Material Mechanics Documentation

Release 0.0.1

Kevin Michael Engel

Nov 06, 2018

Contents:

CHAPTER 1

Material Mechanics

The "Material Mechanics" package contains tools needed in the analysis of the mechanics of materials, including fiber reinforced materials and laminates.

- Free software: MIT license
- Documentation: https://material-mechanics.readthedocs.io.

1.1 Installation

To install simply use pip

```
>>> pip install material_mechanics
```

1.2 Features

Materials:

- Isotropic materials
- Transverse isotropic materials
- Orthotropic materials
- fiber reinforced plastics (FRP)
- Laminates

Analytics:

- Stiffness analysis
- **fracture mechanics of FRP**
  - Puck 2D and 3D

- Classical Lamination Theory (CLT)

1.3 Roadmap

Materials

- Non linear material laws

Analytics

- Fracture mechanics for isotropic materials (von Mises Stress)
- **Addition of damage criteria for FRP**
  - strain criteria for whole FRP laminates
  - Tsai-Wu criterion
- integration of fatigue damage analysis

1.4 Usage

1.5 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Material Mechanics, run this command in your terminal:

```
$ pip install material_mechanics
```

This is the preferred method to install Material Mechanics, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Material Mechanics can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/kemeen/material_mechanics
```

Or download the tarball:

```
$ curl  -OL https://github.com/kemeen/material_mechanics/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Material Mechanics in a project:

```
import material_mechanics as mm
```

3.1 Creating materials

Creating an orthotropic elastic material:

```
import material_mechanics as mm

name = 'Orthotropic Material'
stiffnesses = dict(e1=100000, e2=8000, e3=7000, g12=5000, g13=5000, g23=4000)
poissons = dict(nu12=0.33, nu21=0.02, nu13=0.33, nu31=0.02, nu23=0.33, nu32=0.02)
density = 1.0

material = mm.orthotropic_material(
    name=name, stiffness=stiffnesses, poisson=poissons, density=density
)
```

Creating a transverse isotropic material:

```
import material_mechanics as mm

name = 'cfrp'
stiffnesses = dict(e1=140000, e2=9000, g12=4600)
poissons = dict(nu12=0.3, nu23=0.37)
strengths = None
density = 1.5

material = mm.transverse_isotropic_material(
    name=name, stiffness=stiffnesses, poisson=poissons,
```

(continues on next page)

(continued from previous page)

```
    strength=strengths, density=density
)
```

Creating a fiber reinforced material:

```
import material_mechanics as mm

# fiber definition
name = 'Carbon Fiber HT'
stiffness = dict(e1=230000, e2=13000, g12=50000)
poisson = dict(nu12=0.23, nu23=0.3)
fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)

# matrix definition
name = 'Epoxy Resin'
matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3, density=1.2)

# fiber volume content
phi = 0.65

# fiber reinfored material initialization
frp_material = mm.FiberReinforcedPlastic(
    fiber_material=fiber, matrix_material=matrix, fiber_volume_fraction=phi
)
```

In this example we are using a factory that allows for easy laminate creation using a single material. First we create a FRP that we want to use as the material for each layer

Forst we initialize the FRP material:

```
import material_mechanics as mm

# fiber definition
name = 'Carbon Fiber HT'
stiffness = dict(e1=230000, e2=13000, g12=50000)
poisson = dict(nu12=0.23, nu23=0.3)
fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)

# matrix definition
name = 'Epoxy Resin'
matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3, density=1.2)

# fiber volume content
phi = 0.65

# fiber reinfored material initialization
frp_material = mm.FiberReinforcedPlastic(
    fiber_material=fiber, matrix_material=matrix, fiber_volume_fraction=phi
)
```

Now we will initialize the factory that will create the laminates:

```
# Laminate Factory initialization
LaminateCreator = mm.SingleMaterialLaminateFactory(frp_material)
```

And finally we create a laminate by providing a stacking order:

```
stacking_order = [(0.25, 0),(0.5, 30),(0.4, 60),(0.1, 90)]
laminate = LaminateCreator.get_laminate(stacking)
```

If we want a symmetric laminate, we can define the symmetry plane by the keyword 'symmetry' to either be at the center of the last layer (*'center_layer'*) of the initial stacking or at the bottom of the last provided layer (*'full'*):

```
# symmetric laminate with the center plane of the last layer (0.1, 90) as the␣
↪laminates symmetry plane
laminate = LaminateCreator.get_laminate(stacking, symmetry='center_layer')

# symmetric laminate with the bottom plane of the last layer (0.1, 90) as the␣
↪laminates symmetry plane
laminate = LaminateCreator.get_laminate(stacking, symmetry='full')
```

3.2 Strength analysis

Applying a load and calculating the puck material exertions of a laminate requires to provide the strength of the material of the layer material. in the case of fiber reinforced material five strength parameters are needed. - tensile strength in fiber direction (11_tensile) - compression strength in fiber direction (11_compression) - tensile strength prependicular to the fiber direction (22_tensile) - tensile strength in fiber direction (22_compression) - shear strength under parallel/perpendicular stress (12)

Creating the Laminate:

```
import material_mechanics as mm
import numpy as np

# fiber definition
name = 'Carbon Fiber HT'
stiffness = dict(e1=230000, e2=13000, g12=50000)
poisson = dict(nu12=0.23, nu23=0.3)
fiber = pm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)

# matrix definition including it's strength
name = 'Epoxy Resin'
matrix = pm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3, density=1.2,␣
↪strength=90.0)

# definition of composite material strength at target fiber volume ratio
strength_dict = dict(
    r_11_tensile=2000.0, r_11_compression=1650.0,
    r_22_tensile=70., r_22_compression=240.,
    r_12=105,
)

# fiber reinfored material initialization
frp_material = pm.FiberReinforcedPlastic(
    fiber_material=fiber(), matrix_material=matrix, fiber_volume_fraction=phi,␣
↪name=None, symmetry='mean'
)

# Laminate definition
LaminateCreator = pm.SingleMaterialLaminateFactory(frp_material)
```

