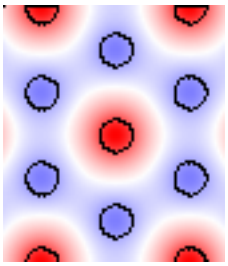
Northumbria University
NEWCASTLE

KD6041

Photonic crystal simulations



Module Handbook

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: September 17, 2023

Contents

1 Setting things up and getting started

1.1 Setting things up

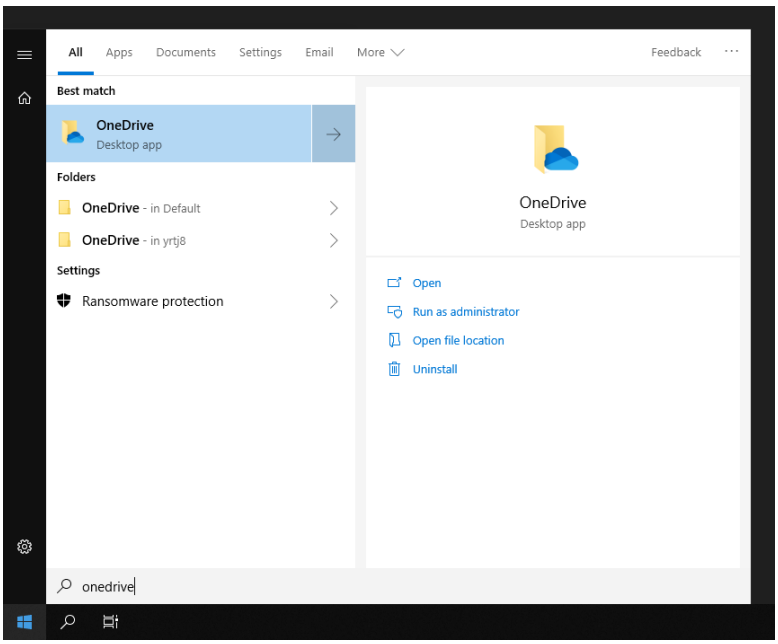
Here are the main setup steps to perform before getting started:

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)*

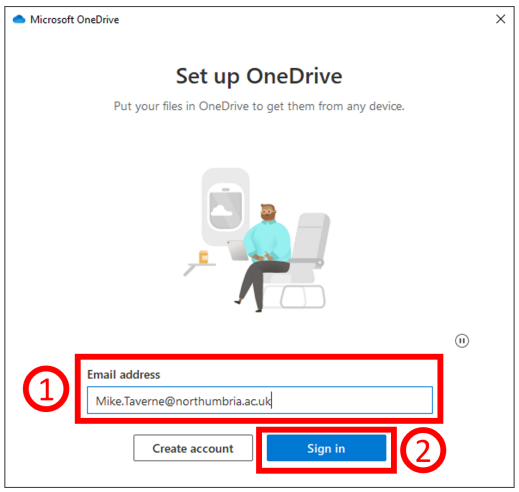
1.1.1 Set up OneDrive

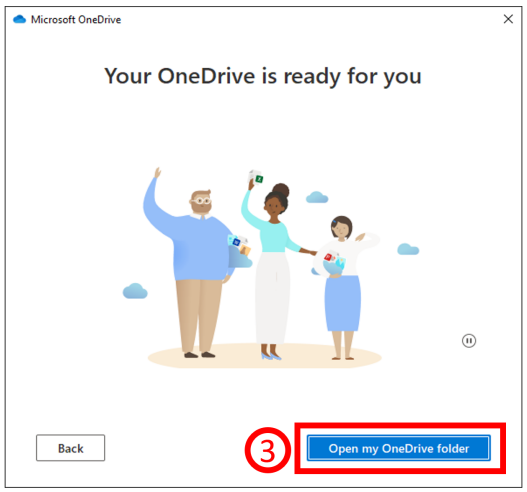
> **!** If you do not set up OneDrive and save your work to it, you risk losing it, as anything on the lab PCs outside the OneDrive can be wiped after logging out or rebooting!

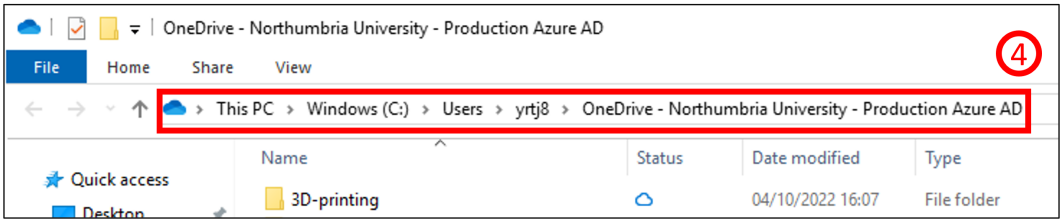
1. Open the *OneDrive* application:



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:





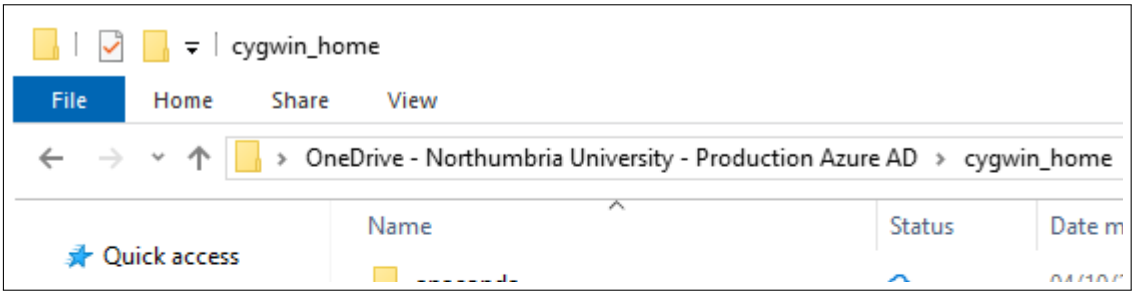


1.1.2 Create a work directory in your OneDrive

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**.

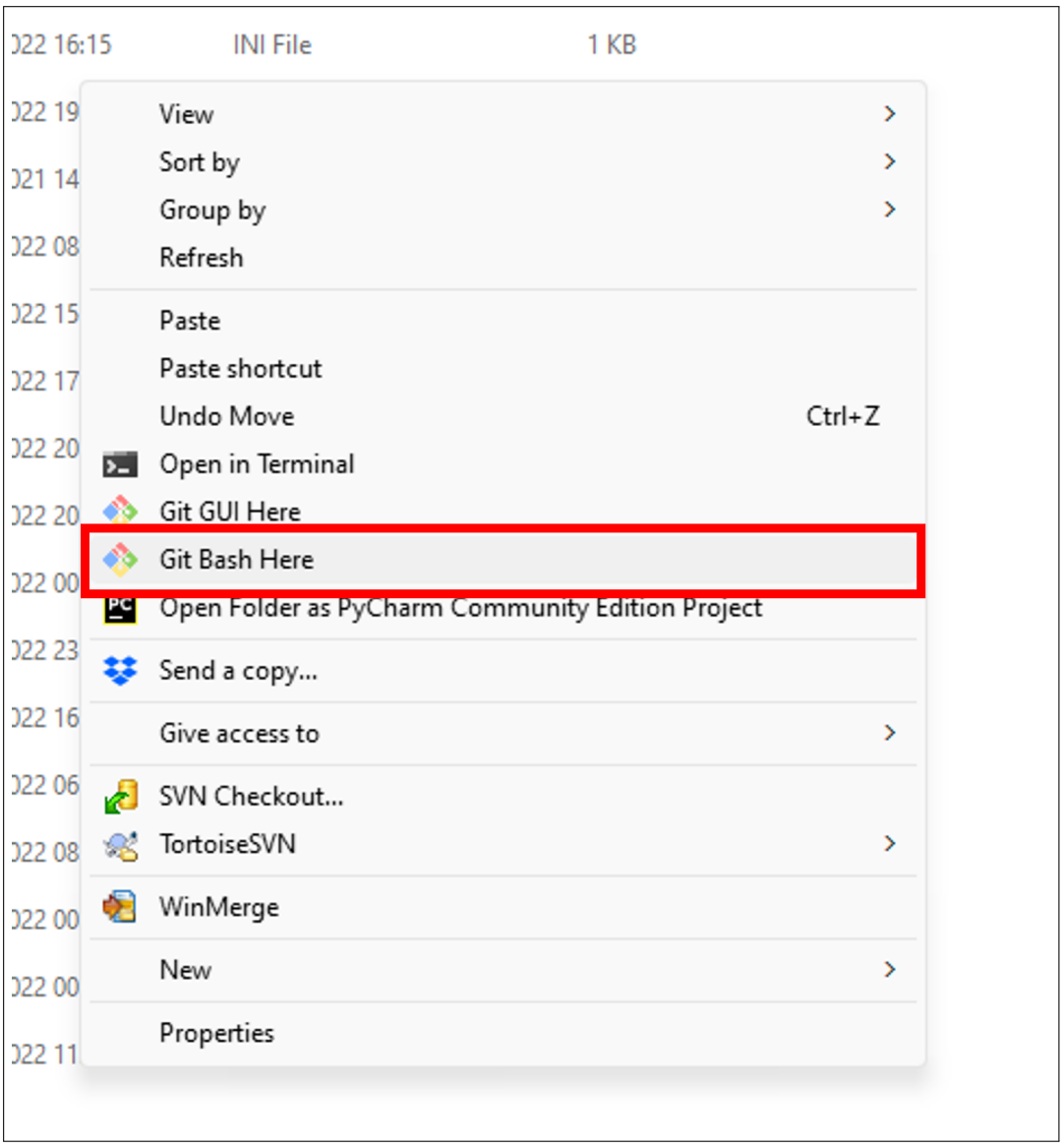


1.1.3 Get the module resources

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 1.1.2.

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



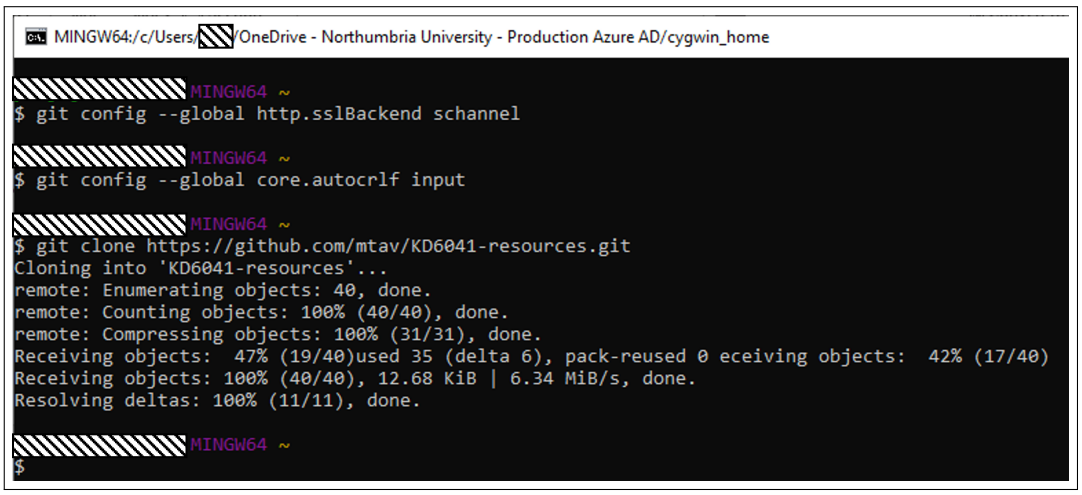
3. You now have a *Git Bash prompt.* To get the resources for this module, run the following commands in it **(one by one!)**:

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

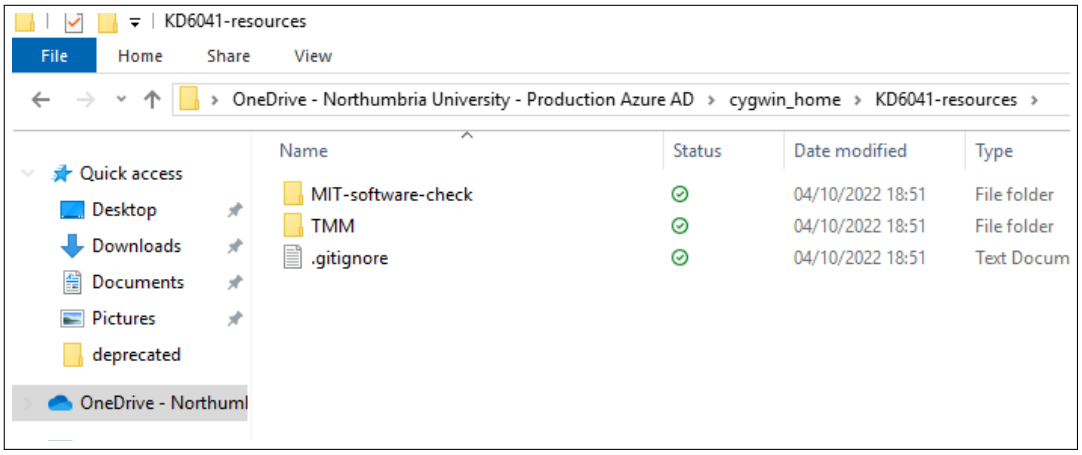
```
git config --global core.autocrlf input
```

```
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:

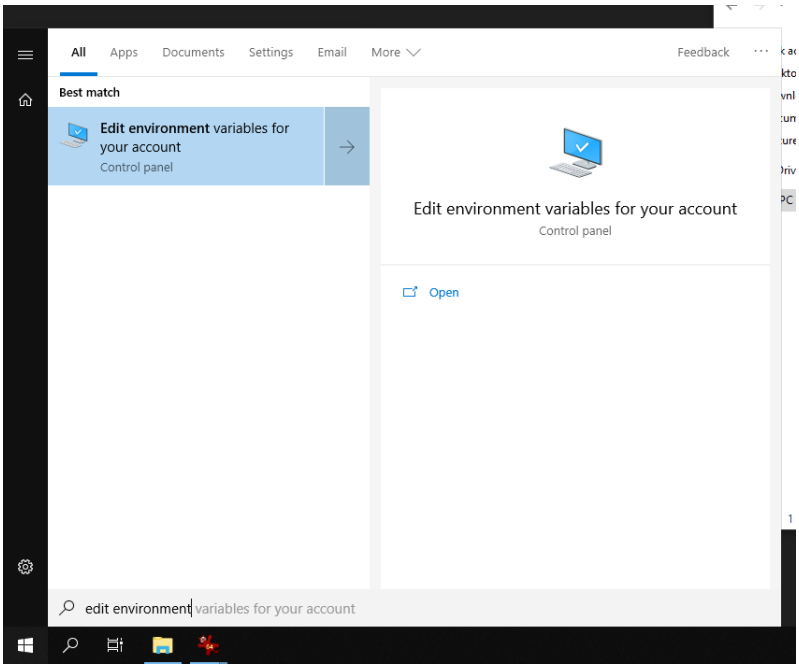


! If you had any trouble with these instructions, you can of course also directly access or download the files from here:
https://github.com/mtav/KD6041-resources
However the benefit of "*cloning*" will be that if there are any updates, you can simply run the "*git pull*" command from a *Git Bash prompt*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

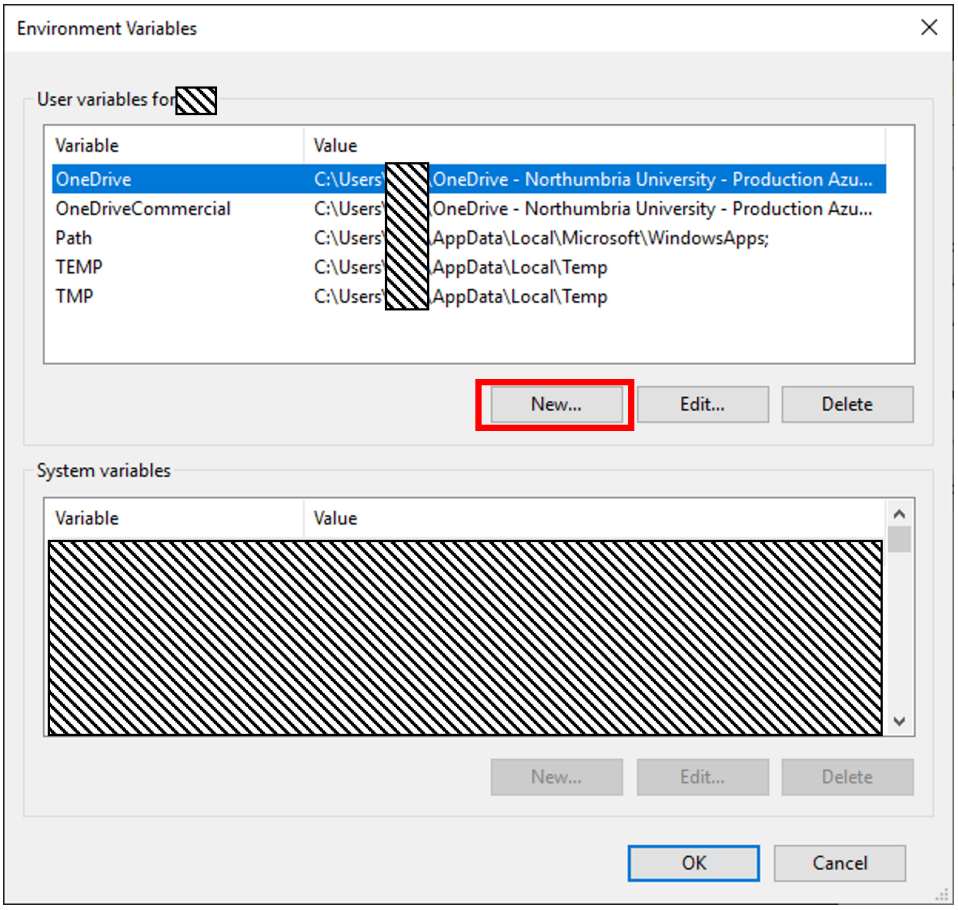
1.1.4 Define environment variables

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...":



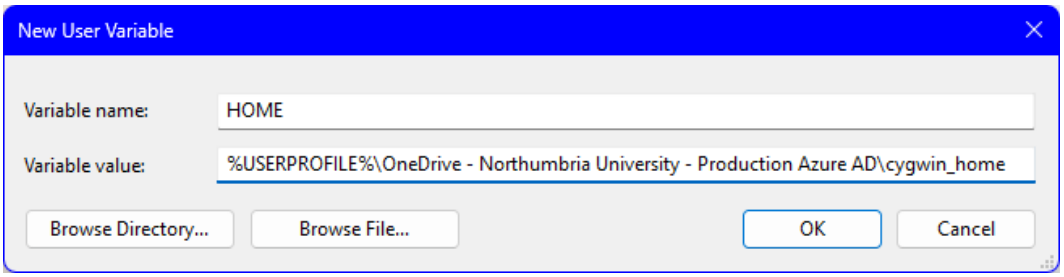
3. Enter the following in the new dialog that appears, then click “OK”.

   Variable name:

   ```
   HOME
   ```

   Variable value:

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



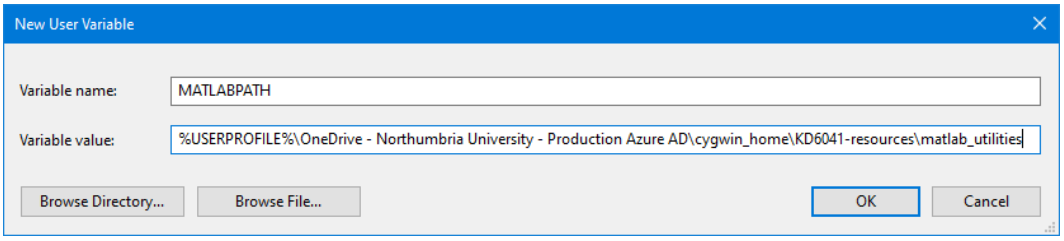
4. Click on “New...” again like in step 2. Then enter the following in the new dialog that appears, then click “OK”.

   Variable name:

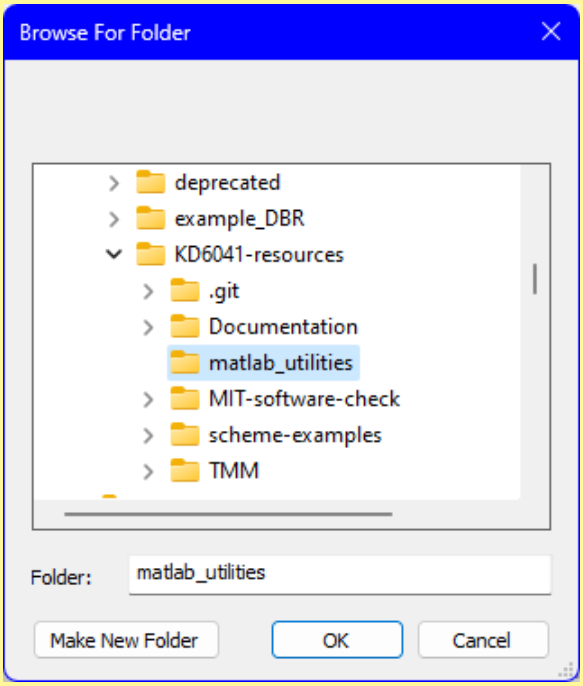
   ```
   MATLABPATH
   ```

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

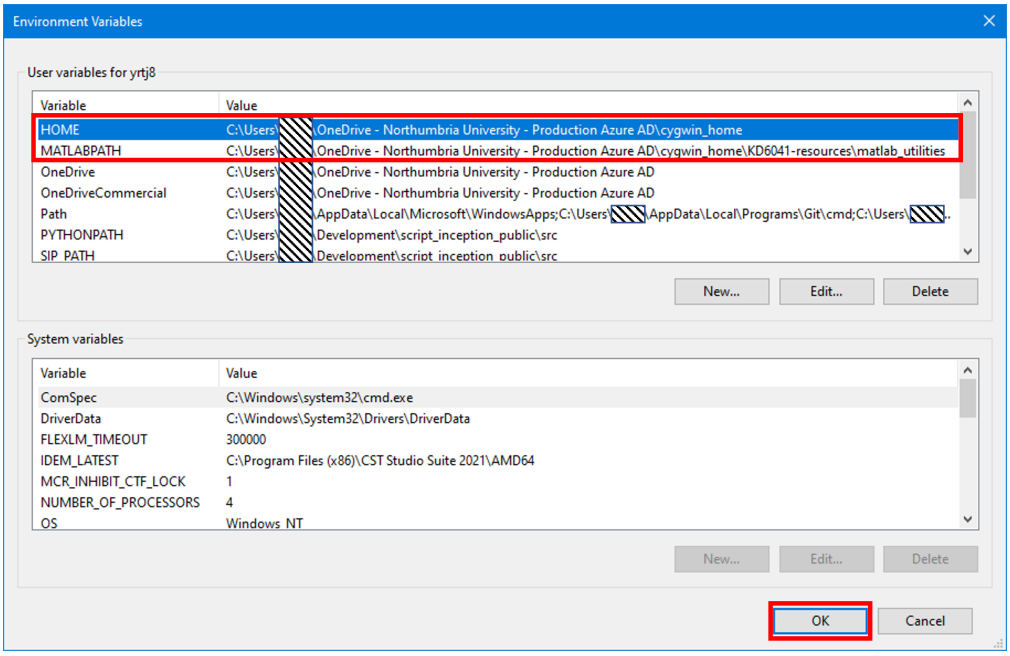
   ```
   %USERPROFILE%\OneDrive - Northumbria University - Production Azure
       AD\cygwin_home\KD6041-resources\matlab_utilities
   ```



The path for *Variable value* should be a single line with a space between "Azure" and "AD". If you have any trouble copy-pasting it, or get errors on step 9, try clicking on "Browse Directory..." and just navigating to the "KD6041-resources\matlab_utilities" directory instead, as in the image below.



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:*

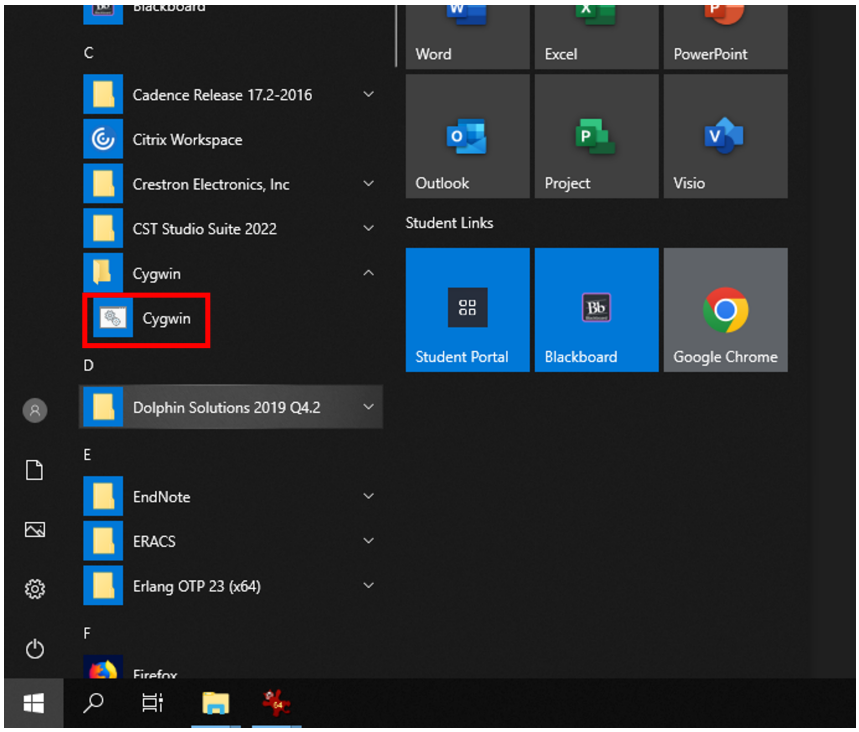
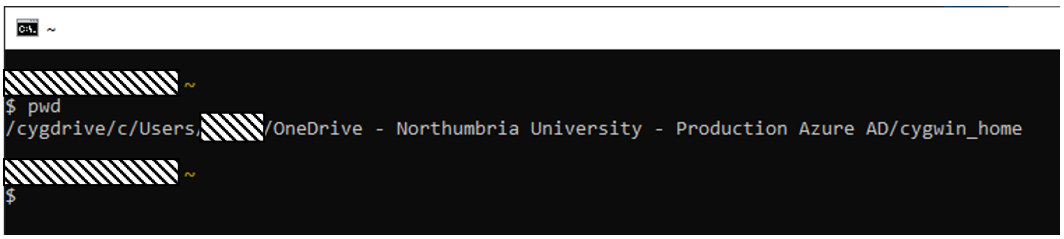


Figure 1.1: Opening Cygwin.

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

