grcwa Documentation

Release 0.1.2

Weiliang Jin

Mar 05, 2021

Contents:

grcwa (autoGradable RCWA) is a python implementation of rigorous coupled wave analysis (RCWA) for arbitrarily shaped photonic crystal slabs, supporting automatic differentation with autograd

CHAPTER 1

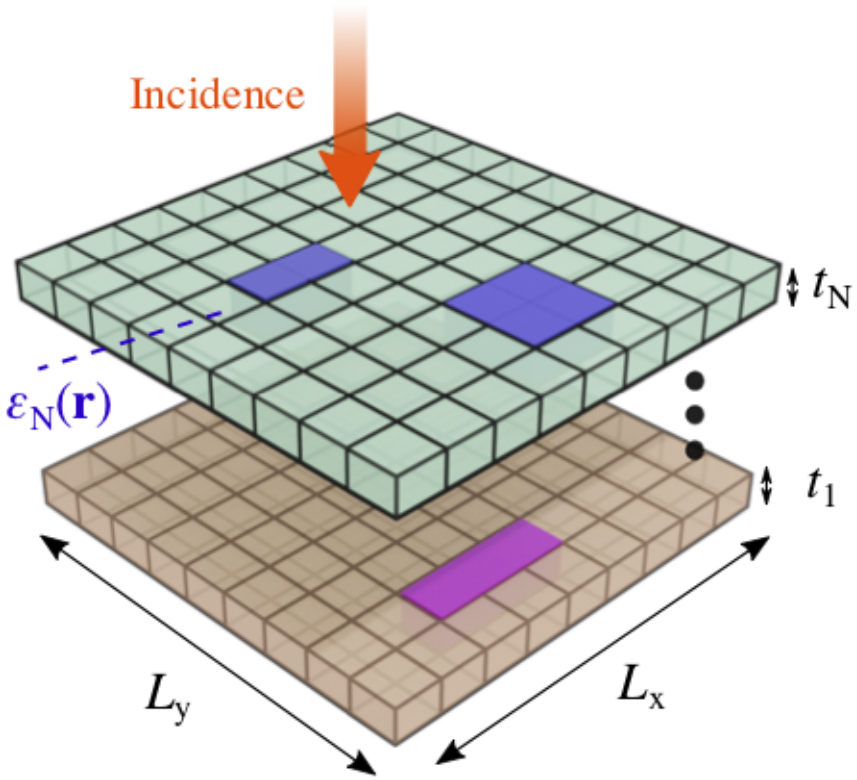
Citing

If you find **grcwa** useful for your research, please cite the following paper:

```
@article{Jin2020,
  title = {Inverse design of lightweight broadband reflector for relativistic␣
↪lightsail propulsion},
  author ={Jin, Weiliang and Li, Wei and Orenstein, Meir and Fan, Shanhui},
  year = {2020},
  journal = {ACS Photonics},
  volume = {7},
  number = {9},
  pages = {2350--2355},
  year = {2020},
  publisher = {ACS Publications}
}
```

CHAPTER 2

Features



RCWA solves EM-scattering problems of stacked photonic crystal slabs. As illustrated in the above figure, the photonic structure can have *N* layers of different thicknesses and independent spatial dielectric profiles. All layers are periodic in the two lateral directions, and invariant along the vertical direction.

- Each photonic crystal layer can have arbitrary dielectric profile on the *2D* grids.
- **autograd** is integrated into the package, allowing for automated and fast gradient evaluations for the sake of large-scale optimizations. Autogradable parameters include dielectric constant on every grid, frequency, angles,

thickness of each layer, and periodicity (however the ratio of periodicity along the two lateral directions must be fixed).