(continues on next page)

(continued from previous page)

```
stacking_order = [(0.25, 0),(0.25, 45),(0.25, 90),(0.25, -45)]
laminate = LaminateCreator.get_laminate(stacking, symmetry='full')
```

Now all that is left to do is to define a load vector and to calculate the results. The load is defined as line loads and line moments with six entries. The damage criterion used when analysing a laminate is the puck2D criterion. The provided load vector consist of the following entries: (n_{xx}, n_{yy}, n_{xy}, m_{xx}, m_{yy}, m_{xy}), where n_{ij} are the line loads and m_{ij} the line moments

Strength analysis:

```
line_load = np.array([250, 34, 55, 4, 34, 11])
max_fb, max_zfb, laminate_exertions = puck.get_laminate_exertions(laminate=fzb_lam,␣
↪line_loads=line_load)
```

The result is a tuple holding the maximum fiber-exertion and inter-fiber-exertion in any layer of the laminate and a list holding a dict for every layer in the laminate with detailed information about that layer, including damage indicators.

CHAPTER 4

Modules

4.1 materials

4.1.1 composites

class material_mechanics.materials.composites.**FiberReinforcedPlastic**(*fiber_material*, *matrix_material*, *fiber_volume_fraction*, **args*, ***kwargs*)

defines a fiber reinforced material consisting of two constituents, fiber and embedding matrix

both materials need to be of a material type derived from *ElasticMaterial*

Parameters

- **fiber_material** (*pymat material*) – fiber material
- **matrix_material** (*pymat material*) – matrix material
- **fiber_volume_fraction** (*float*) – volume fraction of fiber material
- **name** (*str*) – (*optional*) material name. Default is None
- **strength** (*dict of floats*) – (*optional*) material strength values of the composite at the provided fiber volume ratio. Default is None

Example:

```
>>> import material_mechanics as mm
>>>
>>> # fiber definition
>>> name = 'Carbon Fiber HT'
>>> stiffness = dict(e1=230000, e2=13000, g12=50000)
>>> poisson = dict(nu12=0.23, nu23=0.3)
```

(continues on next page)

(continued from previous page)

```
>>> fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)
>>>
>>> # matrix definition
>>> name = 'Epoxy Resin'
>>> matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3,␣
↪density=1.2)
>>>
>>> # fiber volume content
>>> phi = 0.65
>>>
>>> # fiber reinforced material initialization
>>> frp_material = mm.FiberReinforcedPlastic(
>>>     fiber_material=fiber, matrix_material=matrix, fiber_volume_fraction=phi
>>> )
>>> print(frp_material)
Name: CarbonFiberHT_EpoxyResin_65, fiber volume fraction: 0.65, Fiber: Carbon␣
↪Fiber HT, Matrix: Epoxy Resin'
```

fiber_material
return the fiber material of the fiber reinforced material

Returns fiber material

Return type *ElasticMaterial* or derived class

fiber_volume_fraction
return the fiber volume ratio of the fiber reinforced material

Returns fiber volume ratio

Return type float

matrix_material
return the matrix material of the fiber reinforced material

Returns matrix material

Return type *ElasticMaterial* or derived class

class material_mechanics.materials.composites.**Laminate**(**args*, ***kwargs*)
Creates a laminate object with an empty stacking

Note: a factory class for creating laminates exists and it's use for creating laminates is encouraged (see *StandardLaminateFactory*)

Parameters

- **name** (*str*) – (*optional*) name of the laminate, default is None
- **layer_stiffness_symmetry** (*str or None*) – (*optional*) sets the method of enforcement of symmetry in the layer materials stiffness matrix. For details on the algorithm for symmetry enforcement please see *force_symmetry()*. Default is 'upper'. Options: ('mean', 'upper', 'lower', None)

Example:

```
>>> import material_mechanics as mm
>>>
>>> # fiber definition
>>> name = 'Carbon Fiber HT'
>>> stiffness = dict(e1=230000, e2=13000, g12=50000)
>>> poisson = dict(nu12=0.23, nu23=0.3)
>>> fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)
>>>
>>> # matrix definition
>>> name = 'Epoxy Resin'
>>> matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3,␣
↪density=1.2)
>>>
>>> # fiber volume content
>>> phi = 0.65
>>>
>>> # fiber reinforced material initialization
>>> frp_material = mm.FiberReinforcedPlastic(
>>>     fiber_material=fiber, matrix_material=matrix, fiber_volume_fraction=phi
>>> )
>>>
>>> lam_fzb = mm.Laminate()
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=45.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=90.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=135.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=0.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=0.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=135.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=90.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=45.))
```

a_matrix
return the shell stiffness matrix of the laminate as defined in the classical lamination theory

Returns shell stiffness matrix

Return type numpy.ndarray

abd_matrix
return the abd Matrix of the Laminate using classical lamination theory

