pytheas Documentation

Release 1.1.1

Benjamin Vial

Jun 03, 2019

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([analysis, pola, A, lambda0,])	A class for a finite element model of a 2D mono-periodic medium.

1.1.1 pytheas.Periodic2D

class pytheas.**Periodic2D**(*analysis='direct'*, *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 "direct"*) – Analysis type: either direct (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()
Remove gmsh/getdp/python generated files from the temporary folder

compute_solution(*res_list=None*, ***kwargs*)
Compute the solution of the FEM problem using getdp

diffraction_efficiencies(*cplx_effs=False*, *orders=False*)
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)

get_qty(*filename*)
Retrieve a scalar quantity.

Parameters **filename** (*str*) – Name of the txt file to load.

Returns **qty** – The quantity to be loaded.

Return type array

initialize()
Initialize the problem: - make dictionary of parameters - write this dictionary entries to a .dat file - copy the .dat, .geo and .pro files to the temporary folder

make_param_dict()
Build dictionary of parameters. This will be later written to a parameter.dat file that is meant to be read by both gmsh and getdp

mk_tmp_dir()
Create a temporary directory

postpro_absorption()
Compute the absorption coefficient

Returns **Q** – Absorption coefficient

Return type float

postpro_choice(*name*, *filetype*)
Run a postprocessing command with either pos or txt file output.

Parameters

- **name** (*str*) – Name of the post operation as defined in the .pro file.
- **filetype** (*str*) – File type to use (pos or txt)

postpro_fields(*filetype='txt'*, *postop='postop_fields'*)
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.
- **postop** (*str, default "postop_fields"*) – Name of the postoperation

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

postprocess(*postop*)
Run getdp postoperation.

Parameters **postop** (*str*) – Name of the postoperation to run.

ppcmd(*postop*)
Create a postprocessing command

Parameters **postop** (*str*) – Name of the post operation as defined in the .pro file.

rm_tmp_dir()
Remove the temporary directory

update_params()
Update the dictionary of parameters and the corresponding file

Examples using `pytheas.Periodic2D`

- *Simulating diffraction by a 2D metamaterial*

CHAPTER

TWO

PYTHEAS.TOOLS: TOOLS AND UTILITIES

Input/output and utilities.

2.1 Submodules

femio	Tools for gmsh/getdp control and input/output.
utils	Shared utility functions used in pytheas.

2.1.1 pytheas.tools.femio

Tools for gmsh/getdp control and input/output.

pytheas.tools.femio.**mesh_model**(*path_mesh*, *path_geo*, *mesh_format='msh2'*, *dim=None*, *verbose=0*, *other_option=''*)

Mesh the model using Gmsh

pytheas.tools.femio.**postpro_commands**(*postop*, *path_pro*, *path_mesh*, *path_pos=None*, *verbose=0*)

Generate a command list for postprocessing by GetDP (see main.pro file in ./base folder for default available postprocessings, or to add your own)

Parameters

- **postop** (*str*) – The name of the postoperation to perform.
- **path_pro** (*str*) – Path to the .pro file
- **path_mesh** (*str*) – Path to the .msh file
- **path_pos** (*str , optional*) – Path to a file to be read by `gmshread`.
- **verbose** (*int*) – verbosity level
- **to None.** (*Defaults*) –

Returns The list of strings to be oscommanded.

Return type list

2.1.2 pytheas.tools.utils

Shared utility functions used in pytheas.

pytheas.tools.utils.**normalize**(*x*)

Normalize an array between 0 and 1

Parameters **x** (*array-like*) – the quantity to be normalized

Returns **x_norm** – normalized array

Return type array-like

CHAPTER

THREE

PYTHEAS.BASEFEM: BOILERPLATE CLASS FOR FEM MODELS

The *pytheas.basefem* module implements a base class common for FEM models using Gmsh and GetDP. This should be used as a parent class and not directly.