```
pwd
```

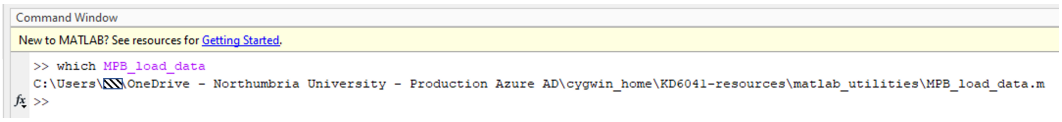


> ! *pwd* stands for *"Print Working Directory"* and is very useful to know where you currently are in the filesystem when using a terminal. In Cygwin, the Windows *"C:"* drive is mapped to *"/cygdrive/c"* and instead of backslashes ("\"), it uses forward slashes ("/") in path names.

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:



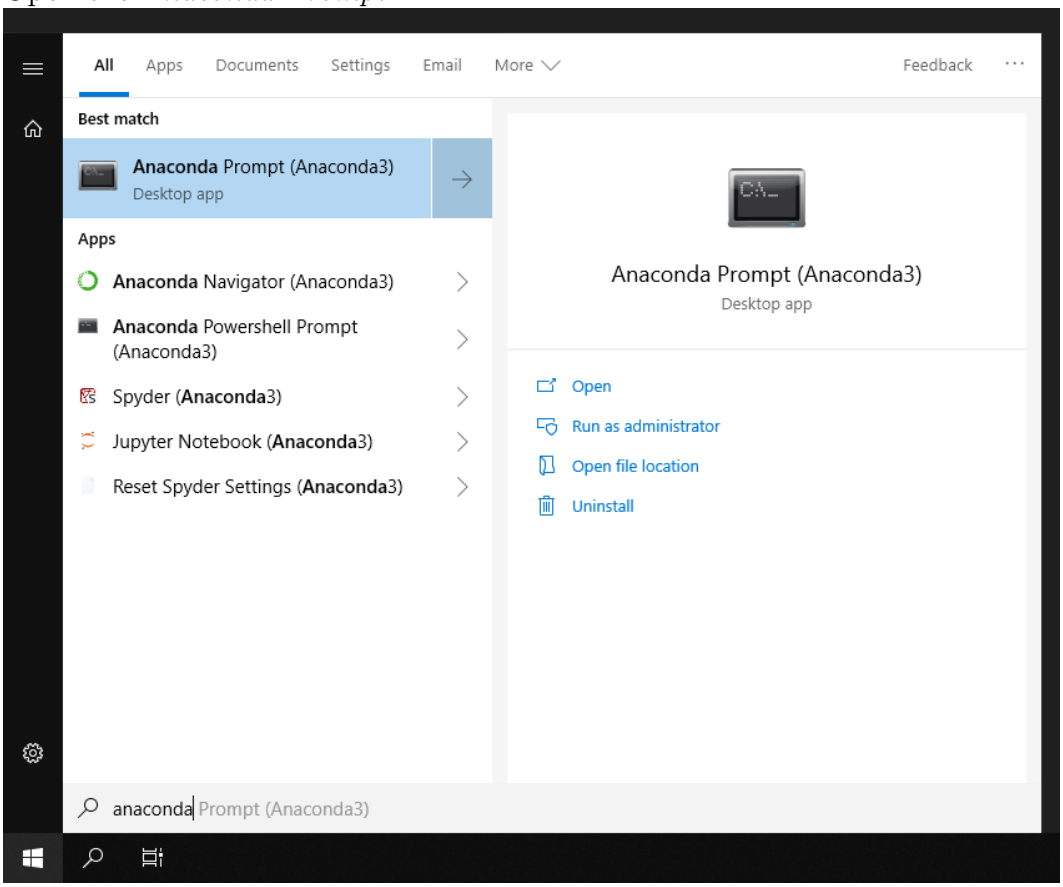
If you get *"'MPB_load_data' not found"* instead, something went wrong. Check the contents of the *MATLABPATH* at step 4 again.

1.1.5 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.

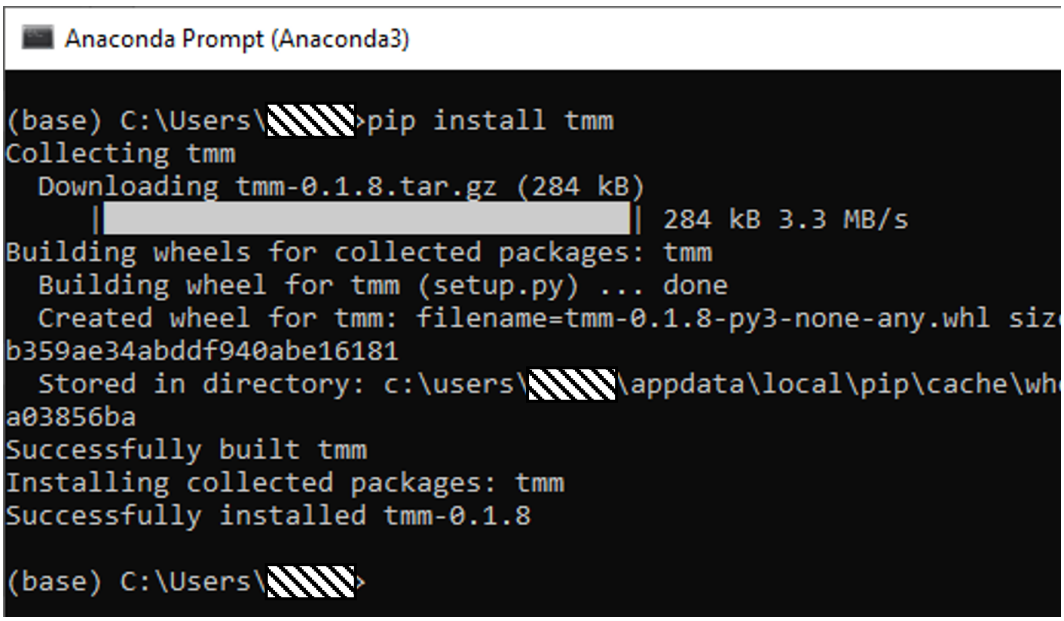
This is a Python module. If you want to follow these installation instructions at home, you will have to install the Python distribution *Anaconda* first. You can get it from here: https://www.anaconda.com/.

1. Open the *Anaconda Prompt*:



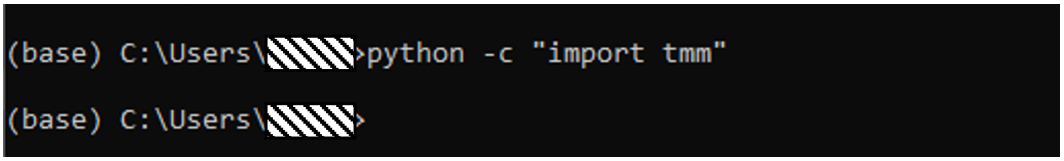
2. Enter the following command in it:

