent parts of the invariants are scaled by deformation- independent coefficients of normalization. The deformation-independent parts are re-added after the scaling.

$$\begin{aligned} I_1 &= c_1(J_1 - J_{1,0}) + J_{1,0} \\ I_2 &= c_2(J_2 - J_{2,0}) + J_{2,0} \\ I_3 &= J_3 \end{aligned} \tag{3.5}$$

Note that the scaling is only applied to the first and second invariant, as the third invariant does not contribute to the strain energy function at the undeformed state.

$$\begin{aligned} E_0 &= E(\lambda = 1) \\ E_0' &= E'(\lambda = 1) \\ E_0'' &= E''(\lambda = 1) \end{aligned} \tag{3.6}$$

The second partial derivative of the strain w.r.t. the stretch must be provided for a reference strain, e.g. the Green-Lagrange strain measure (at the undeformed state).

$$\begin{aligned} J_{1,0}'' &= \frac{3}{2}\left(E_0'' + E_0'\right) \\ J_{2,0}'' &= \frac{3}{2}\left((2E_0(E_0'' + E_0')) - E_0'^2\right) \end{aligned} \tag{3.7}$$

$$\begin{aligned} c_1 &= \frac{J_{1,0,ref}''}{J_{1,0}''} \\ c_2 &= \frac{J_{2,0,ref}''}{J_{2,0}''} \end{aligned} \tag{3.8}$$

The first partial derivatives of the strain energy function w.r.t. the invariants

$$\begin{aligned} \psi_{,1} &= \frac{\partial \psi}{\partial I_1} \\ \psi_{,2} &= \frac{\partial \psi}{\partial I_2} \\ \psi_{,3} &= \frac{\partial \psi}{\partial I_3} \end{aligned} \tag{3.9}$$

and the partial derivatives of the invariants w.r.t. the principal stretches are defined. From here on, this is consistent with any invariant-based hyperelastic material formulation, except for the factors of normalization.

$$\begin{aligned} \frac{\partial I_1}{\partial E_\alpha} &= c_1 \\ \frac{\partial I_2}{\partial E_\alpha} &= c_2\left(E_\beta + E_\gamma\right) \\ \frac{\partial I_3}{\partial E_\alpha} &= E_\beta E_\gamma \end{aligned} \tag{3.10}$$

The first partial derivatives of the strain energy density w.r.t. the principal stretches are required for the principal values of the stress.

$$\frac{\partial \psi}{\partial \lambda_\alpha} = \frac{\partial \psi}{\partial I_1}\frac{\partial I_1}{\partial E_\alpha}\frac{\partial E_\alpha}{\partial \lambda_\alpha} + \frac{\partial \psi}{\partial I_2}\frac{\partial I_2}{\partial E_\alpha}\frac{\partial E_\alpha}{\partial \lambda_\alpha} + \frac{\partial \psi}{\partial I_3}\frac{\partial I_3}{\partial E_\alpha}\frac{\partial E_\alpha}{\partial \lambda_\alpha} \tag{3.11}$$

Furthermore, the second partial derivatives of the strain energy density w.r.t. the principal stretches, necessary for the principal components of the elastic tangent moduli, are carried out. This is done in two steps: first, the second partial derivatives w.r.t. the principal strain components are carried out, followed by the projection to the derivatives w.r.t. the principal stretches.

$$\begin{aligned}\frac{\partial^2\psi}{\partial E_\alpha\,\partial E_\beta} &= \frac{\partial^2\psi}{\partial I_1\,\partial I_1}\frac{\partial I_1}{\partial E_\alpha}\frac{\partial I_1}{\partial E_\beta} + \frac{\partial^2\psi}{\partial I_2\,\partial I_2}\frac{\partial I_2}{\partial E_\alpha}\frac{\partial I_2}{\partial E_\beta} + \frac{\partial^2\psi}{\partial I_3\,\partial I_3}\frac{\partial I_3}{\partial E_\alpha}\frac{\partial I_3}{\partial E_\beta} \\ &+ \frac{\partial^2\psi}{\partial I_1\,\partial I_2}\left(\frac{\partial I_1}{\partial E_\alpha}\frac{\partial I_2}{\partial E_\beta} + \frac{\partial I_2}{\partial E_\alpha}\frac{\partial I_1}{\partial E_\beta}\right) \\ &+ \frac{\partial^2\psi}{\partial I_2\,\partial I_3}\left(\frac{\partial I_2}{\partial E_\alpha}\frac{\partial I_3}{\partial E_\beta} + \frac{\partial I_3}{\partial E_\alpha}\frac{\partial I_2}{\partial E_\beta}\right) \\ &+ \frac{\partial^2\psi}{\partial I_1\,\partial I_3}\left(\frac{\partial I_1}{\partial E_\alpha}\frac{\partial I_3}{\partial E_\beta} + \frac{\partial I_3}{\partial E_\alpha}\frac{\partial I_1}{\partial E_\beta}\right) \\ &+ \frac{\partial\psi}{\partial I_1}\frac{\partial^2 I_1}{\partial E_\alpha\,\partial E_\beta} + \frac{\partial\psi}{\partial I_2}\frac{\partial^2 I_1}{\partial E_\alpha\,\partial E_\beta} + \frac{\partial\psi}{\partial I_3}\frac{\partial^2 I_1}{\partial E_\alpha\,\partial E_\beta}\end{aligned} \tag{3.12}$$

$$\frac{\partial^2\psi}{\partial\lambda_\alpha\,\partial\lambda_\beta} = \frac{\partial E_\alpha}{\partial\lambda_\alpha}\frac{\partial^2\psi}{\partial E_\alpha\,\partial E_\beta}\frac{\partial E_\beta}{\partial\lambda_\beta} + \left(\frac{\partial\psi}{\partial I_1}\frac{\partial I_1}{\partial E_\alpha} + \frac{\partial\psi}{\partial I_2}\frac{\partial I_2}{\partial E_\alpha} + \frac{\partial\psi}{\partial I_3}\frac{\partial I_3}{\partial E_\alpha}\right)\frac{\partial^2 E_\alpha}{\partial\lambda_\alpha\partial\lambda_\alpha} \tag{3.13}$$

gradient(*C*, *statevars*)

The gradient as the partial derivative of the strain energy function w.r.t. the right Cauchy-Green deformation tensor (one half of the second Piola Kirchhoff stress tensor).

hessian(*C*, *statevars*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the right Cauchy-Green deformation tensor (a quarter of the Lagrangian fourth-order elasticity tensor associated to the second Piola-Kirchhoff stress tensor).

class hyperelastic.frameworks.**Invariants**(*material*, *nstatevars=0*, *parallel=False*)

The Framework for a Total-Lagrangian invariant-based isotropic hyperelastic material formulation provides the material behaviour-independent parts for evaluating the second Piola-Kirchhoff stress tensor as well as its associated fourth-order elasticity tensor.

The gradient as well as the hessian of the strain energy function are carried out w.r.t. the right Cauchy-Green deformation tensor. Hence, the work-conjugate stress tensor is one half of the second Piola-Kirchhoff stress tensor and the fourth-order elasticitiy tensor used here is a quarter of the Total-Lagrangian elasticity tensor.

$$\psi(\boldsymbol{C}) = \psi(I_1(\boldsymbol{C}), I_2(\boldsymbol{C}), I_3(\boldsymbol{C})) \tag{3.14}$$

The first and second invariants of the left or right Cauchy-Green deformation tensor are identified as factors of their characteristic polynomial,

$$\begin{aligned}I_1 &= \mathrm{tr}(\boldsymbol{C}) \\ I_2 &= \frac{1}{2}\left(\mathrm{tr}(\boldsymbol{C})^2 - \mathrm{tr}(\boldsymbol{C}^2)\right) \\ I_3 &= \det(\boldsymbol{C})\end{aligned} \tag{3.15}$$

where the Cauchy-Green deformation tensors eliminate the rigid body rotations of the deformation gradient and serve as a quadratic change-of-length measure of the deformation.

$$\begin{aligned}\boldsymbol{C} &= \boldsymbol{F}^T\boldsymbol{F} \\ \boldsymbol{b} &= \boldsymbol{F}\boldsymbol{F}^T\end{aligned} \tag{3.16}$$

The first partial derivatives of the strain energy function w.r.t. the invariants

$$\begin{aligned}\psi_{,1} &= \frac{\partial \psi}{\partial I_1} \\ \psi_{,2} &= \frac{\partial \psi}{\partial I_2} \\ \psi_{,3} &= \frac{\partial \psi}{\partial I_3}\end{aligned} \tag{3.17}$$

and the partial derivatives of the invariants w.r.t. the right Cauchy-Green deformation tensor are defined.

$$\begin{aligned}\frac{\partial I_1}{\partial \boldsymbol{C}} &= \boldsymbol{I} \\ \frac{\partial I_2}{\partial \boldsymbol{C}} &= (I_1 \boldsymbol{I} - \boldsymbol{C}) \\ \frac{\partial I_3}{\partial \boldsymbol{C}} &= I_3 \boldsymbol{C}^{-1}\end{aligned} \tag{3.18}$$

The second Piola-Kirchhoff stress tensor is formulated by the application of the chain rule.

