1 MPB Part 1: 1D Photonic crystals

1.1 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)[1] 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 1.1.

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 1.1: Comparison of TMM and PWE.

1.2 The MPB software

The Free software package *MIT Photonic Bands* (MPB)[1] is a software that can solve the *master equation* (2.10) 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[2] and appendix D of [3].

Figure 1.1 summarizes the inputs and outputs of MPB.

One important aspect of *MPB*, as well as its related software for *FDTD* simulations called *MEEP*[4], 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 $f_N = \frac{f}{c_0/a}$.

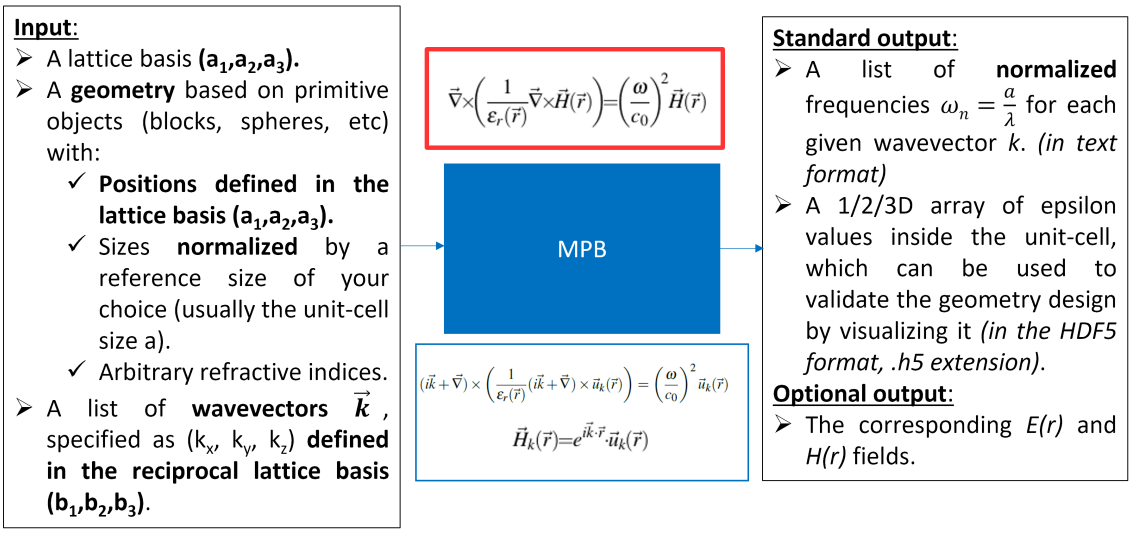


Figure 1.1: Summary of what MPB does.

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 1.2 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 1.2: 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.

1.3 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.

1.3.1 Interactive mode

Let's try the interactive mode first.