```
pip install tmm
```



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

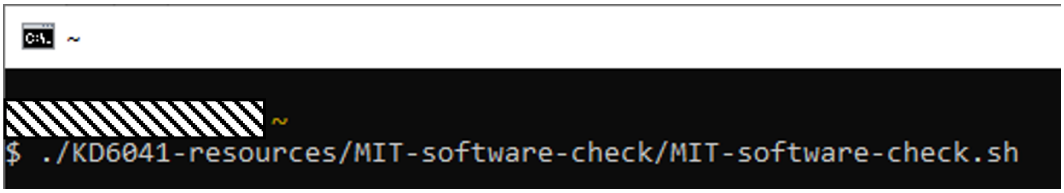
```
python -c "import tmm"
```



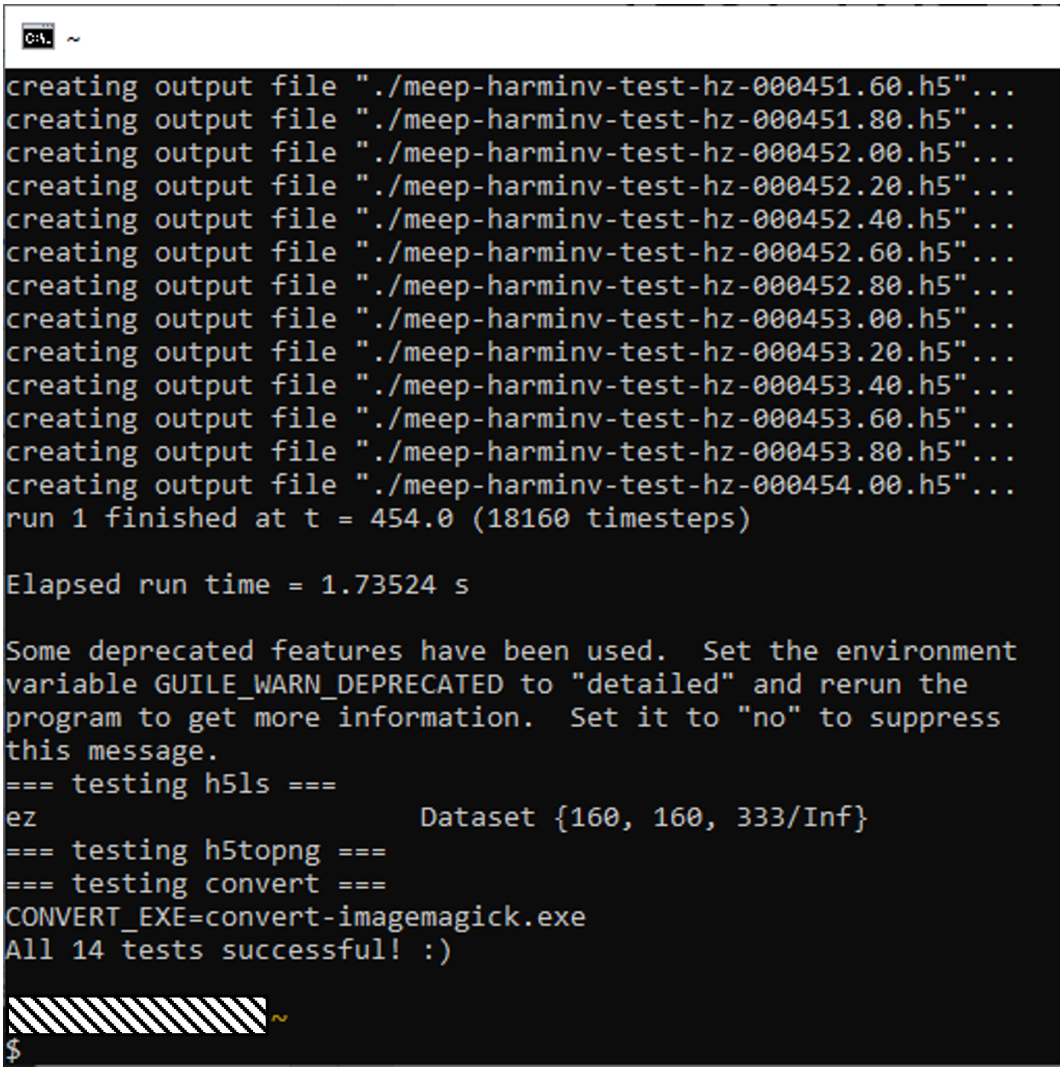
1.1.6 Test the MIT tools

1. Open a *Cygwin terminal* as you did in step 6 of section 1.1.4.
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:



1.2 Getting started with the Python TMM module

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

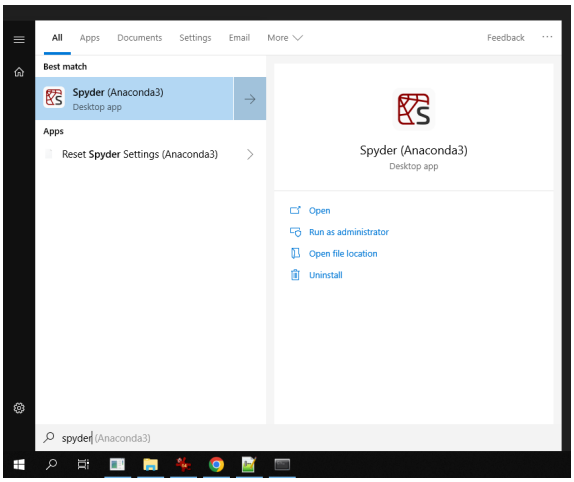


Figure 1.2: Opening *Spyder*.

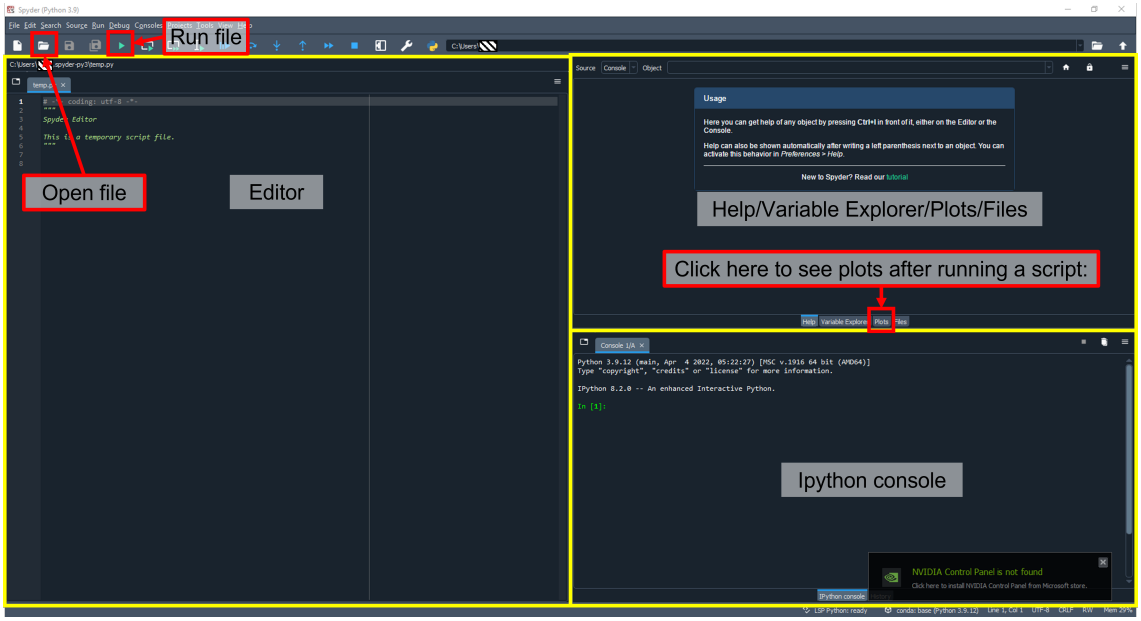
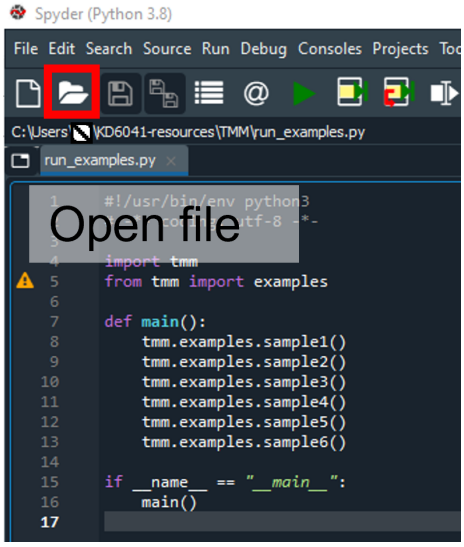
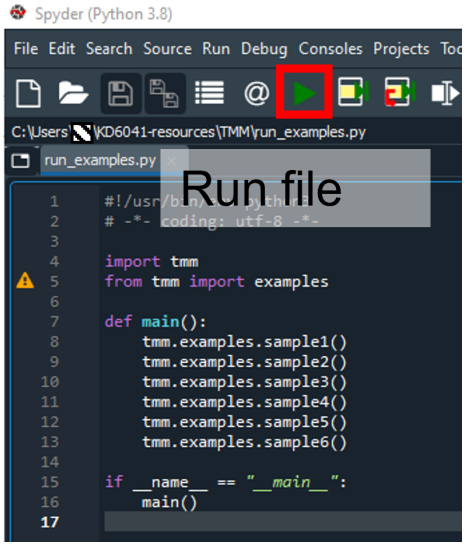


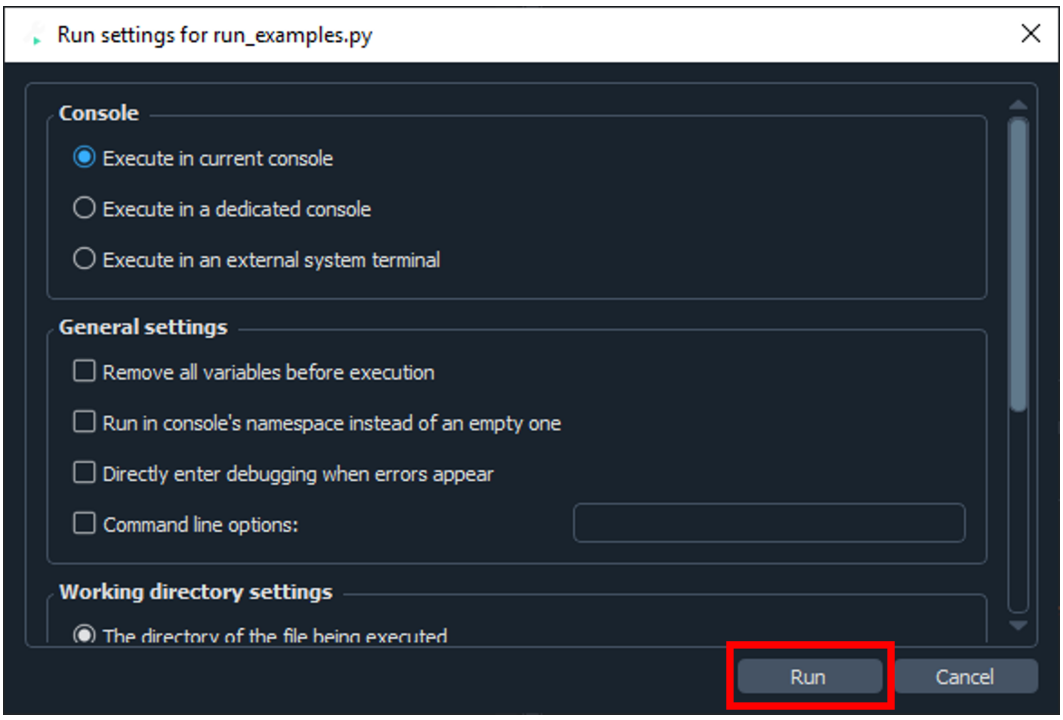
Figure 1.3: The *Spyder* interface.

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

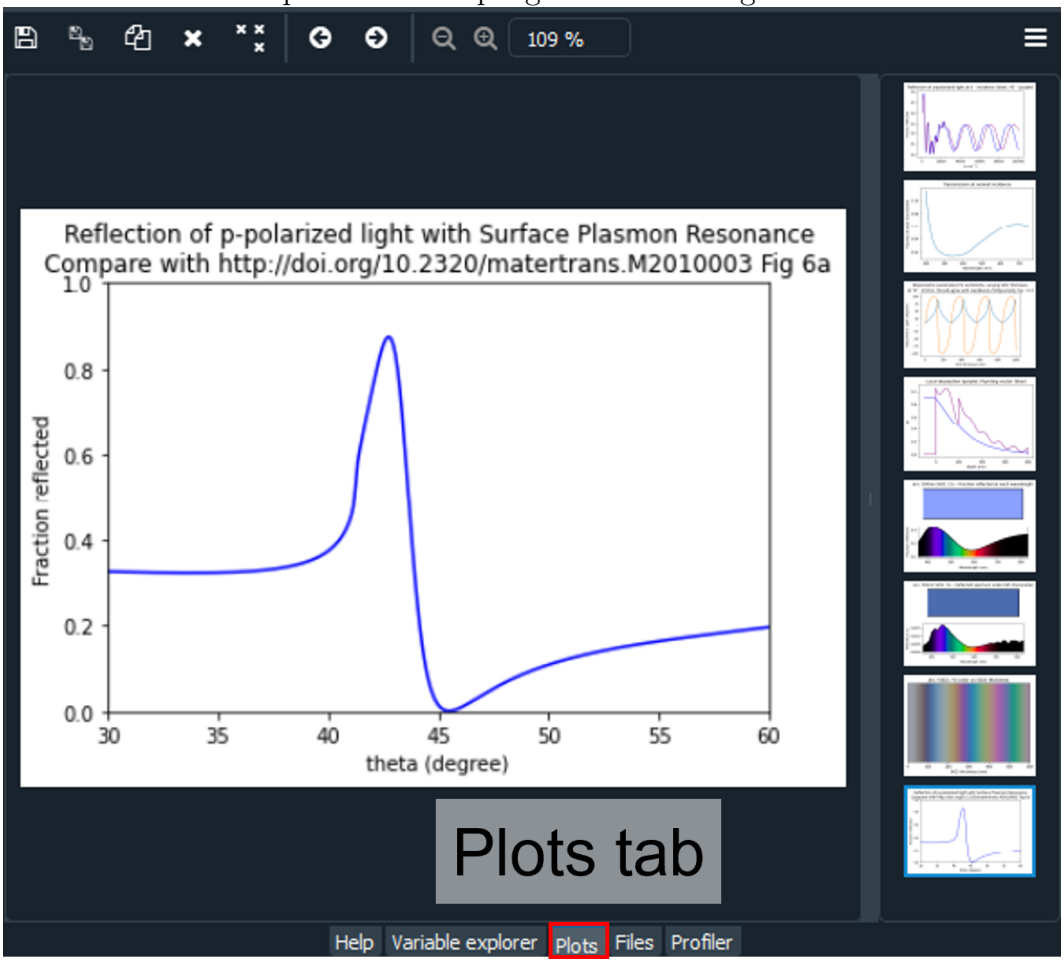




3. If you get a "Run settings" window, just leave the defaults and click on "Run":



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



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

2.1 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 1.1.3, 1.1.5 and 1.2.

2.2 The Transfer-Matrix model

2.2.1 2x2 Matrix formulation for a multilayer system

Let us consider the structure shown in Figure 2.1. 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:

$$\begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} A_s \\ B_s \end{bmatrix} \tag{2.1}$$

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

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

$$\bar{T} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = D_0^{-1} \left\{ \prod_{j=1}^{N} D_j P_j D_j^{-1} \right\} D_s \tag{2.2}$$

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:

$$P_j = \begin{bmatrix} e^{i\Phi_j} & 0 \\ 0 & e^{-i\Phi_j} \end{bmatrix} \tag{2.3}$$

and:

$$\Phi_j = k_{jy} \cdot d_j = \frac{2\pi \cdot n_j \cdot d_j}{\lambda_r} cos(\theta_j) \tag{2.4}$$

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 2.1, where the X-Y plane is the plane of propagation), the surface reflection and transmission coefficients are given by:

$$D_{js} = \begin{bmatrix} 1 & 1 \\ n_j cos(\theta_j) & -n_j cos(\theta_j) \end{bmatrix} \tag{2.5}$$

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 2.1), the surface reflection and transmission coefficients are given by:

$$D_{jp} = \begin{bmatrix} cos(\theta_j) & cos(\theta_j) \\ n_j & -n_j \end{bmatrix} \tag{2.6}$$

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 (2.6).

Thus, for a single layer within the stack:

$$Q_j = D_j P_j D_j^{-1} = \begin{bmatrix} cos(\Phi_j) & \frac{i}{y_j} sin(\Phi_j) \\ iy_j sin(\Phi_j) & cos(\Phi_j) \end{bmatrix} \tag{2.7}$$

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:

$$\begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = \bar{T} \begin{bmatrix} A_s \\ B_s \end{bmatrix} = D_0^{-1} \left\{ \prod_{j=1}^{N} \begin{bmatrix} cos(\Phi_j) & \frac{i}{y_j} sin(\Phi_j) \\ iy_j sin(\Phi_j) & cos(\Phi_j) \end{bmatrix} \right\} D_s \begin{bmatrix} A_s \\ B_s \end{bmatrix} \tag{2.8}$$

Exercise 2.2.1: Equation for p-waves

Write equation 2.8 for *p-waves*.

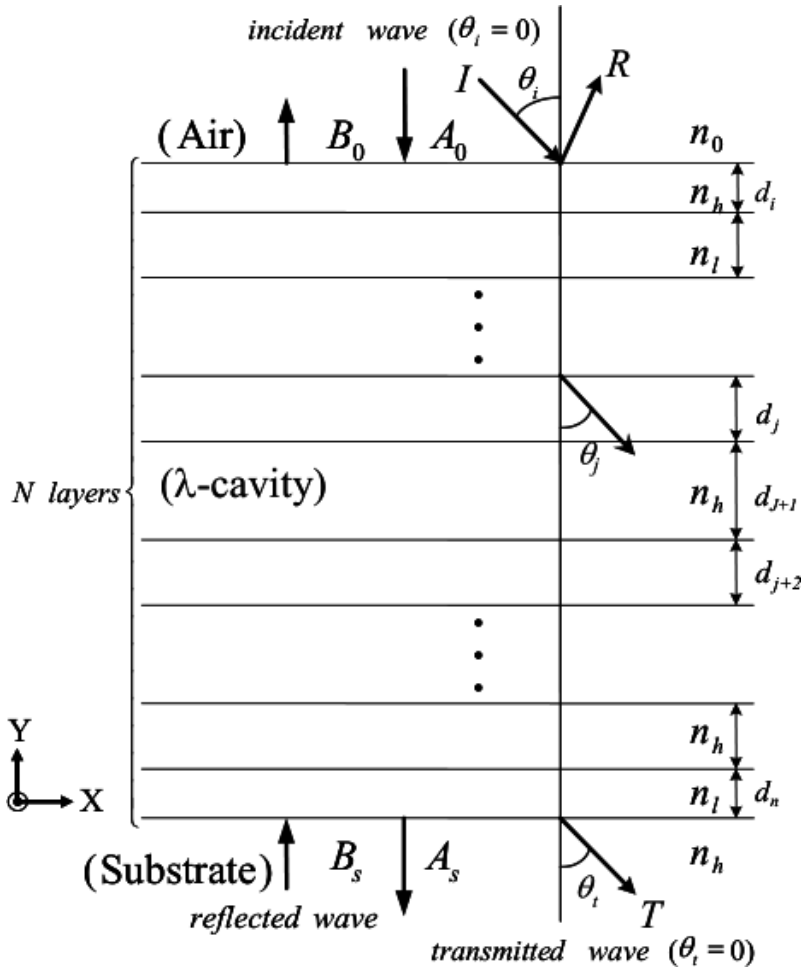


Figure 2.1: Schematic representation of a *Distributed Bragg Reflector (DBR)* made up of N layers of material with alternating refractive indices n_h and n_l. A_0 and B_0 are the forward and backward electric field amplitudes at the input, while A_s and B_s are the forward and backward electric field 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).

2.2.2 Transmittance, reflectance and absorptance

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

$$r = \left(\frac{B_0}{A_0}\right)_{B_s=0} \tag{2.9}$$

$$t = \left(\frac{A_s}{A_0}\right)_{B_s=0} \tag{2.10}$$

Based on equation (2.1), this leads to:

$$r = \frac{M_{21}}{M_{11}} \tag{2.11}$$

$$t = \frac{1}{M_{11}} \tag{2.12}$$

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

$$R = |r|^2 = \left|\frac{M_{21}}{M_{11}}\right|^2 \tag{2.13}$$

Transmittance is given by:

$$T = \frac{n_s cos(\theta_s)}{n_0 cos(\theta_0)}|t|^2 = \frac{n_s cos(\theta_s)}{n_0 cos(\theta_0)}\left|\frac{1}{M_{11}}\right|^2 \tag{2.14}$$

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

$$A = 1 - R - T \tag{2.15}$$

2.3 Guided examples

2.3.1 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 2.2.

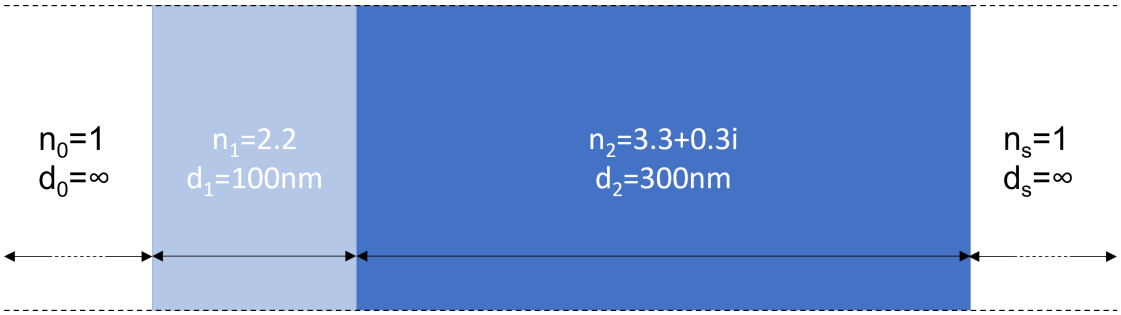


Figure 2.2: Geometry of the two-layer example.

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 1.1.3.

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': (-0.38199021640362935+0.17312587458941395j), 't':
    (-0.03769252771074001-0.3453788012827542j), 'R': 0.17588909388044108, 'T':
    0.12070724302073714, 'power_entering': 0.824110906119559, '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:

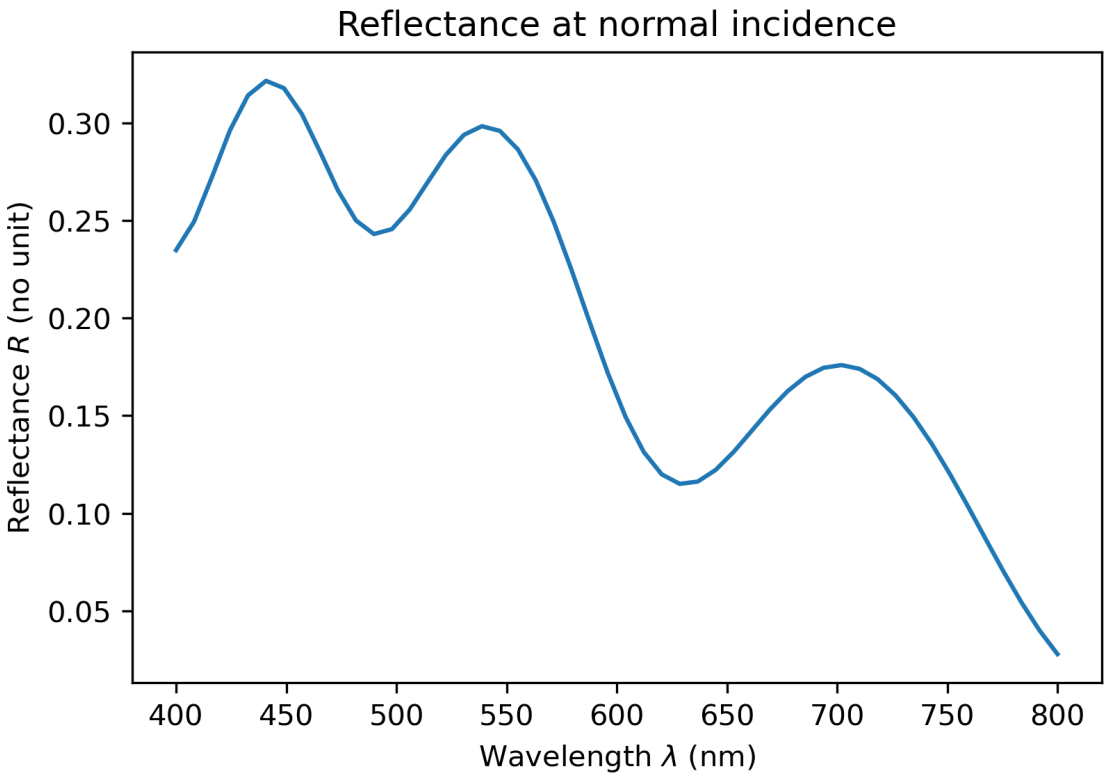


Figure 2.3: Reflectance of the structure from Figure 2.2.

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 2.3.1: Can you explain the behaviour?

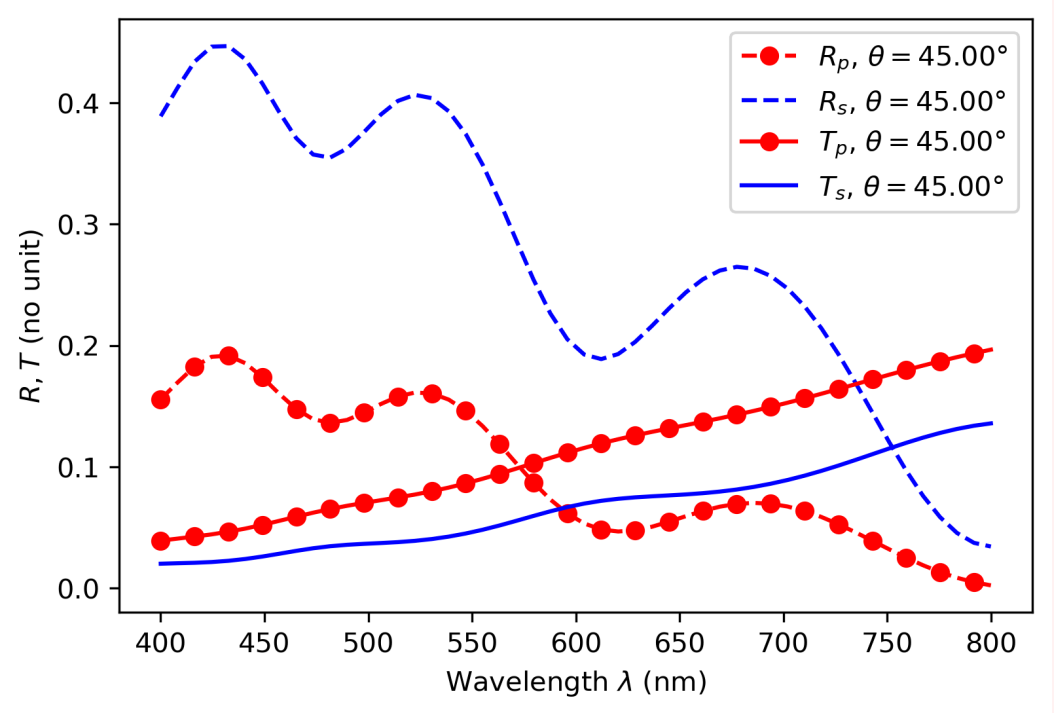
How does the reflectance in Figure 2.3 change with wavelength? Why? What is this phenomenon called?

Exercise 2.3.1: Reflectance at a different incidence angle.

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

Exercise 2.3.2: Comparison of reflectance, transmittance, S and P polarization

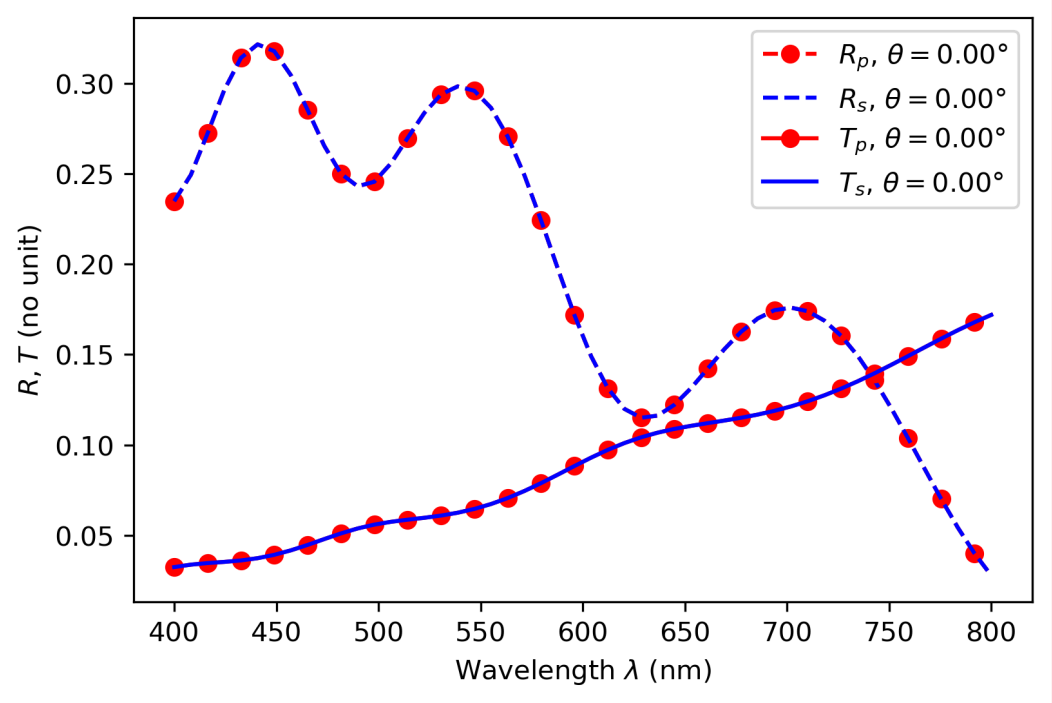
Create a plot to compare reflectance, transmittance, S and P polarization for an incidence angle of 45° like this:



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

Exercise 2.3.3: 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?

2.3.2 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 2.4.

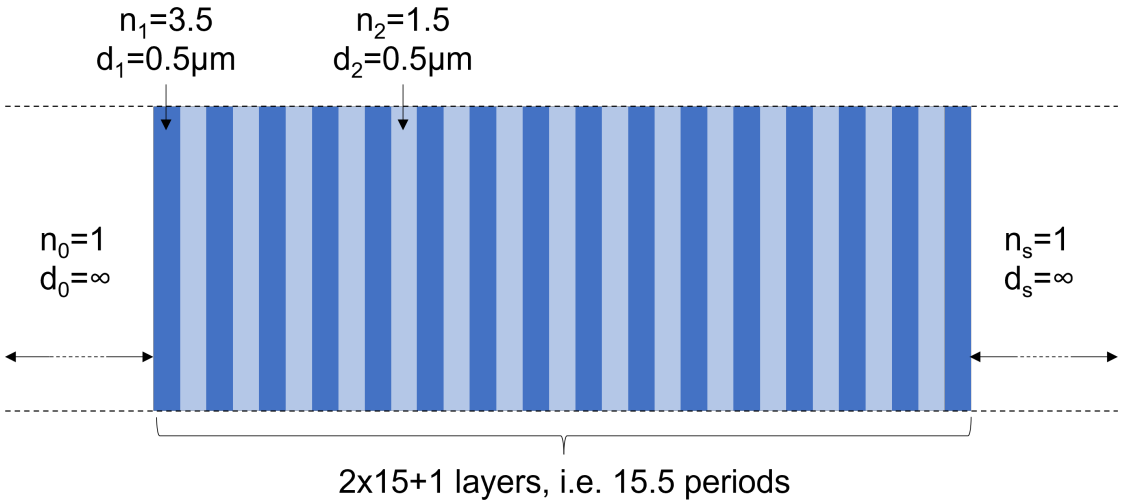


Figure 2.4: Geometry of the DBR example.

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:

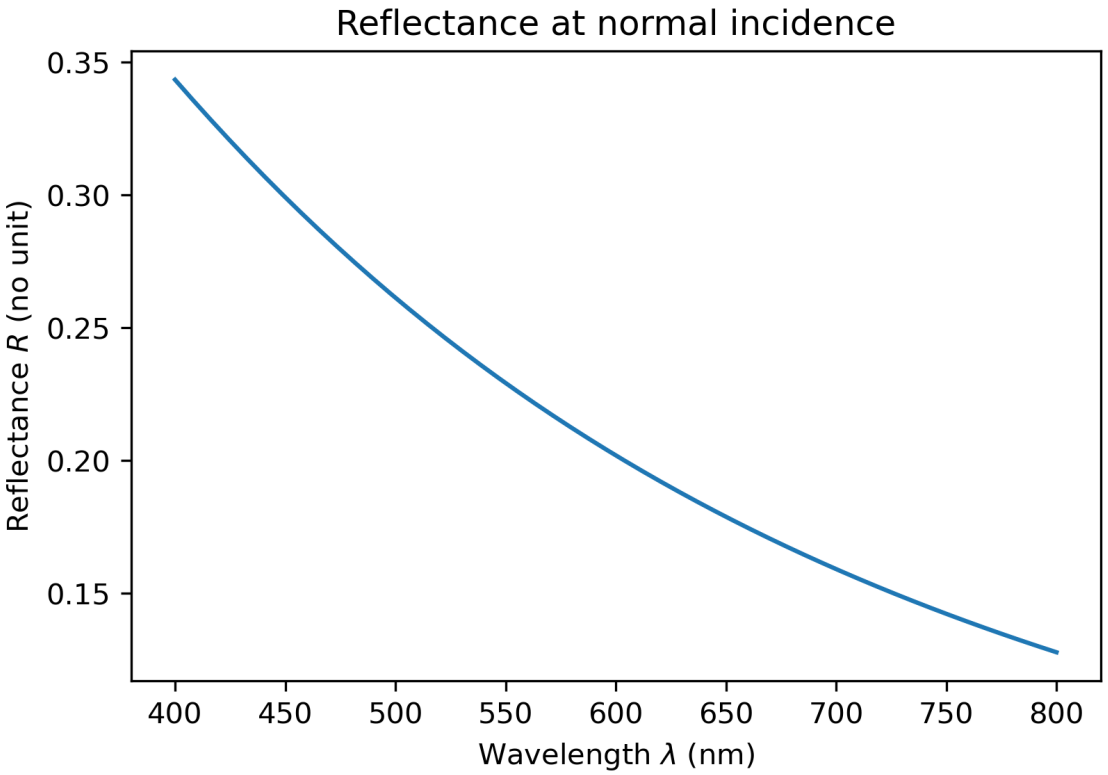


Figure 2.5: New output after changing the geometry.

Question 2.3.2: Bandgaps

a) What is the expected wavelength of the fundamental bandgap for a DBR like the one in Figure 2.4?

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

c) Why does the plot in Figure 2.5 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:

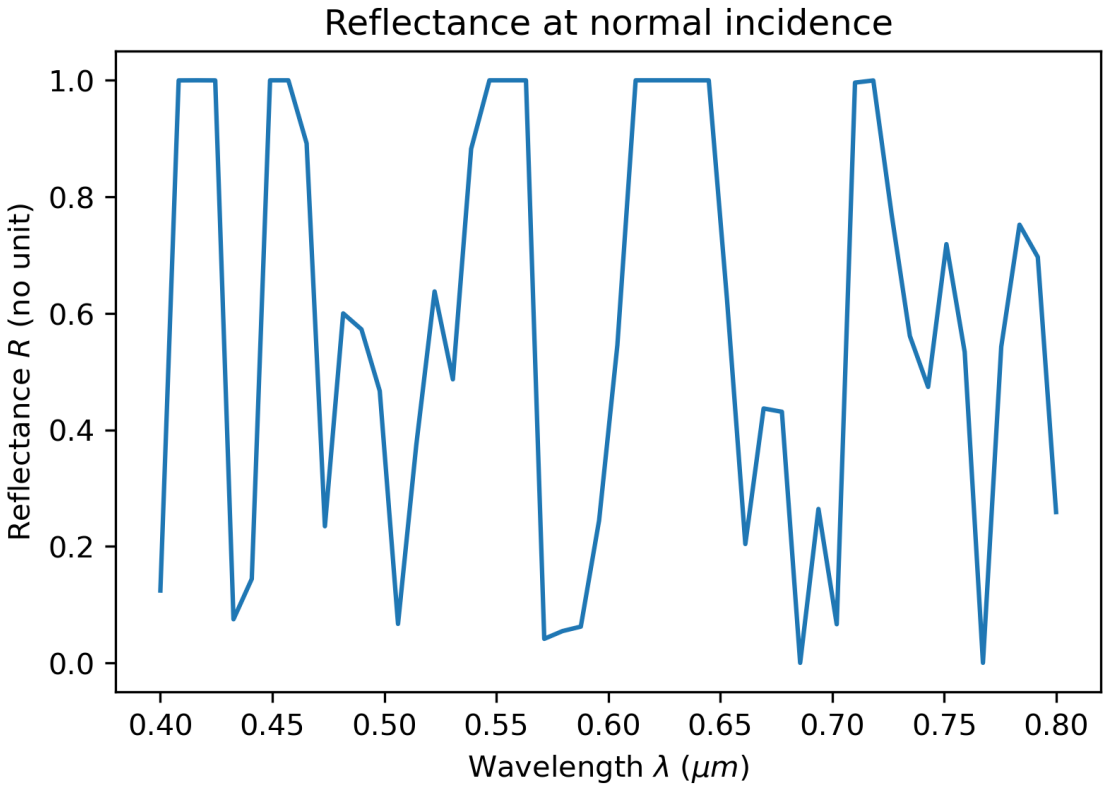


Figure 2.6: New output after correcting the wavelength range.

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:

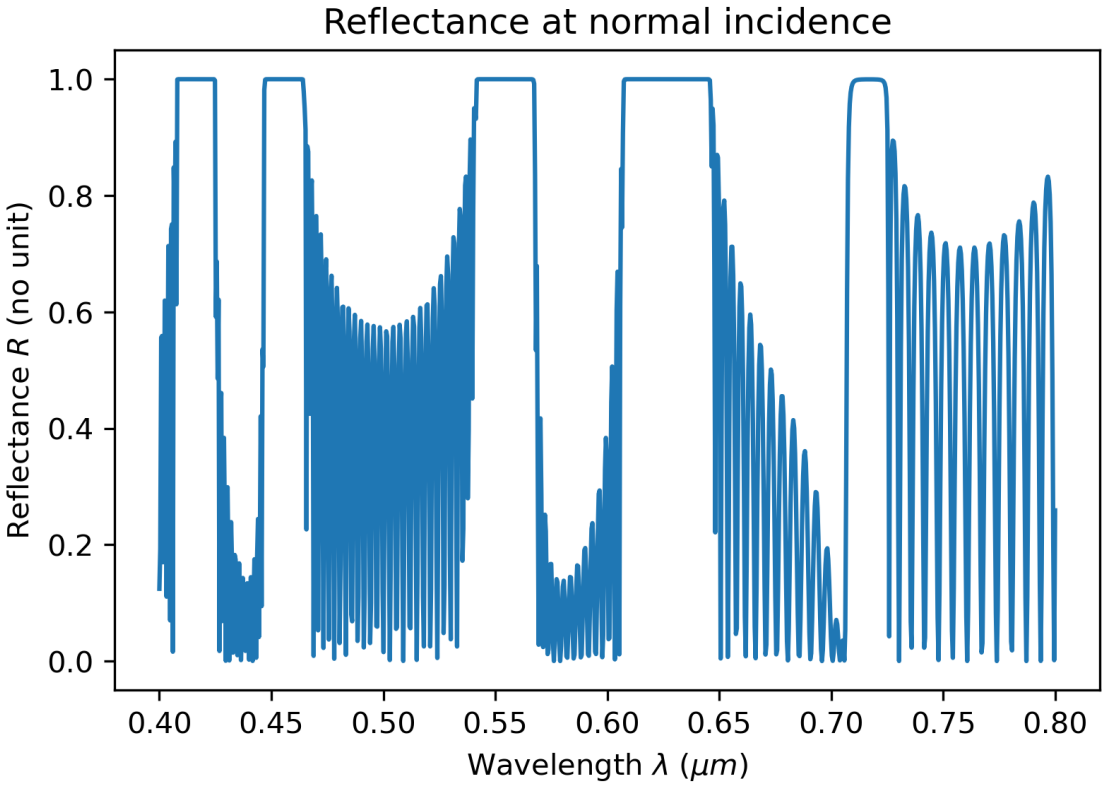


Figure 2.7: New output after using 1000 points.

2.3.3 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:

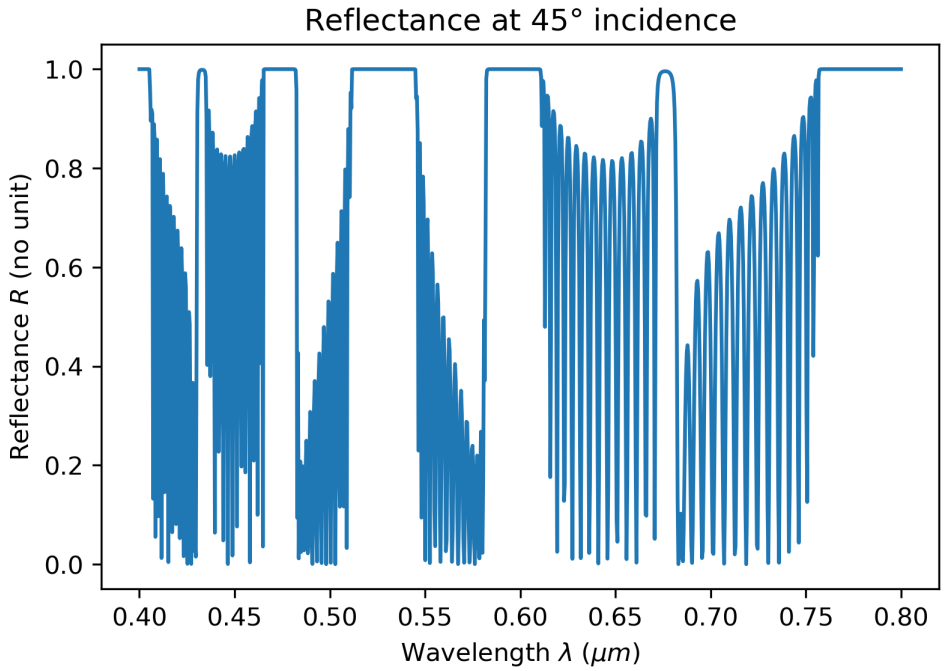


Figure 2.8: DBR reflectance at 45° incidence.

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

Now we want to create something like in Figure 2.9, i.e. plot the reflectance against incidence angle and wavelength:

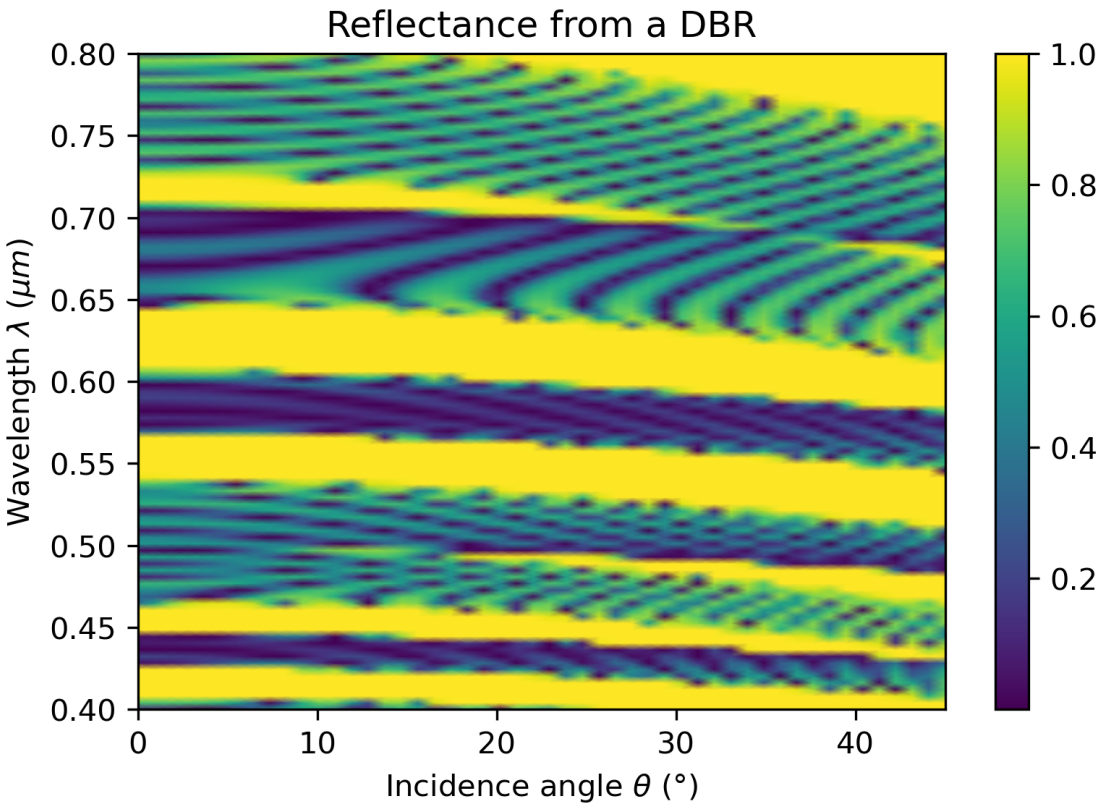


Figure 2.9: Plotting the reflectance against incidence angle and wavelength.

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 2.9.

2.4 Homework

2.4.1 Reflection from a single interface

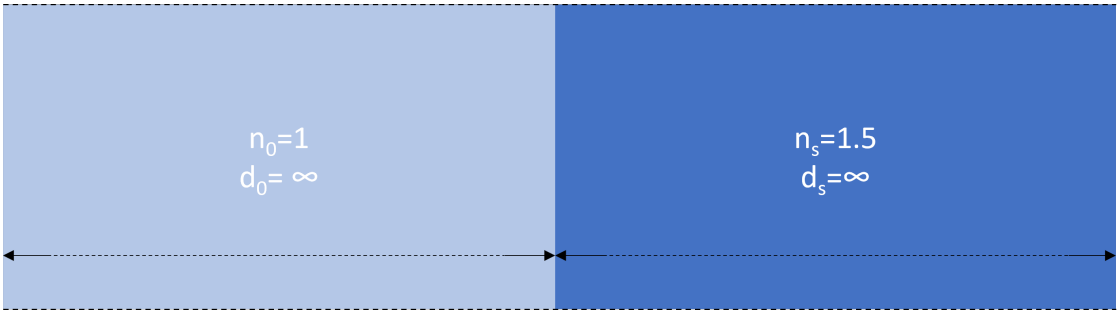


Figure 2.10: A single interface.

1. Plot the reflection from a single interface as illustrated in figure 2.10.

   You can re-use the code from section 2.3.1 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]
```

