pyElli Documentation

Release stable

dobener

Oct 15, 2025

CONTENTS

PyElli is an open-source numerical solver for spectral ellipsometry employing well-known 2x2 and 4x4 solver algorithms. It is intended for a broad range of problems such as simple fitting of layered structures, anisotropic layers and any other polarized light interaction with layered 1D structures. It serves as a system for the day to day ellipsometry task at hand and aims to make optical model generation standardized and reproducible.

PyElli is build to be easily extendable by optical models. However, pyElli comes with batteries included and already offers a wide range of *dispersion models* and the material database of Refractiveindex.info.

Most of the models presented in the comprehensive book of Fujiwara and Collins[1] are present and additionally a lot of other models used by ellipsometry vendor software are included.

The material database offers the dispersions seen on the website and can be accessed by using the `elli.db.RII` module.

To start you may want to dive into *install*. The bast way to start is to have a look at the *basic usage* or the *other examples*.

PyElli consists of a set of classes which work together to create a full light interaction experiment. In the image below you see the set of different classes and how they work together to evaluate a modeled system.

It starts by building a set of dispersions and plugging them into materials classes the specific number of dispersions depends on whether it is an `IsotropicMaterial` or an `AnisotropicMaterial`. These materials classes also support creating effective medium layers for inclusions or roughnesses. The next step is building a `Structure` from these materials. The `Structure` needs as least two materials for the incoming and outgoing materials, but can contain arbitrary more `layers` which are only limited by the computational resources. The `VaryingMixtureLayer` class can also account for gradient changes of materials in z-direction of a layer, which is useful for gradient layers or roughness modeling. As the last step the `Structure` is plugged into an `Experiment`, which contains the experimental conditions, such as light polarization. By evaluating the experiment a `Result` class containing the calculated data is returned. The creation of an experiment can be skipped by calling the `evaluate` method directly on a `Structure` class if you want to use standard experimental settings.

References

[1] H. Fujiwara and R. W. Collins, Spectroscopic Ellipsometry for Photovoltaics, Volume 1: Fundamental Principles and Solar Cell Characterization, Ed. 1, Springer Series in Optical Sciences 212 (2018). https://doi.org/10.1007/978-3-319-75377-5

CHAPTER

ONE

CONTENTS

1.1 Install

The installers for all releases are available at the Python package Index (PyPI).

To install the package in your current virtual environment execute

```
pip install "pyElli[fitting]"
```

This installs pyElli with the additional fitting capabilities and interactive widgets. If you don't want to have this functionality just drop the *[fitting]* in the end.

To increase performance of the 4x4 Solver, it is recommended to install PyTorch manually, as it is too big to include in the standard installation. Installation information can be found at the PyTorch Website. The CPU variant is sufficient, if you want to save some space.

A complete environment for pyElli is also available as a Docker Container. To pull and run it directly just execute

```
docker run -p 8888:8888 domna/pyelli
```

from your local docker install. After startup a link should appear in your console. Click it and you will be directed to a jupyter server with the latest release of pyElli available.

To install the latest development version use:

```
pip install git+https://github.com/pyEllips/pyElli.git
```

The source code is hosted on GitHub, to manually install from source, clone the repository and run *pip install -e .* in the folder to install it in development mode:

```
git clone --recurse-submodules https://github.com/PyEllips/pyElli
cd pyElli
pip install -e .
```

1.2 Examples

1.2.1 Basic usage of pyElli

Basic usage of building a model and fitting it to measurement data of SiO2 on Si.

```
import elli
from elli.fitting import ParamsHist, fit
```

Reading data

We load the data from the generated NeXus file and select the angle we want to analyse. You may set the ANGLE constant to 50 or 60 to select other angles of incidence from the example file. Additionally, we're cutting the wavelength axis to be in between 210 nm and 800 nm. This is because we're using literature values for Si, which are only defined in this wavelength range.

```
ANGLE = 70
psi_delta = elli.read_nexus_psi_delta("SiO2onSi.ellips.nxs").loc[ANGLE].loc[210:800]
```

Setting parameters

As an example we analyse an oxidation layer of SiO2 on Si. Prior to defining our model, we have to set the parameters we want to use. We're going to use a *Cauchy model* for SiO2 and load the Si values from literature values. The parameter names can be chosen freely, but you have to use the exact same name in the later model definition. The package uses lmfit as fitting tool and you may refer to their documentation for details on parameter definition.

```
params = ParamsHist()
params.add("SiO2_n0", value=1.452, min=-100, max=100, vary=False)
params.add("SiO2_n1", value=36.0, min=-40000, max=40000, vary=False)
params.add("SiO2_n2", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_k0", value=0, min=-100, max=100, vary=False)
params.add("SiO2_k1", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_k2", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_d", value=20, min=0, max=40000, vary=True)
```

Load silicon dispersion from the refractiveindexinfo database

You can load any material from the index refractiveindex.info, which is embedded into the software (so you may use it offline, too). Here, we are interested in the literature values for the silicon substrate. First we need to load the database with `rii_db = elli.db.RII()` and then we can query it with `rii_db.get_mat("Si", "Aspnes")` to load this entry.

```
rii_db = elli.db.RII()
Si = rii_db.get_mat("Si", "Aspnes")
```

Building the model

For simple parameter estimation, the fit decorator (**@fit**) in conjunction with the model definition is used. The fitting decorator takes a pandas dataframe containing the psi/delta measurement data (**psi_delta**) and the model parameters (**params**) as an input. It then passes the wavelength from measurement dataframe (**lbda**) and the parameters to the actual model function.

Inside the model function the optical model is built, i.e. the Si literature values are loaded and the fitting parameters are filled into the Cauchy dispersion. For details on how to insert data into the Cauchy model or other optical dispersion models, you may refer to the documentation of pyElli. Please keep in mind that the parameters you use here have to be defined in the parameter object **param**.

From the dispersion model isotropic materials are generated (could also be an anisotropic material, refer to the docs for an overview). This is done by calling the `elli.IsotropicMaterial(...)` function with a dispersion model as a parameter or simply calling `.get_mat()` on a dispersion model. These two approaches are equivalent. From these materials the layer is build, which only consists of the SiO2 layer in this example. The final structure consists of an incoming half-space, the layers and an outgoing half space. Specifically, typically the light is coming from air and finally gets absorbed by the bulk material, in our example this is Si, i.e. we call `elli.Structure(elli.AIR, Layer, Si)`.

To provide simulated data, we have to evaluate the structure by calling the `evaluate(...)` function, which takes the experimental wavelength array **lbda**, **ANGLE** under which the experiment was performed and the solver to be used to solve the transfer-matrix problem. Here, we use a simple 2x2 matrix approach, which splits the interaction in s and p-parts and therefore cannot account for anisotropy. There exist 4x4 matrix solvers as well. You may refer to the *solver documentation* for further details.

Executing the cell below in a jupyter notebook displays a comparison of the simulated / values at the current parameter values with their measured counterparts. Additionally, input fields for each model parameter are shown. You may change the parameters and the calculated data will change accordingly. For clarification the modeled data is shown with *_calc* postfix in the legend.

```
@fit(psi_delta, params)
def model(lbda, params):
    # Generate the cauchy model from the current lmfit parameters
    SiO2 = elli.Cauchy(
        params["SiO2_n0"],
        params["SiO2_n1"],
        params["SiO2_n2"],
        params["SiO2_k0"],
        params["SiO2_k1"],
        params["SiO2_k2"],
    ).get_mat()
    # get_mat() generates an IsotropicMaterial from the dispersion relation

    # Construct the layers you expect in your sample
    # Here, it only consists of one layer SiO2 in between air and Si.
    # We build the structure coming from air, through the layers,
    # represented as an array, and having Si as bulk material.
    structure = elli.Structure(
        elli.AIR,  # Input medium
        [elli.Layer(SiO2, params["SiO2_d"])],  # Overlayer structure
        Si,
    )  # Output medium / Substrate

    # The model should return the evaluation of the structure at the experimental␣
↪wavelengths lbda,
    # the experimental angle ANGLE and it should define a solver to calculate the␣
↪transfer matrix.
    return structure.evaluate(lbda, ANGLE, solver=elli.Solver2x2)
```

Fit and plot fit results

The fit of the data can be executed by calling the fit() function on the model function, which automatically gets attached by the @fit decorator in the cell above. The following cell basically executes the fit and plots a comparison between the measurement and fitted data.

```
fit_stats = model.fit()
model.plot()
```

```
FigureWidget({
    'data': [{'hovertemplate': 'variable=<br>Wavelength=%{x}<br>value=%{y}<extra></extra>
↪',
              'legendgroup': '',
              'line': {'color': '#636efa', 'dash': 'solid'},
```

(continues on next page)

(continued from previous page)

```
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'showlegend': True,
              'type': 'scattergl',
              'uid': 'e4001476-a7e8-4d4d-a579-adf38dc39191',
              'x': array([210., 211., 212., ..., 798., 799., 800.], shape=(1182,)),
              'xaxis': 'x',
              'y': array([36.548481, 36.480022, 36.433453, ...,       nan,       nan,␣
↪   nan],
                         shape=(1182,)),
              'yaxis': 'y'},
             {'hovertemplate': 'variable=<br>Wavelength=%{x}<br>value=%{y}<extra></extra>
↪',
              'legendgroup': '',
              'line': {'color': '#EF553B', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'showlegend': True,
              'type': 'scattergl',
              'uid': 'd847ce2b-2397-43a2-b502-9772e5161a24',
              'x': array([210., 211., 212., ..., 798., 799., 800.], shape=(1182,)),
              'xaxis': 'x',
              'y': array([97.499702, 97.92762 , 98.323624, ...,       nan,       nan,␣
↪   nan],
                         shape=(1182,)),
              'yaxis': 'y'},
             {'hovertemplate': 'variable=_fit<br>Wavelength=%{x}<br>value=%{y}<extra></
↪extra>',
              'legendgroup': '_fit',
              'line': {'color': '#00cc96', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '_fit',
              'showlegend': True,
              'type': 'scattergl',
              'uid': '12378046-5523-43dc-aa21-119f9735e67b',
              'x': array([210., 211., 212., ..., 798., 799., 800.], shape=(1182,)),
              'xaxis': 'x',
              'y': array([       nan,        nan,        nan, ..., 9.18875929, 9.1824316␣
↪',
                          9.17610219], shape=(1182,)),
              'yaxis': 'y'},
             {'hovertemplate': 'variable=_fit<br>Wavelength=%{x}<br>value=%{y}<extra></
↪extra>',
              'legendgroup': '_fit',
              'line': {'color': '#ab63fa', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '_fit',
              'showlegend': True,
```

(continues on next page)

(continued from previous page)

```
            'type': 'scattergl',
            'uid': '78585ed3-26e1-45c9-a7d8-9af188213af0',
            'x': array([210., 211., 212., ..., 798., 799., 800.], shape=(1182,)),
            'xaxis': 'x',
            'y': array([         nan,          nan,          nan, ..., 174.48896139,
                      174.49555625, 174.50214175], shape=(1182,)),
            'yaxis': 'y'}],
    'layout': {'legend': {'title': {'text': 'variable'}, 'tracegroupgap': 0},
               'margin': {'t': 60},
               'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text':
↪'Wavelength'}},
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'value'}}
↪}
})
```

Extracting the optical properties from the fit

Since we want to extract the dispersion relation of a layer in our measured stack, we can use our fitted parameters. The fit parameters are contained in the fits output `params` attribute, i.e. `fit_stats.params` for our example. We can use it to call our dispersion relation we used in our model (here it is a Cauchy dispersion relation) and fill in our fitted value. By calling `get_dielectric_df()` we can get the dielectric function of the material, which is plotted here as an example. `get_dielectric_df` uses a default wavelength range which can also be changed by inputting a wavelength array as a parameter.

```
fitted_model = elli.Cauchy(
    fit_stats.params["SiO2_n0"],
    fit_stats.params["SiO2_n1"],
    fit_stats.params["SiO2_n2"],
    fit_stats.params["SiO2_k0"],
    fit_stats.params["SiO2_k1"],
    fit_stats.params["SiO2_k2"],
)

fitted_model.get_dielectric_df().plot(backend="plotly")
```

We can also call `get_refractive_index_df()` to get the refractive index of the material as dataframe.

```
fitted_model.get_refractive_index_df().plot(backend="plotly")
```

If you want to write your data to a file you simply call pandas `to_csv(...)` function to write a csv file, i.e. for the dielectric function this writes as

```
fitted_model.get_dielectric_df().to_csv("SiO2_diel_func.csv")
```

You may also access a single value of your optical model

```
fit_stats.params["SiO2_n0"].value
```

```
1.452
```

Our simply print the fitted values in a list together with their fitting errors

```
fit_stats.params
```

Show fit statistics

Now, we may also print out the fit statictics from the model fit in the cell above. The fit statistics are simple lmfit fit statistics, too. Typically, one uses chi square values as a figure of merit for the fit quality. It is stored in the chisqr attribute of the fit_stats variable we defined above.

```
fit_stats.chisqr
```

```
np.float64(0.016923374970708012)
```

We can print the full fit statistics, too.

```
fit_stats
```

References

Here you can find the latest jupyter notebook and data files of this example.

Total running time of the script: (0 minutes 1.247 seconds)

1.2.2 Multilayer fit

Fits a multilayer model to an ALD grown TiO2 sample on SiO2 / Si.

```
import elli
from elli.fitting import ParamsHist, fit
```

Load data

Load data collected with Sentech Ellipsometer and cut the spectral range (to use Si Aspnes file)

The sample is an ALD grown TiO2 sample (with 400 cycles) on commercially available SiO2 / Si substrate.

```
tss = elli.read_spectraray_psi_delta("TiO2_400cycles.txt").loc[70.06].loc[400:800]
```

Set start parameters

Here we set the start parameters for the TiO2 and SiO2 layer. We set the SiO2 layer parameters to a fixed value from another fit of the substrate. See the Basic usage example for details on how to perform such a fit. In general it is a good idea to fit your data layer-wise if possible to yield a better fit quality.

```
params = ParamsHist()
params.add("SiO2_n0", value=1.452, min=-100, max=100, vary=False)
params.add("SiO2_n1", value=36.0, min=-40000, max=40000, vary=False)
params.add("SiO2_n2", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_k0", value=0, min=-100, max=100, vary=False)
params.add("SiO2_k1", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_k2", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_d", value=276.36, min=0, max=40000, vary=False)

params.add("TiO2_n0", value=2.236, min=-100, max=100, vary=True)
params.add("TiO2_n1", value=451, min=-40000, max=40000, vary=True)
```

(continues on next page)

(continued from previous page)

```
params.add("TiO2_n2", value=251, min=-40000, max=40000, vary=True)
params.add("TiO2_k0", value=0, min=-100, max=100, vary=False)
params.add("TiO2_k1", value=0, min=-40000, max=40000, vary=False)
params.add("TiO2_k2", value=0, min=-40000, max=40000, vary=False)

params.add("TiO2_d", value=20, min=0, max=40000, vary=True)
```

Load silicon dispersion from the refractiveindexinfo database

You can load any material from the index refractiveindex.info, which is embedded into the software (so you may use it offline, too). Here, we are interested in the literature values for the silicon substrate. First we need to load the database with `rii_db = elli.db.RII()` and then we can query it with `rii_db.get_mat("Si", "Aspnes")` to load this entry.

```
rii_db = elli.db.RII()
Si = rii_db.get_mat("Si", "Aspnes")
```

Building the model

Here the model is build and the experimental structure is returned. For details on this process please refer to the Basic usage example. When executed in an jupyter notebook this displays an interactive graph with which you can select the start parameters before fitting the data.

```
@fit(tss, params)
def model(lbda, params):
    SiO2 = elli.Cauchy(
        params["SiO2_n0"],
        params["SiO2_n1"],
        params["SiO2_n2"],
        params["SiO2_k0"],
        params["SiO2_k1"],
        params["SiO2_k2"],
    ).get_mat()
    TiO2 = elli.Cauchy(
        params["TiO2_n0"],
        params["TiO2_n1"],
        params["TiO2_n2"],
        params["TiO2_k0"],
        params["TiO2_k1"],
        params["TiO2_k2"],
    ).get_mat()

    Layer = [elli.Layer(TiO2, params["TiO2_d"]), elli.Layer(SiO2, params["SiO2_d"])]

    return elli.Structure(elli.AIR, Layer, Si).evaluate(lbda, 70, solver=elli.Solver2x2)
    # Alternative: Use 4x4 Solver with scipy propagator
    # return elli.Structure(elli.AIR, Layer, Si).evaluate(lbda, 70, solver=elli.
↪Solver4x4, propagator=elli.PropagatorExpm())
```

Plot & Fit model

We plot the model to see the deviation with the initial parameters.

```
model.plot()
```

```
FigureWidget({
    'data': [{'hovertemplate': 'variable=<br>Wavelength=%{x}<br>value=%{y}<extra></extra>
↪',
              'legendgroup': '',
              'line': {'color': '#636efa', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'showlegend': True,
              'type': 'scattergl',
              'uid': 'fb5bf86c-aabf-402b-a95a-b0e940e52235',
              'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
                         shape=(1852,)),
              'xaxis': 'x',
              'y': array([32.75947, 32.84076, 32.84675, ...,      nan,      nan,     ␣
↪nan],
                         shape=(1852,)),
              'yaxis': 'y'},
             {'hovertemplate': 'variable=<br>Wavelength=%{x}<br>value=%{y}<extra></extra>
↪',
              'legendgroup': '',
              'line': {'color': '#EF553B', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'showlegend': True,
              'type': 'scattergl',
              'uid': '9eba3673-b7e8-4d01-9af7-7432636bbb67',
              'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
                         shape=(1852,)),
              'xaxis': 'x',
              'y': array([-132.57268, -132.11725, -131.5141 , ...,        nan,       ␣
↪nan,
                                nan], shape=(1852,)),
              'yaxis': 'y'},
             {'hovertemplate': 'variable=_fit<br>Wavelength=%{x}<br>value=%{y}<extra></
↪extra>',
              'legendgroup': '_fit',
              'line': {'color': '#00cc96', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '_fit',
              'showlegend': True,
              'type': 'scattergl',
              'uid': '663a7d35-a879-49f7-8dc1-d9ccd4a05b3d',
              'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
```

(continues on next page)

(continued from previous page)

```
↪799.71891],
                          shape=(1852,)),
              'xaxis': 'x',
              'y': array([        nan,         nan,         nan, ..., 21.13502205, 21.
↪16231399,
                          21.18955407], shape=(1852,)),
              'yaxis': 'y'},
             {'hovertemplate': 'variable=_fit<br>Wavelength=%{x}<br>value=%{y}<extra></
↪extra>',
              'legendgroup': '_fit',
              'line': {'color': '#ab63fa', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '_fit',
              'showlegend': True,
              'type': 'scattergl',
              'uid': 'f29299e0-376f-499f-90fc-09ce40160e05',
              'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
                          shape=(1852,)),
              'xaxis': 'x',
              'y': array([         nan,          nan,          nan, ..., -111.
↪09599625,
                          -110.98086027, -110.86609469], shape=(1852,)),
              'yaxis': 'y'}],
    'layout': {'legend': {'title': {'text': 'variable'}, 'tracegroupgap': 0},
               'margin': {'t': 60},
               'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text':
↪'Wavelength'}},
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'value'}}
↪}
})
```

Now lets perform the fit and plot the comparison of calculation and experimental data afterwards.

```
fit_stats = model.fit()
model.plot()
```

```
FigureWidget({
    'data': [{'hovertemplate': 'variable=<br>Wavelength=%{x}<br>value=%{y}<extra></extra>
↪',
              'legendgroup': '',
              'line': {'color': '#636efa', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'showlegend': True,
              'type': 'scattergl',
              'uid': '47ae734b-cb6f-447a-8539-12f1de4f01f0',
              'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
```

(continues on next page)

(continued from previous page)

```
                        shape=(1852,)),
             'xaxis': 'x',
             'y': array([32.75947, 32.84076, 32.84675, ...,      nan,      nan,     ␣
↪nan],
                        shape=(1852,)),
             'yaxis': 'y'},
            {'hovertemplate': 'variable=<br>Wavelength=%{x}<br>value=%{y}<extra></extra>
↪',
             'legendgroup': '',
             'line': {'color': '#EF553B', 'dash': 'solid'},
             'marker': {'symbol': 'circle'},
             'mode': 'lines',
             'name': '',
             'showlegend': True,
             'type': 'scattergl',
             'uid': '8c339219-8be3-4a41-b89f-aa50858a51ae',
             'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
                        shape=(1852,)),
             'xaxis': 'x',
             'y': array([-132.57268, -132.11725, -131.5141 , ...,        nan,       ␣
↪nan,
                            nan], shape=(1852,)),
             'yaxis': 'y'},
            {'hovertemplate': 'variable=_fit<br>Wavelength=%{x}<br>value=%{y}<extra></
↪extra>',
             'legendgroup': '_fit',
             'line': {'color': '#00cc96', 'dash': 'solid'},
             'marker': {'symbol': 'circle'},
             'mode': 'lines',
             'name': '_fit',
             'showlegend': True,
             'type': 'scattergl',
             'uid': 'c46707d8-305a-4a2f-bc5a-908b1d9b752d',
             'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
                        shape=(1852,)),
             'xaxis': 'x',
             'y': array([       nan,         nan,         nan, ..., 20.20488814, 20.
↪23141868,
                         20.25790068], shape=(1852,)),
             'yaxis': 'y'},
            {'hovertemplate': 'variable=_fit<br>Wavelength=%{x}<br>value=%{y}<extra></
↪extra>',
             'legendgroup': '_fit',
             'line': {'color': '#ab63fa', 'dash': 'solid'},
             'marker': {'symbol': 'circle'},
             'mode': 'lines',
             'name': '_fit',
             'showlegend': True,
             'type': 'scattergl',
             'uid': 'f334f609-ee0c-4b63-8093-7ce816309490',
```

(continues on next page)

(continued from previous page)

```
             'x': array([400.07646, 400.51975, 400.96301, ..., 798.88197, 799.30046,␣
↪799.71891],
                        shape=(1852,)),
             'xaxis': 'x',
             'y': array([          nan,           nan,           nan, ..., -118.
↪64650333,
                         -118.5165033 , -118.38691231], shape=(1852,)),
             'yaxis': 'y'}],
   'layout': {'legend': {'title': {'text': 'variable'}, 'tracegroupgap': 0},
              'margin': {'t': 60},
              'template': '...',
              'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text':
↪'Wavelength'}},
              'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'value'}}
↪}
})
```

We can also have a look at the fit statistics.

```
fit_stats
```

References

Here you can find the latest jupyter notebook and data files of this example.

