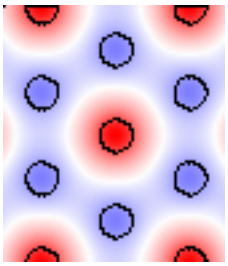
Northumbria University
NEWCASTLE

KD6041

Photonic crystal simulations



Laboratory notes

Dr. Ying-Lung Daniel Ho and Dr. Mike Taverne

Nanophotonic Engineering Laboratory

https://nanophotonicenglab.github.io/

Department of Mathematics, Physics and Electrical Engineering

Northumbria University, Newcastle

Version: October 13, 2022

Contents

1 The Transfer-Matrix Method (TMM), using the Python tmm module

1.1 Introduction

The *Transfer Matrix Method (TMM)* is a method based on matrix multiplication that can only be used for 1D structures, i.e. where the geometry varies along the X axis, but is invariant along Y and Z for example.

The next section gives a quick overview on how it works. You can find a more detailed description and derivation of it in the book "Optical Waves in Layered Media" by Pochi Yeh [1], which is available from the university library here:
https://librarysearch.northumbria.ac.uk/permalink/f/1t01hd3/44UON_ALMA21117968020003181

To set up and learn how to use the Python TMM module, please see sections 2.2.3, 2.2.5 and 2.3 in the appendix.

1.2 The Transfer-Matrix model

1.2.1 2x2 Matrix formulation for a multilayer system

Let us consider the structure shown in Figure 1.1. Light incident from the air above passes through N layers of thickness d_i and of infinite width.

The relation between A_0 (forward light wave), B_0 (backward light wave) at the top of the structure and A_s (forward light wave), B_s (backward light wave) at the bottom can be written as:

$$\begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} A_s \\ B_s \end{bmatrix} \tag{1.1}$$

where M_{ij} are the matrix elements (i and j=1,2).

The matrix expression, from which the total reflectance is computed, is:

$$\bar{T} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = D_0^{-1} \left\{ \prod_{j=1}^{N} D_j P_j D_j^{-1} \right\} D_s \tag{1.2}$$

where D_j is the surface transmission and reflection matrix (*dynamical matrix* of s-waves or p-waves) for the layer j, D_0 the top surface in contact with vacuum and D_s the bottom surface in contact with the substrate. The *propagation matrix* P_j includes the phase shift Φ_j incurred on forward and backward propagation through each layer in:

$$P_j = \begin{bmatrix} e^{i\Phi_j} & 0 \\ 0 & e^{-i\Phi_j} \end{bmatrix} \tag{1.3}$$

and:

$$\Phi_j = k_{jy} \cdot d_j = \frac{2\pi \cdot n_j \cdot d_j}{\lambda_r} cos(\theta_j) \tag{1.4}$$

Here k_{jy} is the y component of the wave vectors, θ_j is the ray angle, λ_r is the cavity resonance wavelength, and d_j is the thickness of the layer j.

For *s-waves*[1] (electric field transverse to the plane of propagation, i.e. electric field along Z in the case of Figure 1.1, where the X-Y plane is the plane of propagation), the surface reflection and transmission coefficients are given by:

$$D_{js} = \begin{bmatrix} 1 & 1 \\ n_j cos(\theta_j) & -n_j cos(\theta_j) \end{bmatrix} \tag{1.5}$$

For *p-waves*[2] (electric field parallel to the plane of propagation, i.e. electric field in the X-Y plane in the case of Figure 1.1), the surface reflection and transmission coefficients are given by:

$$D_{jp} = \begin{bmatrix} cos(\theta_j) & cos(\theta_j) \\ n_j & -n_j \end{bmatrix} \tag{1.6}$$

Here we will proceed with the calculations for *s-waves* using $D_j = D_{js}$, but you should be able to apply the same calculations to p-waves using $D_j = D_{jp}$ from (1.6).

Thus, for a single layer within the stack:

$$Q_j = D_j P_j D_j^{-1} = \begin{bmatrix} cos(\Phi_j) & \frac{i}{y_j} sin(\Phi_j) \\ iy_j sin(\Phi_j) & cos(\Phi_j) \end{bmatrix} \tag{1.7}$$

where $y_j = n_j cos(\theta)$.

[1] The "S" in s-wave comes from the German "Senkrecht", meaning "vertical".
[2] The "P" in p-wave comes from the German "Parallel", meaning "parallel".

Thus the values of A_0 and B_0 represent the input components of the electric field tied to output components A_s and B_s by the relationship:

$$\begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = \bar{T} \begin{bmatrix} A_s \\ B_s \end{bmatrix} = D_0^{-1} \left\{ \prod_{j=1}^{N} \begin{bmatrix} cos(\Phi_j) & \frac{i}{y_j} sin(\Phi_j) \\ iy_j sin(\Phi_j) & cos(\Phi_j) \end{bmatrix} \right\} D_s \begin{bmatrix} A_s \\ B_s \end{bmatrix} \tag{1.8}$$

Exercise 1.2.1: Equation for p-waves

Write equation 1.8 for *p-waves*.

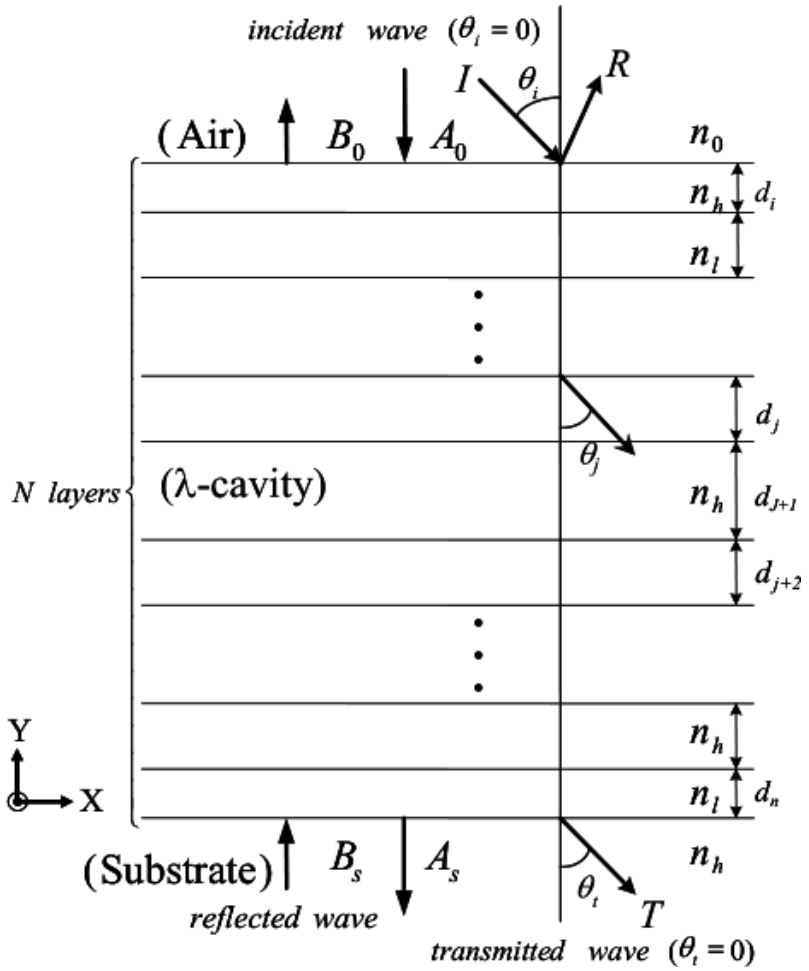


Figure 1.1: Schematic representation of a *Distributed Bragg Reflector (DBR)* made up of N layers of material with alternating refractive indices n_h and n_l. A_0 and B_0 are the forward and backward electric field amplitudes at the input, while A_s and B_s are the forward and backward electric field amplitudes at the output. d_j is the thickness of the layer j and θ_j is the angle between the propagation direction and the normal (subscripts i and t refer to incident and transmitted light).

1.2.2 Transmittance, reflectance and absorptance

If the light is incident from medium 0, the *reflection and transmission coefficients* are defined as:

$$r = \left(\frac{B_0}{A_0}\right)_{B_s=0} \tag{1.9}$$

$$t = \left(\frac{A_s}{A_0}\right)_{B_s=0} \tag{1.10}$$

Based on equation (1.1), this leads to:

$$r = \frac{M_{21}}{M_{11}} \tag{1.11}$$

$$t = \frac{1}{M_{11}} \tag{1.12}$$

Reflectance is defined as the fraction of energy reflected from the structure and is given by:

$$R = |r|^2 = \left|\frac{M_{21}}{M_{11}}\right|^2 \tag{1.13}$$

Transmittance is given by:

$$T = \frac{n_s cos(\theta_s)}{n_0 cos(\theta_0)}|t|^2 = \frac{n_s cos(\theta_s)}{n_0 cos(\theta_0)}\left|\frac{1}{M_{11}}\right|^2 \tag{1.14}$$

where the factor corrects for the difference in phase velocity.
Absorptance, which is defined as the fraction of energy dissipated, is given by:

$$A = 1 - R - T \tag{1.15}$$

1.3 Guided examples

1.3.1 Reflection from two layers

We will start with a simple example of the reflection from a thin non-absorbing layer, on top of a thick absorbing layer, with air on both sides, as illustrated in Figure 1.2.

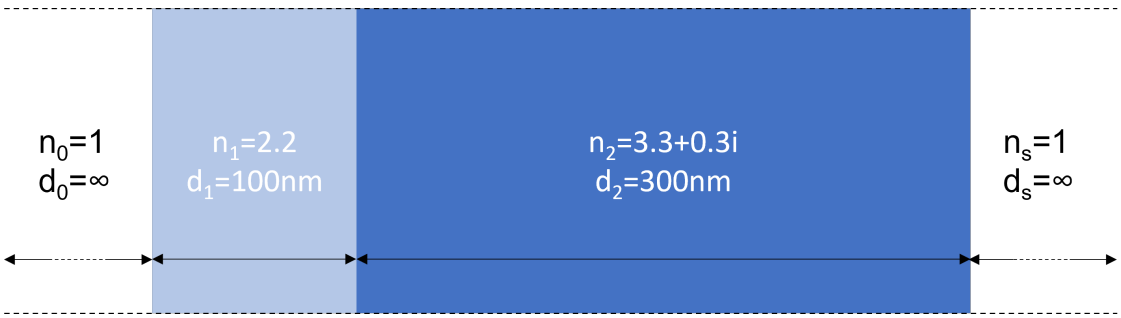


Figure 1.2: Geometry of the two-layer example.

1. Create a new Python script named *normal_incidence.py* in Spyder with the following contents:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Import the main TMM function we will use:
from tmm.tmm_core import coh_tmm

# Import numpy:
import numpy as np

# Import matplotlib for plotting:
import matplotlib.pyplot as plt

def main():
    ##### YOUR CODE GOES HERE #####
    print('Hello world!')

if __name__ == "__main__":
    main()
```

> ! Leading spaces or tabs will not be copied if you copy-paste from this document as a PDF. So make sure you enter them correctly manually! For convenience, you can get this file from the resource folder under "KD6041-resources\TMM\normal_incidence.py". See Section 2.2.3.

If you run the script now, it should simply print "Hello world!".

2. The main TMM function we will use is:

   coh_tmm(pol, n_list, d_list, th_0, lam_vac).

   It takes the following arguments:

   a) *pol* is light polarization, "s" or "p".

   b) *n_list* is the list of refractive indices, in the order that the light would pass through them. The 0'th element of the list should be the semi-infinite medium from which the light enters, the last element should be the semi- infinite medium to which the light exits (if any exits).

   c) *th_0* is the angle of incidence *in radians*: 0 for normal, pi/2 for glancing.

   d) *d_list* is the list of layer thicknesses (front to back). Should correspond one-to-one with elements of n_list. First and last elements should be "np.inf".

   e) *lam_vac* is the vacuum wavelength of the light.

   It returns the following as a dictionary:

   - *r* – reflection amplitude
   - *t* – transmission amplitude
   - *R* – reflected wave power (as fraction of incident)
   - *T* – transmitted wave power (as fraction of incident)
   - *power_entering* – Power entering the first layer, usually (but not always) equal to 1-R (see manual).
   - *vw_list* – n'th element is [v_n,w_n], the forward- and backward-traveling amplitudes, respectively, in the n'th medium just after interface with (n-1)st medium.
   - *kz_list* – normal component of complex angular wavenumber for forward-traveling wave in each layer.
   - *th_list* – (complex) propagation angle (in radians) in each layer
   - *pol*, *n_list*, *d_list*, *th_0*, *lam_vac* – same as input

> ! See `https://pythonhosted.org/tmm/tmm.html` for the official documentation of the tmm package.

3. Delete the *print('Hello world!')* line and add the following lines under "*##### YOUR CODE GOES HERE #####*":

```
# list of layer thicknesses in nm
d_list = [np.inf, 100, 300, np.inf]

# list of refractive indices
n_list = [1, 2.2, 3.3+0.3j, 1]

# call the coh_tmm function
ret = coh_tmm('s', n_list, d_list, 0, 700)

# print the return values obtained:
print(ret)
```

4. If you run the script, you should now see an output like this in the *IPython console* in the bottom right:

```
{'r': (-0.3819902164036294+0.173125874589414j), 't':
    (-0.03769252771074004-0.3453788012827543j), 'R': 0.17588909388044108, 'T':
    0.12070724302073717, 'power_entering': 0.8241109061195588, 'vw_list':
    array([[ 0.        +0.j        ,  0.        +0.j        ],
       [ 0.62309358+0.04721615j, -0.00508379+0.12590973j],
       [-0.20150892+0.46509462j,  0.03097654+0.04448911j],
       [-0.03769253-0.3453788j ,  0.        +0.j        ]]), 'kz_list':
    array([0.00897598+0.j        , 0.01974715+0.j        ,
       0.02962073+0.00269279j, 0.00897598+0.j        ]), 'th_list':
    array([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j]), 'pol': 's', 'n_list': array([1. +0.j
    , 2.2+0.j , 3.3+0.3j, 1. +0.j ]), 'd_list': array([ inf, 100., 300.,  inf]),
    'th_0': 0, 'lam_vac': 700}
```

You can see that the reflected wavepower R is $R = 0.17588909388044108$ for the input wavelength of 700nm that we specified.

To only print out that value, we could use *print(ret['R'])*.

5. We want to plot the reflection as a function of wavelength for wavelengths going from 400 to 800nm. To do this, start by defining a wavelength array by using the *linspace()* function from *numpy*:

```
wvl_list = np.linspace(400, 800)
```

6. Define an empty list R:

```
R=[]
```

7. Populate the list in a for loop:

```
for wvl in wvl_list:
    ret = coh_tmm('s', n_list, d_list, 0, wvl)
    R.append(ret['R'])
```

8. Finally, plot the data using matplotlib's *plot* command, add some labels and a title:

```
plt.plot(wvl_list, R)
plt.xlabel('Wavelength $\lambda$ (nm)')
plt.ylabel('Reflectance $R$ (no unit)')
plt.title('Reflectance at normal incidence')
```

9. You should have obtained a plot like this:

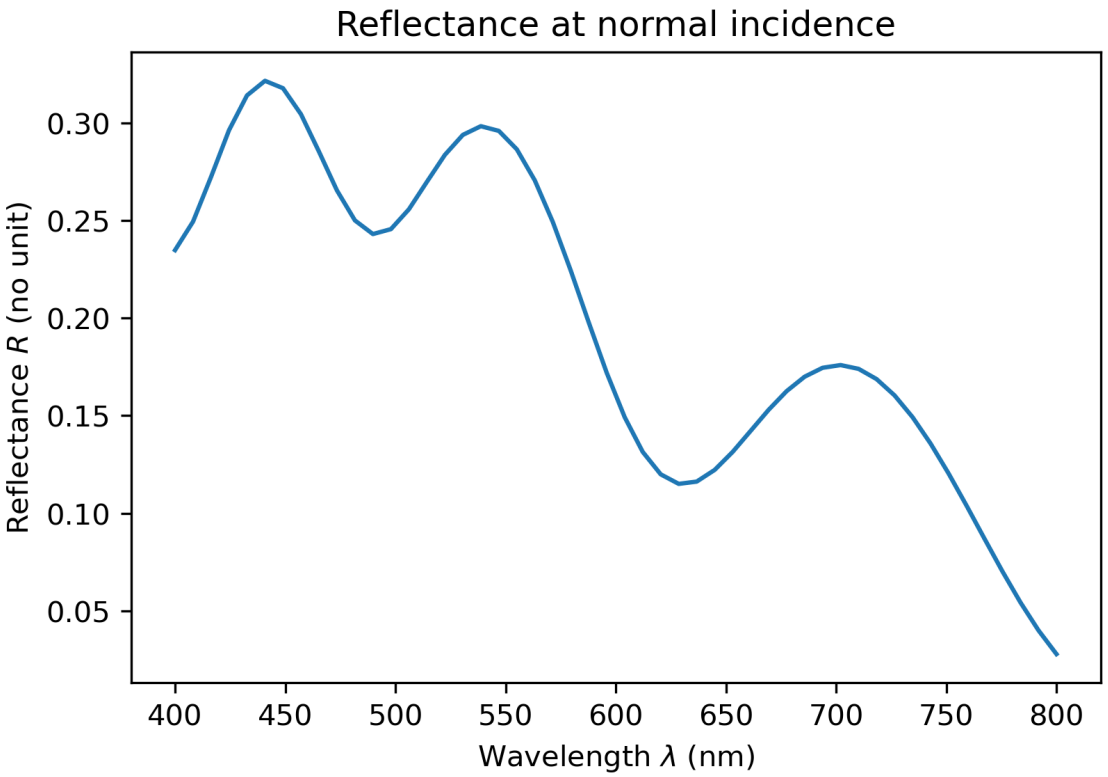


Figure 1.3: Reflectance of the structure from Figure 1.2.

If you are having any trouble, here is what your final code should look like (also available under *KD6041-resources\TMM\normal_incidence.final.py*):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Import the main TMM function we will use:
from tmm.tmm_core import coh_tmm

# Import numpy:
import numpy as np

# Import matplotlib for plotting:
import matplotlib.pyplot as plt

def main():
    ##### YOUR CODE GOES HERE #####
    # list of layer thicknesses in nm
    d_list = [np.inf, 100, 300, np.inf]
    # list of refractive indices
    n_list = [1, 2.2, 3.3+0.3j, 1]
    # call the coh_tmm function
    ret = coh_tmm('s', n_list, d_list, 0, 700)
    # print the return values obtained:
    print(ret)

    wvl_list = np.linspace(400, 800)

    R=[]

    for wvl in wvl_list:
        ret = coh_tmm('s', n_list, d_list, 0, wvl)
        R.append(ret['R'])

    plt.plot(wvl_list, R)
    plt.xlabel('Wavelength $\lambda$ (nm)')
    plt.ylabel('Reflectance $R$ (no unit)')
    plt.title('Reflectance at normal incidence')

if __name__ == "__main__":
    main()
```

Question 1.3.1: Can you explain the behaviour?

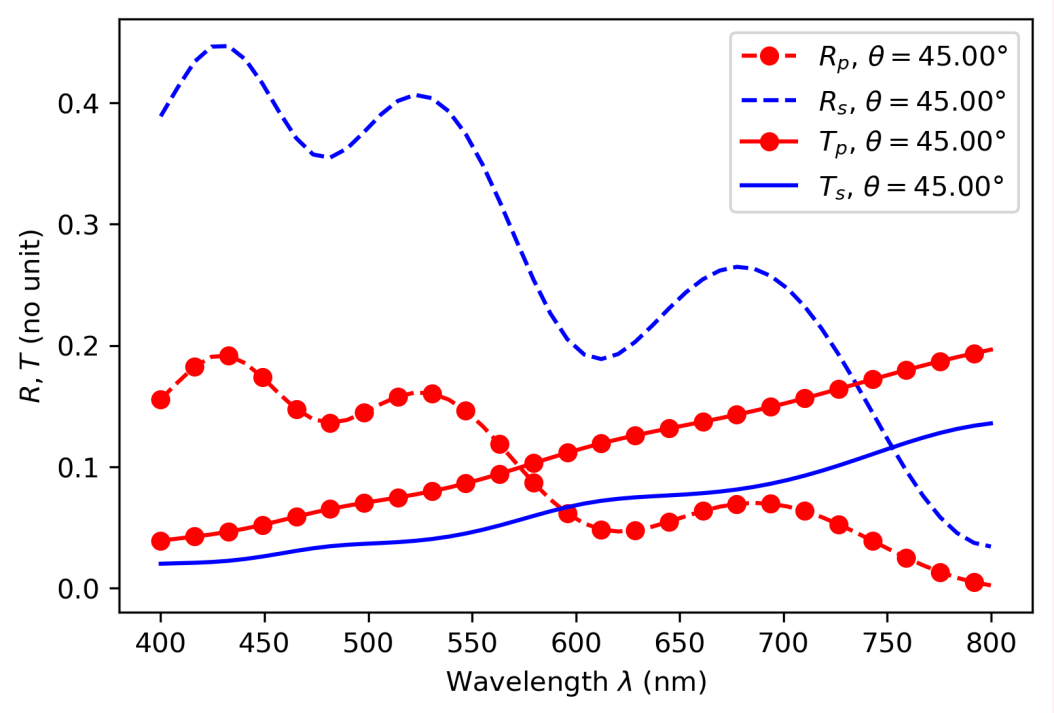
How does the reflectance in Figure 1.3 change with wavelength? Why? What is this phenomenon called?

Exercise 1.3.1: Reflectance at a different incidence angle.

Now plot the reflectance at 45° incidence. Remember to pass the angle *in radians* to coh_tmm!

Exercise 1.3.2: Comparison of reflectance, transmittance, S and P polarization

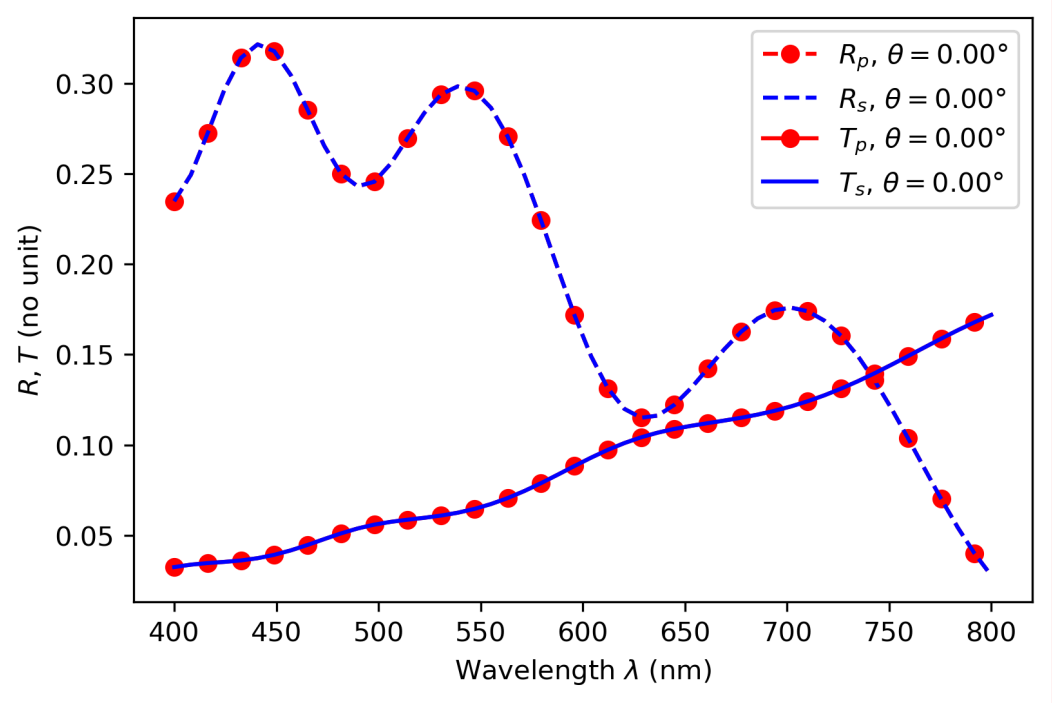
Create a plot to compare reflectance, transmittance, S and P polarization for an incidence angle of 45° like this:



Why is the reflectance for p-waves lower (or respectively their transmittance higher) than for s-waves?

Exercise 1.3.3: What happens at normal incidence?

Create a plot to compare reflectance, transmittance, S and P polarization for an incidence angle of 0° like this:



Why is there no difference between s and p-waves at normal incidence?

1.3.2 Reflection from a single layer

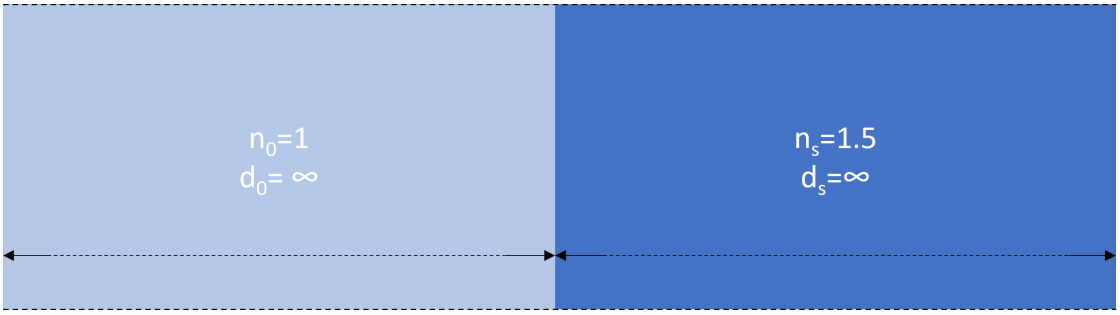


Figure 1.4: A single interface.

To plot the reflection from a single layer, re-use the code from section 1.3.1 and simply re-define the index and thickness list as follows:

```
# list of layer thicknesses in nm
d_list = [np.inf, np.inf]

# list of refractive indices
n_list = [1, 1.5]
```

Question 1.3.2: Can you explain the behaviour?

How does the reflectance change with wavelength this time? Why?

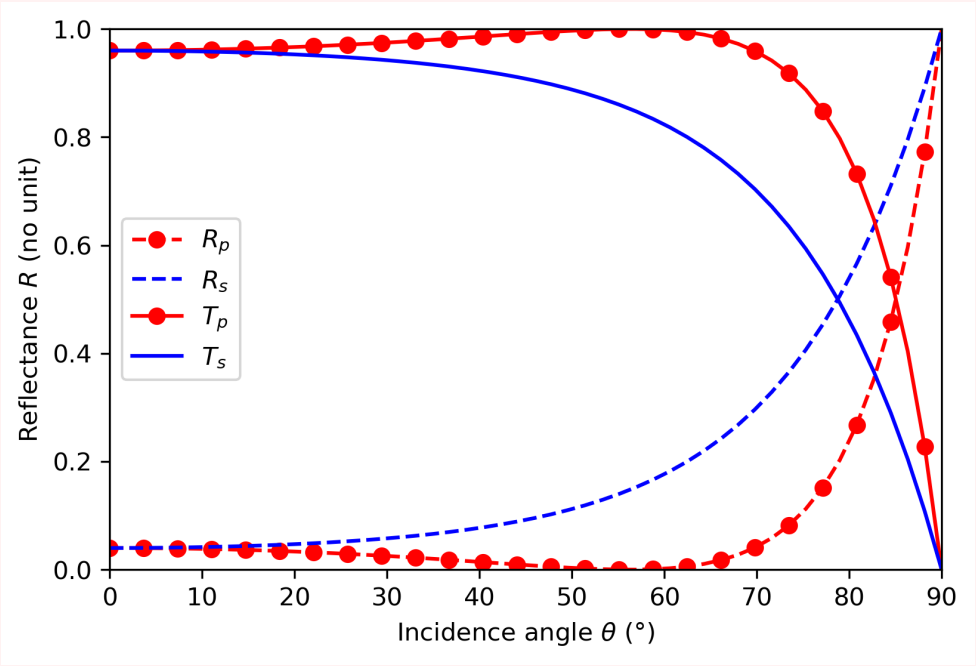
Question 1.3.3: What does the theory say?

What is the theoretical expression of R_p, R_s, T_p and T_s for a single interface between materials of refractive index n_0 and n_s? What are the values of R_p, R_s, T_p and T_s for:

- $n_0 = 1$, $n_s = 1.5$, $\theta = 0°$
- $n_0 = 1$, $n_s = 1.5$, $\theta = 45°$

Exercise 1.3.4: Reflectance and transmittance as a function of angle

1. Instead of plotting the reflectance as a function of wavelength, plot it against the incidence angle, going from 0 to 90, like this:



2. Why does the reflectance for P polarized light (R_p) almost drop to zero around 60°?

3. At what angle exactly is it the lowest?

4. Does it drop to zero?

1.3.3 Normal incidence spectra from a Distributed Bragg Reflector (DBR)

We will now consider a 1D photonic crystal, called a *Distributed Bragg Reflector (DBR)*, as illustrated in Figure 1.5.

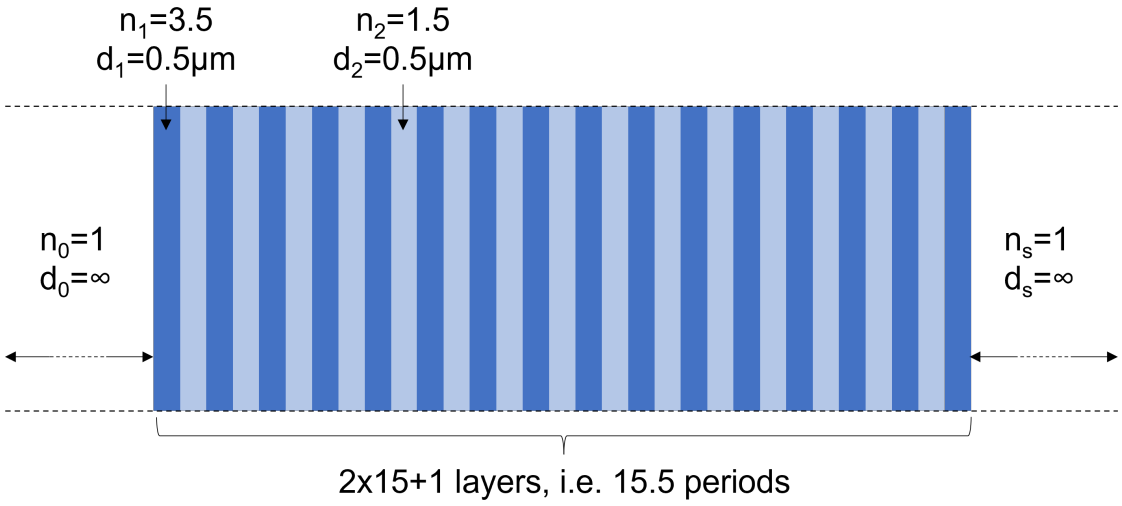


Figure 1.5: Geometry of the DBR example.