$$\frac{\partial \psi}{\partial \boldsymbol{C}} = \frac{\partial \psi}{\partial I_1}\frac{\partial I_1}{\partial \boldsymbol{C}} + \frac{\partial \psi}{\partial I_2}\frac{\partial I_2}{\partial \boldsymbol{C}} + \frac{\partial \psi}{\partial I_3}\frac{\partial I_3}{\partial \boldsymbol{C}} \tag{3.19}$$

Furthermore, the second partial derivatives of the elasticity tensor are carried out.

$$\begin{aligned}\frac{\partial^2 \psi}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}} = &\frac{\partial^2 \psi}{\partial I_1\,\partial I_1}\left(\frac{\partial I_1}{\partial \boldsymbol{C}} \otimes \frac{\partial I_1}{\partial \boldsymbol{C}}\right) \\ &+ \frac{\partial^2 \psi}{\partial I_2\,\partial I_2}\left(\frac{\partial I_2}{\partial \boldsymbol{C}} \otimes \frac{\partial I_2}{\partial \boldsymbol{C}}\right) \\ &+ \frac{\partial^2 \psi}{\partial I_3\,\partial I_3}\left(\frac{\partial I_3}{\partial \boldsymbol{C}} \otimes \frac{\partial I_3}{\partial \boldsymbol{C}}\right) \\ &+ \frac{\partial^2 \psi}{\partial I_1\,\partial I_2}\left(\frac{\partial I_1}{\partial \boldsymbol{C}} \otimes \frac{\partial I_2}{\partial \boldsymbol{C}} + \frac{\partial I_2}{\partial \boldsymbol{C}} \otimes \frac{\partial I_1}{\partial \boldsymbol{C}}\right) \\ &+ \frac{\partial^2 \psi}{\partial I_2\,\partial I_3}\left(\frac{\partial I_2}{\partial \boldsymbol{C}} \otimes \frac{\partial I_3}{\partial \boldsymbol{C}} + \frac{\partial I_3}{\partial \boldsymbol{C}} \otimes \frac{\partial I_2}{\partial \boldsymbol{C}}\right) \\ &+ \frac{\partial^2 \psi}{\partial I_1\,\partial I_3}\left(\frac{\partial I_1}{\partial \boldsymbol{C}} \otimes \frac{\partial I_3}{\partial \boldsymbol{C}} + \frac{\partial I_3}{\partial \boldsymbol{C}} \otimes \frac{\partial I_1}{\partial \boldsymbol{C}}\right) \\ &+ \frac{\partial \psi}{\partial I_1}\frac{\partial^2 I_1}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}} + \frac{\partial \psi}{\partial I_2}\frac{\partial^2 I_2}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}} + \frac{\partial \psi}{\partial I_3}\frac{\partial^2 I_3}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}}\end{aligned} \tag{3.20}$$

The only non material behaviour-related terms which are not already defined during stress evaluation are the second partial derivatives of the invariants w.r.t. the right Cauchy-Green deformation tensor.

$$\begin{aligned}\frac{\partial^2 I_1}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}} &= \nvDash \\ \frac{\partial^2 I_2}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}} &= \boldsymbol{I} \otimes \boldsymbol{I} - \boldsymbol{I} \odot \boldsymbol{I} \\ \frac{\partial^2 I_3}{\partial \boldsymbol{C}\,\partial \boldsymbol{C}} &= I_3 \left(\boldsymbol{C}^{-1} \otimes \boldsymbol{C}^{-1} - \boldsymbol{C}^{-1} \odot \boldsymbol{C}^{-1}\right)\end{aligned} \tag{3.21}$$

`gradient`(*C*, *statevars*)

The gradient as the partial derivative of the strain energy function w.r.t. the right Cauchy-Green deformation tensor (one half of the second Piola Kirchhoff stress tensor).

hessian(*C*, *statevars*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the right Cauchy-Green deformation tensor (a quarter of the Lagrangian fourth-order elasticity tensor associated to the second Piola-Kirchhoff stress tensor).

class hyperelastic.frameworks.**Stretches**(*material*, *nstatevars=0*, *parallel=False*)

The Framework for a Total-Lagrangian stretch-based isotropic hyperelastic material formulation provides the material behaviour-independent parts for evaluating the second Piola-Kirchhoff stress tensor as well as its associated fourth-order elasticity tensor.

The gradient as well as the hessian of the strain energy function are carried out w.r.t. the right Cauchy-Green deformation tensor. Hence, the work-conjugate stress tensor is one half of the second Piola-Kirchhoff stress tensor and the fourth-order elasticitiy tensor used here is a quarter of the Total-Lagrangian elasticity tensor.

$$\psi(\boldsymbol{C}) = \psi(\lambda_\alpha(\boldsymbol{C})) \tag{3.22}$$

The principal stretches (the square roots of the eigenvalues) of the left or right Cauchy-Green deformation tensor are obtained by the solution of the eigenvalue problem,

$$\left(\boldsymbol{C} - \lambda_\alpha^2 \boldsymbol{I}\right) \boldsymbol{N}_\alpha = \boldsymbol{0} \tag{3.23}$$

where the Cauchy-Green deformation tensors eliminate the rigid body rotations of the deformation gradient and serve as a quadratic change-of-length measure of the deformation.

$$\begin{aligned} \boldsymbol{C} &= \boldsymbol{F}^T \boldsymbol{F} \\ \boldsymbol{b} &= \boldsymbol{F} \boldsymbol{F}^T \end{aligned} \tag{3.24}$$

The first partial derivative of the strain energy function w.r.t. a principal stretch

$$\psi_{,\alpha} = \frac{\partial \psi}{\partial \lambda_\alpha} \tag{3.25}$$

and the partial derivative of a principal stretch w.r.t. the right Cauchy-Green deformation tensor is defined

$$\frac{\partial \lambda_\alpha}{\partial \boldsymbol{C}} = \frac{\partial (\lambda_\alpha^2)^{1/2}}{\partial \boldsymbol{C}} = \frac{1}{2\lambda_\alpha} \boldsymbol{M}_\alpha \tag{3.26}$$

with the eigenbase as the dyadic (outer vector) product of eigenvectors.

$$\boldsymbol{M}_\alpha = \boldsymbol{N}_\alpha \otimes \boldsymbol{N}_\alpha \tag{3.27}$$

The second Piola-Kirchhoff stress tensor is formulated by the application of the chain rule and a sum of all principal stretch contributions.

$$\begin{aligned} \frac{\partial \psi}{\partial \boldsymbol{C}} &= \sum_\alpha \frac{\partial \psi}{\partial \lambda_\alpha} \frac{\partial \lambda_\alpha}{\partial \boldsymbol{C}} \\ \boldsymbol{S} &= 2 \frac{\partial \psi}{\partial \boldsymbol{C}} \end{aligned} \tag{3.28}$$

Furthermore, the second partial derivatives of the elasticity tensor are carried out.

$$\begin{aligned} \frac{\partial^2 \psi}{\partial \boldsymbol{C}\, \partial \boldsymbol{C}} = &\sum_\alpha \sum_\beta \frac{\partial^2 \psi}{\partial \lambda_\alpha\, \partial \lambda_\beta} \frac{\partial \lambda_\alpha}{\partial \boldsymbol{C}} \otimes \frac{\partial \lambda_\beta}{\partial \boldsymbol{C}} \\ &+ \sum_\alpha \sum_{\beta \neq \alpha} \frac{\frac{\partial \psi}{\partial \lambda_\alpha^2} - \frac{\partial \psi}{\partial \lambda_\beta^2}}{\lambda_\alpha^2 - \lambda_\beta^2} \left(\boldsymbol{M}_\alpha \odot \boldsymbol{M}_\beta + \boldsymbol{M}_\beta \odot \boldsymbol{M}_\alpha\right) \end{aligned} \tag{3.29}$$

$$\mathbb{C} = 4\frac{\partial^2 \psi}{\partial \boldsymbol{C}\, \partial \boldsymbol{C}} \tag{3.30}$$

In case of repeated equal principal stretches, the rule of d'Hospital is applied.

$$\lim_{\lambda_\beta^2 \to \lambda_\alpha^2} \left(\frac{\frac{\partial \psi}{\partial \lambda_\alpha^2} - \frac{\partial \psi}{\partial \lambda_\beta^2}}{\lambda_\alpha^2 - \lambda_\beta^2} \right) = \left(-\frac{\partial^2 \psi}{\partial \lambda_\alpha^2\, \partial \lambda_\beta^2} + \frac{\partial^2 \psi}{\partial \lambda_\beta^2\, \partial \lambda_\beta^2} \right) \tag{3.31}$$

gradient(*C*, *statevars*)

The gradient as the partial derivative of the strain energy function w.r.t. the right Cauchy-Green deformation tensor (one half of the second Piola Kirchhoff stress tensor).

hessian(*C*, *statevars*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the right Cauchy-Green deformation tensor (a quarter of the Lagrangian fourth-order elasticity tensor associated to the second Piola-Kirchhoff stress tensor).