Total running time of the script: (0 minutes 1.161 seconds)

1.2.3 Custom fitting example

This is a short example for fitting data, which is not covered by the fitting widget. It relies on calling lmfit manually. Therefore one needs to provide a fitting function, which calculates the numerical residual between measurement and model. This notebook is about fitting multiple datasets from different angles of incidents simultaneously, but can be adapted to a wide range of different use cases.

```
import numpy as np
import elli
from elli.fitting import ParamsHist
from lmfit import minimize, fit_report
import matplotlib.pyplot as plt
```

Data import

Read (psi) and (delta) ellipsometric spectra for different angles from a NeXus file, limited to a wavelength range of 210–800 nm, to be compatible with a tabulated Silicon dispersion.

```
data = elli.read_nexus_psi_delta("SiO2onSi.ellips.nxs").loc[
    (slice(None), slice(210, 800)), :
]
lbda = data.loc[50].index.get_level_values("Wavelength").to_numpy()
data
```

Setting up invariant materials and fitting parameters

Set up the optical constants for the substrate from RII and define initial fit parameters for the Cauchy SiO_2 layer.

```
rii_db = elli.db.RII()
Si = rii_db.get_mat("Si", "Aspnes")

params = ParamsHist()
params.add("SiO2_n0", value=1.452, min=-100, max=100, vary=True)
params.add("SiO2_n1", value=36.0, min=-40000, max=40000, vary=True)
params.add("SiO2_n2", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_k0", value=0, min=-100, max=100, vary=False)
params.add("SiO2_k1", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_k2", value=0, min=-40000, max=40000, vary=False)
params.add("SiO2_d", value=20, min=0, max=40000, vary=True)
```

Model helper function

This model function is not strictly needed, but simplifies the fit function, as the model only needs to be defined once.

```
def model(lbda, angle, params):
    SiO2 = elli.Cauchy(
        params["SiO2_n0"],
        params["SiO2_n1"],
        params["SiO2_n2"],
        params["SiO2_k0"],
        params["SiO2_k1"],
        params["SiO2_k2"],
    ).get_mat()

    structure = elli.Structure(
        elli.AIR,
        [elli.Layer(SiO2, params["SiO2_d"])],
        Si,
    )

    return structure.evaluate(lbda, angle, solver=elli.Solver2x2)
```

Defining the fit function

The fit function follows the protocol defined by the lmfit package and needs the parameters dictionary as first argument. It has to return a residual value, which will be minimized. Here psi and delta are used across all angles to calculate the residual, but could be changed to any other measured quantity like transmission or reflection data.

```
def fit_function(params, lbda, data):
    residual = []

    for phi_i in [50, 60, 70]:
        model_result = model(lbda, phi_i, params)

        resid_psi = data.loc[(phi_i, "")].to_numpy() - model_result.psi
        resid_delta = data.loc[(phi_i, "")].to_numpy() - model_result.delta

        residual.append(resid_psi)
```

(continues on next page)

(continued from previous page)

```
        residual.append(resid_delta)

    return np.concatenate(residual)
```

Running the fit

The fitting is performed by calling the minimize function with the fit_function and the needed arguments. It is possible to change the underlying algorithm by providing the method kwarg. The fit_report function provides fitted values, uncertainties, and goodness-of-fit statistics.

```
out = minimize(fit_function, params, args=(lbda, data), method="leastsq")
print(fit_report(out))
```

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 119
    # data points      = 3546
    # variables        = 3
    chi-square         = 157.748841
    reduced chi-square = 0.04452409
    Akaike info crit   = -11031.1779
    Bayesian info crit = -11012.6572
[[Variables]]
    SiO2_n0:  1.63549687 +/- 0.03012997 (1.84%) (init = 1.452)
    SiO2_n1:  27.6408771 +/- 8.75248298 (31.66%) (init = 36)
    SiO2_n2:  0 (fixed)
    SiO2_k0:  0 (fixed)
    SiO2_k1:  0 (fixed)
    SiO2_k2:  0 (fixed)
    SiO2_d:   1.75831329 +/- 0.02404272 (1.37%) (init = 20)
[[Correlations]] (unreported correlations are < 0.100)
    C(SiO2_n0, SiO2_d)  = -0.9916
    C(SiO2_n0, SiO2_n1) = -0.9596
    C(SiO2_n1, SiO2_d)  = +0.9259
```

Plotting the results

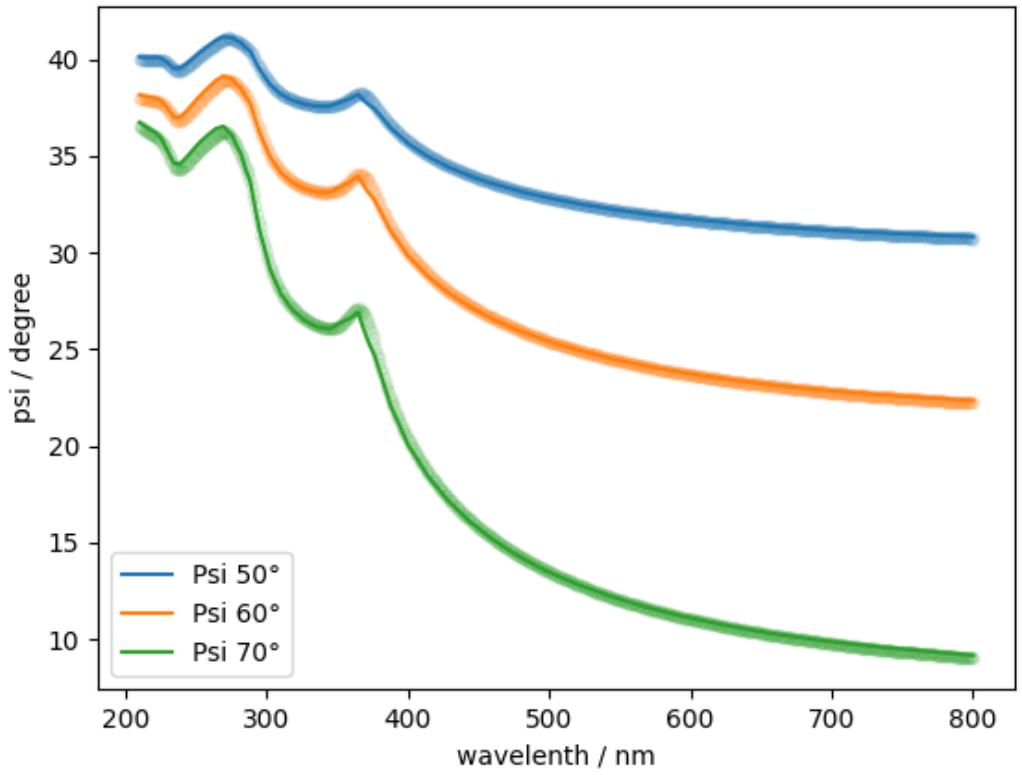
The measurent results are displayed as scatter plot and the PyElli results are overlayed as solid lines.

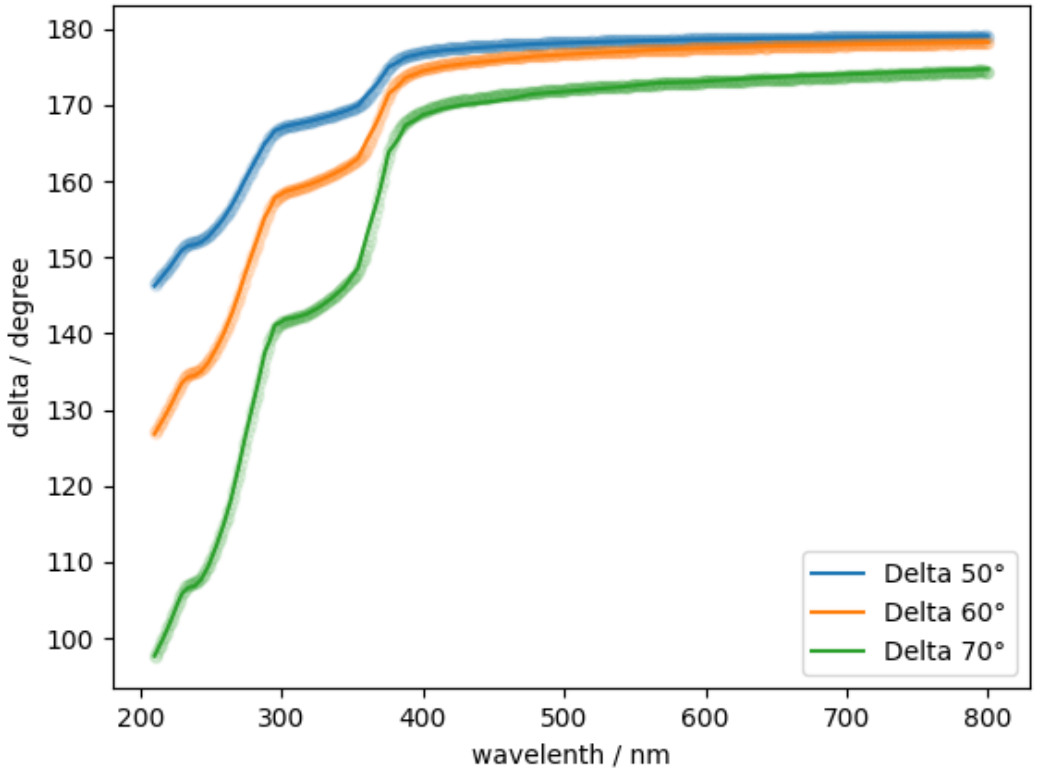
```
fit_50 = model(lbda, 50, out.params)
fit_60 = model(lbda, 60, out.params)
fit_70 = model(lbda, 70, out.params)

fig = plt.figure(dpi=100)
ax = fig.add_subplot(1, 1, 1)
ax.scatter(lbda, data.loc[(50, "")], s=20, alpha=0.1, label="50° Measurement")
ax.scatter(lbda, data.loc[(60, "")], s=20, alpha=0.1, label="Psi 60° Measurement")
ax.scatter(lbda, data.loc[(70, "")], s=20, alpha=0.1, label="Psi 70° Measurement")
(psi50,) = ax.plot(lbda, fit_50.psi, c="tab:blue", label="Psi 50°")
(psi60,) = ax.plot(lbda, fit_60.psi, c="tab:orange", label="Psi 60°")
(psi70,) = ax.plot(lbda, fit_70.psi, c="tab:green", label="Psi 70°")
ax.set_xlabel("wavelenth / nm")
```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel("psi / degree")
ax.legend(handles=[psi50, psi60, psi70], loc="lower left")
fig.canvas.draw()

fig = plt.figure(dpi=100)
ax = fig.add_subplot(1, 1, 1)
ax.scatter(lbda, data.loc[(50, "")], s=20, alpha=0.1, label="Delta 50° Measurement")
ax.scatter(lbda, data.loc[(60, "")], s=20, alpha=0.1, label="Delta 60° Measurement")
ax.scatter(lbda, data.loc[(70, "")], s=20, alpha=0.1, label="Delta 70° Measurement")
(delta50,) = ax.plot(lbda, fit_50.delta, c="tab:blue", label="Delta 50°")
(delta60,) = ax.plot(lbda, fit_60.delta, c="tab:orange", label="Delta 60°")
(delta70,) = ax.plot(lbda, fit_70.delta, c="tab:green", label="Delta 70°")
ax.set_xlabel("wavelenth / nm")
ax.set_ylabel("delta / degree")
ax.legend(handles=[delta50, delta60, delta70], loc="lower right")
fig.canvas.draw()
```





-

Total running time of the script: (0 minutes 1.702 seconds)

1.2.4 Mueller matrix

Exemplary fit of the complete Mueller matrix of a SiO2 on Si measurement.

```
import elli
from elli.fitting import ParamsHist, fit_mueller_matrix
```

Read data

We load the data from an ascii file containing each of the mueller matrix elements. The wavelength range is cut to be in between 210 nm and 820 nm, to stay in the range of the provided literature values for Si. The data is expected to be in a pandas dataframe containing the columns Mxy, where x and y refer to the matrix element inside the mueller matrix. The data is scaled by the M11 element, such that $M_{11} = 1$ for all wavelengths. To show the structure we print the *MM* dataframe. If you load your data from another source make sure it adheres to this form.

```
MM = elli.read_spectraray_mmatrix("Wafer_MM_70.txt").loc[210:820]
print(MM)
```

```
            M11       M12       M13       M14  ...       M41       M42       M43       M44
Wavelength                                      ...
210.30366   1.0 -0.09147 -0.01241 -0.00614  ...  0.00986  0.00048 -0.95574  0.25821
210.76102   1.0 -0.10879 -0.01151 -0.00901  ...  0.00917  0.01319 -0.96899  0.24432
211.21834   1.0 -0.10639 -0.00694 -0.00475  ...  0.00712  0.00891 -0.96191  0.24986
```

(continues on next page)

(continued from previous page)

```
211.67561   1.0 -0.11544 -0.02094 -0.00348  ...  0.01877  0.01726 -0.95951  0.25408
212.13284   1.0 -0.12199  0.02182  0.00480  ... -0.01790 -0.01957 -0.95493  0.28308
...         ...      ...      ...      ...  ...      ...      ...      ...      ...
818.08934   1.0 -0.44285 -0.00048 -0.00288  ...  0.00292 -0.00276 -0.87949  0.15713
818.50589   1.0 -0.44301  0.00376  0.00118  ... -0.00159 -0.01090 -0.87876  0.16422
818.92240   1.0 -0.44245  0.00887  0.00626  ... -0.00585 -0.01388 -0.87960  0.16743
819.33886   1.0 -0.44406 -0.00646 -0.00610  ...  0.00882  0.00424 -0.87907  0.15105
819.75528   1.0 -0.44381  0.01170  0.00550  ... -0.00692 -0.01260 -0.87849  0.16637

[1396 rows x 16 columns]
```

Setting start parameters

Here we set the start parameters for the SiO2 cauchy dispersion and thickness of the layer.

```
params = ParamsHist()
params.add("SiO2_n0", value=1.452, min=-100, max=100, vary=True)
params.add("SiO2_n1", value=36.0, min=-40000, max=40000, vary=True)
params.add("SiO2_n2", value=0, min=-40000, max=40000, vary=True)
params.add("SiO2_k0", value=0, min=-100, max=100, vary=True)
params.add("SiO2_k1", value=0, min=-40000, max=40000, vary=True)
params.add("SiO2_k2", value=0, min=-40000, max=40000, vary=True)
params.add("SiO2_d", value=120, min=0, max=40000, vary=True)
```

Load silicon dispersion from the refractiveindexinfo database

You can load any material from the index refractiveindex.info, which is embedded into the software (so you may use it offline, too). Here, we are interested in the literature values for the silicon substrate. First we need to load the database with `rii_db = elli.db.RII()` and then we can query it with `rii_db.get_mat("Si", "Aspnes")` to load this entry.

```
rii_db = elli.db.RII()
Si = rii_db.get_mat("Si", "Aspnes")
```

Building the model

Here the model is build and the experimental structure is returned. For details on this process please refer to the Basic usage example. When executed in an jupyter notebook this displays an interactive graph with which you can select the start parameters before fitting the data.

```
@fit_mueller_matrix(MM, params, display_single=False, sharex=True, full_scale=False)
def model(lbda, params):
    SiO2 = elli.Cauchy(
        params["SiO2_n0"],
        params["SiO2_n1"],
        params["SiO2_n2"],
        params["SiO2_k0"],
        params["SiO2_k1"],
        params["SiO2_k2"],
    ).get_mat()

    Layer = [elli.Layer(SiO2, params["SiO2_d"])]
```

(continues on next page)

(continued from previous page)

```
    return elli.Structure(elli.AIR, Layer, Si).evaluate(
        lbda, 70, solver=elli.Solver4x4, propagator=elli.PropagatorExpm()
    )
```

Plot & Fit the model

Here we plot the model at the initial parameter set vs. the experimental data.

```
model.plot()
```

```
FigureWidget({
    'data': [{'line': {'color': '#636EFA', 'dash': 'solid'},
              'name': 'M11 ',
              'type': 'scatter',
              'uid': '644ed670-4c8c-43c2-b2a5-a501015654f4',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x',
              'y': array([1., 1., 1., ..., 1., 1., 1.], shape=(1396,)),
              'yaxis': 'y'},
             {'line': {'color': '#636EFA', 'dash': 'dash'},
              'name': 'M11 theory',
              'type': 'scatter',
              'uid': '2243d4dc-4705-4d18-aa89-e1f04e7a7d7d',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x',
              'y': array([1., 1., 1., ..., 1., 1., 1.], shape=(1396,)),
              'yaxis': 'y'},
             {'line': {'color': '#EF553B', 'dash': 'solid'},
              'name': 'M12 ',
              'type': 'scatter',
              'uid': '0f745bf0-a1fd-4cd1-be83-4190ed1e1abc',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x2',
              'y': array([-0.09147, -0.10879, -0.10639, ..., -0.44245, -0.44406, -0.
↪44381],
                         shape=(1396,)),
              'yaxis': 'y2'},
             {'line': {'color': '#EF553B', 'dash': 'dash'},
              'name': 'M12 theory',
              'type': 'scatter',
              'uid': '58e9ac1b-26a1-4630-a059-567af16b315e',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x2',
```

(continues on next page)

(continued from previous page)

```
             'y': array([ 0.48855385,  0.48240998,  0.47406745, ..., -0.27742534, -0.
↪27818747,
                         -0.27894785], shape=(1396,)),
             'yaxis': 'y2'},
            {'line': {'color': '#00CC96', 'dash': 'solid'},
             'name': 'M13 ',
             'type': 'scatter',
             'uid': 'd04ad804-7894-4026-90b7-9b2f55459fb6',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x3',
             'y': array([-0.01241, -0.01151, -0.00694, ...,  0.00887, -0.00646,  0.0117␣
↪],
                        shape=(1396,)),
             'yaxis': 'y3'},
            {'line': {'color': '#00CC96', 'dash': 'dash'},
             'name': 'M13 theory',
             'type': 'scatter',
             'uid': 'c23154d1-05b4-4310-9ded-9acf63b09364',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x3',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y3'},
            {'line': {'color': '#AB63FA', 'dash': 'solid'},
             'name': 'M14 ',
             'type': 'scatter',
             'uid': '554e8fab-37b9-4872-9bed-ac5aa02572e9',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x4',
             'y': array([-0.00614, -0.00901, -0.00475, ...,  0.00626, -0.0061 ,  0.0055␣
↪],
                        shape=(1396,)),
             'yaxis': 'y4'},
            {'line': {'color': '#AB63FA', 'dash': 'dash'},
             'name': 'M14 theory',
             'type': 'scatter',
             'uid': '09f91e70-879b-4358-8244-b97f05f7572f',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x4',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y4'},
            {'line': {'color': '#FFA15A', 'dash': 'solid'},
             'name': 'M21 ',
             'type': 'scatter',
             'uid': '62a1179d-147f-4cc8-890c-091808d24d06',
```

(continues on next page)

(continued from previous page)

```
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x5',
             'y': array([-0.09605, -0.09754, -0.10761, ..., -0.44024, -0.44126, -0.
↪44135],
                        shape=(1396,)),
             'yaxis': 'y5'},
            {'line': {'color': '#FFA15A', 'dash': 'dash'},
             'name': 'M21 theory',
             'type': 'scatter',
             'uid': 'e786d9e5-06c0-48a5-90a3-fa87eddcf6cd',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x5',
             'y': array([ 0.48855385,  0.48240998,  0.47406745, ..., -0.27742534, -0.
↪27818747,
                         -0.27894785], shape=(1396,)),
             'yaxis': 'y5'},
            {'line': {'color': '#19D3F3', 'dash': 'solid'},
             'name': 'M22 ',
             'type': 'scatter',
             'uid': '01ed7e97-8ada-4e89-8565-3422f82a9636',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x6',
             'y': array([0.99426, 0.99475, 0.99637, ..., 0.99723, 0.99661, 0.99606],
                        shape=(1396,)),
             'yaxis': 'y6'},
            {'line': {'color': '#19D3F3', 'dash': 'dash'},
             'name': 'M22 theory',
             'type': 'scatter',
             'uid': 'ba6b0737-b38e-48f8-9049-ace9a2799fab',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x6',
             'y': array([1., 1., 1., ..., 1., 1., 1.], shape=(1396,)),
             'yaxis': 'y6'},
            {'line': {'color': '#FF6692', 'dash': 'solid'},
             'name': 'M23 ',
             'type': 'scatter',
             'uid': '44da94cb-b638-4f8a-8b7e-26cf41d66ada',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x7',
             'y': array([-0.00364, -0.00427, -0.00207, ..., -0.00457,  0.00234, -0.
↪00653],
                        shape=(1396,)),
```

(continues on next page)

(continued from previous page)

```
             'yaxis': 'y7'},
            {'line': {'color': '#FF6692', 'dash': 'dash'},
             'name': 'M23 theory',
             'type': 'scatter',
             'uid': 'dabbe45f-300c-4e86-a70f-51a282ca2f8d',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x7',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y7'},
            {'line': {'color': '#B6E880', 'dash': 'solid'},
             'name': 'M24 ',
             'type': 'scatter',
             'uid': 'cc46df31-8a94-472f-a318-da7889871b55',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x8',
             'y': array([ 0.01084, -0.00325,  0.01123, ..., -0.00263,  0.00227, -0.
↪00311],
                        shape=(1396,)),
             'yaxis': 'y8'},
            {'line': {'color': '#B6E880', 'dash': 'dash'},
             'name': 'M24 theory',
             'type': 'scatter',
             'uid': '24be9419-7b44-40d1-81f0-fb38846bd915',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x8',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y8'},
            {'line': {'color': '#FF97FF', 'dash': 'solid'},
             'name': 'M31 ',
             'type': 'scatter',
             'uid': '3e62da3d-c445-4396-bc0b-527c6bb58abc',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x9',
             'y': array([-0.00278, -0.00634, -0.00219, ...,  0.00055, -0.00105,  0.
↪00037],
                        shape=(1396,)),
             'yaxis': 'y9'},
            {'line': {'color': '#FF97FF', 'dash': 'dash'},
             'name': 'M31 theory',
             'type': 'scatter',
             'uid': '97110c12-8718-4ffc-8b39-2a3ae8db864d',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
```

(continues on next page)

(continued from previous page)

```
             'xaxis': 'x9',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y9'},
            {'line': {'color': '#FECB52', 'dash': 'solid'},
             'name': 'M32 ',
             'type': 'scatter',
             'uid': 'b3307db9-6556-4541-815a-922969b0464a',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x10',
             'y': array([-0.00212, -0.00045,  0.0003 , ..., -0.0018 , -0.00042,  0.
↪00037],
                        shape=(1396,)),
             'yaxis': 'y10'},
            {'line': {'color': '#FECB52', 'dash': 'dash'},
             'name': 'M32 theory',
             'type': 'scatter',
             'uid': '3eed210e-96b1-48c6-93d0-a2d1e516999f',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x10',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y10'},
            {'line': {'color': '#636EFA', 'dash': 'solid'},
             'name': 'M33 ',
             'type': 'scatter',
             'uid': '29d3bd75-4fa6-4660-8dae-83f5781f5487',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x11',
             'y': array([0.27283, 0.26726, 0.2728 , ..., 0.16236, 0.16271, 0.16255],
                        shape=(1396,)),
             'yaxis': 'y11'},
            {'line': {'color': '#636EFA', 'dash': 'dash'},
             'name': 'M33 theory',
             'type': 'scatter',
             'uid': '1f8c33fb-d183-4920-97aa-a2252295d0e3',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x11',
             'y': array([-0.78177026, -0.75168899, -0.71871123, ...,  0.20163327,  0.
↪20156999,
                         0.20150625], shape=(1396,)),
             'yaxis': 'y11'},
            {'line': {'color': '#EF553B', 'dash': 'solid'},
             'name': 'M34 ',
             'type': 'scatter',
             'uid': '2a20f1fd-4667-4973-bc1d-b1be57a967d4',
```

(continues on next page)

(continued from previous page)

```
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x12',
              'y': array([0.95555, 0.9645 , 0.95821, ..., 0.87712, 0.87629, 0.87661],
                         shape=(1396,)),
              'yaxis': 'y12'},
             {'line': {'color': '#EF553B', 'dash': 'dash'},
              'name': 'M34 theory',
              'type': 'scatter',
              'uid': '23644f80-0592-4cee-9147-f6904775b5aa',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x12',
              'y': array([0.38749245, 0.44971578, 0.50863958, ..., 0.93935042, 0.
↪93913858,
                          0.93892669], shape=(1396,)),
              'yaxis': 'y12'},
             {'line': {'color': '#00CC96', 'dash': 'solid'},
              'name': 'M41 ',
              'type': 'scatter',
              'uid': 'c1c32fd5-9ce3-4e9d-9bad-679047e3da78',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x13',
              'y': array([ 0.00986,  0.00917,  0.00712, ..., -0.00585,  0.00882, -0.
↪00692],
                         shape=(1396,)),
              'yaxis': 'y13'},
             {'line': {'color': '#00CC96', 'dash': 'dash'},
              'name': 'M41 theory',
              'type': 'scatter',
              'uid': '510b3cf2-9f24-4413-9fd7-7306ce8d62a1',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x13',
              'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
              'yaxis': 'y13'},
             {'line': {'color': '#AB63FA', 'dash': 'solid'},
              'name': 'M42 ',
              'type': 'scatter',
              'uid': 'b9fc6ad6-75f9-4e3b-8fe2-294932968a14',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x14',
              'y': array([ 0.00048,  0.01319,  0.00891, ..., -0.01388,  0.00424, -0.0126␣
↪],
                         shape=(1396,)),
```

(continues on next page)

(continued from previous page)

```
             'yaxis': 'y14'},
            {'line': {'color': '#AB63FA', 'dash': 'dash'},
             'name': 'M42 theory',
             'type': 'scatter',
             'uid': '022d1757-c2bc-478e-9098-3ece2910e067',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x14',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y14'},
            {'line': {'color': '#FFA15A', 'dash': 'solid'},
             'name': 'M43 ',
             'type': 'scatter',
             'uid': '8b2bf1e8-e50c-4019-9d80-57692aa6e0ed',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x15',
             'y': array([-0.95574, -0.96899, -0.96191, ..., -0.8796 , -0.87907, -0.
↪87849],
                        shape=(1396,)),
             'yaxis': 'y15'},
            {'line': {'color': '#FFA15A', 'dash': 'dash'},
             'name': 'M43 theory',
             'type': 'scatter',
             'uid': 'c77fc9b2-3e23-4f01-bd5f-e4267a583c03',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x15',
             'y': array([-0.38749245, -0.44971578, -0.50863958, ..., -0.93935042, -0.
↪93913858,
                         -0.93892669], shape=(1396,)),
             'yaxis': 'y15'},
            {'line': {'color': '#19D3F3', 'dash': 'solid'},
             'name': 'M44 ',
             'type': 'scatter',
             'uid': 'c0a08d64-bc2d-453a-be24-c37197da9ed5',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x16',
             'y': array([0.25821, 0.24432, 0.24986, ..., 0.16743, 0.15105, 0.16637],
                        shape=(1396,)),
             'yaxis': 'y16'},
            {'line': {'color': '#19D3F3', 'dash': 'dash'},
             'name': 'M44 theory',
             'type': 'scatter',
             'uid': 'a41f2100-c5bd-4432-b264-070d349e042d',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
```

(continues on next page)

(continued from previous page)

```
                        shape=(1396,)),
             'xaxis': 'x16',
             'y': array([-0.78177026, -0.75168899, -0.71871123, ...,  0.20163327,  0.
↪20156999,
                         0.20150625], shape=(1396,)),
             'yaxis': 'y16'}],
    'layout': {'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis10': {'anchor': 'y10', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis11': {'anchor': 'y11', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis12': {'anchor': 'y12', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis13': {'anchor': 'y13', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis14': {'anchor': 'y14', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis15': {'anchor': 'y15', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis16': {'anchor': 'y16', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis2': {'anchor': 'y2', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis3': {'anchor': 'y3', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis4': {'anchor': 'y4', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis5': {'anchor': 'y5', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis6': {'anchor': 'y6', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis7': {'anchor': 'y7', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis8': {'anchor': 'y8', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis9': {'anchor': 'y9', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'yaxis': {'anchor': 'x', 'domain': [0.80625, 1.0]},
               'yaxis10': {'anchor': 'x10', 'domain': [0.26875, 0.4625]},
               'yaxis11': {'anchor': 'x11', 'domain': [0.26875, 0.4625]},
               'yaxis12': {'anchor': 'x12', 'domain': [0.26875, 0.4625]},
               'yaxis13': {'anchor': 'x13', 'domain': [0.0, 0.19375]},
               'yaxis14': {'anchor': 'x14', 'domain': [0.0, 0.19375]},
               'yaxis15': {'anchor': 'x15', 'domain': [0.0, 0.19375]},
               'yaxis16': {'anchor': 'x16', 'domain': [0.0, 0.19375]},
               'yaxis2': {'anchor': 'x2', 'domain': [0.80625, 1.0]},
               'yaxis3': {'anchor': 'x3', 'domain': [0.80625, 1.0]},
               'yaxis4': {'anchor': 'x4', 'domain': [0.80625, 1.0]},
               'yaxis5': {'anchor': 'x5', 'domain': [0.5375, 0.73125]},
               'yaxis6': {'anchor': 'x6', 'domain': [0.5375, 0.73125]},
               'yaxis7': {'anchor': 'x7', 'domain': [0.5375, 0.73125]},
               'yaxis8': {'anchor': 'x8', 'domain': [0.5375, 0.73125]},
               'yaxis9': {'anchor': 'x9', 'domain': [0.26875, 0.4625]}}
})
```