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

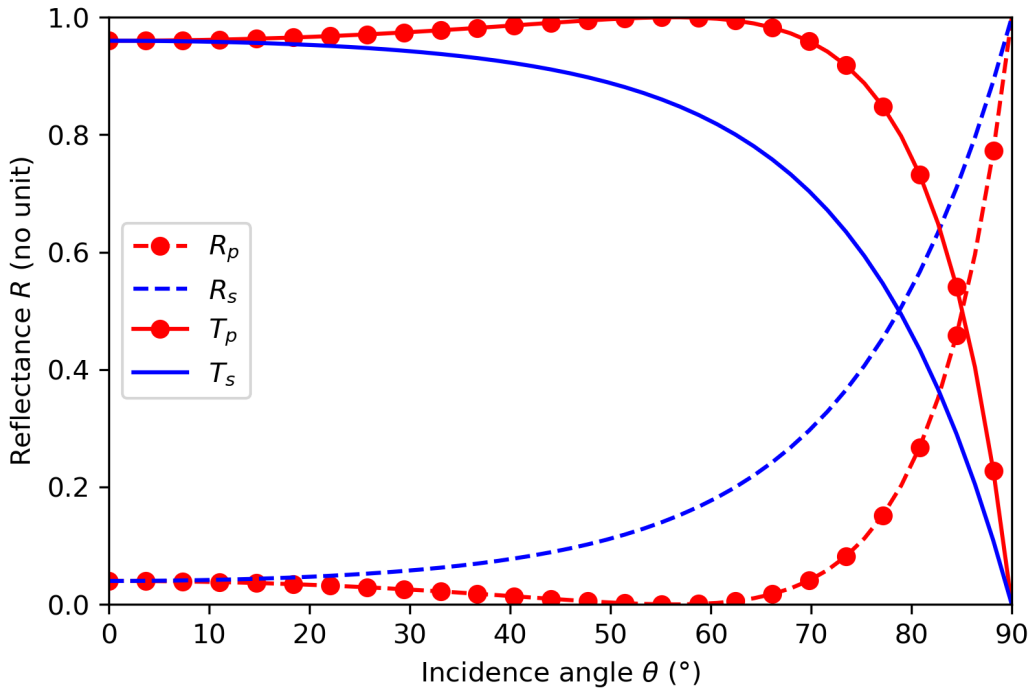
3. What is the theoretical expression of R_p, R_s, T_p and T_s for a single interface between materials of refractive index n_0 and n_s?

4. What are the values of R_p, R_s, T_p and T_s for:

   a) $n_0 = 1$, $n_s = 1.5$, $\theta = 0°$

   b) $n_0 = 1$, $n_s = 1.5$, $\theta = 45°$

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



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

b) At what angle exactly is it the lowest?

c) Does it drop to zero?

2.4.2 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 2.11 summarizes these infos.

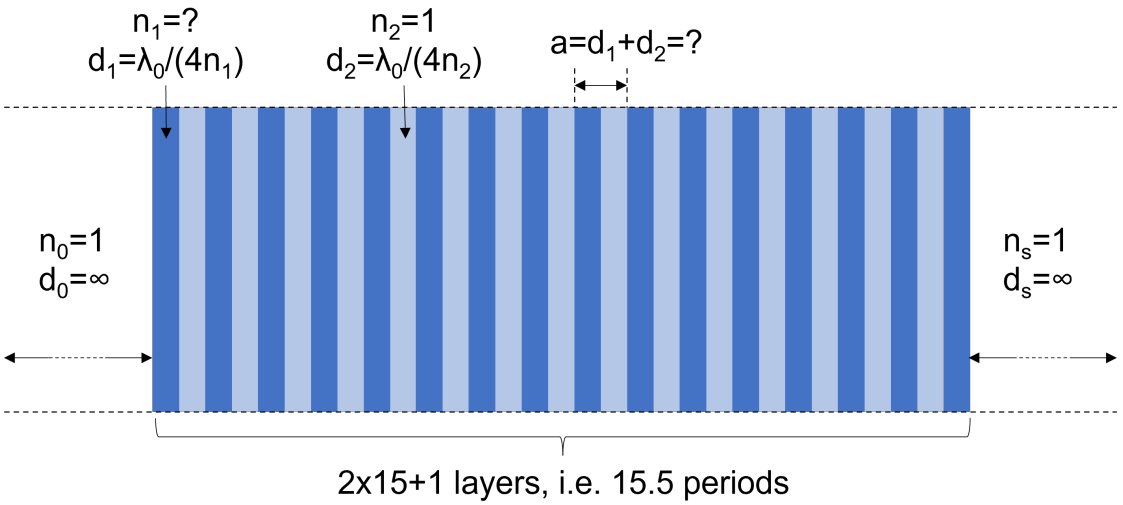


Figure 2.11: Parameters of the DBR.

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?

2.4.3 Design an anti-reflection coating

An anti-reflection coating consists of a thin layer of material deposited onto a substrate as illustrated in Figure 2.12, 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.

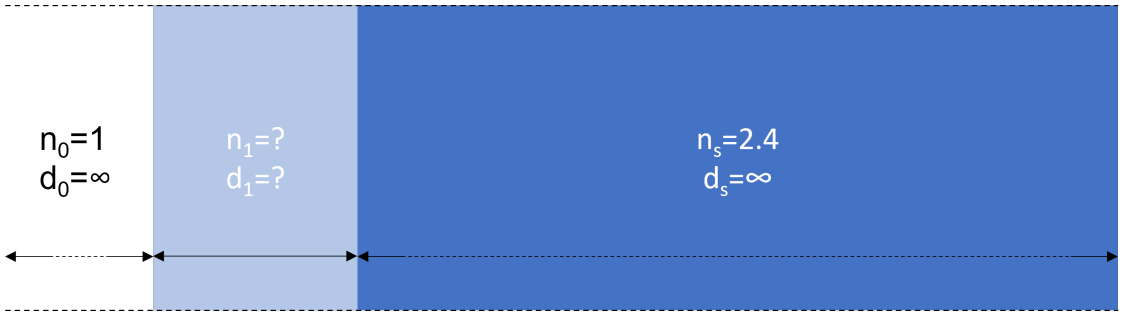


Figure 2.12: The basic design of an anti-reflection coating.

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: $n_s = 2.4$

3 MPB Part 1: 1D Photonic crystals

3.1 Introduction

To compute photonic bandstructures and field distributions for **infinite** photonic crystals, we can use the *Plane-Wave Expansion* (PWE) method. In this module, we will use the Free software package *MIT Photonic Bands* (MPB)[2] to do so.

Unlike the *Transfer-Matrix Method* (TMM) which simulates finite 1D structures that are not necessaily periodic, the *Plane-Wave Expansion* (PWE) simulates only infinite periodic structures. However, these can be 1D, 2D or 3D. While in *TMM*, the whole finite structure is defined, in *PWE*, only a unit-cell is defined. To simulate defects in *PWE*, one needs to use a so-called *supercell*, which contains multiple *primitive unit-cells*. See table 3.1.

Software	Python TMM module	MIT Photonic Bands (MPB)
Method	Transfer-Matrix Method (TMM)	Plane-Wave Expansion (PWE)
Simulates	Finite structures	Infinite structures
Must be periodic?	No	Yes
Supports	1D only	1D, 2D, 3D
Simulation region	Whole structure.	A unit-cell.

Table 3.1: Comparison of TMM and PWE.

3.2 The MPB software

The Free software package *MIT Photonic Bands* (MPB)[2] is a software that can solve the *master equation* (5.10) by using the *Plane-Wave Expansion* method. If you would like to know more about the details of how it does this, you can have a look at the original paper by the developers of MPB[3] and appendix D of [4].

Figure 3.1 summarizes the inputs and outputs of MPB.

One important aspect of *MPB*, as well as its related software for *FDTD* simulations called *MEEP*[5], is that **it uses normalized units**. Before you start defining a new lattice and geometry, you should choose a reference length a. Then you specify all dimensions (geometry sizes like radius, length, etc and length of the basis vectors) in units of a. All the frequencies that you specify or that are returned are then normalized frequencies of the form $f_N = \frac{f}{c_0/a}$.

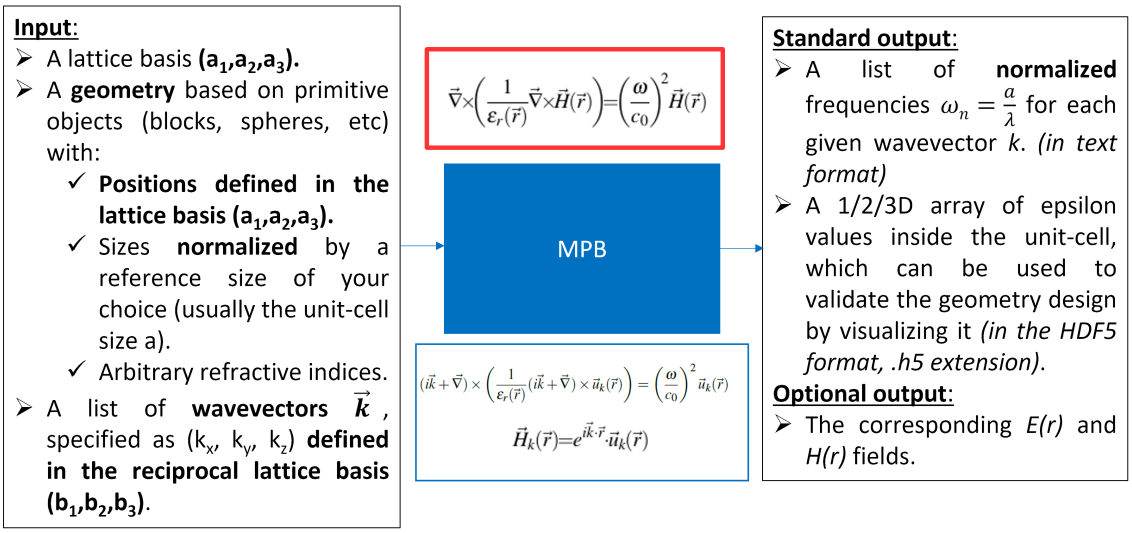


Figure 3.1: Summary of what MPB does.

While you can in principle define $a = 1\mu m$ if you want to specify all dimensions in μm for example, it is more common, and recommended, to use the *lattice constant* of the unit-cell as reference length a, i.e. define a as the length of one of the basis vectors.

Table 3.2 summarizes the way the main quantities are normalized.

	Distance	Time	Speed	Frequency	Angular frequency	Wavevector
Real units	d (metres)	t (seconds)	c_0 (m/s)	f (Hz)	Ω (rad/s)	$\vec{k}$ (m^{-1})
Normalized units	$d_N = \frac{d}{a}$	$t_N = \frac{t}{a/c_0}$	$c_N = \frac{c_0}{c_0} = 1$	$f_N = \frac{f}{c_0/a}$	$\omega_N = \frac{\omega}{2\pi c_0/a}$	$\overrightarrow{k_N} = \frac{\vec{k}}{2\pi/a}$

Table 3.2: Normalization used in *MPB* and *MEEP*.

f_N can also often be found expressed in one of the following ways:

- $f_N = f/(c_0/a) = (c_0/\lambda)/(c0/a) = a/\lambda$
- $f_N = f/(c_0/a) = (\omega/2\pi)/(c_0/a) = \omega/(2\pi c_0/a)$

> Note that by definition $\omega_N = f_N = a/\lambda$ when using normalized units.

MPB uses a scripting language called *Scheme* in which you define the various inputs and then call a run function with various associated outputs.

The basic structure of most MPB input scripts will be as follows:

1. Define new parameters.
2. Set parameters. *(ex: resolution, num-bands)*
3. Define the geometry lattice.
4. Define the geometry.
5. Define the desired k-points.
6. Run the simulation.

3.3 Scheme basics

- MPB input files are written using a programming language called Scheme.
- Scheme consists of a series of function calls of the form: **(func arg1 arg2 arg3...)**
- Comments in the code can be added by prefixing them with a semicolon (;). Example: **(func arg1 arg2 arg...) ; This is a comment.**
- The MPB script files are simple text files with a **.ctl** extension (ctl stands for control).
- To run an MPB script called input.ctl, type the following at a bash command prompt: **mpb input.ctl**, then press enter.

There are two ways to use MPB:

1. **Interactive mode:** You start an MPB shell in which you enter commands and run them.
2. **Script mode:** You create a plain text file with commands in them and pass it to MPB.

3.3.1 Interactive mode

Let's try the interactive mode first.

1. Open Cygwin as in figure 3.2.

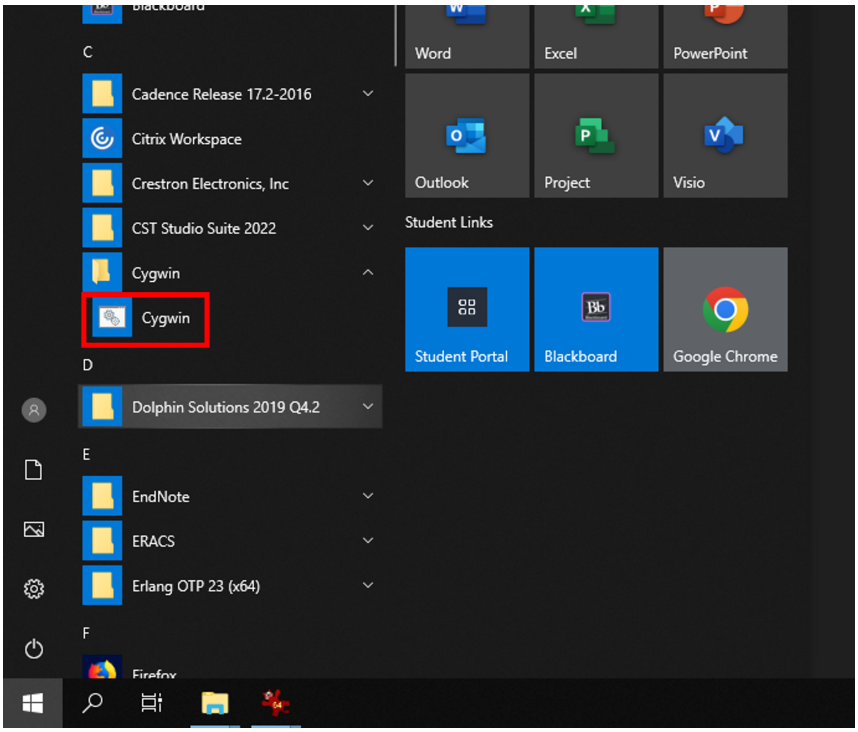


Figure 3.2: Opening Cygwin.

2. Type "mpb" and press enter.
3. Now you can try a simple command such as this one to print out "Hello world!":

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

   Figure 3.3 shows more possible commands you can try.

> ! To exit the interactive mode, enter the command **(exit)** or press **ctrl+D**.

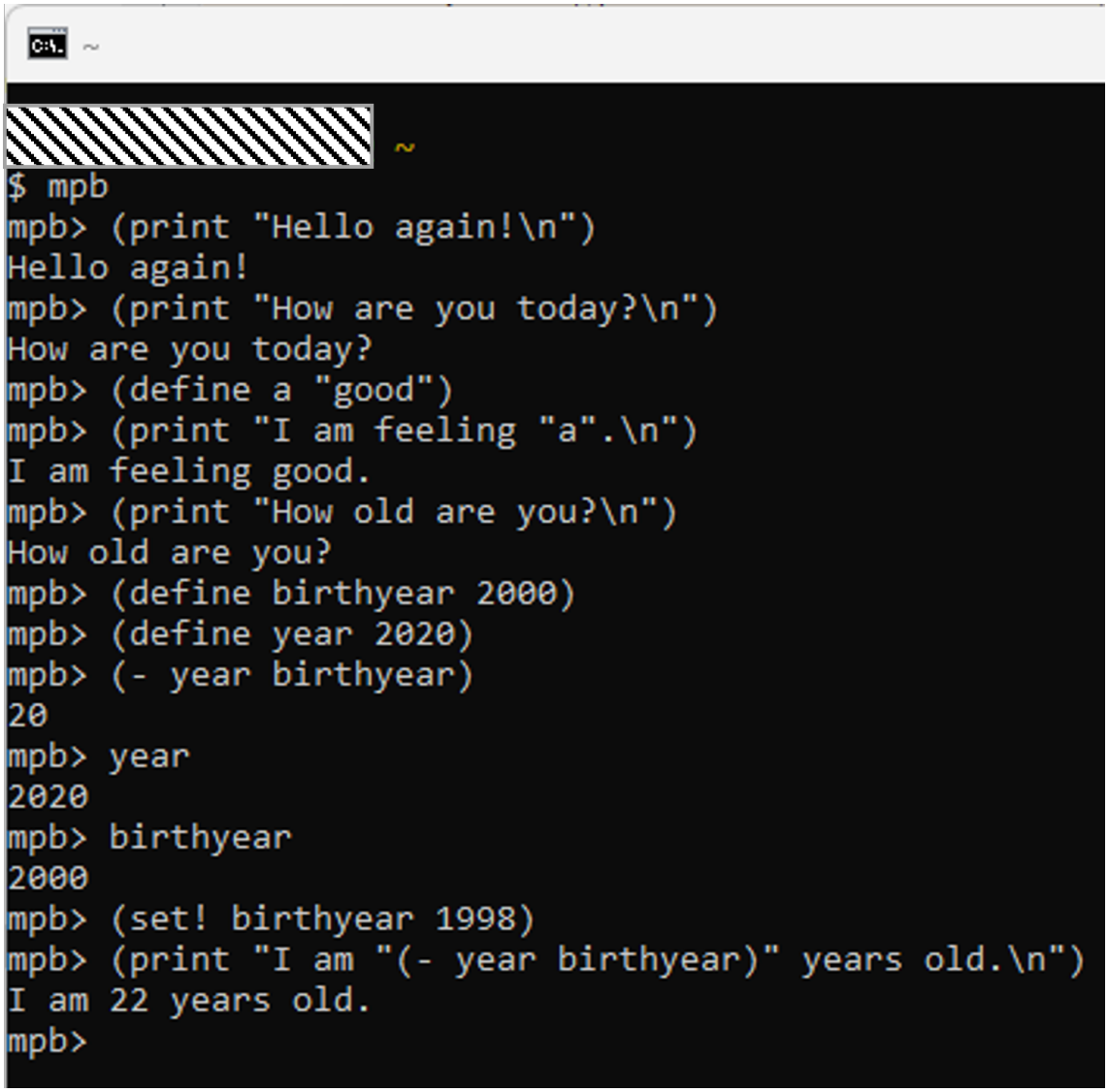


Figure 3.3: MPB's interactive mode.

3.3.2 Script mode

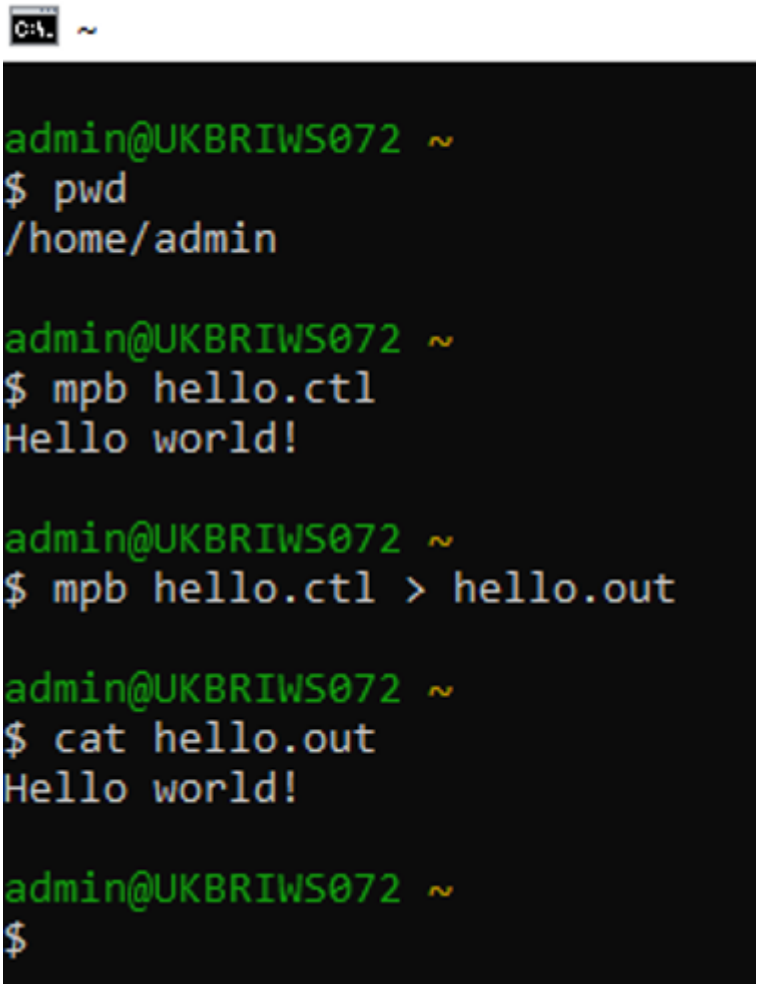
Next let us try the script mode. Before you start, you might want to configure Notepad++ as explained here: *KD6041-resources\Documentation\notepad++_tips.pdf*.