Returns stiffness matrix of the combined shell and plate element

Return type numpy.ndarray

add_layer(*layer*, **args*, ***kwargs*)
add a layer to the laminate

Parameters **layer** (Layer) – layer to add to the end of the laminate

Returns None

Example:

```
>>> import material_mechanics as mm
>>>
>>> # fiber definition
>>> name = 'Carbon Fiber HT'
>>> stiffness = dict(e1=230000, e2=13000, g12=50000)
>>> poisson = dict(nu12=0.23, nu23=0.3)
>>> fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)
>>>
>>> # matrix definition
>>> name = 'Epoxy Resin'
>>> matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3,␣
↪density=1.2)
>>>
>>> # fiber volume content
>>> phi = 0.65
>>>
>>> # fiber reinforced material initialization
>>> frp_material = mm.FiberReinforcedPlastic(
>>>     fiber_material=fiber, matrix_material=matrix, fiber_volume_
↪fraction=phi
>>> )
>>>
>>> lam_fzb = mm.Laminate()
>>> lam_fzb.add_layer(mm.Layer(name='my_layer', material=frp_material,␣
↪thickness=0.25, orientation=45.))
>>> print(lam_fzb)
1 - Name: my_layer, Material: CarbonFiberHT_EpoxyResin_65, Thickness: 0.25,␣
↪Orientation: 45.0'
```

area_weight
return the laminates area weight

Returns laminate area weight in $\frac{g}{cm^2}$

Return type float

b_matrix
Return the coupling matrix of the laminate as defined in the classical lamination theory

Returns coupling matrix

Return type numpy.ndarray

d_matrix
calculates the plate stiffness matrix of the laminate as defined in the classical lamination theory

Returns plate stiffness matrix

Return type numpy.ndarray

density
return the materials density

Returns density in $\frac{g}{cm^3}$

Return type float

get_layer_strains (*laminate_strains*)
return the strains in the layers of the laminate resulting from an external planar loading of the laminate

The laminate strains are $[\varepsilon_x, \varepsilon_y, \gamma_{xy}, \kappa_x, \kappa_y, \kappa_{xy}]$ and are the resulting global laminate strains from an external loading of the laminate, calculated by the classical laminate theory (CLT). This step is done in the *get_strains()*

Parameters **laminate_strains** – global laminate strains

Returns strains of the layers at the bottom and the top of each layer

Return type list of lists of numpy.ndarray

Example:

```
>>> import material_mechanics as mm
>>>
>>> # fiber definition
>>> name = 'Carbon Fiber HT'
>>> stiffness = dict(e1=230000, e2=13000, g12=50000)
>>> poisson = dict(nu12=0.23, nu23=0.3)
>>> fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)
>>>
>>> # matrix definition
>>> name = 'Epoxy Resin'
>>> matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3,␣
↪density=1.2)
>>>
>>> # fiber volume content
>>> phi = 0.65
>>>
>>> # fiber reinforced material initialization
>>> frp_material = mm.FiberReinforcedPlastic(
>>>     fiber_material=fiber, matrix_material=matrix, fiber_volume_
↪fraction=phi
>>> )
>>>
>>> lam_fzb = mm.Laminate()
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=45.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=90.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=135.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=0.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=0.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=135.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=90.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=45.))
>>>
>>> line_loads = np.array([100, 50, 10, 20, 5, 3])
>>>
>>> laminate_strains = lam_fzb.get_strains(line_loads=line_loads)
>>> layer_strains = fzb_lam.get_layer_strains(laminate_strains=laminate_
↪strains)
>>>
```

(continues on next page)

(continued from previous page)

```
>>> for i, ls in enumerate(layer_strains):
>>>     print(f'Layer {i+1}\nBottom strain:{ls[0]}, Top strain:{ls[1]}')
Layer 1
Bottom strain:[ 0.00029482 -0.00024698  0.000611  ], Top strain:[ 0.0003601  -
↪0.00010048  0.00032269]
Layer 2
Bottom strain:[ 2.91153430e-04 -3.15320675e-05 -4.60576048e-04], Top strain:[␣
↪0.00025289  0.00021852 -0.00037935]
Layer 3
Bottom strain:[ 4.60294158e-05  4.25381385e-04 -3.43704880e-05], Top␣
↪strain:[0.00019254 0.00049066 0.00025394]
Layer 4
Bottom strain:[0.00046857 0.00021463 0.00029813], Top strain:[0.00071862 0.
↪00017637 0.0002169 ]
Layer 5
Bottom strain:[0.00071862 0.00017637 0.0002169 ], Top strain:[0.00096868 0.
↪0001381  0.00013568]
Layer 6
Bottom strain:[0.00048555 0.00062123 0.00083057], Top strain:[0.00063206 0.
↪00068651 0.00111889]
Layer 7
Bottom strain:[ 9.98395044e-05  1.21872905e-03 -5.44556546e-05], Top␣
↪strain:[6.15767194e-05 1.46878128e-03 2.67684241e-05]
Layer 8
Bottom strain:[ 0.00075179  0.00077856 -0.0014072 ], Top strain:[ 0.00081708 ␣
↪0.00092507 -0.00169552]
```

get_strains (*line_loads*)

return the global laminate strains resulting from a planar external loading.

The laminate strains are calculated through the laminates ABD matrix. The external load is defined by the line load vector $[n_x, n_y, n_{xy}, m_x, m_y, m_{xy}]$. The line loads n are given in $\frac{N}{mm}$ and the line moments m in N

Parameters **line_loads** (*numpy.ndarray*) – line load vector

Returns laminate strains

Return type numpy.ndarray

Example:

```
>>> import material_mechanics as mm
>>>
>>> # fiber definition
>>> name = 'Carbon Fiber HT'
>>> stiffness = dict(e1=230000, e2=13000, g12=50000)
>>> poisson = dict(nu12=0.23, nu23=0.3)
>>> fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)
>>>
>>> # matrix definition
>>> name = 'Epoxy Resin'
>>> matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3,␣
↪density=1.2)
>>>
>>> # fiber volume content
>>> phi = 0.65
```

(continues on next page)