1. Open Cygwin as in figure 1.2.

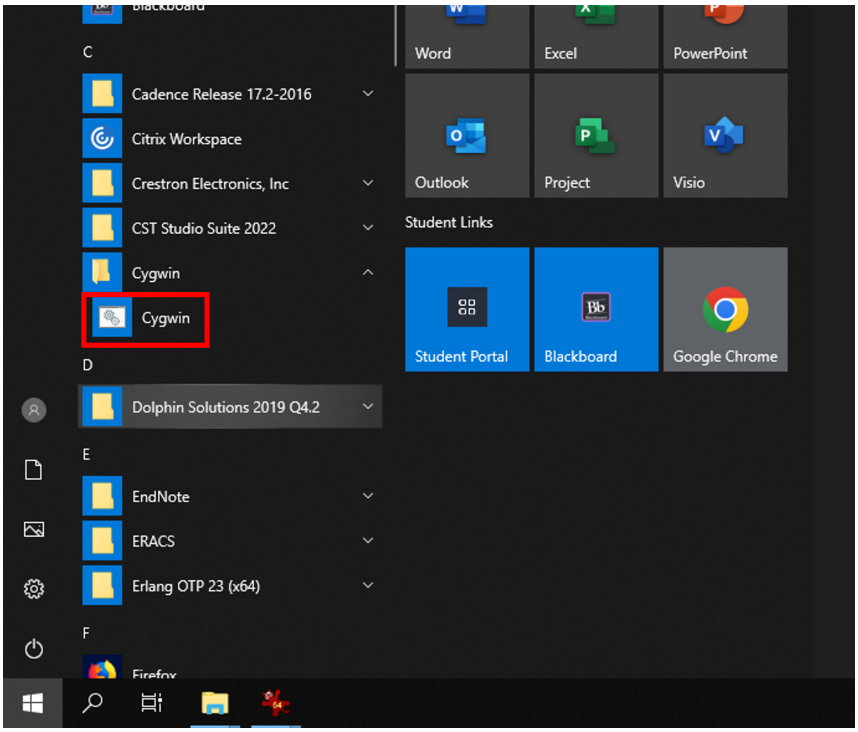


Figure 1.2: Opening Cygwin.

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 1.3 shows more possible commands you can try.

