pytheas Documentation

Release 1.0.0

Benjamin Vial

Dec 16, 2018

CONTENTS

Pytheas is a Python package for creating, running and postprocessing electrodynamic simulations. It is based on open source software Gmsh for creating geometries and mesh generation, and GetDP for solving the underlying partial differential equations with the finite element method.

It features built in models of:

- periodic media in 2D and 3D with computation of diffraction efficiencies
- scattering analysis in 2D and 3D
- Bloch mode analysis of metamaterials
- treatment of open geometries with perfectly matched layers
- tools to define arbitrary permittivity distributions
- quasi-normal mode analysis
- two scale convergence homogenization
- tools for topology optimization in 2D
- built-in refractive index database

The complete project is documented for every submodule.

CHAPTER

ONE

PYTHEAS.PERIODIC2D: 2D METAMATERIALS

The *pytheas.periodic2D* module implements the resolution of the scalar wave equation for TE and TM polarization for mono-periodic stuctures in 2D:

- subject to an incident plane wave (diffraction problem) with calculation of the diffraction efficiencies, absorption and energy balance.
- eigenvalues and eigenmodes (modal analysis)

1.1 Classes

periodic2D.FemModel

CHAPTER

TWO

PYTHEAS.SCATT2D: 2D SCATTERING

The *pytheas.scatt2D* module implements the resolution of the scalar wave equation for TE and TM polarization in 2D:

- subject to an incident plane wave or line source (diffraction problem)
- eigenvalues and eigenmodes (modal analysis)

2.1 Classes

scatt2D.FemModel

CHAPTER

THREE

INDICES AND SEARCH

- genindex
- modindex
- search

CHAPTER

FOUR

EXAMPLES

4.1 Material examples

Examples to show how to retrieve complex refractive index from a database, generating material patterns.

Note: Click *here* to download the full example code

4.1.1 Importing refractive index from a database

Retrieve and plot the refractive index of a material in the refractiveindex.info data.

```
# Code source: Benjamin Vial
# License: MIT

import numpy as np
from pytheas import refractiveindex as ri
import matplotlib.pyplot as plt
```

We can get the refractive index from tabulated data or a formula using the database in the `pytheas.material` module. We will import the measured data from the reference Johnson and Christy [JC1972]. We first specify the file `yamlFile` we want to import:

```
yamlFile = "main/Au/Johnson.yml"
```

We then get the wavelength bounds from the data (in microns) and create a wavelength range to interpolate:

```
bounds = ri.getRange(yamlFile)
lambdas = np.linspace(bounds[0], bounds[1], 300)
```

Then get the refractive index data:

```
ncomplex = ri.get_complex_index(lambdas, yamlFile)
epsilon = ncomplex ** 2
```

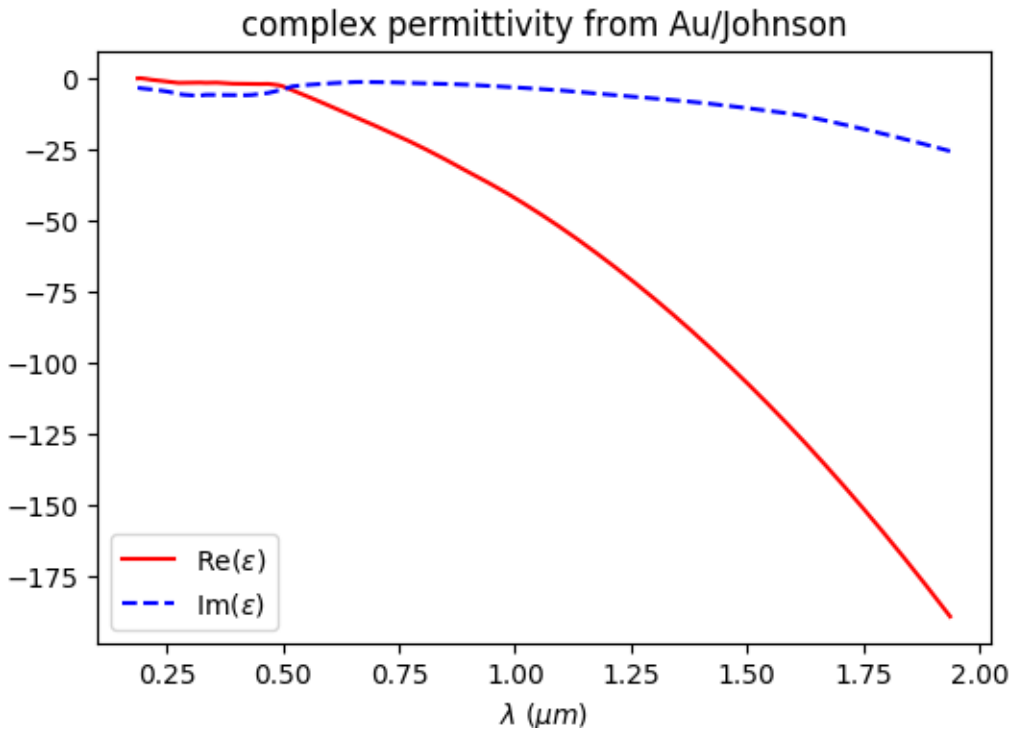
And finally plot it:

```
plt.close("all")
fig, ax = plt.subplots(1, figsize=(6, 4))
plt.plot(lambdas, epsilon.real, "r-", label=r"Re($\varepsilon$)")
plt.plot(lambdas, epsilon.imag, "b--", label=r"Im($\varepsilon$)")
```

