HomoPy

Nicolas Christ

Aug 07, 2023

CONTENTS

Your solution for stiffness problems!

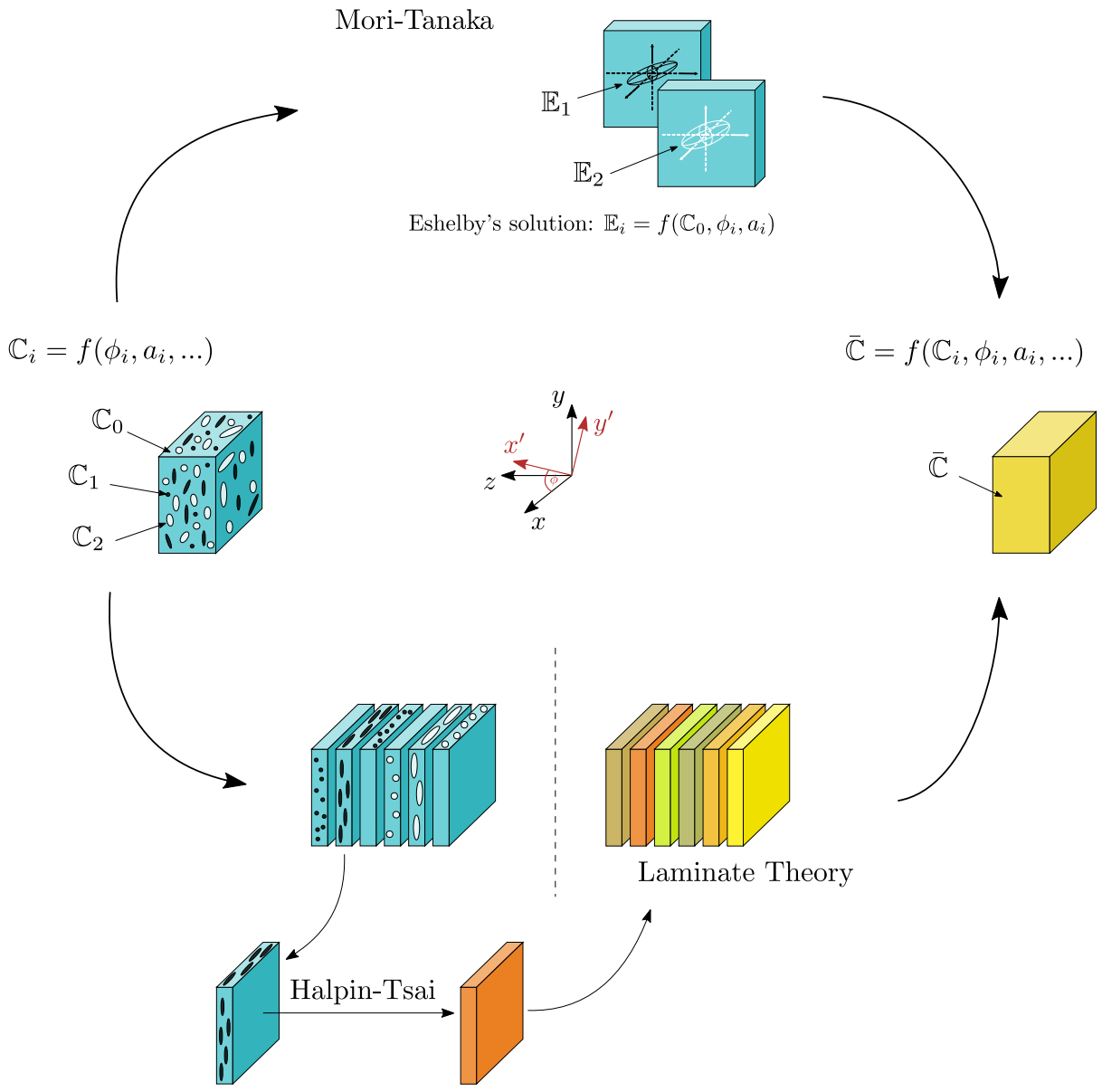


Fig. 1: **Figure 1:** Schematic of implemented homogenization methods. Inspired by [Fu1998].

CHAPTER

ONE

STATEMENT OF NEED

1.1 Statement of need

Current ecological developments call for technical solutions to reduce the ecological footprint of future innovations. Significant opportunities to promote a better eco-balance lie in the development of new material systems, particularly in the area of lightweight construction. Fiber-reinforced polymers (FRP) are a promising class of materials in the field of lightweight construction. The fibers used have high specific strength and stiffness properties, so that less material is required to achieve comparable properties in comparison with conventional materials, e.g. steel. The polymer, hereafter referred to as the matrix, is used to hold the fibers in place and transfer stresses between them. Further literature on FRP can be found in [Christensen2012] and [Chawla2019].

A general challenge in using FRP in engineering is that prediction simulations rely on robust material models. Since it is a highly inhomogeneous material, the computational cost increases dramatically if all components are modeled directly. To circumvent this, homogenization methods have been developed in the last decades. The goal of a homogenization method is to calculate the material properties of a synthetic homogeneous material, which should then effectively behave like the inhomogeneous material.

HomoPy was developed to implement two commonly used homogenization methods, namely the [Mori1973] for 3D stiffness predictions and a shear-lag modified Halpin-Tsai method (based on [Cox1952] and [Halpin1969]) for planar predictions, i.e. laminate predictions. The goal of HomoPy is to provide an open-source implementation of these methods with a particular focus on FRP modeling. Other modules are available, e.g. fiberpy, but to the author's knowledge none of them provides the capability to model hybrid FRPs consisting of different fiber materials and/or geometries. Furthermore, a major advantage in HomoPy is the implementation of the graphical representation of the effective directional stiffnesses according to [Boehlke2001]. Comparing different material systems or FRP tape layup orientations is obviously easier with a graphical representation than comparing up to 21 stiffness components each.

To this point, HomoPy is limited to calculating the effective elastic properties with the two methods mentioned above. Possible extensions for the future include thermal expansion properties and other homogenization methods.

CHAPTER

TWO

SCOPE AND LIMITATIONS

2.1 Scope and limitations

General advice

The homogenization procedures implemented in HomoPy are based on the theory of linear elasticity. As such, rate dependent effects, damage and non-linear effects in general are not taken into account and should not be modelled using HomoPy.

The scope of HomoPy is to model multi-inclusion materials with a special emphasis on fiber reinforced materials. In contrast to other implementations, HomoPy is not limited in the number of inclusion types. Nevertheless, HomoPy is limited in inclusion geometries that are implemented. So far, spheres, spheroids (short and long fibers) and needles (endless fibers) can be selected by the user. The authors would like to emphasize that combinations of different geometries are possible without limitations.

HomoPy does not imply a limitation in material symmetry of matrix material nor fiber material when using Mori-Tanaka, but so far only Isotropy and Transverse-Isotropy can be selected. In the case of the Shear-Lag modified Halpin-Tsai procedure, matrix and inclusion must be be isotropic.

Method specific information is listed below:

Mori-Tanaka

The Mori-Tanaka scheme goes back to Mori and Tanaka (cf. [Mori1973]) and is a mean-field homogenization scheme based on Eshelby's solution (cf. [Eshelby1957]). The implementation so far allows UD (needle), spheroidal and circular inclusions. Our algorithm allows to homogenize materials with different types of fibers/inclusions, each possibily having an individual orientation distrubtion. Being a tensorial homogenization scheme, the fiber orientation tensor is directly included in the calculation and the result is an effective stiffness tensor. The authors would like to emphasize that the classic formulation after [Benveniste1987] results in an effective stiffness tensor which violates thermodynamic requirements, i.e. which does not contain the major symmetry, for when multi-inclusion materials or non-isotropic inclusions are used, respectively. Further readings on this attribute are given in [Qiu1990] and [Weng1990]. To compensate this, HomoPy offers an algorithm introduced in [Segura2023], which always results in symmetric effective stiffnesses.

Halpin-Tsai

The Halpin-Tsai method is an empirical approach to homogenize two isotropic materials (cf. [Halpin1969]). Our approach is modified with the Shear-Lag model after Cox (cf. [Cox1952]), which is also used in [Fu2002] and [Fu2019]. Being a scalar homogenization scheme, it allows to calculate the effective stiffness in the plane which is orthogonal to the isotropic plane within transverse isotropic materials, as it is the case for unidirectional reinforced polymers. The effective stiffness, or Young's modulus, is then a function of the angle to the reinforcing direction. A

fiber distrubtion within the plane is recognized by volume averaging of imaginary plies of individual orientations in analogy to the laminate theory.

CHAPTER

THREE

EXAMPLES

3.1 Jupyter Notebooks

3.1.1 Introduction: A general example

[1]:
```
from homopy.methods import *
from homopy.elasticity import *
from homopy.stiffness_plot import *
```

[2]:
```
# define fiber and matrix properties
carbon_fiber = Isotropy(242e9, 0.1)
v_frac_carbon = 0.25
a_carbon = 347
glass_fiber = Isotropy(80e9, 0.22)
v_frac_glass = 0.25
a_glass = 225
polyamid6 = Isotropy(1.18e9, 0.35)
```