1. Open the file *KD6041-resources\TMM\DBR_TMM.stub.py* in Spyder and save it as *DBR_TMM.py*. It should look like this:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Import the main TMM function we will use:
from tmm.tmm_core import coh_tmm

# Import numpy:
import numpy as np

# Import matplotlib for plotting:
import matplotlib.pyplot as plt

def main():
    ##### YOUR CODE GOES HERE #####
    # list of layer thicknesses in nm
    d_list = [np.inf, 100, 300, np.inf]
    # list of refractive indices
```

```
    n_list = [1, 2.2, 3.3+0.3j, 1]

    wvl_list = np.linspace(400, 800)

    R=[]

    for wvl in wvl_list:
        ret = coh_tmm('s', n_list, d_list, 0, wvl)
        R.append(ret['R'])

    plt.plot(wvl_list, R)
    plt.xlabel('Wavelength $\lambda$ (nm)')
    plt.ylabel('Reflectance $R$ (no unit)')
    plt.title('Reflectance at normal incidence')

if __name__ == "__main__":
    main()
```

2. We are now going to define some variables to make it easier to adapt the code for different parameters. Add the following after "*##### YOUR CODE GOES HERE #####*":

```
n1 = 3.5
d1 = 0.5 # um
n2 = 1.5
d2 = 0.5 # um
Nperiods = 15
```

3. In order to simulate the DBR instead of just two layers, we need to change the variables *d_list* and *n_list*. You could write a for loop to build the list of layer thicknesses and refractive indices, but instead we will use the convenient "+" and "*" operators in Python for lists.

   For example:

   - 3*[a,b] will make a list [a,b,a,b,a,b].
   - [a,b,c]+[d,e,f] will make a list [a,b,c,d,e,f].

So based on this, we replace the definitions of *d_list* and *n_list* with the following:

```
# list of layer thicknesses in nm
d_list = [np.inf] + Nperiods*[d1, d2] + [d1] + [np.inf]
# list of refractive indices
n_list = [1] + Nperiods*[n1, n2] + [n1] + [1]
```

4. If you run the code now, you should get something like this:

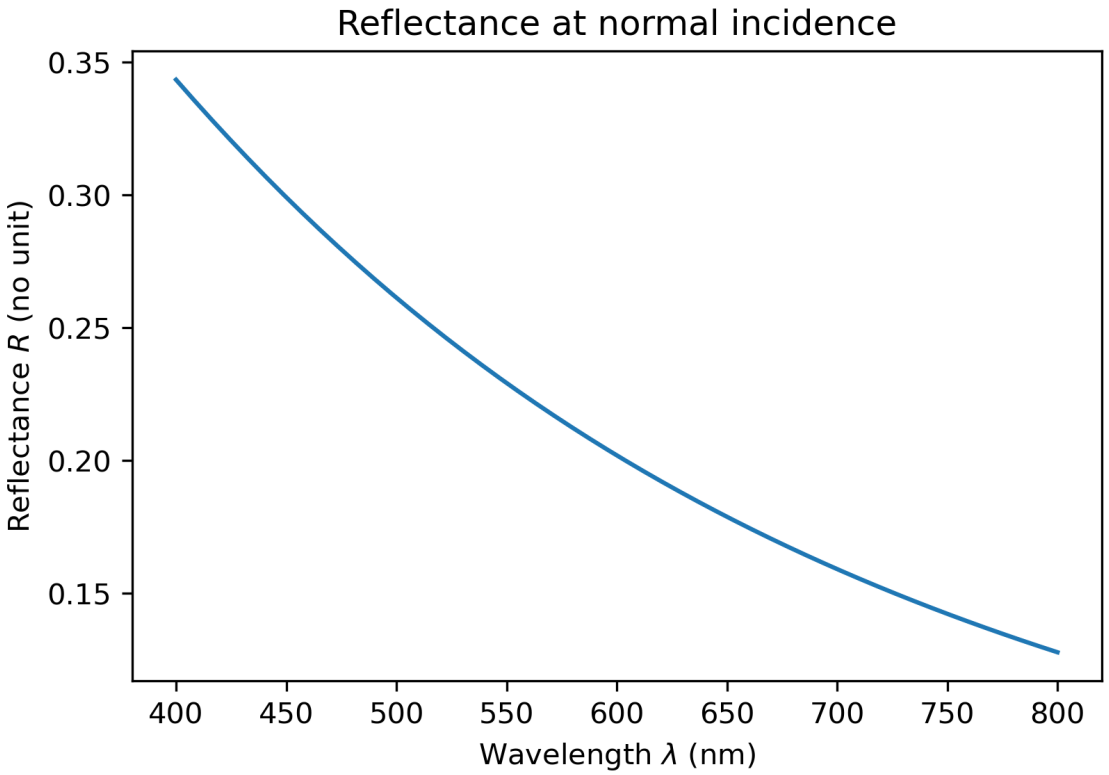


Figure 1.6: New output after changing the geometry.

Question 1.3.4: Bandgaps

a) What is the expected wavelength of the fundamental bandgap for a DBR like the one in Figure 1.5?

b) Where would higher order bandgaps in the 400-800nm range be?

c) Why does the plot in Figure 1.6 not look quite right?

5. Because we specified the thicknesses d_1 and d_2 in μm in the code, we need to also pass the wavelengths in μm. Replace the *wvl_list* definition with the following so that it represents 0.4 to 0.8 μm:

```
wvl_list = np.linspace(0.400, 0.800) # um
```

6. You should also update the label text for the X axis accordingly, so it shows the unit as μm instead of nm:

```
plt.xlabel(r'Wavelength $\lambda$ ($\mu{}m$)')
```

7. If you run the code now, you should get something like this:

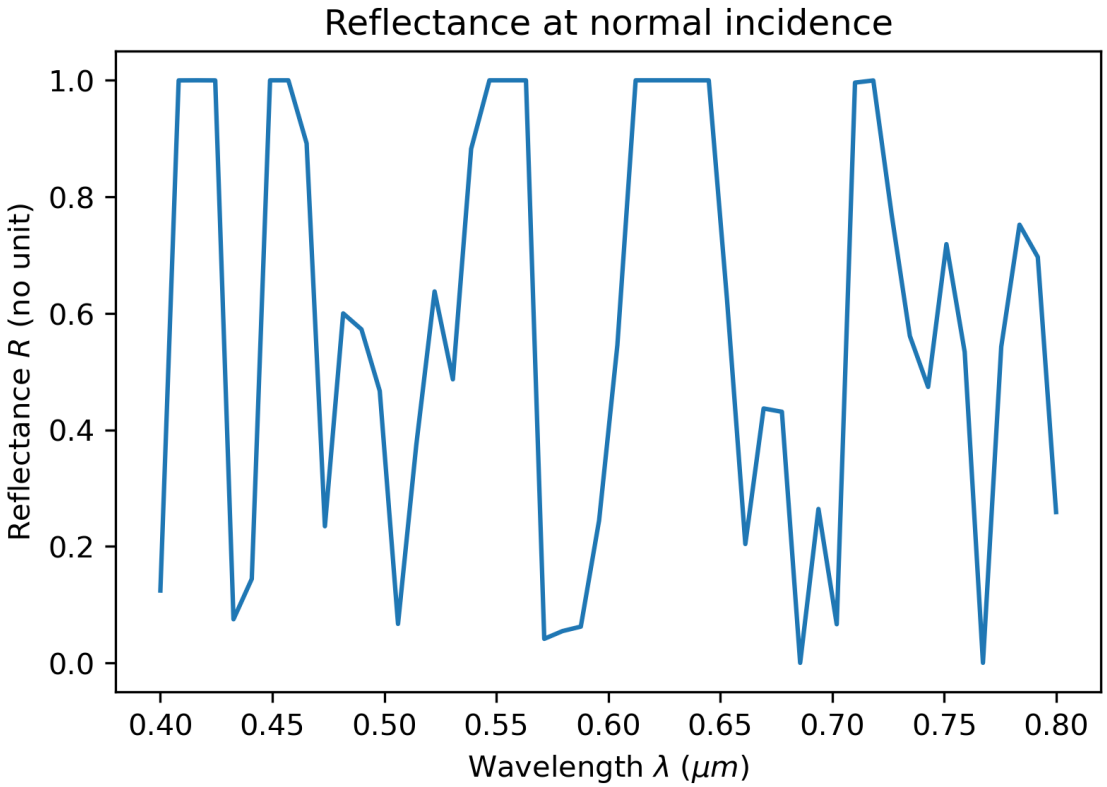


Figure 1.7: New output after correcting the wavelength range.

8. The plot looks like it does not have enough data points. By default, *np.linspace()* only returns 50 points. So let us request 1000 points instead by specifying the number of points in *np.linspace()*:

```
wvl_list = np.linspace(0.400, 0.800, 1000) # um
```

9. If you run the code now, you should get something like this:

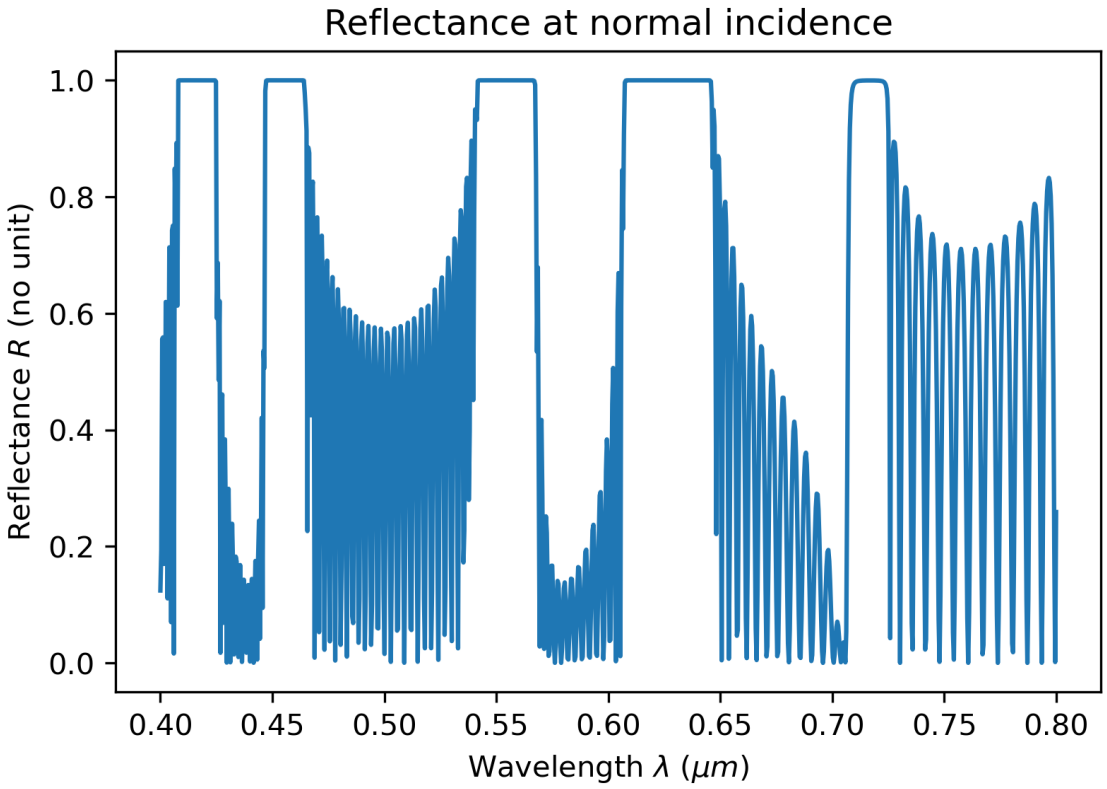


Figure 1.8: New output after using 1000 points.

1.3.4 45 degrees incidence spectra from a Distributed Bragg Reflector (DBR)

1. To plot at a different incidence angle, all we need to do is specify the angle in the call to *coh_tmm*:

   coh_tmm('s', n_list, d_list, ***0****, wvl) -> coh_tmm('s', n_list, d_list,* ***45 degrees****, wvl).*

   However, *coh_tmm* takes in angle values in radians! So we need to convert from radians to degrees first.

To do this, we will use the convenient np.rad2deg() function from numpy. Replace the *coh_tmm* call in the for loop with the following:

```
ret = coh_tmm('s', n_list, d_list, np.deg2rad(45), wvl)
```

2. You may also want to change the title to reflect this change:

```
plt.title('Reflectance at $45\degree$ incidence')
```

3. If you run the code now, you should get something like this:

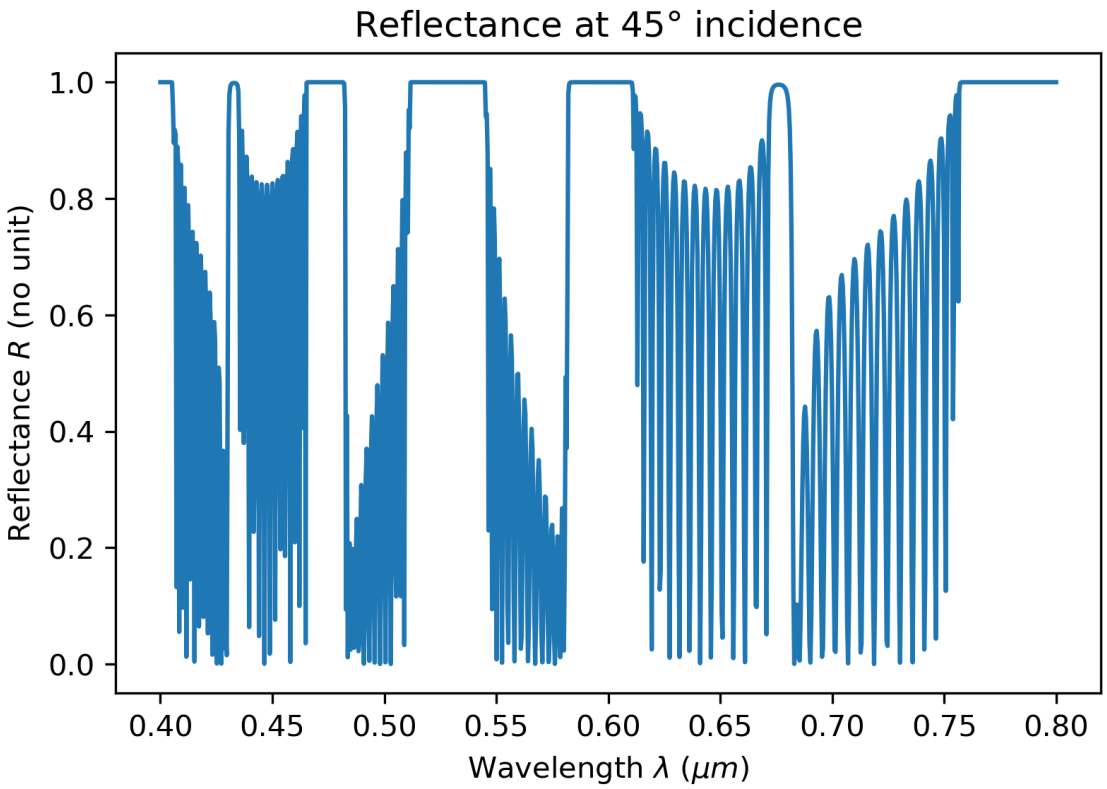


Figure 1.9: DBR reflectance at 45° incidence.

1.3.5 Angular-resolved spectra from a Distributed Bragg Reflector (DBR)

Now we want to create something like in Figure 1.10, i.e. plot the reflectance against incidence angle and wavelength:

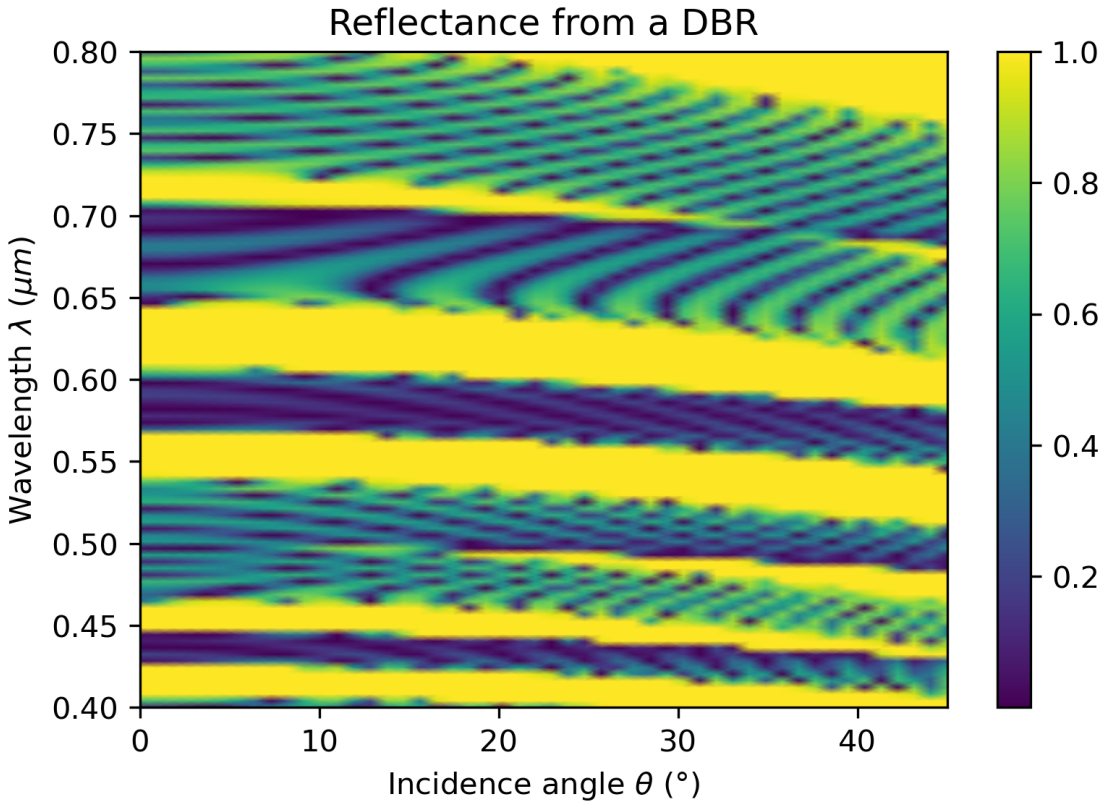


Figure 1.10: Plotting the reflectance against incidence angle and wavelength.

1. Open the file *KD6041-resources\TMM\DBR_TMM_2D.stub.py* in Spyder and save it as *DBR_TMM_2D.py*. It should look like this:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Import the main TMM function we will use:
from tmm.tmm_core import coh_tmm

# Import numpy:
import numpy as np
```

```
# Import matplotlib for plotting:
import matplotlib.pyplot as plt

def main():
    n1 = 3.5
    d1 = 0.5 # um
    n2 = 1.5
    d2 = 0.5 # um
    Nperiods = 15

    # list of layer thicknesses in nm
    d_list = [np.inf] + Nperiods*[d1, d2] + [d1] + [np.inf]
    # list of refractive indices
    n_list = [1] + Nperiods*[n1, n2] + [n1] + [1]

    ##### YOUR CODE GOES HERE #####

if __name__ == "__main__":
    main()
```

2. First we will define the desired input ranges, which this time also include an incidence angle in degrees *angle_deg*. Add the following after "*##### YOUR CODE GOES HERE #####*":

```
angle_deg = np.linspace(0,45)
wvl_list = np.linspace(0.400, 0.800, 100) # um
```

3. Then we create 2D arrays from them by using np.meshgrid():

```
angle_deg_2D, wvl_list_2D = np.meshgrid(angle_deg, wvl_list)
```

4. We also need to create a 2D array of the same size to store the reflectance, so we use the np.ones_like() function as follows:

```
R = np.ones_like(wvl_list_2D)*np.nan
```

What it does is create a 2D array R of the same size as *wvl_list_2D*, but filled with ones. By additionally multiplying by *np.nan* afterwards, we make sure that the array is initially filled with *NaN* values. This will allow you to more

easily detect any missing values in your plots if you make mistakes when filling the array.

5. We will also convert the angles from degrees to radians in advance by adding:

```
angle_rad_2D = np.deg2rad(angle_deg_2D)
```

6. To fill the array, we could use a double for loop using indices i and j and fill R using *R[i,j]=value*. But instead we will use a single loop using the np.ndenumerate() function and the index pairs it returns:

```
for idx, val in np.ndenumerate(wvl_list_2D):
  ret = coh_tmm('s', n_list, d_list, angle_rad_2D[idx], wvl_list_2D[idx])
  R[idx] = ret['R']
```

7. Now we add the code to create a 2D plot using *plt.pcolormesh*:

```
plt.pcolormesh(angle_deg_2D, wvl_list_2D, R, shading='gouraud')
```

The *Gouraud shading* option interpolates the values between the points for which reflectance was calculated to color the 2D surface of the plot. If you want to understand this better, have a look here:

https://matplotlib.org/stable/gallery/images_contours_and_fields/pcolormesh_grids.html

You can also try out the file *KD6041-resources\TMM\pcolormesh_example.py*.

8. And finally we add a colorbar, labels and a title:

```
plt.colorbar()
plt.ylabel(r'Wavelength $\lambda$ ($\mu{}m$)')
plt.xlabel(r'Incidence angle $\theta$ ($\degree$)')
plt.title('Reflectance from a DBR')
```

9. If you run the code now, you should get the same as in Figure 1.10.

1.4 Homework

1.4.1 Design a DBR for >90% reflection at normal incidence from 550 to 650nm

We want to adapt our DBR design, so that it reflects more than 90% ($R > 0.90$) of light coming in at normal incidence ($\theta = 0°$) for wavelengths in the range $\lambda_{min} = 550nm$ to $\lambda_{max} = 650nm$ using a single material to make the DBR, i.e. the other material will be air ($n_2 = 1$).

Using $\lambda_0 = \frac{\lambda_{min}+\lambda_{max}}{2}$, we fix some of the parameters as follows:

- $n_2 = 1$
- $d_1 = \frac{\lambda_0}{4n_1}$
- $d_2 = \frac{\lambda_0}{4n_2}$

Figure 1.11 summarizes these infos.

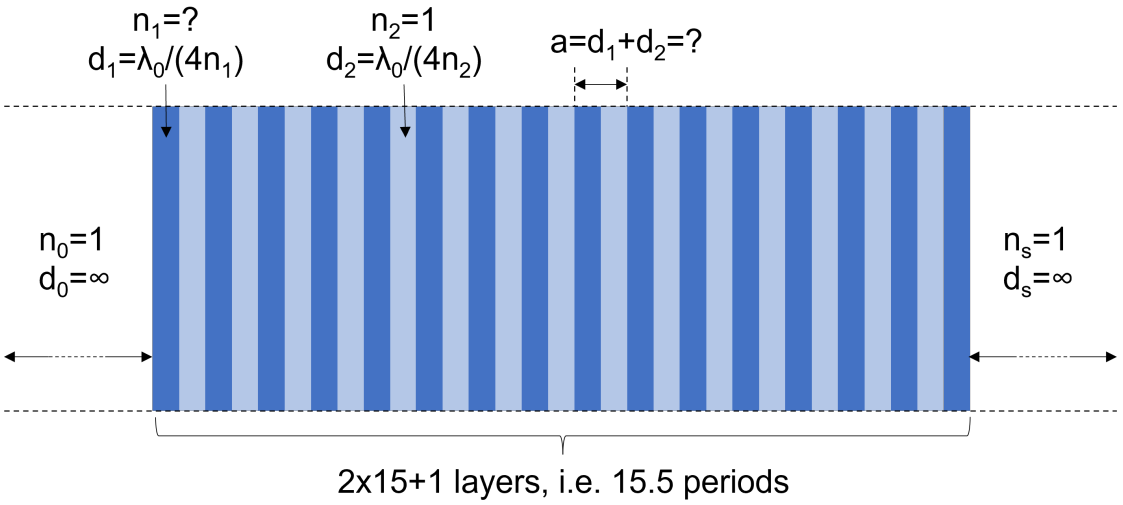


Figure 1.11: Parameters of the DBR.

1. What are the values of λ_0 and d_2?
2. What is the value of n_1 as a function of n_2 and $a = d_1 + d_2$?
3. Create a 2D plot of the reflectance R as a function of the period $a = d_1 + d_2$ and wavelength λ over the ranges $a = 250 - 300nm$ and $\lambda = \lambda_{min} - \lambda_{max}$.
4. Due to nanofabrication limitations, we would like to use the largest possible value for a that satisfies the requested design. What is the largest period a we can use?
5. What are the corresponding values for n_1 and d_1?

1.4.2 Design an anti-reflection coating

An anti-reflection coating consists of a thin layer of material deposited onto a substrate as illustrated in Figure 1.12, with a refractive index n_1 and thickness d_1 chosen so that the reflection at normal incidence is zero for a specific wavelength λ_0 thanks to:

- minimizing the overall reflectance cause by the index contrasts at each interface,
- and to destructive interference on reflection of the waves coming from the first (n_0 to n_1) and second (n_1 to n_s) interface.

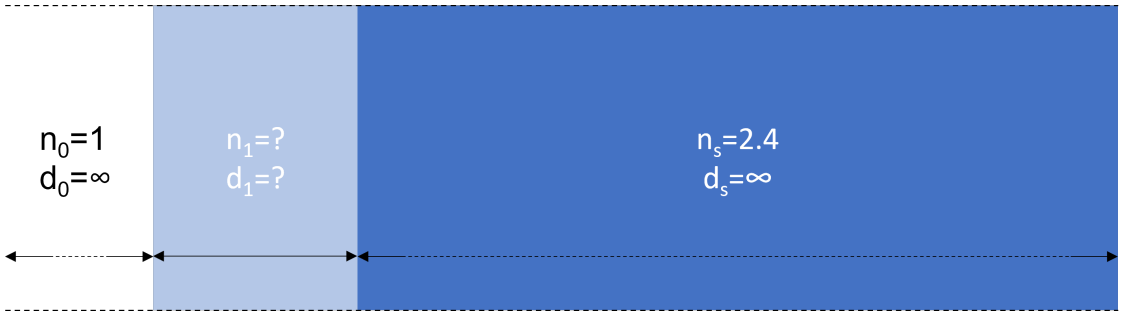


Figure 1.12: The basic design of an anti-reflection coating.

Design an optimal anti-reflection coating for the following parameters:

- Incident wavelength: $\lambda_0 = 637nm$
- Refractive index of the medium the light comes from: $n_0 = 1$
- Refractive index of the medium the light goes into: $n_s = 2.4$

2 Appendix

2.1 Preliminary software setup for personal computers

Section §2.2 is designed for the PCs in ELC103, where most of the software is already set up. To be able to apply them on a personal computer, you will first need to install the following software:

- Notepad++: https://notepad-plus-plus.org/
- Anaconda: https://www.anaconda.com/
- Git bash: https://git-scm.com/download/win
- Cygwin (with MPB and MEEP): The installation of Cygwin will be covered in separate lecture slides.
- Matlab (or GNU Octave)

> **!** If you are using a system other than *Microsoft Windows*, such as *GNU/Linux* or *macOS*, you can contact us if you need help setting things up.