2.1 Tutorial

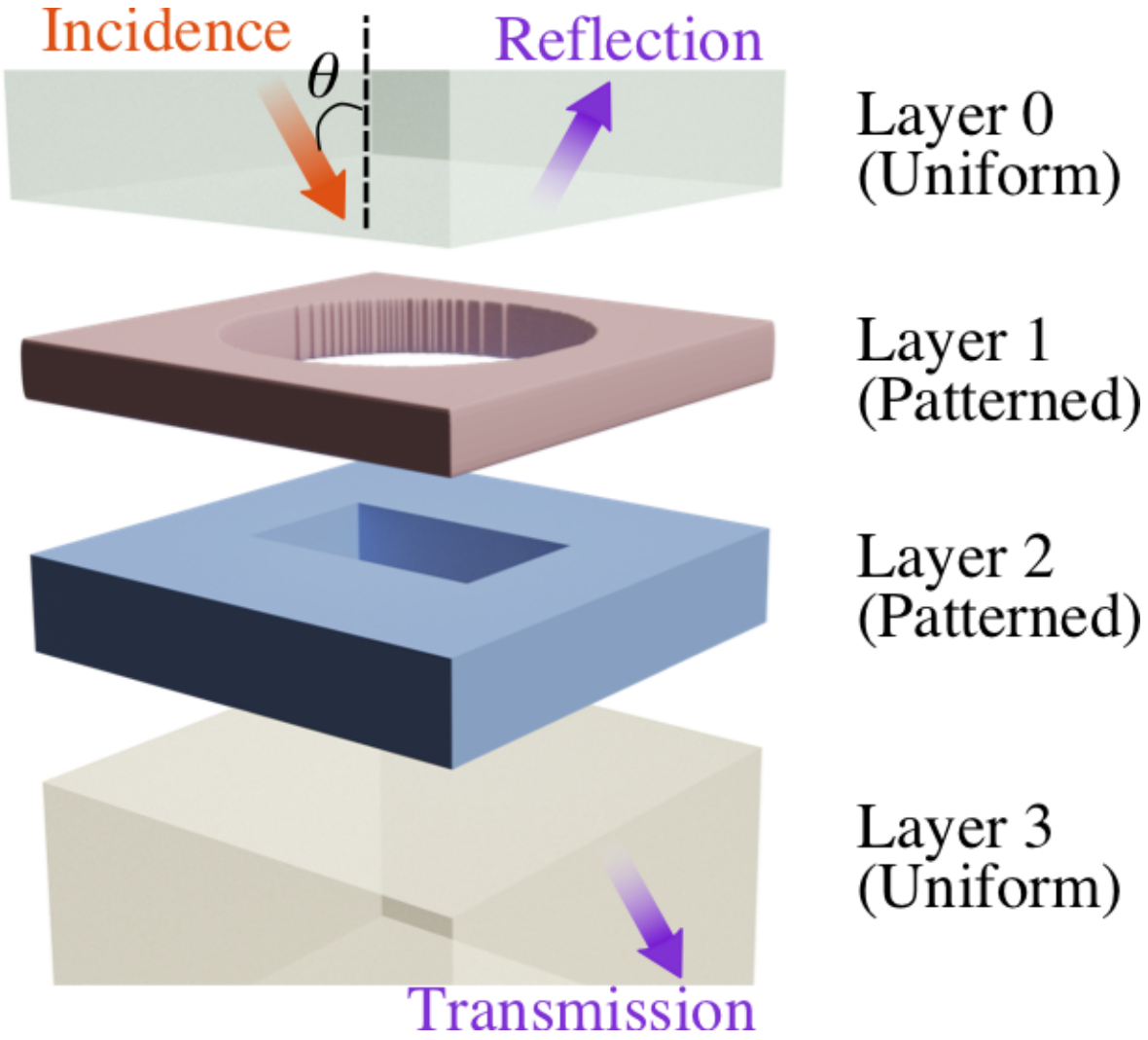
- Installation:

```
$ pip install grcwa
```

Or,

```
$ git clone git://github.com/weiliangjinca/grcwa
$ pip install .
```

- Example 1: transmission and reflection of a square lattice of a hole: see *ex1.py* in the example folder.
- Example 2: Transmission and reflection of two patterned layers: (see *ex2.py* in the example folder), as illustrated in the figure below (only a **unit cell** is plotted)



  - *Periodicity* in the lateral direction is $L_x = L_y = 0.2$, and *frequency* is 1.0.