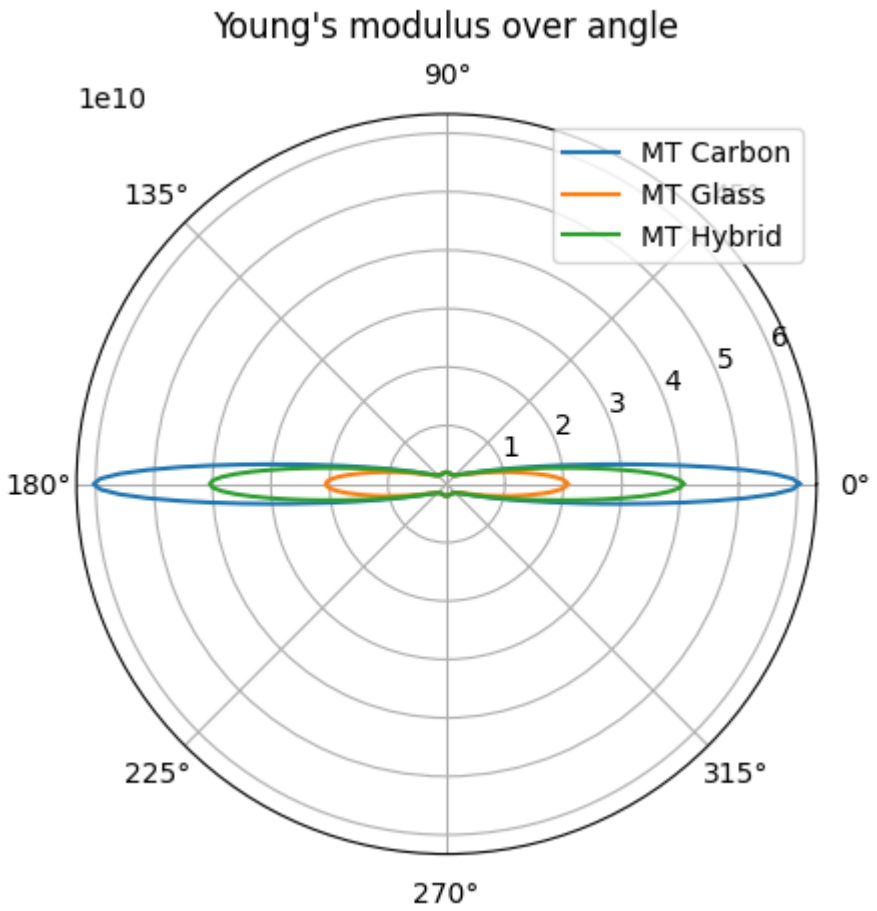
[3]:
```
# Mori Tanaka
mt_carbon = MoriTanaka(polyamid6, carbon_fiber, v_frac_carbon, a_carbon)
mt_glass = MoriTanaka(polyamid6, glass_fiber, v_frac_glass, a_glass)
mt_hybrid = MoriTanaka(
    polyamid6,
    [carbon_fiber, glass_fiber],
    [v_frac_carbon / 2, v_frac_glass / 2],
    [a_carbon, a_glass],
    2 * ["ellipsoid"],
)

# Extract effective stiffness
c_eff_carbon = mt_carbon.get_effective_stiffness()
c_eff_glass = mt_glass.get_effective_stiffness()
c_eff_hybrid = mt_hybrid.get_effective_stiffness()
```

[4]:
```
# polar plot of MT result
plotter = ElasticPlot(USEVOIGT=False)  # we highly recommend using USEVOIGT=False
```

(continues on next page)

(continued from previous page)

```
pC = plotter.polar_plot_E_body(c_eff_carbon, 400, 0, plot=False)
pG = plotter.polar_plot_E_body(c_eff_glass, 400, 0, plot=False)
pH = plotter.polar_plot_E_body(c_eff_hybrid, 400, 0, plot=False)

plotter.polar_plot([pC + ("MT Carbon",), pG + ("MT Glass",), pH + ("MT Hybrid",)])
```

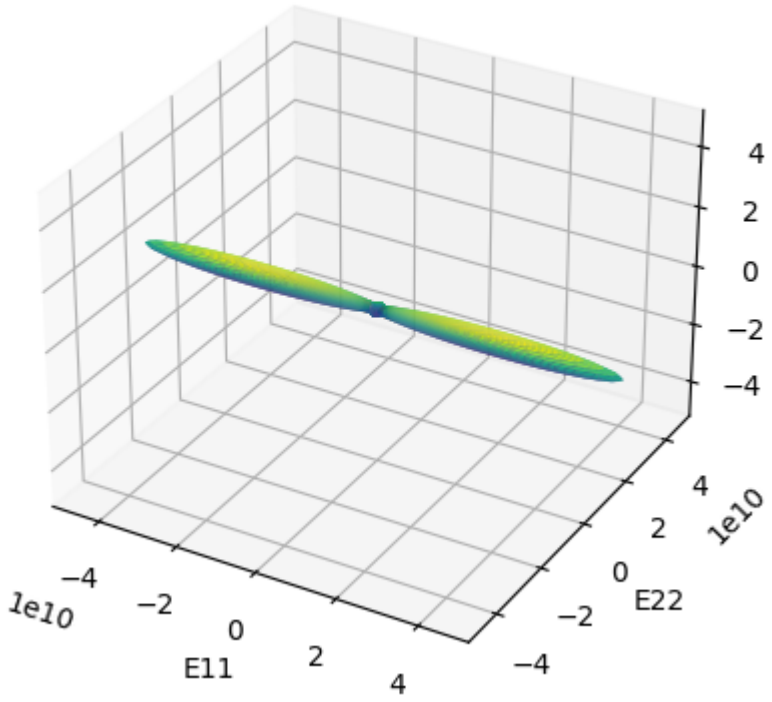


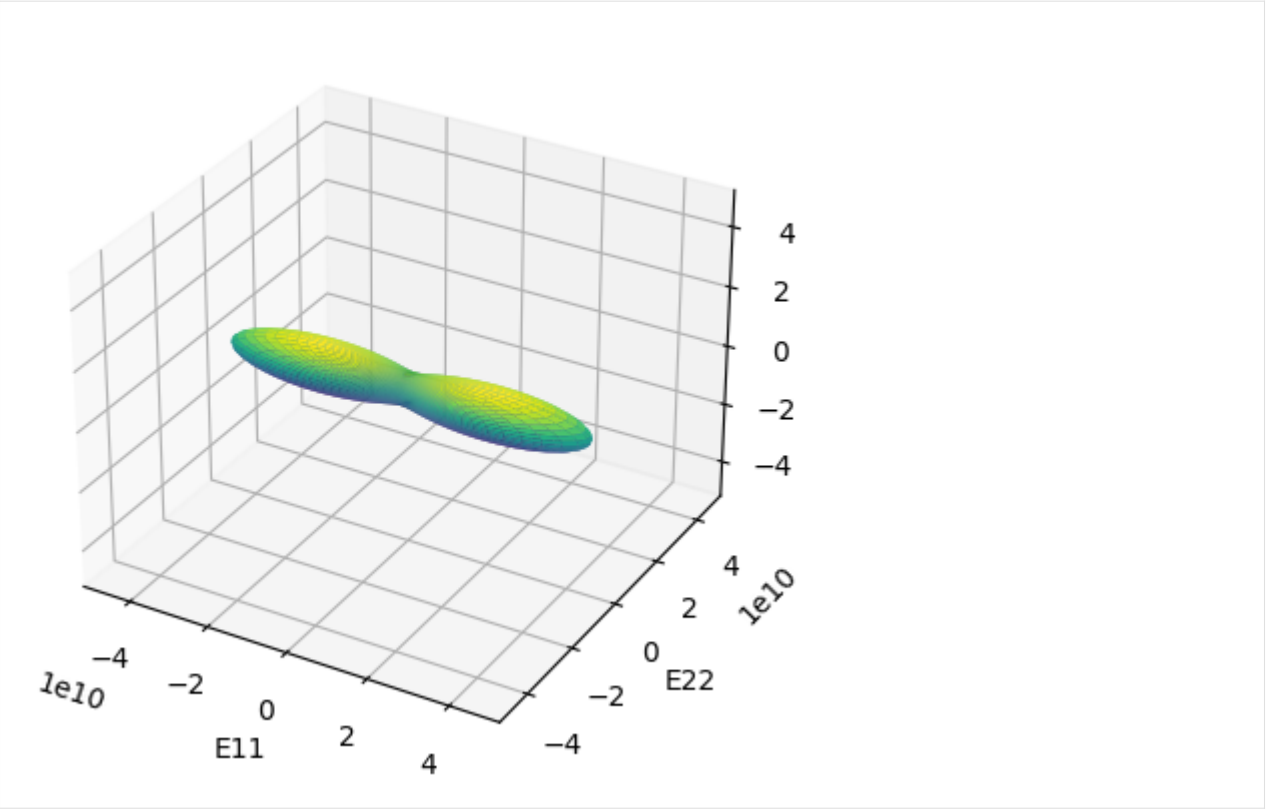
[5]:
```
# Orientation averaging (install fiberoripy through "python3 -m pip install fiberoripy"
# or use your own orientation tensor of 2nd and 4th order)
from fiberoripy.closures import (
    IBOF_closure,
    compute_closure,
    hybrid_closure,
    linear_closure,
    quadratic_closure,
)

# define arbitrary orientation tensor of 2nd order
N2 = np.zeros((3, 3))
N2[0, 0] = 26 / 32
N2[1, 1] = 3 / 32
N2[2, 2] = 3 / 32
```

(continues on next page)

(continued from previous page)

```
# use closure to estimate orientation tensor of 4th order
N4 = IBOF_closure(N2)  # IBOF_closure is recommended by the authors

# calculate average Mori-Tanaka stiffness and compliance
c_eff_carbon_ave = mt_carbon.get_average_stiffness(N4)
s_eff_carbon_ave = np.linalg.inv(c_eff_carbon_ave)

# plot stiffness body of UD reinforcement...
plotter.plot_E_body(
    c_eff_carbon, 400, 200, [5e10, 5e10, 5e10], rcount=500, ccount=500
)  # rcount and ccount should be larger than default
# and averaged reinforcement
plotter.plot_E_body(
    c_eff_carbon_ave, 400, 200, [5e10, 5e10, 5e10], rcount=200, ccount=200
)
```



