Rigorous Coupled Wave Analysis

Release 1.0

May 19, 2023

Contents:

1 Examples using RCWA **1**
- 1.1 Bragg Mirror Reflection Spectra . . . 1
- 1.2 Reflection from a Dispersive Interface (Silicon) . . . 2
- 1.3 Reflection from a Dispersive Interface (SiO2) . . . 3
- 1.4 Ellipsometry of bare Si wafer [BROKEN] . . . 4
- 1.5 Rectangular Lattice Photonic Crystal with Triangular Shape . . . 4
- 1.6 Rectangular Diffraction Grating . . . 5

2 rcwa API **7**

3 Materials in rcwa **9**
- 3.1 Interpolation and Extrapolation . . . 9
- 3.2 Imaginary Sign Convention . . . 10
- 3.3 Example Tabulated Data Files . . . 10
- 3.4 Material Examples . . . 10

4 Mathematics of RCWA **11**
- 4.1 Definitions and Conventions . . . 11
- 4.2 Fields and Field Coefficients . . . 11
- 4.3 Field Coefficients . . . 12
- 4.4 Mode Coefficients . . . 12
- 4.5 Scattering Matrices couple Mode Coefficients . . . 13
- 4.6 Finding Mode coefficients inside an arbitrary layer . . . 13
- 4.7 Find E/H Coefficients inside an arbitrary Layer . . . 14
- 4.8 Finding the electric and magnetic fields inside an arbitrary layer . . . 14

5 Getting Started **15**
- 5.1 Installation . . . 15

6 Hello World Program **17**

7 2D Photonic Crystal Example **19**

8 Features **21**

9 Documentation **23**

10 License **25**

11 Indices and tables 27

CHAPTER 1

Examples using RCWA

These examples can all be found in the *examples* folder in the main *rcwa* package on github.

1.1 Bragg Mirror Reflection Spectra

```
# Author: Jordan Edmunds, Ph.D. Student, UC Berkeley
# Contact: jordan.e@berkeley.edu
# Creation Date: 11/01/2019
#
from rcwa import Layer, LayerStack, Source, Solver, Plotter
import numpy as np
from matplotlib import pyplot as plt

def solve_system():
    designWavelength = 1.3
    startWavelength = 0.6
    stopWavelength = 2.2
    stepWavelength = 0.005

    n1 = 3.5 # refractive index of layer 1 (Si)
    n2 = 1.45 # refractive index of layer 2 (SiO2)
    t1 = designWavelength/4/n1
    t2 = designWavelength/4/n2

    reflectionLayer = Layer(n=1)
    transmissionLayer = Layer(n=n1)
    layer0 = Layer(n=n1, thickness=t1)
    layer1 = Layer(n=n2, thickness=t2)
    layer2 = Layer(n=n1, thickness=t1)
    layer3 = Layer(n=n2, thickness=t2)
    layer4 = Layer(n=n1, thickness=t1)
    layer5 = Layer(n=n2, thickness=t2)
    layer6 = Layer(n=n1, thickness=t1)
```

(continues on next page)

(continued from previous page)

```
    layer7 = Layer(n=n2, thickness=t2)
    layer8 = Layer(n=n1, thickness=t1)
    layer9 = Layer(n=n2, thickness=t2)
    layer10 = Layer(n=n1, thickness=t1)
    stack = LayerStack(layer0, layer1, layer2, layer3, layer4, layer5, layer6, layer7,
↪ layer8, layer9, layer10,
                       incident_layer=reflectionLayer, transmission_
↪layer=transmissionLayer)
    source = Source(wavelength=designWavelength)

    print("Solving system...")
    TMMSolver = Solver(stack, source, 1)
    wavelengths = np.arange(startWavelength, stopWavelength + stepWavelength,
            stepWavelength)

    results = TMMSolver.solve(wavelength=wavelengths)
    return results


if __name__ == '__main__':
    results = solve_system()
    fig, ax = results.plot(x='wavelength', y=['RTot', 'TTot', 'conservation'])
    plt.show()
```

