
pytheas Documentation

Release 1.1.1

Benjamin Vial

Nov 30, 2019

CONTENTS

1	User guide	3
1.1	<code>pytheas.periodic2D</code> : 2D metamaterials	3
1.2	<code>pytheas.scatt2D</code> : 2D scattering	6
1.3	<code>pytheas.tools</code> : tools and utilities	9
2	Appendix	11
2.1	Index	11
2.2	Modules index	11
2.3	Search page	11
3	Examples	13
3.1	Material examples	13
3.2	Periodic 2D examples	14
3.3	Scattering 2D examples	17
	Bibliography	21
	Python Module Index	23

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 easiest way to get started is to install via [PyPi](#):

```
pip install pytheas-pip
```


USER GUIDE

The complete project is documented for every submodule.

1.1 `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) with calculation of the diffraction efficiencies, absorption and energy balance.
- eigenvalues and eigenmodes (modal analysis)

1.1.1 Classes

<code>Periodic2D([analysis, pola, A, lambda0,])</code>	A class for a finite element model of a 2D mono-periodic medium.
---	--

`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()

Compute the solution of the FEM problem using getdp

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

Postprocess diffraction efficiencies.

Parameters

- **cplx_effs** (*bool*) – If *True*, return complex coefficients (amplitude reflection and transmission). If *False*, return real coefficients (power reflection and transmission)
- **orders** (*bool*) – If *True*, computes the transmission and reflection for all the propagating diffraction orders. If *False*, returns the sum of all the propagating diffraction orders.

Returns A dictionary containing the diffraction efficiencies.

Return type `dict`

get_field_map (*name*)

Retrieve a field map.

Parameters *name* (`str`) – Choose between `u` (scattered field), `u_tot` (total field)

Returns A 2D complex array of shape (N_{ix}, N_{iy})

Return type `array`

initialize ()

Initialize the problem parameters.

make_inclusion (*points*, *lcar*='lc_incl', ***kwargs*)

Make a diffractive element geometry from points.

Parameters

- **points** (*array of size* $(N_{points}, 2)$) – The points defining the simply connected 2D geometry of the object.
- **lcar** (`str` (*default* `"lc_incl"`)) – Characteristic length for the mesh.
- ****kwargs** (`dict`) – Extra arguments.

make_mesh (*other_option*=None)

Mesh the geometry using gmsh.

Parameters *other_option* (`str`) – Extra flag to pass to gmsh.

Returns The content of the .msh file.

Return type `str`

mk_tmp_dir ()

Create a temporary directory

open_gmsh_gui (*pos_list*=None)

Open gmsh GUI to visualize geometry and postprocessing results.

Parameters *pos_list* (`list`) – A list of .pos files giving the views to load. By default it will render all the generated views.

postpro_absorption ()

Compute the absorption coefficient

Returns *Q* – Absorption coefficient

Return type `float`

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.

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*
-

1.2 `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)

1.2.1 Classes

<code>Scatt2D()</code>	A class for a finite element model of a 2D medium
------------------------	---

`pytheas.Scatt2D`

class `pytheas.Scatt2D`
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 direct 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 (*res_list=None*)
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`

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 parameters.

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_inclusion (*points*, *lcar*='lc_incl', ***kwargs*)
Make a diffractive element geometry from points.

Parameters

- **points** (*array of size (Npoints, 2)*) – The points defining the simply connected 2D geometry of the object.
- **lcar** (*str (default "lc_incl")*) – Characteristic length for the mesh.
- ****kwargs** (*dict*) – Extra arguments.

make_mesh (*other_option=None*)
Mesh the geometry using gmsh.

Parameters **other_option** (*str*) – Extra flag to pass to gmsh.

Returns The content of the .msh file.

Return type *str*

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*

open_gmsh_gui (*pos_list=None*)
Open gmsh GUI to visualize geometry and postprocessing results.

Parameters **pos_list** (*list*) – A list of .pos files giving the views to load. By default it will render all the generated views.

pola = None

polarisation of the incident plane wave (either TE or TM)

Type `str`

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.

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

Examples using `pytheas.Scatt2D`

- *Simulating diffraction by an object in 2D*

1.3 `pytheas.tools`: tools and utilities

Input/output and utilities.