3.1 Classes

BaseFEM()	Base class for Finite Element models

3.1.1 pytheas.BaseFEM

class pytheas.**BaseFEM**

Base class for Finite Element models

Nix = None

number of x points for postprocessing field maps

Type int

bg_mesh_filename_ = None

Gmsh geo filename for background mesh

Type str

cel = 299792458.0105029

speed of light in vacuum

Type flt

cleanup()

Remove gmsh/getdp/python generated files from the temporary folder

compute_solution(*res_list=None*, ***kwargs*)

Compute the solution of the FEM problem using getdp

dim = None

dimension of the problem

epsilon0 = 8.854187817e-12

vacuum permittivity

Type flt

geom_filename_ = None

Gmsh geometry filename

Type str

get_qty(*filename*)
Retrieve a scalar quantity.

Parameters **filename** (*str*) – Name of the txt file to load.

Returns **qty** – The quantity to be loaded.

Return type array

getdp_verbose = None
GetDP verbose (int between 0 and 4)

Type str

gmsh_verbose = None
Gmsh verbose (int between 0 and 4)

Type str

initialize()
Initialize the problem: - make dictionary of parameters - write this dictionary entries to a .dat file - copy the .dat, .geo and .pro files to the temporary folder

make_param_dict()
Build dictionary of parameters. This will be later written to a parameter.dat file that is meant to be read by both gmsh and getdp

mk_tmp_dir()
Create a temporary directory

mu0 = 1.2566370614359173e-06
vacuum permeability

Type flt

param_filename_ = None
GetDP pro filename

Type str

parmesh = None
global mesh parameter *MeshElementSize = lambda0/(parmesh*n) n*: refractive index

Type flt

parmesh_des = None
design subdomain mesh parameter

Type flt

parmesh_pml = None
PMLs mesh parameter

Type flt

postpro_choice(*name*, *filetype*)
Run a postprocessing command with either pos or txt file output.

Parameters

- **name** (*str*) – Name of the post operation as defined in the .pro file.
- **filetype** (*str*) – File type to use (pos or txt)

postpro_fields(*filetype='txt'*, *postop='postop_fields'*)
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.
- **postop** (*str, default "postop_fields"*) – Name of the postoperation

postprocess(*postop*)
Run getdp postoperation.

Parameters **postop** (*str*) – Name of the postoperation to run.

ppcmd(*postop*)
Create a postprocessing command

Parameters **postop** (*str*) – Name of the post operation as defined in the .pro file.

pro_filename_ = None
GetDP pro filename

Type str

python_verbose = None
python verbose (int between 0 and 1)