1.2 Reflection from a Dispersive Interface (Silicon)

```
# Author: Jordan Edmunds, Ph.D. Student, UC Berkeley
# Contact: jordan.e@berkeley.edu
# Creation Date: 11/01/2019
#
from rcwa import Material, Layer, LayerStack, Source, Solver, Plotter

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

def solve_system():
        startWavelength = 0.25
        stopWavelength = 0.8
        stepWavelength = 0.001

        # Setup the source
        source = Source(wavelength=startWavelength)

        # Setup the materials and geometry
        si = Material(name='Si')

        # Setup the interface
        reflectionLayer = Layer(n=1) # Free space
        transmissionLayer = Layer(material=si)
        stack = LayerStack(incident_layer=reflectionLayer, transmission_
↪layer=transmissionLayer)

        # Setup the solver
```

(continues on next page)

(continued from previous page)

```
        TMMSolver = Solver(stack, source, (1, 1))

        # Setup and run the sweep
        wavelengths = np.arange(startWavelength, stopWavelength + stepWavelength,
                stepWavelength)
        results = TMMSolver.solve(wavelength=wavelengths)
        return results


if __name__ == '__main__':
        results = solve_system()
        results.plot(x='wavelength', y=['RTot', 'TTot', 'conservation'])
        plt.show()
```

1.3 Reflection from a Dispersive Interface (SiO2)

```
# Author: Jordan Edmunds, Ph.D. Student, UC Berkeley
# Contact: jordan.e@berkeley.edu
# Creation Date: 11/01/2019
#
from rcwa import Material, Layer, LayerStack, Source, Solver, Plotter
import numpy as np
from matplotlib import pyplot as plt

def solve_system():
        startWavelength = 0.25
        stopWavelength = 0.85
        stepWavelength = 0.001

        source = Source(wavelength=startWavelength)
        siO2 = Material('SiO2')

        reflectionLayer = Layer(n=1) # Free space
        transmissionLayer = Layer(material=siO2)
        stack = LayerStack(incident_layer=reflectionLayer, transmission_
↪layer=transmissionLayer)

        TMMSolver = Solver(stack, source, (1, 1))
        wavelengths = np.arange(startWavelength, stopWavelength + stepWavelength,
                stepWavelength)

        results = TMMSolver.solve(wavelength=wavelengths)
        return results


if __name__ == '__main__':
        results = solve_system()
        fig, ax = results.plot(x='wavelength', y=['RTot', 'TTot', 'conservation'])
        plt.show()
```

1.4 Ellipsometry of bare Si wafer [BROKEN]

```
# Author: Jordan Edmunds, Ph.D. Student, UC Berkeley
# Contact: jordan.e@berkeley.edu
# Creation Date: 11/01/2019
#
from rcwa import Material, Layer, LayerStack, Source, Solver, Plotter
import numpy as np
from matplotlib import pyplot as plt

def solve_system():
        startWavelength = 0.25
        stopWavelength = 0.850
        stepWavelength = 0.001
        incident_angle = np.radians(75)

        source = Source(wavelength=startWavelength, theta=incident_angle)
        si = Material('Si')

        reflectionLayer = Layer(n=1) # Free space
        transmissionLayer = Layer(material=si)
        stack = LayerStack(incident_layer=reflectionLayer, transmission_
↪layer=transmissionLayer)

        TMMSolver = Solver(stack, source)
        wavelengths = np.arange(startWavelength, stopWavelength + stepWavelength,
                stepWavelength)

        TMMSolver.solve(wavelength=wavelengths)

        tan_psi_predicted = TMMSolver.results['tanPsi']
        cos_delta_predicted = TMMSolver.results['cosDelta']

        fig, ax = plt.subplots()
        ax.plot(wavelengths, tan_psi_predicted)
        ax.plot(wavelengths, cos_delta_predicted)
        ax.legend([r'$Tan(\Psi)$', r'$Cos(\Delta)$'])

        return fig, ax


if __name__ == '__main__':
        fig, ax = solve_system()
        plt.show()
```

1.5 Rectangular Lattice Photonic Crystal with Triangular Shape

```
from rcwa import Source, Layer, LayerStack, Crystal, Solver, example_dir
from rcwa.shorthand import complexArray
import numpy as np
import os


def solve_system():
```

(continues on next page)