We can also plot the residual between measurement and model.

```
model.plot_residual()
```

```
FigureWidget({
    'data': [{'line': {'color': '#636EFA', 'dash': 'solid'},
              'name': 'M11 ',
              'type': 'scatter',
              'uid': 'fd12ff6e-f163-4555-9522-27bd647147bf',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
```

(continues on next page)

(continued from previous page)

```
                        shape=(1396,)),
             'xaxis': 'x',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y'},
            {'line': {'color': '#EF553B', 'dash': 'solid'},
             'name': 'M12 ',
             'type': 'scatter',
             'uid': 'd8c84f28-6d72-4913-bbe4-c82adba993d4',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x2',
             'y': array([0.58002385, 0.59119998, 0.58045745, ..., 0.16502466, 0.
↪16587253,
                         0.16486215], shape=(1396,)),
             'yaxis': 'y2'},
            {'line': {'color': '#00CC96', 'dash': 'solid'},
             'name': 'M13 ',
             'type': 'scatter',
             'uid': 'a48fec2b-8868-4f2a-9386-2ccad42da49d',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x3',
             'y': array([ 0.01241,  0.01151,  0.00694, ..., -0.00887,  0.00646, -0.0117␣
↪],
                        shape=(1396,)),
             'yaxis': 'y3'},
            {'line': {'color': '#AB63FA', 'dash': 'solid'},
             'name': 'M14 ',
             'type': 'scatter',
             'uid': '512e7c36-3560-450a-997a-d465f804a94e',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x4',
             'y': array([ 0.00614,  0.00901,  0.00475, ..., -0.00626,  0.0061 , -0.0055␣
↪],
                        shape=(1396,)),
             'yaxis': 'y4'},
            {'line': {'color': '#FFA15A', 'dash': 'solid'},
             'name': 'M21 ',
             'type': 'scatter',
             'uid': '5bf1f2c5-5b29-476e-acf9-916dc8b6ef95',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x5',
             'y': array([0.58460385, 0.57994998, 0.58167745, ..., 0.16281466, 0.
↪16307253,
                         0.16240215], shape=(1396,)),
             'yaxis': 'y5'},
```

(continues on next page)

(continued from previous page)

```
            {'line': {'color': '#19D3F3', 'dash': 'solid'},
             'name': 'M22 ',
             'type': 'scatter',
             'uid': 'a98ea0d3-325a-4958-83b6-63699c97755c',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x6',
             'y': array([0.00574, 0.00525, 0.00363, ..., 0.00277, 0.00339, 0.00394],
                        shape=(1396,)),
             'yaxis': 'y6'},
            {'line': {'color': '#FF6692', 'dash': 'solid'},
             'name': 'M23 ',
             'type': 'scatter',
             'uid': 'e67c9a89-806d-4d73-ac91-3111827b4b57',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x7',
             'y': array([ 0.00364,  0.00427,  0.00207, ...,  0.00457, -0.00234,  0.
↪00653],
                        shape=(1396,)),
             'yaxis': 'y7'},
            {'line': {'color': '#B6E880', 'dash': 'solid'},
             'name': 'M24 ',
             'type': 'scatter',
             'uid': 'e8c36b08-2182-4878-b5e2-7763b7a1adbc',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x8',
             'y': array([-0.01084,  0.00325, -0.01123, ...,  0.00263, -0.00227,  0.
↪00311],
                        shape=(1396,)),
             'yaxis': 'y8'},
            {'line': {'color': '#FF97FF', 'dash': 'solid'},
             'name': 'M31 ',
             'type': 'scatter',
             'uid': 'b4eaed53-2b95-48e0-80ec-95e92fa1bb5e',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x9',
             'y': array([ 0.00278,  0.00634,  0.00219, ..., -0.00055,  0.00105, -0.
↪00037],
                        shape=(1396,)),
             'yaxis': 'y9'},
            {'line': {'color': '#FECB52', 'dash': 'solid'},
             'name': 'M32 ',
             'type': 'scatter',
             'uid': 'fbea932f-8b59-4379-b91e-a47f9299acc5',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
```

(continues on next page)

(continued from previous page)

```
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x10',
              'y': array([ 0.00212,  0.00045, -0.0003 , ...,  0.0018 ,  0.00042, -0.
↪00037],
                          shape=(1396,)),
              'yaxis': 'y10'},
             {'line': {'color': '#636EFA', 'dash': 'solid'},
              'name': 'M33 ',
              'type': 'scatter',
              'uid': '5da9136c-6cfa-4fc0-a539-46be3d59df51',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x11',
              'y': array([-1.05460026, -1.01894899, -0.99151123, ...,  0.03927327,  0.
↪03885999,
                            0.03895625], shape=(1396,)),
              'yaxis': 'y11'},
             {'line': {'color': '#EF553B', 'dash': 'solid'},
              'name': 'M34 ',
              'type': 'scatter',
              'uid': 'e089d08e-4916-4785-b6cf-8c6dc8574bec',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x12',
              'y': array([-0.56805755, -0.51478422, -0.44957042, ...,  0.06223042,  0.
↪06284858,
                            0.06231669], shape=(1396,)),
              'yaxis': 'y12'},
             {'line': {'color': '#00CC96', 'dash': 'solid'},
              'name': 'M41 ',
              'type': 'scatter',
              'uid': 'de4d7396-0465-47b6-bbd3-a62e787eb5fc',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x13',
              'y': array([-0.00986, -0.00917, -0.00712, ...,  0.00585, -0.00882,  0.
↪00692],
                          shape=(1396,)),
              'yaxis': 'y13'},
             {'line': {'color': '#AB63FA', 'dash': 'solid'},
              'name': 'M42 ',
              'type': 'scatter',
              'uid': '11277148-8cde-4a60-a0ea-75ab4bea8161',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x14',
              'y': array([-0.00048, -0.01319, -0.00891, ...,  0.01388, -0.00424,  0.0126␣
```

(continues on next page)

(continued from previous page)

```
↪],
                          shape=(1396,)),
              'yaxis': 'y14'},
             {'line': {'color': '#FFA15A', 'dash': 'solid'},
              'name': 'M43 ',
              'type': 'scatter',
              'uid': 'ffdb9513-e4d6-4056-b248-ea23dfb4acdc',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x15',
              'y': array([ 0.56824755,  0.51927422,  0.45327042, ..., -0.05975042, -0.
↪06006858,
                           -0.06043669], shape=(1396,)),
              'yaxis': 'y15'},
             {'line': {'color': '#19D3F3', 'dash': 'solid'},
              'name': 'M44 ',
              'type': 'scatter',
              'uid': '8b26dbd8-0b0d-4396-a67f-1a5e3fb4f152',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                          shape=(1396,)),
              'xaxis': 'x16',
              'y': array([-1.03998026, -0.99600899, -0.96857123, ...,  0.03420327,  0.
↪05051999,
                           0.03513625], shape=(1396,)),
              'yaxis': 'y16'}],
    'layout': {'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis10': {'anchor': 'y10', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis11': {'anchor': 'y11', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis12': {'anchor': 'y12', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis13': {'anchor': 'y13', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis14': {'anchor': 'y14', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis15': {'anchor': 'y15', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis16': {'anchor': 'y16', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis2': {'anchor': 'y2', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis3': {'anchor': 'y3', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis4': {'anchor': 'y4', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis5': {'anchor': 'y5', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis6': {'anchor': 'y6', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis7': {'anchor': 'y7', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis8': {'anchor': 'y8', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis9': {'anchor': 'y9', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'yaxis': {'anchor': 'x', 'domain': [0.80625, 1.0]},
               'yaxis10': {'anchor': 'x10', 'domain': [0.26875, 0.4625]},
               'yaxis11': {'anchor': 'x11', 'domain': [0.26875, 0.4625]},
               'yaxis12': {'anchor': 'x12', 'domain': [0.26875, 0.4625]},
               'yaxis13': {'anchor': 'x13', 'domain': [0.0, 0.19375]},
               'yaxis14': {'anchor': 'x14', 'domain': [0.0, 0.19375]},
               'yaxis15': {'anchor': 'x15', 'domain': [0.0, 0.19375]},
               'yaxis16': {'anchor': 'x16', 'domain': [0.0, 0.19375]},
```

(continues on next page)

(continued from previous page)

```
               'yaxis2': {'anchor': 'x2', 'domain': [0.80625, 1.0]},
               'yaxis3': {'anchor': 'x3', 'domain': [0.80625, 1.0]},
               'yaxis4': {'anchor': 'x4', 'domain': [0.80625, 1.0]},
               'yaxis5': {'anchor': 'x5', 'domain': [0.5375, 0.73125]},
               'yaxis6': {'anchor': 'x6', 'domain': [0.5375, 0.73125]},
               'yaxis7': {'anchor': 'x7', 'domain': [0.5375, 0.73125]},
               'yaxis8': {'anchor': 'x8', 'domain': [0.5375, 0.73125]},
               'yaxis9': {'anchor': 'x9', 'domain': [0.26875, 0.4625]}}
})
```

Now we execute a fit and plot the model afterwards.

```
fit_stats = model.fit()
model.plot(full_scale=False)
```

```
FigureWidget({
    'data': [{'line': {'color': '#636EFA', 'dash': 'solid'},
              'name': 'M11 ',
              'type': 'scatter',
              'uid': 'd110c035-580b-45c8-80d0-65c8ff98a314',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x',
              'y': array([1., 1., 1., ..., 1., 1., 1.], shape=(1396,)),
              'yaxis': 'y'},
             {'line': {'color': '#636EFA', 'dash': 'dash'},
              'name': 'M11 theory',
              'type': 'scatter',
              'uid': '75b834bb-c2e5-4698-b38d-0de65112f9c0',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x',
              'y': array([1., 1., 1., ..., 1., 1., 1.], shape=(1396,)),
              'yaxis': 'y'},
             {'line': {'color': '#EF553B', 'dash': 'solid'},
              'name': 'M12 ',
              'type': 'scatter',
              'uid': '27d0524e-dd4f-4d30-a03b-c288953232a7',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x2',
              'y': array([-0.09147, -0.10879, -0.10639, ..., -0.44245, -0.44406, -0.
↪44381],
                         shape=(1396,)),
              'yaxis': 'y2'},
             {'line': {'color': '#EF553B', 'dash': 'dash'},
              'name': 'M12 theory',
              'type': 'scatter',
              'uid': '4c38be69-022a-4d4d-a2eb-b03ed841c39c',
```

(continues on next page)

(continued from previous page)

```
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x2',
             'y': array([-0.09081776, -0.09659845, -0.10218436, ..., -0.44726608, -0.
↪44776779,
                         -0.44826863], shape=(1396,)),
             'yaxis': 'y2'},
            {'line': {'color': '#00CC96', 'dash': 'solid'},
             'name': 'M13 ',
             'type': 'scatter',
             'uid': '6b59262c-eed6-4ee5-b132-1ab2af18d746',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x3',
             'y': array([-0.01241, -0.01151, -0.00694, ...,  0.00887, -0.00646,  0.0117␣
↪],
                        shape=(1396,)),
             'yaxis': 'y3'},
            {'line': {'color': '#00CC96', 'dash': 'dash'},
             'name': 'M13 theory',
             'type': 'scatter',
             'uid': 'd4bfe222-1307-4493-81c8-6e3c40d3173a',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x3',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y3'},
            {'line': {'color': '#AB63FA', 'dash': 'solid'},
             'name': 'M14 ',
             'type': 'scatter',
             'uid': '46f3ad53-a566-47fb-a081-144349b9237e',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x4',
             'y': array([-0.00614, -0.00901, -0.00475, ...,  0.00626, -0.0061 ,  0.0055␣
↪],
                        shape=(1396,)),
             'yaxis': 'y4'},
            {'line': {'color': '#AB63FA', 'dash': 'dash'},
             'name': 'M14 theory',
             'type': 'scatter',
             'uid': 'afa5357c-4e8e-4067-90a0-c29ab2ac6d5c',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x4',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y4'},
```

(continues on next page)

(continued from previous page)

```
            {'line': {'color': '#FFA15A', 'dash': 'solid'},
             'name': 'M21 ',
             'type': 'scatter',
             'uid': '10af9f18-d09c-4823-95a8-42eb92d3e508',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x5',
             'y': array([-0.09605, -0.09754, -0.10761, ..., -0.44024, -0.44126, -0.
↪44135],
                        shape=(1396,)),
             'yaxis': 'y5'},
            {'line': {'color': '#FFA15A', 'dash': 'dash'},
             'name': 'M21 theory',
             'type': 'scatter',
             'uid': '6ed2ba64-8ae5-4070-af7a-7d2dffd4771e',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x5',
             'y': array([-0.09081776, -0.09659845, -0.10218436, ..., -0.44726608, -0.
↪44776779,
                         -0.44826863], shape=(1396,)),
             'yaxis': 'y5'},
            {'line': {'color': '#19D3F3', 'dash': 'solid'},
             'name': 'M22 ',
             'type': 'scatter',
             'uid': '41a398cd-92c7-4e29-aedb-8d2132e318d5',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x6',
             'y': array([0.99426, 0.99475, 0.99637, ..., 0.99723, 0.99661, 0.99606],
                        shape=(1396,)),
             'yaxis': 'y6'},
            {'line': {'color': '#19D3F3', 'dash': 'dash'},
             'name': 'M22 theory',
             'type': 'scatter',
             'uid': 'f781e4c7-6d0d-47d2-a10a-c639a29617a1',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x6',
             'y': array([1., 1., 1., ..., 1., 1., 1.], shape=(1396,)),
             'yaxis': 'y6'},
            {'line': {'color': '#FF6692', 'dash': 'solid'},
             'name': 'M23 ',
             'type': 'scatter',
             'uid': 'f3879508-c1f6-4dd7-85ad-4871988a2a83',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
```

(continues on next page)

(continued from previous page)

```
             'xaxis': 'x7',
             'y': array([-0.00364, -0.00427, -0.00207, ..., -0.00457,  0.00234, -0.
↪00653],
                        shape=(1396,)),
             'yaxis': 'y7'},
            {'line': {'color': '#FF6692', 'dash': 'dash'},
             'name': 'M23 theory',
             'type': 'scatter',
             'uid': '2986ddac-58e6-43fb-8e30-f51de48d86c1',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x7',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y7'},
            {'line': {'color': '#B6E880', 'dash': 'solid'},
             'name': 'M24 ',
             'type': 'scatter',
             'uid': 'b2786919-c897-4030-b7db-e9b89c87b91d',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x8',
             'y': array([ 0.01084, -0.00325,  0.01123, ..., -0.00263,  0.00227, -0.
↪00311],
                        shape=(1396,)),
             'yaxis': 'y8'},
            {'line': {'color': '#B6E880', 'dash': 'dash'},
             'name': 'M24 theory',
             'type': 'scatter',
             'uid': '035b7244-bdd1-4cf0-8b82-190dff9cae08',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x8',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y8'},
            {'line': {'color': '#FF97FF', 'dash': 'solid'},
             'name': 'M31 ',
             'type': 'scatter',
             'uid': 'ec71a36a-3253-4857-b86b-e77a93a1fbab',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x9',
             'y': array([-0.00278, -0.00634, -0.00219, ...,  0.00055, -0.00105,  0.
↪00037],
                        shape=(1396,)),
             'yaxis': 'y9'},
            {'line': {'color': '#FF97FF', 'dash': 'dash'},
             'name': 'M31 theory',
             'type': 'scatter',
```

(continues on next page)

(continued from previous page)

```
             'uid': '0539583f-de9d-48a7-98a7-e948756f2ab1',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x9',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y9'},
            {'line': {'color': '#FECB52', 'dash': 'solid'},
             'name': 'M32 ',
             'type': 'scatter',
             'uid': '42ae6633-4f6c-4fb5-8c5d-cc9f7dda24a2',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x10',
             'y': array([-0.00212, -0.00045,  0.0003 , ..., -0.0018 , -0.00042,  0.
↪00037],
                        shape=(1396,)),
             'yaxis': 'y10'},
            {'line': {'color': '#FECB52', 'dash': 'dash'},
             'name': 'M32 theory',
             'type': 'scatter',
             'uid': '43f72158-f4be-4271-b00a-48283f0d46c0',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x10',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y10'},
            {'line': {'color': '#636EFA', 'dash': 'solid'},
             'name': 'M33 ',
             'type': 'scatter',
             'uid': '1e0f3b54-35f0-4f41-ac19-93c4a64ec481',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x11',
             'y': array([0.27283, 0.26726, 0.2728 , ..., 0.16236, 0.16271, 0.16255],
                        shape=(1396,)),
             'yaxis': 'y11'},
            {'line': {'color': '#636EFA', 'dash': 'dash'},
             'name': 'M33 theory',
             'type': 'scatter',
             'uid': 'b85af292-1d89-4b7e-ac3b-f7b27f6c8830',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x11',
             'y': array([0.28633493, 0.28495866, 0.28318747, ..., 0.15873433, 0.1585769␣
↪,
                         0.15841941], shape=(1396,)),
             'yaxis': 'y11'},
```

(continues on next page)

(continued from previous page)

```
            {'line': {'color': '#EF553B', 'dash': 'solid'},
             'name': 'M34 ',
             'type': 'scatter',
             'uid': '0219b83c-047b-4428-b244-822caa196b23',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x12',
             'y': array([0.95555, 0.9645 , 0.95821, ..., 0.87712, 0.87629, 0.87661],
                        shape=(1396,)),
             'yaxis': 'y12'},
            {'line': {'color': '#EF553B', 'dash': 'dash'},
             'name': 'M34 theory',
             'type': 'scatter',
             'uid': '73bf6797-7710-48e9-b686-b4421d062932',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x12',
             'y': array([0.95381573, 0.95365995, 0.95360538, ..., 0.88020251, 0.
↪87997578,
                         0.87974912], shape=(1396,)),
             'yaxis': 'y12'},
            {'line': {'color': '#00CC96', 'dash': 'solid'},
             'name': 'M41 ',
             'type': 'scatter',
             'uid': 'acadfcc0-4fa0-4cab-a1d5-8c0f4a5b1bba',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x13',
             'y': array([ 0.00986,  0.00917,  0.00712, ..., -0.00585,  0.00882, -0.
↪00692],
                        shape=(1396,)),
             'yaxis': 'y13'},
            {'line': {'color': '#00CC96', 'dash': 'dash'},
             'name': 'M41 theory',
             'type': 'scatter',
             'uid': 'e3e93357-a358-4943-8353-b3bf78bc773c',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x13',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y13'},
            {'line': {'color': '#AB63FA', 'dash': 'solid'},
             'name': 'M42 ',
             'type': 'scatter',
             'uid': '64e5f808-af38-42fc-9baf-86bc66ad1b77',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
```

(continues on next page)

(continued from previous page)

```
             'xaxis': 'x14',
             'y': array([ 0.00048,  0.01319,  0.00891, ..., -0.01388,  0.00424, -0.0126␣
↪],
                        shape=(1396,)),
             'yaxis': 'y14'},
            {'line': {'color': '#AB63FA', 'dash': 'dash'},
             'name': 'M42 theory',
             'type': 'scatter',
             'uid': '7739a6c1-b356-47bb-a4ed-f562ea47ae63',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x14',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y14'},
            {'line': {'color': '#FFA15A', 'dash': 'solid'},
             'name': 'M43 ',
             'type': 'scatter',
             'uid': '2bb29c39-523e-43f9-bfa3-14e0982393c8',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x15',
             'y': array([-0.95574, -0.96899, -0.96191, ..., -0.8796 , -0.87907, -0.
↪87849],
                        shape=(1396,)),
             'yaxis': 'y15'},
            {'line': {'color': '#FFA15A', 'dash': 'dash'},
             'name': 'M43 theory',
             'type': 'scatter',
             'uid': '13e47436-9400-444b-a5f3-1b654a9e0996',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x15',
             'y': array([-0.95381573, -0.95365995, -0.95360538, ..., -0.88020251, -0.
↪87997578,
                         -0.87974912], shape=(1396,)),
             'yaxis': 'y15'},
            {'line': {'color': '#19D3F3', 'dash': 'solid'},
             'name': 'M44 ',
             'type': 'scatter',
             'uid': '78485006-b438-48e6-9c35-bbf5bf62b734',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x16',
             'y': array([0.25821, 0.24432, 0.24986, ..., 0.16743, 0.15105, 0.16637],
                        shape=(1396,)),
             'yaxis': 'y16'},
            {'line': {'color': '#19D3F3', 'dash': 'dash'},
             'name': 'M44 theory',
```

(continues on next page)

(continued from previous page)

```
              'type': 'scatter',
              'uid': 'bb9b8927-dff9-422e-939a-6caf41318af8',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x16',
              'y': array([0.28633493, 0.28495866, 0.28318747, ..., 0.15873433, 0.1585769␣
↪',
                          0.15841941], shape=(1396,)),
              'yaxis': 'y16'}],
    'layout': {'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis10': {'anchor': 'y10', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis11': {'anchor': 'y11', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis12': {'anchor': 'y12', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis13': {'anchor': 'y13', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis14': {'anchor': 'y14', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis15': {'anchor': 'y15', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis16': {'anchor': 'y16', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis2': {'anchor': 'y2', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis3': {'anchor': 'y3', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis4': {'anchor': 'y4', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis5': {'anchor': 'y5', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis6': {'anchor': 'y6', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis7': {'anchor': 'y7', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis8': {'anchor': 'y8', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis9': {'anchor': 'y9', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'yaxis': {'anchor': 'x', 'domain': [0.80625, 1.0]},
               'yaxis10': {'anchor': 'x10', 'domain': [0.26875, 0.4625]},
               'yaxis11': {'anchor': 'x11', 'domain': [0.26875, 0.4625]},
               'yaxis12': {'anchor': 'x12', 'domain': [0.26875, 0.4625]},
               'yaxis13': {'anchor': 'x13', 'domain': [0.0, 0.19375]},
               'yaxis14': {'anchor': 'x14', 'domain': [0.0, 0.19375]},
               'yaxis15': {'anchor': 'x15', 'domain': [0.0, 0.19375]},
               'yaxis16': {'anchor': 'x16', 'domain': [0.0, 0.19375]},
               'yaxis2': {'anchor': 'x2', 'domain': [0.80625, 1.0]},
               'yaxis3': {'anchor': 'x3', 'domain': [0.80625, 1.0]},
               'yaxis4': {'anchor': 'x4', 'domain': [0.80625, 1.0]},
               'yaxis5': {'anchor': 'x5', 'domain': [0.5375, 0.73125]},
               'yaxis6': {'anchor': 'x6', 'domain': [0.5375, 0.73125]},
               'yaxis7': {'anchor': 'x7', 'domain': [0.5375, 0.73125]},
               'yaxis8': {'anchor': 'x8', 'domain': [0.5375, 0.73125]},
               'yaxis9': {'anchor': 'x9', 'domain': [0.26875, 0.4625]}}
})
```

For comparison we plot the residual again to have a figure of merit for the fit quality

```
model.plot_residual()
```

```
FigureWidget({
    'data': [{'line': {'color': '#636EFA', 'dash': 'solid'},
              'name': 'M11 ',
```

(continues on next page)

(continued from previous page)

```
             'type': 'scatter',
             'uid': 'c6d8a182-3600-4855-bf19-c3d4dccaf380',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x',
             'y': array([0., 0., 0., ..., 0., 0., 0.], shape=(1396,)),
             'yaxis': 'y'},
            {'line': {'color': '#EF553B', 'dash': 'solid'},
             'name': 'M12 ',
             'type': 'scatter',
             'uid': '8a1e8f48-2d01-41c1-a49f-8709f68059fb',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x2',
             'y': array([ 0.00065224,  0.01219155,  0.00420564, ..., -0.00481608, -0.
↪00370779,
                         -0.00445863], shape=(1396,)),
             'yaxis': 'y2'},
            {'line': {'color': '#00CC96', 'dash': 'solid'},
             'name': 'M13 ',
             'type': 'scatter',
             'uid': 'dfe57b5d-8d37-438b-99b1-806894e527c4',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x3',
             'y': array([ 0.01241,  0.01151,  0.00694, ..., -0.00887,  0.00646, -0.0117␣
↪],
                        shape=(1396,)),
             'yaxis': 'y3'},
            {'line': {'color': '#AB63FA', 'dash': 'solid'},
             'name': 'M14 ',
             'type': 'scatter',
             'uid': 'b106ae68-4f9a-4cfc-8dc3-257bdd905ec7',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x4',
             'y': array([ 0.00614,  0.00901,  0.00475, ..., -0.00626,  0.0061 , -0.0055␣
↪],
                        shape=(1396,)),
             'yaxis': 'y4'},
            {'line': {'color': '#FFA15A', 'dash': 'solid'},
             'name': 'M21 ',
             'type': 'scatter',
             'uid': '692f3b54-6765-4f29-97e4-ade23046c3a1',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x5',
```

(continues on next page)

(continued from previous page)

```
              'y': array([ 0.00523224,  0.00094155,  0.00542564, ..., -0.00702608, -0.
↪00650779,
                          -0.00691863], shape=(1396,)),
              'yaxis': 'y5'},
             {'line': {'color': '#19D3F3', 'dash': 'solid'},
              'name': 'M22 ',
              'type': 'scatter',
              'uid': '3f7cfc7f-b612-480c-9c86-aaefb631bddf',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x6',
              'y': array([0.00574, 0.00525, 0.00363, ..., 0.00277, 0.00339, 0.00394],
                         shape=(1396,)),
              'yaxis': 'y6'},
             {'line': {'color': '#FF6692', 'dash': 'solid'},
              'name': 'M23 ',
              'type': 'scatter',
              'uid': 'a6fce54a-f887-4707-a563-87ae844bb053',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x7',
              'y': array([ 0.00364,  0.00427,  0.00207, ...,  0.00457, -0.00234,  0.
↪00653],
                         shape=(1396,)),
              'yaxis': 'y7'},
             {'line': {'color': '#B6E880', 'dash': 'solid'},
              'name': 'M24 ',
              'type': 'scatter',
              'uid': 'b3c9856b-6b51-429e-bccd-93de84c8b228',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x8',
              'y': array([-0.01084,  0.00325, -0.01123, ...,  0.00263, -0.00227,  0.
↪00311],
                         shape=(1396,)),
              'yaxis': 'y8'},
             {'line': {'color': '#FF97FF', 'dash': 'solid'},
              'name': 'M31 ',
              'type': 'scatter',
              'uid': '6d4bda14-2395-46f5-a637-d806b3f6d8bd',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x9',
              'y': array([ 0.00278,  0.00634,  0.00219, ..., -0.00055,  0.00105, -0.
↪00037],
                         shape=(1396,)),
              'yaxis': 'y9'},
             {'line': {'color': '#FECB52', 'dash': 'solid'},
```