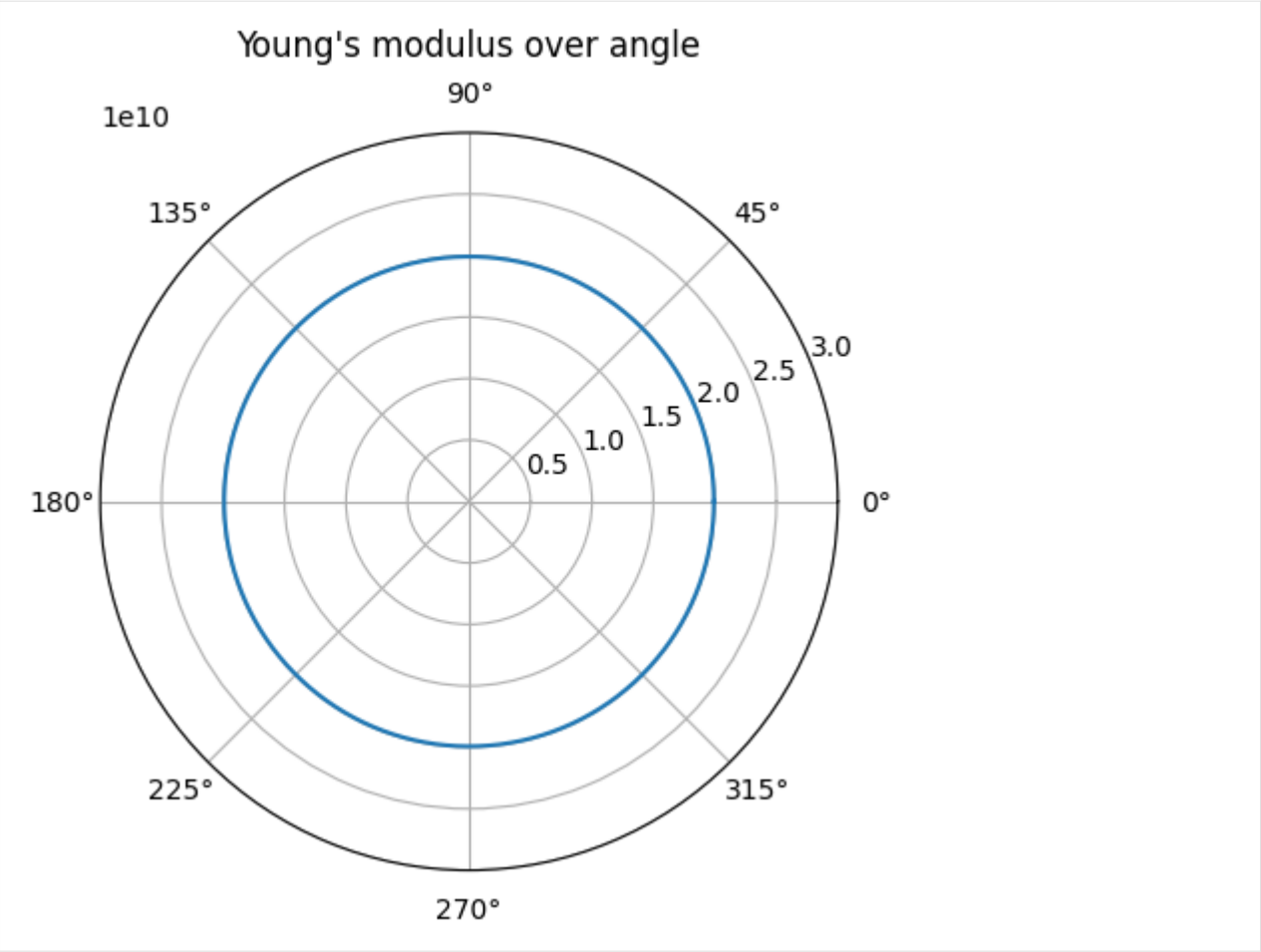


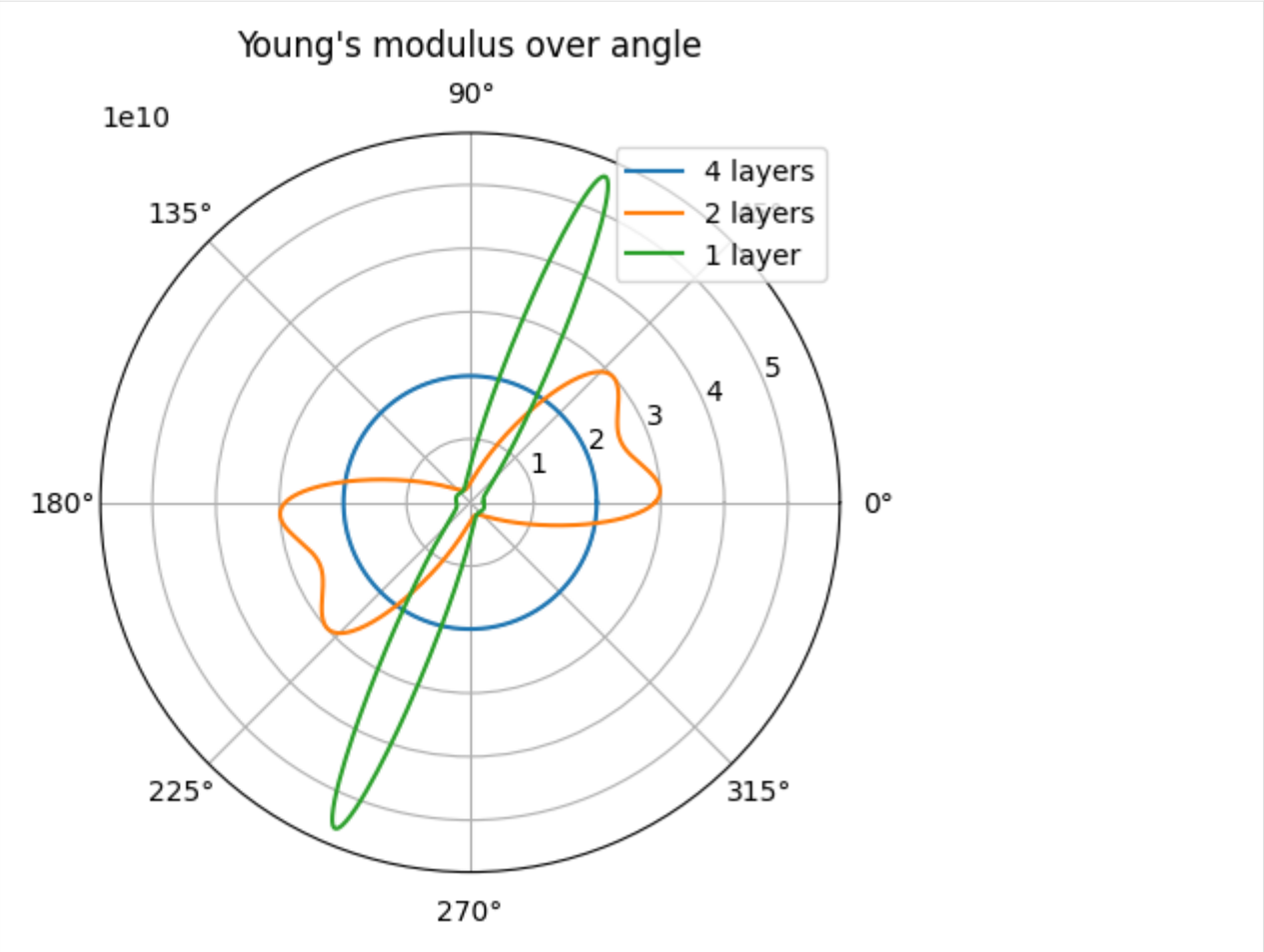
[6]:
```
# Halpin-Tsai

# define properties
E_carbon = 242e9
G_carbon = 105e9
nu_carbon = 0.1
l_carbon = 1.5e-3
r_carbon = 7.2 / 2 * 1e-6
vol_carbon = 0.25
E_pa6 = 1.18e9
G_pa6 = 0.4e9
nu_pa6 = 0.35

ht_carbon = HalpinTsai(
    E_carbon,
    E_pa6,
    G_carbon,
    G_pa6,
    nu_carbon,
    nu_pa6,
    l_carbon,
    r_carbon,
    vol_carbon,
)

# create array of laminas and according angles
```

(continues on next page)

(continued from previous page)

```
laminas = 4 * [
    ht_carbon.get_effective_stiffness()
]  # individual stiffnesses possible, e.g. [stiffness1, stiffness2, ...]
pi = np.pi
angles = [0, pi / 4, -pi / 4, pi / 2]

# define laminate
laminate1 = Laminate(laminas, angles)

# plot laminate directly via...
plotter.polar_plot_laminate(laminate1.get_effective_stiffness(), 1000, limit=3e10)

# repeat...
laminas = 2 * [ht_carbon.get_effective_stiffness()]
angles = [0, pi / 4]
laminate2 = Laminate(laminas, angles)

laminas = 1 * [ht_carbon.get_effective_stiffness()]
angles = [3 / 8 * pi]
laminate3 = Laminate(laminas, angles)

pHT1 = plotter.polar_plot_laminate(
    laminate1.get_effective_stiffness(), 1000, plot=False
)
pHT2 = plotter.polar_plot_laminate(
    laminate2.get_effective_stiffness(), 1000, plot=False
)
pHT3 = plotter.polar_plot_laminate(
    laminate3.get_effective_stiffness(), 1000, plot=False
)

# ...or plot a collection of laminates
plotter.polar_plot([pHT1 + ("4 layers",), pHT2 + ("2 layers",), pHT3 + ("1 layer",)])
```


Young's modulus over angle
90°
1e10
135°
45°
3.0
2.5
2.0
1.5
1.0
0.5
180°
0°
225°
315°
270°



3.1.2 Mori Tanaka: Why symmetrize?

The following notebook is intended to explain the various Mori-Tanaka implementations and solution strategies used in HomoPy. The goal is not to derive the equations, but to provide a learning by doing example. A prerequisite is a basic understanding of homogenization methods in the field of elasticity.

The starting point is the formulation after Benveniste (1987), which can also be found in Brylka (2017)

$$\bar{\mathbb{C}}^{\mathrm{MT}} = \mathbb{C}_m + c_f \left\langle (\mathbb{C}_{f,\alpha} - \mathbb{C}_m)\mathbb{A}^{\mathrm{SIP}}_{f,\alpha} \right\rangle_f \left(c_m \mathbb{I}^{\mathrm{S}} + c_f \left\langle \mathbb{A}^{\mathrm{SIP}}_{f,\alpha} \right\rangle_f\right)^{-1},$$

where $\bar{\mathbb{C}}^{\mathrm{MT}}$ is the effective stiffness, $\mathbb{C}_m$ is the matrix stiffness, $\mathbb{C}_{f,\alpha}$ is the stiffness of the inclusion of type α, c_m and c_f are the volume fractions of matrix and fiber, respectively, $\mathbb{A}^{\mathrm{SIP}}_{f,\alpha}$ is the strain localization tensor based on Eshelby's solution for the inclusion shape of type α and $\mathbb{I}^{\mathrm{S}}$ is the symmetric identity of order four. The brackets $\langle\cdot\rangle_f$ indicate the fiber volume average.

The problem with this formulation is that depending on the different shapes of inclusions and elastic symmetries (isotropy, transverse isotropy, ...) of the constituents, the effective stiffness is not guaranteed to be major symmetric, which violates thermodynamic principles.

To demonstrate this, we start with a simple example:

First, we assume a single inclusion type, namely carbon fibers aligned with the x-axis in a polyamide-6 matrix. For simplicity, the carbon fiber is assumed to be isotropic, while in reality it should be transverse isotropic.