- The incident light has an angel *pi*/10.

```
import grcwa
import numpy as np
grcwa.set_backend('autograd') # if autograd needed

 # lattice constants
 L1 = [0.2,0]
 L2 = [0,0.2]
 # Truncation order (actual number might be smaller)
 nG = 101
 # frequency
 freq = 1.
 # angle
 theta = np.pi/10
 phi = 0.

 # setup RCWA
 obj = grcwa.obj(nG,L1,L2,freq,theta,phi,verbose=1)
```

- Geometry: the thicknesses of the four layers are 0.1,0.2,0.3, and 0.4. For patterned layers, we consider total grid points N_x * N_y = 100*100 within the unit cell.
- Dielectric constant: 2.0 for the 0-th layer; 4.0 (1.0) for the 1st layer in the orange (void) region; 6.0 (1.0) for the 2nd layer in the bule (void) region; and 3.0 for the last layer.

```
Np = 2 # number of patterned layers
Nx = 100
Ny = 100

thick0 = 0.1
pthick = [0.2,0.3]
thickN = 0.4

ep0 = 2.
epN = 3.

obj.Add_LayerUniform(thick0,ep0)
for i in range(Np):
    obj.Add_LayerGrid(pthick[i],Nx,Ny)
obj.Add_LayerUniform(thickN,epN)

obj.Init_Setup()
```

- Patterned layer: the 1-th layer a circular hole of radius 0.5 L_x, and the 2-nd layer has a square hole of 0.5 L_x

```
radius = 0.5
a = 0.5

ep1 = 4.
ep2 = 6.
epbkg = 1.

# coordinate
x0 = np.linspace(0,1.,Nx)
y0 = np.linspace(0,1.,Ny)
x, y = np.meshgrid(x0,y0,indexing='ij')
```

(continues on next page)

(continued from previous page)

```
# layer 1
epgrid1 = np.ones((Nx,Ny))*ep1
ind = (x-.5)**2+(y-.5)**2<radius**2
epgrid1[ind]=epbkg

# layer 2
epgrid2 = np.ones((Nx,Ny))*ep2
ind = np.logical_and(np.abs(x-.5)<a/2 and np.abs(y-.5)<a/2))
epgrid2[ind]=epbkg

# combine epsilon of all layers
epgrid = np.concatenate((epgrid1.flatten(),epgrid2.flatten()))
obj.GridLayer_geteps(epgrid)
```

- Incident light is *s*-polarized

```
planewave={'p_amp':0,'s_amp':1,'p_phase':0,'s_phase':0}
obj.MakeExcitationPlanewave(planewave['p_amp'],planewave['p_phase'],planewave[
↪'s_amp'],planewave['s_phase'],order = 0)

# solve for R and T
R,T= obj.RT_Solve(normalize=1)
```

- Example 3: topology optimization of reflection of a single patterned layer, see *ex3.py* in the example folder.

2.2 Installation

2.2.1 Stable release

To install grcwa, run this command in your terminal:

```
$ pip install grcwa
```

This is the preferred method to install grcwa, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2.2 From sources

The sources for grcwa can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/weiliangjinca/grcwa
```

Or download the tarball:

```
$ curl -OJL https://github.com/weiliangjinca/grcwa/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

2.3 Usage

To use grcwa in a project:

```
import grcwa
```

To enable autograd:

```
grcwa.set_backend('autograd')
```

To initialize the RCWA:

```
obj = grcwa.obj(nG,L1,L2,freq,theta,phi,verbose=0) # verbose=1 for output the actual␣
↪nG
```

To add layers, the order of adding will determine the layer order (1st added layer is 0-th layer, 2nd to be 1st layer, and so forth):

```
obj.Add_LayerUniform(thick0,ep0) # uniform slab
obj.Add_LayerGrid(thickp,Nx,Ny) # patterned layer

# after add all layers:
obj.Init_Setup()
```