2.2 Setting things up

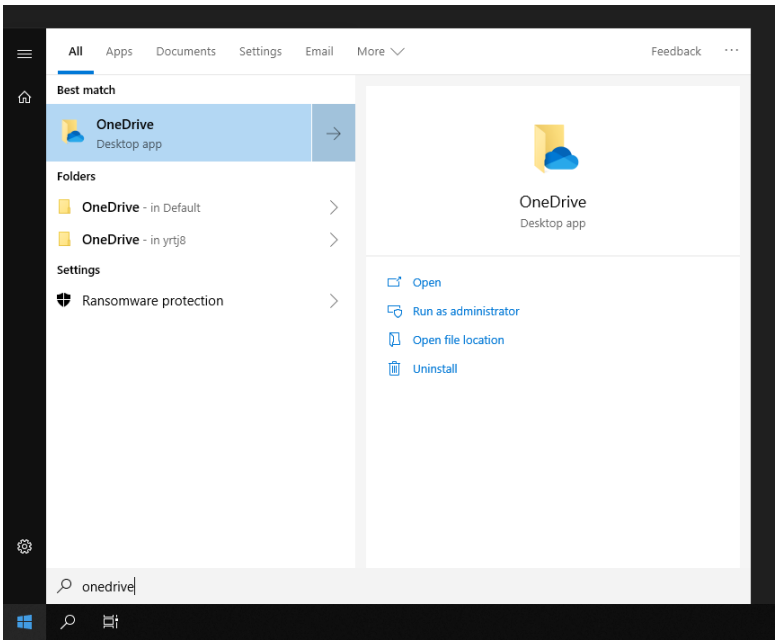
Here are the main setup steps to perform before getting started:

1. Set up OneDrive *(to do the first time you use a new lab PC)*
2. Create a work directory in your OneDrive *(only needs to be done once)*
3. Get the module resources *(only needs to be done once)*
4. Define environment variables *(to do the first time you use a new lab PC)*
5. Install the Python TMM module *(to do the first time you use a new lab PC)*
6. Test the MIT tools *(optional)*

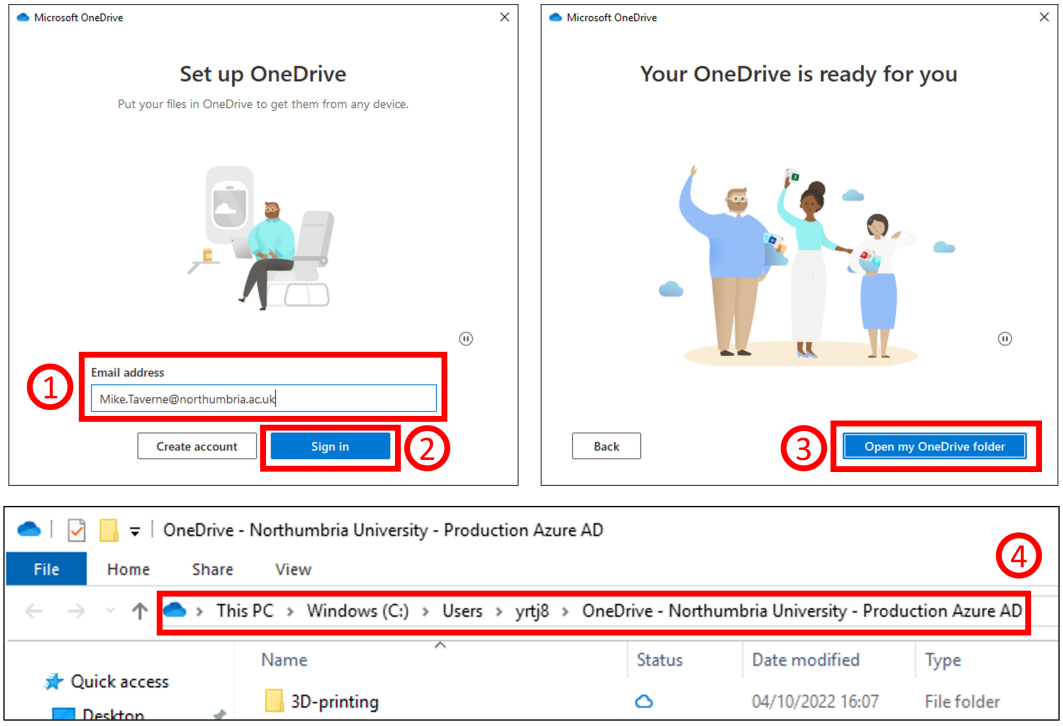
2.2.1 Set up OneDrive

> **!** If you do not set up OneDrive and save your work to it, you risk losing it, as anything on the lab PCs outside the OneDrive can be wiped after logging out or rebooting!

1. Open the *OneDrive* application:



2. Enter your e-mail address (the "long one" of the form *First.Last@northumbria.ac.uk*). Then click *"Sign in"*, follow the instructions, click through the information dialogs and finally click on *"Open my OneDrive folder"* to make sure it worked:

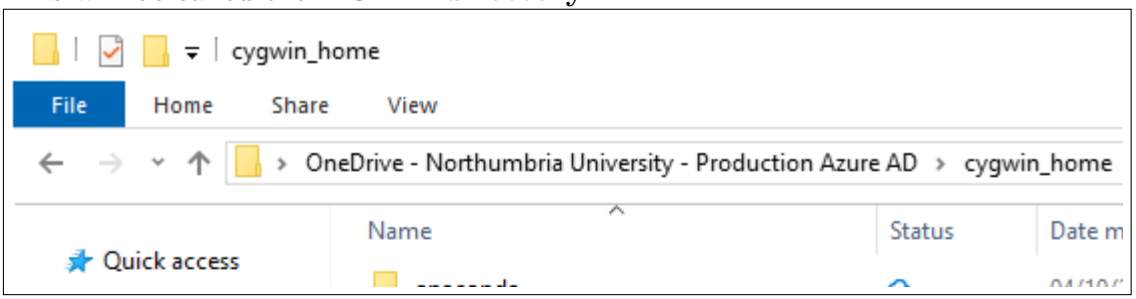


2.2.2 Create a work directory in your OneDrive

Just create a new *folder* (also called *directory*) named *"cygwin_home"* at the root of your OneDrive folder, i.e. the full path should be of the form:

```
%USERPROFILE%\OneDrive - Northumbria University - Production Azure AD\cygwin_home
```

This will be called the **HOME directory**.

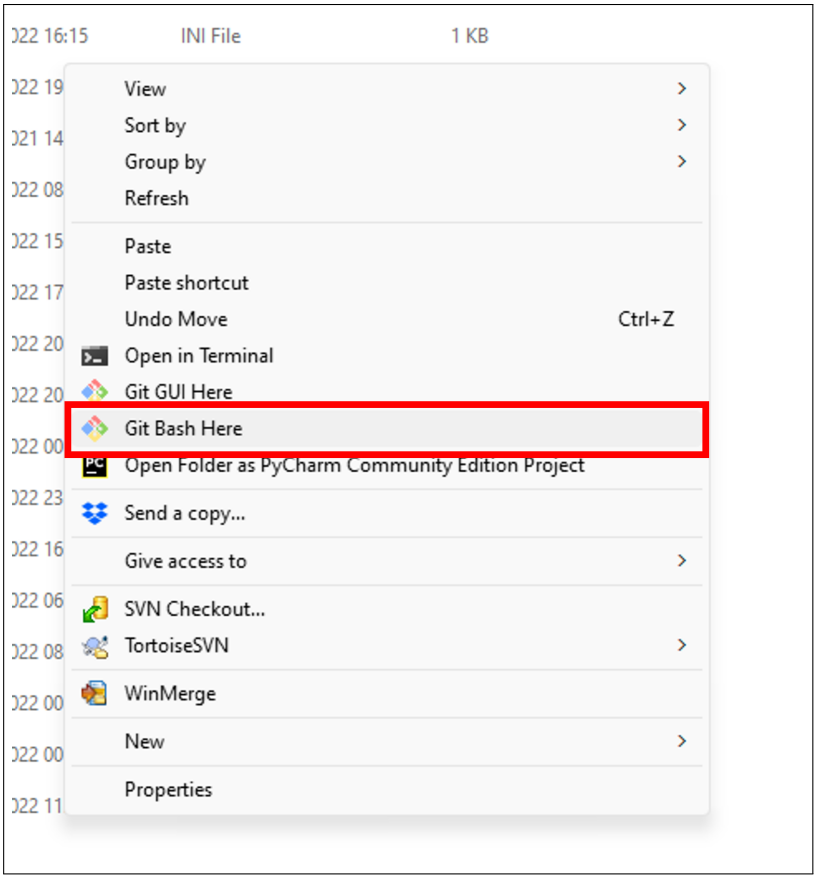


2.2.3 Get the module resources

The purpose of this guide is to download some extra resources that will be used in the module to your computer.

1. Open the *Windows File Explorer* and navigate to the *HOME directory* you created in section 2.2.2.

2. Open a context-menu by right-clicking in the File Explorer away from any files and select *"Git Bash Here"*:



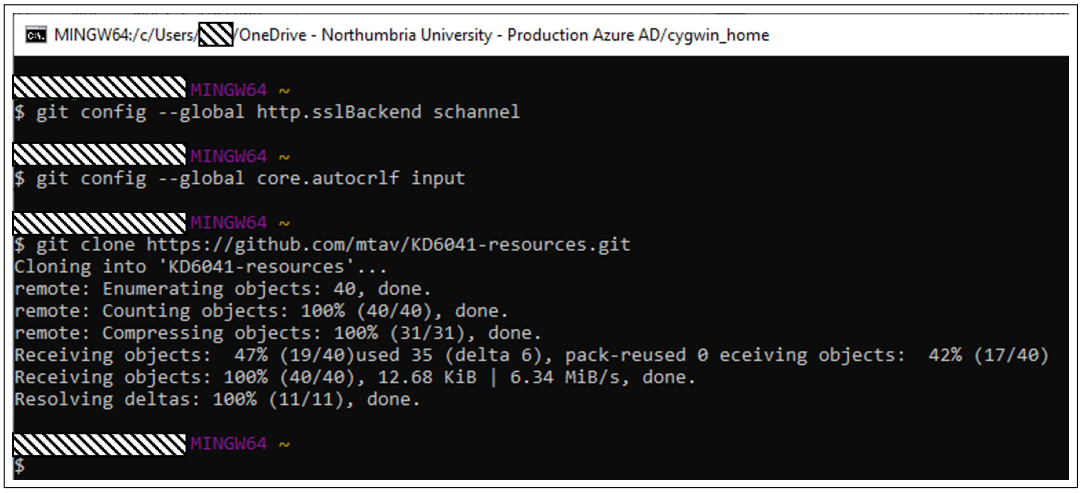
3. You now have a *Git Bash prompt*. To get the resources for this module, run the following commands in it **(one by one!)**:

```
git config --global http.sslBackend schannel
```

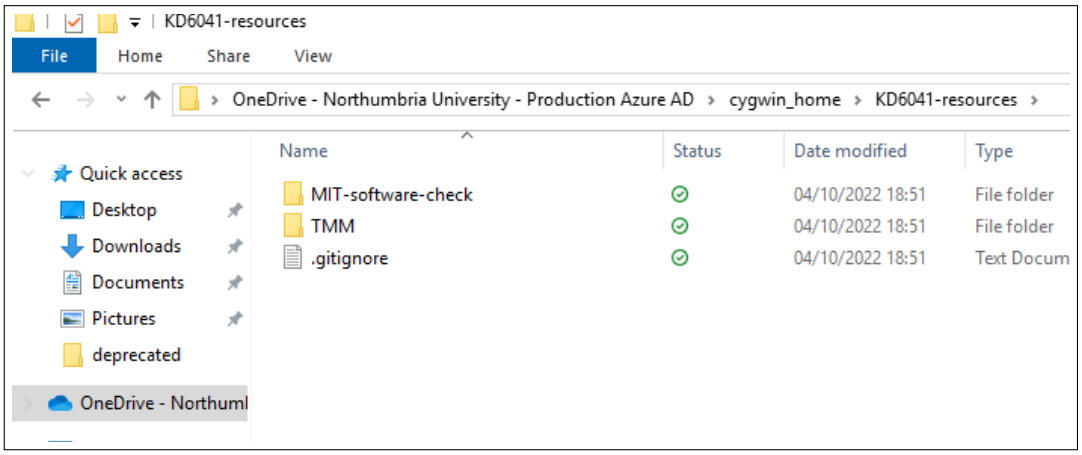
```
git config --global core.autocrlf input
```

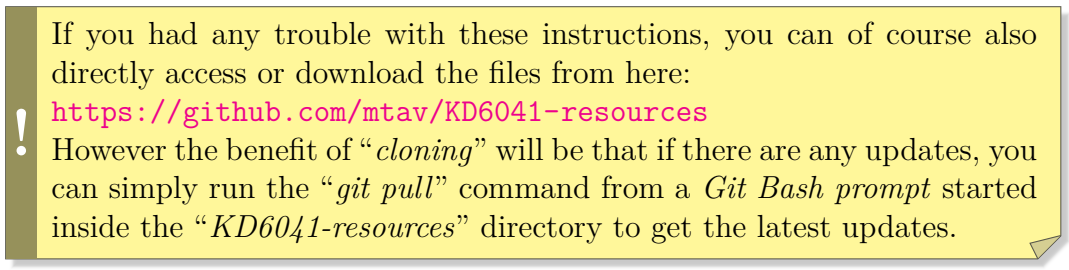
```
git clone https://github.com/mtav/KD6041-resources.git
```

> **!** To paste into the *Git Bash prompt*, use **right-click** → **paste**, as the *ctrl-V* shortcut may not work!



4. You should now see a new *"KD6041-resources"* directory in your *HOME* directory:



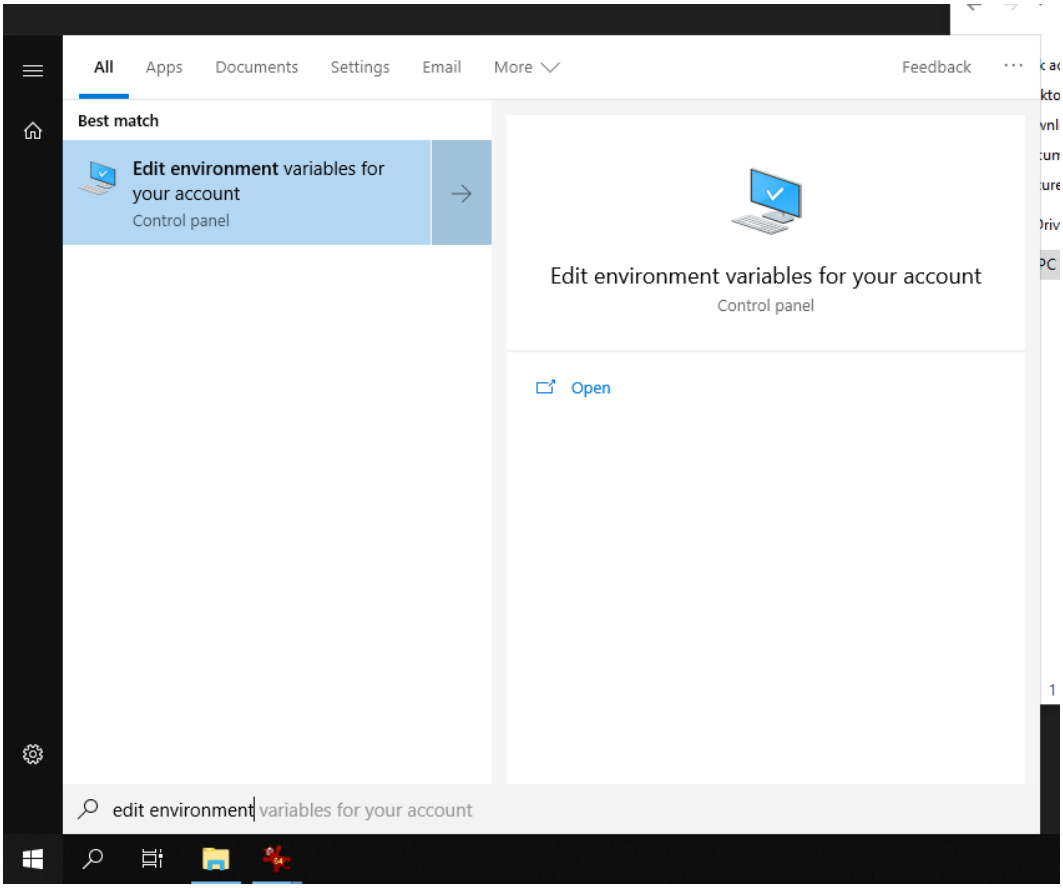


If you had any trouble with these instructions, you can of course also directly access or download the files from here:
https://github.com/mtav/KD6041-resources
However the benefit of "*cloning*" will be that if there are any updates, you can simply run the "*git pull*" command from a *Git Bash prompt* started inside the "*KD6041-resources*" directory to get the latest updates.