[1]:
```
import numpy as np

np.set_printoptions(linewidth=100, precision=4)

from homopy.methods import *
from homopy.elasticity import *
from homopy.stiffness_plot import *

# define fiber and matrix properties
carbon_fiber = Isotropy(242e9, 0.1)
v_frac_carbon = 0.25
a_carbon = 347
polyamid6 = Isotropy(1.18e9, 0.35)

# define the MT homogenization and print it's effective stiffness
mt_carbon = mt_carbon = MoriTanaka(
    polyamid6, carbon_fiber, v_frac_carbon, a_carbon, shape="ellipsoid"
)
print(mt_carbon.effective_stiffness66)
```

```
[[6.0989e+10 1.1464e+09 1.1464e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [1.1464e+09 2.7483e+09 1.4048e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [1.1464e+09 1.4048e+09 2.7483e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 1.3435e+09 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.4507e+09 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.4507e+09]]
```

The result is a symmetric, i.e. thermodynamically consistent, transversely isotropic effective stiffness. The symmetry can also be validated by using

[2]:
```
mt_carbon.is_symmetric()
```

```
Left minor symmetry: passed
Right minor symmetry: passed
Major symmetry: passed
```

So far, there is nothing to complain about. This is due to the fact, that the Mori Tanaka formulation always results in a symmetric stiffness, when the matrix is isotropic, and the **single** inclusion is isotropic.

When we add glass fibers (or any other type of inclusion in that sense) to the mix, we suddenly get a different behaviour, as the next example will demonstrate.

[3]:
```
# define fiberproperties
glass_fiber = Isotropy(80e9, 0.22)
v_frac_glass = 0.25
a_glass = 225

# define the MT homogenization and print it's effective stiffness
mt_hybrid = MoriTanaka(
    polyamid6,
    [carbon_fiber, glass_fiber],
```

(continues on next page)

(continued from previous page)

```
    [v_frac_carbon / 2, v_frac_glass / 2],
    [a_carbon, a_glass],
    2 * ["ellipsoid"],
)
print(mt_hybrid.effective_stiffness66)
print("\n")
mt_hybrid.is_symmetric()
```

```
[[4.1198e+10 1.2154e+09 1.2154e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [1.2155e+09 2.7383e+09 1.3995e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [1.2155e+09 1.3995e+09 2.7383e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 1.3388e+09 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.4436e+09 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.4436e+09]]


Left minor symmetry: passed
Right minor symmetry: passed
Major symmetry: failed
The rel. residuum for major sym. is: res = 6.406621895094511e-06
```

We see that the effective stiffness does not contain the major symmetry and thus is not thermodynamically consistent. The relative residuum indicates that the major symmetry is only violated 'a bit', but it is violated. Therefore, a motivation is given to use an alternative algorithm which ensures that the effective stiffness will be symmetric.

By abbreviating the classic formulation above with

$$\bar{\mathbb{C}}^{\mathrm{MT}} = \mathbb{C}_m + c_f \underbrace{\left\langle (\mathbb{C}_{f,\alpha} - \mathbb{C}_m)\mathbb{A}_{f,\alpha}^{\mathrm{SIP}} \right\rangle_f}_{\mathbb{X}} \underbrace{\left(c_m \mathbb{I}^{\mathrm{S}} + c_f \left\langle \mathbb{A}_{f,\alpha}^{\mathrm{SIP}} \right\rangle_f \right)^{-1}}_{\mathbb{Y}^{-1}},$$

Segura et al. (2023) uses the relation

$$\mathbb{X}\mathbb{Y}^{-1} = \left(\mathbb{Y}\mathbb{X}^{-1}\right)^{-1}$$

to derive the alternative form

$$\bar{\mathbb{C}}^{\mathrm{MT}} = \mathbb{C}_m + c_f \left(c_m \left\langle (\mathbb{C}_{f,\alpha} - \mathbb{C}_m)\mathbb{A}_{f,\alpha}^{\mathrm{SIP}} \right\rangle_f^{-1} + c_f \underbrace{\left\langle \mathbb{A}_{f,\alpha}^{\mathrm{SIP}} \right\rangle_f \left\langle (\mathbb{C}_{f,\alpha} - \mathbb{C}_m)\mathbb{A}_{f,\alpha}^{\mathrm{SIP}} \right\rangle_f^{-1}}_{\mathbb{Z}} \right)^{-1},$$

where the left term in the paranthesis is always symmetric. Therefore, only the right term, i.e. $\mathbb{Z}$, needs to be symmetrized, to recover the major symmetry, i.e.

$$\bar{\mathbb{C}}^{\mathrm{MT}} = \mathbb{C}_m + c_f \left(c_m \left\langle (\mathbb{C}_{f,\alpha} - \mathbb{C}_m)\mathbb{A}_{f,\alpha}^{\mathrm{SIP}} \right\rangle_f^{-1} + \frac{c_f}{2}(\mathbb{Z} + \mathbb{Z}^{\mathsf{T}}) \right)^{-1}.$$

HomoPy uses this formulation, when the 'symmetrize' flag is set to True

[4]:
```
mt_hybrid_sym = MoriTanaka(
    polyamid6,
    [carbon_fiber, glass_fiber],
```

(continues on next page)

(continued from previous page)

```
    [v_frac_carbon / 2, v_frac_glass / 2],
    [a_carbon, a_glass],
    2 * ["ellipsoid"],
    symmetrize=True,
)
print(mt_hybrid_sym.effective_stiffness66)
print("\n")
mt_hybrid_sym.is_symmetric()
```

```
[[4.1198e+10 1.2155e+09 1.2155e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [1.2155e+09 2.7383e+09 1.3995e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [1.2155e+09 1.3995e+09 2.7383e+09 0.0000e+00 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 1.3388e+09 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.4436e+09 0.0000e+00]
 [0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.4436e+09]]


Left minor symmetry: passed
Right minor symmetry: passed
Major symmetry: passed
```

3.1.3 Interactive Notebook: Hybrid MT vs. HT

Unfortunately, this Notebook does not render interactively in the docs. If you would like to check out the interactive features, we recommend cloning the repository to your local machine, or using the Binder container.

[1]:
```
import numpy as np
from homopy.elasticity import *
from homopy.methods import *
from homopy.stiffness_plot import *
from homopy.tensor import *
```

[2]:
```
# define fiber and matrix properties for HT

# carbon (assuming isotropic fiber)
E_carbon = 242e9
G_carbon = 105e9
nu_carbon = 0.1
l_carbon = 1.5e-3
r_carbon = 7.2 / 2 * 1e-6
vol_carbon = 0.25

# glass (is isotropic)
E_glass = 80e9
nu_glass = 0.22
G_glass = 1 / 2 * E_glass / (1 + nu_glass)
l_glass = 0.8e-3
r_glass = 20 / 2 * 1e-6
```

(continues on next page)

(continued from previous page)

```
vol_glass = 0.25

# PA6
E_pa6 = 1.18e9
nu_pa6 = 0.35
G_pa6 = 1 / 2 * E_pa6 / (1 + nu_pa6)
```

[3]:
```
# define fiber and matrix properties for MT

# carbon
carbon_fiber = Isotropy(E_carbon, nu_carbon)
v_frac_carbon = vol_carbon
a_carbon = l_carbon / (2 * r_carbon)

# glass
glass_fiber = Isotropy(E_glass, nu_glass)
v_frac_glass = vol_glass
a_glass = l_glass / (2 * r_glass)

# PA6
polyamid6 = Isotropy(E_pa6, nu_pa6)
```

[4]:
```
# create HT and MT objects

ht_carbon = HalpinTsai(
    E_carbon,
    E_pa6,
    G_carbon,
    G_pa6,
    nu_carbon,
    nu_pa6,
    l_carbon,
    r_carbon,
    vol_carbon,
)

ht_glass = HalpinTsai(
    E_glass,
    E_pa6,
    G_glass,
    G_pa6,
    nu_glass,
    nu_pa6,
    l_glass,
    r_glass,
    vol_glass,
)
```

(continues on next page)

(continued from previous page)

```
mt_hybrid = MoriTanaka(
    polyamid6,
    [carbon_fiber, glass_fiber],
    [v_frac_carbon / 2, v_frac_glass / 2],
    [a_carbon, a_glass],
    2 * ["ellipsoid"],
    symmetrize=True,
)
```

[5]:
```
# give n fibers a random distribtution and go from here

pi = np.pi
angle_disc = 1 / 40 * pi  # angle discretization
distribution = "gauss"

# missuse ElasticPlot()
plotter = ElasticPlot()

n = 1000
theta = pi / 2


def random_fibers(n, distribution="random", sigma=0.1 * pi, mean_angle=0):
    fiber_dirs = []
    angles = []
    for i in range(n):
        if distribution == "random":
            phi = pi * np.random.rand() - pi / 2  # equally likely -> isotropic
        elif distribution == "gauss":
            phi = np.clip(
                np.random.normal(0, sigma), -pi / 2, pi / 2
            )  # gaussian distribution -> change values to shift

        phi += mean_angle
        while np.any(phi > pi / 2) or np.any(phi < -pi / 2):
            if phi > pi / 2:
                phi -= pi
            elif phi < -pi / 2:
                phi += pi

        fiber_dirs.append(plotter._dir_vec(phi, theta))
        angles.append(phi)
    return np.array(fiber_dirs), angles


fiber_dirs_carbon, angles_carbon = random_fibers(
    n, distribution=distribution, sigma=0.1 * pi
)
fiber_dirs_glass, angles_glass = random_fibers(
    n, distribution=distribution, sigma=0.2 * pi
)
```

(continues on next page)

(continued from previous page)