(continues on next page)

(continued from previous page)

```
plt.xlabel(r"$\lambda$ ($\mu m$)")
plt.title("complex permittivity from " + yamlFile[5:][:-4])
plt.legend(loc=0)
plt.show()
```



Total running time of the script: (0 minutes 0.048 seconds)

4.2 Periodic 2D examples

Examples to show how to simulate a mono periodic medium (metamaterial) with the finite element method and post-processing the results (fields maps and diffraction efficiencies).

Note: Click *here* to download the full example code

4.2.1 Simulating diffraction by a 2D metamaterial

Finite element simulation of the diffraction of a plane wave a mono-periodic grating and calculation of diffraction efficiencies.

First we import the required modules and class

```
# Code source: Benjamin Vial
# License: MIT
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import matplotlib.pyplot as plt
from pytheas import genmat
from pytheas import Periodic2D
```

Then we need to instanciate the class `FemModel`:

```
fem = Periodic2D()
```

The model consist of a single unit cell with quasi-periodic boundary conditions in the x direction enclosed with perfectly matched layers (PMLs) in the y direction to truncate the semi infinite media. From top to bottom:

- PML top
- superstrate (incident medium)
- layer 1
- design layer: this is the layer containing the periodic pattern, can be continuous or discrete
- layer 2
- substrate
- PML bottom

We define here the opto-geometric parameters:

```
# opto-geometric parameters  ------------------------------------------
mum = 1e-6  #: flt: the scale of the problem (here micrometers)
fem.d = 0.4 * mum  #: flt: period
fem.h_sup = 1.0 * mum  #: flt: "thickness" superstrate
fem.h_sub = 1.0 * mum  #: flt: "thickness" substrate
fem.h_layer1 = 0.1 * mum  #: flt: thickness layer 1
fem.h_layer2 = 0.1 * mum  #: flt: thickness layer 2
fem.h_des = 0.4 * mum  #: flt: thickness layer design
fem.h_pmltop = 1.0 * mum  #: flt: thickness pml top
fem.h_pmlbot = 1.0 * mum  #: flt: thickness pml bot
fem.a_pml = 1  #: flt: PMLs parameter, real part
fem.b_pml = 1  #: flt: PMLs parameter, imaginary part
fem.eps_sup = 1  #: flt: permittivity superstrate
fem.eps_sub = 11  #: flt: permittivity substrate
fem.eps_layer1 = 1  #: flt: permittivity layer 1
fem.eps_layer2 = 1  #: flt: permittivity layer 2
fem.eps_des = 1  #: flt: permittivity layer design
fem.lambda0 = 0.6 * mum  #: flt: incident wavelength
fem.theta_deg = 0.0  #: flt: incident angle
fem.pola = "TE"  #: str: polarization (TE or TM)
fem.lambda_mesh = 0.6 * mum  #: flt: incident wavelength
#: mesh parameters, correspond to a mesh size of lambda_mesh/(n*parmesh),
#: where n is the refractive index of the medium
fem.parmesh_des = 15
fem.parmesh = 13
fem.parmesh_pml = fem.parmesh * 2 / 3
fem.type_des = "elements"
```