CHAPTER

FOUR

MODELS

class hyperelastic.models.invariants.**ThirdOrderDeformation**(*C10=0*, *C01=0*, *C11=0*, *C20=0*, *C30=0*, *strain=False*)

Third Order Deformation isotropic hyperelastic material formulation based on the first and second invariant of the right Cauchy-Green deformation tensor. The strain energy density per unit undeformed volume is given as a sum of multivariate polynomials.

$$\psi(I_1, I_2) = \sum_{(i+j)\geq 1}^{(i+2j)\leq 3} C_{ij}\ (I_1 - 3)^i (I_2 - 3)^j \tag{4.1}$$

The first partial derivatives of the strain energy density w.r.t. the invariants are given below.

$$\begin{aligned} \frac{\partial \psi}{\partial I_1} &= \sum_{i\geq 1} C_{ij}\ i\ (I_1 - 3)^{i-1} (I_2 - 3)^j \\ \frac{\partial \psi}{\partial I_2} &= \sum_{j\geq 1} C_{ij}\ (I_1 - 3)^i\ j\ (I_2 - 3)^{j-1} \end{aligned} \tag{4.2}$$

Furthermore, the second partial derivatives of the strain energy density w.r.t. the invariants are carried out.

$$\begin{aligned} \frac{\partial^2 \psi}{\partial I_1\ \partial I_1} &= \sum_{i\geq 2} C_{ij}\ i\ (i-1)\ (I_1 - 3)^{i-2} (I_2 - 3)^j \\ \frac{\partial^2 \psi}{\partial I_2\ \partial I_2} &= \sum_{j\geq 2} C_{ij}\ (I_1 - 3)^i\ j\ (j-1)\ (I_2 - 3)^{j-2} \\ \frac{\partial^2 \psi}{\partial I_1\ \partial I_2} &= \sum_{i\geq 1, j\geq 1} C_{ij}\ i\ (I_1 - 3)^{i-1}\ j\ (I_2 - 3)^{j-1} \end{aligned} \tag{4.3}$$

gradient(*I1*, *I2*, *I3*, *statevars*)

The gradient as the partial derivative of the strain energy function w.r.t. the invariants.

hessian(*I1*, *I2*, *I3*, *statevars*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the invariants.

class hyperelastic.models.invariants.**TorchModel**(*fun*, ***kwargs*)

Isotropic hyperelastic material formulation based on a given strain energy density function `fun(I1, I2, I3, **kwargs)` per unit undeformed volume. The gradients are carried out by automatic differentiation using PyTorch.

$$\psi = \psi(I_1, I_2, I_3) \tag{4.4}$$

Note

PyTorch uses single-precision by default. This must be considered in numeric simulations, i.e. the error tolerance should not exceed `np.sqrt(torch.finfo(torch.float).eps)` (approx. `tol=5e-4`). For double-precision, enable `torch.float64` as default.

```
import torch

torch.set_default_dtype(torch.float64)
```

Examples

```
>>> import hyperelastic
```

```
>>> def yeoh(I1, I2, I3, C10, C20, C30):
>>>     "Yeoh isotropic hyperelastic material formulation."
>>>     return C10 * (I1 - 3) + C20 * (I1 - 3) ** 2 + C30 * (I1 - 3) ** 3
```

```
>>> model = hyperelastic.models.invariants.TorchModel(
>>>     yeoh, C10=0.5, C20=-0.05, C30=0.02
>>> )
>>> framework = hyperelastic.InvariantsFramework(model)
>>> umat = hyperelastic.DistortionalSpace(framework)
```

gradient(*I1*, *I2*, *I3*, *statevars*, *tensor=False*, *numpy=True*, *create_graph=False*, *retain_graph=False*)

The gradient as the partial derivative of the strain energy function w.r.t. the invariants.

hessian(*I1*, *I2*, *I3*, *statevars*, *numpy=True*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the invariants.

class hyperelastic.models.stretches.**GeneralizedInvariantsModel**(*material*, *fun*, ***kwargs*)

Generalized-invariants isotropic hyperelastic material formulation based on the principal stretches.

$$\psi = \psi\left(I_1\left(E_1, E_2, E_3\right), I_2\left(E_1, E_2, E_3\right), I_3\left(E_1, E_2, E_3\right)\right) \tag{4.5}$$

The three principal invariants

$$\begin{aligned} J_1 &= E_1 + E_2 + E_3 \\ J_2 &= E_1 E_2 + E_2 E_3 + E_1 E_3 \\ J_3 &= E_1 E_2 E_3 \end{aligned} \tag{4.6}$$

are formulated on a one-dimensional strain-stretch relation.

$$\begin{aligned} E_\alpha &= f(\lambda_\alpha) \\ E'_\alpha &= f'(\lambda_\alpha) = \frac{\partial f(\lambda_\alpha)}{\partial \lambda_\alpha} \\ E''_\alpha &= f''(\lambda_\alpha) = \frac{\partial^2 f(\lambda_\alpha)}{\partial \lambda_\alpha \, \partial \lambda_\alpha} \end{aligned} \tag{4.7}$$

Depending on the strain-stretch relation, the invariants contain deformation- independent values.

$$\begin{aligned} J_{1,0} &= J_1(E_\alpha(\lambda_\alpha = 1)) \\ J_{2,0} &= J_2(E_\alpha(\lambda_\alpha = 1)) \\ J_{3,0} &= J_3(E_\alpha(\lambda_\alpha = 1)) \end{aligned} \tag{4.8}$$

The deformation-dependent parts of the invariants are scaled by deformation- independent coefficients of normalization. The deformation-independent parts are re-added after the scaling.

$$\begin{aligned} I_1 &= c_1(J_1 - J_{1,0}) + J_{1,0} \\ I_2 &= c_2(J_2 - J_{2,0}) + J_{2,0} \\ I_3 &= J_3 \end{aligned} \tag{4.9}$$

Note that the scaling is only applied to the first and second invariant, as the third invariant does not contribute to the strain energy function at the undeformed state.

$$\begin{aligned} E_0 &= E(\lambda = 1) \\ E_0' &= E'(\lambda = 1) \\ E_0'' &= E''(\lambda = 1) \end{aligned} \tag{4.10}$$

The second partial derivative of the strain w.r.t. the stretch must be provided for a reference strain, e.g. the Green-Lagrange strain measure (at the undeformed state).

$$\begin{aligned} J_{1,0}'' &= \frac{3}{2}\left(E_0'' + E_0'\right) \\ J_{2,0}'' &= \frac{3}{2}\left((2E_0(E_0'' + E_0')) - E_0'^2\right) \end{aligned} \tag{4.11}$$

$$\begin{aligned} c_1 &= \frac{J_{1,0,ref}''}{J_{1,0}''} \\ c_2 &= \frac{J_{2,0,ref}''}{J_{2,0}''} \end{aligned} \tag{4.12}$$

The first partial derivatives of the strain energy function w.r.t. the invariants

$$\begin{aligned} \psi_{,1} &= \frac{\partial \psi}{\partial I_1} \\ \psi_{,2} &= \frac{\partial \psi}{\partial I_2} \\ \psi_{,3} &= \frac{\partial \psi}{\partial I_3} \end{aligned} \tag{4.13}$$

and the partial derivatives of the invariants w.r.t. the principal stretches are defined. From here on, this is consistent with any invariant-based hyperelastic material formulation, except for the factors of normalization.

$$\begin{aligned} \frac{\partial I_1}{\partial E_\alpha} &= c_1 \\ \frac{\partial I_2}{\partial E_\alpha} &= c_2\left(E_\beta + E_\gamma\right) \\ \frac{\partial I_3}{\partial E_\alpha} &= E_\beta E_\gamma \end{aligned} \tag{4.14}$$

The first partial derivatives of the strain energy density w.r.t. the principal stretches are required for the principal values of the stress.

$$\frac{\partial \psi}{\partial \lambda_\alpha} = \frac{\partial \psi}{\partial I_1}\frac{\partial I_1}{\partial E_\alpha}\frac{\partial E_\alpha}{\partial \lambda_\alpha} + \frac{\partial \psi}{\partial I_2}\frac{\partial I_2}{\partial E_\alpha}\frac{\partial E_\alpha}{\partial \lambda_\alpha} + \frac{\partial \psi}{\partial I_3}\frac{\partial I_3}{\partial E_\alpha}\frac{\partial E_\alpha}{\partial \lambda_\alpha} \tag{4.15}$$

Furthermore, the second partial derivatives of the strain energy density w.r.t. the principal stretches, necessary for the principal components of the elastic tangent moduli, are carried out. This is done in two steps: first, the

second partial derivatives w.r.t. the principal strain components are carried out, followed by the projection to the derivatives w.r.t. the principal stretches.