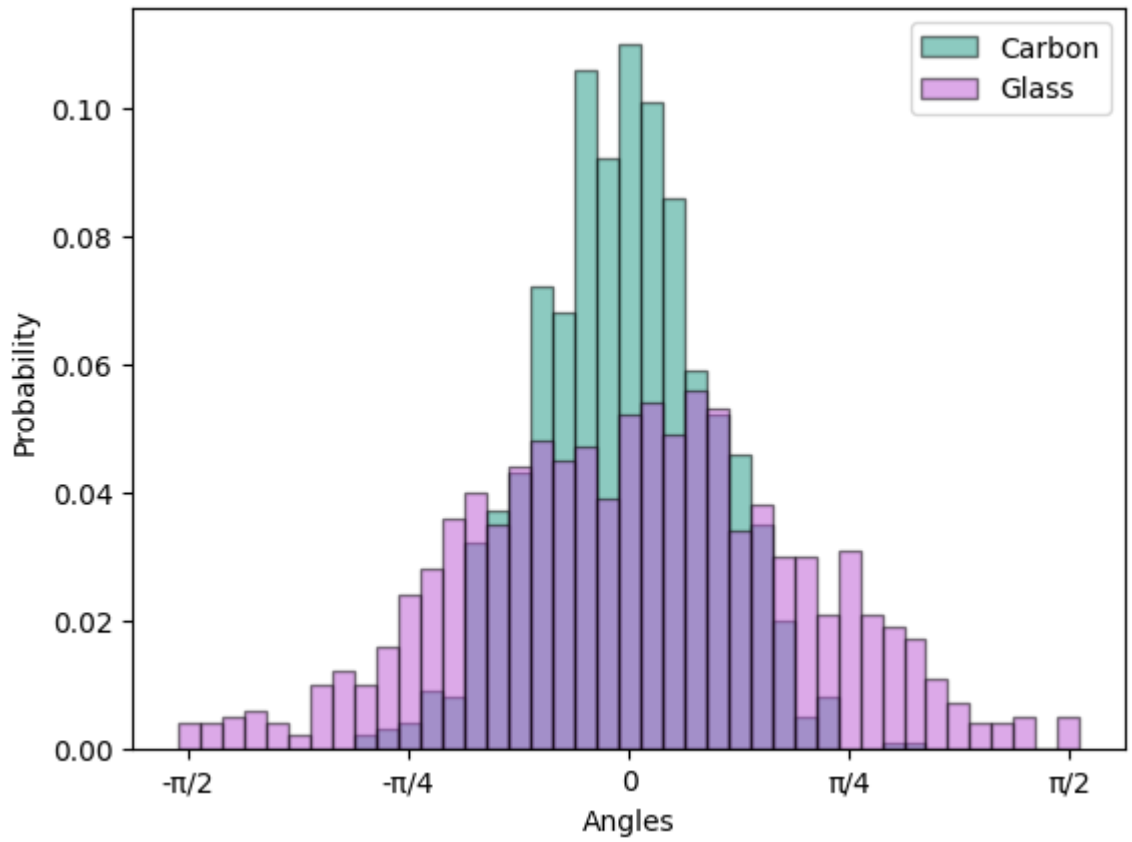
```
# define function to calculate a discrete planar fiber distribution with a given angle␣
↪spectrum


def get_distribution_histogram(angles, angle_disc):
    """
    Return the discrete orientation probability histogram for measured fiber_dirs.

    Paramters:
        - angles : array of shape (n,1)
            Measured fiber angles.
        - angle_spec : float in [0,pi]
            Angle spectrum.

    Returns:
        - angle_probs : ndarray of shape (pi/angle_spec, 1)
            Histogram of fiber orientation.
    """
    measurements = len(angles)
    angle_probs = []
    for i in np.arange(-pi / 2 - angle_disc / 2, pi / 2, angle_disc):
        # print("[{}, {}, {}]".format(i, i+angle_spec/2, i + angle_spec))
        hits = 0
        for angle in angles:
            if angle > i and angle <= i + angle_disc:
                hits += 1
        angle_probs.append([i + angle_disc / 2, hits / measurements])
    return angle_probs


hist_carbon = np.array(get_distribution_histogram(angles_carbon, angle_disc))
hist_glass = np.array(get_distribution_histogram(angles_glass, angle_disc))
```

[6]:
```
# plot angle distribution

import matplotlib.pyplot as plt

x_carbon = hist_carbon[:, 0]
y_carbon = hist_carbon[:, 1]

x_glass = hist_glass[:, 0]
y_glass = hist_glass[:, 1]

plt.bar(
    x_carbon,
    y_carbon,
    width=angle_disc,
    align="center",
    color=(27 / 256, 151 / 256, 131 / 256),
```

(continues on next page)

(continued from previous page)

```
    edgecolor="k",
    alpha=0.5,
    label="Carbon",
)
plt.bar(
    x_glass,
    y_glass,
    width=angle_disc,
    align="center",
    color="mediumorchid",
    edgecolor="k",
    alpha=0.5,
    label="Glass",
)
plt.xlabel("Angles")
plt.ylabel("Probability")
plt.xticks(
    np.arange(-pi / 2, pi / 2 + pi / 4, step=(pi / 4)),
    ["-/2", "-/4", "0", "/4", "/2"],
)


plt.legend()
plt.show()
```



[7]:
```
# build orientation tensor of 4th and 2nd order


def get_N(fiber_dirs):
    N4 = np.zeros((3, 3, 3, 3))
    for fiber_dir in fiber_dirs:
        N4 += np.einsum("i,j,k,l->ijkl", fiber_dir, fiber_dir, fiber_dir, fiber_dir)
    N4 /= len(fiber_dirs)
    return N4


N4_carbon = get_N(fiber_dirs_carbon)
N4_glass = get_N(fiber_dirs_glass)
```

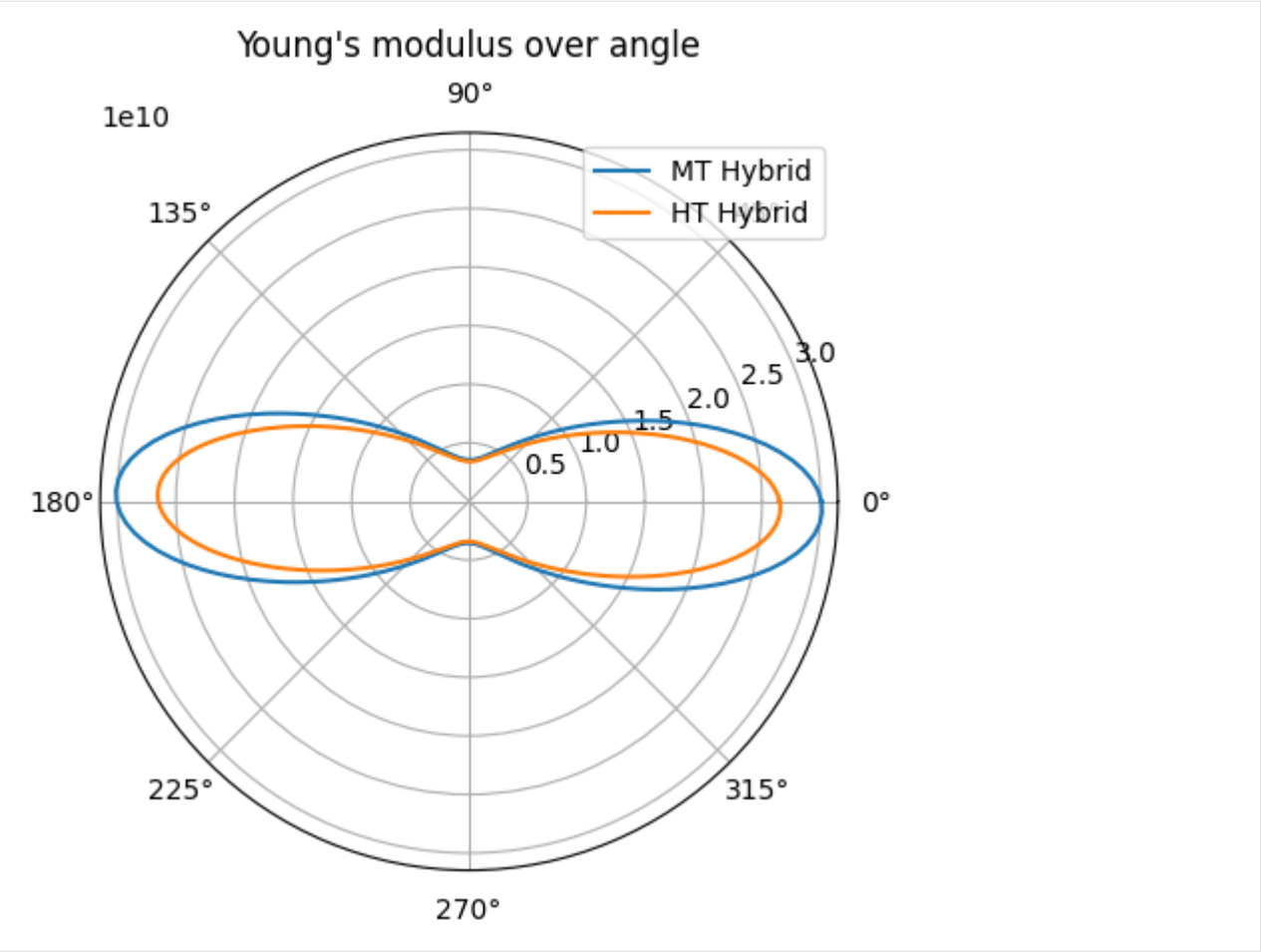
[8]:
```
# average MT

mt_hybrid.get_average_stiffness([N4_carbon, N4_glass])

# build laminate from HT results

laminas_carbon = len(angles_carbon) * [ht_carbon.get_effective_stiffness()]
laminas_glass = len(angles_glass) * [ht_glass.get_effective_stiffness()]
```

(continues on next page)

(continued from previous page)

```
laminas_combined = laminas_carbon + laminas_glass
angles_combined = angles_carbon + angles_glass

ht_laminate = Laminate(laminas_combined, angles_combined)
```

[9]:
```
# Compare MT and HT after orientation averaging

# mt_hybrid_plot = plotter.polar_plot_E_body(np.linalg.inv(mt_hybrid.effective_
↪stiffness66), 200, 0, plot=True)


mt_hybrid_plot_ave = plotter.polar_plot_E_body(
    mt_hybrid.effective_stiffness66, 200, 0, plot=False
)
ht_hybrid_plot_ave = plotter.polar_plot_laminate(
    ht_laminate.effective_stiffness33, 1000, plot=False
)


plotter.polar_plot(
    [mt_hybrid_plot_ave + ("MT Hybrid",), ht_hybrid_plot_ave + ("HT Hybrid",)]
)
```



[10]:
```
"""
Same plot as above, just with %matplotlib inline.
The reason for this is that %matplotlib notebook does not seem to be working with
binder currently. The above version is kept, because it looks a bit more crispy...

This is an interactive plot to show how the fiber distributions of each constituent
affects the effective porperties. Open in jupyter notebook to get full interaction
support. We recommend using the option Cell -> Current Outputs -> Toggle
"""

%matplotlib inline
%config InlineBackend.figure_format = 'svg' # makes everything super crispy
from ipywidgets import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


def update_fiber_dist(sig_c=0.1, dist_c = 'gauss', sig_g=0.2, dist_g = 'gauss',
                      vol_c=0.25, vol_g=0.25, mean_angle_c=0, mean_angle_g=0,
                      n=1000, angle_disc=1/40*np.pi):
```

(continues on next page)