1. Open *Notepad++*.
2. Open the file "KD6041-resources\scheme-basics\hello.ctl" in it and save it as "hello.ctl" in your HOME directory. It should look like this:

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

3. Next open *Cygwin* again if it isn't already open.
4. Then run this command in it to run the script:

```
mpb hello.ctl
```



5. Try making some change to the script and running it again.
6. To store the output of the script to a file, there are two options:
   a) Storing the output without showing it on the screen:

```
mpb hello.ctl > hello.out
```

   b) Storing the output, but also showing it on the screen:

```
mpb hello.ctl | tee hello.out
```

7. You can check the contents of a text file using the *cat* command:

```
cat hello.out
```

8. Alternatively, you can just open the file using the Windows File Explorer.
9. Try storing the output in *hello.out* and opening it in *Notepad++*.

3.3.3 Defining variables

There are 2 ways of defining variables:

- If you do not need the ability to change the parameter outside the script:

```
(define x 42)
```

- If you do need the ability to change the parameter outside the script:

```
(define-param x 42)
```

Once a variable is defined, its value can be changed in the following corresponding ways:

```
(set! x 42)
(set-param! x 42)
```

The difference between *define* and *set* with or without *"-param"* is that by adding *"-param"* you can override the parameters by passing them by command-line.

1. To test this, open the script *KD6041-resources\scheme-basics\example-1.ctl*:

```
(print "=== Custom defined user variables without '-param': \n")
(define foo 42)
(print "after define: foo=" foo "\n")
(set! foo 45)
(print "after set!: foo=" foo "\n")
(print "=== Custom defined user variables with '-param': \n")
(define-param foo_param 42)
(print "after define-param: foo_param=" foo_param "\n")
(set-param! foo_param 45)
(print "after set-param!: foo_param=" foo_param "\n")
(print "=== MPB input variables: \n")
(print "before set-param!: resolution=" resolution "\n")
(set-param! resolution 256)
(print "after set-param!: resolution=" resolution "\n")
(exit)
```

2. Now in the cygwin terminal, run the following commands:

```
cd ~/KD6041-resources/scheme-basics/
```

```
mpb example-1.ctl
```

3. Now try it with some custom parameters like this:

```
mpb foo=5 foo_param=55 resolution=555 example-1.ctl
```

Notice how *foo* is not changed, while *foo_param* and the built-in MPB parameter *resolution* do get changed. This is summarized in figure 3.4.

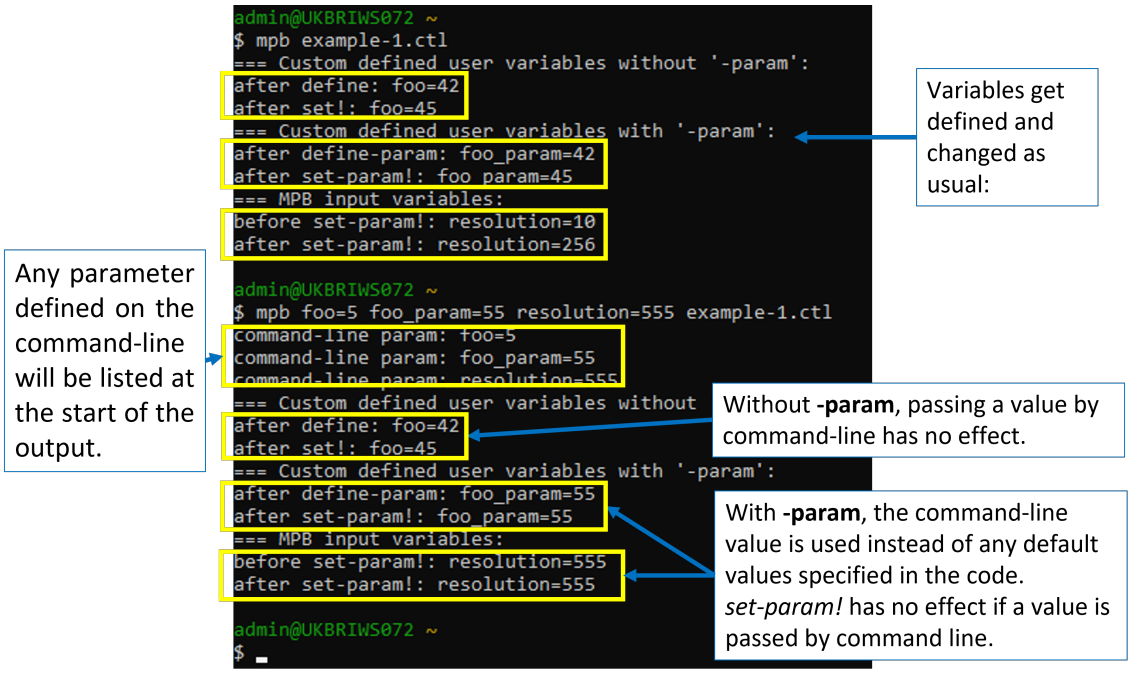


Figure 3.4: Passing parameters by command-line.

3.3.4 Functions and mathematical operations

Functions in Scheme are defined as follows:

```
(define (FUNCTION_NAME ARG1 ARG2 ...)
        STATEMENTS...
        FINAL_STATEMENT ; The return value will be that of the final statement
)
```

The function is then called as follows:

```
(FUNCTION_NAME ARG1 ARG2 ...)
```

Mathematical operations in Scheme follow the same principle. So to add numbers for example, you would do:

```
(+ 1 2 3 4) ; = 1+2+3+4 = 10
(- 1 2 3 4) ; = 1-2-3-4 = -8
(* 1 2 3 4) ; = 1*2*3*4 = 24
(/ 1 2 3 4) ; = 1/2/3/4 = 1/24
```

If you only pass one number, the "+" and "-" functions will assume that you are adding or subtracting from zero, while the "*" and "/" functions will assume that you are multiplying or dividing by one:

```
(+ 2) ; = 0+2 = 2
(- 2) ; = 0-2 = -2
(* 2) ; = 1*2 = 2
(/ 2) ; = 1/2 = 1/2
```

Example 3.3.1: Fresnel equation in Scheme

As an example, this is how you would write the reflection amplitude for S polarization $r_s = \frac{n_0 cos(\theta_0) - n_s cos(\theta_s)}{n_0 cos(\theta_0) + n_s cos(\theta_s)}$ in Scheme:

```
(/
  (- (* n0 (cos theta_0)) (* ns (cos theta_s)))
  (+ (* n0 (cos theta_0)) (* ns (cos theta_s)))
)
```

You will find an example of a simple function *f(x)=x+1* in *KD6041-resources\scheme-basics\example-2.ctl*:

```
; simple function returning
(define (f x)
  (define a 1) ; variable definition
  (+ a x) ; the last value gets returned
)

; test function
(print "f(2)=" (f 2) "\n")
```

If you run this script, you will find that you end up at an MPB prompt:

```
$ mpb example-2.ctl
f(2)=3
mpb> (f 3)
4
mpb> (f 300)
301
mpb> (exit)
```

This allows you to run addional commands interactively to test the function for example.

If you do not want a script to end up in interactive mode, simply add the command **(exit)** at the end of your script.

3.3.5 Conditional statements

```
(define-param x? true)
(define foo 555)
(if x?
  (begin
    (print "x? is true\n")
    (set! foo 47)
  )
  (begin
    (print "x? is false\n")
    (set! foo 2)
  )
)
(print "foo = " foo "\n\n")
(exit)
```

Figure 3.5: *KD6041-resources\scheme-basics\example-3.ctl*

- Inequality operators

```
(define-param x 42)
(print "x = 2: ") (if (= x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x < 2: ") (if (< x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x <= 2: ") (if (<= x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x > 2: ") (if (> x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x >= 2: ") (if (>= x 2) (print "TRUE\n") (print "FALSE\n") )
(exit)
```

Figure 3.6: *KD6041-resources\scheme-basics\example-4.ctl*

- Other useful operators

```
(and ARG1 ARG2) ; True if both ARG1 and ARG2 are true.
(or ARG1 ARG2) ; True if ARG1 or ARG2 is true.
(not ARG) ; Invert logical value.
(equal? "hello world" "hello") ; Compare strings.
```

3.3.6 Loops

There are at least 3 ways to write loops in Scheme:

1. The classic way, in Scheme, is to write a tail-recursive function:

```
(define (doit x x-max dx)
  (if (<= x x-max)
    (begin
      ...perform loop body with x...
      (doit (+ x dx) x-max dx)
    )
  )
)
(doit a b dx) ; execute loop from a to b in steps of dx
```

2. There is also a do-loop construct in Scheme that you can use:

```
(do
  ( (x a (+ x dx)) ) ; <variable> <init> <step>
  (  (> x b) ) ; <test>
  ...perform loop body with x...
)
```

3. If you have a list of values of x that you want to loop over, then you can use map:

```
(map (lambda (x) ...do stuff with x... ) list-of-x-values)
```

3.4 Guided example: Compute the bandgap of a 1D Photonic crystal (DBR)

1. Plot the first 3 photonic bands at normal incidence (i.e. $\vec{k}$ transverse to the layers) for a Distributed Bragg Reflector (DBR) with the following parameters:
   - $n_1 = 1$
   - $n_2 = 3.2$
   - $d_1 = d_2 = 0.16\mu m$
   - $a = 0.32\mu m$
2. What is the range of the fundamental bandgap in µm?
3. What is the midgap wavelength λ_0?
4. What is the gap-midgap ratio $\frac{\Delta\omega}{\omega_0}$? (For a gap going from frequency ω_1 to ω_2, we define the frequency width $\Delta\omega = |\omega_2 - \omega_1|$ and the midgap frequency $\omega_0 = \frac{2\pi c_0}{\lambda_0}$.)
5. How is the reflectance obtained from the Transfer-Matrix Method (TMM) linked to the photonic bandstructure obtained using the Plane-Wave Expansion (PWE) method?

3.4.1 Preliminary calculations

1. First we need to define a reference length. Here we choose $a = d_1 + d_2$ as illustrated in figure 3.7.

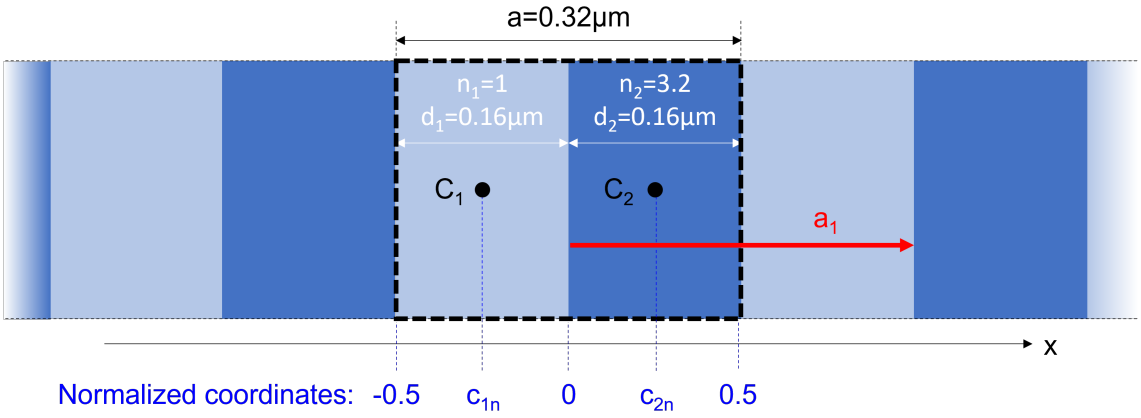


Figure 3.7: The DBR to simulate. For MPB, we only consider a unit-cell as indicated by the dashed black rectangle.

2. Next, we define the lattice vectors:
   $\vec{a_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \vec{x}$, $\vec{a_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \vec{y}$, $\vec{a_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \vec{z}$

3. The corresponding reciprocal lattice vectors are:
   $\vec{b_1} = \frac{2\pi}{a}\vec{x}$, $\vec{b_2} = \frac{2\pi}{a}\vec{y}$, $\vec{b_3} = \frac{2\pi}{a}\vec{z}$

4. The layers will be represented by blocks positioned based on their centres C_1 and C_2 as defined in the lattice basis, i.e.:
   - $C_1 = -0.25\vec{a_1} \Rightarrow C_1 = \begin{pmatrix} -0.25 \\ 0 \\ 0 \end{pmatrix}$
   - $C_2 = +0.25\vec{a_1} \Rightarrow C_2 = \begin{pmatrix} +0.25 \\ 0 \\ 0 \end{pmatrix}$

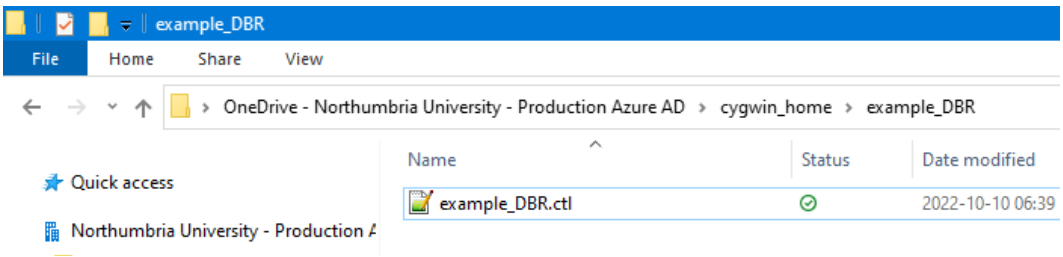
5. Their normalized thicknesses will be:
   - $d_{1n} = \frac{d_1}{a} = 0.5$
   - $d_{2n} = \frac{d_2}{a} = 0.5$

3.4.2 Creating and visualizing the geometry

> ! If you have trouble following the instructions, you can have a look at what the final code should look like in listing 1.

1. Create a new directory named *"example_DBR"* in your *HOME* directory.
2. Open the file *KD6041-resources\MPB\example_DBR.stub.ctl* in *Notepad++* and save it as *"example_DBR.ctl"* in the newly created *"example_DBR"* directory. It should look like this:

```
;;;;; Define new parameters.

;;;;; Set parameters.

;;;;; Define the geometry lattice.

;;;;; Define the geometry.

;;;;; Define the desired k-points.

;;;;; Run the simulation.
```



3. Define n_1 and n_2 as parameters by adding the following code in "*example_DBR.ctl*":

```
;;;;; Define parameters.
(define-param n1 1)
(define-param n2 3.2)
```

4. Now, we will define the geometry lattice. Because our lattice is simply the standard cartesian lattice, which is the default one used in MPB, there is no need to specify the vectors. However, to save on computing resources, we can reduce our simulation to a 1D simulation by setting the lattice size in Y and Z to *no-size*. To do so, add the following code:

```
;;;;; Define the geometry lattice.
(set! geometry-lattice
  (make lattice
    (size 1 no-size no-size)
  )
)
```

5. Define the geometry, which consists of just two *blocks* (one unit-cell), with the following code:

```
;;;;; Define the geometry.
(set! geometry (list
  (make block
    (center -0.25 0 0)
    (material (make dielectric (index n1)))
    (size 0.5 infinity infinity)
  )
  (make block
    (center 0.25 0 0)
    (material (make dielectric (index n2)))
    (size 0.5 infinity infinity)
  )
))
```

6. Before we define our k-points, let us check the geometry we just defined. To do so, we need to run the simulation first. So we will add a call to the *(run-tm)* function:

```
;;;;; Run the simulation.
(run-tm)
```

7. Open a *Cygwin terminal* (See figure 3.8).

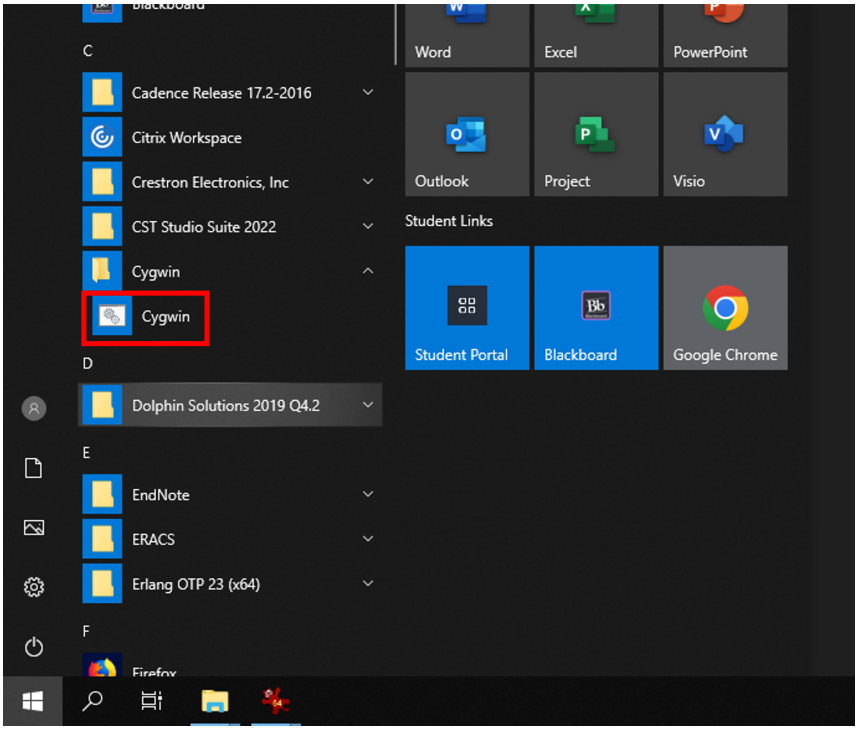


Figure 3.8: Opening Cygwin.

8. Run this command in it to change into the "*example_DBR*" directory:

```
cd ~/example_DBR
```

9. Run the "*example_DBR.ctl*" script you created now, by running the following command in the same terminal:

```
mpb example_DBR.ctl
```

10. If you run the *ls* command now to list the files in that directory, you should see a newly created *"example_DBR-epsilon.h5"* file. See figure 3.9. Of course you can also check this in the Windows File Explorer.

```
ls
```

11. This *"example_DBR-epsilon.h5"* file contains the epsilon distribution inside a single unit-cell. In order to visualize 3 unit-cells, we create a new dataset *data-new* in it by running the following command:

```
mpb-data -m 3 example_DBR-epsilon.h5
```

> *mpb-data* manual: https://manpages.debian.org/testing/mpb/mpb-data.1.en.html

12. Now we convert the newly created dataset into a PNG image by using the *h5topng* command:

```
h5topng -X10 -Y100 example_DBR-epsilon.h5:data-new
```

> **!**
>
> - *-X 10* means "scale the image 10 times in the X direction".
> - *-Y 100* means "scale the image 100 times in the Y direction".
>
> See the *h5topng* manual for more infos:
> https://manpages.org/h5topng

13. You should now have a newly created *example_DBR-epsilon.png* image that looks like figure 3.10.

Figure 3.10: Three unit-cells of the DBR using the default resolution=10.

14. The default resolution is 10. Increase it to 64 by adding the following line in the *"Set parameters"* section **(before *(run-tm)*)**:

```
;;;;; Set parameters.
(set-param! resolution 64) ;; Set resolution.
```

15. Now redo steps 9 to 12. You should now have a newly created *example_DBR-epsilon.png* image that looks like figure 3.11.

Figure 3.11: Three unit-cells of the DBR using resolution=64.

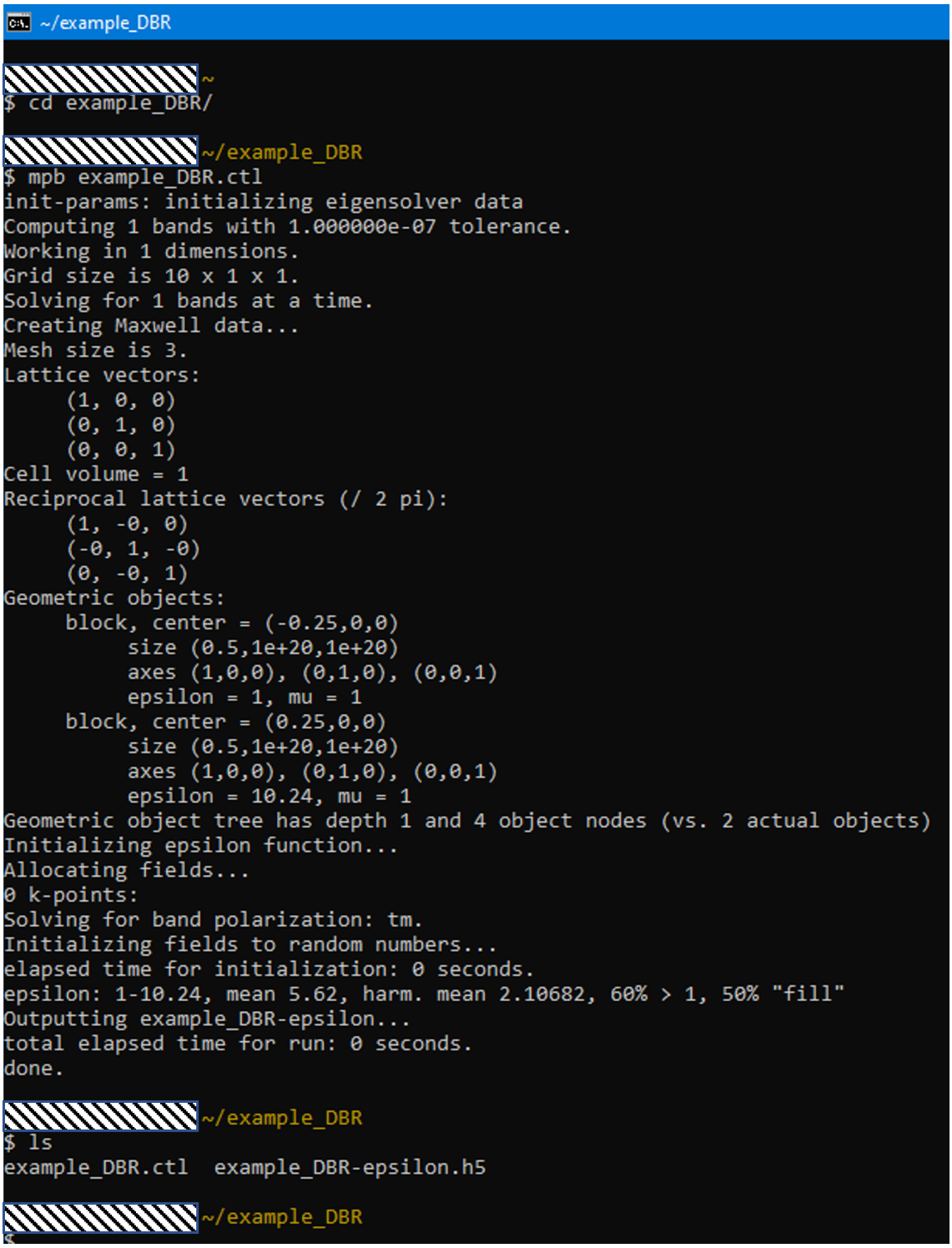


Figure 3.9: Output after the first run without k-points.

3.4.3 Computing and plotting the photonic bands

1. First, set the number of bands we want to calculate. Add the following to the *"Define new parameters"* section **(add it before *(run-tm)*)**:

```
(set-param! num-bands 3) ;; Set number of bands.
```

2. Since we are only interested in the photonic bands for $\vec{k}$ transverse to the layers, i.e. $\vec{k}$ parallel to $\vec{x}$ and since we only need to compute the bands for $\vec{k}$ inside the first Brillouin zone [because $\omega_i(\vec{k}) = \omega_i(\vec{k} + \vec{a_1})$], it is sufficient to use values of $\vec{k}$ going from $\frac{-\pi}{a}\vec{x} = -0.5\vec{b_1}$ to $\frac{+\pi}{a}\vec{x} = +0.5\vec{b_1}$.

   MPB defines k-points in the reciprocal lattice $(\vec{b_1}, \vec{b_2}, \vec{b_3})$, so the following code will do what we need **(add it before *(run-tm)*)**:

```
;;;;; Define the desired k-points.
(set! k-points (list
  (vector3 -0.5 0 0)
  (vector3  0   0 0); Gamma
  (vector3  0.5 0 0)
))
```

   We added the Γ point to get a nice symmetrical plot and also to see what happens when $\vec{k}$ is a multiple of $\frac{2\pi}{a}\vec{x}$.