$$\begin{aligned}\frac{\partial^2\psi}{\partial E_\alpha\,\partial E_\beta} &= \frac{\partial^2\psi}{\partial I_1\,\partial I_1}\frac{\partial I_1}{\partial E_\alpha}\frac{\partial I_1}{\partial E_\beta} + \frac{\partial^2\psi}{\partial I_2\,\partial I_2}\frac{\partial I_2}{\partial E_\alpha}\frac{\partial I_2}{\partial E_\beta} + \frac{\partial^2\psi}{\partial I_3\,\partial I_3}\frac{\partial I_3}{\partial E_\alpha}\frac{\partial I_3}{\partial E_\beta} \\ &+ \frac{\partial^2\psi}{\partial I_1\,\partial I_2}\left(\frac{\partial I_1}{\partial E_\alpha}\frac{\partial I_2}{\partial E_\beta} + \frac{\partial I_2}{\partial E_\alpha}\frac{\partial I_1}{\partial E_\beta}\right) \\ &+ \frac{\partial^2\psi}{\partial I_2\,\partial I_3}\left(\frac{\partial I_2}{\partial E_\alpha}\frac{\partial I_3}{\partial E_\beta} + \frac{\partial I_3}{\partial E_\alpha}\frac{\partial I_2}{\partial E_\beta}\right) \\ &+ \frac{\partial^2\psi}{\partial I_1\,\partial I_3}\left(\frac{\partial I_1}{\partial E_\alpha}\frac{\partial I_3}{\partial E_\beta} + \frac{\partial I_3}{\partial E_\alpha}\frac{\partial I_1}{\partial E_\beta}\right) \\ &+ \frac{\partial\psi}{\partial I_1}\frac{\partial^2 I_1}{\partial E_\alpha\,\partial E_\beta} + \frac{\partial\psi}{\partial I_2}\frac{\partial^2 I_1}{\partial E_\alpha\,\partial E_\beta} + \frac{\partial\psi}{\partial I_3}\frac{\partial^2 I_1}{\partial E_\alpha\,\partial E_\beta}\end{aligned} \tag{4.16}$$

$$\frac{\partial^2\psi}{\partial\lambda_\alpha\,\partial\lambda_\beta} = \frac{\partial E_\alpha}{\partial\lambda_\alpha}\frac{\partial^2\psi}{\partial E_\alpha\,\partial E_\beta}\frac{\partial E_\beta}{\partial\lambda_\beta} + \left(\frac{\partial\psi}{\partial I_1}\frac{\partial I_1}{\partial E_\alpha} + \frac{\partial\psi}{\partial I_2}\frac{\partial I_2}{\partial E_\alpha} + \frac{\partial\psi}{\partial I_3}\frac{\partial I_3}{\partial E_\alpha}\right)\frac{\partial^2 E_\alpha}{\partial\lambda_\alpha\partial\lambda_\alpha} \tag{4.17}$$

gradient(*stretches*, *statevars*)

The gradient as the partial derivative of the strain energy function w.r.t. the principal stretches.

hessian(*stretches*, *statevars*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the principal stretches.

class hyperelastic.models.stretches.**Ogden**(*mu*, *alpha*)

Ogden isotropic hyperelastic material formulation based on the principal stretches. The strain energy density per unit undeformed volume is given as a sum of individual contributions from the principal stretches.

$$\begin{aligned}\psi(\lambda_\alpha) &= \sum_\alpha \psi_\alpha(\lambda_\alpha) \\ \psi_\alpha(\lambda_\alpha) &= \frac{2\mu}{k^2}\left(\lambda_\alpha^k - 1\right)\end{aligned} \tag{4.18}$$

The first partial derivatives of the strain energy density w.r.t. the principal stretches are required for the principal values of the stress.

$$\frac{\partial\psi}{\partial\lambda_\alpha} = \frac{2\mu}{k}\lambda_\alpha^{k-1} \tag{4.19}$$

Furthermore, the second partial derivatives of the strain energy density w.r.t. the principal stretches, necessary for the principal components of the elastic tangent moduli, are carried out.

$$\frac{\partial^2\psi}{\partial\lambda_\alpha\,\partial\lambda_\alpha} = \frac{2\mu(k-1)}{k}\lambda_\alpha^{k-2} \tag{4.20}$$

gradient(*stretches*, *statevars*)

The gradient as the partial derivative of the strain energy function w.r.t. the principal stretches.

hessian(*stretches*, *statevars*)

The hessian as the second partial derivatives of the strain energy function w.r.t. the principal stretches.

CHAPTER

FIVE

MATH

Warning

Shear terms are not doubled for strain-like tensors, instead all math operations take care of the reduced vector storage.

Symmetric properties of dyadic products

The minor **and** major-symmetric property indicates whether the fourth-order tensor as a result of a dyadic product of two symmetric second-order tensors may be transferred into a reduced matrix storage. Special cases of minor but not major-symmetry and vice versa exist but are not shown here.

Function	$\boldsymbol{A} \neq \boldsymbol{B}$	$\boldsymbol{A} = \boldsymbol{B}$
dya()		✓
cdya()	✓	✓
cdya_ik()		
cdya_il()		

hyperelastic.math.**astensor**(*A*, *mode=2*)

Convert a three-dimensional tensor from symmetric (Voigt-notation) vector/matrix storage into full array-storage.

Parameters

- **A** (*np.ndarray*) – A three-dimensional second- or fourth-order tensor in reduced symmetric (Voigt) vector/matrix storage.
- **mode** (*int, optional*) – The mode, 2 for second-order and 4 for fourth-order tensors (default is 2).

Returns

A three-dimensional second- or fourth-order tensor in full array-storage.

Return type

np.ndarray

Notes

This is the inverse operation of *asvoigt()*.

For a symmetric 3x3 second-order tensor $C_{ij} = C_{ji}$, the entries are re-created from a 6x1 vector.

$$\boldsymbol{C} = \begin{bmatrix} C_{11} & C_{22} & C_{33} & C_{12} & C_{23} & C_{13} \end{bmatrix}^T \longrightarrow \boldsymbol{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{12} & C_{22} & C_{23} \\ C_{13} & C_{23} & C_{33} \end{bmatrix} \tag{5.1}$$

For a (at least minor) symmetric 3x3x3x3 fourth-order tensor $A_{ijkl} = A_{jikl} = A_{ijlk} = A_{jilk}$, the entries are re-created from a 6x6 matrix.

$$\begin{aligned} \mathbb{A} &= \begin{bmatrix} A_{1111} & A_{1122} & A_{1133} & A_{1112} & A_{1123} & A_{1113} \\ A_{2211} & A_{2222} & A_{2233} & A_{2212} & A_{2223} & A_{2213} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ A_{1311} & A_{1322} & A_{1333} & A_{1312} & A_{1323} & A_{1313} \end{bmatrix} \\ \longrightarrow & \begin{bmatrix} A_{1111} & A_{1112} & A_{1113} & A_{1121} & A_{1122} & A_{1123} & A_{1131} & A_{1132} & A_{1133} \\ A_{1211} & A_{1212} & A_{1213} & A_{1221} & A_{1222} & A_{1223} & A_{1231} & A_{1232} & A_{1233} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ A_{3111} & A_{3112} & A_{3113} & A_{3121} & A_{3122} & A_{3123} & A_{3131} & A_{3132} & A_{3133} \end{bmatrix} \end{aligned} \tag{5.2}$$

Examples

```
>>> from hyperelastic.math import asvoigt, astensor
>>> import numpy as np
```

```
>>> C = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> C6 = asvoigt(C, mode=2)
>>> D = astensor(C6, mode=2)
>>> np.allclose(C, D)
True
```

```
>>> D
array([[1. , 1.3, 1.5],
       [1.3, 1.1, 1.4],
       [1.5, 1.4, 1.2]])
```

```
>>> A = np.einsum("ij,kl", C, C)
>>> A66 = asvoigt(A, mode=4)
>>> B = astensor(A66, mode=4)
>>> np.allclose(A, B)
True
```

hyperelastic.math.**asvoigt**(*A*, *mode=2*)

Convert a three-dimensional tensor from full array-storage into reduced symmetric (Voigt-notation) vector/matrix storage.

Parameters

- **A** (*np.ndarray*) – A three-dimensional second- or fourth-order tensor in full array-storage.
- **mode** (*int, optional*) – The mode, 2 for second-order and 4 for fourth-order tensors (default is 2).

Returns
A three-dimensional second- or fourth-order tensor in reduced symmetric (Voigt) vector/matrix storage.

Return type
np.ndarray

Notes

This is the inverse operation of *astensor()*.

For a symmetric 3x3 second-order tensor $C_{ij} = C_{ji}$, the upper triangle entries are inserted into a 6x1 vector, starting from the main diagonal, followed by the consecutive next upper diagonals.

$$\boldsymbol{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{12} & C_{22} & C_{23} \\ C_{13} & C_{23} & C_{33} \end{bmatrix} \quad \longrightarrow \boldsymbol{C} = \begin{bmatrix} C_{11} & C_{22} & C_{33} & C_{12} & C_{23} & C_{13} \end{bmatrix}^T \tag{5.3}$$

For a (at least minor) symmetric 3x3x3x3 fourth-order tensor $A_{ijkl} = A_{jikl} = A_{ijlk} = A_{jilk}$, rearranged to 9x9, the upper triangle entries are inserted into a 6x6 matrix, starting from the main diagonal, followed by the consecutive next upper diagonals.

$$\begin{bmatrix} A_{1111} & A_{1112} & A_{1113} & A_{1121} & A_{1122} & A_{1123} & A_{1131} & A_{1132} & A_{1133} \\ A_{1211} & A_{1212} & A_{1213} & A_{1221} & A_{1222} & A_{1223} & A_{1231} & A_{1232} & A_{1233} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ A_{3111} & A_{3112} & A_{3113} & A_{3121} & A_{3122} & A_{3123} & A_{3131} & A_{3132} & A_{3133} \end{bmatrix} \\ \longrightarrow \mathbb{A} = \begin{bmatrix} A_{1111} & A_{1122} & A_{1133} & A_{1112} & A_{1123} & A_{1113} \\ A_{2211} & A_{2222} & A_{2233} & A_{2212} & A_{2223} & A_{2213} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ A_{1311} & A_{1322} & A_{1333} & A_{1312} & A_{1323} & A_{1313} \end{bmatrix} \tag{5.4}$$

Examples

```
>>> from hyperelastic.math import asvoigt
>>> import numpy as np
```

```
>>> C = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> asvoigt(C, mode=2)
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5])
```

```
>>> A = np.einsum("ij,kl", C, C)
>>> asvoigt(A, mode=4)
array([[1.  , 1.1 , 1.2 , 1.3 , 1.4 , 1.5 ],
       [1.1 , 1.21, 1.32, 1.43, 1.54, 1.65],
       [1.2 , 1.32, 1.44, 1.56, 1.68, 1.8 ],
       [1.3 , 1.43, 1.56, 1.69, 1.82, 1.95],
       [1.4 , 1.54, 1.68, 1.82, 1.96, 2.1 ],
       [1.5 , 1.65, 1.8 , 1.95, 2.1 , 2.25]])
```

hyperelastic.math.**cdya**(*A*, *B*, *out=None*)

The full-symmetric crossed-dyadic product of two symmetric second-order tensors in reduced vector storage.