(continued from previous page)

```
>>>
>>> # fiber reinforced material initialization
>>> frp_material = mm.FiberReinforcedPlastic(
>>>     fiber_material=fiber, matrix_material=matrix, fiber_volume_
↪fraction=phi
>>> )
>>>
>>> lam_fzb = mm.Laminate()
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=45.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=90.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=135.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=0.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=0.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=135.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=90.))
>>> lam_fzb.add_layer(mm.Layer(material=frp_material, thickness=0.25,␣
↪orientation=45.))
>>>
>>> line_loads = np.array([100, 50, 10, 20, 5, 3])
>>>
>>> lam_fzb.get_strains(line_loads=line_loads)
array([ 0.00071862,  0.00017637,  0.0002169 ,  0.00100021, -0.00015305, -0.
↪0003249 ])
```

layer_count
Return the number of layers in the laminate

Returns number of layers

Return type int

thickness
Return the laminates thickness

Returns laminate thickness in mm

Return type float

z_values
return the lower and top coordinate of each layer

Returns List of tupels. Each tuple holds the upper an lower coordinate of the layer in reference to the laminate central plane

Return type list of tuples

class material_mechanics.materials.composites.**Layer**(*material*, *thickness*, *orientation*, **args*, ***kwargs*)

Layer class to be used in the Laminate class

Parameters

- **material** (*ElasticMaterial or derived class*) – The material of the layer

- **thickness** (*float*) – thickness of the layer in mm
- **orientation** (*float*) – orientation of the layer in $degrees$
- **name** (*str or None*) – (*optional*) name to reference the layer by

Example:

```
>>> import material_mechanics as mm
>>>
>>> # fiber definition
>>> name = 'Carbon Fiber HT'
>>> stiffness = dict(e1=230000, e2=13000, g12=50000)
>>> poisson = dict(nu12=0.23, nu23=0.3)
>>> fiber = mm.transverse_isotropic_material(name=name, stiffness=stiffness,␣
↪poisson=poisson, density=1.74)
>>>
>>> # matrix definition
>>> name = 'Epoxy Resin'
>>> matrix = mm.isotropic_material(name=name, stiffness=3200.0, poisson=0.3,␣
↪density=1.2)
>>>
>>> # fiber volume content
>>> phi = 0.65
>>>
>>> # fiber reinforced material initialization
>>> frp_material = mm.FiberReinforcedPlastic(
>>>     fiber_material=fiber, matrix_material=matrix, fiber_volume_fraction=phi
>>> )
>>> layer = mm.Layer(name='my_layer',material=frp_material, thickness=0.25,␣
↪orientation=45.)
>>> print(layer)
Name: my_layer, Material: CarbonFiberHT_EpoxyResin_65, Thickness: 0.25,␣
↪Orientation: 45.0
```

area_weight
return the area weight

Returns area weight of the layer in $\frac{g}{cm^2}$

Return type float

compliance_matrix(**args*, ***kwargs*)
calculate the compliance matrix of the layer in the laminate coordinate system (i.e. in the 0° direction)

Returns layer compliance matrix

Return type numpy.ndarray

material
return layer material

Returns layer material

Return type *ElasticMaterial* or derived class

name
return the name of the layer

Returns layer name

Return type str

orientation
return layer orientation

Returns layer orientation in $degree$

Return type float

stiffness_matrix(**args*, ***kwargs*)
calculates the stiffness matrix of the layer in the laminate coordinate system

Returns layer stiffness matrix

Return type numpy.ndarray

thickness
return layer thickness

Returns layer thickness in mm

Return type float

4.1.2 elastic_materials

class material_mechanics.materials.elastic_materials.**ElasticMaterial**(*stiffness*, *poisson*, **args*, ***kwargs*)

Creates an elastic material

Parameters

- **stiffness** (*list*) – stiffness values of the material in $\frac{N}{mm^2}$
- **poisson** (*list*) – poisson ratios of the material
- **strength** (*list*) – (*optional*) strength values of the material in $\frac{N}{mm^2}$, default is None
- **density** (*float*) – (*optional*) material density in $\frac{g}{cm^3}$, default is None

Example:

```
import material_mechanics as mm

stiffness = [(10000, 3000),(8000, 2500),(5000, 1800)]
poisson = [(0.3, 0.02),(0.28, 0.02),(0.34, 0.34)]

mat = mm.materials.elastic_materials.ElasticMaterial(stiffness=stiffness,␣
↪poisson=poisson)
```

This class is not available at the top level since the use of the factory functions is encouraged. The same result as in the above example can be achieved by using the function *orthotropic_material()*

compliance_matrix
Return the materials compliance matrix for three-dimensional stress states

Returns compliance matrix (6x6)

Return type numpy.ndarray

compliance_matrix_2d
Return the materials compliance matrix for two-dimensional stress states

Returns compliance matrix (3x3)

Return type numpy.ndarray

density
return the material density :return: density :rtype: float

get_poisson(*index=None*, **args*, ***kwargs*)
return a poisson ratio of the material in the requested direction

Parameters

- **index** (*int, str or None*) – (*optional*) index of the requested poisson ratio, default is None, returning the major poisson ratio in direction ν_{12}

  Options:

  – ν_{23}: (23, '23', 'yz', 'YZ')

  – ν_{32}: (32, '32', 'zy', 'ZY')

  – ν_{13}: (13, '13', 'xz', 'XZ')

  – ν_{31}: (31, '31', 'zx', 'ZX')

  – ν_{23}: (12, '12', 'xy', 'XY', None)

  – ν_{32}: (21, '21', 'yx', 'YX')

- **precision** (*int*) – number of decimal points of the requested poisson ratio

Returns requested poisson ratio

Return type float

Note: poisson ratios are in international notation. The first index points to the causing strain. The second index points to the resulting strain.