(continued from previous page)

```
    data_path = os.path.join(example_dir, 'triangleData.csv')
    devicePermittivityCellData = np.transpose(np.loadtxt(data_path, delimiter=','))
    devicePermeabilityCellData = 1 + 0 * devicePermittivityCellData

    reflectionLayer = Layer(er=2.0, ur=1.0)
    transmissionLayer = Layer(er=9.0, ur=1.0)

    wavelength = 2
    deg = np.pi / 180
    k0 = 2*np.pi/wavelength
    theta = 60 * deg
    phi = 30*deg
    pTEM = 1/np.sqrt(2)*complexArray([1,1j])
    source = Source(wavelength=wavelength, theta=theta, phi=phi, pTEM=pTEM,␣
↪layer=reflectionLayer)
    t1, t2 = complexArray([1.75, 0, 0]), complexArray([0, 1.5, 0])

    crystalThickness = 0.5

    numberHarmonics = (3, 3)

    deviceCrystal = Crystal(t1, t2, er=devicePermittivityCellData,␣
↪ur=devicePermeabilityCellData)
    layer1 = Layer(crystal=deviceCrystal, thickness=crystalThickness)
    layerStack = LayerStack(layer1, incident_layer=reflectionLayer, transmission_
↪layer=transmissionLayer)

    rcwa_solver = Solver(layerStack, source, numberHarmonics)
    results = rcwa_solver.solve()
    return results


if __name__ == '__main__':
    results = solve_system()
    # Get the amplitude reflection and transmission coefficients
    (rxCalculated, ryCalculated, rzCalculated) = (results['rx'], results['ry'],␣
↪results['rz'])
    (txCalculated, tyCalculated, tzCalculated) = (results['tx'], results['ty'],␣
↪results['tz'])

    # Get the diffraction efficiencies R and T and overall reflection and␣
↪transmission coefficients R and T
    (R, T, RTot, TTot) = (results['R'], results['T'], results['RTot'], results['TTot
↪'])
    print(RTot, TTot, RTot + TTot)
```

1.6 Rectangular Diffraction Grating

```
from rcwa import Source, Layer, LayerStack, Crystal, Solver, RectangularGrating
from rcwa.shorthand import complexArray
import numpy as np


def solve_system():
```

(continues on next page)

(continued from previous page)

```
    reflection_layer = Layer(er=1.0, ur=1.0)
    transmission_layer = Layer(er=9.0, ur=1.0)

    wavelength = 0.5
    deg = np.pi / 180
    k0 = 2*np.pi/wavelength
    theta = 60 * deg
    phi = 1*deg
    pTEM = 1/np.sqrt(2)*complexArray([1,1j])
    source = Source(wavelength=wavelength, theta=theta, phi=phi, pTEM=pTEM,␣
↪layer=reflection_layer)

    crystal_thickness = 0.5

    N_harmonics = 11

    grating_layer = RectangularGrating(period=2, thickness=0.5, n=4, n_void=1, nx=500)
    layer_stack = LayerStack(grating_layer, incident_layer=reflection_layer,␣
↪transmission_layer=transmission_layer)

    solver_1d = Solver(layer_stack, source, N_harmonics)
    results = solver_1d.solve()

    return results

if __name__ == '__main__':
    results = solve_system()

    # Get the amplitude reflection and transmission coefficients
    (rxCalculated, ryCalculated, rzCalculated) = (results['rx'], results['ry'],␣
↪results['rz'])
    (txCalculated, tyCalculated, tzCalculated) = (results['tx'], results['ty'],␣
↪results['tz'])

    # Get the diffraction efficiencies R and T and overall reflection and␣
↪transmission coefficients R and T
    (R, T, RTot, TTot) = (results['R'], results['T'], results['RTot'], results['TTot
↪'])
    print(RTot, TTot, RTot+TTot)
```

CHAPTER 2

rcwa API

The main user-facing classes: `Solver`, `Layer`, `LayerStack`, `Source`, `Material`, `Crystal`, and `Plotter`. The `Layer` class is used to set up the individual simulation layers and define their properties, including thickness, material properties, and crystal structures (if applicable). The `Material` class can be passed into the `Layer` class to define a layer whose material properties are dependent on refractive index. The `Crystal` class is used to define layers which are periodic in x and y, and is also passed into the `Layer` class. Once all individual layers are constructed, they are used to create a `LayerStack`, which contains information about which region is considered the "incident" region and which is the "transmission" region (both are semi-infinite).