(continues on next page)

(continued from previous page)

```
             'name': 'M32 ',
             'type': 'scatter',
             'uid': '85e13efb-9a62-42b7-9c07-ec42e20b02d5',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x10',
             'y': array([ 0.00212,  0.00045, -0.0003 , ...,  0.0018 ,  0.00042, -0.
↪00037],
                        shape=(1396,)),
             'yaxis': 'y10'},
            {'line': {'color': '#636EFA', 'dash': 'solid'},
             'name': 'M33 ',
             'type': 'scatter',
             'uid': 'f82af53b-9e1e-4be1-9428-0932acafa191',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x11',
             'y': array([ 0.01350493,  0.01769866,  0.01038747, ..., -0.00362567, -0.
↪0041331 ,
                         -0.00413059], shape=(1396,)),
             'yaxis': 'y11'},
            {'line': {'color': '#EF553B', 'dash': 'solid'},
             'name': 'M34 ',
             'type': 'scatter',
             'uid': 'a85031dd-acc7-4298-b943-379a8ca794a2',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x12',
             'y': array([-0.00173427, -0.01084005, -0.00460462, ...,  0.00308251,  0.
↪00368578,
                          0.00313912], shape=(1396,)),
             'yaxis': 'y12'},
            {'line': {'color': '#00CC96', 'dash': 'solid'},
             'name': 'M41 ',
             'type': 'scatter',
             'uid': '5104e1e2-a3a3-41f1-9177-62f20cb272ea',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                        shape=(1396,)),
             'xaxis': 'x13',
             'y': array([-0.00986, -0.00917, -0.00712, ...,  0.00585, -0.00882,  0.
↪00692],
                        shape=(1396,)),
             'yaxis': 'y13'},
            {'line': {'color': '#AB63FA', 'dash': 'solid'},
             'name': 'M42 ',
             'type': 'scatter',
             'uid': '54b423e9-6008-4a01-a836-645c8e7d3497',
             'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
```

(continues on next page)

(continued from previous page)

```
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x14',
              'y': array([-0.00048, -0.01319, -0.00891, ...,  0.01388, -0.00424,  0.0126␣
↪],
                         shape=(1396,)),
              'yaxis': 'y14'},
             {'line': {'color': '#FFA15A', 'dash': 'solid'},
              'name': 'M43 ',
              'type': 'scatter',
              'uid': 'd9d24c94-4760-4f71-8d3c-95d8c61919fc',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x15',
              'y': array([ 0.00192427,  0.01533005,  0.00830462, ..., -0.00060251, -0.
↪00090578,
                          -0.00125912], shape=(1396,)),
              'yaxis': 'y15'},
             {'line': {'color': '#19D3F3', 'dash': 'solid'},
              'name': 'M44 ',
              'type': 'scatter',
              'uid': '7d89e8d4-4ffb-4ec0-96a3-19b729cb7bfa',
              'x': array([210.30366, 210.76102, 211.21834, ..., 818.9224 , 819.33886,␣
↪819.75528],
                         shape=(1396,)),
              'xaxis': 'x16',
              'y': array([ 0.02812493,  0.04063866,  0.03332747, ..., -0.00869567,  0.
↪0075269 ,
                          -0.00795059], shape=(1396,)),
              'yaxis': 'y16'}],
    'layout': {'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis10': {'anchor': 'y10', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis11': {'anchor': 'y11', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis12': {'anchor': 'y12', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis13': {'anchor': 'y13', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis14': {'anchor': 'y14', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis15': {'anchor': 'y15', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis16': {'anchor': 'y16', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis2': {'anchor': 'y2', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis3': {'anchor': 'y3', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis4': {'anchor': 'y4', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis5': {'anchor': 'y5', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'xaxis6': {'anchor': 'y6', 'domain': [0.2625, 0.475], 'matches': 'x'},
               'xaxis7': {'anchor': 'y7', 'domain': [0.525, 0.7375], 'matches': 'x'},
               'xaxis8': {'anchor': 'y8', 'domain': [0.7875, 1.0], 'matches': 'x'},
               'xaxis9': {'anchor': 'y9', 'domain': [0.0, 0.2125], 'matches': 'x'},
               'yaxis': {'anchor': 'x', 'domain': [0.80625, 1.0]},
               'yaxis10': {'anchor': 'x10', 'domain': [0.26875, 0.4625]},
               'yaxis11': {'anchor': 'x11', 'domain': [0.26875, 0.4625]},
               'yaxis12': {'anchor': 'x12', 'domain': [0.26875, 0.4625]},
```

(continues on next page)

(continued from previous page)

```
                'yaxis13': {'anchor': 'x13', 'domain': [0.0, 0.19375]},
                'yaxis14': {'anchor': 'x14', 'domain': [0.0, 0.19375]},
                'yaxis15': {'anchor': 'x15', 'domain': [0.0, 0.19375]},
                'yaxis16': {'anchor': 'x16', 'domain': [0.0, 0.19375]},
                'yaxis2': {'anchor': 'x2', 'domain': [0.80625, 1.0]},
                'yaxis3': {'anchor': 'x3', 'domain': [0.80625, 1.0]},
                'yaxis4': {'anchor': 'x4', 'domain': [0.80625, 1.0]},
                'yaxis5': {'anchor': 'x5', 'domain': [0.5375, 0.73125]},
                'yaxis6': {'anchor': 'x6', 'domain': [0.5375, 0.73125]},
                'yaxis7': {'anchor': 'x7', 'domain': [0.5375, 0.73125]},
                'yaxis8': {'anchor': 'x8', 'domain': [0.5375, 0.73125]},
                'yaxis9': {'anchor': 'x9', 'domain': [0.26875, 0.4625]}}
})
```

We may also print the fit statistics.

```
fit_stats
```

References

Here you can find the latest jupyter notebook and data files of this example.

Total running time of the script: (0 minutes 2.197 seconds)

1.2.5 TiO2/SiO2 Bragg mirror

TiO2/SiO2 Bragg mirror simulation

Authors: O. Castany, M.Müller

This notebook simulates a Bragg mirror composed of alternating layers of TiO_2 and SiO_2, each a quarter-wavelength thick at a central wavelength of 1550 nm. The mirror contains 8.5 periods, with air as the incident medium and glass as the substrate.

```
import elli
import elli.plot as elliplot
import matplotlib.pyplot as plt
import numpy as np

np.set_printoptions(suppress=True, precision=3)
```

Material definition

We define all required materials with constant (non-dispersive) refractive indicies. Air as incidence material and glass as exit material. The Bragg mirror materials are SiO2 and TiO2.

```
air = elli.AIR
glass = elli.ConstantRefractiveIndex(1.5).get_mat()
SiO2 = elli.ConstantRefractiveIndex(1.47).get_mat()
TiO2 = elli.ConstantRefractiveIndex(2.23 + 1j * 5.2e-4).get_mat()
```

Create layers and structure

The SiO2 and TiO2 layers are set to the thickness of an quarterwaveplate of the respective material at 1550 nm.

```
lbda0 = 1550

d_SiO2 = elli.get_qwp_thickness(SiO2, lbda0)
d_TiO2 = elli.get_qwp_thickness(TiO2, lbda0)

print("Thickness of the SiO2 QWP: {} nm".format(d_SiO2))
print("Thickness of the TiO2 QWP: {} nm".format(d_TiO2))

L_SiO2 = elli.Layer(SiO2, d_SiO2)
L_TiO2 = elli.Layer(TiO2, d_TiO2)
```

```
Thickness of the SiO2 QWP: 263.6054421768708 nm
Thickness of the TiO2 QWP: 173.76681614349775 nm
```

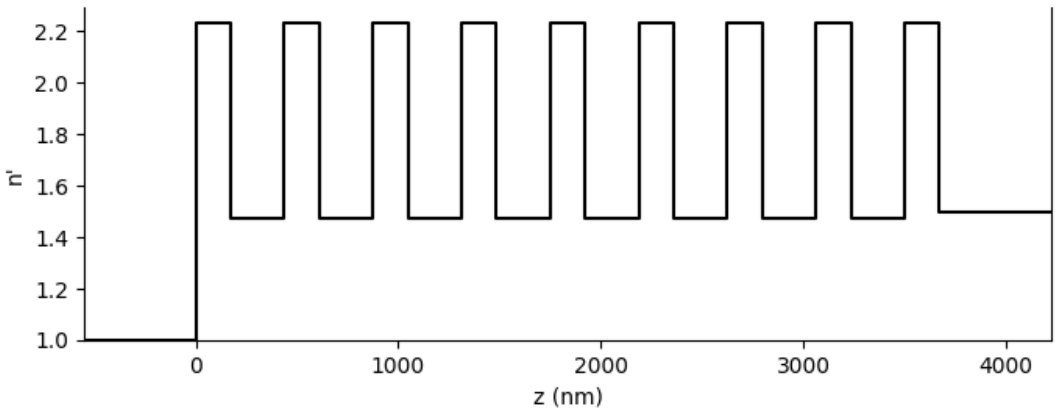
Create layers and structure

We now build the multilayer stack: Alternating layers of TiO2 / SiO2: - 8 full periods + 1 TiO2 layer at the end → “8.5 periods” - Placed between air (front) and glass (back)

```
# Define periodic stack
# Repeated layers: 8.5 periods: Parameters (Substructure, repetitions, number of layers␣
↪before, number of layers after the stack)
layerstack = elli.RepeatedLayers([L_TiO2, L_SiO2], 8, 0, 1)

# Build full structure
s = elli.Structure(air, [layerstack], glass)
```

Structure Graph: Visualization of the refractive index profile in z-direction.

```
elliplot.draw_structure(s)
```




```
<Axes: xlabel='z (nm)', ylabel="n'">
```

Optical calculation

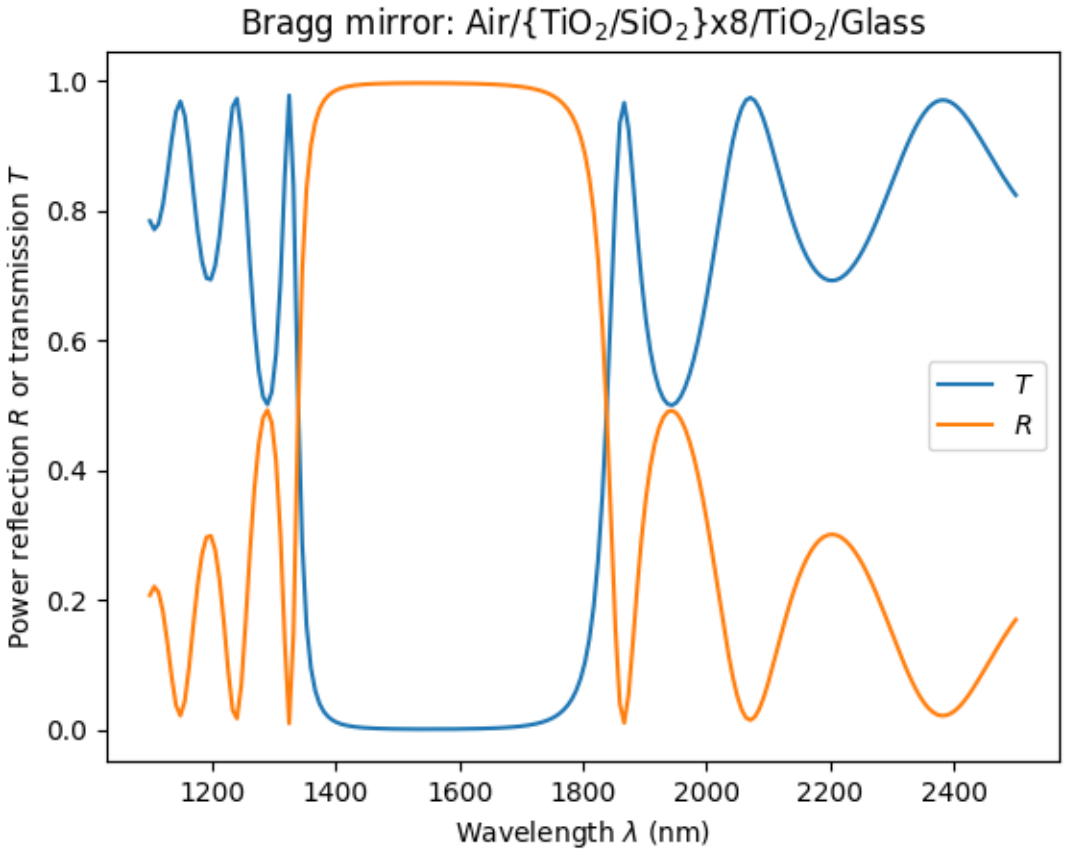
Calculation of the reflection and transmission of the structure at normal incidence (0°).

```
(lbda1, lbda2) = (1100, 2500)
lbda_list = np.linspace(lbda1, lbda2, 200)

data = s.evaluate(lbda_list, 0)

R = data.R
T = data.T
```

Reflection and Transmission data

The plot shows the high reflection in the stopband region around the design wavelength and the interference patterns around it.

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(lbda_list, T, label="$T$")
ax.plot(lbda_list, R, label="$R$")
ax.legend(loc="center right")
ax.set_xlabel(r"Wavelength $\lambda$ (nm)")
ax.set_ylabel(r"Power reflection $R$ or transmission $T$")
ax.set_title(r"Bragg mirror: Air/{TiO$_2$/SiO$_2$}x8/TiO$_2$/Glass")
plt.show()
```



Total running time of the script: (0 minutes 0.248 seconds)

1.2.6 Cholesteric Liquid Crystal Simulation

Example of a cholesteric liquid crystal

Authors: O. Castany, C. Molinaro, M. Müller

This notebook demonstrates how to model and simulate the optical properties of a cholesteric liquid crystal using pyElli. We will build a helical structure and observe its characteristic selective reflection of circularly polarized light.

```
import elli
import elli.plot as elliplot
import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import c, pi
```

Setup materials

We start by defining the materials for our model. The liquid crystal will be sandwiched between two layers of isotropic glass with a constant refractive index of 1.55.

The liquid crystal (LC) itself is uniaxial and will be rotated to have the extraordinary axis orientated in x-direction.

```
glass = elli.IsotropicMaterial(elli.ConstantRefractiveIndex(1.55))
front = back = glass

# Define the ordinary (no) and extraordinary (ne) refractive indices for the LC
(no, ne) = (1.5, 1.7)
Dn = ne - no
n_med = (ne + no) / 2

# Create a uniaxial material with ne oriented along the z-axis by default
LC = elli.UniaxialMaterial(
    elli.ConstantRefractiveIndex(no), elli.ConstantRefractiveIndex(ne)
)

# Rotate the material so the extraordinary axis is along x instead of z
R = elli.rotation_v_theta(elli.E_Y, 90)  # rotation of pi/2 along y
LC.set_rotation(R)  # apply rotation from z to x
```

Building the Helical Structure

The defining feature of a cholesteric liquid crystal is its helical structure. The distance over which the director rotates by 360° is known as the cholesteric pitch (p).

We model this by first creating a single TwistedLayer that represents one half-turn of the helix (180° rotation over a distance of p/2). We then stack this layer multiple times using RepeatedLayers to build the full thickness (7.5p) of the liquid crystal cell.

```
# Cholesteric pitch (nm):
p = 650

# One half turn of a right-handed helix:
TN = elli.TwistedLayer(LC, p / 2, angle=180, div=35)

# Repetition the helix layer
N = 15  # number half pitch repetitions
h = N * p / 2
L = elli.RepeatedLayers([TN], N)
s = elli.Structure(front, [L], back)
```

Setting Calculation Parameters

Finally, we define the wavelength range for our simulation around the central wavelength of the reflection band, which is determined by the average refractive index and the pitch.

```
# Define wavelength range in nm for the calculation
lbda_min, lbda_max = 800, 1200
lbda_B = p * n_med  # Expected center of the reflection band
lbda_list = np.linspace(lbda_min, lbda_max, 100)
```

Analytical calculation for the maximal reflection

For a cholesteric liquid crystal, we can analytically calculate the approximate edges of the reflection band and the maximum reflectance. This serves as a good check for our full simulation.

```
# Theoretical maximum reflectance
R_th = np.tanh(Dn / n_med * pi * h / p) ** 2

# Theoretical wavelength edges of the reflection band
lbda_B1, lbda_B2 = p * no, p * ne
```

Calculation with pyElli

Now we run the simulation using the s.evaluate() method. This calculates the Jones matrix elements for transmission and reflection. They are extracted for linear (s and p), as well as circular polarized light (R and L)

```
# Run the full optical simulation for the defined structure and wavelength list
data = s.evaluate(lbda_list, 0)

# Extract transmission coefficients for linear polarization
T_pp = data.T_pp
T_ps = data.T_ps
T_ss = data.T_ss
T_sp = data.T_sp

# Calculate transmission for unpolarized incident light
T_nn = 0.5 * (T_pp + T_ps + T_sp + T_ss)

# Transmission coefficients for 's' and 'p' polarized light, with
# unpolarized measurement.
T_ns = T_ps + T_ss
T_np = T_pp + T_sp

# Right-circular wave is reflected in the stop-band.
# A right-handed helix reflects right-circular (RR) light.
R_RR = data.Rc_RR
R_LR = data.Rc_LR
T_RR = data.Tc_RR
T_LR = data.Tc_LR

# Left-circular wave is transmitted in the full spectrum.
# T_RL, R_RL, R_LL close to zero, T_LL close to 1.
T_LL = data.Tc_LL
R_LL = data.Rc_LL
```

Plotting

Finally, we plot the results. The plot will show the transmission and reflection spectra for various polarizations.

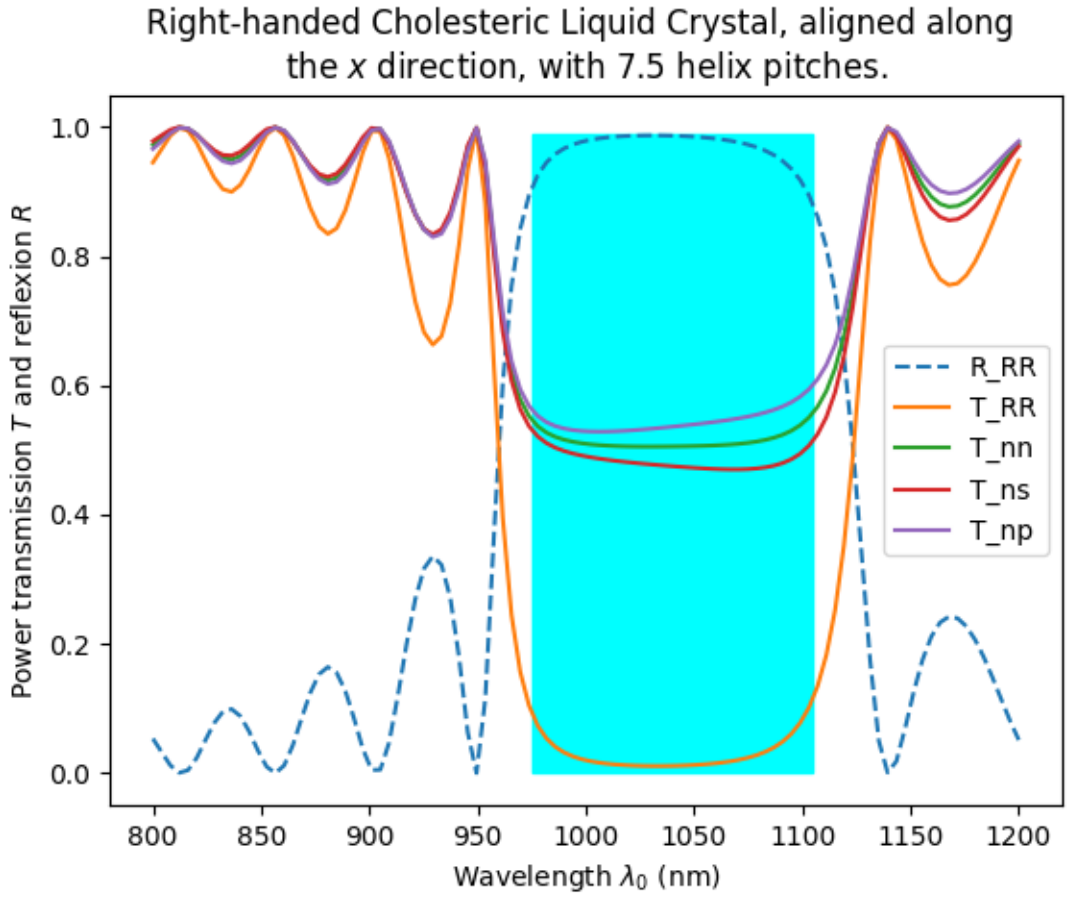
We also draw a shaded rectangle representing the analytically calculated photonic stop-band. Inside this band, we expect to see strong reflection for a specific circular polarization. As our model is a right-handed helix, it should strongly reflect right-circularly polarized light (R_RR).

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

# Draw rectangle for   [p·no, p·ne], and T  [0, R_th]
rectangle = plt.Rectangle((lbda_B1, 0), lbda_B2 - lbda_B1, R_th, color="cyan")
```

(continues on next page)

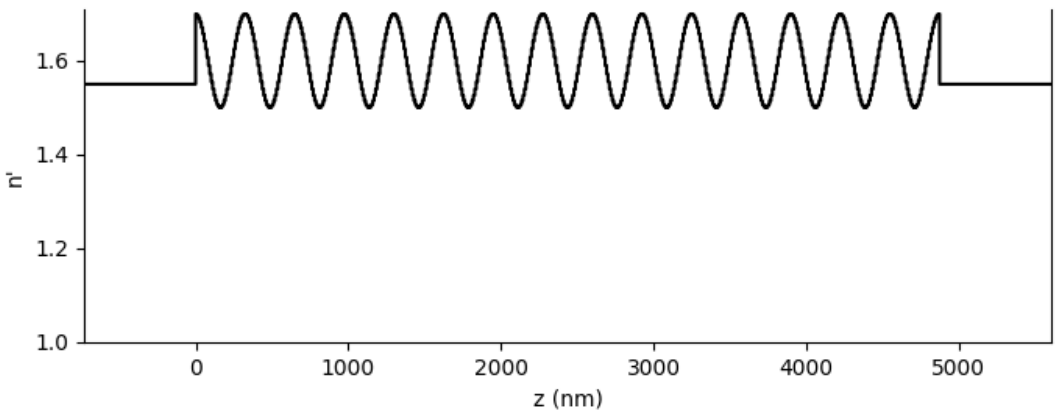
(continued from previous page)

```
ax.add_patch(rectangle)

ax.plot(lbda_list, R_RR, "--", label="R_RR")
ax.plot(lbda_list, T_RR, label="T_RR")
ax.plot(lbda_list, T_nn, label="T_nn")
ax.plot(lbda_list, T_ns, label="T_ns")
ax.plot(lbda_list, T_np, label="T_np")

ax.legend(loc="center right", bbox_to_anchor=(1.00, 0.50))

ax.set_title(
    "Right-handed Cholesteric Liquid Crystal, aligned along \n"
    + "the $x$ direction, with {:.1f} helix pitches.".format(N / 2.0)
)
ax.set_xlabel(r"Wavelength $\lambda_0$ (nm)")
ax.set_ylabel(r"Power transmission $T$ and reflexion $R$")
plt.show()
```



Additionally we can visualize the refractive index n x direction of the structure in dependance of z position.

```
elliplot.draw_structure(s)
```




```
<Axes: xlabel='z (nm)', ylabel="n'">
```

Total running time of the script: (0 minutes 0.384 seconds)

1.2.7 Interface reflection

1.2.8 Reflection on interfaces between two materials

Authors: O. Castany, C. Molinaro, M. Müller

This notebook analyzes the optical behavior at the interface between two isotropic materials with refractive indices n1 and n2.

It computes the reflection and transmission coefficients as functions of the incidence angle, for both s- and p-polarized light, using the analytical Fresnel equations and comparing those to pyElli's output.