Example:

```
import material_mechanics as mm

stiffness = [(10000, 3000),(8000, 2500),(5000, 1800)]
poisson = [(0.3, 0.02),(0.28, 0.02),(0.34, 0.34)]

mat = mm.materials.elastic_materials.ElasticMaterial(stiffness=stiffness,␣
↪poisson=poisson)

nu12 = mat.get_poisson()
nu12 = mat.get_poisson(12)
nu32 = mat.get_poisson('32')
```

get_stiffness(*index=None*, **args*, ***kwargs*)
return a stiffness value of the material in the requested direction

Parameters

- **index** (*int, str or None*) – (*optional*) index of the requested stiffness value, default is None, returning the major stiffness in direction '11'

  Options:

  – E_{11}: (1, 11, '1', '11', None, 'x', 'X')

  – E_{22}: (2, 22, '2', '22', 'y', 'Y')

- E_{33}: (3, 33, '3', '33', 'z', 'Z')
- G_{23}, G_{32}: (23, 32, '23', '32', 'yz', 'YZ', 'zy', 'ZY')
- G_{13}, G_{32}: (13, 31, '13', '31', 'xz', 'XZ', 'zx', 'ZX')
- G_{12}, G_{21}: (12, 21, '12', '21', 'xy', 'XY', 'yx', 'YX')

- **precision** (*int*) – number of decimal points of the requested stiffness

Returns requested stiffness

Return type float

Example:

```
import material_mechanics as mm

stiffness = [(10000, 3000),(8000, 2500),(5000, 1800)]
poisson = [(0.3, 0.02),(0.28, 0.02),(0.34, 0.34)]

mat = mm.materials.elastic_materials.ElasticMaterial(stiffness=stiffness,␣
↪poisson=poisson)

e1 = mat.get_stiffness()
e1 = mat.get_stiffness(11)
g12 = mat.get_stiffness('12')
```

get_strength(*index=None*, *direction=None*, **args*, ***kwargs*)

return strength value of the material in the requested direction

Parameters

- **index** (*int, str or None*) – (*optional*) index of the requested stiffness value, default is None, returning the major stiffness in direction '11'

  Options:

  - $R_{11}^{+/-}$: (1, 11, '1', '11', None, 'x', 'X')
  - $R_{22}^{+/-}$: (2, 22, '2', '22', 'y', 'Y')
  - $R_{33}^{+/-}$: (3, 33, '3', '33', 'z', 'Z')
  - R_{23}: (23, '23', 'yz', 'YZ')
  - R_{32}: (32, '32', 'zy', 'ZY')
  - R_{13}: (13, '13', 'xz', 'XZ')
  - R_{31}: (31, '31', 'zx', 'ZX')
  - R_{12}: (12, '12', 'xy', 'XY')
  - R_{21}: (21, '21', 'yx', 'YX')

- **direction** (*str or None*) – (*optional*) tensile or compression, default is None (resulting in tensile strength values)

  Options:

  - tensile: (1, '1', '+', 'tensile', 't', 'plus', 'p', 'positive', 'pos', None)
  - compression: (-1, '-1', '-', 'compression', 'c', 'minus', 'm', 'negative', 'neg', 'n')

- **precision** (*int or None*) – (*optional*) number of decimal points of the requested stiffness

Returns requested stiffness

Return type float

Example:

```
import material_mechanics as mm

stiffness = [(10000, 3000),(8000, 2500),(5000, 1800)]
poisson = [(0.3, 0.02),(0.28, 0.02),(0.34, 0.34)]

mat = mm.materials.elastic_materials.ElasticMaterial(stiffness=stiffness,␣
↪poisson=poisson)

r1 = mat.get_strength()
r1 = mat.get_strength(11)
r12 = mat.get_strength('12')
```

stiffness_matrix
Return the materials stiffness matrix for three-dimensional stress states

Returns stiffness matrix (6x6)

Return type numpy.ndarray

stiffness_matrix_2d
Return the materials stiffness matrix for two-dimensional stress states

Returns stiffness matrix (3x3)

Return type numpy.ndarray

4.1.3 material_factories

class material_mechanics.materials.material_factories.**ChangedFvcFrp**(*material*, **args*, ***kwargs*)
A factory class to get an FRP material from an existing FRP with changed fiber volume content

Parameters **material** (FiberReinforcedPlastic) – the base FRP Material from which to calculate the generated materials

get_material(*phi*, **args*, ***kwargs*)
return a fiber reinforced plastic with the same constituents as the base fiber reinforced plastic but changed fiber volume content

Parameters **phi** (*float*) – fiber volume content of the new fiber reinforced plastic

Returns a fiber reinforced plastic material with the provided fiber volume content

Return type *FiberReinforcedPlastic*

class material_mechanics.materials.material_factories.**ChangedFvcPuckSet**(*puck_set*, *resin_strength*)
A factory class to generate puck sets from an existing puck set with changed fiber volume content

Parameters **puck_set** (PuckStrengthSet) – the base puck set from which to generate new puck sets

get_material(*phi*, **args*, ***kwargs*)
returns a puck set with changed fiber volume content based on the puck set provided at initiation

Parameters **phi** (*float*) – fiber volume content of the new puck set

Returns puck set with the provided fiber volume content

Return type *PuckStrengthSet*

class material_mechanics.materials.material_factories.**StandardLaminateFactory**(**args*, ***kwargs*)

Factory class for creating laminates

Parameters

- **material** (*ElasticMaterial or derived*) – (*optional*) Default material for each layer of the laminate. May be overwritten in layer dicts. Default is None
- **layer_thickness** (*float or None*) – (*optional*) Default layer thickness for the laminate. May be overwritten in layer dicts. Default is None
- **symmetry** (*str or None*) – (*optional*) symmetry mode of laminate, default is None