The `Source` class is used to define the excitation source - the wavelength, polarization, and incident angle.

Once the user has created a `Source` and `LayerStack` class, these are passed into the `Solver` class, which then runs the simulation, and makes the results available as a dictionary `Solver.results`.

CHAPTER 3

Materials in rcwa

The *Material* class handles materials whose material properties (permittivity, permeability, or refractive index) change as a function of wavelength. There are three primary ways to describe materials:

1. Constant numerical value

This is the simplest model for a material - a constant permittivity / permeability or refractive index, passed in with the *er*, *ur* and *n* arguments.

2. Custom function of wavelength

In addition to being constants, *er*, *ur*, and *n* may be user-specified single-argument functions, which take wavelength as an argument. Units of wavelength in this case do not matter and returns a (potentially complex) scalar.

3. Tabulated data

If a value is passed to the `filename` argument, then the `Material` will load in that file. Csv formats are supported by default, with the first axis being wavelength and the second axis being the complex-valued refractive index. You may instead choose to separate the real- and imaginary parts into two separate columns. This works as well. An example tabulated data file is shown below.

4. Using a materials database - tabulated or dispersion formula

The refractiveindex.info database is supported by default, and contains many user-specified materials (i.e. `Si`, `SiO2`, `Ti`, `Pt`. This database provides both tabulated data and dispersion formulas (see dispersion formulas), depending on the material used. CAUTION: if using this database, wavelength units must be specified in micrometers.

3.1 Interpolation and Extrapolation

If using tabulated data, you may use any wavelength between the minimum and maximum values in the table, and the permittivity will be linearly interpolated if the desired wavelength falls between two tabulated points. If the desired wavelength falls outside the minimum or maximum, `rcwa` will attempt to extrapolate using the slope near the boundary and give a warning.

3.2 Imaginary Sign Convention

Lossy materials are represented with a positive imaginary refractive index (i.e. `2 + 0.1j`. Materials with gain are represented by a negative imaginary refractive index (i.e. `2 - 0.1j`).

3.3 Example Tabulated Data Files

Below is what an example file for tabulated n/k data should look like, let's call it `custom_material.csv`:

And a valid alternative `custom_material2.csv`:

Note: the files must have a header.

3.4 Material Examples

CHAPTER 4

Mathematics of RCWA

This section is for developers who want to understand the implementation of this package, for example, to implement their own manipulations on top of it or extend functionality. It follows the formulation laid out by Rumpf. This is intended as a reference for developers with a fairly advanced electromagnetics and mathematics background, and assumes a knowledge of electromagnetic modes, vectors, matrices, matrix multiplication, and vectors and matrices composed of other vectors and matrices.

4.1 Definitions and Conventions

N_x: number of harmonics along x-direction
N_y: number of harmonics along y-direction.
$N_{tot} = N_x * N_y$: Total number of harmonics
n_x: Used to index the x-harmonics. Ranges from negative to positive.
n_y: Used to index the y-harmonics. Ranges from negative to positive.
E: Electric Field
H: Magnetic Field
s: Electtric field (Fourier) coefficient
u: Magnetic field (Fourier) coefficient
c: Mode coefficient
W: Electric field coefficient mode matrix
V: Magnetic field coefficient mode matrix

Layer "0" is the incident layer. Layer "1" is the layer closest to the incident region.

4.2 Fields and Field Coefficients

The formulation of rigorous coupled wave analysis involves decomposing the electric and magnetic fields into their plane wave components. Rather than working with the electric field E_x directly, we work instead with the field

coefficients s_x.

$$E_x(x,y,z) = \sum_{n_x=-\infty}^{\infty} \sum_{n_y=-\infty}^{\infty} s_x(n_x,n_y,z) * e^{-j(k_x(n_x,n_y)*x + k_y(n_x,n_y)*y)}$$

Where

$$k_x(n_x,n_y) = k_{x,incident} - n_x * T_{1,x} - n_y * T_{2,x}$$
$$k_y(n_x,n_y) = k_{y,incident} - n_x * T_{1,y} - n_y * T_{2,y}$$

where T_1 and T_2 are the lattice parameters of the photonic crystal under study. If a 1D grating is used, $T_2 = 0$. For thin films, T_1 is also zero, and only one harmonic needs to be represented - the incident harmonic.