Import required libraries

```
import elli
import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import c, pi
```

Structure definition

We define an optical system with two materials:

- Frontside medium (air) with refractive index n = 1.0
- Back medium (glass) with n = 1.5

We also prepare the vacuum wavevector k0 and a range of incidence angles from 0° to 89°.

```
# Define refractive indices for the two media
n1 = 1
n2 = 1.5

# Create isotropic materials using pyElli's ConstantRefractiveIndex model
```

(continues on next page)

(continued from previous page)

```
front = elli.IsotropicMaterial(elli.ConstantRefractiveIndex(n1))
back = elli.IsotropicMaterial(elli.ConstantRefractiveIndex(n2))

# Define the optical structure: just a single interface with no internal layers
s = elli.Structure(front, [], back)

# Set wavelength and wavevector
lbda = 1000
k0 = 2 * pi / lbda

# Define a range of incidence angles from 0° to 89°
Phi_list = np.linspace(0, 89, 90)
```

Analytical calculation

We calculate the Fresnel reflection and transmission coefficients for s- and p-polarized light at an interface.

```
# Convert incidence angles from degrees to radians
Phi_i = np.deg2rad(Phi_list)

# Snell's law (allowing complex values for total internal reflection)
Phi_t = np.arcsin((n1 * np.sin(Phi_i) / n2).astype(complex))

kz1 = n1 * k0 * np.cos(Phi_i)
kz2 = n2 * k0 * np.cos(Phi_t)

# Fresnel coefficients
r_s = (kz1 - kz2) / (kz1 + kz2)
t_s = 1 + r_s
r_p = (kz1 * n2**2 - kz2 * n1**2) / (kz1 * n2**2 + kz2 * n1**2)
t_p = np.cos(Phi_i) * (1 - r_p) / np.cos(Phi_t)

# Reflectance and Transmittance
R_th_ss = abs(r_s) ** 2
R_th_pp = abs(r_p) ** 2
t2_th_ss = abs(t_s) ** 2
t2_th_pp = abs(t_p) ** 2

# The power transmission coefficient (correction for energy conservation) is T = Re(kz2/
↪kz1) × |t|^2
correction = np.real(kz2 / kz1)
T_th_ss = correction * t2_th_ss
T_th_pp = correction * t2_th_pp
```

Calculation with pyElli

Here, we simulate the same interface using pyElli. Each angle is evaluated using `Structure.evaluate()` and then grouped in a `ResultList` object to get angle dependent arrays.

```
data = elli.ResultList([s.evaluate(lbda, Phi_i) for Phi_i in Phi_list])

R_pp = data.R_pp
```

(continues on next page)

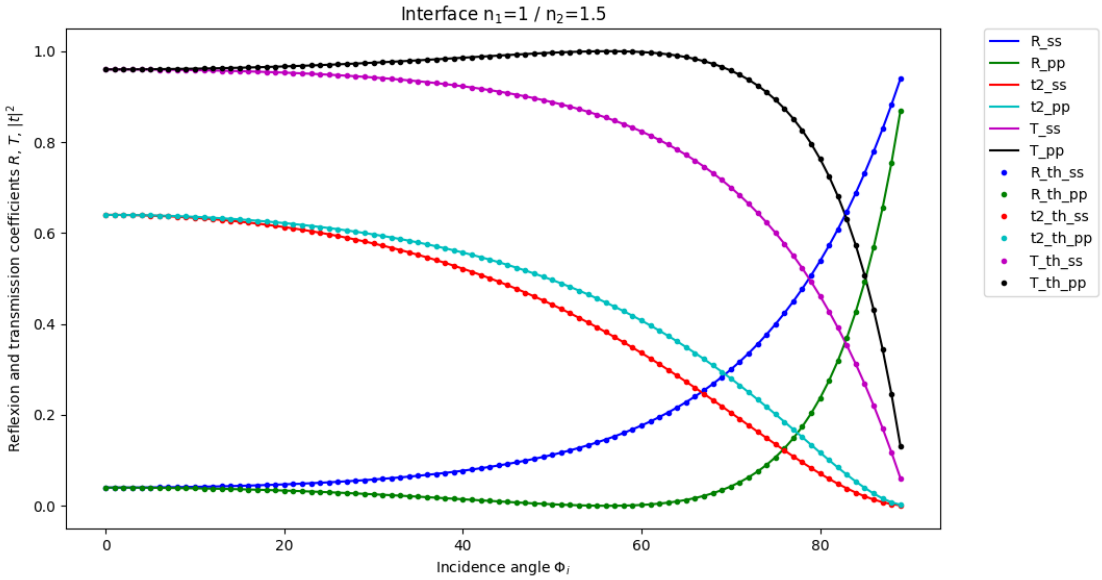
(continued from previous page)

```
R_ss = data.R_ss

T_pp = data.T_pp
T_ss = data.T_ss

t2_pp = np.abs(data.t_pp) ** 2
t2_ss = np.abs(data.t_ss) ** 2
```

Comparing results

- Theoretical results as dotted lines
- PyElli results as solid lines

```
fig = plt.figure(figsize=(12.0, 6.0))
plt.rcParams["axes.prop_cycle"] = plt.cycler("color", "bgrcmk")
ax = fig.add_axes([0.1, 0.1, 0.7, 0.8])

d = np.vstack((R_ss, R_pp, t2_ss, t2_pp, T_ss, T_pp)).T
lines1 = ax.plot(Phi_list, d)
legend1 = ("R_ss", "R_pp", "t2_ss", "t2_pp", "T_ss", "T_pp")

d = np.vstack((R_th_ss, R_th_pp, t2_th_ss, t2_th_pp, T_th_ss, T_th_pp)).T
lines2 = ax.plot(Phi_list, d, ".")
legend2 = ("R_th_ss", "R_th_pp", "t2_th_ss", "t2_th_pp", "T_th_ss", "T_th_pp")

ax.legend(
    lines1 + lines2,
    legend1 + legend2,
    loc="upper left",
    bbox_to_anchor=(1.05, 1),
    borderaxespad=0.0,
)

ax.set_title("Interface n$_1$={:} / n$_2$={:}".format(n1, n2))
ax.set_xlabel(r"Incidence angle $\Phi_i$ ")
ax.set_ylabel(r"Reflexion and transmission coefficients $R$, $T$, $|t|^2$")

plt.show()
```



Total running time of the script: (0 minutes 0.109 seconds)

1.3 API Reference

1.3.1 Dispersion Database

class elli.db.**RII**

Helper class to load tabulated dielectric functions from the https://refractiveindex.info database.

The database object has to be initialized to load the entire catalog:

```
RII = elli.db.RII()
```

After initialization the Class provides a dataframe with all entries at RII.catalog, it follows the naming schema used on the website: Entries are categorized by Shelf, Book and Page.

Shelfs are broad categories (inorganics, organics, glasses, other), and ignored in the following.

Book and Page are used to access entries with this helper: For inorganic and organic materials 'book' uses the sum formula of the compound and 'page' is an identifier of the author of the publication. In the case of glasses, book is the manufacturer and page the name of the glass.

Materials can be searched by the RII.search() method. Specific fields can be searched by providing a column keyword:

```
RII.search("Aspnes", column="author")
```

Dispersions or respective materials can be loaded by calling these methods:

```
gold_material = RII.get_mat("Au", "Johnson")
gold_dispersion = RII.get_dispersion("Au", "Johnson")
```

get_comment(*book*, *page*)

Reads the measurement/calculation information of the selected dispersion.

Parameters

- **book** (*str*) – Name of the Material, named 'Book' on the website and the database. E.g. 'Au'
- **page** (*str*) – Name of the Source, named 'Page' on the website and the database. E.g. 'Johnson'

Returns
Dispersion information.

Return type
str

`get_dispersion`(*book*, *page*)

Load a dispersion from the refractive index database. Selection by material and source identifiers.

Parameters

- **book** (*str*) – Name of the Material, named 'Book' on the website and the database. E.g. 'Au'
- **page** (*str*) – Name of the Source, named 'Page' on the website and the database. E.g. 'Johnson'

Returns
A dispersion object containing the tabulated data.

Return type
Dispersion

`get_mat`(*book*, *page*)

Load a dispersion from the refractive index database and generates an isotropic material. Selection by material and source identifiers.

Parameters

- **book** (*str*) – Name of the Material, named 'Book' on the website and the database. E.g. 'Au'
- **page** (*str*) – Name of the Source, named 'Page' on the website and the database. E.g. 'Johnson'

Returns
A material object build from the tabulated dispersion data.

Return type
IsotropicMaterial

`get_reference`(*book*, *page*)

Reads the reference information from the selected dispersion.

Parameters

- **book** (*str*) – Name of the Material, named 'Book' on the website and the database. E.g. 'Au'
- **page** (*str*) – Name of the Source, named 'Page' on the website and the database. E.g. 'Johnson'

Returns
Reference information.

Return type
str

search(*query*, *column='all'*, *wavelength_filter=None*, *fuzzy=True*)

Search the catalog by the query string in the requested column.

Parameters

- **query** (*str, List[str]*) – String or list of strings to search.
- **column** (*str, optional*) – Column-strings or list of strings to search.
- **wavelength_filter** (*float, int, List[float, int]*) – Wavelengths in nm included in the results. Default to None.
- **fuzzy** (*bool, optional*) – Search approximate entries. Defaults to True.

Returns

Filtered Catalog dataframe.

Return type

pd.DataFrame

1.3.2 Dispersions

The dispersions are the central part of pyElli and the transfer-matrix method. They describe the change of dielectric function or refractive index with the wavelength. In pyElli the default wavelength unit is nm. Each dispersion has two distinct sets of parameters:

- Parameters which can be given only once (single parameters).
- Parameters which can be given in multiple sets (repeated parameters), e.g. a set of oscillator parameters.

The syntax for each of the parameter sets is different. For the single parameters they are given in the class constructor:

```
Cauchy(n0=1.458, n1=3.54e-3, n2=0, k0=0, k1=0, k2=0)
```

Repeated parameters are added via the add() function:

```
Sellmeier().add(A=1, B=1).add(A=1, B=2)
```

For dispersions having both, single and repeated parameters can be used together:

```
TaucLorentz(Eg=2).add(A=10, E=2.5, C=0.1)
```

If parameters are not fully provided, they are set to their respective default values. The available parameters and their respective default values are given in the respective class documentation.

All classes inherit from the abstract base class *Dispersion*. It provides basic functionality, such as returning dataframes or arrays containing the wavelength dependent dielectric function of the dispersion relation at current parameter set.

Dispersions can be added with the + operator, or if you want to chain more than two dispersions together you may have a look at the *DispersionSum* class.

PyElli also provides tabulated dispersions from the Refractiveindex.info database. They can be accessed with the `RII` class.

Constant Refractive Index

class elli.dispersions.**ConstantRefractiveIndex**(**args*, ***kwargs*)

Constant refractive index.

Single parameters:

n
The constant value of the refractive index. Defaults to 1.

Repeated parameters:
–

Output:

$$\varepsilon(\lambda) = \boldsymbol{n}^2$$

Epsilon Infinity

`class elli.dispersions.EpsilonInf(*args, **kwargs)`

Constant epsilon infinity.

Single parameters:

eps
Constant value for the constant epsilon. Defaults to 1.

Repeated parameters:
–

Output:

$$\varepsilon(\lambda) = \textbf{eps}$$

Cauchy

`class elli.dispersions.Cauchy(*args, **kwargs)`

Cauchy dispersion.

Single parameters:

n0
Defaults to 1.5.

n1
Defaults to 0. Unit in nm^2.

n2
Defaults to 0. Unit in nm^4.

k0
Defaults to 0.

k1
Defaults to 0. Unit in nm^2.

k2
Defaults to 0. Unit in nm^4.

Repeated parameters:
–

Output:

$$\varepsilon^{1/2}(\lambda) = \boldsymbol{n_0} + 100\boldsymbol{n_1}/\lambda^2 + 10^7\boldsymbol{n_2}/\lambda^4 + i(\boldsymbol{k_0} + 100\boldsymbol{k_1}/\lambda^2 + 10^7\boldsymbol{k_2}/\lambda^4)$$

class elli.dispersions.**CauchyUrbach**(**args*, ***kwargs*)

Cauchy dispersion, with an Urbach Tail absorption.

Single parameters:

n0
Defaults to 1.5.

B
Defaults to 0. Unit in 1/eV^2.

C
Defaults to 0. Unit in 1/eV^4.

D
Defaults to 0.

Eg
Defaults to 0. Unit in eV.

Eu
Defaults to 1. Unit in eV.

Repeated parameters:

–

Output:

$$n(E) = \boldsymbol{n_0} + \boldsymbol{B}E^2 + \boldsymbol{C}E^4 + i\boldsymbol{D}\exp(\frac{E - \boldsymbol{E_g}}{\boldsymbol{E_u}})$$

References

- Fujiwara: Spectroscopic Ellipsometry: Principles and Applications, John Wiley & Sons Ltd, 2007, p. 258

class elli.dispersions.**CauchyCustomExponent**(**args*, ***kwargs*)

Cauchy dispersion with custom exponents.

Single parameters:

n0
Defaults to 1.5.

Repeated parameters:

f
Defaults to 0.

e
Defaults to 1.

Output:

$$\varepsilon^{1/2}(\lambda) = \boldsymbol{n_0} + \sum_j \boldsymbol{f}_j \cdot \lambda^{\boldsymbol{e}_j}$$

Sellmeier

class elli.dispersions.**Sellmeier**(**args*, ***kwargs*)

Sellmeier dispersion.

Single parameters:

–

Repeated parameters:

A
Coefficient for n^2 contribution. Defaults to 0.

B
Resonance wavelength. Defaults to 0. Unit in μm^2.

Output:

$$\varepsilon(\lambda) = 1 + \sum_j \boldsymbol{A}_j \cdot \lambda^2/(\lambda^2 - \boldsymbol{B}_j)$$

With j as the index of the respective oscillator.

class elli.dispersions.**SellmeierCustomExponent**(**args*, ***kwargs*)

Sellmeier dispersion with custom exponents.

Single parameters:

–

Repeated parameters:

A
Coefficient for n^2 contribution. Defaults to 0.

e_A
Exponent for the wavelength in the numerator. Defaults to 1.

B
Resonance wavelength. Defaults to 0. Unit in μm^{-2}.

e_B
Exponent for B. Defaults to 1.

Output:

$$\varepsilon(\lambda) = \sum_j \boldsymbol{A}_j \cdot \lambda^{\boldsymbol{e_A}j}/(\lambda^2 - \boldsymbol{B}_j^{\boldsymbol{e_B}j})$$

With j as the index of the respective oscillator.

Drude

Energy parameters

class elli.dispersions.**DrudeEnergy**(**args*, ***kwargs*)

Drude dispersion model with parameters in units of energy. Drude models in the literature typically contain an additional epsilon infinity value. Use *EpsilonInf* to add this parameter or simply add a number, e.g. DrudeEnergy() + 2, where 2 is the value of epsilon infinity.

Single parameters:

A
Amplitude of Drude oscillator. Defaults to 0. Unit in eV^2

gamma
Broadening of Drude oscillator. Defaults to 0. Unit in eV.

Repeated parameters:

–

Output:

$$\varepsilon(E) = \boldsymbol{A}/(E^2 - i \cdot \boldsymbol{gamma} \cdot E)$$

Resistivity parameters

class elli.dispersions.**DrudeResistivity**(**args*, ***kwargs*)

Drude dispersion model with resistivity based parameters. Drude models in the literature typically contain an additional epsilon infinity value. Use *EpsilonInf* to add this parameter or simply do DrudeEnergy() + eps_inf.

Single parameters:

rho_opt
Optical resistivity. Defaults to 1. Unit in -cm.

tau
Mean scattering time. Defaults to 1. Unit in s.

Repeated parameters:

–

Output:

$$\varepsilon(E) = \hbar/(\varepsilon_0 \cdot \boldsymbol{rho\_opt} \cdot \boldsymbol{tau} \cdot E^2 - i \cdot \hbar \cdot E)$$

where $\hbar$ is the planck constant divided by 2π and ε_0 is the vacuum dielectric permittivity.

Lorentz

Wavelength parameters

class elli.dispersions.**LorentzLambda**(**args*, ***kwargs*)

Lorentz dispersion law with parameters in units of wavelengths.

Single parameters:

–

Repeated parameters:

A
Amplitude of the oscillator. Defaults to 1.

lambda_r
Resonance wavelength. Defaults to 0. Unit in nm.

gamma
Broadening of the oscillator. Defaults to 0. Unit in nm.

Output:

$$\varepsilon(\lambda) = 1 + \sum_j \boldsymbol{A}_j \cdot \lambda^2/(\lambda^2 - \boldsymbol{lambda\_r}_j^2 + i \cdot \boldsymbol{gamma}_j \cdot \lambda)$$

The summation index j refers to the respective oscillator.

Energy parameters

class elli.dispersions.**LorentzEnergy**(**args*, ***kwargs*)

Lorentz dispersion law with parameters in units of energy.

Single parameters:

–

Repeated parameters:

A
Amplitude of the oscillator. Defaults to 1.

E
Resonance energy. Defaults 0. Unit in eV.

gamma
Broadening of the oscillator. Defaults to 0. Unit in eV.

Output:

$$\varepsilon(E) = 1 + \sum_j \boldsymbol{A}_j / \left(E^2 - \boldsymbol{E}_j^2 + i \cdot \boldsymbol{gamma}_j \cdot E\right)$$

With j as the index for the respective oscillator.

Tauc-Lorentz

class elli.dispersions.**TaucLorentz**(**args*, ***kwargs*)

Tauc-Lorentz dispersion law. Model by Jellison and Modine.

Single parameters:

Eg
Bandgap energy (eV). Defaults to 1.

Repeated parameters:

A
Strength of the absorption. Typically 10 < A < 200. Defaults to 20.

E
Lorentz resonance energy (eV). Always keep E > Eg!!. Defaults to 1.5.

C
Lorentz broadening (eV). Typically 0 < Ci < 10. Defaults to 1.

Output:

The Tauc lorentz dispersion. Please refer to the references for a full formula.

References

- G.E. Jellision and F.A. Modine, Appl. Phys. Lett. 69 (3), 371-374 (1996)
- Erratum, G.E. Jellison and F.A. Modine, Appl. Phys. Lett 69 (14), 2137 (1996)
- H. Chen, W.Z. Shen, Eur. Phys. J. B. 43, 503-507 (2005)

Cody-Lorentz

class elli.dispersions.**CodyLorentz**(**args*, ***kwargs*)

Cody-Lorentz dispersion law. Model by Ferlauto et al.

Single parameters:

Eg
Bandgap energy (eV). Defaults to 1.6.

A
Amplitude (eV). Defaults to 100.

Et
Energy at which the Urbach tail starts (eV). Defaults to 1.8.

gamma
Broadening (eV). Defaults to 2.4.

Ep
Distance from bandgap for transition from Cody type absorption to Lorentz type absorption (eV). Defaults to 0.8.

E0
Lorentz resonance energy (eV). Defaults to 3.6.

Eu
Exponential decay of the Urbach tail (eV). Defaults to 0.05.

Repeated parameters:
–

Output:
The Cody-Lorentz dispersion. Please refer to the references for a full formula.

References

- Ferlauto et al., J. Appl. Phys. 92, 2424 (2002)

Gaussian

class elli.dispersions.**Gaussian**(**args*, ***kwargs*)

Dispersion law with gaussian oscillators.

Single parameters:
–

Repeated parameters:

A
Amplitude of the oscillator. Defaults to 1.

E
Central energy. Defaults to 1. Unit in eV.

sigma
Broadening of the Gaussian. Defaults to 1. Unit in eV.

Output:

$$\varepsilon(E) = \sum_j 2 \cdot \boldsymbol{A}_j / \sqrt{} \cdot (D\left(2 \cdot \sqrt{2 \cdot \ln(2)} \cdot (E + \boldsymbol{E}_j)/\boldsymbol{sigma}_j\right) - D\left(2 \cdot \sqrt{2 \cdot \ln(2)} \cdot (E - \boldsymbol{E}_j)/\boldsymbol{sigma}_j\right) + i \cdot \left(\boldsymbol{A}_j \cdot \exp\left(-(4 \cdot \ln(2) \cdot (E - \boldsymbol{E}_j)/\boldsymbol{sigma}_j\right)^2 - \boldsymbol{A}_j \cdot \exp\left(-(4 \cdot ln(2) \cdot (E + \boldsymbol{E}_j)/\boldsymbol{sigma}_j\right)^2\right)$$

D is the Dawson function. The summation index j is the index of the respective oscillator.

References

- De Sousa Meneses, Malki, Echegut, J. Non-Cryst. Solids 351, 769-776 (2006)
- Peiponen, Vartiainen, Phys. Rev. B. 44, 8301 (1991)
- Fujiwara, Collins, Spectroscopic Ellipsometry for Photovoltaics Volume 1, Springer International Publishing AG, 2018, p. 137

Tanguy

class elli.dispersions.**Tanguy**(**args*, ***kwargs*)

Fractional dimensional Tanguy model. This model is an analytical expression of Wannier excitons, including bound and unbound states.

Single parameters:

A
Amplitude (eV). Defaults to 1.

d
Dimensionality 1 < d <= 3. Defaults to 3.

gamma
Excitonic broadening (eV). Defaults to 0.1.

R
Excitonic binding energy (eV). Defaults to 0.1.

Eg
Optical band gap energy (eV). Defaults to 1.

a
Sellmeier coefficient for background dielectric constant (eV^2). Defaults to 0.

b
Sellmeier coefficient for background dielectric constant (eV^2). Defaults to 0.

Repeated parameters.
–

Output:
The Tanguy dispersion. Since the formula is rather long it is not written here. Please refer to the references for a full formula.

References

- C. Tanguy, Phys. Rev. Lett. 75, 4090 (1995). Errata, Phys. Rev. Lett. 76, 716 (1996).
- C. Tanguy, Phys. Rev. B. 60. 10660 (1990).

Poles

class elli.dispersions.**Poles**(**args*, ***kwargs*)

Dispersion law for an UV and IR pole, i.e. Lorentz oscillators outside the fitting spectral range and zero broadening.

Single parameters:

A_ir
IR Pole amplitude. Defaults to 1. Unit in eV^2.

A_uv
UV Pole amplitude. Defaults to 1. Unit in eV^2.

E_uv
UV Pole energy. Defaults to 6. Unit in eV.

Repeated parameters:

–

Output:

$$\varepsilon(E) = \boldsymbol{A\_ir}/E^2 + \boldsymbol{A\_uv}/(\boldsymbol{E\_uv}^2 - E^2)$$

Polynomial

class elli.dispersions.**Polynomial**(**args*, ***kwargs*)

Polynomial expression for the dielectric function.

Single parameters:

e0
Defaults to 1.

Repeated parameters:

f
Defaults to 0.

e
Defaults to 0.

Output:

$$\varepsilon(\lambda) = \boldsymbol{\varepsilon_0} + \sum_j \boldsymbol{f}_j \cdot \lambda^{\boldsymbol{e}_j}$$

Tabulated values

Refractive index values

class elli.dispersions.**Table**(**args*, ***kwargs*)

Dispersion specified by a table of wavelengths (nm) and refractive index values. Please not that this model will produce errors for wavelengths outside the provided wavelength range.

Single parameters:

lbda (list)
Wavelengths in nm. This value must be provided.

n
Complex refractive index values in the convention n + ik. This value must be provided.

kind
Type of interpolation (see scipy.interpolate.interp1d for more information). Defaults to 'linear'.

Repeated parameters:
–

Output:
The interpolation in the given wavelength range.

Epsilon values

`class elli.dispersions.TableEpsilon(*args, **kwargs)`

Dispersion specified by a table of wavelengths (nm) and dielectric function values. Please not that this model will produce errors for wavelengths outside the provided wavelength range.

Single parameters:

lbda (list)
Wavelengths in nm. This value must be provided.

epsilon
Complex dielectric function values in the convention 1 + i2. This value must be provided.

kind
Type of interpolation (see scipy.interpolate.interp1d for more information). Defaults to 'linear'.

Repeated parameters:
–

Output:
The interpolation in the given wavelength range.

`dielectric_function(lbda)`

Calculates the dielectric function in a given wavelength window.

Parameters
lbda (*npt.ArrayLike*) – The wavelength window with unit nm.

Returns
The dielectric function for each wavelength point.

Return type
npt.NDArray

Pseudo dielectric function

`class elli.dispersions.PseudoDielectricFunction(*args, **kwargs)`

A pseudo dielectric function generated from experimental psi/delta values. Please note that the pseudo dielectric function can lead to unphysical behaviour, such as negative refractive indices or other spurious artifacts. Additionally, this formula is only valid for a bulk absorbing material and yields wrong results for layered materials. Therefore, it is preferable to use the pseudo dielectric function only as a helper for constructing other dispersion models.

Single parameters:

angle
The measurement angle in degree under which the psi/delta values where obtained.

lbda
The wavelength region of the measurement data. Units in nm.

psi
The psi values of the measurement. Units in degree.

delta
The delta values of the measurement. Units in degree.

Repeated parameters:
–

Output:

$$\varepsilon(\lambda) = \sin^2(\Theta) \cdot \left(1 + \tan^2(\Theta)\left(\frac{1-\rho}{1+\rho}\right)\right)$$

With

$$\rho = \tan(\Psi) \cdot \exp(-i\Delta)$$

Θ is the angle of incidence.

`dielectric_function`(*lbda*)

Calculates the dielectric function in a given wavelength window.

Parameters
lbda (*npt.ArrayLike*) – The wavelength window with unit nm.

Returns
The dielectric function for each wavelength point.

Return type
npt.NDArray

Spectraray tables

`class elli.dispersions.TableSpectraRay`(*path*)

Helper class to load spectraray's tabulated dielectric functions.

Parameters
path (*str*) – Defines the folder where the Spectraray files are saved.

`load_dispersion_table`(*fname*)

Load a dispersion table from a ascii file in the spectraray materials ascii format. This only accounts for tabulated dielectric function data. Spectraray also stores dispersion data in other formats, but this function is not able to read these other formats.

Parameters
fname (*str*) – The filename of the spectraray ascii file.

Returns
A dispersion object containing the tabulated data.

Return type
TableEpsilon

Abstract classes

These classes serve as basic interfaces for dispersions and convenient + generalized handling.

Dispersion

This is the abstract base class which is the parent class of all dispersions. It supplies basic functionalities, such as extracting wavelength dependent epsilon or refractive index values.

class elli.dispersions.base_dispersion.**Dispersion**(**args*, ***kwargs*)

A dispersion based on a dielectric function formulation.

as_index()

Returns this class as IndexDispersion. This method may be used to add dielectric and index based dispersions. Please ensure that you know what you are doing as building dielectric and index based dispersions is normally mathematically wrong.

DispersionFactory

This factory class returns a fully initialized class identified by a string. This is useful if you determine the appropriate dispersion relation during runtime of your program.

class elli.dispersions.base_dispersion.**DispersionFactory**

A factory class for dispersion objects

static get_dispersion(*identifier*, **args*, ***kwargs*)

Creates a Dispersion object identified by its string name and initializes it with the given parameters.

Parameters
identifier (*str*) – Identifier of the Dispersion object, e.g. Cauchy.

Returns
The Dispersion object initialized with the given parameters.