3. To get more than 3 k-points, we now add 10 extra points between the specified k-points by using the *interpolate* function as follows **(add it after the previous code and before *(run-tm)*)**:

```
(set! k-points (interpolate 10 k-points)) ;; add extra k-points
```

4. This time we are going to store the output from the MPB run to a file called *example_DBR.out* by doing this:

```
mpb example_DBR.ctl | tee example_DBR.out
```

5. And then filter out the lines with the word *"freq"* by using the *grep* command and store the output in *example_DBR.dat*:

```
grep freq example_DBR.out > example_DBR.dat
```

6. If you open *example_DBR.dat* in *Notepad++*, you should see data like this:

```
tmfreqs:, k index, k1, k2, k3, kmag/2pi, tm band 1, tm band 2, tm band 3
tmfreqs:, 1, -0.5, 0, 0, 0.5, 0.169682, 0.283059, 0.661386
tmfreqs:, 2, -0.454545, 0, 0, 0.454545, 0.166338, 0.286204, 0.658581
tmfreqs:, 3, -0.409091, 0, 0, 0.409091, 0.157246, 0.294688, 0.650875
tmfreqs:, 4, -0.363636, 0, 0, 0.363636, 0.144244, 0.306622, 0.639774
tmfreqs:, 5, -0.318182, 0, 0, 0.318182, 0.128848, 0.320402, 0.626699
tmfreqs:, 6, -0.272727, 0, 0, 0.272727, 0.111999, 0.334938, 0.612709
tmfreqs:, 7, -0.227273, 0, 0, 0.227273, 0.0942467, 0.349453, 0.598602
tmfreqs:, 8, -0.181818, 0, 0, 0.181818, 0.0759154, 0.363268, 0.585086
tmfreqs:, 9, -0.136364, 0, 0, 0.136364, 0.0572075, 0.375648, 0.572917
tmfreqs:, 10, -0.0909091, 0, 0, 0.0909091, 0.0382576, 0.385699, 0.563007
tmfreqs:, 11, -0.0454545, 0, 0, 0.0454545, 0.0191628, 0.39238, 0.556407
tmfreqs:, 12, 0, 0, 0, 0, 0, 0.394741, 0.554072
tmfreqs:, 13, 0.0454545, 0, 0, 0.0454545, 0.0191628, 0.39238, 0.556407
tmfreqs:, 14, 0.0909091, 0, 0, 0.0909091, 0.0382576, 0.385699, 0.563007
tmfreqs:, 15, 0.136364, 0, 0, 0.136364, 0.0572075, 0.375648, 0.572917
tmfreqs:, 16, 0.181818, 0, 0, 0.181818, 0.0759154, 0.363268, 0.585086
tmfreqs:, 17, 0.227273, 0, 0, 0.227273, 0.0942467, 0.349453, 0.598602
tmfreqs:, 18, 0.272727, 0, 0, 0.272727, 0.111999, 0.334938, 0.612709
tmfreqs:, 19, 0.318182, 0, 0, 0.318182, 0.128848, 0.320402, 0.626699
tmfreqs:, 20, 0.363636, 0, 0, 0.363636, 0.144244, 0.306622, 0.639774
tmfreqs:, 21, 0.409091, 0, 0, 0.409091, 0.157246, 0.294688, 0.650875
tmfreqs:, 22, 0.454545, 0, 0, 0.454545, 0.166338, 0.286204, 0.658581
tmfreqs:, 23, 0.5, 0, 0, 0.5, 0.169682, 0.283059, 0.661386
```

The columns before the bands give the index, coordinates and magnitudes of the wavevectors $\vec{k}$ used. The coordinates are defined so that:

$$\vec{k} = k_1\vec{b_1} + k_2\vec{b_2} + k_3\vec{b_3} \tag{3.1}$$

So if we define $k_x = \vec{k} \cdot \vec{x}$, we have $k_1 = \frac{k_x}{2\pi/a}$ in this case where a cartesian lattice was used.

7. Open Matlab and set its working directory to the *"example_DBR"* directory.

8. You should now be able to successfully run the following command in *Matlab*:

```
s = MPB_load_data('example_DBR.dat');
```

This will have created a structure *s* containing the various columns of *example_DBR.dat*, as well as one array of size $N_{k-points} \times N_{bands}$ for the bands.

9. Now you can plot the bands with a simple plot command such as:

```
plot(s.k1, s.fn);
```

to plot the normalized frequencies $f_N = a/\lambda$, as a function of the $k_1 = \frac{k_x}{2\pi/a}$.

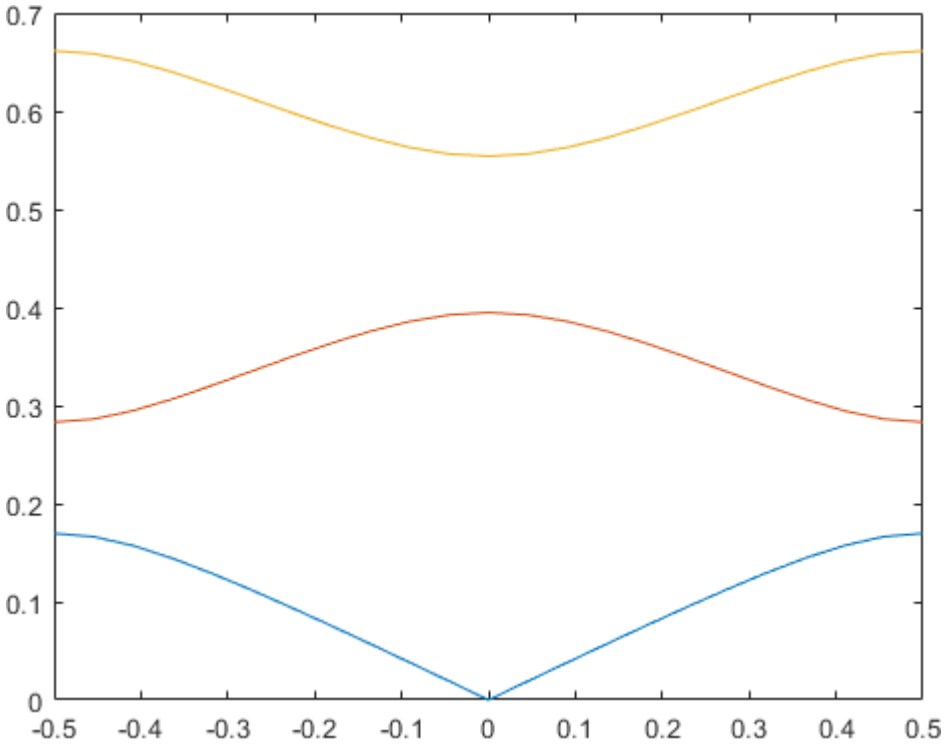


Figure 3.12: Our first photonic bandstructure!

10. The first bandgap is between the top of the first band and the bottom of the second band, i.e. the maximum of the first band and the minimum of the second band. So to get those values, we run the following Matlab commands:

```
>> max(s.fn(:,1))

ans =

    0.1697

>> min(s.fn(:,2))

ans =

    0.2831
```

11. These are normalized frequencies $f_n = a/\lambda$, so to get the corresponding wavelengths λ, we just need to calculate a/f_n:

```
>> a = 0.32; % um
>> a/max(s.fn(:,1))
ans =
    1.8859
>> a/min(s.fn(:,2))
ans =
    1.1305
```

We can now answer the questions 2-4:

- The range of the fundamental bandgap is: $1.131\mu m$ - $1.886\mu m$.
- The midgap wavelength is $\lambda_0 \simeq 1.508\mu m$.
- The gap-midgap ratio is $\frac{\Delta\omega}{\omega_0} \simeq 50.085\%$.

3.4.4 Final scripts and common workflow

If you had any trouble following the above instructions, listing 1 shows the final .ctl file you should obtain.

In addition, it is possible to simplify the process of running MPB and plotting the results by combining all commands entered in the Cygwin terminal and Matlab in scripts, which are just plain text files with appropriate extensions.

In the case of this example:

- The Cygwin terminal commands go into a bash script named ***example_DBR_final.sh***, as in listing 2.
- The Matlab commands go into a Matlab script named ***example_DBR_final.m***, as in listing 3.

3.4.4.1 Preliminary setup:

1. Create a directory named *"example_DBR"* in your *HOME* directory (i.e. in *"cygwin_home"*). You can also re-use the one from the previous section.
2. Copy the following files into ***example_DBR*** from the resource directory ***KD6041-resources\MPB*** (or create them):
   - ***example_DBR_final.ctl*** (listing 1)
   - ***example_DBR_final.sh*** (listing 2)
   - ***example_DBR_final.m*** (listing 3)

```
;;;;; Define new parameters.
(define-param n1 1)
(define-param n2 3.2)

;;;;; Set parameters.
(set-param! resolution 64) ;; Set resolution.
(set-param! num-bands 3) ;; Set number of bands.

;;;;; Define the geometry lattice.
(set! geometry-lattice
  (make lattice
    (size 1 no-size no-size)
  )
)

;;;;; Define the geometry.
(set! geometry (list
  (make block
    (center -0.25 0 0)
    (material (make dielectric (index n1)))
    (size 0.5 infinity infinity)
  )
  (make block
    (center 0.25 0 0)
    (material (make dielectric (index n2)))
    (size 0.5 infinity infinity)
  )
))

;;;;; Define the desired k-points.
(set! k-points (list
  (vector3 -0.5 0 0)
  (vector3 0 0 0); Gamma
  (vector3 0.5 0 0)
))
(set! k-points (interpolate 10 k-points)) ;; add extra k-points

;;;;; Run the simulation.
(run-tm)
```

Listing 1: MPB CTL script: *KD6041-resources\MPB\example_DBR_final.ctl*

```
#!/bin/bash
mpb example_DBR_final.ctl | tee example_DBR_final.out
grep freq example_DBR_final.out > example_DBR_final.dat
mpb-data -m 3 example_DBR_final-epsilon.h5
h5topng -X10 -Y100 example_DBR_final-epsilon.h5:data-new
```

Listing 2: Bash script: *KD6041-resources\MPB\example_DBR_final.sh*

```
% optional: clean up before running script
close all;
clear all;

% load data
s = MPB_load_data('example_DBR_final.dat');
a = 0.32; % um

% get edges of the fundamental bandgap (between band 1 and 2)
gap_edges_fn = [max(s.fn(:,1)), min(s.fn(:,2))];
gap_edges_lambda = a./ gap_edges_fn;

% create new figure
figure;

% plot against normalized frequency
subplot(1,2,1);
plot(s.k1, s.fn);
xlabel('k_x/(2\pi/a)');
ylabel('a/\lambda (no unit)');
% highlight edges and middle of the bandgap
yline(gap_edges_fn(1), 'k--'); % gap edge
yline(mean(gap_edges_fn), 'k--'); % gap middle
yline(gap_edges_fn(2), 'k--'); % gap edge

% plot against wavelength
subplot(1,2,2);
plot(s.k1, a ./ s.fn);
xlabel('k_x/(2\pi/a)');
ylabel('\lambda (\mum)');
ylim([gap_edges_lambda(1)-1, gap_edges_lambda(2)+1]);
% highlight edges and middle of the bandgap
yline(gap_edges_lambda(1), 'k--'); % gap edge
yline(mean(gap_edges_lambda), 'k--'); % gap middle
yline(gap_edges_lambda(2), 'k--'); % gap edge

% print out gap information
fprintf('The range of the fundamental bandgap is: %.3f um - %.3f um\n',...
  min(gap_edges_lambda), max(gap_edges_lambda));
fprintf('The midgap wavelength is: %.3f um\n',...
  mean(gap_edges_lambda));
fprintf('The gap-midgap ratio is: %.3f%%\n',...
  100*(max(gap_edges_fn)-min(gap_edges_fn))/mean(gap_edges_fn));

% save figure
saveas(gcf, 'example_DBR_final.svg');
```

Listing 3: Matlab script: *KD6041-resources\MPB\example_DBR_final.m*

3.4.4.2 The workflow:

Once you created those files, you can create a plot like the one in figure 3.13 (an improved version of figure 3.12) by doing the following:

1. Run this in the *Cygwin terminal* to change into the directory containing your files:

```
cd ~/example_DBR
```

2. Run the bash script ***example_DBR_final.sh*** *(listing 2)* in the *Cygwin terminal* to generate the *.dat file* containining the band data:

```
bash example_DBR_final.sh
```

3. Open ***example_DBR_final.m*** *(listing 3)* in *Matlab* and run it from there to create the plot.

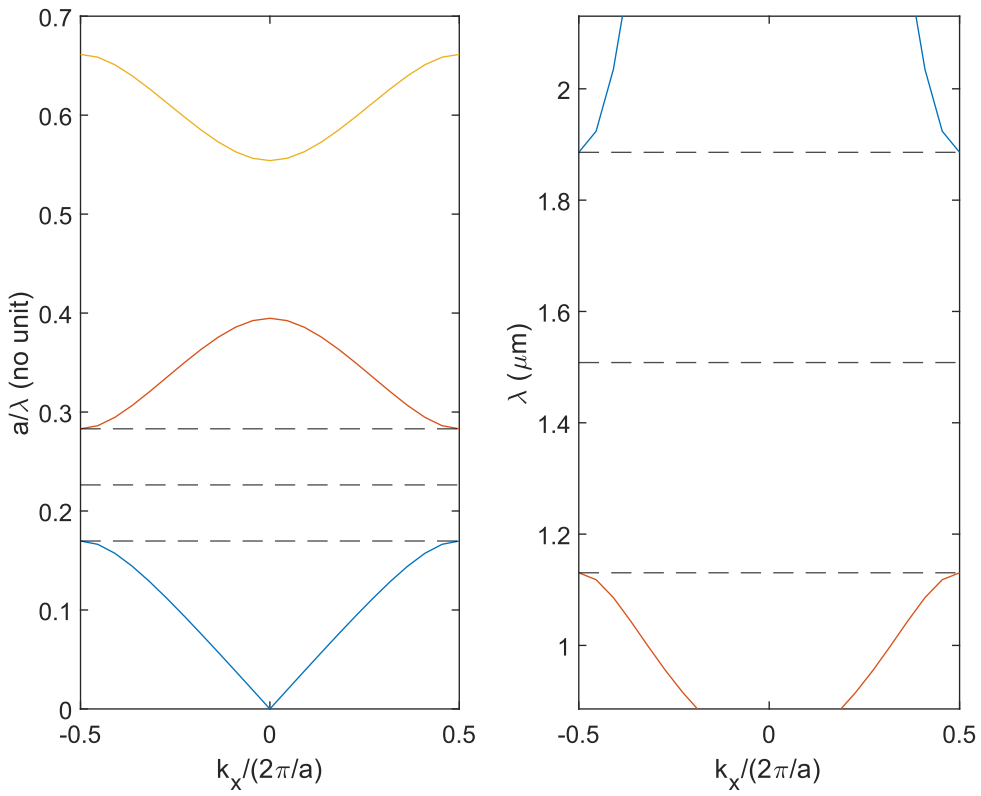


Figure 3.13: Bandstructure of the DBR. The dashed lines show the edges and centre of the fundamental bandgap.

3.4.5 Comparison between TMM and PWE

To compare the results from the PWE method with those from a TMM simulation, we will need to export the data from the Python script, so we can import it in Matlab. This can easily be done with a code of this form to save the python variables *"x"* and *"y"* to a .mat files named *"TMM_data.mat"*:

```
from scipy.io import savemat
mdic = {"x": x, "y": y}
savemat("TMM_data.mat", mdic)
```

Then you can load it into Matlab, by running this command in it:

```
load('TMM_data.mat');
```

The "x" and "y" variables should now be visible in the Matlab workspace.

This will allow you to create the combined plots of figure 3.14 in Matlab.

The scripts to do so are in listings 4 and 5. Copy them into your *example_DBR* directory that you used before.

The procedure to create the plots is similar to the one in section 3.4.4, with the extra step of running the Python script before running the Matlab script, i.e.:

1. Run this in the *Cygwin terminal* to change into the directory containing your files *(already done normally)*:

   ```
   cd ~/example_DBR
   ```

2. Run the bash script ***example_DBR_final.sh*** *(listing 2)* in the *Cygwin terminal* to generate the *.dat file* containining the band data *(already done normally)*:

   ```
   bash example_DBR_final.sh
   ```

3. Run ***DBR_TMM_vs_PWE.py*** *(listing 4)* in Spyder. *(The extra step.)*

4. Open ***DBR_TMM_vs_PWE.m*** *(listing 5)* in *Matlab* and run it from there to create the plot. *(The new Matlab script.)*

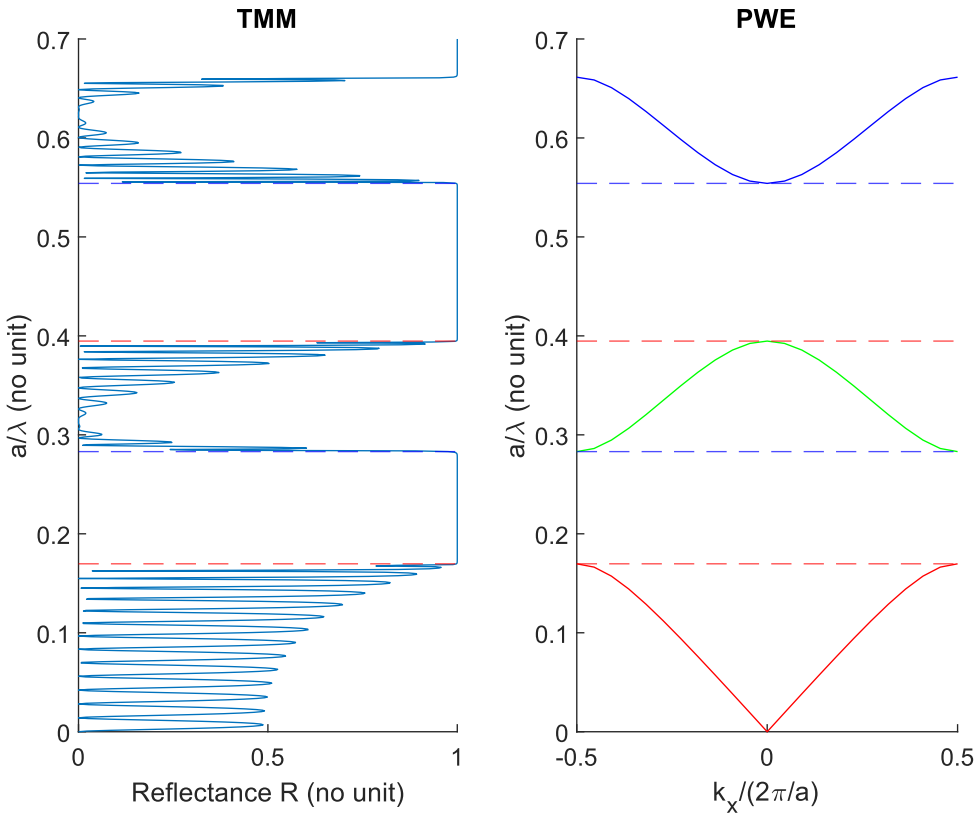


Figure 3.14: Comparison of the reflectance obtained using TMM and the photonic bandstructure. The red and blue dashed lines indicate the bottom and top of the photonic bandgaps respectively.

From figure 3.14, we can see that the peaks in reflection correspond to the gaps in the photonic bands. This is because the photonic bands correspond to solutions of the master equation (5.10), i.e. they represent valid frequency, wavevector combinations $(\omega, \vec{k})$ at which the light can travel through the photonic crystal. Inside the bandgaps, there are no solutions, which means the light cannot travel through the photonic crystal. And since it is also not absorbed (see assumptions made in section §5.3), it is strongly reflected instead.
Note however that we have only considered wavevectors along the X direction here! So these are not necessarily so-called *"full photonic bandgaps"* where light is reflected from any direction!

```python
from tmm.tmm_core import coh_tmm
import numpy as np
import matplotlib.pyplot as plt

# DBR parameters
n1 = 1
d1 = 0.16 # um
n2 = 3.2
d2 = 0.16 # um
Nperiod = 15
a = d1 + d2

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

# normalized frequency range
fn = np.linspace(0, 0.7, 1000)
# wavelength range
wvl_list = a/fn

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

# plot R as a function of a/lambda
plt.plot(fn, R, label='no defect')
plt.xlabel(r'$a/\lambda$ (no unit)')
plt.ylabel('Reflectance $R$ (no unit)')
plt.title('Reflectance at $0\degree$ incidence')

# Save the data to .mat file that is easy to load in Matlab.
from scipy.io import savemat
mdic = {"wvl_list": wvl_list,
        "R": R,
         'n1': n1,
         'd1': d1,
         'n2': n2,
         'd2': d2,
         'Nperiod': Nperiod,
         'a': a}
savemat("TMM_data.mat", mdic)
```

Listing 4: Python script: *KD6041-resources\MPB\DBR_TMM_vs_PWE.py*

```
% optional: clean up before running script
close all;
clear all;

% load the TMM data
load('TMM_data.mat');
% load the MPB data
MPB_data = MPB_load_data('example_DBR_final.dat');

% converting to double in case a is an integer:
a = double(a);

% get the gap infos
gap_infos = MPB_getGaps(MPB_data);

% set the Y axis range
y_limits = [0,0.7];

% create the figure
figure;

% plot the TMM data
subplot(1,2,1); hold on;
title('TMM');
plot(R, a./wvl_list);
ylabel('a/\lambda (no unit)');
xlabel('Reflectance R (no unit)');
ylim(y_limits);
% highlight edges of the bandgap
yline(gap_infos.fullgaps.bottom, 'r--');
yline(gap_infos.fullgaps.top, 'b--');

% plot the MPB data
subplot(1,2,2); hold on;
title('PWE');
plot(MPB_data.k1, MPB_data.fn(:,1), 'r-');
plot(MPB_data.k1, MPB_data.fn(:,2), 'g-');
plot(MPB_data.k1, MPB_data.fn(:,3), 'b-');
xlabel('k_x/(2\pi/a)');
ylabel('a/\lambda (no unit)');
ylim(y_limits);
% highlight edges of the bandgap
yline(gap_infos.fullgaps.bottom, 'r--');
yline(gap_infos.fullgaps.top, 'b--');

% save the figure
saveas(gcf, 'DBR_TMM_vs_PWE.svg');
```

Listing 5: Matlab script: *KD6041-resources\MPB\DBR_TMM_vs_PWE.m*

3.5 Homework: Extended DBR study

We will now consider the same 1D photonic crystal as in the last session, as illustrated in Figure 3.15. We define the period a as $a = d_1 + d_2$.

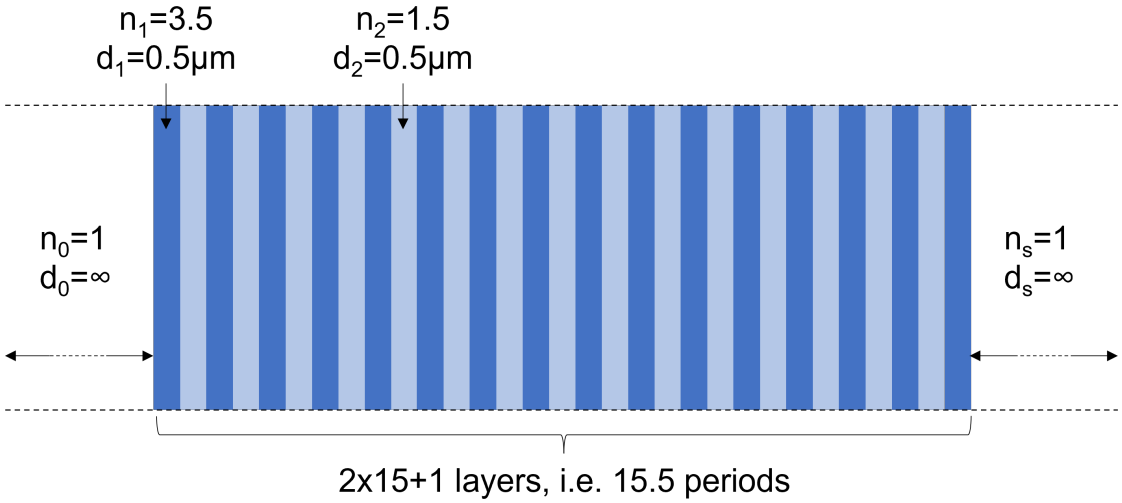


Figure 3.15: Geometry of the DBR example.

1. Theoretical predictions:
   a) Based on the constructive interference criteria for waves reflected at the first n_0/n_1 interface and the ones reflected after the first two layers (n_2/n_1 interface), where do you expect the fundamental bandgap to be (i.e. what central wavelength λ_0)?
   b) On what wavelengths should the other gaps be centered?
   c) What are the corresponding gap positions in terms of normalized frequency a/λ?
2. Simulations:
   a) Plot the corresponding first 6 photonic bands for $k_x/(2\pi/a)$ going from $-1/2$ to $+1/2$ (X axis transverse to the layers, i.e. horizontal in figure 3.15).
   b) Do the photonic bandgaps match the values found in question 1(c)?
   c) Plot the reflectance as a function of the normalized frequency a/λ where a is the period $a = d_1 + d_2$ and λ is the wavelength going over the range $\lambda = 400 - 800nm$.
   d) Create a plot similar to figure 3.14 based on the new DBR parameters to compare the reflectance plot to the photonic bandstructure for $\lambda = 400 - 800nm$. You might have to increase the number of bands to reach higher frequencies.

3. Effect of parameter changes:

   a) We will now focus on the first 6 bands only. So re-use your code from 2(a) to plot the bandstructure against normalized frequency a/λ.

   b) Calculate the index contrast n_1/n_2 and the gap-midgap ratio $\frac{\Delta\omega}{\omega_0}$ of the first gap between band 1 and band 2 for the following parameters:

n_1	3.5	3.5	3.5	4.0	4.5
n_2	2.5	2.0	1.5	1.5	1.5
n_1/n_2					
$\frac{\Delta\omega}{\omega_0}$					

   c) How does the gap-midgap ratio $\frac{\Delta\omega}{\omega_0}$change when you increase the index contrast n_1/n_2?

   d) What happens to the photonic bands and the bandgaps when $n_1 d_1 = n_2 d_2$ (assuming $d_1 \neq d_2$ and $n_1 \neq n_2$ to avoid a completely homogeneous structure)? What are the corresponding values of d_1 and d_2 as a function of the fundamental wavelength λ_0 and the refractive index values n_1 and n_2?

4 MPB Part 2: 2D Photonic crystals

4.1 Direct lattice and reciprocal lattice

Crystals are defined using unit-cells, which are then replicated using lattice vectors $\vec{a_1}$, $\vec{a_2}$, $\vec{a_3}$ as illustrated in figure 4.1.

This lattice in "real space" is called **direct lattice**.

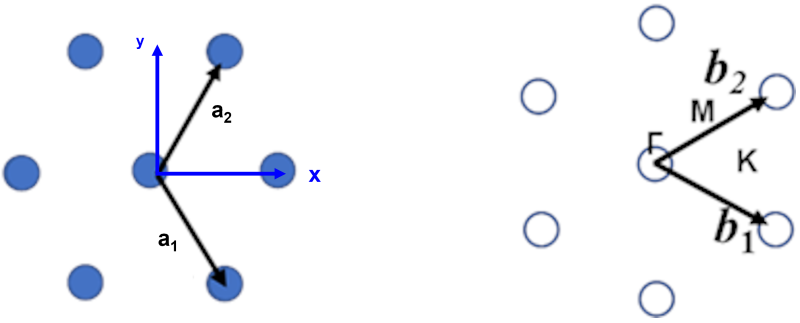


Figure 4.1: The real space and reciprocal space lattice of the triangular (or hexagonal) lattice.