To feed the epsilon profile for patterned layer:

```
# x is a 1D array: np.concatenate((epgrid1.flatten(),epgrid2.flatten(),...))
obj.GridLayer_geteps(x)
```

To scale the periodicity in the both lateral directions simultaneously (as an autogradable parameter):

```
obj.Init_Setup(Pscale=scale) # period will be scale*Lx and scale*Ly
```

Fourier space truncation options

```
obj.Init_Setup(Gmethod=0) # 0 for circular, 1 for rectangular
```

To define planewave excitation:

```
obj.MakeExcitationPlanewave(p_amp,p_phase,s_amp,s_phase,order = 0)
```

To define incidence light other than planewave:

```
obj.a0 = ... # forward
obj.bN = ... # backward, each have a length 2*obj.nG, for the 2 lateral directions
```

To normalize output when the 0-th media is not vacuum, or for oblique incidence:

```
R, T = obj.RT_Solve(normalize = 1)
```

To get Poynting flux by order:

```
Ri, Ti = obj.RT_Solve(byorder=1) # Ri(Ti) has length obj.nG, too see which order,␣
↪check obj.G; too see which kx,ky, check obj.kx obj.ky
```

To get amplitude of eigenvectors at some layer at some zoffset

```
ai,bi = obj.GetAmplitudes(which_layer,z_offset)
```

To get real-space epsilon profile reconstructured from the truncated Fourier orders

```
ep = obj.Return_eps(which_layer,Nx,Ny,component='xx') # For patterned layer component
↪= 'xx','xy','yx','yy','zz'; For uniform layer, currently it's assumed to be
↪isotropic
```

To get Fourier amplitude of fields at some layer at some zoffset

```
E,H = obj.Solve_FieldFourier(which_layer,z_offset) #E = [Ex,Ey,Ez], H = [Hx,Hy,Hz]
```

To get fields in real space on grid points

```
E,H = obj.Solve_FieldOnGrid(which_layer,z_offset) # #E = [Ex,Ey,Ez], H = [Hx,Hy,Hz]
```

To get volume integration with respect to some convolution matrix M defined for 3 directions, respectively:

```
val = obj.Volume_integral(which_layer,Mx,My,Mz,normalize=1)
```

To compute Maxwell stress tensor, integrated over the z-plane:

```
Tx,Ty,Tz = obj.Solve_ZStressTensorIntegral(which_layer)
```

2.4 Convention

- The vacuum permittivity, permeability, and speed of light are *1*.
- The time harmonic convention is *exp(-i omega t)*.

2.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/weiliangjinca/grcwa/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

grcwa could always use more documentation, whether as part of the official grcwa docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/weiliangjinca/grcwa/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.5.2 Get Started!

Ready to contribute? Here's how to set up *grcwa* for local development.

1. Fork the *grcwa* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/grcwa.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv grcwa
$ cd grcwa/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 grcwa tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/weiliangjinca/grcwa/pull_requests and make sure that the tests pass for all supported Python versions.

2.5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_grcwa
```

2.5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

2.6 Credits

My implementation of RCWA received helpful discussions from Dr. Zin Lin. Many details of implementations were referred to a RCWA package implemented in c called S4. The idea of integrating **Autograd** into RCWA package rather than deriving adjoint-variable gradient by hand was inspired by a discussion with Dr. Ian Williamson and Dr. Momchil Minkov. The backend and many other styles follow their implementation in legume. Haiwen Wang and Cheng Guo provided useful feedback. Lastly, the template was credited to **Cookiecutter_** and the **'audreyr/cookiecutter-pypackage'_**.

2.6.1 Development Lead

- Weiliang Jin <jwlaaa@gmail.com>

2.6.2 Contributors

None yet. Why not be the first?

2.7 History

2.7.1 0.1.2 (2020-11-01)

- Add example for hexagonal lattice

2.7.2 0.1.1 (2020-05-18)

- Fix license

2.7.3 0.1 (2020-05-12)

- First release on PyPI.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search