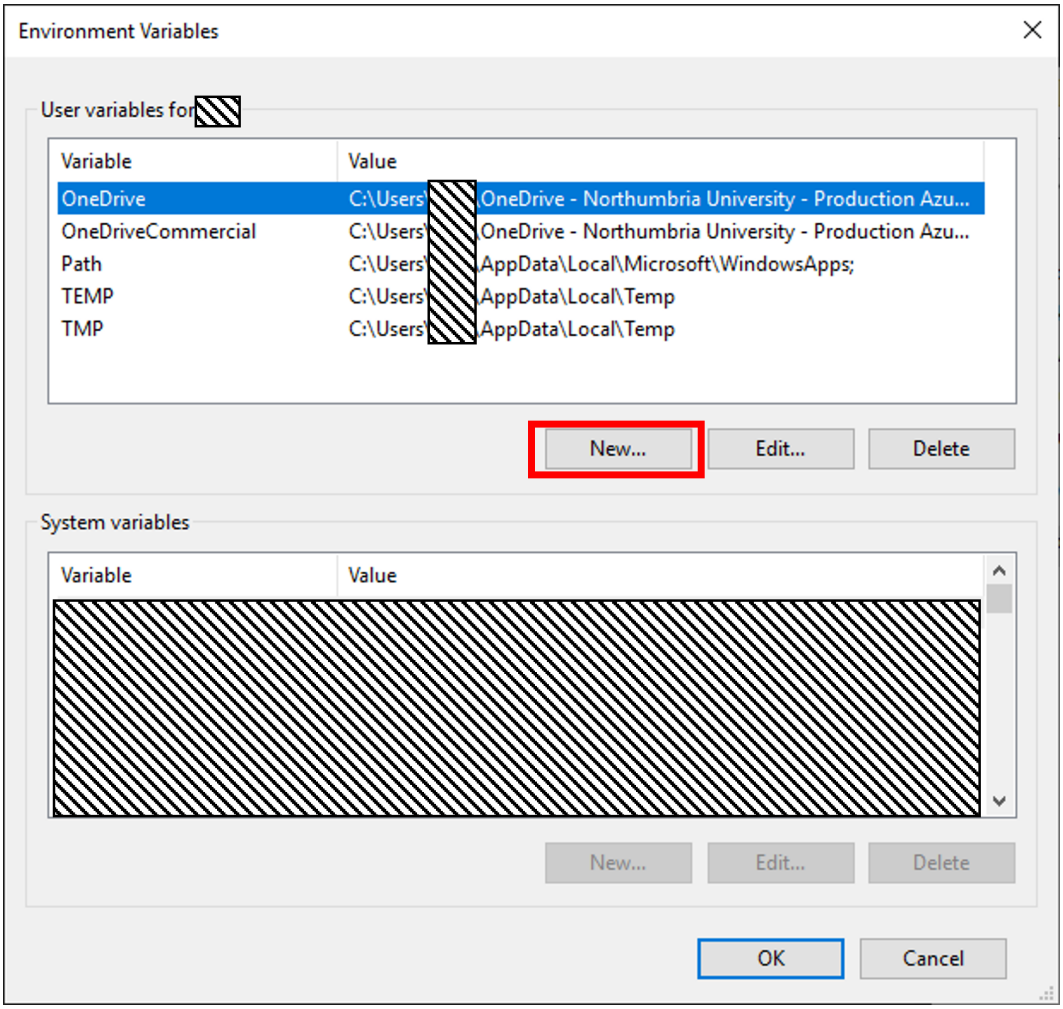
2.2.4 Define environment variables

Since we will be working a lot from the command-line using the *Cygwin terminal*, it is useful to define an *environment variable* called *HOME* that defines your default *work directory*. This will make sure that whenever you start *Cygwin*, it will start in that directory. At the same time, we will also define the *MATLABPATH* environment variable, to make it easier to load some utility functions from the module resources.

1. Open the *"Edit environment variables for your account"* dialog:



2. Click on “New...”:



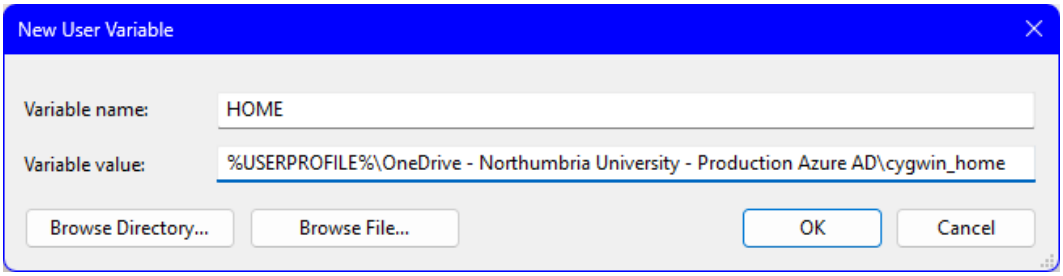
3. Enter the following in the new dialog that appears, then click “OK”.

   Variable name:

   ```
   HOME
   ```

   Variable value:

   ```
   %USERPROFILE%\OneDrive - Northumbria University - Production Azure AD\cygwin_home
   ```



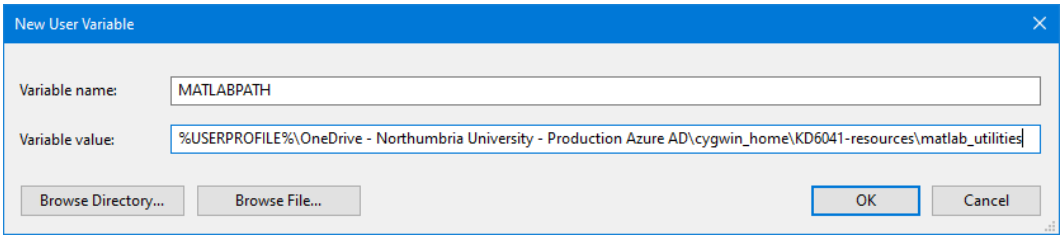
4. Click on “New...” again like in step 2. Then enter the following in the new dialog that appears, then click “OK”.

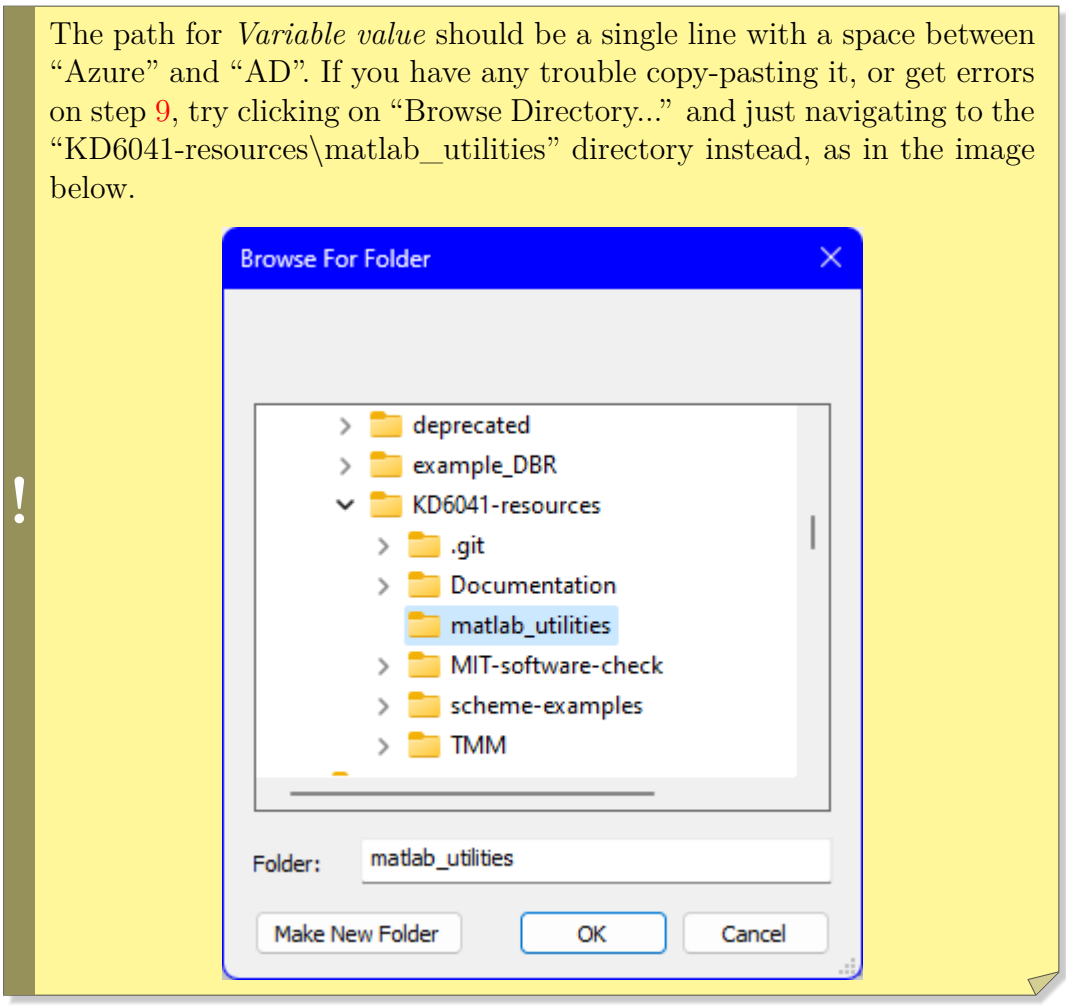
   Variable name:

   ```
   MATLABPATH
   ```

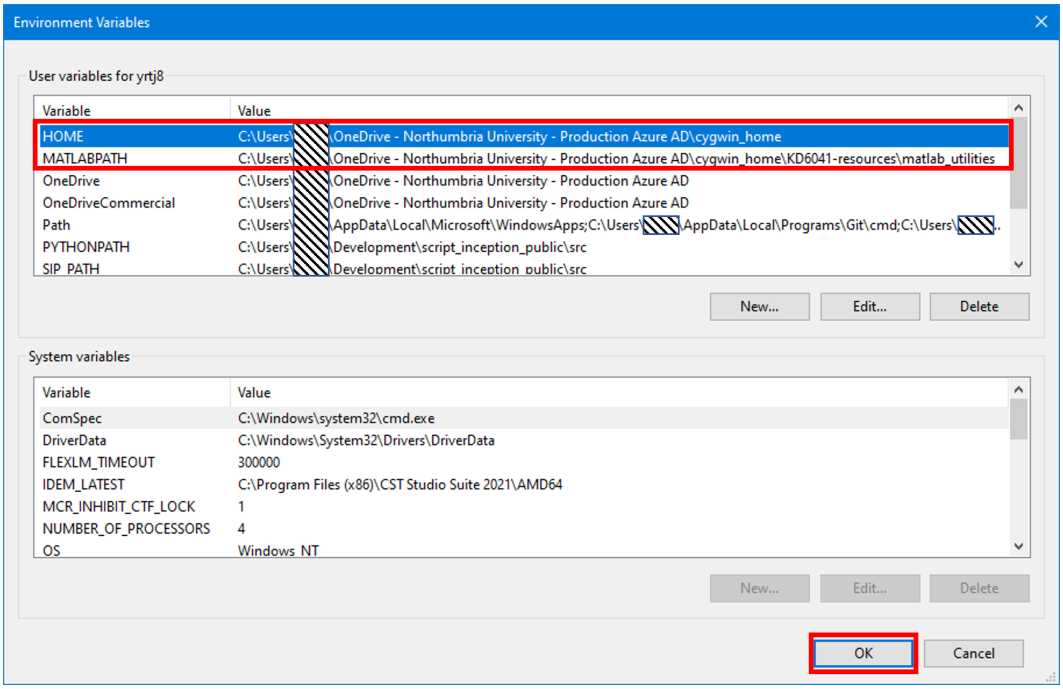
   Variable value *(This should be on one line)*:

   ```
   %USERPROFILE%\OneDrive - Northumbria University - Production Azure
       AD\cygwin_home\KD6041-resources\matlab_utilities
   ```





5. Make sure *HOME* and *MATLABPATH* are defined as in the following figure, then click OK again to close the initial dialog:



6. To test that that the *HOME* setup worked, open *Cygwin:*

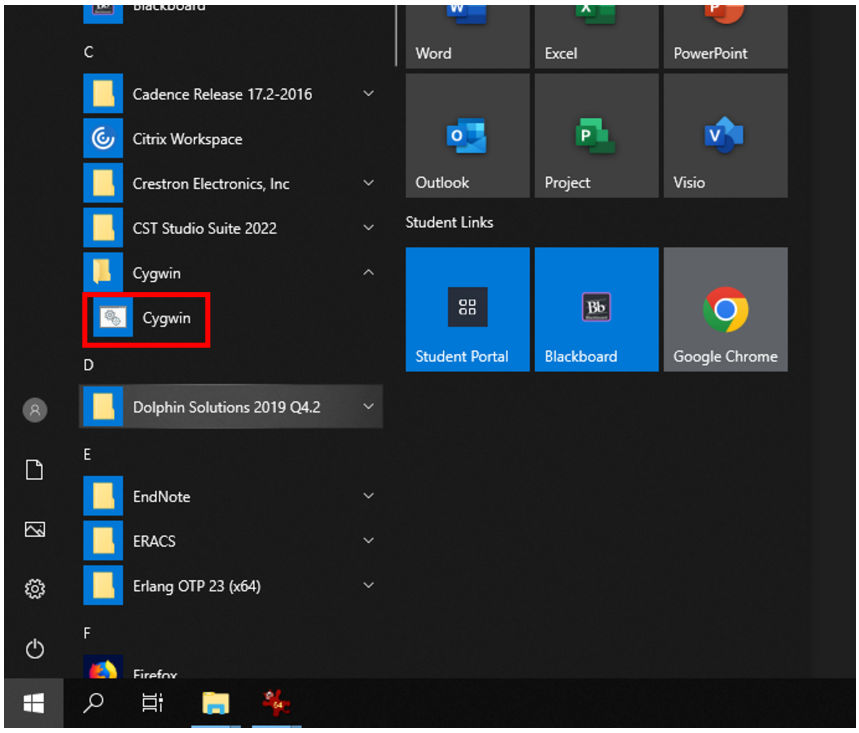
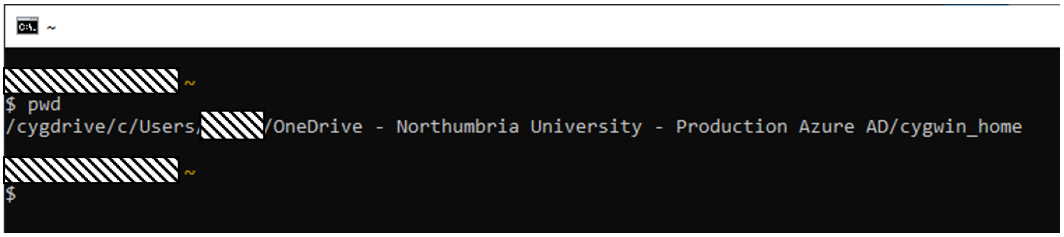


Figure 2.1: Opening Cygwin.