Type str

rm_tmp_dir()
Remove the temporary directory

update_params()
Update the dictionary of parameters and the corresponding file

CHAPTER

FOUR

INDICES AND SEARCH

- genindex
- modindex
- search

CHAPTER

FIVE

EXAMPLES

5.1 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

5.1.1 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.get_wl_range(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
```

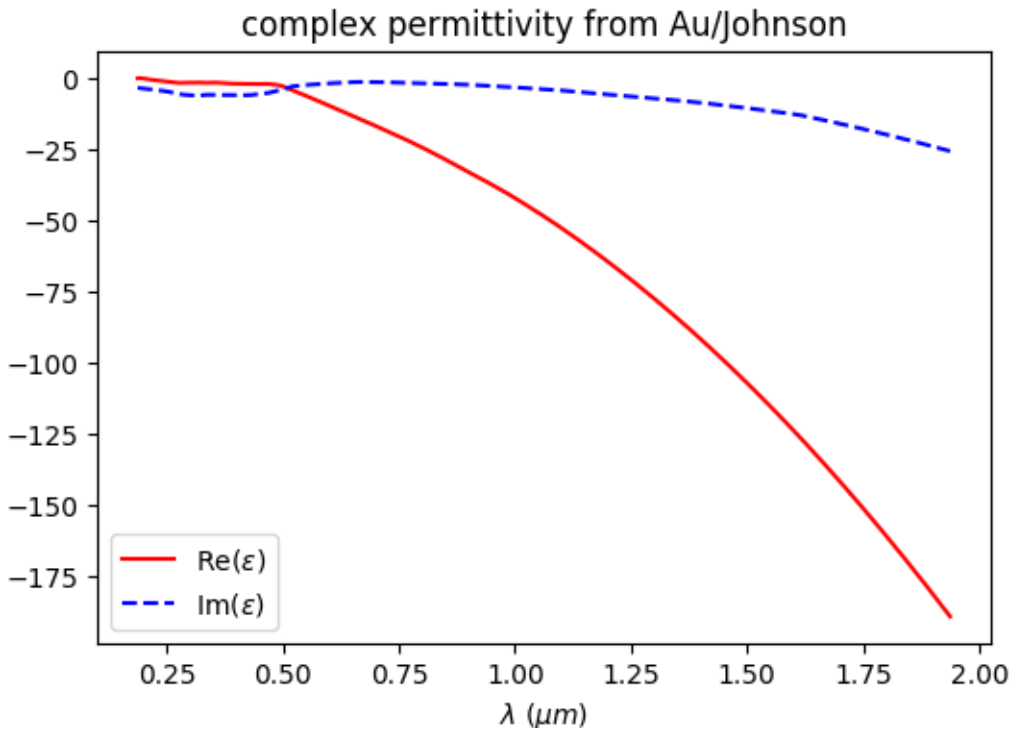
And finally plot it:

```
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 0.277 seconds)

5.2 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

5.2.1 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

import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
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 = 3  #: 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
```

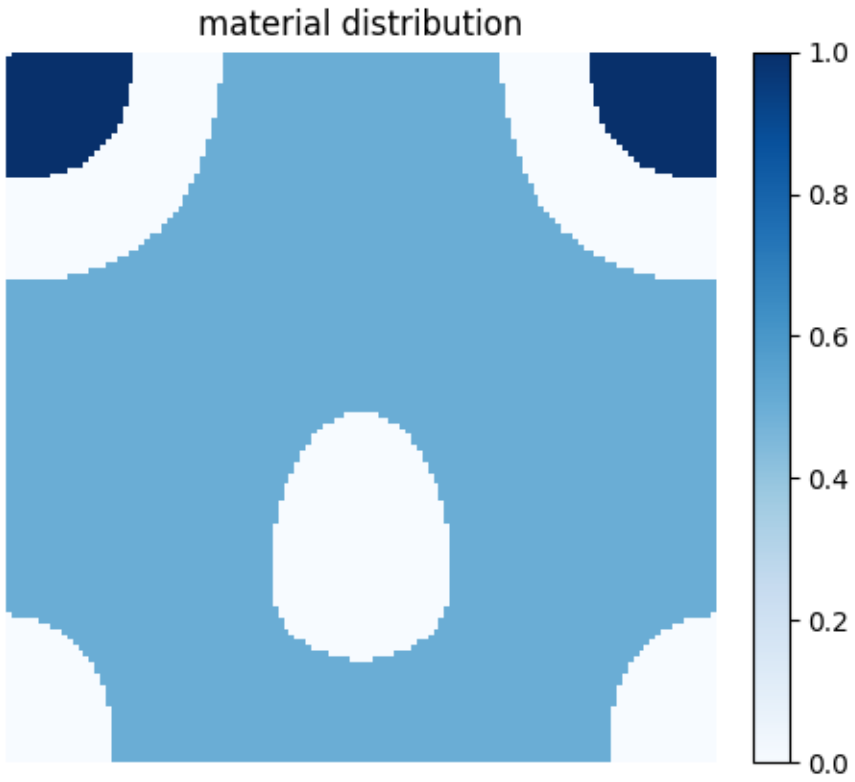
(continues on next page)

(continued from previous page)

```
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': 0.359684236393363, 'T': 0.5279938129207201, 'Q': 0.
↪1128454301571147, 'B': 1.000523479471198}
```

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': 0.20523268509785825, 'T': 0.7489741597039322, 'Q': 0.
↪04426949819770286, 'B': 0.9984763429994933}
```

Total running time of the script: (0 minutes 3.120 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