Return type
DispersionLaw

DispersionSum

This object can be used to add an arbitrary number of dispersions. The overloaded + operator of the dispersion base class creates an instance of this object with two classes as parameters. If you want to chain a lot of dispersions, it may be more performant to use this class directly.

class elli.dispersions.base_dispersion.**DispersionSum**(**disps*)

Represents a sum of two dispersions

dielectric_function(*lbda*)

Calculates the dielectric function in a given wavelength window.

Parameters
lbda (*npt.ArrayLike*) – The wavelength window with unit nm.

Returns
The dielectric function for each wavelength point.

Return type
npt.NDArray

InvalidParameters

This exception is thrown whenever a dispersion got invalid parameters.

exception elli.dispersions.base_dispersion.**InvalidParameters**

Exception for invalid dispersion parameters.

1.3.3 Materials & EMA

The Materials submodule provides the classes to build complex materials from *dispersion models*.

The simplest provided material is the *IsotropicMaterial*. It takes only one Dispersion which is used for all three axis of the material. It can also be created by calling dispersion.get_mat().

For Crystals with two or three different dispersions, there are the *UniaxialMaterial* and *UniaxialMaterial* respectively.

The respective materials can be rotated to achieve different crystal orientations. A good visualization for these rotations can be found in Figure 6.10 of Fujiwara's book 'Spectroscopic Ellipsometry'[1].

Additionally two materials can be combined via various :ref:'Effective medium approximations', to create mixtures or account for interface roughness.

References

Abstract base classes

class elli.materials.**Material**

Base class for materials (abstract class).

get_refractive_index(*lbda*)

Gets the refractive index tensor for wavelength 'lbda'.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Refractive index tensor.

Return type
npt.NDArray

abstract get_tensor(*lbda*)

Gets the permittivity tensor of the material for wavelength 'lbda'.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Permittivity tensor.

Return type
npt.NDArray

class elli.materials.**MixtureMaterial**(*host_material*, *guest_material*, *fraction*)

Abstract Class for mixed materials.

Creates a material mixture from two materials.

Parameters

[1] H. Fujiwara, Spectroscopic Ellipsometry, Principles and Applications, Chichester, UK, John Wiley & Sons, Ltd (2007). https://doi.org/10.1002/9780470060193

- **host_material** (Material) – Host Material.
- **guest_material** (Material) – Material incorporated in the host.
- **fraction** (*float*) – Fraction of the guest material (Range 0 - 1).

get_tensor(*lbda*)

Gets the permittivity tensor of the material for wavelength 'lbda'.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Permittivity tensor.

Return type
npt.NDArray

abstract get_tensor_fraction(*lbda*, *fraction*)

Gets the permittivity tensor of the material for wavelength 'lbda', while overwriting the set fraction. Used in VaryingMixtureLayers.

Parameters

- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **fraction** (*float*) – Fraction of the guest material used for evaluation. (Range 0 - 1).

Returns
Permittivity tensor.

Return type
npt.NDArray

set_constituents(*host_material*, *guest_material*)

Sets Materials in the mixture.

Parameters

- **host_material** (Material) – Host Material.
- **guest_material** (Material) – Material incorporated in the host.

set_fraction(*fraction*)

Sets fraction and checks if fraction is in range from 0 to 1.

Parameters
fraction (*float*) – Fraction of the guest material (Range 0 - 1).

class elli.materials.**SingleMaterial**

Base class for non-mixed materials (abstract class).

get_tensor(*lbda*)

Gets the permittivity tensor of the material for wavelength 'lbda'.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Permittivity tensor.

Return type
npt.NDArray

abstract set_dispersion()

Sets dispersion relation of the material.

set_rotation(*r*)

Sets rotation of the Material.

Parameters
r (*npt.NDArray*) – rotation matrix (from *rotation_euler* or others)

Isotropic and non-isotropic materials

class elli.materials.**IsotropicMaterial**(*dispersion*)

Isotropic material.

Creates isotropic material with a dispersion.

Parameters
dispersion (Dispersion) – Dispersion relation of all three crystal directions.

set_dispersion(*dispersion*)

Sets dispersion relation of the isotropic material.

Parameters
dispersion (Dispersion) – Dispersion relation of all three crystal directions.

class elli.materials.**UniaxialMaterial**(*dispersion_o*, *dispersion_e*)

Uniaxial material.

Creates a uniaxial material with two dispersions.

Parameters

- **dispersion_o** (Dispersion) – Dispersion relation for ordinary crystal axes (x and y direction).
- **dispersion_e** (Dispersion) – Dispersion relation for extraordinary crystal axis (z direction).

set_dispersion(*dispersion_o*, *dispersion_e*)

Sets dispersion relations of the uniaxial material.

Parameters

- **dispersion_o** (Dispersion) – Dispersion relation for ordinary crystal axes (x and y direction).
- **dispersion_e** (Dispersion) – Dispersion relation for extraordinary crystal axis (z direction).

class elli.materials.**BiaxialMaterial**(*dispersion_x*, *dispersion_y*, *dispersion_z*)

Biaxial material.

Creates a biaxial material with three dispersions.

Parameters

- **dispersion_x** (Dispersion) – Dispersion relation for x crystal axes.
- **dispersion_y** (Dispersion) – Dispersion relation for y crystal axes.
- **dispersion_z** (Dispersion) – Dispersion relation for z crystal axes.

set_dispersion(*dispersion_x*, *dispersion_y*, *dispersion_z*)

Sets dispersion relations of the biaxial material.

Parameters

- **dispersion_x** (Dispersion) – Dispersion relation for x crystal axes.
- **dispersion_y** (Dispersion) – Dispersion relation for y crystal axes.
- **dispersion_z** (Dispersion) – Dispersion relation for z crystal axes.

Effective medium approximations

class elli.materials.**VCAMaterial**(*host_material*, *guest_material*, *fraction*)

Mixture Material approximated with a simple virtual crystal like average.

$$\varepsilon_{\text{eff}} = (1 - f)\varepsilon_h + f\varepsilon_g$$

where:

- ε_{eff} is the effective permittivity of host/mixture material,
- ε_h is the permittivity of the host mixture material,
- ε_g is the permittivity of the guest mixture material and
- f is the volume fraction of the guest in the host material.

Creates a material mixture from two materials.

Parameters

- **host_material** (Material) – Host Material.
- **guest_material** (Material) – Material incorporated in the host.
- **fraction** (*float*) – Fraction of the guest material (Range 0 - 1).

get_tensor_fraction(*lbda*, *fraction*)

Gets the permittivity tensor of the marterial for wavelength 'lbda', while overwriting the set fraction.

Parameters

- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **fraction** (*float*) – Fraction of the guest material used for evaluation. (Range 0 - 1).

Returns
Permittivity tensor.

Return type
npt.NDArray

class elli.materials.**BruggemanEMA**(*host_material*, *guest_material*, *fraction*)

Mixture Material approximated with the Bruggeman formula for isotropic spherical inclusions.

Returns one of the two analytical solutions to this quadratic equation:

$$2\varepsilon_{\text{eff}}^2 + [(3f - 2)\varepsilon_a + (1 - 3f)\varepsilon_b]\varepsilon_{\text{eff}} - \varepsilon_a \cdot \varepsilon_b = 0$$

where ε_{eff} is the effective permittivity of host/mixture material, ε_a is the permittivity of the first mixture material, ε_b is the permittivity of the second mixture material and f is the volume fraction of material a in the material b.

References

- Ph.J. Rouseel; J. Vanhellemont; H.E. Maes. (1993) Thin Solid Films, 234, 423-427

Creates a material mixture from two materials.

Parameters

- **host_material** (Material) – Host Material.
- **guest_material** (Material) – Material incorporated in the host.
- **fraction** (*float*) – Fraction of the guest material (Range 0 - 1).

get_tensor_fraction(*lbda*, *fraction*)

Gets the permittivity tensor of the material for wavelength 'lbda', while overwriting the set fraction.

Parameters

- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **fraction** (*float*) – Fraction of the guest material used for evaluation. (Range 0 - 1).

Returns
Permittivity tensor.

Return type
npt.NDArray

class elli.materials.**MaxwellGarnettEMA**(*host_material*, *guest_material*, *fraction*)

Mixture Material approximated with the Maxwell Garnett formula. It is valid for spherical inclusions with small volume fraction.

$$\varepsilon_{\text{eff}} = \varepsilon_h \frac{2f(\varepsilon_g - \varepsilon_h) + \varepsilon_g + 2\varepsilon_h}{2\varepsilon_h + \varepsilon_g - f(\varepsilon_g - \varepsilon_h)}$$

where:

- ε_{eff} is the effective permittivity of host/mixture material,
- ε_h is the permittivity of the host mixture material,
- ε_g is the permittivity of the guest mixture material and
- f is the volume fraction of the guest in the host material.

Creates a material mixture from two materials.

Parameters

- **host_material** (Material) – Host Material.
- **guest_material** (Material) – Material incorporated in the host.
- **fraction** (*float*) – Fraction of the guest material (Range 0 - 1).

get_tensor_fraction(*lbda*, *fraction*)

Gets the permittivity tensor of the material for wavelength 'lbda', while overwriting the set fraction.

Parameters

- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **fraction** (*float*) – Fraction of the guest material used for evaluation. (Range 0 - 1).

Returns
Permittivity tensor.

Return type
npt.NDArray

class elli.materials.**LooyengaEMA**(*host_material*, *guest_material*, *fraction*)

Mixture Material approximated with Looyenga's formula. Valid for materials with small contrast.

$$\varepsilon_{\text{eff}} = ((1-f)\varepsilon_h^{1/3} + f\varepsilon_g^{1/3})^3$$

where:

- ε_{eff} is the effective permittivity of host/mixture material,
- ε_h is the permittivity of the host mixture material,
- ε_g is the permittivity of the guest mixture material and
- f is the volume fraction of the guest in the host material.

References

Looyenga, H. (1965). Physica, 31(3), 401–406.

Creates a material mixture from two materials.

Parameters

- **host_material** (*Material*) – Host Material.
- **guest_material** (*Material*) – Material incorporated in the host.
- **fraction** (*float*) – Fraction of the guest material (Range 0 - 1).

get_tensor_fraction(*lbda*, *fraction*)

Gets the permittivity tensor of the material for wavelength 'lbda', while overwriting the set fraction.

Parameters

- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **fraction** (*float*) – Fraction of the guest material used for evaluation. (Range 0 - 1).

Returns
Permittivity tensor.

Return type
npt.NDArray

1.3.4 Structures

The virtual representation of a sample is finally build using the Structure class.

A *Structure* consists of one IsotropicMaterial used as semi-infinite entry material and a second arbitrary Material as exit material. A list of Layers can be added as desired in between these materials, to create a 1D model of layered media.

The basic *Layer* consists of a material and an assigned thickness. An arbitrary sequence of layers can be stacked and repeated by *RepeatedLayers*, to create Bragg-mirror or multiple quantum well structures.

There are also classes to approximate layers with varying properties along the z-axis (inhomogeneous layers) by creating multiple thinner homogeneous slices:

- *TwistedLayer* is able to represent rotating materials, like twisted nematic materials.
- *VaryingMixtureLayer* takes an *MixtureMaterial* and uses a gradient as mixture fraction.

Layers

Abstract base classes

class elli.structure.**AbstractLayer**

Abstract class for a layer.

abstract get_permittivity_profile(*lbda*)

Returns the permittivity profile of the layer for the given wavelengths.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Returns list of tuples [(thickness, dielectric tensor), ...]

Return type
List[Tuple[float, npt.NDArray]]

class elli.structure.**InhomogeneousLayer**

Abstract base class for inhomogeneous layers with varying properties in z-direction.

get_permittivity_profile(*lbda*)

Returns the permittivity profile of the layer for the given wavelengths. The tensor is evaluated in the middle of each slice.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Returns list of tuples [(d1, epsilon1), (d2, epsilon2), ...]

Return type
List[Tuple[float, npt.NDArray]]

get_slices()

Returns z slicing with the position relative to this layer, not to the whole structure.

Returns
array of 'z' positions [z0, z1,... , zmax], with z0 = 0 and zmax = z{d+1}

Return type
npt.NDArray

abstract get_tensor(*z*, *lbda*)

Returns permittivity tensor matrix for position 'z'.

set_divisions(*div*)

Defines the number of slices to simulate the layer.

Parameters
div (*int*) – Number of slices for the layer

set_material(*material*)

Defines the material for the layer.

Parameters
material (Material) – Material object

set_thickness(*thickness*)

Defines the thickness of the layer in nm.

Parameters
thickness (*float*) – Thickness of the layer in nm.

Homogeneous layers

class elli.structure.**Layer**(*material*, *thickness*)

Homogeneous layer of dielectric material.

New layer of material 'material', with thickness 'thickness'

Parameters

- **material** (Material) – Material object
- **thickness** (*float*) – Thickness of layer (in nm)

get_permittivity_profile(*lbda*)

Returns the permittivity profile of the layer for the given wavelengths.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Returns a list containing one tuple [(thickness, dielectric tensor)]

Return type
List[Tuple[float, npt.NDArray]]

set_material(*material*)

Defines the material for the layer.

Parameters
material (Material) – Material object

set_thickness(*thickness*)

Defines the thickness of the layer in nm.

Parameters
thickness (*float*) – Thickness of the layer in nm.

class elli.structure.**RepeatedLayers**(*layers*, *repetitions*, *before=0*, *after=0*)

Repeated structure of layers.

Create a repeated structure of layers.

Example : For layers [1,2,3] with n=2, before=1 and after=0, the structure will be 3123123.

Parameters

- **layers** (*List[AbstractLayer]*) – List of the repeated layers, starting from z=0
- **repetitions** (*int*) – Number of repetitions
- **before** (*int, optional*) – Number of additional layers before the first period. Defaults to 0.
- **after** (*int, optional*) – Number of additional layers after the last period. Defaults to 0.

`get_permittivity_profile`(*lbda*)

Returns the permittivity profile of the layer for the given wavelengths.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Returns list of tuples [(thickness, dielectric tensor), ...]

Return type
List[Tuple[float, npt.NDArray]]

`set_layers`(*layers*)

Set list of layers.

Parameters
layers (*List[AbstractLayer]*) – List of the repeated layers, starting from z=0

`set_repetitions`(*repetitions*, *before=0*, *after=0*)

Defines the number of repetitions and the first and last layers.

Example : For layers [1,2,3] with n=2, before=1 and after=0, the structure will be 3123123.

Parameters

- **repetitions** (*int*) – Number of repetitions
- **before** (*int, optional*) – Number of additional layers before the first period. Defaults to 0.
- **after** (*int, optional*) – Number of additional layers after the last period. Defaults to 0.

Inhomogeneous layers

`class elli.structure.TwistedLayer`(*material*, *thickness*, *div*, *angle*)

Twisted layer. The material gets rotated around the z axis.

Creates a layer with a twisted material.

Parameters

- **material** (Material) – Material object
- **thickness** (*float*) – Thickness of layer (in nm)
- **div** (*int*) – Number of slices for the layer
- **angle** (*float*) – rotation angle over the distance 'd' (in degrees)

`get_tensor`(*z*, *lbda*)

Gets permittivity tensor matrix for position 'z' and wavelength 'lbda'.

Parameters

- **z** (*float*) – Position in the layer (in nm)
- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Permittivity tensor for position 'z' and wavelength 'lbda'.

Return type
npt.NDArray

set_angle(*angle*)

Defines the total twist angle of this layer.

Parameters
angle (*float*) – Rotation angle over the thickness 'd' of the layer (in degrees)

class elli.structure.**VaryingMixtureLayer**(*material*, *thickness*, *div*, *fraction_modulation=<function VaryingMixtureLayer.<lambda>>*)

Mixture layer, with varying fraction dependent on z Position.

Note: The set fraction of the mixture material will be ignored and replaced by the result of the fraction modulation function.

Parameters

- **material** (MixtureMaterial) – MixtureMaterial object
- **d** (*float*) – Thickness of layer (in nm)
- **div** (*int*) – Number of slices for the layer
- **fraction_modulation** (*Callable[[float], float]*) – Function to modify the fraction amount, takes float from 0 to 1 (top to bottom of layer), should return fraction at that level. Defaults to a linear profile (100% host material to 100% guest material).

get_tensor(*z*, *lbda*)

Gets permittivity tensor matrix for position 'z' and wavelength 'lbda'.

Parameters

- **z** (*float*) – Position in the layer (in nm)
- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Permittivity tensor for position 'z' and wavelength 'lbda'.

Return type
npt.NDArray

set_fraction_modulation(*fraction_modulation=<function VaryingMixtureLayer.<lambda>>*)

Sets function for variation of the mixture over the layer

Parameters
fraction_modulation (*Callable[[float], float]*) – Function to modify the fraction amount, takes float from 0 to 1 (top to bottom of layer), should return fraction at that level. Defaults to a linear profile (100% host material to 100% guest material).

set_material(*material*)

Defines the material for the varying mixture layer.

Parameters
material (MixtureMaterial) – Material object

Structure Class

class elli.structure.**Structure**(*front*, *layers*, *back*)

Description of the whole structure.

Consists of:

- front half-space (incident)

- layer succession
- back half-space (exit)

Parameters

- **front** (IsotropicMaterial) – IsotropicMaterial used as front half space
- **layers** (*List[Layer]*) – List of Layers, starting from z=0
- **back** (Material) – Material used as back half space

evaluate(*lbda*, *theta_i*, *solver=<class 'elli.solver4x4.Solver4x4'>*, ***solver_kwargs*)

Return the Evaluation of the structure for the given parameters with standard settings.

Parameters

- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **theta_i** (*float*) – Incident angle of the experiment (in degrees).
- **solver** (Solver, *optional*) – Choose which solver class is used. Defaults to Solver4x4.
- **solver_kwargs** (*optional*) – Keyword arguments for the Solver can be appended as arguments.

Returns
Result of the experiment.

Return type
Result

get_permittivity_profile(*lbda*)

Returns the permittivity profile of the complete structure for the given wavelengths.

Parameters
lbda (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).

Returns
Returns list of tuples [(thickness, dielectric tensor), ...]

Return type
List[Tuple[float, npt.NDArray]]

set_back_material(*material*)

Defines the back half-space material.

Parameters
material (Material) – Material used as back half space

set_front_material(*material*)

Defines the front half-space material. Has to be isotropic.

Parameters
material (IsotropicMaterial) – IsotropicMaterial used as front half space

set_layers(*layers*)

Sets sequence of layers.

Parameters
layers (*List[Layer]*) – List of Layers, starting from z=0

1.3.5 Experiment

The experiment class defines the external conditions of the sample.

It requires a Structure to evaluate and includes the information about the incident light beam:

- the wavelengths λ
- the incidence angle θ_i
- and the polarization, which can be given by a Jones or Stokes vector

The evaluate method can be called, to start the calculation of the optical properties. To choose a Solver to be used in the calculation, the solver class is provided as an argument and an object will be created automatically.

The experiment class is only needed in special cases an can be skipped by calling *elli.structure.Structure.evaluate()*.

class elli.experiment.**Experiment**(*structure*, *lbda*, *theta_i*, *vector=None*)

Bases: object

Description of a virtual experiment to simulate the behavior of a structure.

Creates a virtual experiment to simulate the behavior of a structure.

Parameters

- **structure** (Structure) – Structure object to evaluate.
- **lbda** (*npt.ArrayLike*) – Single value or array of wavelengths (in nm).
- **theta_i** (*float*) – Incident angle (in degrees).
- **vector** (*npt.ArrayLike, optional*) – Jones or Stokes vector of incident light. Defaults to diagonal polarization ([1, 0, 1, 0]).

evaluate(*solver=<class 'elli.solver4x4.Solver4x4'>*, ***solver_kwargs*)

Evaluates the experiment with the given solver.

Parameters

- **solver** (Solver, *optional*) – Choose which solver class is used. Defaults to Solver4x4.
- **solver_kwargs** (*optional*) – Keyword arguments for the Solver can be appended as arguments.

Returns
Result of the experiment.

Return type
Result

jones_vector = None

lbda = None

set_lbda(*lbda*)

Set experiment wavelengths.

Parameters
lbda (*npt.ArrayLike*) – single value or array of wavelengths (in nm).

set_structure(*structure*)

Defines the Structure to evaluate.

Parameters
structure (Structure) – Structure object to evaluate.

set_theta(*theta_i*)

Set incident angle to evaluate.

Parameters
theta_i (*float*) – Incident angle (in degrees).

set_vector(*vector*)

Defines the Jones or Stokes vector of the incident Light.

Jones: [1, 0]: horizontal polarized [0, 1]: vertical polarized [1/sqrt(2), 1/sqrt(2)]: diagonal polarized [1/sqrt(2),-1/sqrt(2)]: anti-diagonal polarized

Stokes: [1,0,0,0]: unpolarized light [1,1,0,0]: horizontal polarized [1,-1,0,0]: vertical polarized [1,0,1,0]: diagonal polarized [1,0,-1,0]: anti-diagonal polarized

Parameters
vector (*npt.ArrayLike*) – Jones or Stokes vector of incident light.

stokes_vector = None

structure = None

theta_i = None

1.3.6 Solvers

For calculation of light interaction in material stacks the transfer matrix method is used. In pyElli the *Solver* classes provide the necessary toolset for two kinds of transfer matrix algorithms. They are not intended to be used directly, but rather to be provided in the evaluation in the *Structure* class. The *Solver2x2* is a simple and fast algorithm for isotropic materials. It splits the calculation into two 2x2 matrices, one for the s and one for the p polarized light.

The *Solver4x4* is a more complex algorithm for anisotropic materials. It employs a full 4x4 matrix formulation for all light interaction. It is based on the Berreman matrix formalism[1]. In the Berreman formalism a propagator for matrix exponentials is needed. pyElli provides different implementations to be used in the calculation of the transfer matrices. The *PropagatorEig* is based on solving the eigenvalues of the first order approximation of the matrix exponential. Although, it is very fast it is not very accurate. The *PropagatorExpm* is solving the matrix exponential by the Pade approximation. It can use SciPy as backend, but for performance-critical tasks, it is recommended to install PyTorch.

References

Solver base class (Solver)

class elli.solver.**Solver**(*experiment*)

Bases: ABC

Solver base class to evaluate Experiment objects. Here the experiment and structure get unpacked and the simulation results get returned.

The actual simulation is handled by subclasses. Therefore, this class should never be called directly.

abstract calculate()

experiment = None

jones_vector = None

[1] Dwight W. Berreman, "Optics in Stratified and Anisotropic Media: 4×4-Matrix Formulation," J. Opt. Soc. Am. 62, 502-510 (1972)

lbda = None

permittivity_profile = None

structure = None

theta_i = None

2x2 Matrix Solver (Solver2x2)

class elli.solver2x2.**Solver2x2**(*experiment*)

Bases: *Solver*

Solver class to evaluate Experiment objects. Simple but fast 2x2 transfer matrix method. Cannot handle anisotropy or anything fancy, thus Jonas and Mueller matrices cannot be calculated (respective functions return None).

calculate()

Calculates the transfer matrix for the given material stack

static fresnel(*n_i*, *n_t*, *th_i*, *th_t*)

Calculate fresnel coefficients at the interface of two materials

Parameters

- **n_i** – Refractive index of the material of the incident wave
- **n_t** – Refractive index of the material of the transmitted wave
- **thi_i** – Incident angle of the incident wave
- **thi_t** – Refracted angle of the transmitted wave

Returns

s-polarized reflection coefficient r_p: p-polarized reflection coefficient t_s: s-polarized transmission coefficient t_p: p-polarized transmission coefficient

Return type

r_s

static is_forward_angle(*n*, *theta*)

list_snell(*n_list*)

4x4 Matrix Solver (Solver4x4)

class elli.solver4x4.**Propagator**

Bases: ABC

Propagator abstract base class.

abstract calculate_propagation(*delta*, *thickness*, *lbda*)

Calculates propagation for a given Delta matrix and layer thickness.

Parameters

- **delta** (*npt.NDArray*) – Delta Matrix
- **thickness** (*float*) – Thickness of layer (nm)
- **lbda** (*npt.ArrayLike*) – Wavelengths to evaluate (nm)

Returns
Propagator for the given layer

Return type
npt.NDArray

class elli.solver4x4.**PropagatorEig**

Bases: *Propagator*

Propagator class using the eigenvalue decomposition method.

calculate_propagation(*delta*, *thickness*, *lbda*)

Calculates propagation for a given Delta matrix and layer thickness with eigenvalue decomposition.

Parameters

- **delta** (*npt.NDArray*) – Delta Matrix
- **thickness** (*float*) – Thickness of layer (nm)
- **lbda** (*npt.ArrayLike*) – Wavelengths to evaluate (nm)

Returns
Propagator for the given layer

Return type
npt.NDArray

class elli.solver4x4.**PropagatorExpm**(*backend='automatic'*)

Bases: *Propagator*

Propagator class using the Padé approximation of the matrix exponential.

The Propagator can use two different backends: SciPy and PyTorch. The default installation only provides SciPy. PyTorch is faster and will be used automatically if available. If you want to install PyTorch please follow the instructions at https://pytorch.org/get-started/locally/.

Parameters
backend (*Literal["torch", "scipy", "automatic"], optional*) – Setting to change the linear algebra provider. Defaults to "automatic".

calculate_propagation(*delta*, *thickness*, *lbda*)

Calculates propagation for a given Delta matrix and layer thickness with the Padé approximation of the matrix exponential.

Parameters

- **delta** (*npt.NDArray*) – Delta Matrix
- **thickness** (*float*) – Thickness of layer (nm)
- **lbda** (*npt.ArrayLike*) – Wavelengths to evaluate (nm)

Returns
Propagator for the given layer

Return type
npt.NDArray

class elli.solver4x4.**PropagatorLinear**

Bases: *Propagator*

Propagator class using a simple linear approximation of the matrix exponential.

calculate_propagation(*delta*, *thickness*, *lbda*)

Calculates propagation for a given Delta matrix and layer thickness with a linear approximation of the matrix exponential.