7. Now run the following command in it and check that the output corresponds to the HOME directory you defined:

```
pwd
```

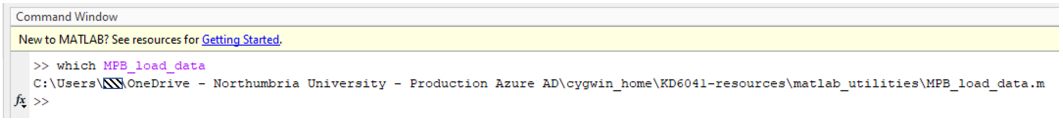


> ! *pwd* stands for *"Print Working Directory"* and is very useful to know where you currently are in the filesystem when using a terminal. In Cygwin, the Windows *"C:"* drive is mapped to *"/cygdrive/c"* and instead of backslashes ("\"), it uses forward slashes ("/") in path names.

8. To test that the *MATLABPATH* setup worked, open *Matlab*, then run the following command in it:

```
which MPB_load_data
```

9. Check that your output looks like this:



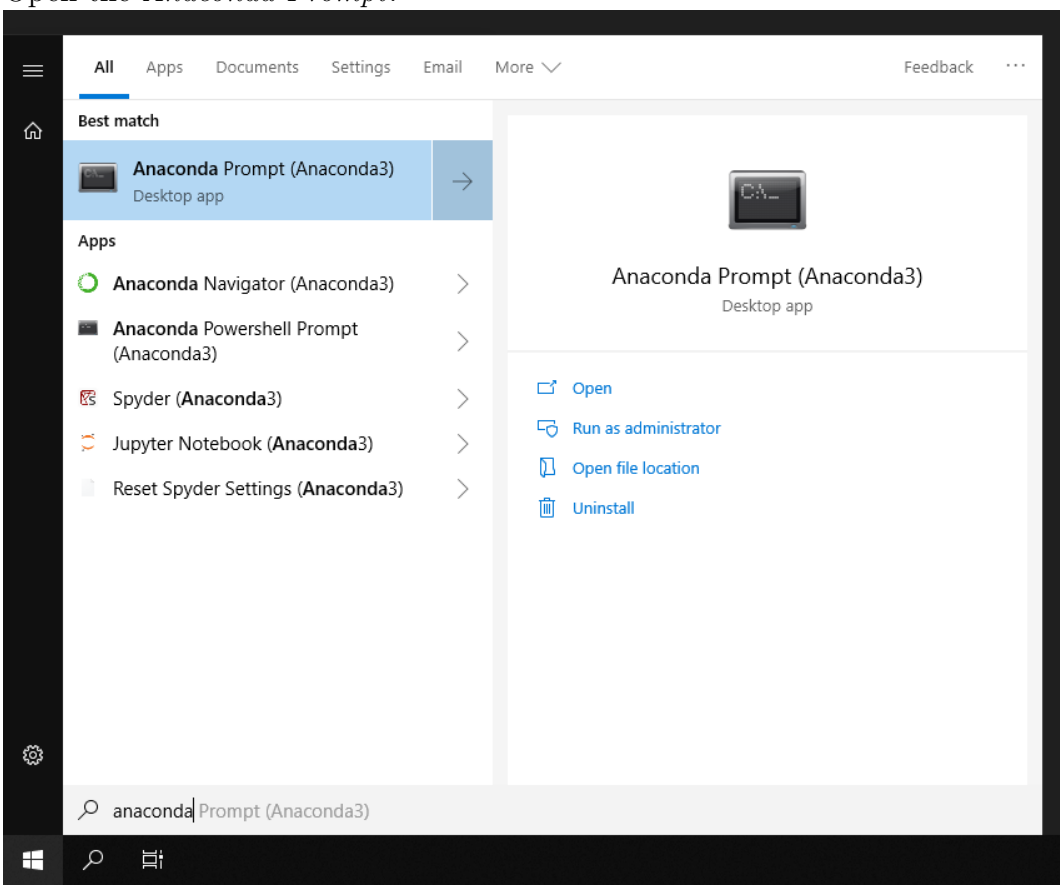
If you get *"'MPB_load_data' not found"* instead, something went wrong. Check the contents of the *MATLABPATH* at step 4 again.

2.2.5 Install the Python TMM module

We will use the Python TMM module to run TMM calculations. You can find its documentation here: https://pythonhosted.org/tmm/tmm.html.

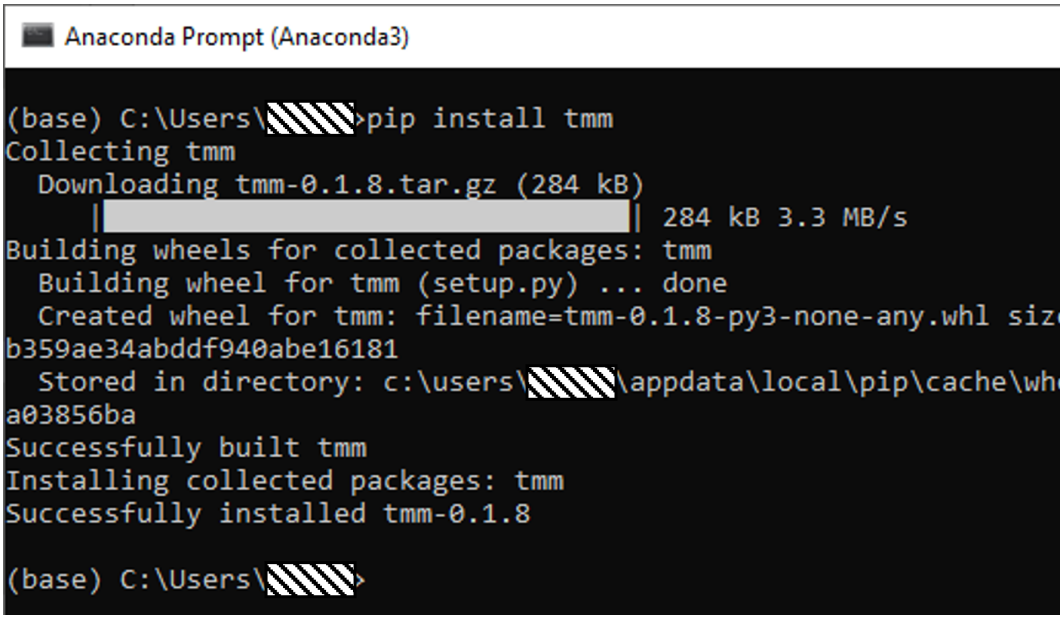
This is a Python module. If you want to follow these installation instructions at home, you will have to install the Python distribution *Anaconda* first. You can get it from here: https://www.anaconda.com/.

1. Open the *Anaconda Prompt*:



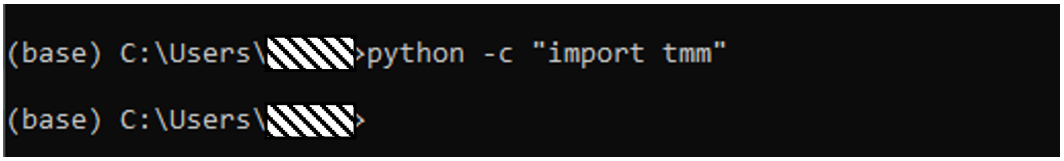
2. Enter the following command in it:

```
pip install tmm
```



3. To test the installation, you can run the following command in the same prompt:

```
python -c "import tmm"
```

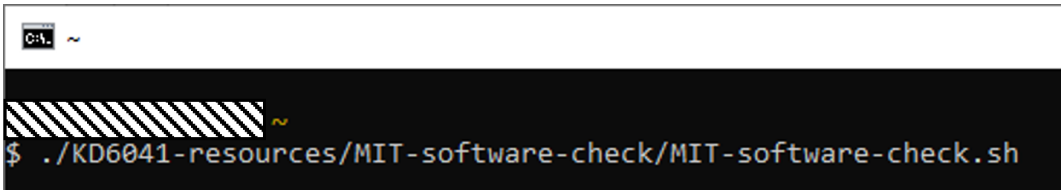


2.2.6 Test the MIT tools

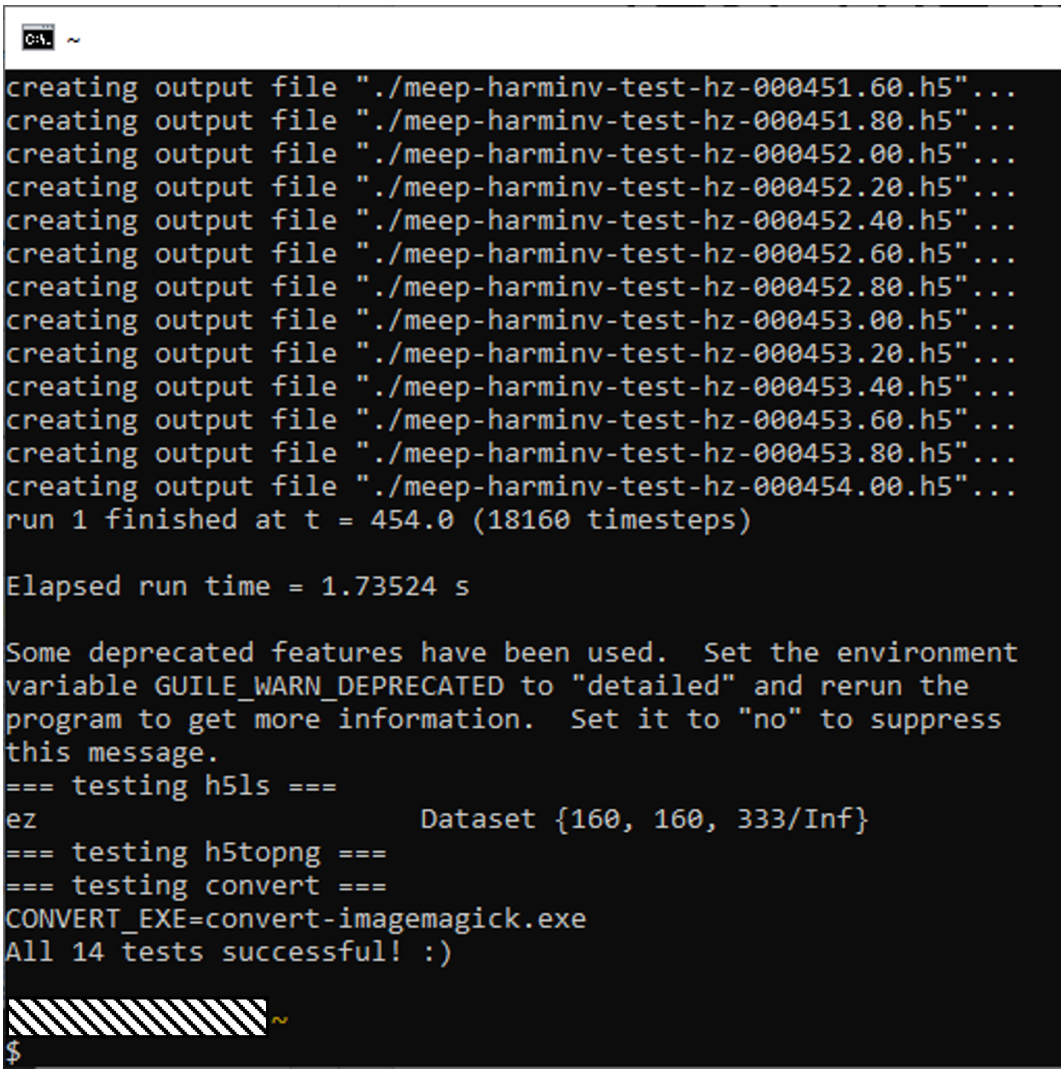
1. Open a *Cygwin terminal* as you did in step 6 of section 2.2.4.

2. Run the following command in it:

```
bash ./KD6041-resources/MIT-software-check/MIT-software-check.sh
```



3. When it completes, it should look like this if all tests passed successfully:



2.3 Getting started with the Python TMM module

1. Open the Python Integrated Development Environment (IDE) *Spyder*:

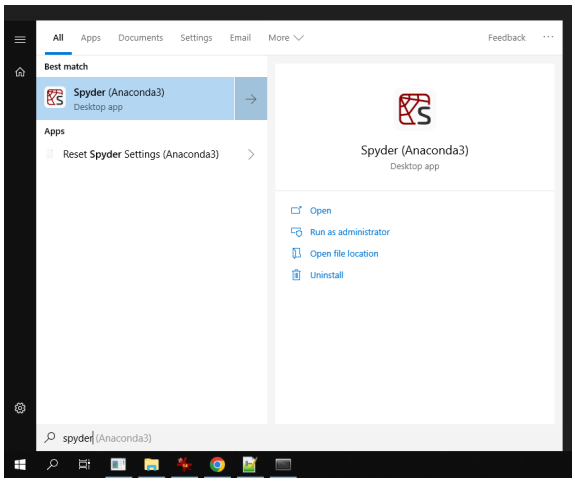


Figure 2.2: Opening *Spyder*.

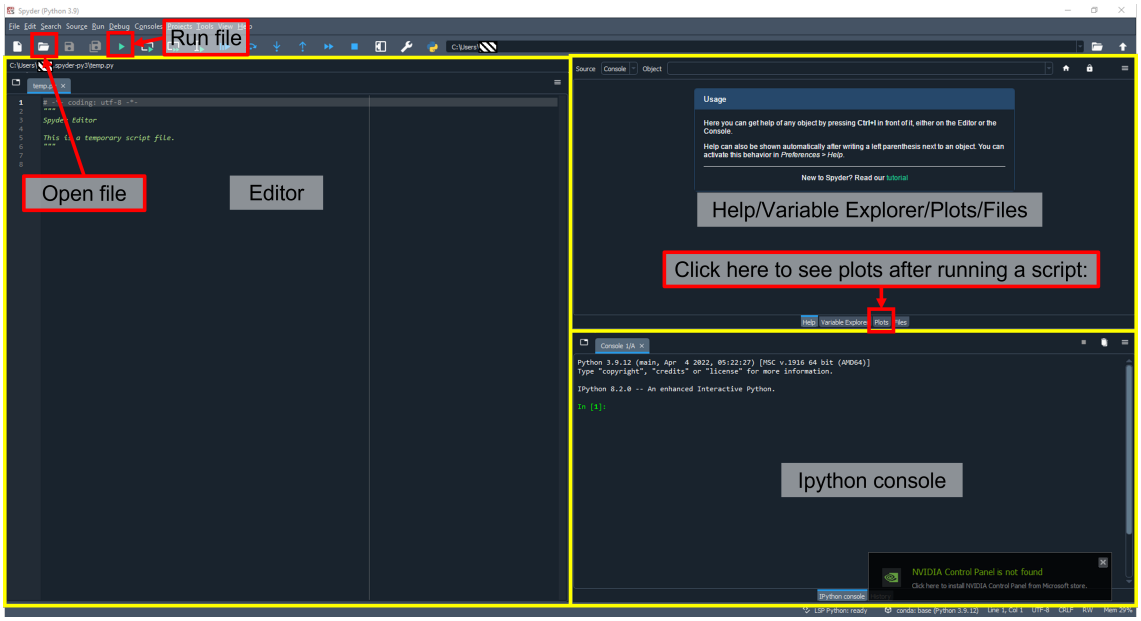
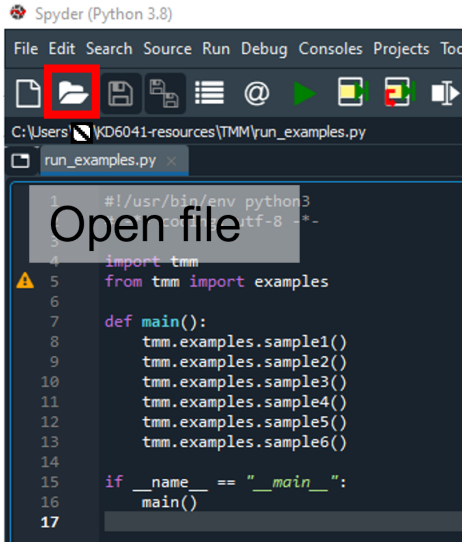
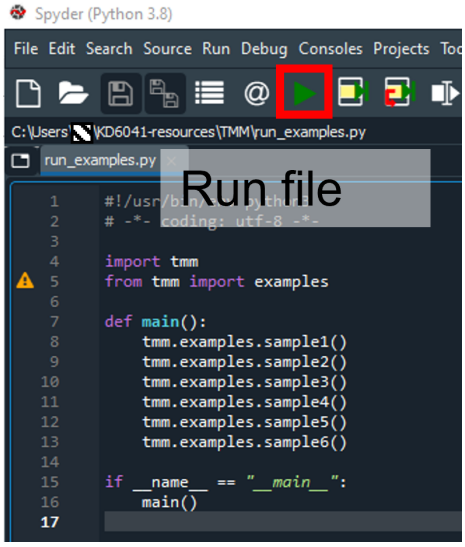