Parameters

- **A** (*np.ndarray*) – First symmetric second-order tensor in reduced vector storage.
- **B** (*np.ndarray*) – Second symmetric second-order tensor in reduced vector storage.

- **out** (*np.ndarray or None, optional*) – A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned (default is None).

Returns
Symmetric crossed-dyadic product in reduced matrix storage.

Return type
np.ndarray

Notes

The result of the symmetric crossed-dyadic product of two symmetric second order tensors is a minor- and major-symmetric fourth-order tensor.

$$\begin{aligned}\mathbb{C} &= \frac{1}{2}\left(\boldsymbol{A}\odot\boldsymbol{B}+\boldsymbol{B}\odot\boldsymbol{A}\right)\\ \mathbb{C} &= \frac{1}{4}\left(\boldsymbol{A}\overline{\otimes}\boldsymbol{B}+\boldsymbol{A}\underline{\otimes}\boldsymbol{B}+\boldsymbol{B}\overline{\otimes}\boldsymbol{A}+\boldsymbol{B}\underline{\otimes}\boldsymbol{A}\right)\\ \mathbb{C}_{ijkl} &= \frac{1}{4}\left(A_{ik}\,B_{jl}+A_{il}\,B_{kj}+B_{ik}\,A_{jl}+B_{il}\,A_{kj}\right)\end{aligned} \tag{5.5}$$

Note

The technical implementation is based on an answer of Jérôme Richard (see stackoverflow).

Examples

```
>>> from hyperelastic.math import asvoigt, astensor, cdya
>>> import numpy as np
```

```
>>> C = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> D = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> CD = (np.einsum("ik,jl", C, D) + np.einsum("il,kj", C, D)) / 2
>>> DC = (np.einsum("ik,jl", D, C) + np.einsum("il,kj", D, C)) / 2
```

```
>>> A = (CD + DC) / 2
```

```
>>> C6 = asvoigt(C, mode=2)
>>> D6 = asvoigt(D, mode=2)
>>> cdya(C6, D6)
array([[1.2   , 1.82  , 2.25  , 1.48  , 2.025 , 1.65  ],
       [1.82  , 1.21  , 1.82  , 1.485 , 1.485 , 1.825 ],
       [2.25  , 1.82  , 1.2   , 2.025 , 1.48  , 1.65  ],
       [1.48  , 1.485 , 2.025 , 1.515 , 1.7375, 1.7575],
       [2.025 , 1.485 , 1.48  , 1.7375, 1.515 , 1.7575],
       [1.65  , 1.825 , 1.65  , 1.7575, 1.7575, 1.735 ]])
```

```
>>> np.allclose(A, astensor(cdya(C6, D6), mode=4))
True
```

```
>>> np.allclose(A, astensor(cdya(D6, C6), mode=4))
True
```

hyperelastic.math.**cdya_ik**(*A*, *B*, *out=None*)

The overlined-dyadic product of two symmetric second-order tensors in reduced vector storage, where the inner indices (the second index of the first tensor and the first index of the second tensor) are interchanged.

Parameters

- **A** (*np.ndarray*) – First symmetric second-order tensor in reduced vector storage.
- **B** (*np.ndarray*) – Second symmetric second-order tensor in reduced vector storage.
- **out** (*np.ndarray or None, optional*) – A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned (default is None).

Returns

Overlined-dyadic product in full-array storage.

Return type

np.ndarray

Notes

The result of the overlined-dyadic product of two symmetric second order tensors is a major- (but not minor-) symmetric fourth-order tensor. This is also the case for $\boldsymbol{A} = \boldsymbol{B}$.

$$\begin{aligned} \mathbb{C} &= \boldsymbol{A} \overline{\otimes} \boldsymbol{B} \\ \mathbb{C}_{ijkl} &= A_{ik}\, B_{jl} \end{aligned} \tag{5.6}$$

Examples

```
>>> from hyperelastic.math import asvoigt, cdya_ik
>>> import numpy as np
```

```
>>> C = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> D = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> A = np.einsum("ik,jl", C, D)
```

```
>>> C6 = asvoigt(C, mode=2)
>>> D6 = asvoigt(D, mode=2)
```

```
>>> np.allclose(A, cdya_ik(C6, D6))
True
```

hyperelastic.math.**cdya_il**(*A*, *B*, *out=None*)

The underlined-dyadic product of two symmetric second-order tensors in reduced vector storage, where the right indices of the two tensors are interchanged.

Parameters

- **A** (*np.ndarray*) – First symmetric second-order tensor in reduced vector storage.
- **B** (*np.ndarray*) – Second symmetric second-order tensor in reduced vector storage.

- **out** (*np.ndarray or None, optional*) – A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned (default is None).

Returns
Underlined-dyadic product in full-array storage.

Return type
np.ndarray

Notes

The result of the underlined-dyadic product of two symmetric second order tensors is a non-symmetric fourth-order tensor. In case of $\boldsymbol{A} = \boldsymbol{B}$, the fourth-order tensor is major-symmetric.

$$\begin{aligned} \mathbb{C} &= \boldsymbol{A} \underline{\otimes} \boldsymbol{B} \\ \mathbb{C}_{ijkl} &= A_{il}\, B_{kj} \end{aligned} \tag{5.7}$$

Examples

```
>>> from hyperelastic.math import asvoigt, cdya_il
>>> import numpy as np
```

```
>>> C = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> D = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> A = np.einsum("il,kj", C, D)
```

```
>>> C6 = asvoigt(C, mode=2)
>>> D6 = asvoigt(D, mode=2)
```

```
>>> np.allclose(A, cdya_il(C6, D6))
True
```

hyperelastic.math.**ddot**(*A*, *B*, *mode=(2, 2)*)

The double-dot product of two symmetric tensors in reduced vector storage, where the two innermost indices of both tensors are contracted.

Parameters

- **A** (*np.ndarray*) – First symmetric second- or fourth-order tensor in reduced vector storage.
- **B** (*np.ndarray*) – Second symmetric second- or fourth-order tensor in reduced vector storage.
- **mode** (*2-tuple, optional*) – The mode, 2 for second-order and 4 for fourth-order tensors (default is (2, 2)).

Returns
Double-dot product of two symmetric tensors in scalar or reduced vector/matrix storage.

Return type
np.ndarray

Notes

$$C = \boldsymbol{A} : \boldsymbol{B}$$
$$C = A_{ij} : B_{ij} \tag{5.8}$$

$$\boldsymbol{C} = \boldsymbol{A} : \mathbb{B}$$
$$C_{kl} = A_{ij} : \mathbb{B}_{ijkl} \tag{5.9}$$

$$\boldsymbol{C} = \mathbb{B} : \boldsymbol{A}$$
$$C_{ij} = \mathbb{B}_{ijkl} : A_{kl} \tag{5.10}$$

$$\mathbb{C} = \mathbb{A} : \mathbb{B}$$
$$\mathbb{C}_{ijmn} = \mathbb{A}_{ijkl} : \mathbb{A}_{klmn} \tag{5.11}$$

Examples

```
>>> from hyperelastic.math import asvoigt, ddot
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> B = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> A4 = np.einsum("ij,kl", A, B)
>>> B4 = (np.einsum("ik,jl", A, A) + np.einsum("il,kj", A, A)) / 2
```

```
>>> ddot(asvoigt(A), asvoigt(B), mode=(2, 2))
15.39
```

```
>>> ddot(asvoigt(A), asvoigt(B4, mode=4), mode=(2, 4))
array([18.899, 18.863, 21.698, 18.903, 20.253, 20.316])
```

```
>>> ddot(asvoigt(B4, mode=4), asvoigt(A), mode=(4, 2))
array([18.899, 18.863, 21.698, 18.903, 20.253, 20.316])
```

```
>>> ddot(asvoigt(A4, mode=4), asvoigt(B4, mode=4), mode=(4, 4))
array([[18.519 , 18.787 , 21.944 , 18.675 , 20.326 , 20.225 ],
       [20.3709, 20.6657, 24.1384, 20.5425, 22.3586, 22.2475],
       [22.2228, 22.5444, 26.3328, 22.41  , 24.3912, 24.27  ],
       [24.0747, 24.4231, 28.5272, 24.2775, 26.4238, 26.2925],
       [25.9266, 26.3018, 30.7216, 26.145 , 28.4564, 28.315 ],
       [27.7785, 28.1805, 32.916 , 28.0125, 30.489 , 30.3375]])
```

hyperelastic.math.**det**(*A*)

The determinant of a symmetric second-order tensor in reduced vector storage.