(continued from previous page)

```
    ht_carbon = HalpinTsai(
        E_carbon,
        E_pa6,
        G_carbon,
        G_pa6,
        nu_carbon,
        nu_pa6,
        l_carbon,
        r_carbon,
        vol_c,
    )

    ht_glass = HalpinTsai(
        E_glass,
        E_pa6,
        G_glass,
        G_pa6,
        nu_glass,
        nu_pa6,
        l_glass,
        r_glass,
        vol_g,
    )


    fibers_carbon, angles_carbon = random_fibers(n, distribution=dist_c, sigma=sig_c*pi,␣
↪mean_angle=mean_angle_c)
    hist_carbon = np.array(get_distribution_histogram(angles_carbon, angle_disc=angle_
↪disc))
    x_carbon = hist_carbon[:,0]
    y_carbon = hist_carbon[:,1]

    fibers_glass, angles_glass = random_fibers(n, distribution=dist_g, sigma=sig_g*pi,␣
↪mean_angle=mean_angle_g)
    hist_glass = np.array(get_distribution_histogram(angles_glass, angle_disc=angle_
↪disc))
    x_glass = hist_glass[:,0]
    y_glass = hist_glass[:,1]

    N4_carbon = get_N(fibers_carbon)
    N4_glass = get_N(fibers_glass)

    # mt_hybrid_ave = mt_hybrid.get_average_stiffness([N2_carbon, N2_glass], [N4_carbon,␣
↪N4_glass], method='benveniste')

    mt_hybrid = MoriTanaka(polyamid6, [carbon_fiber, glass_fiber], [vol_c/2, vol_g/2],
                           [a_carbon, a_glass], 2*['ellipsoid'], N4=[N4_carbon, N4_
↪glass], symmetrize=True)

    laminas_carbon = len(angles_carbon)*[ht_carbon.get_effective_stiffness()]
    laminas_glass = len(angles_glass)*[ht_glass.get_effective_stiffness()]
```

(continues on next page)

(continued from previous page)

```
    laminas_combined = laminas_carbon + laminas_glass
    angles_combined = angles_carbon + angles_glass

    ht_laminate = Laminate(laminas_combined, angles_combined)

    mt_hybrid_plot_ave = plotter.polar_plot_E_body(mt_hybrid.effective_stiffness66, 200,␣
↪0, plot=False)
    ht_hybrid_plot_ave = plotter.polar_plot_laminate(ht_laminate.effective_stiffness33,␣
↪1000, plot=False)


    fig = plt.figure(figsize=(6,6))
    ax1 = plt.subplot(311)
    ax1.set_ylim([0, 0.2])


    line1 = ax1.bar(x_carbon,y_carbon,width=angle_disc,align='center', color=(27/256,151/
↪256,131/256),edgecolor='k', alpha=0.5, label='Carbon')
    line2 = ax1.bar(x_glass,y_glass,width=angle_disc,align='center', color='mediumorchid
↪',edgecolor='k', alpha=0.5, label='Glass')

    plt.legend()

    ax1.set_xlabel('Angles')
    ax1.set_ylabel('Probability')
    ax1.set_xticks(np.arange(-pi/2, pi/2+pi/4, step=(pi/4)), ['-/2','-/4', '0', '/4','/2
↪'])

    ax2 = plt.subplot(212, projection='polar')
    ax2.set_ylim([0,6e10])

    line3, = ax2.plot(mt_hybrid_plot_ave[0], mt_hybrid_plot_ave[1], label='MT Hybrid')
    line4, = ax2.plot(ht_hybrid_plot_ave[0], ht_hybrid_plot_ave[1], label='HT Hybrid')

    plt.legend()


sig_c = widgets.FloatSlider(value=0.1, min=0.0, max=0.3, step=0.01, description='SD carb.
↪:',
                            layout=Layout(width='250px'))
sig_g = widgets.FloatSlider(value=0.1,min=0.0, max=0.3, step=0.01, description='SD glass:
↪',
                            layout=Layout(width='250px'))
mean_angle_c = widgets.FloatSlider(value=0.0, min=-np.pi/2, max=np.pi/2, step=np.pi/16,␣
↪description='Mean angle carb.:',
                            layout=Layout(width='250px'), style = {'description_width':
↪'initial'})
mean_angle_g = widgets.FloatSlider(value=0.0, min=-np.pi/2, max=np.pi/2, step=np.pi/16,␣
↪description='Mean angle glass:',
                            layout=Layout(width='250px'), style = {'description_width':
↪'initial'})
dist_c = widgets.Dropdown(options=['gauss', 'random'],
```

(continues on next page)

(continued from previous page)

```
    value='gauss',
    disabled=False,
    description='Dist. carb.:',
    layout=Layout(width='180px')
)
dist_g = widgets.Dropdown(options=['gauss', 'random'],
    value='gauss',
    disabled=False,
    description='Dist. glass:',
    layout=Layout(width='180px')
)
vol_frac_c = widgets.BoundedFloatText(
    value=0.25,
    min=0,
    max=1,
    step=0.01,
    description='v_frac carb.:',
    disabled=False,
    layout=Layout(width='150px')
)
vol_frac_g = widgets.BoundedFloatText(
    value=0.25,
    min=0,
    max=1,
    step=0.01,
    description='v_frac glass:',
    disabled=False,
    layout=Layout(width='150px')
)

HBox_carbon = widgets.HBox([sig_c,dist_c,vol_frac_c,mean_angle_c])
HBox_glass = widgets.HBox([sig_g,dist_g,vol_frac_g,mean_angle_g])
ui = widgets.VBox([HBox_carbon, HBox_glass])

out = widgets.interactive_output(update_fiber_dist, {'sig_c': sig_c, 'dist_c': dist_c,
↪'sig_g': sig_g,
                                                     'dist_g': dist_g, 'vol_c': vol_frac_
↪c,
                                                     'vol_g': vol_frac_g, 'mean_angle_c':␣
↪mean_angle_c,
                                                     'mean_angle_g': mean_angle_g})

display(out, ui)
```

```
Output()
```

```
VBox(children=(HBox(children=(FloatSlider(value=0.1, description='SD carb.:',␣
↪layout=Layout(width='250px'), ma...
```

CHAPTER

FOUR

CODE

4.1 Modules

4.1.1 homopy.tensor module

Created on Wed Apr 27 21:09:24 2022

@author: nicolas.christ@kit.edu

Tensor class for basic arithmetic operations. More information on tensor representation in Voigt and Mandel notations are given in [Brannon2018].

class homopy.tensor.**Tensor**

Bases: `object`

Tensor class to help with basic arithmetic operations on tensor space.

Variables

- **~Tensor.e1** (*ndarray of shape (3,)*) – Vector 1 of orthonormalbasis of 1st order tensors.
- **~Tensor.e2** (*ndarray of shape (3,)*) – Vector 2 of orthonormalbasis of 1st order tensors.
- **~Tensor.e3** (*ndarray of shape (3,)*) – Vector 3 of orthonormalbasis of 1st order tensors.
- **~Tensor.B** (*ndarray of shape (3, 3, 6)*) – Orthonormalbasis of 4th order tensors in normalized Voigt notation.

_diade(*di*, *dj*)

Return diadic product of two directional vectors. This is used to calculate the basis tensors in the normalized Voigt notation.

Parameters

- **di** (*ndarray of shape (3,)*) – Directional vector #1.
- **dj** (*ndarray of shape (3,)*) – Directional vector #2.

Returns

ndarray of shape (3, 3) – Tensor of 2nd order in tensor notation.

_diade4(*bi*, *bj*)

Return diadic product of two tensors. This is used to transfer stiffness tensors from normalized Voigt notation to regular tensor notation.

Parameters

- **bi** (*ndarray of shape (3, 3)*) – Orthonormal basis tensor #1.
- **bj** (*ndarray of shape (3, 3)*) – Orthonormal basis tensor #2.

Returns
ndarray of shape (3, 3, 3, 3) – Tensor of 4th order in tensor notation.

tensor_product(*tensor_a*, *tensor_b*)

Return the mapping of one tensor of 4th order to another in the normalized Voigt notation.

Parameters

- **tensor_a** (*ndarray of shape (6, 6)*) – Tensor #1.
- **tensor_b** (*ndarray of shape (6, 6)*) – Tensor #2.

Returns
ndarray of shape (6, 6) – Resulting mapping.

matrix2voigt(*matrix*)

Return the Voigt notation of a tensor of 2nd order calculated from the regular tensor notation.

Parameters
matrix (*ndarray of shape (3, 3)*) – Tensor of 2nd order in regular tensor notation.

Returns
ndarray of shape (6,) – Tensor in Voigt notation.

matrix2mandel(*matrix*)

Return the normalized Voigt notation of a tensor of 2nd order calculated from the regular tensor notation.

Parameters
matrix (*ndarray of shape (3, 3)*) – Tensor of 2nd order in regular tensor notation.

Returns
ndarray of shape (6,) – Tensor in normalized Voigt notation.

tensor2mandel(*tensor*)

Return the normalized Voigt (Mandel) notation of a tensor of 4th order calculated fromthe regular tensor notation.

Parameters
tensor (*ndarray of shape (3, 3, 3, 3)*) – Tensor of 4th order in regular tensor notation.

Returns
ndarray of shape (6, 6) – Tensor in normalized Voigt notation.

mandel2tensor(*mandel*)

Return the regular tensor notation of a tensor calculated from the normalized Voigt (Mandel) notation.

Parameters
mandel (*ndarray of shape (6, 6)*) – Tensor of 4th order in normalized Voigt notation.

Returns
tensor (*ndarray of shape (3, 3, 3, 3)*) – Tensor in regular tensor notation.

4.1.2 homopy.elasticity module

Created on Wed Apr 27 21:09:24 2022

@author: nicolas.christ@kit.edu

Module that contains the linear elastic stiffness classes of Isotropy and Transverse Isotropy.

class homopy.elasticity.**Elasticity**

Bases: *Tensor*

Elasticity class to express generic elasitc stiffness tensors. The class inherits from the Tensor class.

Variables

- **~Elasticity.stiffness3333** (*ndarray of shape (3, 3, 3, 3)*) – Stiffness values in the regular tensor notation in Pa.
- **~Elasticity.stiffness66** (*ndarray of shape (6, 6)*) – Stiffness values in the normalized Voigt notation in Pa.

class homopy.elasticity.**TransverseIsotropy**(*E1*, *E2*, *G12*, *G23*, *nu12*)

Bases: *Elasticity*

Transverse Isotropy class to express transverse-isotropic elasitc stiffness tensors. The class inherits from the Elasticity class.