  Options:

  – None: No symmetry is forced

  – 'center_layer': The symmetry plane of the last layer is the symmetry plane for the laminate, so all but the last layer are mirrored by this plane

  – 'full': (*catches all strings but 'center_layer'*) all layers are mirrored at the plane defining the lower border of the last plane, including the last layer

get_laminate(*layers*, **args*, ***kwargs*)

Return a Laminate

Parameters

- **layers** (*list of dicts*) – stacking of a laminate defined as dicts. each dict holds the parameters of the layer ('thickness' in mm, 'orientation' in $degree$, 'material'). 'thickness' and 'material' may be set at the initiation of the class, in which case they may be omitted in the layer dicts. If provided at method call, init values of parameters will be overwritten.
- **symmetry** (*str or None*) – (*optional*) symmetry mode of laminate, default is None

  Options:

  – None: No symmetry is forced

  – 'center_layer': The symmetry plane of the last layer is the symmetry plane for the laminate, so all but the last layer are mirrored by this plane

  – 'full': (*catches all strings but 'center_layer'*) all layers are mirrored at the plane defining the lower border of the last plane, including the last layer

Returns Laminate

Return type *Laminate*

class material_mechanics.materials.material_factories.**StandardLayerFactory**(**args*, ***kwargs*)

Standard Factory for Layers

Standard values for the parameters may be provided at initiation. Parameters that did not receive a value at initiation, need to be set when the get_layer method is called. If not a ValueError is raised.

Parameters

- **material** (*ElasticMaterial or None*) – Layer material
- **thickness** (*float or None*) – Thickness of the layer in mm

- **orientation** (*float or None*) – Orientation of the Layerin *degree*

Note: All materials derived from *ElasticMaterial*, like *FiberReinforcedPlastic*, can be used in a Layer.

Example:

```
import material_mechanics as mm

stiffness = dict(e1=140000, e2=9000, g12=4600)
poisson = dict(nu12=0.3, nu23=0.37)
density = 1.5

cfk = mm.transverse_isotropic_material(
    name='CFK',
    stiffness=stiffness,
    poisson=poisson,
    strength=None,
    density=density)

steel = mm.isotropic_material(
    name='Steel',
    stiffness=200000.0,
    poisson=0.34,
    strength=700,
    density=7.9
)

FixedMaterialLayerCreator = mm.StandardLayerFactory(material=cfk)
layer_1 = FixedMaterialLayerCreator.get_layer(thickness=0.1, orientation=30)
layer_2 = FixedMaterialLayerCreator.get_layer(thickness=0.1, orientation=60)
layer_3 = FixedMaterialLayerCreator.get_layer(thickness=0.1, orientation=90)


FixedThicknessLayerCreator = mm.StandardLayerFactory(thickness=0.2)
layer_1 = FixedThicknessLayerCreator.get_layer(material=steel, orientation=30)
layer_2 = FixedThicknessLayerCreator.get_layer(material=cfk, orientation=45)
layer_3 = FixedThicknessLayerCreator.get_layer(material=steel, orientation=60)

# standard values may also be overwritten
layer_3 = FixedThicknessLayerCreator.get_layer(thickness=0.1, material=steel,␣
↪orientation=60)
```

get_layer(**args*, ***kwargs*)

Return a layer with a material, thickness and orientation.

If the values for material, thickness and/or orientation where provided at initiation, they may be omitted or be overwritten.

Parameters

- **material** (*ElasticMaterial*) – (*optional*) Layer material, default is material provided at initialization of class
- **thickness** (*float*) – (*optional*) Thickness of the layer, default is thickness provided at initialization of class
- **orientation** (*float*) – (*optional*) Orientation of the Layer, default is orientation provided at initialization of class

Returns Layer

Return type *Layer*

material_mechanics.materials.material_factories.**isotropic_material**(*stiffness*, *poisson*, *density*, **args*, ***kwargs*)

create a material with isotropic properties

Parameters

- **density** (*float*) – material density in $\frac{g}{cm^3}$
- **stiffness** (*float*) – material stiffness in $\frac{N}{mm^2}$
- **poisson** (*float*) – material poisson's ratio
- **strength** (*float*) – material strength in $\frac{N}{mm^2}$

Returns material with isotropic properties

Return type *ElasticMaterial*

material_mechanics.materials.material_factories.**orthotropic_material**(*stiffness*, *poisson*, *density*, **args*, ***kwargs*)

create a material with orthotropic properties

Note: poisson ratios are in international notation. The first index points to the causing strain. The second index points to the resulting strain.

Parameters

- **density** (*float*) – material density in $\frac{g}{cm^3}$
- **stiffness** (*dict*) – dictionary of material stiffness in $\frac{N}{mm^2}$
- **poisson** (*dict*) – material poisson ratios
- **strength** (*dict*) – dictionary of material strengths in $\frac{N}{mm^2}$

Returns material with isotropic properties

Return type *ElasticMaterial*

material_mechanics.materials.material_factories.**transverse_isotropic_material**(*stiffness*, *poisson*, *density*, **args*, ***kwargs*)

create a material with transverse isotropic properties

Note: poisson ratios are in international notation. The first index points to the causing strain. The second index points to the resulting strain.

Parameters

- **density** (*float*) – material density in $\frac{g}{cm^3}$
- **stiffness** (*dict*) – dictionary of material stiffnesses in $\frac{N}{mm^2}$
- **poisson** (*dict*) – material poisson ratios
- **strength** (*dict*) – dictionary of material strengths in $\frac{N}{mm^2}$

Returns material with isotropic properties

Return type *ElasticMaterial*

4.1.4 material_law

This module holds possible material laws

class material_mechanics.materials.material_law.**HookesLaw**(*stiffness*, *poisson*, **args*, ***kwargs*)

representation of Hookes Law of elasticity for two- and three-dimensional stress states in materials

Note: poisson ratios are in international notation. The first index points to the causing strain. The second index points to the resulting strain.

