pytheas Documentation

Release 0.1.1

Benjamin Vial

Nov 22, 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.

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) and calculation of the diffraction efficiencies, absorption and energy balance.

- eigenvalues and eigenmodes (modal analysis)

1.1 Classes

periodic2D.FemModel([analysis, pola, A,])	A class for a finite element model of a 2D mono-periodic medium.

1.1.1 pytheas.periodic2D.FemModel

class pytheas.periodic2D.**FemModel**(*analysis='diffraction', pola='TE', A=1, lambda0=1, lambda_mesh=1, theta_deg=0, d=0.8, h_sup=1, h_sub=1, h_layer1=0.1, h_layer2=0.1, h_des=1.0, h_pmltop=1.0, h_pmlbot=1.0, a_pml=1, b_pml=1, eps_sup=(1+0j), eps_sub=(1+0j), eps_layer1=(1+0j), eps_layer2=(1+0j), eps_des=(1+0j), eps_incl=(1+0j), mu_incl=(1+0j), mu_des=(1+0j)*)

A class for a finite element model of a 2D mono-periodic medium.

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 2

- design layer: this is the layer containing the periodic pattern, can be continuous or discrete

- layer 1

- substrate

- PML bottom

Parameters

- **analysis** (*str, default "diffraction"*) – Analysis type: either diffraction (plane wave) or modal (spectral problem)
- **pola** (*str, default "TE"*) – Polarization case: either TE (E along z) or TM (H along z)
- **A** (*float, default 1*) – Incident plane wave amplitude
- **lambda0** (*float, default 1*) – Incident plane wave wavelength in free space
- **lambda_mesh** (*float, default 1*) – Wavelength to use for meshing
- **theta_deg** (*float, default 0*) – Incident plane wave angle (in degrees). Light comes from the top (travels along -y if normal incidence, `theta_deg=0` is set)
- **d** (*float, default 0.8*) – Periodicity
- **h_sup** (*float, default 1*) – Thickness superstrate
- **h_sub** (*float, default 1*) – Thickness substrate
- **h_layer1** (*float, default 0.1*) – Thickness layer 1
- **h_layer2** (*float, default 0.1*) – Thickness layer 2
- **h_des** (*float, default 1*) – Thickness layer design
- **h_pmltop** (*float, default 1*) – Thickness pml top
- **h_pmlbot** (*float, default 1*) – Thickness pml bot
- **a_pml** (*float, default 1*) – PMLs complex y-stretching parameter, real part
- **b_pml** (*float, default 1*) – PMLs complex y-stretching parameter, imaginary part
- **eps_sup** (*complex, default (1 - 0 * 1j)*) – Permittivity superstrate
- **eps_sub** (*complex, default (1 - 0 * 1j)*) – Permittivity substrate
- **eps_layer1** (*complex, default (1 - 0 * 1j)*) – Permittivity layer 1
- **eps_layer2** (*complex, default (1 - 0 * 1j)*) – Permittivity layer 2
- **eps_des** (*complex, default (1 - 0 * 1j)*) – Permittivity layer design
- **eps_incl** (*complex, default (1 - 0 * 1j)*) – Permittivity inclusion

cleanup()
  Clean gmsh/getdp generated files

diffraction_efficiencies()
  Postprocess diffraction efficiencies

get_field_map(*name*)
  Retrieve a field map.

  Parameters name (*str {'u', 'u_tot'}*) – u (scattered field), u_tot (total field)

  Returns field

  Return type array, shape (self.Nix, self.Niy)

postpro_absorption()
  Compute the absorption coefficient

  Returns Q – Absorption coefficient

  Return type float

postpro_fields (*filetype='txt'*)
Compute the field maps and output to a file.

> **Parameters filetype** (`str, default "txt"`) – Type of output files. Either txt (to be read by the method get_field_map in python) or pos to be read by gmsh/getdp.

postpro_fields_cuts ()
Compute the field cuts in substrate and superstarte

> **Returns**
> - **u_diff_t** (*array-like*) – Transmitted field cuts
> - **u_diff_r** (*array-like*) – Reflected field cuts

Examples using `pytheas.periodic2D.FemModel`

- *Simulating diffraction by a 2D metamaterial*

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()	A class for a finite element model of a 2D medium

2.1.1 pytheas.scatt2D.FemModel

class pytheas.scatt2D.**FemModel**
　　A class for a finite element model of a 2D medium

　　A = None
　　　　incident plane wave amplitude

　　　　　　Type flt

　　Ni_theta = None
　　　　number of theta points for computing the angular dependance of the modal coupling coefficients