Parameters

- **E1** (*float*) – Young's modulus in longitudinal direction in Pa.
- **E2** (*float*) – Young's modulus in transverse direction in Pa.
- **G12** (*float*) – Shear modulus in the longitudinal-transverse plane in Pa.
- **G23** (*float*) – Shear modulus in the transverse-transverse plane in Pa.
- **nu12** (*float*) – Poisson's ratio in longitudinal direction (dimensionless).

Variables

- **~TransverseIsotropy.E1** (*float*) – Young's modulus in longitudinal direction in Pa.
- **~TransverseIsotropy.E2** (*float*) – Young's modulus in transverse direction in Pa.
- **~TransverseIsotropy.G12** (*float*) – Shear modulus in the longitudinal-transverse plane in Pa.
- **~TransverseIsotropy.G23** (*float*) – Shear modulus in the transverse-transverse plane in Pa.
- **~TransverseIsotropy.nu12** (*float*) – Poisson's ratio in longitudinal direction (dimensionless).
- **~TransverseIsotropy.nu23** (*float*) – Poisson's ratio in transverse direction (dimensionless).

_get_stiffness()

Calculate the stiffness parameters for both notations.

class homopy.elasticity.**Isotropy**(*E*, *nu*)

Bases: *TransverseIsotropy*

Isotropy class to express isotropic elasitc stiffness tensors. The class inherits from the Transverse Isotropy class.

Parameters

- **E** (*float*) – Young's modulus in Pa.
- **nu** (*float*) – Poisson's ratio (dimensionless).

Variables

- **~Isotropy.E** (*float*) – Young's modulus in Pa.
- **~Isotropy.nu** (*float*) – Poisson's ratio (dimensionless).
- **~Isotropy.lam** (*float*) – First Lamé constant in Pa.
- **~Isotropy.mu** (*float*) – Second Lamé constant in Pa.

`_get_lambda()`

Return the first Lamé constant from other material parameters.

Returns

float – First Lamé constant in Pa.

`_get_mu()`

Return the second Lamé constant from other material parameters.

Returns

float – Second Lamé constant in Pa.

4.1.3 homopy.methods module

Created on Wed Apr 27 21:09:24 2022

@author: nicolas.christ@kit.edu

Mori-Tanaka Homogenization after [Benveniste1987]. Multi-inclusion implementation after [Brylka2017]. Eshelby's tensor is taken from [Tandon1984] but can also be found in [Gross2016]. Thoroughly literature on Eshelby's tensor can also be found in [Mura1987] (pp. 74 ff.). Halpin-Tsai homogenization after [Fu2019] (pp. 143 ff.). Also, the effective planar stiffness matrix for the Halpin-Tsai homogenization is based on the laminate analogy approach after [Fu2019] (pp. 155 ff.).

class homopy.methods.**MoriTanaka**(*matrix*, *fiber*, *v_frac*, *a_ratio*, *shape='ellipsoid'*, *N4=None*, *symmetrize=False*)

Bases: *Tensor*

Mori Tanaka class to calculate the homogenized stiffness for fiber reinforced polymers with possibly different types of inclusions. The class inherits from the Tensor class.

Parameters

- **matrix** (Elasticity) – Polymer matrix material in normalized Voigt notation.
- **fiber** (Elasticity *or list of* Elasticity) – Fiber material in normalized Voigt notation.
- **v_frac** (*float*) – Volume fraction of the fiber material within the matrix material.
- **a_ratio** (*float or list of floats*) – Aspect ratio of the fiber material.
- **shape** (*string or list of strings, default='ellipsoid'*) – Flag to determine which assumptions are taken into consideration for the geometry of the fiber (options: 'ellipsoid', 'sphere', 'needle')
- **N4** (*ndarray or list of ndarrays of shape (3, 3, 3, 3), default=None*) – Orientation tensor(s) of 4th order.

- **symmetrize** (*boolean, default='False'*) – Flag to determine whether the effective and orientation averaged stiffnesses shall be symmetrized. For this the method in [Segura2023] is used.

Variables

- **~MoriTanaka.matrix** (Elasticity) – Polymer matrix material.
- **~MoriTanaka.fiber** (Elasticity *or list of* Elasticity) – Fiber (or other inclusion) materials.
- **~MoriTanaka.Cm** (*ndarray of shape (6, 6)*) – Stiffness of matrix material in normalized Voigt notation in Pa.
- **~MoriTanaka.eye** (*ndarray of shape (6, 6)*) – Identity tensor in normalized Voigt notation.
- **~MoriTanaka.N2** (*ndarray or list of ndarrays of shape (3, 3)*) – Orientation tensor(s) of 2nd order.
- **~MoriTanaka.N4** (*ndarray or list of ndarrays of shape (3, 3, 3, 3)*) – Orientation tensor(s) of 4th order.
- **~MoriTanaka.effective_stiffness3333** (*ndarray of shape (3, 3, 3, 3)*) – Holds the stiffness values in the regular tensor notation in Pa. When orientations are given, these are included directly.
- **~MoriTanaka.effective_stiffness66** (*ndarray of shape (6, 6)*) – Holds the stiffness values in the normalized Voigt notation in Pa. When orientations are given, these are included directly.

_get_eshelby(*a_ratio*, *return_dim='66'*, *shape='ellipsoid'*)

Return the Eshelby tensor according to the fiber type.

Parameters

- **a_ratio** (*float*) – Aspect ratio of fiber (dimensionless).
- **return_dim** (*string, default='66'*) – Flag to determine whether the tensor should be returned in normalized Voigt or regular tensor notation (options: '66', '3333')
- **shape** (*string, default='ellipsoid'*) – Flag to determine which assumptions are taken into consideration for the geometry of the fiber (options: 'ellipsoid', 'sphere', 'needle')

Returns

S (*ndarray of shape (6, 6) or (3, 3, 3, 3)*) – Eshelby inclusion tensor.

get_effective_stiffness()

Return the effective stiffness of the composite material, based on Eq. 14a in [Benveniste1987].

Returns

C_eff (*ndarray of shape (6, 6)*) – Homogenized stiffness tensor in the normalized Voigt notation in Pa.

get_average_stiffness(*N4*, *return_dim='66'*)

Return the averaged effective stiffness based on orientation tensors. Overwrites the object variables self.effective_stiffness66 and self.effective_stiffness3333.

Parameters

- **N4** (*ndarray or list of ndarrays of shape (3, 3, 3, 3)*) – Orientation tensor of 4th order.

- **return_dim** (*string, default='66'*) – Flag to determine whether the tensor should be returned in normalized Voigt or regular tensor notation (options: '66', '3333')

Returns

ndarray of shape (6, 6) or (3, 3, 3, 3) – Averaged stiffness tensor in normalized Voigt or regular tensor notation in Pa.

static get_orientation_average(*tensor*, *N2*, *N4*)

Return the orientation average of a tensor after [Advani1987], Eq. 29.

Parameters

- **tensor** (*ndarray of shape (3, 3, 3, 3)*) – Tensor to be averaged (must be transversely isotropic).
- **N2** (*ndarray of shape (3, 3)*) – Orientation tensor of 2nd order.
- **N4** (*ndarray of shape (3, 3, 3, 3)*) – Orientation tensor of 4th order.

Returns

ave_tensor (*ndarray of shape (3, 3, 3, 3)*) – Orientation average of given tensor.

is_symmetric()

Print the symmetry status of the effective stiffness.

class homopy.methods.**HalpinTsai**(*E_f*, *E_m*, *G_f*, *G_m*, *nu_f*, *nu_m*, *l_f*, *r_f*, *vol_f*, *package='hex'*)

Bases: object

Halpin Tsai class to calculate the homogenized stiffness for fiber reinforced polymers as laminas. This is then used as input for the Laminate class.

Parameters

- **E_f** (*float*) – Young's modulus of fiber in Pa.
- **E_m** (*float*) – Young's modulus of matrix in Pa.
- **G_f** (*float*) – Shear modulus of fiber in Pa.
- **G_m** (*float*) – Shear modulus of matrix in Pa.
- **nu_f** (*float*) – Poisson ratio of fiber (dimensionless).
- **nu_m** (*float*) – Poisson ratio of matrix (dimensionless).
- **l_f** (*float*) – Average length of fiber in m.
- **r_f** (*float*) – Average radius of fiber in m.
- **vol_f** (*float*) – Poisson ratio of matrix (dimensionless).
- **package** (*string, default: hex*) – Package structure of fibers in composite (options: 'hex', 'square').

Variables

- **~HalpinTsai.E_f** (*float*) – Young's modulus of fiber in Pa.
- **~HalpinTsai.E_m** (*float*) – Young's modulus of matrix in Pa.
- **~HalpinTsai.G_f** (*float*) – Shear modulus of fiber in Pa.
- **~HalpinTsai.G_m** (*float*) – Shear modulus of matrix in Pa.
- **~HalpinTsai.nu_f** (*float*) – Poisson ratio of fiber (dimensionless).
- **~HalpinTsai.nu_m** (*float*) – Poisson ratio of matrix (dimensionless).

- **~HalpinTsai.l_f** (*float*) – Average length of fiber in m.
- **~HalpinTsai.r_f** (*float*) – Average radius of fiber in m.
- **~HalpinTsai.vol_f** (*float*) – Poisson ratio of matrix (dimensionless).
- **~HalpinTsai.package** (*string, default: hex*) – Package structure of fibers in composite (options: 'hex', 'square').
- **~HalpinTsai.effective_stiffness33** (*ndarray of shape (3, 3)*) – Holds the stiffness values in the reduced, normalized Voigt notation in Pa.

_get_effective_parameters()

Calculates the effective parameters of a single lamina for given constituent parameters.

Raises

ValueError – Package can only be "hex" or "square".

get_effective_stiffness()

Return the planar stiffness based on the effective parameters of a single lamina.

Returns

C (*ndarray of shape (3, 3)*) – Planar stiffness of lamina.

class homopy.methods.**Laminate**(*lamina_stiffnesses*, *angles*, *vol_fracs=None*)

Bases: object

Class to average over n laminas from Halpin-Tsai homogenization.

Parameters

- **lamina_stiffnesses** (*array of shape (n,)*) – Individual stiffness of n laminas in Pa.
- **angles** (*array of shape (n,)*) – Individual angle of ith lamina in radians.
- **vol_fracs** (*array of shape (n,)*) – Volume fraction of ith lamina (must sum to 1). If None is given, each lamina is averaged equally.

Variables

~Laminate.effective_stiffness33 (*ndarray of shape (3, 3)*) – Holds the stiffness values in the reduced, normalized Voigt notation in Pa.

get_effective_stiffness()

Return effective stiffness of laminate.