Based on this direct lattice, we can define another lattice in "spatial frequency space" called the **reciprocal lattice** using the following equations to define its basis vectors $\vec{b_1}$, $\vec{b_2}$, $\vec{b_3}$:

$$\begin{cases} \vec{b_1} = \frac{2\pi \vec{a_2} \times \vec{a_3}}{\vec{a_1} \cdot (\vec{a_2} \times \vec{a_3})} \\ \vec{b_2} = \frac{2\pi \vec{a_3} \times \vec{a_1}}{\vec{a_1} \cdot (\vec{a_2} \times \vec{a_3})} \\ \vec{b_3} = \frac{2\pi \vec{a_1} \times \vec{a_2}}{\vec{a_1} \cdot (\vec{a_2} \times \vec{a_3})} \end{cases} \tag{4.1}$$

! See section §5.4 if you need a reminder on vector operations.

4.1.1 Other crystallography concepts

- **Unit-cell:** A unit cell is a repeating unit formed by the vectors spanning the points of a lattice.
- **Primitive unit-cell:** A unit-cell of minimum area or volume.
- **Wigner-Seitz cells:** The Wigner-Seitz cell is a primitive cell which has been constructed by applying Voronoi decomposition to a crystal lattice. See https://en.wikipedia.org/wiki/Wigner%E2%80%93Seitz_cell for more information.
- **Brillouin zone:** The Wigner–Seitz cell in the reciprocal space.
- **Irreducible Brillouin zone:** The Brillouin zone reduced by all of the symmetries in the point group of the lattice (point group of the crystal).
- **Critical points (or high symmetry points):** Corners and centres of the irreducible Brillouin Zone with standard symbols/labels. See https://en.wikipedia.org/wiki/Brillouin_zone#Critical_points.

4.1.1.1 Constructing the Wigner-Seitz cell

Figure 4.2 illustrates the construction of a Wigner–Seitz primitive cell.

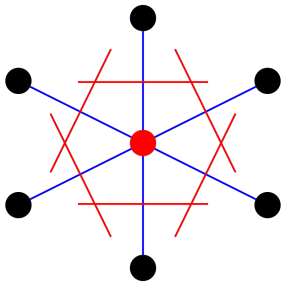


Figure 4.2: Construction of a Wigner–Seitz primitive cell.

1. The cell may be chosen by first picking a lattice point.
2. After a point is chosen, lines are drawn to all nearby lattice points.
3. At the midpoint of each line, another line is drawn normal to each of the first set of lines.
4. The smallest area enclosed in this way is called the Wigner–Seitz primitive cell.

4.1.1.2 Why the Brillouin zone matters

When using the Plane-Wave Expansion method, the solutions ω and $\vec{H}$ for any wavevector $\vec{k'} = \vec{k} + \vec{G}$ where $\vec{G} = l \cdot \vec{b_1} + m \cdot \vec{b_2} + n \cdot \vec{b_3}$ and l, m, n are integers will be the same as for the wavevector $\vec{k}$. See figure 4.3.

For this reason, it is sufficient to only consider wavevectors $\vec{k}$ inside the Brillouin zone.

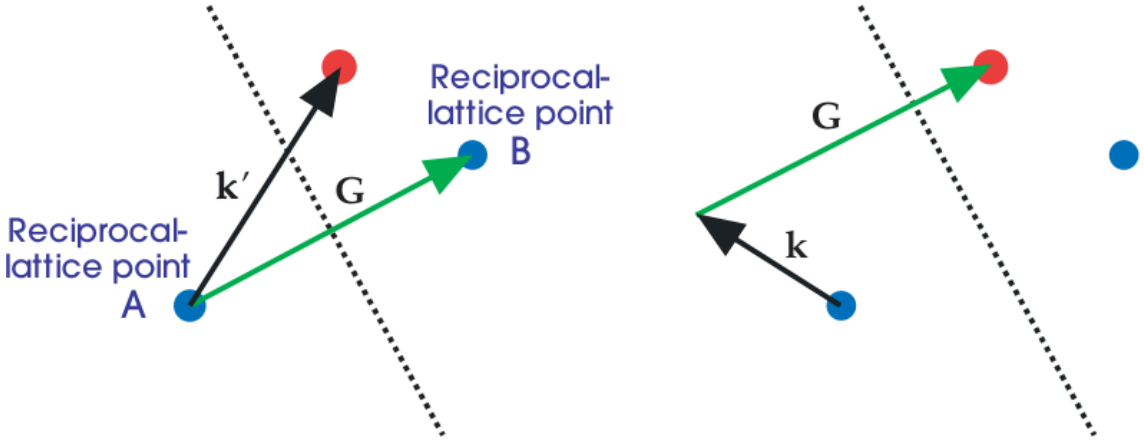


Figure 4.3: Characterization of the Brillouin zone. The dotted line is the perpendicular bisector of the line joining two reciprocal lattice points (blue). If we choose the left point as the origin, any lattice vector (such as$\vec{k'}$) that reaches to an arbitrary point on the other side (red) can be expressed as the sum of a same-side vector (such as $\vec{k}$) plus a reciprocal lattice vector $\vec{G}$.

! See appendix B in reference [4] for more details.

4.1.2 General implementation in MPB

- How to define the *direct lattice* in MPB:

```
(set! geometry-lattice
  (make lattice
    (basis1 a1x a1y a1z) ; direction of a1
    (basis2 a2x a2y a2z) ; direction of a2
    (basis3 a3x a3y a3z) ; direction of a3
    (basis-size a1 a2 a3) ; length of the lattice vectors a1,a2,a3
    (size 1 1 1) ; number of unit-cells along each lattice vector
  )
)
```

> **!** *basis1*, *basis2* and *basis3* are the three lattice directions of the crystal, specified in the cartesian basis. The lengths of these vectors are ignored--only their directions matter. The lengths are determined by the *basis-size* property. These vectors are then used as a basis for all other 3-vectors in the ctl file (except *k-points*, see below). They default to the x, y, and z directions, respectively.

The *size* property can be used to reduce the dimensionality of the computation, as we have seen before:

- 1D: `(size 1 no-size no-size)`
- 2D: `(size 1 1 no-size)`
- 3D: `(size 1 1 1)`

- How to define the wavevectors $\vec{k}$ (*k-points*) to solve for in MPB:

```
(set! k-points (list
  (vector3 k1 k2 k3); k = k1*b1 + k2*b2 + k3*b3
  ...
  (vector3 k1 k2 k3); k = k1*b1 + k2*b2 + k3*b3
))
```

> **!** The k-points are defined in the reciprocal lattice, i.e. the real wavevector in the example above is $\vec{k} = k_1\vec{b_1} + k_2\vec{b_2} + k_3\vec{b_3}$.

See https://mpb.readthedocs.io/en/latest/Scheme_User_Interface/#lattice for the official documentation.

4.2 Guided example: The square lattice, a 2D photonic crystal

4.2.1 Calculate the reciprocal lattice vectors

Figure 4.4 shows a simple *square lattice*. The *direct lattice* vectors are defined as follows:

$$\begin{cases} \vec{a_1} = a_{12}\vec{x} \\ \vec{a_2} = a_{12}\vec{y} \\ \vec{a_3} = a_3\vec{z} \end{cases} \tag{4.2}$$

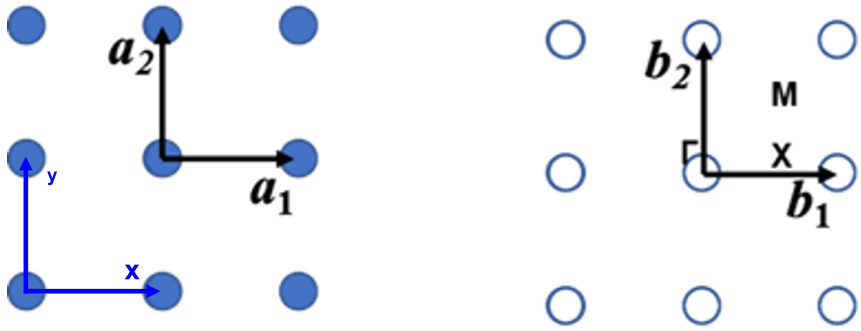


Figure 4.4: The real space and reciprocal space lattice of the square lattice.

1. Draw the following on top of figure 4.4:
   a) On the real-space lattice:
      i. The *Wigner-Seitz cell.*
      ii. One *alternative unit cell*
   b) On the reciprocal lattice:
      i. The *Brillouin zone*
      ii. An *irreducible Brillouin zone* with the critical points Γ, X, M:
         A. Γ: Center of the Brillouin zone
         B. X: Center of an edge
         C. M: Corner
2. Calculate the coordinates in the provided cartesian basis ($\vec{x}$, $\vec{y}$, $\vec{z} = \vec{x} \times \vec{y}$) (in blue in figure 4.10) of the following:
   a) The real (direct) lattice basis vectors: $\vec{a_1}$, $\vec{a_2}$, $\vec{a_3}$
   b) The reciprocal lattice basis vectors: $\vec{b_1}$, $\vec{b_2}$, $\vec{b_3}$
   c) The coordinates of the critical points: Γ, X, M

4.2.1.1 Solution

Figure 4.5: The real space and reciprocal space lattice of the square lattice. The Wigner-Seitz cell (red) and one alternative unit cell (grey dotted) are indicated. The full Brillouin zones are outlined (green) and the irreducible Brillouin zones shown (green shaded area).

Question 1

Question 2 See equation (4.1) for the general formula for calculating the reciprocal lattice vectors. Since all three formulas for $\vec{b_1}$, $\vec{b_2}$, $\vec{b_3}$ have the same denominator $\vec{a_1} \cdot (\vec{a_2} \times \vec{a_3})$, we will start by calculating that.

$$\begin{aligned} \vec{a_1} \cdot (\vec{a_2} \times \vec{a_3}) &= a_{12}\vec{x} \cdot (a_{12}\vec{y} \times a_3\vec{z}) \\ &= a_{12}a_{12}a_3(\vec{x} \cdot (\vec{y} \times \vec{z})) \\ &= a_{12}a_{12}a_3(\vec{x} \cdot \vec{x}) \\ &= a_{12}a_{12}a_3 \end{aligned}$$

The new expressions for the reciprocal lattice vectors are now:

- $\vec{b_1} = \frac{2\pi \vec{a_2} \times \vec{a_3}}{a_{12} a_{12} a_3}$
- $\vec{b_2} = \frac{2\pi \vec{a_3} \times \vec{a_1}}{a_{12} a_{12} a_3}$
- $\vec{b_3} = \frac{2\pi \vec{a_1} \times \vec{a_2}}{a_{12} a_{12} a_3}$

This leads to:

$$\begin{aligned}
\vec{b_1} &= 2\pi \frac{\vec{a_2} \times \vec{a_3}}{a_{12} a_{12} a_3} \\
&= 2\pi \frac{a_{12} a_3}{a_{12} a_{12} a_3} \cdot (\vec{y} \times \vec{z}) \\
&= \frac{2\pi}{a_{12}} \vec{x} \\
\vec{b_2} &= 2\pi \frac{\vec{a_3} \times \vec{a_1}}{a_{12} a_{12} a_3} \\
&= 2\pi \frac{a_3 a_{12}}{a_{12} a_{12} a_3} \cdot (\vec{z} \times \vec{x}) \\
&= \frac{2\pi}{a_{12}} \vec{y} \\
\vec{b_3} &= 2\pi \frac{\vec{a_1} \times \vec{a_2}}{a_{12} a_{12} a_3} \\
&= 2\pi \frac{a_{12} a_{12}}{a_{12} a_{12} a_3} \cdot (\vec{x} \times \vec{y}) \\
&= \frac{2\pi}{a_3} \vec{z}
\end{aligned}$$

The critical points of the Brillouin zone are:

- Γ: Center of the Brillouin zone
- X: Center of an edge
- M: Corner

So $\Gamma = \vec{0}$, while for X and M, there are 4 possible points. Here we will choose the M and X points as defined in figure 4.5, i.e. X is the midpoint of $\vec{b_1}$ and M is the centre of the square formed by $\vec{b_1}$ and $\vec{b_2}$. This allows us to write Γ, X and M in terms of the reciprocal lattice vectors as follows:

$$\begin{cases} \Gamma = \vec{0} \\ X = \frac{1}{2}\vec{b_1} \\ M = \frac{1}{2}(\vec{b_1} + \vec{b_2}) \end{cases}$$

$$\begin{aligned}
X &= \frac{1}{2}\vec{b_1} \\
&= \frac{2\pi}{a_{12}} \frac{\vec{x}}{2} \\
&= \frac{2\pi}{a_{12}} \begin{pmatrix} 1/2 \\ 0 \\ 0 \end{pmatrix} \\
M &= \frac{1}{2}(\vec{b_1} + \vec{b_2}) \\
&= \frac{1}{2}(\frac{2\pi}{a_{12}}\vec{x} + \frac{2\pi}{a_{12}}\vec{y}) \\
&= \frac{2\pi}{a_{12}} \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \end{pmatrix}
\end{aligned}$$

Summary, in column vector form for all:

- $\vec{a_1} = a_{12} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \vec{a_2} = a_{12} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \vec{a_3} = a_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- $\vec{b_1} = \frac{2\pi}{a_{12}} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \vec{b_2} = \frac{2\pi}{a_{12}} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \vec{b_3} = \frac{2\pi}{a_3} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- $\Gamma = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, X = \frac{2\pi}{a_{12}} \begin{pmatrix} 1/2 \\ 0 \\ 0 \end{pmatrix}, M = \frac{2\pi}{a_{12}} \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \end{pmatrix}$

4.2.2 Compute the photonic band structure

We are now going to compute the photonic band structure of a two-dimensional square lattice of dielectric rods in air, based on the geometry defined in figure 4.4 and figure 4.6. The parameters will be as follows:

- Relative permittivity of the rods: $\varepsilon_r = 8.9 \Longrightarrow n_{rods} = \sqrt{\varepsilon_r} \simeq 2.98$
- The background (or backfill) material is air: $\varepsilon_r = 1 \Longleftrightarrow n_{backfill} = 1$
- Radius of the rods: $r = 0.2 \cdot a$ (where $a = a_{12}$)

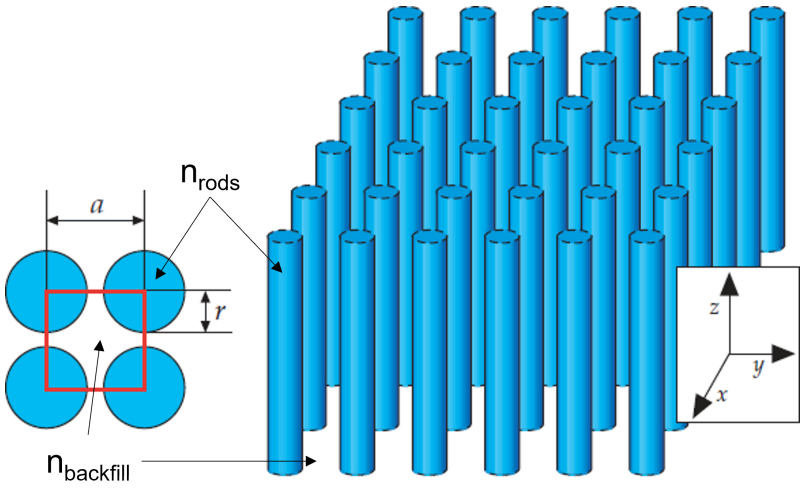


Figure 4.6: The square lattice geometry.

4.2.2.1 Creating the plot

The necessary scripts can be found in *KD6041-resources\MPB\square_lattice* (make sure to update your resource directory like last time) and are shown in listings 6, 7 and 8. The resulting plot is shown in figure 4.7 and the generated geometry images in figure 4.8.

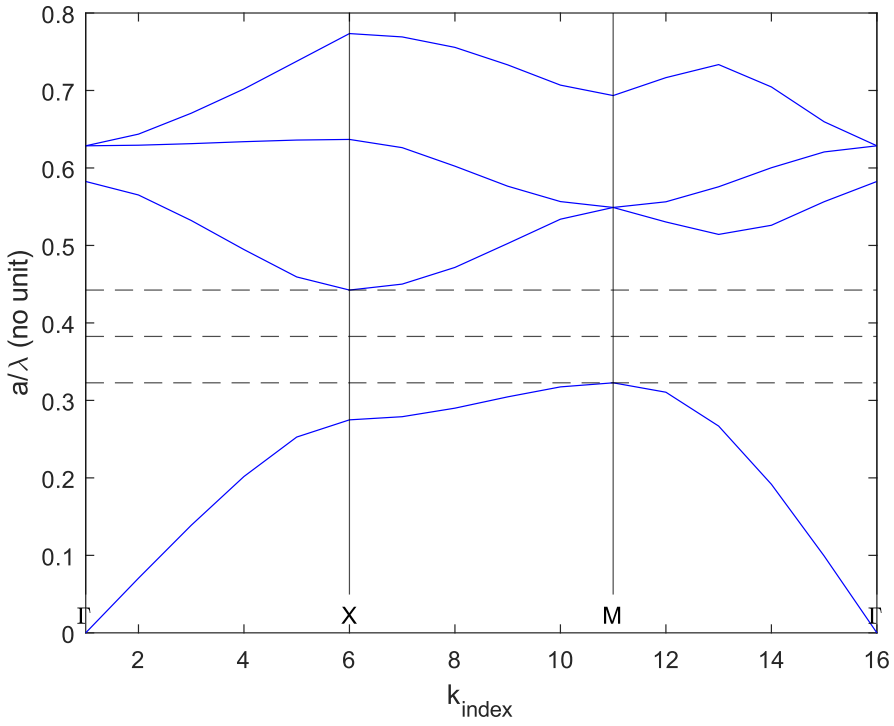


Figure 4.7: The photonic bandstructure of the square lattice of dielectric rods in air for TM modes (magnetic fields in the XY plane, electric field along Z). The dashed horizontal black lines indicate the bottom, middle and top of the fundamental bandgap. The solid vertical black lines indicate the critical k-points Γ, X, M.

Reminder on how to run such a set of scripts:

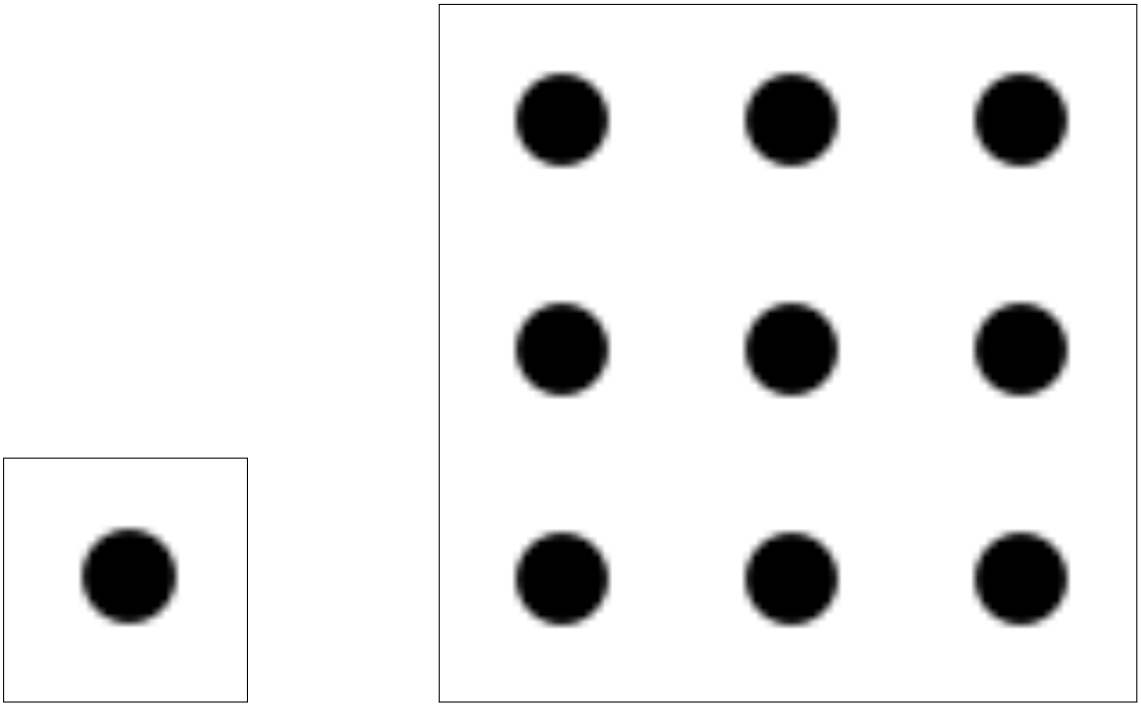
1. Open Cygwin.
2. Change into the directory with the scripts:

```
cd ~/KD6041-resources/MPB/square_lattice
```

3. Run the Bash script in the Cygwin terminal:

```
bash square_lattice.sh
```

4. Open the Matlab script *square_lattice.m* in Matlab and run it from the directory it is in.



(a) The primitive unit-cell.

(b) A crystal made of 3x3 primitive unit-cells.

Figure 4.8: The generated output images.

```
;;;;; Define new parameters.
(define-param a12 1)
(define-param a3 1)

;;;;; Set parameters.
(set-param! num-bands 4)
(set-param! resolution 32)

;;;;; Define the geometry lattice.
(set! geometry-lattice
  (make lattice
    (basis1 1 0 0) ; a1
    (basis2 0 1 0) ; a2
    (basis3 0 0 1) ; a3
    (basis-size a12 a12 a3) ; length of the lattice vectors
    (size 1 1 no-size) ; make the simulation 2D
  )
)

;;;;; Define the geometry.
; The background material.
(set! default-material (make dielectric (epsilon 1)))
; The geometry
(set! geometry
  (list (make cylinder
              (center 0 0 0)
              (radius 0.2)
              (height infinity)
              (material (make dielectric (epsilon 8.9)))
  ))
)

;;;;; Define the desired k-points.
(set! k-points (list (vector3 0 0 0)     ; Gamma -> k_index = 1
                     (vector3 0.5 0 0)   ; X     -> k_index = 1 + 4 + 1= 6
                     (vector3 0.5 0.5 0) ; M     -> k_index = 6 + 4 + 1=11
                     (vector3 0 0 0)))   ; Gamma -> k_index = 11+ 4 + 1=16
; Add 4 points between consecutive initial k-points.
(set! k-points (interpolate 4 k-points))

;;;;; Run the simulation.
(run-tm); run in TM mode
```

Listing 6: MPB CTL script: *KD6041-resources\MPB\square_lattice\square_lattice.ctl*

```
#!/bin/bash
mpb square_lattice.ctl | tee square_lattice.out
grep freq square_lattice.out > square_lattice.dat
# Create an image of the primitive unit-cell:
mpb-data -r -n 320 -m 1 square_lattice-epsilon.h5
h5topng -o square_lattice-1x1.png -X10 -Y10 square_lattice-epsilon.h5:data-new
# Create an image of 3x3 unit-cells:
mpb-data -r -n 320 -m 3 square_lattice-epsilon.h5
h5topng -o square_lattice-3x3.png -X10 -Y10 square_lattice-epsilon.h5:data-new
```

Listing 7: Bash script: *KD6041-resources\MPB\square_lattice\square_lattice.sh*

```
% Defining some default options to label the vertical lines:
set(groot,'defaultConstantLineLabelOrientation', 'horizontal');
set(groot,'defaultConstantLineLabelVerticalAlignment', 'bottom');
set(groot,'defaultConstantLineLabelHorizontalAlignment', 'center');

% load data
s = MPB_load_data('square_lattice.dat');

% get edges of the fundamental bandgap (between band 1 and 2)
gap_edges_fn = [max(s.fn(:,1)), min(s.fn(:,2))];

% plot against normalized frequency
figure; % create new figure
plot(s.kindex, s.fn, 'b-');
xlabel('k_{index}');
ylabel('a/\lambda (no unit)');
xlim([s.kindex(1), s.kindex(end)]);
% highlight edges and middle of the bandgap
yline(gap_edges_fn(1), 'k--'); % gap edge
yline(mean(gap_edges_fn), 'k--'); % gap middle
yline(gap_edges_fn(2), 'k--'); % gap edge
% indicate high-symmetry points
xline(1, 'k-', '\Gamma');
xline(6, 'k-', 'X');
xline(11, 'k-', 'M');
xline(16, 'k-', '\Gamma');
```

Listing 8: Matlab script: *KD6041-resources\MPB\square_lattice\square_lattice.m*

4.2.2.2 MPB code explanations

The main differences compared to the 1D photonic crystal code from last time are:

1. Making the simulation 2D by using `(size 1 1 no-size)`.

2. We used `(set! default-material (make dielectric (epsilon 1)))` to define a backfill material, which can be useful if you want to study the inverse structure with circular air holes in a dielectric.

3. The geometry now only consists of a single cylinder instead of two blocks.

4. We used the critical points Γ, X, M, Γ as main k-points based on the previously calculated coordinates. Our k-points will follow the outline of the triangle visible on the right in figure 4.5.