1.3.1 Submodules

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

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](#)

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](#)

APPENDIX

2.1 Index

2.2 Modules index

2.3 Search page

EXAMPLES

3.1 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

3.1.1 Importing refractive index from a database

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

```
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 `ymlFile` we want to import:

```
ymlFile = "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(ymlFile)
print(bounds[0], bounds[1])
lambdas = np.linspace(0.4, 0.8, 300)
```

Out:

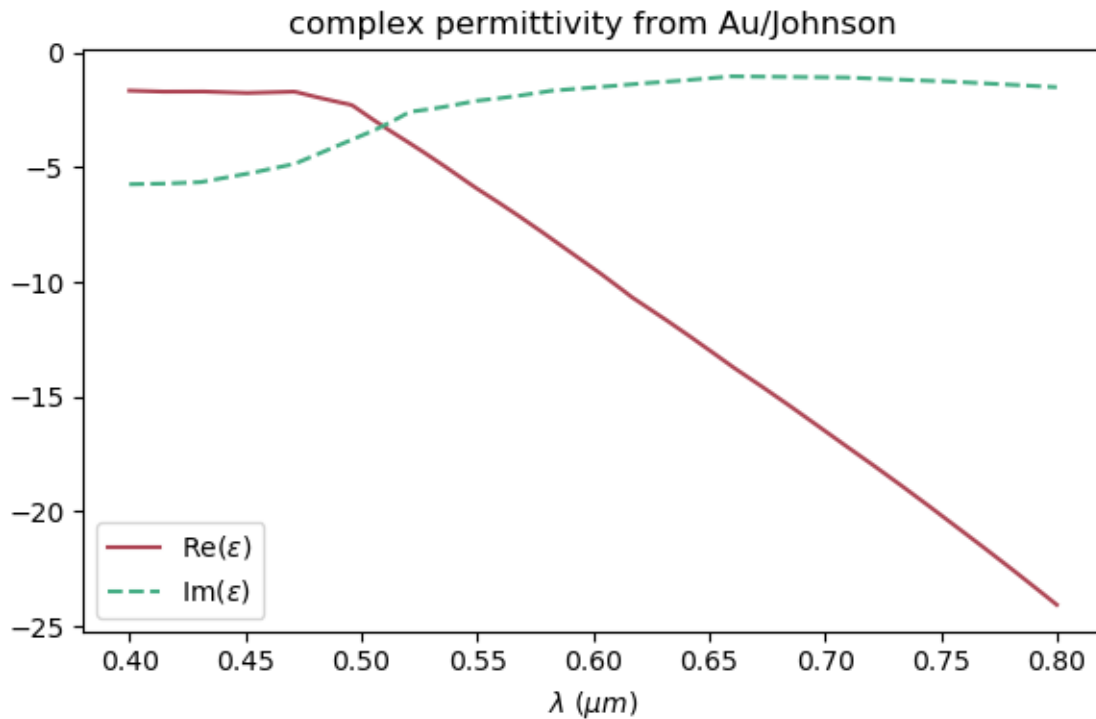
```
0.1879 1.937
```

Then get the refractive index data:

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

And finally plot it:

```
fig, ax = plt.subplots(1, figsize=(6, 4))
ax.plot(lambdas, epsilon.real, "-", c="#ad4453", label=r"Re($\varepsilon$)")
ax.plot(lambdas, epsilon.imag, "--", c="#44ad84", label=r"Im($\varepsilon$)")
ax.set_xlabel(r"$\lambda$ ($\mu$ m)")
ax.set_title("complex permittivity from " + yamlFile[5][:4])
ax.legend(loc=0)
plt.tight_layout()
```



Total running time of the script: (0 minutes 0.250 seconds)

3.2 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

3.2.1 Simulating diffraction by a 2D metamaterial

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

First we import the required modules and class

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

Then we need to instantiate 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
```