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

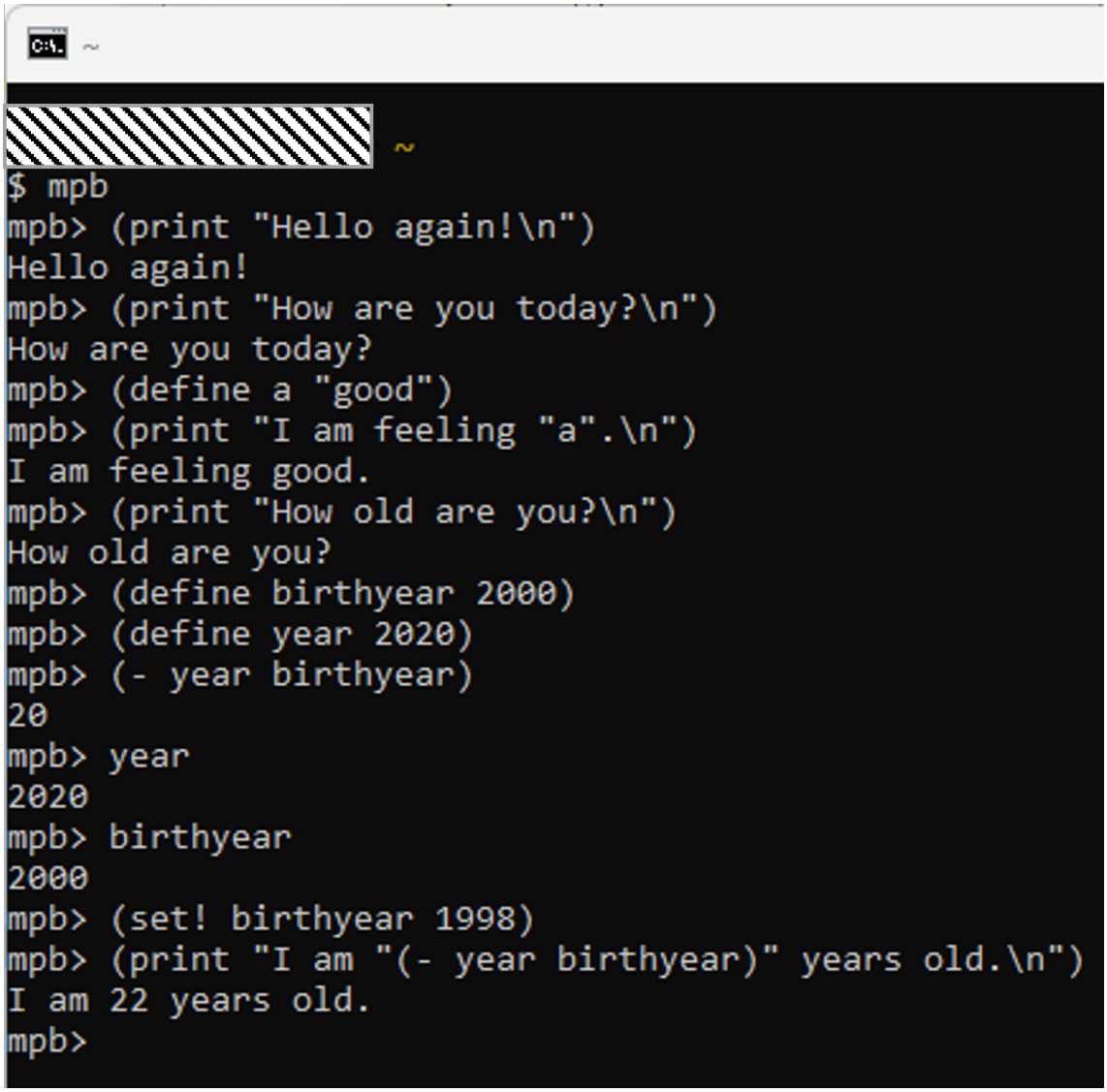


Figure 1.3: MPB's interactive mode.

1.3.2 Script mode

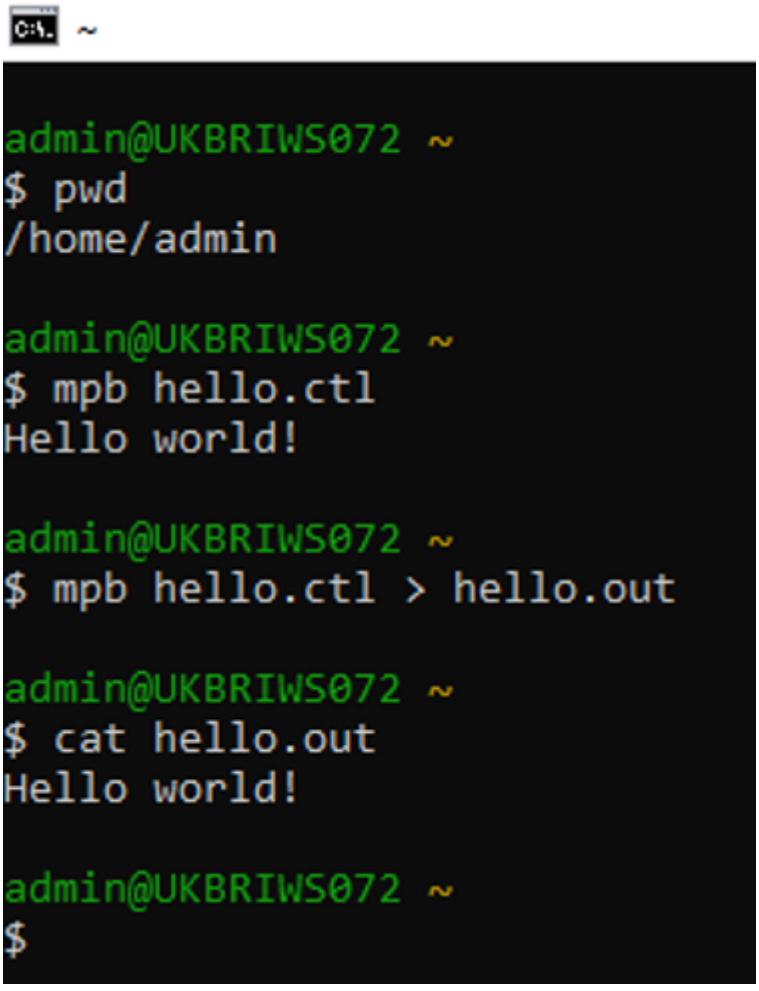
Next let us try the script mode. Before you start, you might want to configure Notepad++ as explained here: *KD6041-resources\Documentation\notepad++_tips.pdf*.

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++*.

1.3.3 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 1.4.

