

---

# pytheas Documentation

*Release 0.1.2*

**Benjamin Vial**

**Dec 14, 2018**



# CONTENTS

<b>1</b>	<b>pytheas.periodic2D: 2D metamaterials</b>	<b>3</b>
1.1	Classes . . . . .	3
<b>2</b>	<b>pytheas.scatt2D: 2D scattering</b>	<b>7</b>
2.1	Classes . . . . .	7
<b>3</b>	<b>Indices and search</b>	<b>11</b>
<b>4</b>	<b>Examples</b>	<b>13</b>
4.1	Material examples . . . . .	13
4.2	Periodic 2D examples . . . . .	14
	<b>Bibliography</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>



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 structures 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

---

<code>periodic2D.FemModel([analysis, pola, A, ])</code>	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** ()

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

**compute\_solution** (\*\*kwargs)

Compute the solution of the FEM problem using getdp

**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)

**get\_qty** (filename)

Run a postprocessing command with either pos or txt file output.



**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')

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 superstrate

**Returns**

- **u\_diff\_t** (*array-like*) – Transmitted field cuts
- **u\_diff\_r** (*array-like*) – Reflected field cuts

**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.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

---

<code>scatt2D.FemModel()</code>	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()**  
Remove gmsh/getdp/python generated files from the temporary folder

**compute\_solution(\*\*kwargs)**  
Compute the solution of the FEM problem using getdp

**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`

**get\_qty(filename)**  
Run a postprocessing command with either pos or txt file output.  
**Parameters** **filename** (`str`) – Name of the txt file to load.  
**Returns** **qty** – The quantity to be loaded.  
**Return type** `array`

**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`

**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

**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

**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

**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`

**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)

**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

**scan\_dist\_ratio** = **None**

such that  $scan\_dist = \min(h\_sup, h\_sub)/scan\_dist\_ratio$

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`

**update\_params** ()

Update the dictionary of parameters and the corresponding file

**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.

```
# 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:

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

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

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

Then get the refractive index data:

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

(continues on next page)

(continued from previous page)

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



Total running time of the script: ( 0 minutes 2.047 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
```

(continues on next page)

(continued from previous page)

```
from pytheas.tools.plottools import *
from pytheas.material import genmat
from pytheas.periodic2D import FemModel
```

Then we need to instantiate the class FemModel:

```
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:

```
# 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
```

(continues on next page)

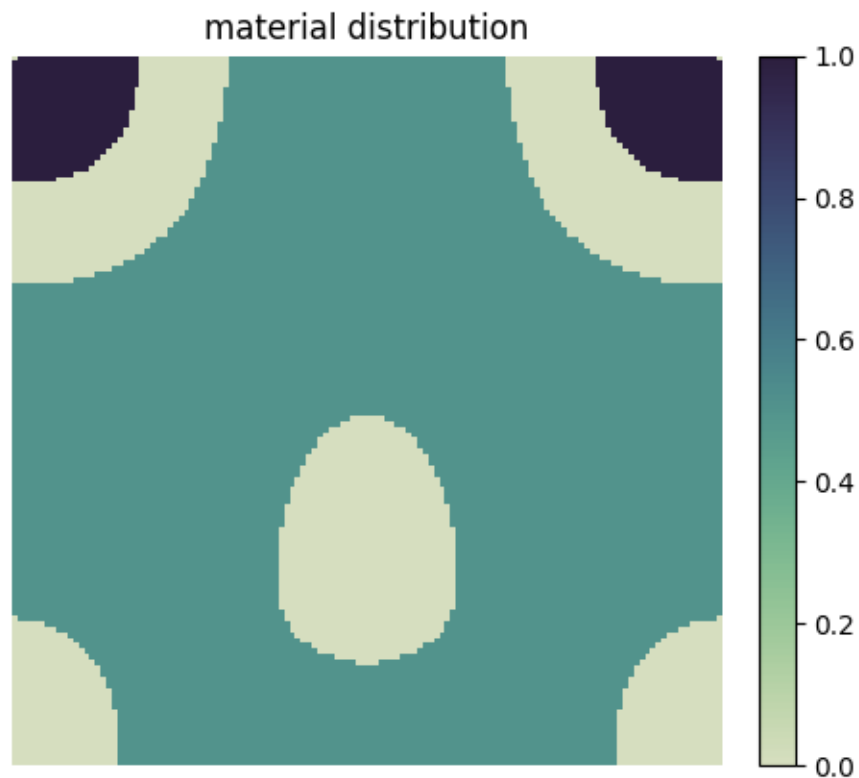
(continued from previous page)

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

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

```
genmat.np.random.seed(100)
mat = genmat.MaterialDensity() # instantiate
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)
```



We now assign the permittivity

```
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 5.590 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

`pytheas.periodic2D`, [3](#)

`pytheas.scatt2D`, [7](#)