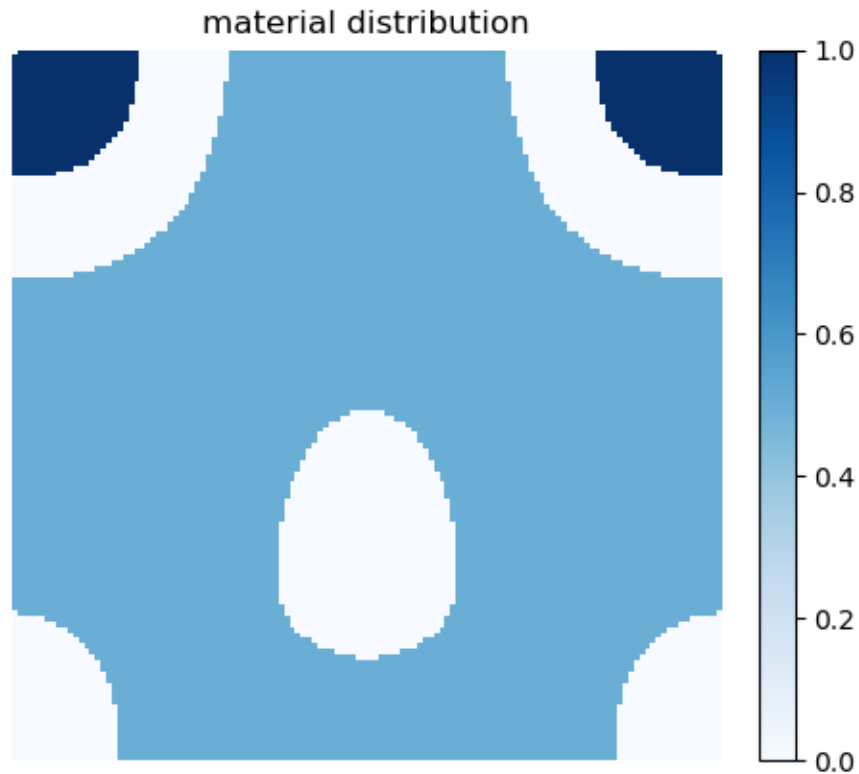
(continues on next page)

(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() # 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
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': 0.42749531344001657, 'T': 0.45592852708133014, 'Q': 0.
↪1177478267191832, 'B': 1.00117166724053}
```

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': 0.2052478291947634, 'T': 0.7359213200135426, 'Q': 0.
↪05719724751542301, 'B': 0.998366396723729}
```

Total running time of the script: (0 minutes 2.367 seconds)

3.3 Scattering 2D examples

Examples to show how to simulate a the 2D scattering off an object subject to a plane wave or line source harmonic excitation.

Note: Click [here](#) to download the full example code

3.3.1 Simulating diffraction by an object in 2D

Finite element simulation of the diffraction by an object illuminated by a plane wave or a line source. Calculation of scattering width and getting the field maps.

```
import numpy as np
import matplotlib.pyplot as plt
from pytheas import Scatt2D

plt.ion()

pi = np.pi
```

Then we need to instantiate the class Scatt2D:

```
fem = Scatt2D()
fem.rm_tmp_dir()
```

```
# We define here the opto-geometric parameters:

# opto-geometric parameters -----
mum = 1  #: flt: the scale of the problem (here micrometers)
fem.lambda0 = 0.6 * mum  #: flt: incident wavelength
fem.pola = "TE"  #: str: polarization (TE or TM)
fem.theta_deg = 30.0  # 0: coming from top (y>0)
fem.hx_des = 1.0 * mum  #: flt: x thickness box
fem.hy_des = 1.0 * mum  #: flt: y thickness box
fem.h_pml = fem.lambda0  #: flt: thickness pml
fem.space2pml_L, fem.space2pml_R = fem.lambda0 * 2, fem.lambda0 * 2
fem.space2pml_T, fem.space2pml_B = fem.lambda0 * 2, fem.lambda0 * 2
fem.eps_des = 1  #: flt: permittivity design box
fem.eps_host = 1.0
fem.eps_incl = 11.0 - 1e-2 * 1j
#: mesh parameters, correspond to a mesh size of lambda_mesh/(n*parmesh),
#: where n is the refractive index of the medium
fem.lambda_mesh = 0.6 * mum  #: flt: incident wavelength
fem.parmesh_des = 10
fem.parmesh_incl = 10
fem.parmesh = 10
fem.parmesh_pml = fem.parmesh * 2 / 3

fem.Nix = 201
fem.Niy = 201
# fem.getdp_verbose = 4
# fem.gmsh_verbose = 4
# fem.python_verbose = 1
```