Parameters

- **stiffness** (*List or tuple of lists or tuples of floats*) – List or tuples of pairs of stiffness values. Each pair holds the stiffness in a major axis of the material and the shear stiffness in the plane perpendicular to that axis. ((σ_{11}, G_{23}) for the 1-direction)
- **poisson** (*List or tuple of lists or tuples of floats*) – List or tuples of pairs of poisson ratios. each pair holds the two poisson ratios perpendicular to a major material axis. So the first entry holds (ν_{23}, ν_{32}).

compliance_matrix

returns the compliance matrix for three-dimensional stress states

Returns compliance matrix (6x6)

Return type numpy.ndarray

compliance_matrix_2d

returns the compliance matrix for two-dimensional stress states

Returns compliance matrix (3x3)

Return type numpy.ndarray

stiffness_matrix

returns the stiffness matrix for three-dimensional stress states

Returns stiffness matrix (6x6)

Return type numpy.ndarray

stiffness_matrix_2d
returns the stiffness matrix for two-dimensional stress states

Returns stiffness matrix (3x3)

Return type numpy.ndarray

4.2 strength

4.2.1 puck

This module provides all necessary tools to calculate the two- and three dimensional puck material exertions of fiber reinforced plastics. For further reading and details on the theory please check out the refereed literature.

Literature

class material_mechanics.strength.puck.**PuckStrengthSet**(*material*, **args*, ***kwargs*)
Class providing the necessary methods to calculate the Puck Criterion

For further details please check *[Sch07] [Puc04] [Puc02]*

Parameters

- **material** (FiberReinforcedPlastic) – composite material for the puck strength criterion
- **p_plp** (*float*) – (*optional*) slope parameter $p^{+}_{\perp\parallel}$, default: 0.27
- **p_plm** (*float*) – (*optional*) slope parameter $p^{-}_{\perp\parallel}$, default: 0.27
- **p_ppp** (*float*) – (*optional*) slope parameter $p^{+}_{\perp\perp}$, default: 0.3
- **p_ppm** (*float*) – (*optional*) slope parameter $p^{-}_{\perp\perp}$, default: 0.35
- **m_sf** (*float*) – (*optional*) magnification factor $m_{\sigma,f}$ for fiber strain ε_1 due to perpendicular stress (σ_2, σ_3), default: 1.1

get_fiber_exertion(*stress_vector*, **args*, ***kwargs*)
returns fiber strain for provided stress vector

Parameters

- **stress_vector** (*numpy.ndarray*) – stress vector in fiber coordinate system (σ_{11}, σ_{22}, σ_{33}, σ_{23}, σ_{31}, σ_{21})
- **precision** (*int*) – (*optional*) number of decimal points to which to return the calculated fiber exertion, default is 2

Returns fiber strain

get_max_inter_fiber_exertion(*stress_vector*, **args*, ***kwargs*)
returns the maximum inter fiber strain and the angle of the plane in which it occurs

Parameters

- **stress_vector** (*numpy.ndarray*) – stress vector in th fiber coordinate system (σ_{11}, σ_{22}, σ_{33}, σ_{23}, σ_{31}, σ_{21})
- **precision** (*int*) – (*optional*) number of decimal points to which to return the calculated inter-fiber exertion, default is 2

- **angle_precision** (*int*) – (*optional*) number of decimal points to which to calculate the angle of the plane in which maximum strain occurs, default is 2

Returns

2-tuple

- maximum inter fiber strain
- angle of plane in which maximum strain occurs

inter_fiber_exertion(*stress_vector*, *theta*)

calculates the inter-fiber exertion in the plane defined by the angle Θ

Parameters

- **stress_vector** (*numpy.ndarray*) – stress in fiber coordinates (σ_{11}, σ_{22}, σ_{33}, σ_{23}, σ_{31}, σ_{21})
- **theta** (*float*) – angle defining the plane in radians

Returns inter fiber exertion

puck_exertion_2d(*stress_vector*, **args*, ***kwargs*)

calculates the puck exertions under the assumption of a two-dimensional stress state

For details on the meaning of the parameters s and m, please check *[Sch07] [Puc04] [Puc02]*

Parameters

- **stress_vector** (*numpy.ndarray*) – stress vector in fiber coordinate system (σ_{11}, σ_{22}, σ_{33}, σ_{23}, σ_{31}, σ_{21})
- **s** (*float*) – (*optional*) fraction of fiber exertion at which σ_{11} starts to influence damage initiation
- **m** (*float*) – (*optional*) minimal value of η_w at which damage due to fiber and inter-fiber exertion simultaniously occur.
- **ret_type** (*str*) – (*optional*) sets the return format for the results ('tuple', 'array', 'dict'), default is 'tuple'
- **precision** (*int*) – (*optional*) number of decimal points to which to return the calculated exertions, default is 2

Returns

fiber exertion (*fe_fb*), inter-fiber exertion without σ_{11} influence (*fe_0*), inter-fiber exertion including σ_{11} influence (*fe_1*), fracture mode (*mode*)

formats:

- 'tuple' (*default*): tuple(fe_fb, fe_0, fe_1, mode)
- 'array': tuple(np.array([fe_fb, fe_0, fe_1]), mode)
- 'dict': dict(fe_fb=fe_fb, fe_0=fe_0, fe_1=fe_1, mode=mode)

material_mechanics.strength.puck.**find_min_stress_angle**(*func*, **args*, ***kwargs*)

Find the minimal value of the provided function

Parameters

- **func** (*function*) – function to be minimized
- **precision** (*int*) – (optional) sets the number of decimal points to which the angle is to be calculated, default is 0

- **offset** (*float*) – (optional) sets the offset used in the provided func, default is 1.0

Returns

material_mechanics.strength.puck.**get_laminate_exertions**(*laminate*, *line_loads*, **args*, ***kwargs*)

calculates the material exertion of every layer of the laminate from the laminate loads.