Parameters

A (*np.ndarray*) – Symmetric second-order tensor in reduced vector storage.

Returns

Determinant of a symmetric second-order tensor in reduced vector storage.

Return type

np.ndarray

Examples

```
>>> from hyperelastic.math import asvoigt, det
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> B = asvoigt(A)
```

```
>>> det(B)
0.3169999999999993
```

```
>>> np.allclose(det(B), np.linalg.det(A))
True
```

hyperelastic.math.**dev**(*A*)

The deviatoric part of a three-dimensional second-order tensor.

Parameters
A (*np.ndarray*) – Symmetric second-order tensor in reduced vector storage.

Returns
Deviatoric part of the symmetric second-order tensor in reduced vector storage.

Return type
np.ndarray

Notes

$$\operatorname{dev}(\boldsymbol{C}) = \boldsymbol{C} - \frac{\operatorname{tr}(\boldsymbol{C})}{3}\boldsymbol{I} \tag{5.12}$$

hyperelastic.math.**dot**(*A*, *B*, *mode=(2, 2)*)

The dot product of two symmetric second-order tensors in reduced vector storage, where the second index of the first tensor and the first index of the second tensor are contracted.

Parameters

- **A** (*np.ndarray*) – First symmetric second-order tensor in reduced vector storage.
- **B** (*np.ndarray*) – Second symmetric second-order tensor in reduced vector storage.
- **mode** (*2-tuple, optional*) – The mode, 2 for second-order and 4 for fourth-order tensors (default is (2, 2)).

Returns
Dot product of two symmetric second-order tensors in reduced vector storage.

Return type
np.ndarray

Notes

$$\begin{aligned} C &= \boldsymbol{A}\,\boldsymbol{B} \\ C_{ij} &= A_{ik}B_{kj} \end{aligned} \tag{5.13}$$

Examples

```
>>> from hyperelastic.math import asvoigt, dot
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> B = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> C = dot(asvoigt(A), asvoigt(B), mode=(2, 2))
>>> C
array([[5.27, 4.78, 4.69],
       [5.2 , 4.85, 4.78],
       [5.56, 5.2 , 5.27]])
```

```
>>> D = A @ B
>>> np.allclose(C, D)
True
```

hyperelastic.math.**dya**(*A*, *B*, *out=None*)

The dyadic product of two symmetric second-order tensors in reduced vector storage.

Parameters

- **A** (*np.ndarray*) – First symmetric second-order tensor in reduced vector storage.
- **B** (*np.ndarray*) – Second symmetric second-order tensor in reduced vector storage.
- **out** (*np.ndarray or None, optional*) – A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned (default is None).

Returns

Dyadic product in reduced matrix storage.

Return type

np.ndarray

Notes

The result of the dyadic product of two symmetric second order tensors is a minor- (but not major-) symmetric fourth-order tensor. For the case of $\boldsymbol{A} = \boldsymbol{B}$, the result is both major- and minor- symmetric.

$$\begin{aligned} \mathbb{C} &= \boldsymbol{A} \otimes \boldsymbol{B} \\ \mathbb{C}_{ijkl} &= A_{ij}\, B_{kl} \end{aligned} \tag{5.14}$$

Examples

```
>>> from hyperelastic.math import asvoigt, astensor, dya
>>> import numpy as np
```

```
>>> C = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> D = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> A = np.einsum("ij,kl", C, D)
```

```
>>> C6 = asvoigt(C, mode=2)
>>> D6 = asvoigt(D, mode=2)
```

```
>>> np.allclose(A, astensor(dya(C6, D6), mode=4))
True
```

hyperelastic.math.**eigh**(*A*, *fun=None*)

Eigenvalues and -bases of matrix A.

hyperelastic.math.**eye**(*A=None*)

A 3x3 tensor in Voigt-storage with ones on the diagonal and zeros elsewhere. The dimension is taken from the input argument (a symmetric second-order tensor in reduced vector storage).

Parameters

A (*np.ndarray or None, optional*) – Symmetric second- or fourth-order tensor in reduced vector storage (default is None).

Returns

Identity matrix in reduced vector storage.

Return type

np.ndarray

Notes

$$\boldsymbol{I} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.15}$$

Examples

```
>>> from hyperelastic.math import asvoigt, eye
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> B = asvoigt(A)
>>> eye(B)
array([1., 1., 1., 0., 0., 0.])
```

hyperelastic.math.**inv**(*A*, *determinant=None*, *out=None*)

The inverse of a symmetric second-order tensor in reduced vector storage.

Parameters

- **A** (*np.ndarray*) – Symmetric second-order tensor in reduced vector storage.
- **determinant** (*np.ndarray or None, optional*) – The determinant of the symmetric second-order tensor (default is None).
- **out** (*np.ndarray or None, optional*) – A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned (default is None).

Returns

Inverse of a symmetric second-order tensor in reduced vector storage.

Return type

np.ndarray

Notes

$$\boldsymbol{C}\boldsymbol{C}^{-1} = \boldsymbol{I} \tag{5.16}$$

Examples

```
>>> from hyperelastic.math import asvoigt, inv
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> B = asvoigt(A)
```

```
>>> inv(B)
array([-2.01892744, -3.31230284, -1.86119874,  1.70347003,  1.73501577, 0.5362776 ])
```

```
>>> np.allclose(dot(B, inv(B)), np.eye(3))
True
```

hyperelastic.math.**trace**(*A*)

The trace of a symmetric second-order tensor in reduced vector storage as the sum of the main diagonal.

Parameters
A (*np.ndarray*) – Symmetric second-order tensor in reduced vector storage.

Returns
Trace of a symmetric second-order tensor in reduced vector storage.

Return type
np.ndarray

Notes

$$\begin{aligned} \mathrm{tr}\,(\boldsymbol{C}) &= \boldsymbol{C} : \boldsymbol{I} = C_{11} + C_{22} + C_{33} \\ C_{kk} &= C_{ij} : \delta_{ij} \end{aligned} \tag{5.17}$$

Examples

```
>>> from hyperelastic.math import asvoigt, trace
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
```

```
>>> trA = trace(asvoigt(A))
>>> trA
3.3
```

```
>>> np.allclose(trA, np.trace(A))
True
```

hyperelastic.math.**transpose**(*A*)

The major-transpose of a symmetric fourth-order tensor in reduced vector storage.

Parameters
A (*np.ndarray*) – Symmetric fourth-order tensor in reduced vector storage.

Returns
Major-transpose of a symmetric fourth-order tensor in reduced vector storage.

Return type
np.ndarray

Notes

$$\mathbb{A}^{T}_{ijkl} = \mathbb{A}_{klij} \tag{5.18}$$

Examples

```
>>> from hyperelastic.math import asvoigt, dya, transpose
>>> import numpy as np
```

```
>>> A = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2]).reshape(3, 3)
>>> B = np.array([1.0, 1.3, 1.5, 1.3, 1.1, 1.4, 1.5, 1.4, 1.2])[::-1].reshape(3, 3)
```

```
>>> C = dya(asvoigt(A), asvoigt(B))
>>> C
array([[1.2 , 1.1 , 1.  , 1.4 , 1.3 , 1.5 ],
       [1.32, 1.21, 1.1 , 1.54, 1.43, 1.65],
       [1.44, 1.32, 1.2 , 1.68, 1.56, 1.8 ],
       [1.56, 1.43, 1.3 , 1.82, 1.69, 1.95],
       [1.68, 1.54, 1.4 , 1.96, 1.82, 2.1 ],
       [1.8 , 1.65, 1.5 , 2.1 , 1.95, 2.25]])
```

```
>>> transpose(C)
array([[1.2 , 1.32, 1.44, 1.56, 1.68, 1.8 ],
       [1.1 , 1.21, 1.32, 1.43, 1.54, 1.65],
       [1.  , 1.1 , 1.2 , 1.3 , 1.4 , 1.5 ],
       [1.4 , 1.54, 1.68, 1.82, 1.96, 2.1 ],
       [1.3 , 1.43, 1.56, 1.69, 1.82, 1.95],
       [1.5 , 1.65, 1.8 , 1.95, 2.1 , 2.25]])
```

```
>>> D = astensor(C, mode=4)
>>> E = astensor(transpose(C), mode=4)
>>> np.allclose(D, np.einsum("klij", E))
True
```

hyperelastic.math.**tril_from_triu**(*A*, *dim=6*, *out=None*)

Copy upper triangle values to lower triangle values of a nxn tensor inplace.

hyperelastic.math.**triu_from_tril**(*A*, *dim=6*, *out=None*)

Copy lower triangle values to upper triangle values of a nxn tensor inplace.

CHAPTER

SIX

LAB

Define `Experiments` and `Simulations` in the `Lab` and perform a `Curve-Fit` to `optimize` material parameters.