Here we define an ellipsoidal rod as the scatterer:

```
fem.inclusion_flag = True

def ellipse(Rinclx, Rincly, rot_incl, x0, y0):
    c, s = np.cos(rot_incl), np.sin(rot_incl)
    Rot = np.array([[c, -s], [s, c]])
    nt = 360
    theta = np.linspace(-pi, pi, nt)
    x = Rinclx * np.sin(theta)
    y = Rincly * np.cos(theta)
    x, y = np.linalg.linalg.dot(Rot, np.array([x, y]))
    points = x + x0, y + y0
    return points

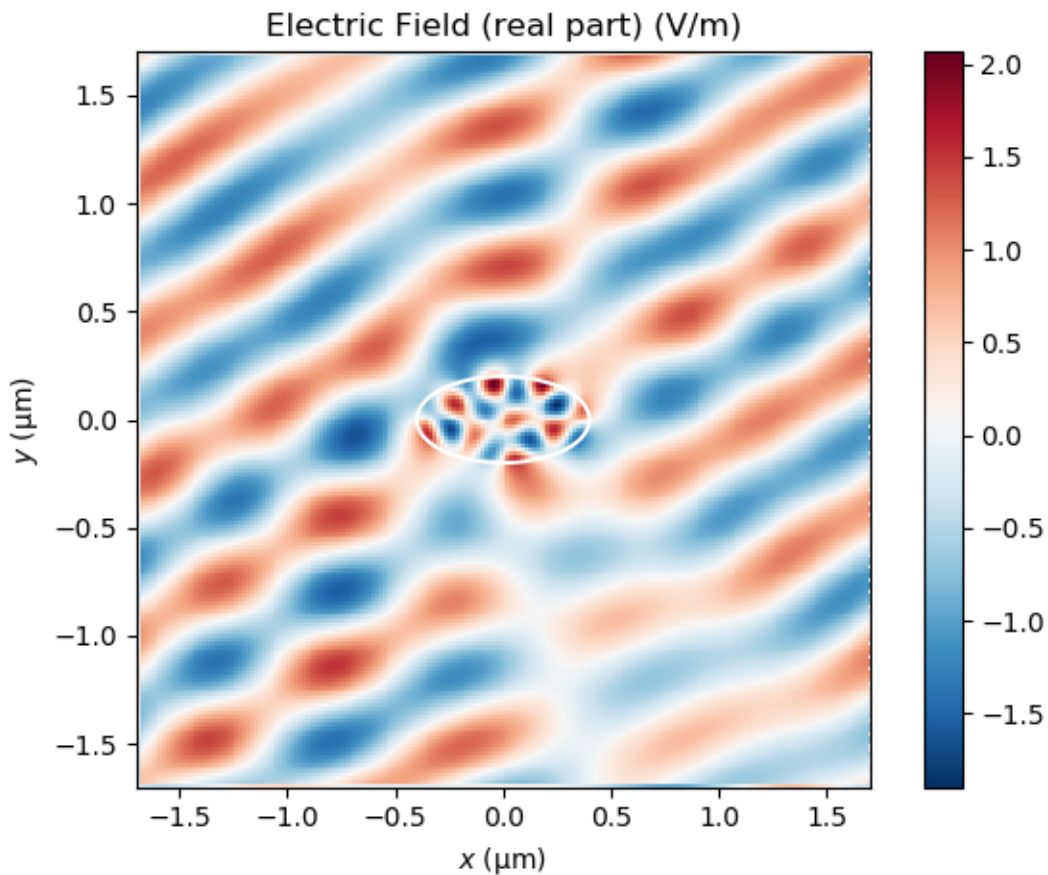
rod = ellipse(0.4 * mum, 0.2 * mum, 0, 0, 0)
```

Initialize, mesh and compute the solution:

```
fem.initialize()
fem.make_inclusion(rod)
fem.make_mesh()
fem.compute_solution()
```

Get the electric field and plot it:

```
fem.postpro_fields()
u_tot = fem.get_field_map("u_tot.txt")
fig, ax = plt.subplots()
E = u_tot.real
plt.imshow(E, cmap="RdBu_r", extent=(fem.domX_L, fem.domX_R, fem.domY_B, fem.domY_T))
plt.plot(rod[0], rod[1], "w")
plt.xlabel(r"$x$ ($\rm \mu m$)")
plt.ylabel(r"$y$ ($\rm \mu m$)")
plt.title(r"Electric Field (real part) (V/m)")
plt.colorbar()
plt.tight_layout()
```