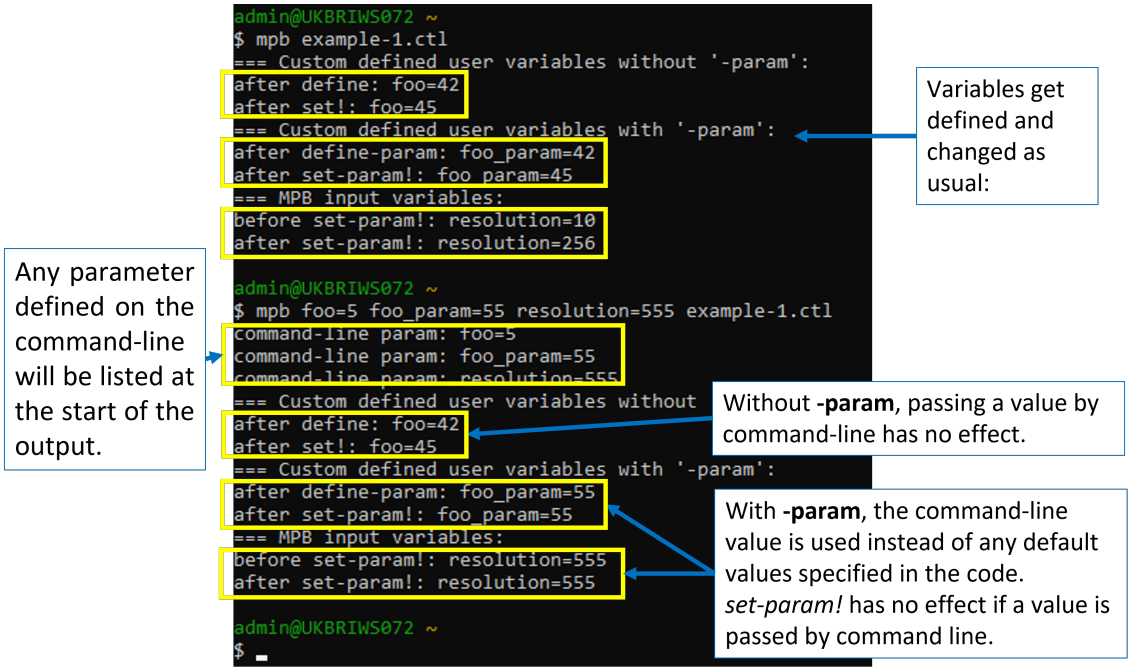


Figure 1.4: Passing parameters by command-line.

1.3.4 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 1.3.1: 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.

1.3.5 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 1.5: *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 1.6: *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.
```

1.3.6 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)
```

1.4 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?

1.4.1 Preliminary calculations

1. First we need to define a reference length. Here we choose $a = d_1 + d_2$ as illustrated in figure 1.7.

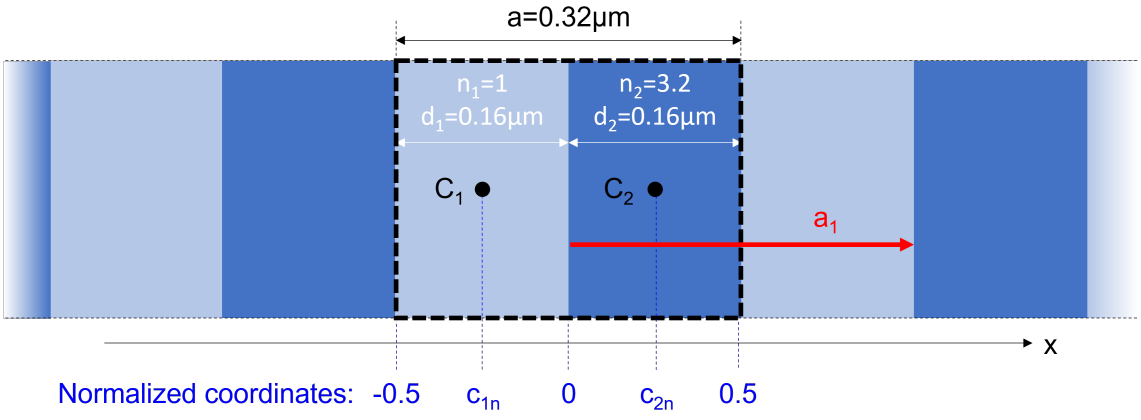


Figure 1.7: The DBR to simulate. For MPB, we only consider a unit-cell as indicated by the dashed black rectangle.