> **Attention**
>
> This section is under construction .

class hyperelastic.lab.**Biaxial**(*label=None*)

Incompressible biaxial tension/compression load case.

$$\boldsymbol{F} = \text{diag}\left(\begin{bmatrix}\lambda & \lambda & \frac{1}{\lambda^2}\end{bmatrix}\right) \tag{6.1}$$

defgrad(*stretch*)

Return the Deformation Gradient tensor from given stretches.

stress(*F*, *P*, *axis=0*, *traction_free=-1*)

Normal force per undeformed area for a given deformation gradient of an incompressible deformation and the first Piola-Kirchhoff stress tensor.

Parameters

- **F** (*ndarray*) – The deformation gradient.
- **P** (*ndarray*) – The first Piola-Kirchhoff stress tensor.
- **axis** (*int, optional*) – The primary axis where the longitudinal stretch is applied on (default is 0).
- **traction_free** (*int, optional*) – The secondary axis where the traction-free transverse stretch results from the constraint of incompressibility (default is -1).

Returns

The one-dimensional normal force per undeformed area.

Return type

ndarray

class hyperelastic.lab.**Experiment**(*label*, *displacement*, *force*, *area=1.0*, *length=1.0*, *time=None*, *temperature=None*)

Results of an experiment along with methods to convert and plot the data.

`label`

The title of the experiment.

Type
str

`displacement`

The measured or applied displacement data.

Type
array_like

`force`

The measured or applied force data.

Type
array_like

`area`

The undeformed reference cross-sectional area used to evaluate the stress.

Type
float

`length`

The undeformed reference length used to evaluate the stretch.

Type
float

`time`

The timetrack of the measurement.

Type
array_like or None

`temperature`

The measured or applied temperature data.

Type
array_like or None

`stretch`

The stretch as the calculated ratio of the deformed vs. the undeformed length.

Type
array_like

`plot_force_displacement`(**args*, *xlabel='Displacement d'*, *ylabel='Force F'*, *ax=None*, *label=None*, ***kwargs*)

Create a force-displacement plot.

`plot_stress_stretch`(**args*, *ax=None*, *xlabel='Stretch l/L'*, *ylabel='Force per undeformed area F/A'*, *label=None*, ***kwargs*)

Create a stress-stretch plot.

`stress`()

Evaluate the stress as force per undeformed area.

class hyperelastic.lab.**IncompressibleHomogeneousStretch**

An incompressible homogeneous stretch load case with a longitudinal stretch and perpendicular transverse stretches in principal directions. This class is intended to be subclassed by another class with a *.defgrad()* method for the evaluation of the deformation gradient as utilized by the `Uniaxial`, `Planar` and `Biaxial` load cases.

Notes

The Cauchy stress for an incompressible material is given by

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}' + p\boldsymbol{I} \tag{6.2}$$

where the Cauchy stress is converted to the first Piola-Kirchhoff stress tensor.

$$\boldsymbol{P} = J\boldsymbol{\sigma}\boldsymbol{F}^{-T} \tag{6.3}$$

The deformation gradient and its determinant are evaluated for the homogeneous incompressible deformation.

$$\begin{aligned} \boldsymbol{F} &= \operatorname{diag}\begin{pmatrix}\begin{bmatrix}\lambda_1 & \lambda_2 & \lambda_3\end{bmatrix}\end{pmatrix} \\ J &= \lambda_1\lambda_2\lambda_3 = 1 \end{aligned} \tag{6.4}$$

This enables the evaluation of the normal force per undeformed area, where quantities in the traction-free transverse direction are denoted with a subscript $(\bullet)_t$.

$$\frac{N}{A} = P - P_t\frac{\lambda_t}{\lambda} \tag{6.5}$$

stress(*F*, *P*, *axis=0*, *traction_free=-1*)

Normal force per undeformed area for a given deformation gradient of an incompressible deformation and the first Piola-Kirchhoff stress tensor.

Parameters

- **F** (*ndarray*) – The deformation gradient.
- **P** (*ndarray*) – The first Piola-Kirchhoff stress tensor.
- **axis** (*int, optional*) – The primary axis where the longitudinal stretch is applied on (default is 0).
- **traction_free** (*int, optional*) – The secondary axis where the traction-free transverse stretch results from the constraint of incompressibility (default is -1).

Returns
The one-dimensional normal force per undeformed area.

Return type
ndarray

class hyperelastic.lab.**Optimize**(*experiments*, *simulations*, *parameters*, *mask=None*)

Take lists of experiments and simulations and find material parameters for the simulation model to obtain a best possible representation of the experiments by the simulations.

experiments

A list of `Experiment`.

Type
list

simulations

A list of `Simulation`.

Type
list

parameters

The material parameters.

Type
array_like

mask

A list of masks to take the optimization-relevant data points.

Type
list of array_like

Examples

Three different test specimens are subjected to displacement-controlled uniaxial, planar and biaxial tension. The applied displacement and reaction force data is used to create the `Experiments`. The test specimen for the uniaxial load case has a cross-sectional area of $A = 25$ mm^2 and a length of $L = 100$ mm.

```
>>> area = 25
>>> length = 100
```

Some synthetic experimental data is generated to demonstrate the capabilities of the optimization.

```
>>> import numpy as np
```

```
>>> displacement = np.linspace(0, 2 * length, 100)
>>> stretch = 1 + displacement / length
>>> force = (stretch - 1 / stretch ** 2 + (stretch - 1)**5 / 10) * area
```

With this reference experimental data at hand, the list of experiments is created. In this example, the displacement and force data as well as the cross-sectional area are scaled from the synthetic uniaxial experimental data to the other (synthetic) experiments.

```
>>> from hyperelastic import lab
```

```
>>> experiments = [
>>>     lab.Experiment(
>>>         label="Uniaxial Tension",
>>>         displacement=displacement,
>>>         force=force,
>>>         area=area,
>>>         length=length,
>>>     ),
>>>     lab.Experiment(
>>>         label="Planar Tension",
>>>         displacement=displacement[::2],
>>>         force=force[::2],
>>>         area=area / (8 / 7),
>>>         length=length,
```

(continues on next page)

(continued from previous page)

```
>>>     ),
>>>     lab.Experiment(
>>>         label="Biaxial Tension",
>>>         displacement=displacement[::2] / 2,
>>>         force=force[::2],
>>>         area=area / (4 / 5),
>>>         length=length,
>>>     ),
>>> ]
```

A function which takes the material parameters and returns the hyperelastic constitutive material formulation has to be provided for the simulation objects. Here, we use an isotropic invariant-based *third-order deformation* material formulation.

```
>>> def material(**kwargs):
>>>     "A third-order deformation material formulation."
>>>
>>>     tod = hyperelastic.models.invariants.ThirdOrderDeformation(**kwargs)
>>>     framework = hyperelastic.InvariantsFramework(tod)
>>>
>>>     return hyperelastic.DeformationSpace(framework)
```

The list of labels of the material parameters is used for all simulation objects.

```
>>> labels = ["C10", "C01", "C11", "C20", "C30"]
```

Next, the simulations for all three loadcases are created. It is important to take the stretches from the according experiments.

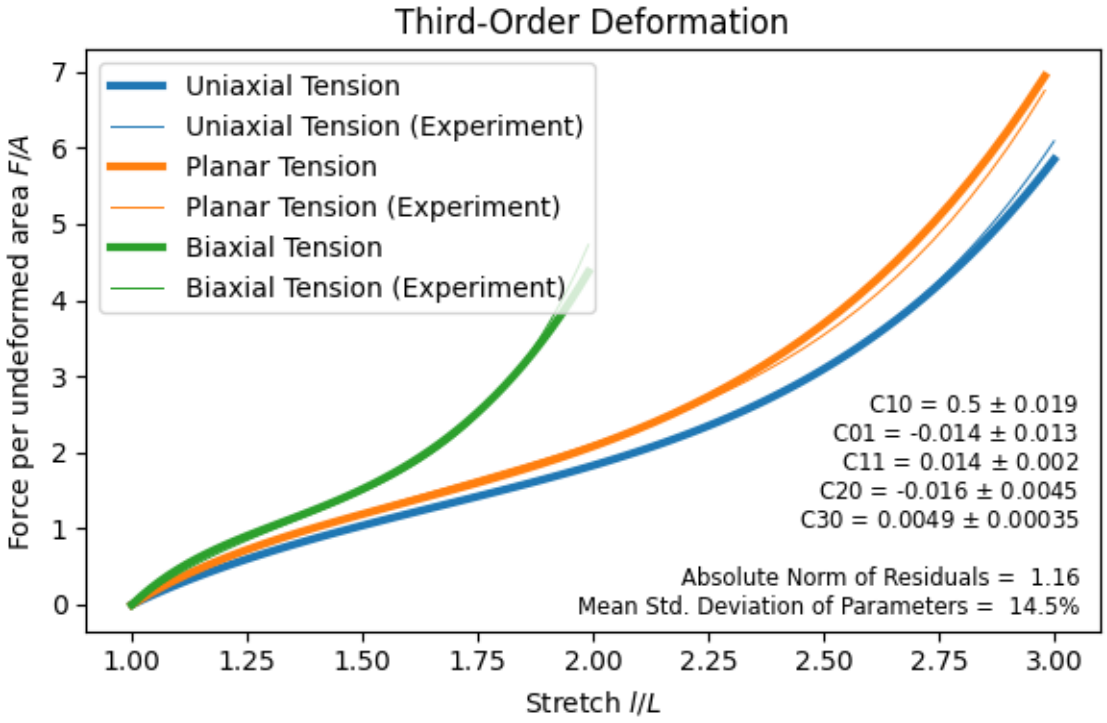
```
>>> simulations = [
>>>     lab.Simulation(
>>>         loadcase=lab.Uniaxial(),
>>>         stretch=experiments[0].stretch,
>>>         material=material,
>>>         labels=labels,
>>>     ),
>>>     lab.Simulation(
>>>         loadcase=lab.Planar(),
>>>         stretch=experiments[1].stretch,
>>>         material=material,
>>>         labels=labels,
>>>     ),
>>>     lab.Simulation(
>>>         loadcase=lab.Biaxial(),
>>>         stretch=experiments[2].stretch,
>>>         material=material,
>>>         labels=labels,
>>>     ),
>>> ]
```