5. To create the plot, we plot against the k-index this time instead of against $k_x/(2\pi/a)$: `plot(s.kindex, s.fn, 'b-');`

6. We also added vertical lines to indicate the critical points critical k-points Γ, X, M:

```
xline(1, 'k-', '\Gamma');
xline(6, 'k-', 'X');
xline(11, 'k-', 'M');
xline(16, 'k-', '\Gamma');
```

   Why 1, 6, 11 and 16? Because we added 4 interpolated k-points between the 4 initially specified k-points Γ, X, M, Γ, we will have 1+4+1+4+1+4+1=16 k-points in total. However, among those 16 k-points the 4 initial ones correspond to:

   - $k_{index} = 1 \rightarrow \Gamma$
   - $k_{index} = 1 + 4 + 1 = 6 \rightarrow X$
   - $k_{index} = 1 + 4 + 1 + 4 + 1 = 11 \rightarrow M$
   - $k_{index} = 1 + 4 + 1 + 4 + 1 + 4 + 1 = 16 \rightarrow \Gamma$

4.2.2.3 MPB output analysis

It can be useful to look at the output of the MPB script we just ran, which should be visible in the Cygwin terminal after running the Bash script, but also in the generated output file *square_lattice.out*. You can use it to check the reciprocal lattice vectors and k-point coordinates calculated in question 2. See figure 4.9.

4.3 Homework

4.3.1 Calculate the reciprocal lattice vectors of a 2D photonic crystal (triangular/hexagonal lattice)

Figure 4.10 shows a simple *triangular* (also called *hexagonal*) lattice. The *direct lattice* vectors are defined as follows:

$$\begin{cases} \vec{a_1} = a_{12}\left(\frac{1}{2}\vec{x} - \frac{\sqrt{3}}{2}\vec{y}\right) \\ \vec{a_2} = a_{12}\left(\frac{1}{2}\vec{x} + \frac{\sqrt{3}}{2}\vec{y}\right) \\ \vec{a_3} = a_3\vec{z} \end{cases} \tag{4.3}$$

1. Draw the following on top of figure 4.10:
   a) On the real-space lattice:
      i. The *Wigner-Seitz cell.*
      ii. One *alternative unit cell*
   b) On the reciprocal lattice:
      i. The *Brillouin zone*
      ii. An *irreducible Brillouin zone* with the critical points Γ, M, K:
         A. Γ: Center of the Brillouin zone
         B. M: Center of an edge
         C. K: Corner joining two hexagonal faces
2. Calculate the coordinates in the provided cartesian basis ($\vec{x}$, $\vec{y}$, $\vec{z} = \vec{x} \times \vec{y}$) (in blue in figure 4.10) of the following:
   a) The real (direct) lattice basis vectors: $\vec{a_1}$, $\vec{a_2}$, $\vec{a_3}$
   b) The reciprocal lattice basis vectors: $\vec{b_1}$, $\vec{b_2}$, $\vec{b_3}$
   c) The coordinates of the critical points: Γ, M, K

4.3.2 Compute the photonic band structure of a 2D photonic crystal (triangular/hexagonal lattice)

We want to compute the photonic band structure of a two-dimensional hexagonal lattice of air rods in a dielectric slab, as illustrated in figure 4.11. The lattice is the same as in section 4.3.1. The parameters will be as follows:

- $n_{backfill} = 3.2$
- $n_{holes} = 1$

- Distance between holes: $a = 0.4\mu m$, with $a = a_{12} = \|\vec{a_1}\| = \|\vec{a_2}\|$
- Hole radius: $r = 0.14\mu m$, i.e. $r/a = 0.35$

Tasks:

1. Define the corresponding geometry in MPB.
2. Set the *k-points list* in MPB to Γ, M, K, Γ based on the coordinates you found in section 4.3.1, then add 4 extra interpolated points between them.
3. Plot the first eight TE-bands (electric field $\vec{E}$ in the XY plane, i.e. magnetic field $\vec{H}$ aligned with Z) against the previously defined *k-points* in terms of:
   a) normalized frequency a/λ
   b) wavelength in μm.

   ! To compute TE bands instead of TM bands, you can just replace "*run-tm*" with "*run-te*".
4. What is the range of the fundamental TE bandgap in µm? *(use TE mode, resolution=32)*

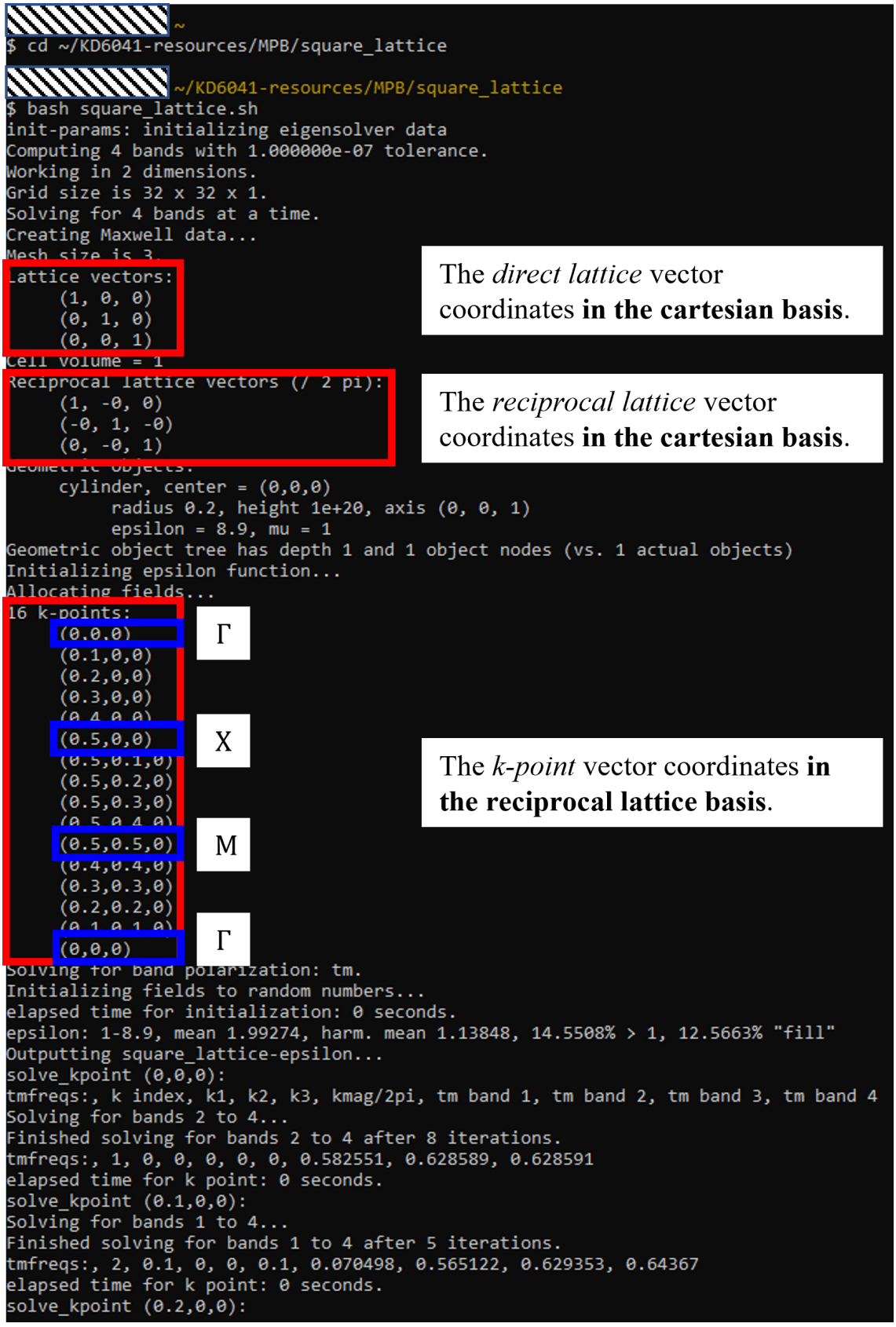


Figure 4.9: You can use MPB to calculate or validate coordinates of reciprocal lattice vectors and k-points.

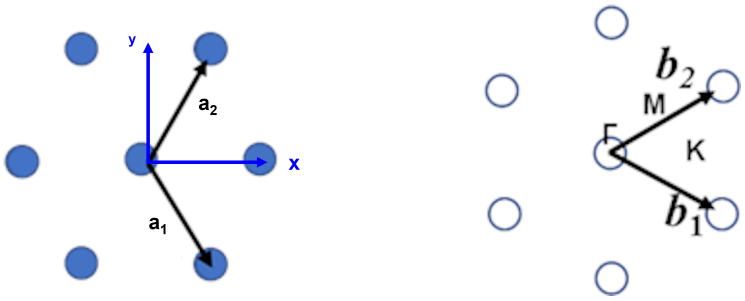


Figure 4.10: The real space and reciprocal space lattice of the triangular (or hexagonal) lattice.

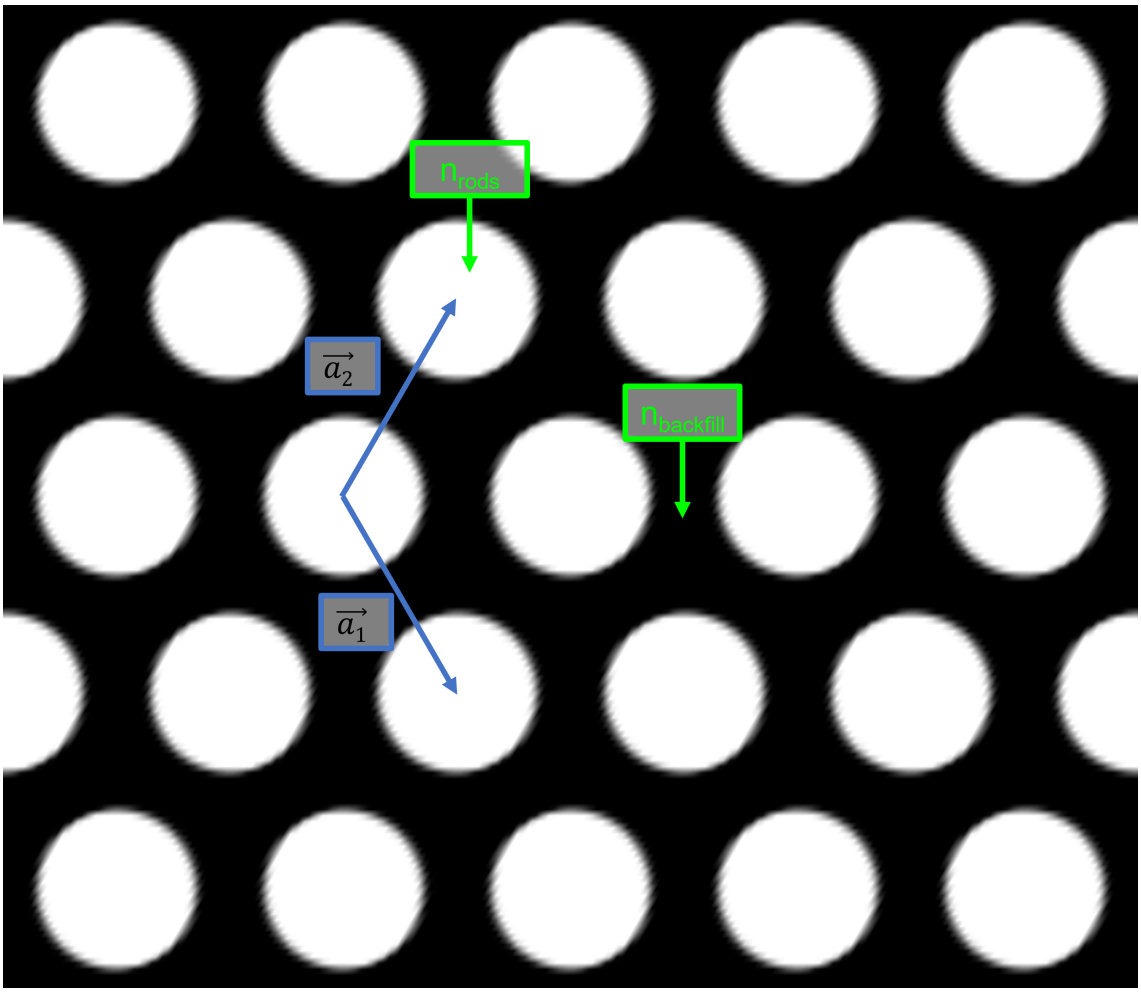


Figure 4.11: An inverse hexagonal photonic crystal.

5 Appendix

5.1 Preliminary software setup for personal computers

Section §1.1 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.

5.2 Introduction to Scheme

5.2.1 Introduction

Both MPB [2] and MEEP [5] 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.

5.2.2 Using Scheme

5.2.2.1 Setting up Notepad++

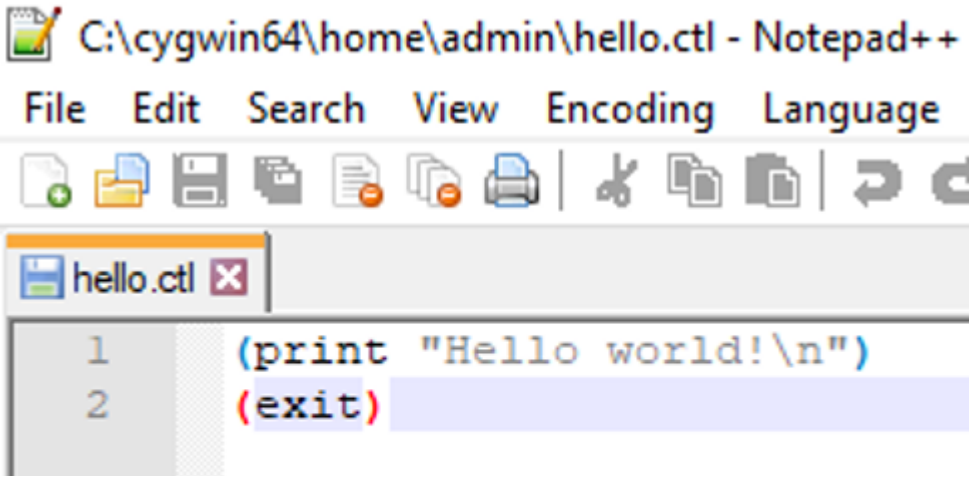
On Windows, a good text editor for Scheme is Notepad++. It is installed on the lab PCs already, but if you are using your own computer, you can get it here:
`https://notepad-plus-plus.org/`
See *KD6041-resources\Documentation\notepad++_tips.pdf* for a recommended setup.

5.2.2.2 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)
```

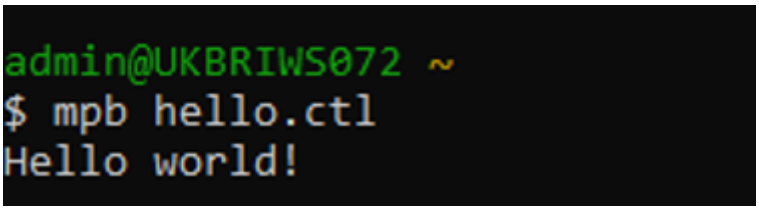
Listing 9: Also available from KD6041-resources\scheme-examples\hello.ctl



4. Now open a *Cygwin terminal* (see figure 1.1) and run the following command in it:

```
mpb hello.ctl
```

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



5.2.2.3 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/

5.3 The Plane-Wave Expansion (PWE) method

The following is based on chapters 2 and 3 from [4]. The propagation of all electromagnetic waves, from radio waves, through visible light, all the way to gamma rays, is governed by the four *microscopic Maxwell equations*, which link the electric and magnetic fields $\vec{E}$ and $\vec{B}$. However, when working with matter on a macroscopic level, it is more convenient to use the four *macroscopic Maxwell equations*, which are defined as follows:

$$\begin{cases} \vec{\nabla} \times \vec{H} - \frac{\partial \vec{D}}{\partial t} &= \vec{J} \\ \vec{\nabla} \times \vec{E} + \frac{\partial \vec{B}}{\partial t} &= \vec{0} \\ \vec{\nabla} \cdot \vec{D} &= \rho \\ \vec{\nabla} \cdot \vec{B} &= 0 \end{cases} \tag{5.1}$$

where $\vec{D}$ and $\vec{H}$ are the displacement and magnetizing fields, and ρ and $\vec{J}$ are the free charge and current densities.

The components of the displacement field $\vec{D}$ are related to the components of the electric field $\vec{E}$ via the power series:

$$\frac{D_i}{\varepsilon_0} = \sum_j \varepsilon_{ij} E_j + \sum_{j,k} \chi^{(2)}_{ijk} E_j E_k + \mathcal{O}(E^3) \tag{5.2}$$

where ε is the dielectric tensor and $\chi^{(2)}$ is the second-order nonlinear susceptibility tensor.

We now make the following approximations:

- $\varepsilon = \varepsilon(\vec{r}, t, \omega)$ does not depend on time: $\frac{\partial \varepsilon}{\partial t} = 0$
- $\varepsilon = \varepsilon(\vec{r}, t, \omega)$ does not depend on frequency: $\frac{\partial \varepsilon}{\partial \omega} = 0$, i.e. we neglect material dispersion. Instead, we simply choose the value of the dielectric constant appropriate to the frequency range of the physical system we are considering.
- No free charges: $\rho = 0$
- No currents: $\vec{J} = \vec{0}$
- We assume field strengths small enough to remain in the linear regime, so that $\chi^{(2)}$ (and all higher-order terms $\chi^{(n)}$) can be neglected:
  $\forall n \geq 2,\ \chi^{(n)} = 0 \Rightarrow D_i/\varepsilon_0 = \sum_j \varepsilon_{ij} E_j$
- We assume the material is macroscopic and isotropic, so that $\varepsilon_{ij} = \delta_{ij} \cdot \varepsilon_r(\vec{r})$, where $\varepsilon_r(\vec{r})$ is a scalar dielectric function, called the relative permittivity. $\vec{D}$ is then simply related to $\vec{E}$ by: $\vec{D} = \varepsilon_0 \varepsilon_r(\vec{r}) \vec{E}$.
- We focus primarily on non-absorbing materials: $\varepsilon_r(\vec{r}) \in \mathbb{R}^+$

- The materials used are not magnetizable at the optical frequencies considered[6]: $\mu_r(\vec{r}) = 1$

With all of these assumptions, $\vec{D}$ and $\vec{H}$ can easily be expressed as a function of $\vec{E}$ and $\vec{B}$ as follows:

$$\vec{D}(\vec{r},t) = \varepsilon_0 \varepsilon_r(\vec{r}) \vec{E}(\vec{r},t) \tag{5.3}$$

$$\vec{H}(\vec{r},t) = \frac{1}{\mu_0} \vec{B}(\vec{r},t) \tag{5.4}$$

And the Maxwell equations simplify to:

$$\begin{cases} \vec{\nabla} \times \vec{H}(\vec{r},t) - \varepsilon_0 \varepsilon_r(\vec{r}) \frac{\partial \vec{E}(\vec{r},t)}{\partial t} &= \vec{0} \\ \vec{\nabla} \times \vec{E}(\vec{r},t) + \mu_0 \frac{\partial \vec{H}(\vec{r},t)}{\partial t} &= \vec{0} \\ \vec{\nabla} \cdot \left[\varepsilon_r(\vec{r}) \vec{E}(\vec{r},t) \right] &= 0 \\ \vec{\nabla} \cdot \vec{H}(\vec{r},t) &= 0 \end{cases} \tag{5.5}$$

Because the Maxwell equations are linear, we can expand the fields into a set of harmonic modes of the form:

$$\begin{cases} \vec{H}(\vec{r},t) = & \vec{H}(\vec{r}) e^{-i\omega t} \\ \vec{E}(\vec{r},t) = & \vec{E}(\vec{r}) e^{-i\omega t} \end{cases} \tag{5.6}$$

The set of equations 5.5 can then be reduced to the following *master equation*:

$$\boxed{\vec{\nabla} \times \left(\frac{1}{\varepsilon_r(\vec{r})} \vec{\nabla} \times \vec{H}(\vec{r}) \right) = \left(\frac{\omega}{c_0} \right)^2 \vec{H}(\vec{r})} \tag{5.7}$$

This corresponds to an eigenvalue problem and means that only a discrete set of eigenvectors $\vec{H}(\vec{r})$ with corresponding eigenvalues $\left(\frac{\omega}{c_0}\right)^2$ are allowed inside the photonic crystal.

In the case of a three-dimensional periodic system, due to the translational symmetries, the modes $\vec{H}(\vec{r})$ take the following form (*Bloch's theorem*):

$$\boxed{\vec{H}_k(\vec{r}) = e^{i\vec{k}\cdot\vec{r}} \cdot \vec{u}_k(\vec{r})} \tag{5.8}$$

where $\vec{k}$ is a *Bloch wave vector* defined in the reciprocal lattice and $\vec{u}_k(\vec{r})$ is a periodic function on the lattice, i.e. $\vec{u}_k(\vec{r}) = \vec{u}_k(\vec{r} + \vec{R})$ for all lattice vectors $\vec{R}$.

Equation 5.7 now becomes:

$$\vec{\nabla} \times \left(\frac{1}{\varepsilon_r(\vec{r})} \vec{\nabla} \times \vec{H}_k(\vec{r}) \right) = \left(\frac{\omega}{c_0} \right)^2 \vec{H}_k(\vec{r}) \tag{5.9}$$

By inserting equation (5.8) into it, we get another form of the *master equation* that depends on $\vec{k}$:

$$\boxed{(i\vec{k}+\vec{\nabla})\times\left(\frac{1}{\varepsilon_r(\vec{r})}(i\vec{k}+\vec{\nabla})\times\vec{u}_k(\vec{r})\right)=\left(\frac{\omega}{c_0}\right)^2\vec{u}_k(\vec{r})} \tag{5.10}$$

The Maxwell equation $\vec{\nabla}\cdot\vec{H}(\vec{r},t)=0$ additionally leads to a transversality condition on $\vec{u}_k(\vec{r})$:

$$(i\vec{k}+\vec{\nabla})\cdot\vec{u}_k(\vec{r})=0 \tag{5.11}$$

Given a real wavevector $\vec{k}$, solving equation (5.10) leads to a discrete set of real ω values, which can be labeled by a band index n. which correspond to non-decaying modes. When $\vec{k}$ changes continuously, so do the $\omega_n(\vec{k})$ solutions.

It is possible to design structures where there is a range of frequencies, for which no real $\vec{k}$ exists yielding a non-decaying mode. This is called a full photonic bandgap. Since we assumed that there is no absorption, this means that frequencies within this range will be perfectly reflected.

5.4 Vector operations

Consider two arbitrary 3D vectors $\vec{u}$ and $\vec{v}$ with the following coordinates in a cartesian basis:

$$\vec{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{5.12}$$

The **dot product** is then defined as:

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + u_3 v_3 \tag{5.13}$$

And the **cross product** as:

$$\vec{u} \times \vec{v} = \begin{pmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{pmatrix} \tag{5.14}$$

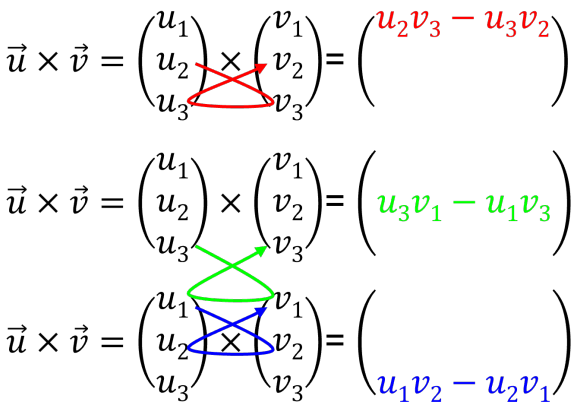
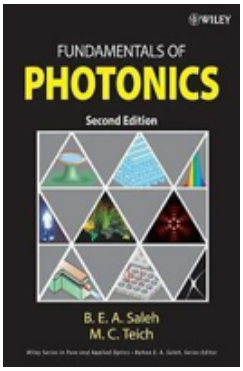
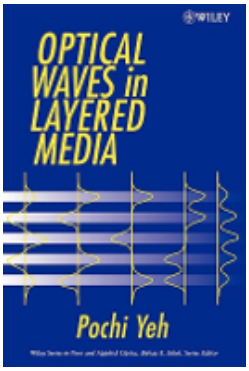


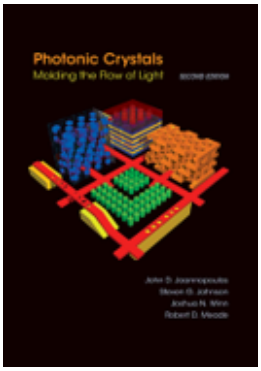
Figure 5.1: One way of remembering the cross-product formula: To calculate each coordinate, draw a "Gamma" symbol between the two "next" rows of coordinates. Keep in mind that it is cyclic, so after 3 you go back to 1. Then you just calculate the difference between the product of the first two coordinates and the last two coordinates.

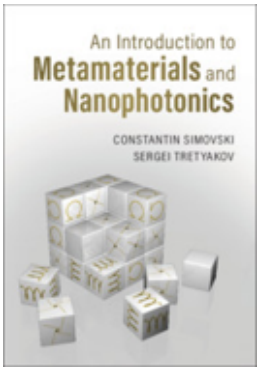
5.5 Useful references

- **Fundamentals of photonics**, Bahaa E. A. Saleh, Malvin Carl Teich [7]
  - 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 [4]
  - 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) [8]
  - 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] S. Johnson and J. Joannopoulos, "Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis." *Optics express*, vol. 8, no. 3, pp. 173–190, jan 2001. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=OPEX-8-3-173

[4] 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/

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

[6] A. M. Fox, *Quantum optics: an introduction*. Oxford University Press, USA, 2006. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=Q-4dIthPuL4C&oi=fnd&pg=PR15&dq=Quantum+Optics+An+Introduction&ots=FHhYNrancH&sig=pKJGSPPWihzKaJT8GEHrFdY2pa4

[7] 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

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