The provided laminate loads (line_loads) are presumed to be line loads in $\frac{N}{mm}$ and line moments in N, defined in the laminate coordinate system. The function returns maximum values for the fiber and inter-fiber exertions of all layers as well as a list of dicts, holding details for every layer. Every dictionary in the results list has the following entries:

- fb: fiber-exertion
- zfb_0: inter_fiber exertion without the influence of fiber parallel stress
- zfb_1: inter_fiber exertion including the influence of fiber parallel stress
- mode: the fracture mode as defined by Puck
- stress: The stress in the layer in layer coordinates
- strain: The strain in the layer in layer coordinates

Parameters

- **laminate** (Laminate) – a laminate object
- **line_loads** (*numpy.ndarray*) – an array of line loads

Returns

a tuple consisting of

- maximum fiber exertion in any layer in the laminate
- maximum inter-fiber exertion in any layer in the laminate
- a list of dicts holding the results of every layer

Return type tuple

material_mechanics.strength.puck.**get_stress_transformation_matrix**(*theta*, *theta_in_deg=False*)

returns a stress transformation matrix that transforms a stress vector from fiber coordinates into a strength plane with a normal orientation perpendicular to the fiber orientation. Theta defines the angle to the perpendicular orientation in the laminate plane.

Parameters

- **theta** (*float*) – rotation angle
- **theta_in_deg** (*bool*) – flag for parameter theta. If true unit of theta is assumed to be degree if not radians.

Returns transformation matrix

Return type numpy.ndarray

4.3 tools

4.3.1 functions

material_mechanics.tools.functions.**force_symmetry**(*matrix*, *symmetry*)
Enforce symmetry in a given matrix

Parameters

- **matrix** (*numpy.ndarray*) – matrix with equal number of rows and columns
- **symmetry** (*str or None*) – method of symmetry enforcement.

  Options:

  - None: No symmetry is being enforced
  - 'upper': upper-right elements are mirrored to lower-left elements ($n_{ij} = n_{ji};\ if: i > j$)
  - 'upper': lower-left elements are mirrored to upper-right elements ($n_{ij} = n_{ji};\ if: i < j$)
  - 'upper': upper-right elements are mirrored to lower-left elements ($n_{ij} = \frac{n_{ji}+n_{ij}}{2};\ if: i \neq j$)

Returns

material_mechanics.tools.functions.**get_T_strain_2d**(*theta*)
returns a 2d strain transformation matrix that transforms a global strain vector to a strain vector in fiber coordinates with a normal orientation perpendicular to the fiber orientation. Theta defines the angle to the perpendicular orientation in the laminate plane.

Parameters **theta** – rotation angle

Returns transformation matrix (numpy.ndarray)

material_mechanics.tools.functions.**get_T_strain_3d**(*theta*)
returns a 3d strain transformation matrix that transforms a global strain vector to a strain vector in fiber coordinates with a normal orientation perpendicular to the fiber orientation. Theta defines the angle to the perpendicular orientation in the laminate plane.

Parameters **theta** – rotation angle

Returns transformation matrix (numpy.ndarray)

material_mechanics.tools.functions.**get_strain_at_z**(*global_strains*, *z*)
returns the local strain interpolated from two given stresses

Parameters

- **global_strains** –
- **z** – location at which stress should be calculated

Returns Stress at z location

material_mechanics.tools.functions.**get_stress_transformation_matrix**(*theta*)
returns a stress transformation matrix that transforms a stress vector from fiber coordinates into a strength plane with a normal orientation perpendicular to the fiber orientation. Theta defines the angle to the perpendicular orientation in the laminate plane.

Parameters **theta** – rotation angle

Returns transformation matrix (numpy.ndarray)

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/kemeen/material_mechanics/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Material Mechanics could always use more documentation, whether as part of the official Material Mechanics docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/kemeen/material_mechanics/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *material_mechanics* for local development.

1. Fork the *material_mechanics* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/material_mechanics.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv material_mechanics
$ cd material_mechanics/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 material_mechanics tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/kemeen/material_mechanics/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_material_mechanics
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Kevin Michael Engel <kevin.m.engel@gmail.com>

6.2 Contributors

- Joscha Krieglsteiner <j.krieglsteiner@tu-bs.de>
- Alexander Herwig <a.herwig@tu-bs.de>
- Onur Deniz <o.deniz@tu-bs.de>

CHAPTER 7

History

7.1 0.1.0 (2018-11-01)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Bibliography

[Sch07] *German* H. Schurmann, Konstruieren mit Faser-Kunststoff-Verbunden. Berlin, Heidelberg, 2007.

[Puc04] *English* A. Puck and H. Schürmann, "Failure analysis of FRP laminates by means of physically based phenomenological models," Fail. Criteria Fibre-Reinforced-Polymer Compos., pp. 832–876, Jan. 2004.

[Puc02] *English* [1] A. Puck, J. Kopp, and M. Knops, "Guidelines for the determination of the parameters in Puck's action plane strength criterion," Compos. Sci. Technol., vol. 62, no. 3, pp. 371–378, Feb. 2002.

Python Module Index

m

Index

H

I

L

M

N

O

P

S

T

Z