Both the list of experiments and the list of simulations are passed to `Optimize`, where its curve-fit method acts as a simple wrapper for `scipy.optimize.curve_fit`. The initial material parameters are all set to one.

```
>>> optimize = lab.Optimize(
>>>     experiments=experiments,
>>>     simulations=simulations,
>>>     parameters=np.ones(5),
>>> )
```

```
>>> parameters, pcov = optimize.curve_fit(method="lm")
>>> parameters
array([ 0.50430357, -0.01413309,  0.0141219 , -0.01641752,  0.00492179])
```

```
>>> fig, ax = optimize.plot(title="Third-Order Deformation")
```



curve_fit(**args*, ***kwargs*)

Use non-linear least squares to fit a list of functions to a list of data.

init_curve_fit(**args*, ***kwargs*)

mean_relative_std()

Return the relative mean of the standard deviations of the material parameters, normalized by the absolute mean-values of the parameters.

norm_residuals()

Return the norm of the residuals.

plot(*title=None*)

class hyperelastic.lab.**Planar**(*label=None*)

Incompressible planar (shear) tension/compression load case.

$$\boldsymbol{F} = \text{diag}\left(\begin{bmatrix}\lambda & 1 & \frac{1}{\lambda}\end{bmatrix}\right) \tag{6.6}$$

defgrad(*stretch*)

Return the Deformation Gradient tensor from given stretches.

stress(*F*, *P*, *axis=0*, *traction_free=-1*)

Normal force per undeformed area for a given deformation gradient of an incompressible deformation and the first Piola-Kirchhoff stress tensor.

Parameters

- **F** (*ndarray*) – The deformation gradient.
- **P** (*ndarray*) – The first Piola-Kirchhoff stress tensor.
- **axis** (*int, optional*) – The primary axis where the longitudinal stretch is applied on (default is 0).
- **traction_free** (*int, optional*) – The secondary axis where the traction-free transverse stretch results from the constraint of incompressibility (default is -1).

Returns
The one-dimensional normal force per undeformed area.

Return type
ndarray

class hyperelastic.lab.**Simulation**(*loadcase*, *stretch*, *labels*, *material*, *parameters=None*)

Results of a simulation along with methods to convert and plot the data.

loadcase

A class with methods for evaluating the deformation gradient and the stress as normal force per undeformed area, e.g. `Uniaxial`.

Type
class

stretch

The stretch as the ratio of the deformed vs. the undeformed length.

Type
ndarray

labels

A list of the material parameter labels.

Type
list of str

material

A class with a method for evaluating the gradient of the strain energy function w.r.t. the deformation gradient, e.g. `DistortionalSpace`.

Type
class

parameters

The material parameters.

Type
array_like

Examples

The material model response behaviour of a hyperelastic material model formulation is evaluated for a `uniaxial tension` load case. A given stretch data is used to create the `Simulation` object.

```
>>> import numpy as np
>>> import hyperelastic
>>> from hyperelastic import lab
>>>
>>> stretch = np.linspace(0.7, 2.5, 181)
```

A function which takes the material parameters and returns the hyperelastic constitutive material formulation has to be provided for the simulation objects. Here, we use an isotropic invariant-based *third-order deformation* material formulation.

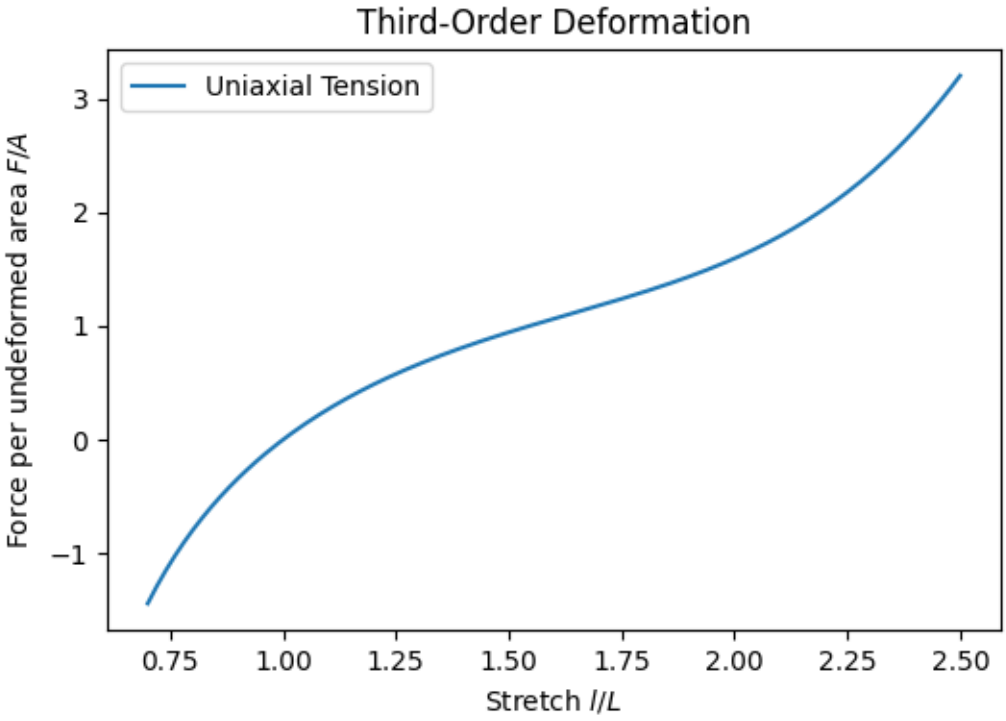
```
>>> def material(**kwargs):
>>>     "A third-order deformation material formulation."
>>>
>>>     tod = hyperelastic.models.invariants.ThirdOrderDeformation(**kwargs)
>>>     framework = hyperelastic.InvariantsFramework(tod)
>>>
>>>     return hyperelastic.DeformationSpace(framework)
```

A list of (string) labels is used to apply the list or array of parameter values to the material formulation.

```
>>> simulation = lab.Simulation(
>>>     loadcase=lab.Uniaxial(),
>>>     stretch=stretch,
>>>     material=material,
>>>     labels=["C10", "C01", "C11", "C20", "C30"],
>>>     parameters=[0.4, 0.1, 0.02, -0.04, 0.01],
>>> )
```

The stress-stretch plot returns a figure which visualizes the force per undeformed area vs. the ratio of the undeformed and deformed length.

```
>>> fig, ax = simulation.plot_stress_stretch()
>>>
>>> ax.legend()
>>> ax.set_title("Third-Order Deformation")
```



plot_force_displacement(**args*, *xlabel='Displacement d'*, *ylabel='Force F'*, *ax=None*, *label=None*, ***kwargs*)

Create a force-displacement plot.

plot_stress_stretch(**args*, *ax=None*, *xlabel='Stretch l/L'*, *ylabel='Force per undeformed area F/A'*, *label=None*, ***kwargs*)

Create a stress-stretch plot.

stress()

Evaluate the stress as force per undeformed area.

stress_curve_fit(*x*, **parameters*)

Evaluate the stress as force per undeformed area for given material parameters.

class hyperelastic.lab.**Uniaxial**(*label=None*)

Incompressible uniaxial tension/compression load case.

$$\boldsymbol{F} = \operatorname{diag}\left(\begin{bmatrix}\lambda & \frac{1}{\sqrt{\lambda}} & \frac{1}{\sqrt{\lambda}}\end{bmatrix}\right) \tag{6.7}$$

defgrad(*stretch*)

Return the Deformation Gradient tensor from given stretches.

stress(*F*, *P*, *axis=0*, *traction_free=-1*)

Normal force per undeformed area for a given deformation gradient of an incompressible deformation and the first Piola-Kirchhoff stress tensor.

Parameters

- **F** (*ndarray*) – The deformation gradient.

- **P** (*ndarray*) – The first Piola-Kirchhoff stress tensor.
- **axis** (*int, optional*) – The primary axis where the longitudinal stretch is applied on (default is 0).
- **traction_free** (*int, optional*) – The secondary axis where the traction-free transverse stretch results from the constraint of incompressibility (default is -1).

Returns
The one-dimensional normal force per undeformed area.

Return type
ndarray

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

I

L

M

N

O

P

S

T

U