Note that the field coefficients are, in general, a function of z.

4.3 Field Coefficients

The electric (s) and magnetic (u) field coefficients are internally represented by vectors which have both x- and y-components, and these are different for each. Taking just electric field coefficient vector in the i-th layer:

$$s_i(z) = \begin{bmatrix} s_{i,x}(z) \\ s_{i,y}(z) \end{bmatrix}$$

The length of the $s_{i,x}$ and $s_{i,y}$ vectors have length equal to N_{tot}, so the total length of the s_i vector is $N_{tot} * 2$. These vectors are indexed (n_x, n_y). For a $N_x = 3$ and $N_y = 3$ setup, the vector looks like this:

$$s_{i,x} = \begin{bmatrix} s_{i,x}(1,1) \\ s_{i,x}(1,2) \\ s_{i,x}(1,3) \\ s_{i,x}(2,1) \\ s_{i,x}(2,2) \\ s_{i,x}(2,3) \\ s_{i,x}(3,1) \\ s_{i,x}(3,2) \\ s_{i,x}(3,3) \end{bmatrix}$$

The x-component of the zero-order harmonic corresponds to the field coefficient $s_{i,x}(2,2)$.

These field coefficients are, in general, functions of the z-coordinate. This makes them awkward to work with directly, which is why we typically work with *mode coefficients* instead.

4.4 Mode Coefficients

Internally, this package typically works with *mode coefficients*, referred to using the symbol c. Forward-propagating mode coefficients inside the i^{th} layer are represented with the symbol c_i^+ and backward-propagating mode coefficients inside the i^{th} layer are represented with the symbol c_i^-. Each mode has a single value of k_z, the propagation constant in the z-direction. You could say this is what *defines* a mode. For planar films, the modes are always plane waves. For 1D- and 2D-photonic crystals, they require several plane waves (harmonics) to represent.

Just as with field coefficients, both the x- and y-components of the mode coefficients must be kept internally. As with the field coefficients, both forward- and backward-propagating mode coefficients are represented with the x-components followed by the y-components.

$$c_i^+ = \begin{bmatrix} c_{i,x}^+ \\ c_{i,y}^+ \end{bmatrix}$$

$c_{i,x}^+$ and $c_{i,y}^+$ are vectors with length equal to the total number of harmonics (i.e. for a 2D photonic crystal if $N_x = 3$ and $N_y = 3$, then $N_{tot} = N_x * N_y = 9$, and the length of the total vector c_i^+ is 18.

4.5 Scattering Matrices couple Mode Coefficients

The scattering matrix for the i^{th} layer relates the forward- and backward propagating mode coefficients between the i^{th} and $i + 1^{th}$ layer in the following way:

$$\begin{bmatrix} c_i^- \\ c_{i+1}^+ \end{bmatrix} = \begin{bmatrix} S_{i,11} & S_{i,12} \\ S_{i,21} & S_{i,22} \end{bmatrix} \begin{bmatrix} c_i^+ \\ c_{i+1}^- \end{bmatrix}$$

This can be rearranged to solve for the $i + 1^{th}$ mode coefficients given the i^{th} mode coefficients.

4.5.1 Electric Field Coefficients

The mode coefficients in the i^{th} layer can be related to directly to the x- and y-components of the electric fields (s_x and s_y) using the electric mode matrix, represented with the symbol W. This matrix takes the forward- and backward-traveling mode coefficients and converts them into the electric field coefficients. The values of k_z for each mode are also needed, which are assembled diagonally along the λ matrix.

$$\begin{bmatrix} s_{i,x} \\ s_{i,y} \end{bmatrix} = W.e^{-\lambda z}.c_i^+ + W.e^{-\lambda z}.c_i^-$$