Returns

C_eff (*ndarray of shape (3, 3)*) – Effective stiffness of laminate in Pa.

static rotate_stiffness(*lamina_stiffness*, *angle*)

Return planarly rotated stiffness matrix. The planar rotation matrix around the z-axis has the form

$$\underline{R} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

from which the transformation matrix was extracted in accordance to [Slawinksi2010], Eq. 5.2.19.

Parameters

- **lamina_stiffness** (*ndarray of shape (3, 3)*) – Stiffness matrix of lamina in Pa.
- **angle** (*float*) – Planar angle to rotate the stiffness matrix about in radiants.

Returns

rot_stiffness (*ndarray of shape (3, 3)*) – Rotated stiffness matrix in Pa.

4.1.4 homopy.stiffness_plot module

Created on Thu Apr 28 16:24:54 2022

@author: nicolas.christ@kit.edu

3D and Polar plot of Young's modulus body based on [Boehlke2001].

class homopy.stiffness_plot.**ElasticPlot**(*USEVOIGT=False*)

Bases: *Tensor*

Plot class to visualize tensorial results.

Parameters
USEVOIGT (*boolean*) – Flag which determines the usage of Voigt (USEVOIGT=True), or Normalized Voigt / Mandel (USEVOIGT=False).

Variables
~ElasticPlot.USEVOIGT (*boolean*) – Flag which determines the usage of Voigt (USEVOIGT=True), or Normalized Voigt / Mandel (USEVOIGT=False).

matrix_reduction(*matrix*)

Return the reduced notation (Voigt or normalized Voigt) depending on which one is ought to be used.

Parameters
matrix (*ndarray of shape (3, 3)*) – Tensor in regular tensor notation.

Returns
ndarray of shape (6,) – Tensor in Voigt or normalized Voigt notation.

static _get_reciprocal_E(*didi*, *S*)

Return the reciprocal of Young's modulus (compliance) in the direction of di.

Parameters

- **didi** (*ndarray of shape (6,)*) – Directional tensor.
- **S** (*ndarray of shape (6, 6)*) – Compliance tensor in Voigt or normalized Voigt notation.

Returns
float – Scalar compliance value in direction of di.

_get_E(*di*, *S*)

Return Young's modulus in the direction of di.

Parameters

- **di** (*ndarray of shape (3,)*) – Directional vector.
- **S** (*ndarray of shape (6, 6)*) – Compliance tensor in Voigt or normalized Voigt notation.

Returns
float – Scalar stiffness value in direction of didi.

static _dir_vec(*phi*, *theta*)

Return directional vector based on angular parametrization.

Parameters

- **phi** (*float*) – First angle in [0, 2*pi].
- **theta** (*float*) – Second angle in [0, pi].

Returns
ndarray of shape (3,) – Directional vector.

`plot_E_body`(*C*, *o*, *p*, *bound=None*, *rcount=200*, *ccount=200*)

Plot stiffness body.

Parameters

- **C** (*ndarray of shape (6, 6)*) – Stiffness tensor in Voigt or normalized Voigt notation.
- **o** (*int*) – Number of discretization steps for first angle.
- **p** (*int*) – Number of discretization steps for second angle.
- **bound** (*array of shape (3,), default=None*) – Boundaries for the 3 axis for the visualization. If None, boundaries are set automatically.
- **rcount** (*int*) – Maximum number of samples used in first angle direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Defaults to 200.
- **ccount** (*int*) – Maximum number of samples used in second angle direction. If the input data is larger, it will be downsampled (by slicing) to these numbers of points. Defaults to 200.

`polar_plot_E_body`(*C*, *o*, *angle*, *plot=True*)

Plot slice of stiffness body.

Parameters

- **C** (*ndarray of shape (6, 6)*) – Stiffness tensor in Voigt or normalized Voigt notation.
- **o** (*int*) – Number of discretization steps for first angle.
- **angle** (*float*) – Angle to determine the angular orientation of the slice. Does not work yet.
- **plot** (*boolean*) – Determines whether the plot will be displayed. If False, only the meta-data of the plot will be returned.

Returns

- **rad** (*ndarray of shape (n+1,)*) – Angular positions for polar plot.
- **E** (*ndarray of shape (n+1,)*) – Sitffness at corresponding angle.

`static polar_plot`(*data*)

Polar plot of multiple Stiffness bodies in one plot. For this use the data generated from polar_plot_E_body or polar_plot_laminate with plot=False.

Parameters
data (*list*) – Data to be plotted with angluar position, stiffness and an optional string for the label in the plot.

`polar_plot_laminate`(*laminate_stiffness*, *o*, *limit=None*, *plot=True*)

Polar plot stiffness body of laminate. This method should be used for laminate results for the Halpin-Tsai homogenization.

Parameters

- **laminate_stiffness** (*ndarray of shape (3, 3)*) – Planar stiffness matrix in Voigt or normalized Voigt notation.
- **o** (*int*) – Number of discretization steps for first angle.

- **limit** (*float*) – Limit of radial axis in polar plot.
- **plot** (*boolean*) – Determines whether the plot will be displayed. If False, only the metadata of the plot will be returned.

Returns

- **rad** (*ndarray of shape (n+1,)*) – Angular positions for polar plot.
- **E** (*ndarray of shape (n+1,)*) – Sitffness at corresponding angle.

get_E_laminate(*C*, *phi*)

Return Young's modulus of laminate as a function of angle omega.

Parameters

- **C** (*ndarray of shape (3, 3)*) – Stiffness of laminate in default (orthonormal) coordinate system.
- **phi** (*float*) – Angle of orientation in radians.

Returns

E (*float*) – Young's modulus in angle direction

CHAPTER

FIVE

INSTALLING

Install HomoPy by running:

```
pip install homopy
```

CHAPTER

SIX

ACKNOWLEDGMENT

6.1 Acknowledgment

The research documented in this repository has been funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) - project number 255730231. The support by the German Research Foundation within the International Research Training Group “Integrated engineering of continuous-discontinuous long fiber reinforced polymer structures“ (GRK 2078) is gratefully acknowledged.

CHAPTER

SEVEN

REFERENCES

7.1 References

BIBLIOGRAPHY

[Advani1987] Advani, S.G. and Tucker III, C.L. (1987), 'The Use of Tensors to Describe and Predict Fiber Orientation in Short Fiber Composites', *Journal of Rheology*, pp. 751-784, Available at: https://doi.org/10.1122/1.549945

[Benveniste1987] Benveniste, Y. (1987), 'A new approach to the application of Mori-Tanaka's theory in composite materials', *Mechanics of Materials*, pp. 147-157, Available at: https://doi.org/10.1016/0167-6636(87)90005-6

[Boehlke2001] Böhlke, T. and Brüggemann, C. (2001), 'Graphical representation of the generalized Hooke's law', *Technische Mechanik*, 21 (2), pp. 145-158

[Brannon2018] Brannon, R. M. (2018), 'Voigt and Mandel components', in *Rotation, Reflection, and Frame Changes*, IOP Publishing

[Brylka2017] Brylka, B. (2017), *Charakterisierung und Modellierung der Steifigkeit von langfaserverstärktem Polypropylen*, KIT Scientific Publishing

[Chawla2019] Chawla, K. K. (2019), *Composite Materials*, Springer International Publishing

[Christensen2012] Christensen, R. M. (2012), *Mechanics of Composite Materials*, Dover Publications

[Cox1952] Cox, H. L. (1952), 'The elasticity and strength of paper and other fibrous materials', *British Journal of Applied Physics*, 3 (3), pp. 72–79, Available at: https://doi.org/10.1088/0508-3443/3/3/302

[Eshelby1957] Eshelby, J.D. (1957), 'The determination of the elastic field of an ellipsoidal inclusion, and related problems', *Proceedings of the Royal Society of London*, A 241 (1957), pp. 376–396, Available at: https://doi.org/10.1098/rspa.1957.0133

[Fu1998] Fu, S.Y., Lauke, B. (1998), 'The elastic modulus of misaligned short-fiber-reinforced polymers', *Composites Science and Technology*, 58 (3), pp. 389-400, Available at: https://doi.org/10.1016/S0266-3538(97)00129-2

[Fu2002] Fu, S.Y., Xu, G., Mai, Y.W. (2002), 'On the elastic modulus of hybrid particle/short-fiber/polymer composites', *Composites Part B: Engineering*, 33 (4), pp. 291–299, Available at https://doi.org/10.1016/S1359-8368(02)00013-6

[Fu2019] Fu, S.Y., Lauke, B., Mai, Y.W. (2019), *Science and engineering of short fibre-reinforced polymer composites*, Woodhead Publishing

[Gross2016] Gross, D. and Seelig, T. (2016), *Bruchmechanik*, Springer Berlin Heidelberg

[Halpin1969] Halpin, J.C. (1969), 'Effects of environmental factors on composite materials', *Air Force Materials Laboratory, Air Force Systems Command*

[Mori1973] Mori, T. and Tanaka, K. (1973), 'Average stress in matrix and average elastic energy of materials with misfitting inclusions', *Acta Metallurgica*, 21 (5), pp. 571-574, Available at: https://doi.org/10.1016/0001-6160(73)90064-3

[Mura1987] Mura, T. (1987), *Micromechanics of defects in solids*, Springer Dordrecht

[Qiu1990] Qiu, Y.P., Weng, G.J. (1990), 'On the application of mori-tanaka's theory involving transversely isotropic spheroidal inclusions', *International Journal of Engineering Science*, 28 (11) 1121-1137, Available at: https://doi.org/10.1016/0020-7225(90)90112-V

[Segura2023] Segura, N.J., Pichler, B.L.A. and Hellmich, C. (2023), 'Concentration tensors preserving elastic symmetry of multiphase composites', *Mechanics of Materials*, Available at: https://doi.org/10.1016/j.mechmat.2023.104555

[Slawinksi2010] Slawinski, M.A. (2010), *Waves and rays in elastic continua*, World Scientific

[Tandon1984] Tandon, G.P. and Weng, G.J. (1984), 'The effect of aspect ratio of inclusions on the elastic properties of unidirectionally aligned composites', *Polymer Composites*, pp. 327-333, Available at: https://doi.org/10.1002/pc.750050413

[Weng1990] Weng, G.J. (1990), 'The theoretical connection between mori-tanaka's theory and the hashin-shtrikman-walpole bounds', *International Journal of Engineering Science*, 28 (11) 1111–1120, Available at: https://doi.org/10.1016/0020-7225(90)90111-U

PYTHON MODULE INDEX

h

INDEX

Symbols

E

G

H

I

L

M

P

R

T