Do a near to far field transform and get the normalized scattering width:

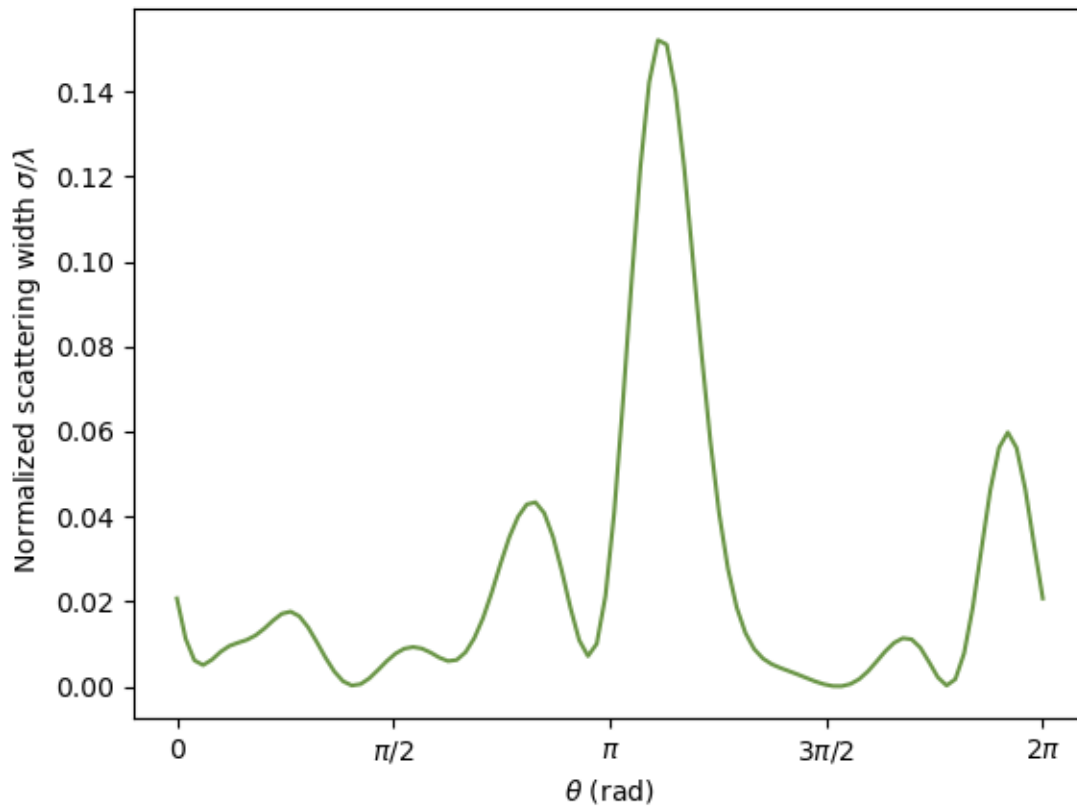
```
ff = fem.postpro_fields_n2f()
theta = np.linspace(0, 2 * pi, 100)
scs = fem.normalized_scs(ff, theta)

fig, ax = plt.subplots()
plt.plot(theta / pi, scs, "-", c="#699545")
plt.xlabel(r"$\theta$ (rad)")
plt.ylabel(r"Normalized scattering width $\sigma/\lambda$")
ax.xaxis.set_ticks([0, 0.5, 1, 1.5, 2])
ax.xaxis.set_ticklabels(["0", "$\pi/2$", "$\pi$", "$3\pi/2$", "$2\pi$"])
```

(continues on next page)

(continued from previous page)

```
scs_integ = np.trapz(scs, theta) / (2 * pi)
print("Normalized SCS", scs_integ)
```



Out:

```
Normalized SCS 0.025727174259510977
```

Total running time of the script: (0 minutes 16.454 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`, [6](#)
- `pytheas.tools`, [9](#)
- `pytheas.tools.femio`, [10](#)
- `pytheas.tools.utils`, [10](#)