4.5.2 Magnetic Field Coefficients

Similarly, the mode coefficients in the i^{th} layer can be related to the x- and y-components of the magnetic field coefficients u_x and u_y using the magnetic mode matrix, represented with the symbol V, along with, as before, the λ matrix.

$$\begin{bmatrix} u_{i,x} \\ u_{i,y} \end{bmatrix} = -V.e^{-\lambda z}.c_i^+ + V.e^{-\lambda z}.c_i^-$$

4.6 Finding Mode coefficients inside an arbitrary layer

Once the scattering matrices for each layer S_i are known, the incident mode coefficients s_0 are known and the global scattering matrix S_{global} is known, the mode coefficients in an arbitrary layer can be calculated.

First, the mode coefficients in the incident region must be found. To do this, you can find the m. c_0^+ using the $W_{incident}$ matrix and $s_{incident}$ vector (which contains the x- and y-components of the zero-order harmonic), and c_0^- can be calculated from the global scattering matrix. Note that $s_{incident}$ is not the same as s_0, which also contains the reflected field coefficients.

$$c_0^+ = W_{incident}^{-1} s_{incident} c_0^- = S_{global,11} c_0^+$$

Then, by applying the formula below as many times as is required, mode coefficients within the desired layer can be found:

$$c_{i+1}^- = S_{i,12}^{-1} c_i^- - S_{i,12}^{-1} S_{i,11} c_i +$$
$$c_{i+1}^+ = S_{i,21} c_i^{+1} + S_{i,22} c_{i+1} -$$

4.7 Find E/H Coefficients inside an arbitrary Layer

Once the mode coefficients have been found, the electric and magnetic field coefficients can be found as described previously. Note that at this point, the field coefficients will still be a function of the z coordinate.

4.8 Finding the electric and magnetic fields inside an arbitrary layer

4.8.1 Electric Fields

Once the electric field coefficients are known, the electric fields can be calculated using the formula above for a certain value of x and y, as a function of z.

First, the diagonal k-matrices K_x and K_y can be converted into vectors k_x and k_y. Then, the complex exponential can be evaluated element-wise directly:

$$e_{vec} = e^{-j(k_x * x + k_y * y)}$$

This is itself a vector with the same length as s_x. The dot product (NOT inner product, there is no complex conjugation) of this vector and the exponential vector e_{vec} finally yields the x-component of the electric field. This may be repeated for the y-component as well, by dotting with the s_y vector.

$$E_x(x, y, z) = e_{vec} \cdot s_x$$

CHAPTER 5

Getting Started

5.1 Installation

The recommended way to install this software is with *pip*:

And that's it!

CHAPTER 6

Hello World Program

To run a simple example, run:

This should run an example with a 10-layer bragg mirror (also known as a [dielectric mirror](https://en.wikipedia.org/wiki/Dielectric_mirror)), which can have very high reflectance near its design wavelength, and output the reflectance and transmission as a function of wavelength.

CHAPTER 7

2D Photonic Crystal Example

The examples folder also contains a 2D photonic crystal example called *triangular_photonic_crystal.py*. Running it will print the reflection and transmission coefficients, but diving into the example you can extract any quantities of interest: amplitude reflection coefficients, diffraction efficiencies for each harmonic, etc.

CHAPTER 8

Features

- Implements 1D Transfer Matrix Method for homogenous layers
- Implements full rectangular 2D RCWA for periodic layers
- Arbitrary incident wave polarization (circular, linear, elliptical)
- Arbitrary incident wave angle of incidence
- Exactly solves Maxwell's Equations for arbitrary layer stacks of any thickness
- Compute reflected power, transmitted power, and S-parameters
- Easy to use class-based syntax
- Large, fast-to-run test suite

CHAPTER 9

Documentation

This project is documented on [Read The Docs](https://rcwa.readthedocs.io/en/latest/). For additional information, including downloading examples, you can view this project on [github](https://github.com/edmundsj/RCWA).

Author: Jordan Edmunds, UC Irvine Alumnus, UC Berkeley Ph.D. Student

Date Started: 2020/01/05

CHAPTER 10

License

This project is distributed under the [MIT license](https://mit-license.org/).

CHAPTER 11

Indices and tables

- genindex
- modindex
- search