Parameters

- **delta** (*npt.NDArray*) – Delta Matrix
- **thickness** (*float*) – Thickness of layer (nm)
- **lbda** (*npt.ArrayLike*) – Wavelengths to evaluate (nm)

Returns
Propagator for the given layer

Return type
npt.NDArray

class elli.solver4x4.**Solver4x4**(*experiment*, *propagator=<elli.solver4x4.PropagatorExpm object>*)

Bases: *Solver*

Solver class to evaluate Experiment objects. Based on Berreman's 4x4 method.

static build_delta_matrix(*k_x*, *eps*)

Calculates Delta matrix for given permittivity and reduced wave number.

Parameters

- **k_x** (*npt.ArrayLike*) – reduce wave number, Kx = kx/k0
- **eps** (*npt.NDArray*) – permittivity tensor

Returns
Delta 4x4 matrix: infinitesimal propagation matrix

Return type
npt.NDArray

calculate()

Calculates transition matrices for every element in the structure and resulting Jones matrices.

Returns
Result object with calculation results

Return type
Result

static get_k_z(*material*, *lbda*, *k_x*)

Calculates Kz in a material

Parameters

- **material** (*Material*) – Material of the half-space
- **lbda** (*npt.ArrayLike*) – Wavelengths to evaluate (nm)
- **k_x** (*npt.ArrayLike*) – Reduced wavenumber, Kx = kx/k0

Returns
value of Kz in the material

Return type
npt.NDArray

static transition_matrix_halfspace(*delta*)

Returns transition exit matrix L for any half-space.

Sort eigenvectors of the Delta matrix according to propagation direction first, then according to y component.

Returns eigenvectors ordered like (s+,s-,p+,p-)

Parameters
delta (*npt.NDArray*) – Delta 4x4 matrix: infinitesimal propagation matrix

Returns
Translation matrix for semi-infinite half-spaces

Return type
npt.NDArray

static transition_matrix_iso_halfspace(*k_x*, *epsilon*, *inv=False*)

Returns transition incident or exit matrix L for isotropic half-spaces.

Parameters

- **k_x** (*npt.ArrayLike*) – Reduced wavenumber, Kx = kx/k0
- **epsilon** (*npt.ArrayLike*) – dielectric tensor
- **inv** (*bool, optional*) – If True, returns inverse transition matrix L^-1, used for the incident Matrix Li. Defaults to False.

Returns
transition matrix L

Return type
npt.NDArray

1.3.7 Result

After a calculation is completed, the Solver returns an object of the Result class.

It is used to store the evaluated experiment and all resulting optical properties like Jones matrices and ellipsometric parameters (psi, delta, rho, mueller matrices). A list of all properties is given below.

All properties will return an array in the length of the provided wavelength array of the requested property.

These can be accessed by different methods:

- With dot notation: `result.property`
- With the get method: `result.get('property')`

For Matrix properties, a specific value can be requested using a `property_ij` notation. As `i` and `j` the respective polarization identifiers or numerical indices can be used (r/s or R/L or 1…4).

To make handling multiple experiments easier, they can be grouped into a list and provided to a ResultList object. It provides the same methods for data output as the single Result. The Output is returned as array over the list of results. If needed, these arrays can be averaged and used like a Result object for fitting.

class elli.result.**AveragedResultList**(*results=None*)

Bases: *ResultList*

ResultList with averaging over all results. Can be used as drop-in replacement for Result objects, if for example thickness inhomogeneities need to be simulated.

Creates an ResultList object.

Parameters

- **results** (*List[Result], optional*) – List of results to store. Defaults to None.
- **mean** (*bool, optional*) – Returns the average of all results. Defaults to False.

class elli.result.**Result**(*experiment*, *jones_matrix_r*, *jones_matrix_t*, *power_correction=None*)

Bases: object

Record of a simulation result.

Creates result object, to store simulation data. Gets called by solvers.

Parameters

- **experiment** (Experiment) – Evaluated experiment, with structure and experimental parameters.
- **jones_matrix_r** (*npt.NDArray*) – Jones matrix for the reflection direction.
- **jones_matrix_t** (*npt.NDArray*) – Jones matrix for the transmission direction.
- **power_correction** (*npt.NDArray*) – Correction factors, to get the power transmission values.

property R

Returns the absolute reflectance for unpolarized light.

$$R = (R_{pp} + R_{ss})/2$$

property R_matrix

Returns the reflectance matrix separated for s and p polarization.

$$M_R = \begin{bmatrix} R_{pp} & R_{ps} \\ R_{sp} & R_{ss} \end{bmatrix}$$

property Rc_matrix

Returns the reflectance matrix for circular polarizations.

$$M_{Rc} = \begin{bmatrix} R_{LL} & R_{LR} \\ R_{RL} & R_{RR} \end{bmatrix}$$

property T

Returns the absolute transmittance for unpolarized light.

$$T = (T_{pp}/T_{ss})/2$$

property T_matrix

Returns the transmittance matrix separated for s and p polarization.

$$M_T = \begin{bmatrix} T_{pp} & T_{ps} \\ T_{sp} & T_{ss} \end{bmatrix}$$

property Tc_matrix

Returns the transmittance matrix with the for circular polarizations.

$$M_{Tc} = \begin{bmatrix} T_{LL} & T_{LR} \\ T_{RL} & T_{RR} \end{bmatrix}$$

as_delta_range(*lower*, *upper*)

Returns this result in another delta range

Parameters

- **lower** (*int*) – The lower delta range. Should be in the range of -360 and 0.
- **upper** (*int*) – The upper delta range. Should be in the range of 0 and 360.

Raises

- **TypeError** – Raised when either lower or upper is not an integer.
- **ValueError** – Range must be (0, 180) or a 360 degree range. Otherwise a ValueError is raised.

property delta

Returns the ellipsometric angle Δ in reflection direction.

It results from:

$$\rho = \tan\psi \exp(-i\Delta)$$

property delta_matrix

Returns the matrix of the ellipsometric parameter Δ in reflection direction.

$$M_\Delta = \begin{bmatrix} \Delta_{\text{pp}} & \Delta_{\text{ps}} \\ \Delta_{\text{sp}} & 0 \end{bmatrix}$$

property delta_matrix_t

Returns the matrix of the ellipsometric parameter Δ_t in transmission direction.

$$M_{\Delta,\text{t}} = \begin{bmatrix} \Delta_{\text{t,pp}} & \Delta_{\text{t,ps}} \\ \Delta_{\text{t,sp}} & 0 \end{bmatrix}$$

property delta_t

Returns the ellipsometric angle Δ_t in transmission direction.

It results from:

$$\rho_\text{t} = \tan\psi_\text{t} \exp(-i\Delta_\text{t})$$

get(*name*)

Return the data for the requested variable ‘name’.

Parameters

name (*str*) – Variable name to return. Examples for ‘name’:

- ’r_sp’ : Amplitude reflection coefficient from ‘s’ to ‘p’ polarization.
- ’r_LR’ : Reflection from circular right to circular left polarization.
- ’T_pp’ : Power transmission coefficient from ‘p’ to ‘p’ polarization.
- ’_ps’, ‘_pp’ : Ellipsometry parameters.
- ’psi’, ‘delta’, ‘rho’: Reduced ellipsometry parameters, the whole matrices are returned by ‘psi_matrix’.

Returns
Array of data.

Return type
npt.NDArray

property jones_matrix_r

Returns the Jones matrix with the amplitude reflection coefficients.

$$M_{\mathrm{r}} = \begin{bmatrix} r_{\mathrm{pp}} & r_{\mathrm{ps}} \\ r_{\mathrm{sp}} & r_{\mathrm{ss}} \end{bmatrix}$$

property jones_matrix_rc

Returns the Jones matrix with the amplitude reflection coefficients for circular polarization.

$$M_{\mathrm{rc}} = \begin{bmatrix} r_{\mathrm{LL}} & r_{\mathrm{LR}} \\ r_{\mathrm{RL}} & r_{\mathrm{RR}} \end{bmatrix}$$

property jones_matrix_t

Returns the Jones matrix with the amplitude transmission coefficients.

$$M_{\mathrm{t}} = \begin{bmatrix} t_{\mathrm{pp}} & t_{\mathrm{ps}} \\ t_{\mathrm{sp}} & t_{\mathrm{ss}} \end{bmatrix}$$

property jones_matrix_tc

Returns the Jones matrix with the amplitude transmission coefficients for circular polarization.

$$M_{\mathrm{tc}} = \begin{bmatrix} t_{\mathrm{LL}} & t_{\mathrm{LR}} \\ t_{\mathrm{RL}} & t_{\mathrm{RR}} \end{bmatrix}$$

property mueller_matrix

Returns the Mueller matrix for reflection, calculated from the rho matrix.

property psi

Returns the ellipsometric angle ψ in reflection direction.

It results from:

$$\rho = \tan\psi \exp(-i\Delta)$$

property psi_matrix

Returns the matrix of the ellipsometric parameter ψ in reflection direction.

$$M_{\psi} = \begin{bmatrix} \psi_{\mathrm{pp}} & \psi_{\mathrm{ps}} \\ \psi_{\mathrm{sp}} & 45 \end{bmatrix}$$

property psi_matrix_t

Returns the matrix of the ellipsometric parameter ψ_{t} in transmission direction.

$$M_{\psi,\mathrm{t}} = \begin{bmatrix} \psi_{\mathrm{t,pp}} & \psi_{\mathrm{t,ps}} \\ \psi_{\mathrm{t,sp}} & 45 \end{bmatrix}$$

property psi_t

Returns the ellipsometric angle ψ_t in transmission direction.

It results from:

$$\rho_\mathrm{t} = \tan\psi_\mathrm{t} \exp(-i\Delta_\mathrm{t})$$

property rho

Returns the ellipsometric parameter ρ in reflection direction.

It is calculated by dot product of the rho matrix M_ρ and the Jones vector $\vec{E}$ of the incident light beam. It then takes the ρ_pp element and returns it.

$$M_{\rho,\,\mathrm{exp}} = M_\rho \cdot \vec{E}$$

property rho_matrix

Returns the matrix of the ellipsometric parameter ρ in reflection direction.

$$M_\rho = \begin{bmatrix} \rho_\mathrm{pp} & \rho_\mathrm{ps} \\ \rho_\mathrm{sp} & 1 \end{bmatrix} = r_\mathrm{ss} \begin{bmatrix} r_\mathrm{pp}/r_\mathrm{ss} & r_{ps}/r_\mathrm{ss} \\ r_\mathrm{sp}/r_\mathrm{ss} & 1 \end{bmatrix}$$

property rho_matrix_t

Returns the matrix of the ellipsometric parameter ρ_t in reflection direction.

$$M_{\rho,\mathrm{t}} = \begin{bmatrix} \rho_\mathrm{t,pp} & \rho_\mathrm{t,ps} \\ \rho_\mathrm{t,sp} & 1 \end{bmatrix} = t_\mathrm{ss} \begin{bmatrix} t_\mathrm{pp}/t_\mathrm{ss} & t_\mathrm{ps}/t_\mathrm{ss} \\ t_\mathrm{sp}/t_\mathrm{ss} & 1 \end{bmatrix}$$

property rho_t

Returns the ellipsometric parameter ρ_t in transmission direction.

class elli.result.**ResultList**(*results=None*)

Bases: object

Class to make a row of Results easier to handle.

Creates an ResultList object.

Parameters

- **results** (*List[Result], optional*) – List of results to store. Defaults to None.
- **mean** (*bool, optional*) – Returns the average of all results. Defaults to False.

append(*result*)

Append a single Result to the ResultList.

Parameters
result (Result) – Additional Result to store.

1.3.8 Fitting and plotting

Interactive fitting

PyElli offers several classes and decorators to make fitting easy. The central idea is to construct a class containing the measurement data and an optical model which is fitted to the data with lmfit. Since pyElli uses lmfit under the hood you may take advantage of its vast capabilities.

To make creation of the fitting classes as easy as possible pyElli contains decorators to automatically instantiate the class by providing a function containing the optical model.

Here you see an example of invoking such a decorator with a measurement dataframe **psi_delta** and parameters **params**, an lmfit Parameter or *ParamsHist* object to create a *FitRho* class.

```
@fit(psi_delta, params)
def model(lbda, params):
    ...
```

In the `model` function the actual optical model should be constructed and an *Experiment* object should be returned. A detailed example on how to use this decorator you find in the *basic usage* example.

Psi/Delta fitting

Fitting decorator and class to fit Psi/Delta experiments. Decorator functions for convenient fitting

class elli.fitting.decorator_psi_delta.**FitRho**(*exp_data*, *params*, *model*, *angle=70*, ***kwargs*)

Bases: *FitDecorator*

A class to fit psi/delta or rho based ellipsometry data with two degress of freedom

Intialize the psi/delta fitting class

Parameters

- **exp_data** (*pd.DataFrame*) – The dataframe containing an experimental mueller matrix. It should contain 2 columns with labels and .
- **params** (*Parameters*) – Fitting start parameters
- **model** (*Callable[[npt.NDArray, Parameters], Result]*) – A function taking wavelengths as first parameter and fitting parameters as second, which returns a pyEllis Result object. This function contains the actual model which should be fitted.
- **angle** (*float, optional*) – The angle of incident of the measurement. Used to calculate the Pseudo-Dielectric function. Defaults to 70.

create_widgets()

Create ipywidgets for parameter estimation

fit(*method='leastsq'*)

Execute lmfit with the current fitting parameters

Parameters

method (*str, optional*) – The fitting method to use. Any method supported by scipys curve_fit is allowed. Defaults to 'leastsq'.

Returns

The fitting result

Return type

Result

fit_function(*params*, *lbda*, *rhor*, *rhoi*)

The fit function to minimize the fitting problem

Parameters

- **params** (*Parameters*) – The lmfit fitting Parameters to construct the simulation
- **lbda** (*npt.NDArray*) – Wavelengths in nm

- **rhor** (*npt.NDArray*) – The real part of the experimental rho
- **rhoi** (*npt.NDArray*) – The imaginary part of the experimental rho

Returns
Residual between the calculation with current parameters and experimental data

Return type
npt.NDArray

get_model_data(*params=None*, *repr='psi-delta'*, *append_exp_data=False*)

Gets the data from the provided modle with the provided parameters. If no parameters are provided, the fitted parameters are used (which default to the initial parameters if no fit has been triggered).

Parameters

- **params** (*Parameters, optional*) – The parameters to calculate the model with. If not provided, the fitted parameters are used. Defaults to None.
- **repr** (*str, optional*) – The representation of the model and experiment data. Valid values are 'psi-delta' or 'rho'. Defaults to 'psi-delta'.
- **append_exp_data** (*bool, optional*) – Appends the experimental data if set to True. Defaults to False.

Raises
ValueError – Raised if the representation is not a valid value.

Returns
The model results

Return type
pd.DataFrame

plot()

Plot the fit results as Psi/Delta

plot_rho()

Plot the fit results as Rho

set_pseudo_diel(*update_exp=False*, *update_names=False*)

Sets Plot to Pseudo Dielectric function values

Parameters

- **update_exp** (*bool, optional*) – Flag to change the experimental data as well. Defaults to False.
- **update_names** (*bool, optional*) – Flag to change the label names. Defaults to False.

set_psi_delta(*update_exp=False*, *update_names=False*)

Sets Plot to Psi/Delta values

Parameters

- **update_exp** (*bool, optional*) – Flag to change the experimental data as well. Defaults to False.
- **update_names** (*bool, optional*) – Flag to change the label names. Defaults to False.

set_residual(*update_exp=False*, *update_names=False*)

Sets plots to residual values

Parameters

- **update_exp** (*bool, optional*) – Flag to change the experimental data as well. Defaults to False.
- **update_names** (*bool, optional*) – Flag to change the label names. Defaults to False.

set_rho(*update_exp=False*, *update_names=False*)

Sets Plot to Rho values

Parameters

- **update_exp** (*bool, optional*) – Flag to change the experimental data as well. Defaults to False.
- **update_names** (*bool, optional*) – Flag to change the label names. Defaults to False.

update_selection(*change=None*)

Update plot after selection of displayed data

Parameters
change (*dict, optional*) – A dictionary containing the ipywidgets change event

elli.fitting.decorator_psi_delta.**fit**(*exp_data*, *params*, *angle=70*, ***kwargs*)

A parameters decorator for fitting psi/delta valus. Displays an ipywidget float box for each fitting parameter and an interactive plot to estimate parameters.

Parameters

- **exp_data** (*pd.DataFrame*) – The dataframe containing an experimental mueller matrix. It should contain 2 columns with labels and .
- **params** (*Parameters*) – Fitting start parameters
- **angle** (*float, optional*) – The angle of incident of the measurement. Used to calculate the Pseudo-Dielectric function. Defaults to 70.

Returns
fitting parameters as second, which returns a pyEllis Result object. This function contains the actual model which should be fitted and is automatically provided when used as a decorator.

Return type
Callable[[npt.NDArray, Parameters], *Result*]

Mueller matrix fitting

Fitting decorator and class to fit mueller matrix experiments. Decorator functions for convenient fitting of Mueller matrices

class elli.fitting.decorator_mmatrix.**FitMuellerMatrix**(*exp_mm*, *params*, *model*, ***kwargs*)

Bases: *FitDecorator*

A class to fit mueller matrices to experimental data

Intialize the mueller matrix fitting class

Parameters

- **exp_mm** (*pd.DataFrame*) – The dataframe containing an experimental mueller matrix. It should contain 16 columns with labels Mxy, where xy are the matrix positions.
- **params** (*Parameters*) – Fitting start parameters
- **model** (*Callable[[npt.NDArray, Parameters], Result]*) – A function taking wavelengths as first parameter and fitting parameters as second, which returns a pyEllis Result object. This function contains the actual model which should be fitted

- ****display_single** (*bool*) – Returns a figure containing a single graph, if set to true. Returns a grid of figures otherwise.
- ****sharex** (*bool*) – Ties the zoom of the x-axes together for grid view.
- ****full_scale** (*bool*) – Sets the y-axis scale to [-1, 1] if set to True.

create_widgets()

Create ipywidgets for parameter estimation

fit(*method='leastsq'*)

Execute lmfit with the current fitting parameters

Parameters
method (*str, optional*) – The fitting method to use. Any method supported by scipys curve_fit is allowed. Defaults to 'leastsq'.

Returns
The fitting result

Return type
Result

fit_function(*params*, *lbda*, *mueller_matrix*)

The fit function to minimize the fitting problem

Parameters

- **params** (*Parameters*) – The lmfit fitting Parameters to construct the simulation
- **lbda** (*npt.NDArray*) – Wavelengths in nm
- **mueller_matrix** (*pd.DataFrame*) – The experimental data to compare to the fitted model

Returns

Residual between the calculation
with current parameters and experimental data

Return type
npt.NDArray

get_model_data(*params=None*, *append_exp_data=False*)

Gets the data from the provided model with the provided parameters. If no parameters are provided, the fitted parameters are used (which default to the initial parameters if no fit has been triggered).

Parameters

- **params** (*Parameters, optional*) – The parameters to calculate the model with. If not provided, the fitted parameters are used. Defaults to None.
- **append_exp_data** (*bool, optional*) – Appends the experimental data if set to True. Defaults to False.

Returns
The model results

Return type
pd.DataFrame

plot(***kwargs*)

Plot the fit results

Parameters

- ****display_single** (*bool*) – Returns a figure containing a single graph, if set to true. Returns a grid of figures otherwise.
- ****sharex** (*bool*) – Ties the zoom of the x-axes together for grid view.
- ****full_scale** (*bool*) – Sets the y-axis scale to [-1, 1] if set to True.

Returns
The figure containing the data

Return type
go.Figure

plot_residual(***kwargs*)

Plots the residual between the fit and the experimental data

Parameters

- ****display_single** (*bool*) – Returns a figure containing a single graph, if set to true. Returns a grid of figures otherwise.
- ****sharex** (*bool*) – Ties the zoom of the x-axes together for grid view.
- ****full_scale** (*bool*) – Sets the y-axis scale to [-1, 1] if set to True.

Returns
The figure containing the data

Return type
go.Figure

update_selection(*_=None*)

Update plot after selection of displayed data

Parameters
_ (*dict, optional*) – No function. Just for compliance with ABC.

elli.fitting.decorator_mmatrix.**fit_mueller_matrix**(*exp_mm*, *params*, ***kwargs*)

A parameters decorator for fitting mueller matrices. Displays an ipywidget float box for each fitting parameter and an interactive plot to estimate parameters.

Parameters

- **exp_mm** (*pd.DataFrame*) – The dataframe containing an experimental mueller matrix. It should contain 16 columns with labels Mxy, where xy are the matrix positions.
- **params** (*Parameters*) – Fitting start parameters
- ****display_single** (*bool*) – Returns a figure containing a single graph, if set to true. Returns a grid of figures otherwise.
- ****sharex** (*bool*) – Ties the zoom of the x-axes together for grid view.
- ****full_scale** (*bool*) – Sets the y-axis scale to [-1, 1] if set to True.

Returns
A function taking wavelengths as first parameter and fitting parameters as second, which returns a pyEllis Result object. This function contains the actual model which should be fitted and is automatically provided when used as a decorator.

Return type
Callable[[npt.NDArray, Parameters], *Result*]

elli.fitting.decorator_mmatrix.**mmatrix_to_dataframe**(*exp_df*, *mueller_matrix*, *identifier=None*)

Reshape a numpy 4x4 array containing mueller matrix elements to a dataframe with columns Mxy. The index labels for each column are taken from the provided exp_df.

Parameters

- **exp_df** (*pd.DataFrame*) – The experimental dataframe providing the index and columns for the newly generated dataframe.
- **mueller_matrix** (*npt.NDArray*) – Data to be reshaped into a dataframe
- **identifier** (*str, optional*) – An identifier to append to each column name, in the form Mxy_<identifier>, where Mxy is the old column name. Defaults to None.

Returns
Contains the data from mueller_matrix in the shape of exp_df

Return type
pd.DataFrame

Fitting base class

This is the base class providing basic fitting features. This class is not intended to be used directly, it rather should be inherited from in additional fitting classes. Abstract base class for Decorator functions for convenient fitting

class elli.fitting.decorator.**FitDecorator**

Bases: ABC

The abstract base class for fitting decorators. Providing features for fit, undo and redo buttons.

abstract fit(*method=''*)

Execute lmfit with the current fitting parameters

Parameters
method (*str, optional*) – The fitting method to use. Any method supported by scipys curve_fit is allowed. Defaults to 'leastsq'.

Returns
The fitting result

Return type
Result

fit_button_clicked()

Fit and update plot after the fit button has been clicked

Parameters
selected (*dict*) – Dict containing the current widget information of the selection dropdown.

abstract get_model_data(*params=None*, *append_exp_data=False*)

Gets the data from the provided model with the provided parameters. If no parameters are provided, the fitted parameters are used (which default to the initial parameters if no fit has been triggered).

Parameters

- **params** (*Parameters, optional*) – The parameters to calculate the model with. If not provided, the fitted parameters are used. Defaults to None.

- **append_exp_data** (*bool, optional*) – Appends the experimental data if set to True. Defaults to False.

Returns
The model results

Return type
pd.DataFrame

re_undo_button_clicked(*button*)

Redo or undo an operation on the parameters history object

Parameters
button (*widgets.Button*) – The button instance, which triggered the event.

reset_to_init_params()

Resets the parameters to the initial values

to_csv(**args*, *fname*, *params=None*, *append_exp_data=False*, ***kwargs*)

Saves the current model to csv. This is just a wrapper to pandas Dataframe and any argument to pandas to_csv may be passed as function arguments.

Parameters

- **fname** (*str*) – The file name to save the data to.
- **params** (*Parameters, optional*) – The parameters to calculate the model with. If not provided, the fitted parameters are used. Defaults to None.
- **append_exp_data** (*bool, optional*) – Appends the experimental data if set to True. Defaults to False.

update_params(*change*)

Update plot after a change of fitting parameters

Parameters

- **change** (*dict*) – A dictionary containing the ipywidgets change event
- **selected** (*dict*) – The selected value of the data display dropdown widget

abstract update_selection(*change=None*)

Update plot after selection of displayed data

Parameters
change (*dict, optional*) – A dictionary containing the ipywidgets change event

update_widgets()

Updates the widget values according to the current parameters.

elli.fitting.decorator.**is_in_notebook**()

Checks whether the current shell is in a jupyter notebook.

Returns
True if the shell is in jupyter, False otherwise.

Return type
bool

Parameter class

The parameter class extending lmfit's Parameter class by a history of parameter changes. ParmsHist provides a wrapper around lmfit.Parameters to keep track of the changes made to the parameters.

class elli.fitting.params_hist.**ParamsHist**

Bases: Parameters

A wrapper around lmfit.Parameters to keep track of the changes made to the parameters.

Parameters
usersyms (*dict, optional*) – Dictionary of symbols to add to the asteval.Interpreter (default is None).

clear_history()

Clears the parameters history

commit()

Saves the current parameter set to history.

property history

Gets the entire history

Returns
The history

Return type
List[Parameters]

property history_len

The current length of the history

Returns
The length of the history

Return type
int

property max_history_len

The maximum length of the history

Returns
The maximum length of the history

Return type
int

pop()

Gets to the previous history version and deletes the current element.

revert(*hist_pos*)

Reverts to an older history version and keeps the entire history.

Parameters
hist_pos (*int*) – The history position to revert to.

tracked_add(**args*, ***kwargs*)

Adds a parameter and keeps track of the change in history

update_params(*parameters*)

Updates the current parameters from a lmfit parameters object.

Parameters
parameters (*lmfit.Parameters*) – The lmfit parameters object to update the values from.

update_value(*key*, *value*)

Updates a parameter and keeps track of the change in history

Parameters

- **key** (*str*) – The key to be updated
- **value** (*float*) – The value the key should be updated to

Plotting

Mueller matrix

This is a helper class to plot a mueller matrix dataframe. Plotting functions for Mueller matrices

elli.plot.mueller_matrix.**plot_mmatrix**(*dataframes*, *colors=None*, *dashes=None*, *names=None*, *single=True*, *full_scale=False*, *sharex=False*)

Takes multiple Mueller matrix dataframes with columns Mxy for matrix postion x,y and plots them together. Needs plotly as additional requirement to work.