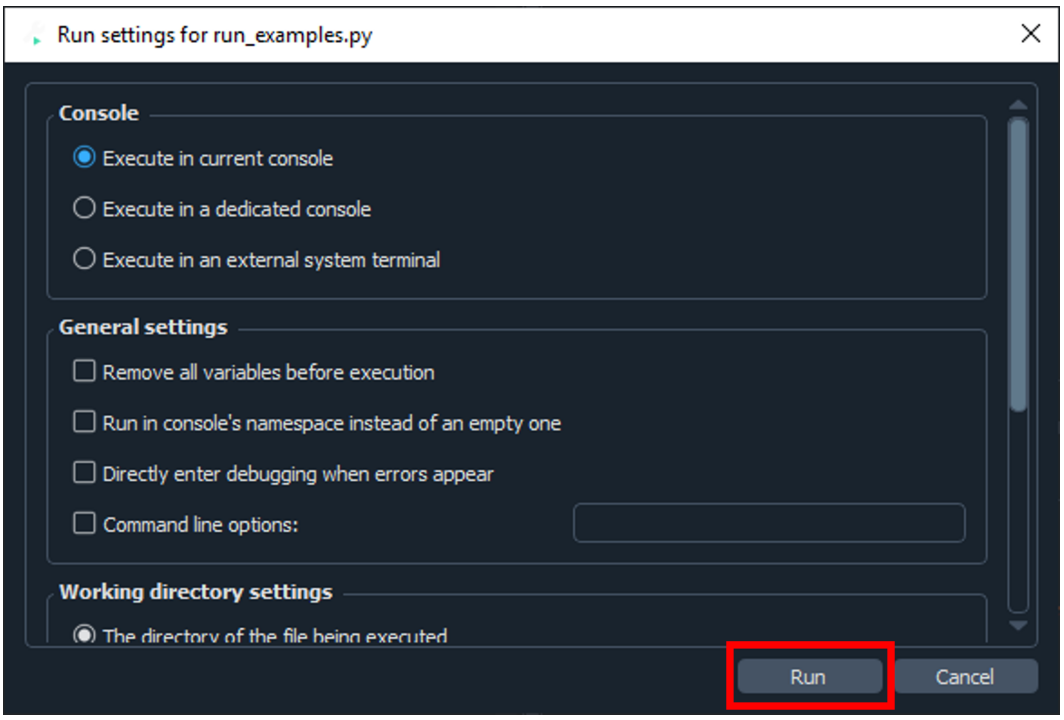
Figure 2.3: The *Spyder* interface.

2. Open the script *"KD6041-resources\TMM\run_examples.py"* in Spyder and then click on the "Run file" button:

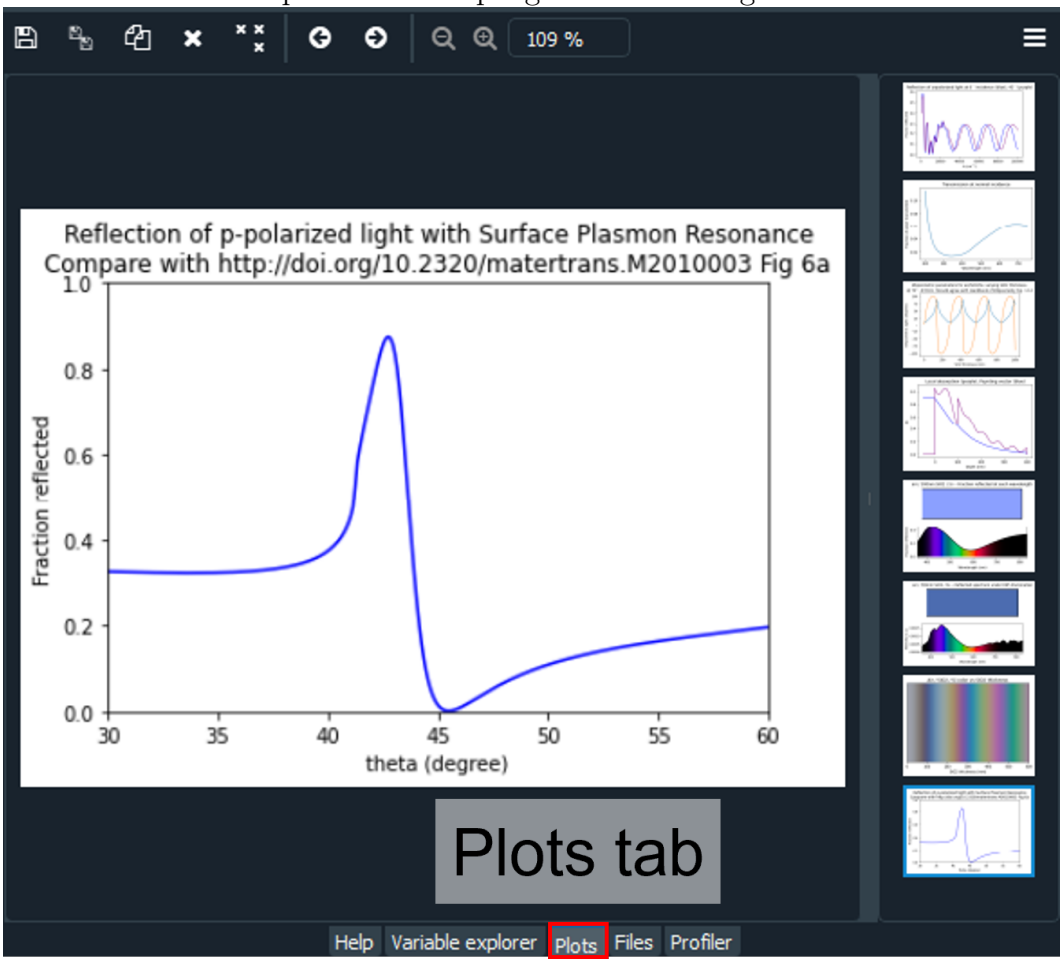




3. If you get a "Run settings" window, just leave the defaults and click on "Run":



4. You should now see plots in the top right after clicking on the “Plots” tab:



2.4 Introduction to Scheme

2.4.1 Introduction

Both MPB [2] and MEEP [3] use a scripting language called Scheme. Scheme is a simplified derivative of Lisp, and is a small and beautiful dynamically typed, lexically scoped, functional language.

2.4.2 Using Scheme

2.4.2.1 Setting up Notepad++

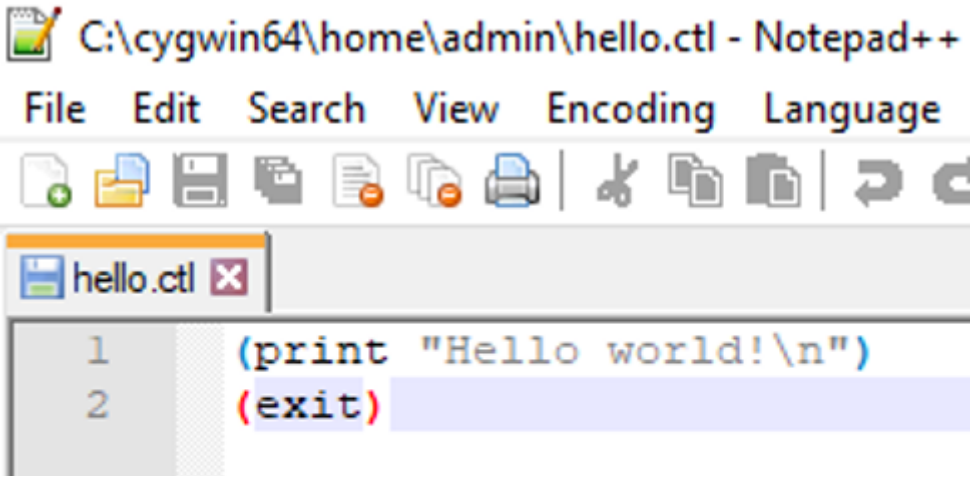
On Windows, a good text editor for Scheme is Notepad++. It is installed on the lab PCs already, but if you are using your own computer, you can get it here:
https://notepad-plus-plus.org/
See *KD6041-resources\Documentation\notepad++_tips.pdf* for a recommended setup.

2.4.2.2 Getting started

1. Open *Notepad++*.
2. Create a new file in it named *"hello.ctl"* and save it in your *work directory*.
3. Enter the following code in it:

```
(print "Hello world!\n")
(exit)
```

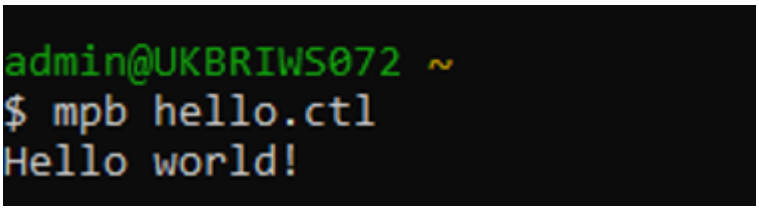
Listing 1: Also available from KD6041-resources\scheme-examples\hello.ctl



4. Now open a *Cygwin terminal* (see figure 2.1) and run the following command in it:

```
mpb hello.ctl
```

5. You should see something like this after running it:



2.4.2.3 Where to find more information on Scheme

- An introduction to the basics of using Scheme and MPB: KD6041-resources\Documentation\scheme+mpb_basics.pdf

- Some example scripts: KD6041-resources\scheme-examples

- A Scheme tutorial: `http://ds26gte.github.io/tyscheme/index.html`

  ! When using this tutorial:

  – Instead of *mzscheme*, use the command *mpb* in Cygwin.

  – Instead of *mzscheme -r hello.scm*, use *mpb hello.scm*.

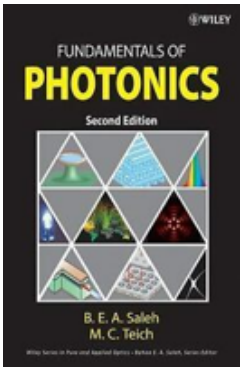
  – Note that due to some bug, every ending double quote, in the document, is printed twice. So instead of *"Hello, World!""*, use *"Hello, World!"*, instead of *(load "hello.scm"")*, use *(load "hello.scm")*, etc.

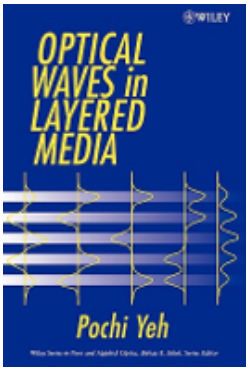
- Useful links, as well as some useful tips and tricks: `https://mpb.readthedocs.io/en/latest/Guile_and_Scheme_Information/`

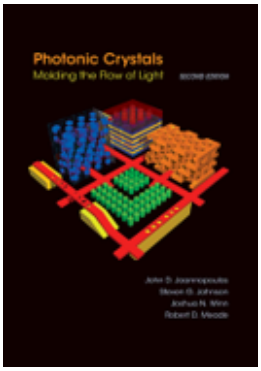
- A Scheme/MPB tutorial: `https://mpb.readthedocs.io/en/latest/Scheme_Tutorial/`

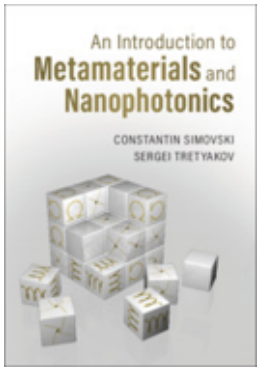
2.5 Useful references

- **Fundamentals of photonics**, Bahaa E. A. Saleh, Malvin Carl Teich [4]
  - Library link: https://librarysearch.northumbria.ac.uk/permalink/f/1t01hd3/44UON_ALMA51117773960003181
- **Optical Waves in Layered Media**, Pochi Yeh [1]
  - Library link: https://librarysearch.northumbria.ac.uk/permalink/f/1t01hd3/44UON_ALMA21117968020003181
- **Photonic crystals: molding the flow of light**, John D. Joannopoulos, Steven G. Johnson, Joshua N. Winn, and Robert D. Meade [5]
  - Available for free at: http://ab-initio.mit.edu/book
  - Library link: https://librarysearch.northumbria.ac.uk/permalink/f/1t01hd3/44UON_ALMA21117688410003181
- **An Introduction to Metamaterials and Nanophotonics**, Constantin Simovski, and Sergei Tretyakov (Hardcover – October 15, 2020) [6]
  - DOI link: https://doi.org/10.1017/9781108610735









Bibliography

[1] P. Yeh, *Optical waves in layered media*, ser. Wiley series in pure and applied optics. Wiley New York, 1988, vol. 95. [Online]. Available: http://books.google.com/books?id=-yZBAQAAIAAJ

[2] "MPB: MIT Photonic Bands." [Online]. Available: http://ab-initio.mit.edu/wiki/index.php/MPB

[3] "MEEP: MIT Electromagnetic Equation Propagation." [Online]. Available: http://ab-initio.mit.edu/wiki/index.php/Meep

[4] B. E. A. Saleh and M. C. Teich, *Fundamentals of Photonics*, ser. Wiley Series in Pure and Applied Optics. Wiley, 2007. [Online]. Available: https://books.google.co.uk/books?id=Ve8eAQAAIAAJ

[5] J. D. Joannopoulos, S. G. Johnson, J. N. Winn, and R. D. Meade, *Photonic Crystals: Molding the Flow of Light*, 2nd ed. Princeton University Press, 2008. [Online]. Available: http://ab-initio.mit.edu/book/

[6] C. Simovski and S. Tretyakov, *An introduction to metamaterials and nanophotonics*. Cambridge University Press, 2020. [Online]. Available: https://doi.org/10.1017/9781108610735