　　　　　　Type int

　　Nibox_x = None
　　　　number of x interpolation points on the design box

　　　　　　Type int

　　Nibox_y = None
　　　　number of y interpolation points on the design box

　　　　　　Type int

　　Nin2f_x = None
　　　　number of x interpolation points for near to far field calculations

　　　　　　Type int

　　Nin2f_y = None
　　　　number of y interpolation points for near to far field calculations

　　　　　　Type int

Nix = None
> number of x points for postprocessing field maps
>
> > **Type** int

a_pml = None
> PMLs parameter, real part
>
> > **Type** flt

analysis = None
> analysys type (either diffraction or modal)
>
> > **Type** str

b_pml = None
> PMLs parameter, imaginary part
>
> > **Type** flt

beam_flag = None
> beam?

cleanup()
> Clean gmsh/getdp generated files

dom_des = None
> design domain number (check .geo/.pro files)

eps_des = None
> permittivity scattering box
>
> > **Type** flt

eps_host = None
> permittivity host
>
> > **Type** flt

eps_incl = None
> permittivity inclusion
>
> > **Type** flt

eps_sub = None
> permittivity substrate
>
> > **Type** flt

h_pml = None
> thickness pml
>
> > **Type** flt

hx_des = None
> x - thickness scattering box (design)
>
> > **Type** flt

hy_des = None
> y - thickness scattering box
>
> > **Type** flt

lambda0 = None
> incident plane wave wavelength in free space

> **Type** flt

lambda0search = None
> wavelength around which to search eigenvalues
>
> > **Type** flt

lambda_mesh = None
> wavelength to use for meshing
>
> > **Type** flt

ls_flag = None
> line source position

nb_slice = None
> number of y slices points for postprocessing diffraction efficiencies
>
> > **Type** int

neig = None
> number of eigenvalues searched for in modal analysis
>
> > **Type** int

pola = None
> polarisation of the incident plane wave (either TE or TM)
>
> > **Type** str

scan_dist_ratio = None
> such that *scan_dist = min(h_sup, hsub)/scan_dist_ratio*
>
> > **Type** flt

theta_deg = None
> incident plane wave angle (in degrees). Light comes from the top (travels along -y if normal incidence, *theta_deg=0* is set)
>
> > **Type** flt

xpp = None
> coords of point for PostProcessing

ypp = None
> coords of point for PostProcessing

INDICES AND SEARCH

- genindex
- modindex
- search

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.

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

from pytheas.material.refractiveindex import *
from pytheas.tools.plottools import *
```

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:

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

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

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

Then get the refractive index data:

```python
ncomplex = get_complex_index(lambdas, yamlFile)
epsilon = (ncomplex**2)
```
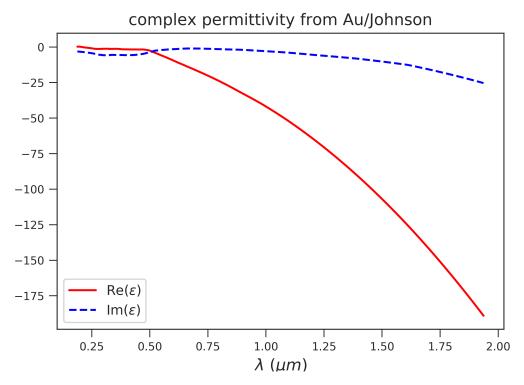
And finally plot it:

```python
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($\varepsilon$)')
plt.xlabel(r'$\lambda$ ($\mu m$)')
```

```
plt.title("complex permittivity from " + yamlFile[5:][:-4])
plt.legend(loc=0)
plt.show()
```

complex permittivity from Au/Johnson

Total running time of the script: (0 minutes 2.126 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 `femmodel` module and some utility functions:

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

import numpy as np
```

```python
from pytheas.tools.plottools import *
from pytheas.material import genmat

from pytheas import periodic2D
from pytheas.periodic2D import FemModel, utils
```

Then we need to instanciate the class `FemModel`:

```python
fem = FemModel()
```

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:

```python
# 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. * mum  #: flt: "thickness" superstrate
fem.h_sub = 1. * 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. * mum  #: flt: thickness pml top
fem.h_pmlbot = 1. * 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.   #: 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

```python
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
matprop = [1.4, 4 - 0.02 * 1j, 2]  # refractive index values

mat._threshold_val = np.random.permutation(mat.threshold_val)
mat.pattern = mat.discrete_pattern
fig, ax = plt.subplots()
mat.plot_pattern(fig, ax, cmap=cmap)
```


material distribution

We now assign the permittivity

```python
fem.register_pattern(mat.pattern, mat._threshold_val)
fem.matprop_pattern = matprop
```

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 5.470 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