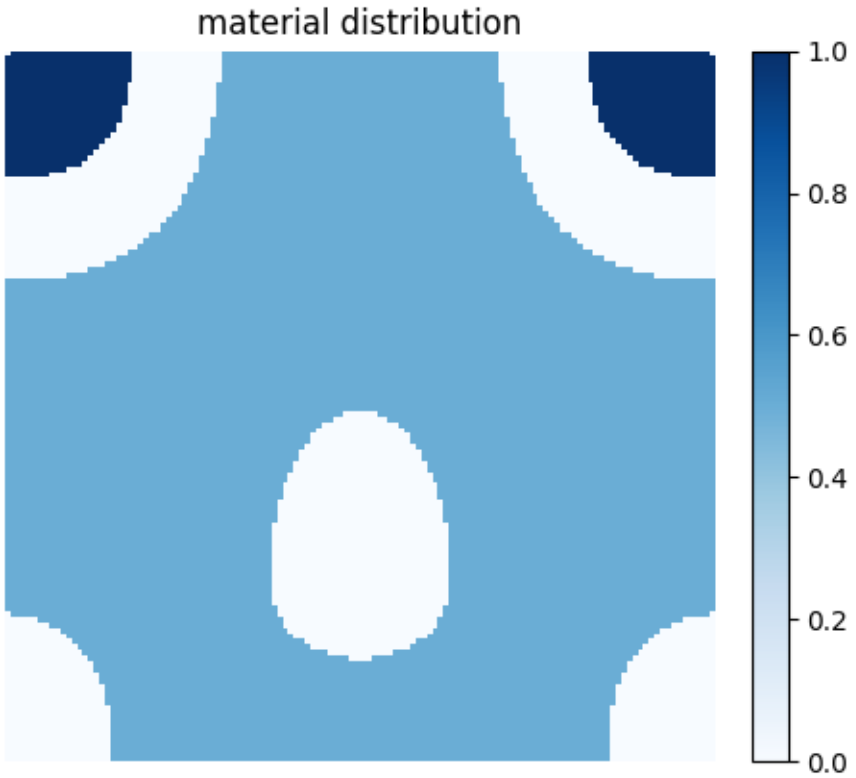
We then initialize the model (copying files, etc) and mesh the unit cell using gmsh

```
fem.getdp_verbose = 0
fem.gmsh_verbose = 0

fem.initialize()
mesh = fem.make_mesh()
```

We use the `genmat` module to generate a material pattern

```
genmat.np.random.seed(100)
mat = genmat.MaterialDensity()  # instanciate
mat.n_x, mat.n_y, mat.n_z = 2 ** 7, 2 ** 7, 1  # sizes
mat.xsym = True  # symmetric with respect to x?
mat.p_seed = mat.mat_rand  # fix the pattern random seed
mat.nb_threshold = 3  # number of materials
mat._threshold_val = np.random.permutation(mat.threshold_val)
mat.pattern = mat.discrete_pattern
fig, ax = plt.subplots()
mat.plot_pattern(fig, ax)
```



We now assign the permittivity

```
fem.register_pattern(mat.pattern, mat._threshold_val)
fem.matprop_pattern = [1.4, 4 - 0.02 * 1j, 2]  # refractive index values
```

Now were ready to compute the solution:

```
fem.compute_solution()
```

Finally we compute the diffraction efficiencies, absorption and energy balance

```
effs_TE = fem.diffraction_efficiencies()
print("efficiencies TE", effs_TE)
```

Out:

```
efficiencies TE {'R': 0.5416794889618843, 'T': 0.3545625265103274, 'Q': 0.
↪1158907418802392, 'B': 1.012132757352451}
```

It is fairly easy to switch to TM polarization:

```
fem.pola = "TM"
fem.compute_solution()
effs_TM = fem.diffraction_efficiencies()
print("efficiencies TM", effs_TM)
```

Out:

```
efficiencies TM {'R': 0.4440749322167822, 'T': 0.4804765728980809, 'Q': 0.
↪05938397641376305, 'B': 0.9839354815286261}
```

Total running time of the script: (0 minutes 3.876 seconds)

BIBLIOGRAPHY

[JC1972] (**P. B. Johnson and R. W. Christy. Optical constants of the noble** metals, Phys. Rev. B 6, 4370-4379 (1972)).

PYTHON MODULE INDEX

p