Parameters

- **dataframes** (*Union[pd.Dataframe, List[pd.DataFrame]]*) – A dataframe or a list of dataframes containing data of the same index.
- **colors** (*Optional[List[str]], optional*) – A list of colors which are cycled for each dataframes index. Defaults to None.
- **dashes** (*Optional[List[str]], optional*) – A list of dash line styles which are cycled for each dataframes index. Defaults to None.
- **names** (*Optional[List[str]], optional*) – A name for each dataframe index. Defaults to None.
- **single** (*bool, optional*) – Uses a single plot if set and a grid if not set. Defaults to True.
- **full_scale** (*bool, optional*) – Sets the y-axis limits to [-1, 1] if set. Defaults to False.
- **sharex** (*bool, optional*) – Ties the zooming of the x-axis together for each plot in grid view. Defaults to False.

Returns
A plotly figure containing the data from dataframes as a grid or single view.

Return type
go.Figure

Structure

Plots a refractive index slice through a stack of materials.

elli.plot.structure.**draw_structure**(*structure*, *lbda=1000*, *method='graph'*, *margin=0.15*)

Draw the structure.

‘method’ : ‘graph’ or ‘section’ Returns : Axes object

elli.plot.structure.**get_index_profile**(*structure*, *lbda*, *v=array([1, 0, 0])*)

Returns refractive index profile.

'v'

[Unit vector, direction of evaluation of the refraction index.] Default value is v = e_x.

elli.plot.structure.**get_permittivity_profile**(*structure*, *lbda*)

Returns permittivity tensor profile.

1.3.9 Reading and writing

Read and write NeXus files in the ellipsometry standard. NeXus is a format originating from large beam line facitilies and is adapted for other smaller labratory experiments to supply an agreed standard for data sharing. For now only reading is supported but in the future there will also be a writer to store the whole optical model and fit inside the NeXus file.

class elli.importer.nexus.**NexusGroupNames**(*entry='entry'*, *sample='sample'*, *instrument='instrument'*)

Contains nexus group names to read from the nexus file

property full_instrument_path

Get the full instrument path with prepended entry name, e.g. entry/instrument

property full_sample_path

Get the full sample path with prepended entry name, e.g. entry/sample

elli.importer.nexus.**read_nexus_materials**(*filename*)

Read the optical materials from a nexus file.

Parameters
filename (*str*) – The nexus filename.

elli.importer.nexus.**read_nexus_psi_delta**(*nxs_filename*, *group_names=None*)

Read a NeXus file containing Psi and Delta data.

Parameters
nxs_filename (*str*) – The filename of the NeXus file.

Raises
ValueError – Is raised when the data is not stored as psi / delta value.

Returns
Psi/Delta DataFrame. The index is a multiindex consisting of the angle of incidents as first column and the wavelength as second column.

Return type
pd.DataFrame

elli.importer.nexus.**read_nexus_rho**(*nxs_filename*)

Reads rho value from NeXus datafile. Currently, this works only with psi / delta representation in the NeXus file.

Raises
ValueError – Is raised when the data is not stored as psi / delta value.

Returns
DataFrame containing the measured data as imaginary rho value. The index is a multiindex consisting of the angle of incidents as first column and the wavelength as second column.

Return type
pd.DataFrame

A helper class to load data from SpectraRay ASCII Files. It only supplies a rudimentary loading of standard psi/delta values and misses some other features.

elli.importer.spectraray.**read_spectraray_mmatrix**(*fname*, *sep='\\s+'*, *decimal='.'*)

Read a mueller matrix spectraray ascii file. Only reads the first entry and does not support reading multiple angles. For multiple angles you have to save the data in multiple files.

Parameters

- **fname** (*str*) – Filename of the measurement ascii file.
- **sep** (*str, optional*) – Data separator in the datafile. Defaults to "s+".
- **decimal** (*str, optional*) – Decimal separator in the datafile. Defaults to ".".

Returns
DataFrame containing the psi/delta data in the format to be further processes inside pyElli.

Return type
pd.DataFrame

elli.importer.spectraray.**read_spectraray_psi_delta**(*fname*, *sep='\\s+'*, *decimal='.'*)

Read a psi/delta spectraray ascii file.

Parameters

- **fname** (*str*) – Filename of the measurement ascii file.
- **sep** (*str, optional*) – Data separator in the datafile. Defaults to "s+".
- **decimal** (*str, optional*) – Decimal separator in the datafile. Defaults to ".".

Returns
DataFrame containing the psi/delta data in the format to be further processes inside pyElli.

Return type
pd.DataFrame

elli.importer.spectraray.**read_spectraray_rho**(*fname*, *sep='\\s+'*, *decimal='.'*)

Read a psi/delta spectraray ascii file and converts it to rho values.

Parameters

- **fname** (*str*) – Filename of the measurement ascii file.
- **sep** (*str, optional*) – Data separator in the datafile. Defaults to "s+".
- **decimal** (*str, optional*) – Decimal separator in the datafile. Defaults to ".".

Returns
DataFrame containing the rho data in the format to be further processes inside pyElli.

Return type
pd.DataFrame

A helper class to load data from Woollam ASCII Files. It supports loading of standard psi/delta values.

elli.importer.woollam.**is_float**(*line*)

Checks whether the given line is a float

Parameters
line (*str*) – The line presumably containing a float

Returns
True if it is a float, False otherwise

Return type
bool

elli.importer.woollam.**read_woollam_psi_delta**(*fname*)

Read a psi/delta woollam ascii file.

Parameters
fname (*str*) – Filename of the measurement ascii file.

Returns
DataFrame containing the psi/delta data in the format to be further processes inside pyElli.

Return type
pd.DataFrame

elli.importer.woollam.**read_woollam_rho**(*fname*)

Read a psi/delta woollam ascii file and converts it to rho values.

Parameters
fname (*str*) – Filename of the measurement ascii file.

Returns
DataFrame containing the rho data in the format to be further processes inside pyElli.

Return type
pd.DataFrame

elli.importer.woollam.**scale_to_nm**(*unit*, *dataframe*)

Scales the wavelength axis of the given dataframe from the provided unit to nm.

Parameters

- **unit** (*str*) – The unit string. Should be pint compatible.
- **dataframe** (*pd.DataFrame*) – The dataframe which axis should be scaled.

Returns
The dataframe with scaled wavelength axis.

Return type
pd.DataFrame

1.3.10 Helper functions

Helper functions for generation rotation matrices and conversions for wavelengths and ellipsometric quantities.

class elli.utils.**DeltaRange**(*lower: int*, *upper: int*)

elli.utils.**calc_pseudo_diel**(*rho*, *angle*, *output='eps'*)

Calculates the pseudo dielectric function of a measurement from rho.

Parameters

- **rho** (*pandas.DataFrame*) – Measurement DataFrame containing rho as complex number as column and wavelength as index
- **angle** (*float*) – Angle of measurement in degree
- **output** (*str, optional*) – Output format for dielectric function. ‘n’: refractive index, ‘eps’: Dielectric function as two-column pandas.DataFrame, ‘epsi’: Dielectric function as imaginary number. Defaults to ‘eps’.

Returns
Frame containing the pseudo dielectric function or refractive index.

Return type
pandas.DataFrame

elli.utils.**calc_rho**(*psi_delta*)

Calculate rho from a Psi-Delta DataFrame. The Psi-Delta DataFrame should be structured as follows:

index
Wavelength

column ‘’
Psi from measurement

column ‘’
Delta from measurement

This format is as returned from SpectraRay.read_psi_delta_file(...).

Parameters
psi_delta (*pandas.DataFrame*) – DataFrame containing Psi+Delta Measurement data

Returns
Frame containing rho as an imaginary number.

Return type
pandas.DataFrame

elli.utils.**conversion_energy2frequency**(*E*)

Converts energy values to frequency values.

$$f = \boldsymbol{E}/\hbar$$

Parameters
E (*npt.ArrayLike*) – Single value or array of energies in eV.

Returns
Frequency in Hz.

Return type
npt.ArrayLike

elli.utils.**conversion_frequency2energy**(*f*)

Converts frequency values to energy values.

$$E = \boldsymbol{f} \cdot \hbar$$

Parameters
f (*npt.ArrayLike*) – Single value or array of frequencies in Hz.

Returns
Energy in eV.

Return type
npt.ArrayLike

elli.utils.**conversion_wavelength_energy**(*value*)

Converts wavelength values to energy values and vice versa.

$$value_{\text{target}} = c \cdot \hbar/\boldsymbol{value}$$

Parameters
value (*npt.ArrayLike*) – Single value or array of wavelengths in nm or energy in eV.

Returns
Energy in eV or wavelength in nm.

Return type
npt.ArrayLike

elli.utils.**conversion_wavelength_frequency**(*value*)

Converts wavelength values to frequency values and vice versa.

$$value_{\text{target}} = c/\boldsymbol{value}$$

Parameters
value (*npt.ArrayLike*) – Single value or array of wavelengths in nm or frequencies in Hz.

Returns
Frequencies in Hz or wavelengths in nm.

Return type
npt.ArrayLike

elli.utils.**conversion_wavelength_wavenumber**(*value*)

Converts wavelength values to wavenumber values and vice versa.

$$value_{\text{target}} = 1/\boldsymbol{value}$$

Parameters
value (*npt.ArrayLike*) – Single value or array of wavelengths in nm or wavenumbers in cm^{-1}.

Returns
Wavenumbers in cm^{-1} or wavelengths in nm.

Return type
npt.ArrayLike

elli.utils.**convert_delta_range**(*delta_values*, *lower*, *upper*)

Shifts the delta values to a given range.

Parameters

- **delta_values** (*npt.ArrayLike*) – Delta values to be converted.
- **lower** (*int*) – The lower delta range. Should be in the range of -360 and 0.
- **upper** (*int*) – The upper delta range. Should be in the range of 0 and 360.

Returns
Delta values with shifted range.

Return type
npt.ArrayLike

elli.utils.**convert_psi_delta_to_isotropic_mueller_matrix**(*psi_delta_df*)

Extract aois and wavelengths values from psi_delta pandas.DataFrame and convert it to a Muellermatrix; only works for isotropic media.

Parameters
psi_delta_df (*pd.DataFrame*) – dataFrame returned from data importers'

Returns
pyElli-compatible DataFrame of the Mueller matrix

Return type
pd.DataFrame

elli.utils.**detect_encoding**(*fname*)

Detects the encoding of file fname. :param fname: Filename :type fname: str

Returns
Encoding identifier string.

Return type
str

elli.utils.**get_qwp_thickness**(*material*, *lbda*)

Return the thickness of a material in nm for a quarter wave plate at wavelength 'lbda'.

Parameters

- **material** (*Material*) – Material object of the quarter wave plate
- **lbda** (*float*) – Wavelength (in nm) at which the quarter wave plate is calculated

Returns
Thickness (in nm) of quarter wave plate

Return type
float

elli.utils.**rotation_euler**(*p*, *n*, *r*)

Returns rotation matrix defined by Euler angles p, n, r.

Successive rotations : z,x',z' Note : The inverse rotation is -r, -n, -p

Parameters

- **p** (*float*) – precession angle, 1st rotation, around z (0..360°).
- **n** (*float*) – nutation angle, 2nd rotation, around x' (0..180°).
- **r** (*float*) – 3rd rotation, around z' (0..360°).

Returns
rotation matrix M_R

Return type
npt.NDArray

elli.utils.**rotation_v**(*v*)

Returns rotation matrix defined by a rotation vector v.

The calculation is made with the matrix exponential $M_R = \exp(W)$, with $W_{ij} = -_{ijk}V_k$, where $_{ijk}$ is the Levi-Civita antisymmetric tensor. If V is separated in a unit vector v and a magnitude , V = ·v, with = V, the calculation of the matrix exponential is avoided, and only sin() and cos() are needed instead.

Note : The inverse rotation is -v

Parameters
v (*npt.ArrayLike*) – rotation vector (list or array)

Returns
rotation matrix M_R

Return type
npt.NDArray

elli.utils.**rotation_v_theta**(*v*, *theta*)

Returns rotation matrix defined by a unit rotation vector and an angle.

Notes : The inverse rotation is (v,-theta)

Parameters

- **v** (*npt.ArrayLike*) – unit vector orienting the rotation (list or array)
- **theta** (*float*) – rotation angle around v in degrees

Returns
rotation matrix M_R

Return type
npt.NDArray

1.3.11 Kramers-Kronig relations

Calculate Kramers-Kronig relations according to Maclaurin's formula[1]. Here, the differential formulation is used, which means that the transformation always diverges to zero at infinity, leaving a free infinity offset. This offset is typically referred to as $\epsilon(\infty)$ in spectroscopy.

Since the Kramers-Kronig relation integrates over the whole spectrum for each point, it is very sensitive to sampling changes and non-zero values. To obtain best values the y-axis should be zero anywhere outside the integration range. However, since this is not possible for every dispersion relation the calculation should be done on a much wider range than the actual interval used. Additionally, the discretisation steps of the x-axis should be kept constant.

For the transformation of the real to imaginary part you need to be especially cautios. Typically, in spectroscopy the real part of the dielectric function is non-zero throughout the whole spectrum and especially $\epsilon(\infty) \neq 0$. This makes the Kramers-Kronig transformation virtually impossible in these cases as it suffers from major uncertanties. Hence, this transformation is included in pyElli only for completeness and for the special cases it may be applicable.

References

elli.kkr.kkr.**im2re**(*im*, *x*)

Calculates the differential Kramers-Kronig relation from the imaginary to real part according to Maclaurin's formula.

The underlying formula reads:

$$\Delta\Re(x_i) = \Re(x_i) - \Re(\infty) = \frac{2}{\pi}\int_0^\infty \frac{x\Im(x)}{x^2 - x_i^2}dx$$

Parameters

- **im** (*numpy.ndarray*) – The imaginary values to transform.
- **x** (*numpy.ndarray*) – The axis on which to transform.

Returns
The transformed real part.

Return type
numpy.ndarray

elli.kkr.kkr.**im2re_reciprocal**(*im*, *x*)

Calculates the differential Kramers-Kronig relation from the imaginary to real part according to Maclaurin's formula. This function assumes a reciprocal x-axis, e.g. wavelength in spectroscopy.

[1] Ohta and Ishida, Appl. Spectroscopy 42, 952 (1988), https://doi.org/10.1366/0003702884430380

The underlying formula reads:

$$\Delta\Re(x_i) = \Re(x_i) - \Re(\infty) = \frac{2}{\pi}\int_0^\infty \frac{x\Im(x)}{1-\frac{x^2}{x_i^2}}dx$$

Parameters

- **im** (*numpy.ndarray*) – The imaginary values to transform.
- **x** (*numpy.ndarray*) – The reciprocal axis on which to transform.

Returns
The transformed real part.

Return type
numpy.ndarray

elli.kkr.kkr.**re2im**(*re*, *x*)

Calculates the differential Kramers-Kronig relation from the real to imaginary part according to Maclaurin's formula.

The underlying formula reads:

$$\Delta\Im(x_i) = \Im(x_i) - \Im(\infty) = \frac{2}{\pi}\int_0^\infty \frac{x_i\Re(x)}{x^2-x_i^2}dx$$

Parameters

- **re** (*numpy.ndarray*) – The real values to transform.
- **x** (*numpy.ndarray*) – The axis on which to transform.

Returns
The transformed imaginary part.

Return type
numpy.ndarray

elli.kkr.kkr.**re2im_reciprocal**(*re*, *x*)

Calculates the differential Kramers-Kronig relation from the real to imaginary part according to Maclaurin's formula. This function assumes a reciprocal x-axis, e.g. wavelength in spectroscopy.

The underlying formula reads:

$$\Delta\Im(x_i) = \Im(x_i) - \Im(\infty) = \frac{2}{\pi}\int_0^\infty \frac{\Re(x)}{x_i-\frac{x^2}{x_i}}dx$$

Parameters

- **re** (*numpy.ndarray*) – The real values to transform.
- **x** (*numpy.ndarray*) – The reciprocal axis on which to transform.

Returns
The transformed imaginary part.

Return type
numpy.ndarray

CHAPTER

TWO

MISC

2.1 Contributing

Welcome to `pyElli` contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but `other kinds of contributions` are also appreciated.

If you are new to using git or have never collaborated in a project previously, please have a look at contribution-guide.org.

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, `Python Software Foundation's Code of Conduct` is a good reference in terms of behavior guidelines.

2.1.1 Issue Reports

If you experience bugs or general issues with `pyElli`, please have a look on the issue tracker. If you don't see anything useful there, please feel free to fire an issue report.

```
Please don't forget to include the closed issues in your search.
Sometimes a solution was already reported, and the problem is considered
solved.
```

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

2.1.2 Documentation Improvements

You can help improve `pyElli` docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

`pyElli` documentation uses Sphinx as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way was a code contribution.

2.1.3 Code Contributions

PyElli consists of different classes representing different parts of an ellipsometric experiment. You can see an overview graph in the documentation. If you want to add a dispersion, solver or material class to extend the present functionality you can do so by creating a new class under the respective abstract base classes.

Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the issue tracker to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated `virtual environment` to avoid any problems with your installed Python packages. This can easily be done via either **virtualenv**

```
virtualenv <PATH TO VENV>
source <PATH TO VENV>/bin/activate
```

or **Miniconda**

```
conda create -n pyElli python=3 six virtualenv pytest pytest-cov
conda activate pyElli
```

Clone the repository

1. Create an user account on github if you do not already have one.
2. Fork the project repository click on the *Fork* button near the top of the page. This creates a copy of the code under your account on github.
3. Clone this copy to your local disk

   ```
   git clone --recurse-submodules https://github.com/PyEllips/pyElli
   cd pyElli
   ```

4. You should run

   ```
   pip install -e ".[fitting,dev]"
   ```

   which installs the package in development mode and all extra requirements in your current virtualenv.

Implement your changes

1. Create a branch to hold your changes

   ```
   git checkout -b my-feature
   ```

   and start making changes. Never work on the master branch!
2. Start your work on this branch. Don't forget to add docstrings to new functions, modules and classes, especially if they are part of public APIs.
3. Add yourself to the list of contributors in `AUTHORS.md`.
4. When you're done editing, do

   ```
   git add <MODIFIED FILES>
   git commit
   ```

   to record your changes in git.

   Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

Moreover, writing a `descriptive commit message` is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

to look for recurring communication patterns.

5. Please check that your changes don't break any unit tests with:

```
pytest
```

6. If you added features, please add them to the documentation and check the documentation status locally by running

```
python setup.py build_sphinx
```

This should create html pages in `build/sphinx/html` for you.

7. We are using the python black formatter throughout the code and sphinx-lint for the docstrings. We supply a pre-commit hook which you can install accordingly to pre-commits documentation, which checks black formatting before the code is committed. For sphinx you have to invoke it manually by installing with

```
pip install sphinx-lint
```

and running

```
sphinx-lint
```

in the project directory.

Submit your contribution

1. If everything works fine, push your local branch to github with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and create a pull request to send your changes for review.

2.2 License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
```

(continues on next page)

(continued from previous page)

```
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.
```

(continues on next page)

(continued from previous page)

```
  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
```

(continues on next page)

(continued from previous page)

```
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
```

(continues on next page)

(continued from previous page)

```
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
```

(continues on next page)

(continued from previous page)

```
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
```

(continues on next page)

(continued from previous page)

```
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).
```

(continues on next page)

(continued from previous page)

```
  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
```

(continues on next page)

(continued from previous page)

```
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
```

(continues on next page)

(continued from previous page)

```
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
```

(continues on next page)

(continued from previous page)

```
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.
```

(continues on next page)

(continued from previous page)

```
  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.
```

(continues on next page)

(continued from previous page)

```
  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS
```

2.3 Contributors

- Marius Müller marius.mueller@tutanota.de
- Florian Dobener pyelli@schroedingerscat.org

2.4 Acknowledgements

- Olivier Castany and C. Molinaro Berreman4x4
- Solver2x2 based on Steve Byrnes' tmm
- Mikhail Polyanskiy's refractiveindex.info database and Pavel Dmitriev's pyTMM for his importer script for the database

2.5 Changelog

2.5.1 Version 0.22.6 - 2025-10-16

Added

- Improved Examples
- Updated refractiveindex.info database

2.5.2 Version 0.22.5 - 2025-09-23

Added

- Tests for windows

Fixed

- Encoding problems of refractiveindex.info files under windows

2.5.3 Version 0.22.4

Bugfixes

- Fix for an error when copying of an IndexDispersion created from a Dispersion
- Fix for reading of woollam data if no dpolE is present

2.5.4 Version 0.22.3

Bugfixes

- Pin Plotly to a version smaller than 6, to avoid problems with the fitting widget (#203).

2.5.5 Version 0.22.2

Bugfixes

- Fix publishing workflow

2.5.6 Version 0.22.1

New

- convert_delta_range function

Bugfixes

- Allow all valid Delta ranges in the Result class

2.5.7 Version 0.22.0

New

- Python 3.13 support
- Updated dependencies
- Updated refractive index database

Bugfixes

- SciPy performance regression fixed with Version 1.15

2.5.8 Version 0.21.2

Bugfixes

- Include the RII database submodule in the Pypi package again

2.5.9 Version 0.21.1

Bugfixes

- Fix documentation generation

2.5.10 Version 0.21.0

New

- Reenable pytorch 4x4solver
- Use OIDC for publishing
- Add custom fitting example
- Add chardet encoding detection
- Add reader for accurion data

Bugfixes

- Fix deprecation warnings

2.5.11 Version 0.20.0

New

- Vectorized KKR
- NumPy 2.0.0 support

Bug fixes

- Fix root selection mistakes for Bruggeman EMA

Known issues

- Performance regression for newer SciPy versions of expm solver (#178)

2.5.12 Version 0.19.0

New

- Python 3.12 Support
- Averaging of ResultLists
- Breaking change: Renamed RII.load_dispersion to RII.get_dispersion to be more consistent.

2.5.13 Version 0.18.1

Bug fixes

- Bump pygments from 2.13.0 to 2.15.0
- Bump tornado from 6.3.2 to 6.3.3
- Fixes wvase importer

2.5.14 Version 0.18.0

What's Changed

- Update spectraray importer
- Update RII database and use new file structure
- Bump scipy from 1.9.1 to 1.10.0
- Adds woollam importer

2.5.15 Version 0.17.0

New

- Support for NXopt based nexus definitions
- Interpolation order for refractiveindex.info database
- Speed up for formula dispersions

Bug fixes

- Better error messages
- Raise error when solver2x2 gets a negative k

2.5.16 Version 0.16.0

New

- Support for nexus dispersion files
- Formula based dispersions
- Cauchy Urbach dispersion
- Filtering by range for the refractive index database

Bug fixes

- Fixed a bug where incompatible dispersions were created in refractive index database
- Created separated index and dielectric dispersions
- Fixed checks for DispersionSums when dispersions are entered via *args

2.5.17 Version 0.15.1

New

- Support for Python 3.11

- Pseudo dielectric dispersion

Bug fixes

- Small fix in reading of rii dispersions
- Don't allow adding of refractive index based dispersions
- Don't allow adding of tabular dispersions

2.5.18 Version 0.15.0

New

- Include the Refractiveindex.info database and import and search scripts to access it
- Kramers-Kronig-Relationship conversions for dielectric functions
- Add Cody-Lorentz dispersion model

2.5.19 Version 0.14.1

Bug fixes:

- Installs the correct extra requirements in the docker container

2.5.20 Version 0.14.0

Breaking changes:

- Moved importers into the importers submodule
- Split the spectraray class into a class to load a dispersion table and an importer

Bug fixes:

- Introduced proper dependency management
- Changed the dependency of extra install requirements for fitting and testing

2.5.21 Version 0.13.0

Breaking Changes:

- Moved math submodule into utils, to avoid name conflict with Python's math module

Bug fixes:

- Fix error in eigenvalue sorting (PropergatorEig and non-isotropic backmaterials)
- Pin version of ipywidgets to keep plotly working

2.5.22 Version 0.12.0

Breaking Changes:

- Renamed the conversion functions and added more
- result.R and .T now return the reflectance/transmittance instead of the respective matrix, which can be accessed with .R_matrix/.T_matrix
- Renamed PropagatorExpmScipy to PropagatorExpm

- Removed Torch and Tensorflow Solvers

New:

- Added a lot of documentation
- Added a Bruggeman EMA Material
- Support for transmissive Ellipsometry

Bug fixes:

- Fix nan values in MaxwellGarnettEMA

2.5.23 Version 0.11.0

- Adds reading functionality for NeXus files.

2.5.24 Version 0.10.1

- The fitting module is not imported at top-level anymore. It has now to be imported by `elli.fitting`.

2.5.25 Version 0.10.0

- Dispersions are now addressed by their name only (instead of Dispersion...)
- Dispersions are initialized with two distinguished set of parameters for parameters which are set once and parameters which may be set multiple times (for oscillators etc). They can be added by invoking the `add` command on the respective class.
- There is a new factory class `DispersionFactory` to get a dispersion from it's string name, i.e. `DispersionFactory.get_dispersion(...)`

2.5.26 Version 0.9.2

- Fitting decorator buttons (fit, undo, redo)
- Data export for decorators
- Changed file loading to -pi, pi delta convention
- Fixed Jones vector default
- Included more documentation

2.5.27 Version 0.9.1

- Automated build of Docker images
- Benchmarking setup
- Added more documentation
- Renamed a lot of functions for PEP8 compliance
- More Bug fixes

2.5.28 Version 0.9.0

- Adapted examples to the new codebase
- Added a first batch of unit tests
- Reimplemented inhomogeneous and mixed materials
- Changed the sign convention of the E_rp vector
- Improve fitting decorators
- Added plotting and fitting for Mueller matrices
- Added type hints
- Various calculation Bug fixes

2.5.29 Version 0.1.0

- Initial Version after un-forking
- Complete rewrite of module structure
- Vectorization of calculations for massive speed-up
- Packaging as PyPi package
- Added additional dispersion models
- Added helper utilities for data-handling

PYTHON MODULE INDEX

e

INDEX

E

F

G

H

I

J

L

M

N

P

R

S

T

U

V