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.:
   - $C_1 = -0.25\vec{a_1} \Rightarrow C_1 = \begin{pmatrix} -0.25 \\ 0 \\ 0 \end{pmatrix}$
   - $C_2 = +0.25\vec{a_1} \Rightarrow C_2 = \begin{pmatrix} +0.25 \\ 0 \\ 0 \end{pmatrix}$

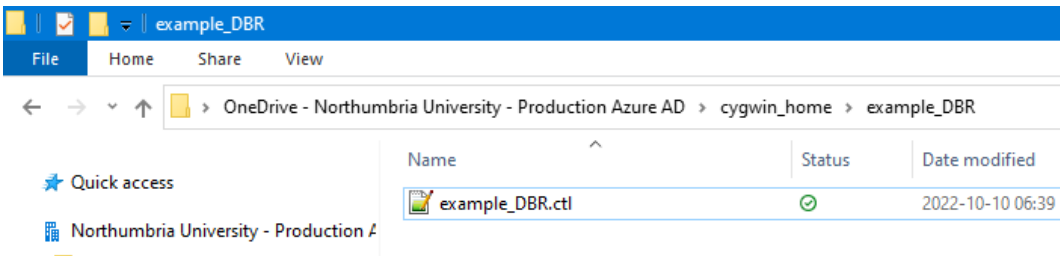
5. Their normalized thicknesses will be:
   - $d_{1n} = \frac{d_1}{a} = 0.5$
   - $d_{2n} = \frac{d_2}{a} = 0.5$

1.4.2 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 1.8).

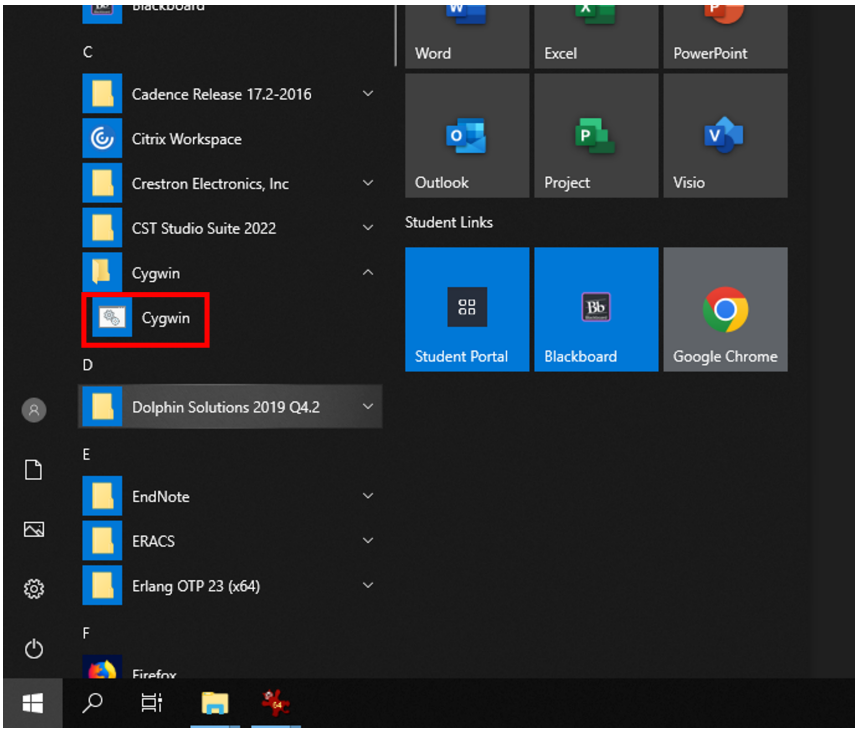


Figure 1.8: Opening Cygwin.

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 1.9. Of course you can also check this in the Windows File Explorer.

```
ls
```

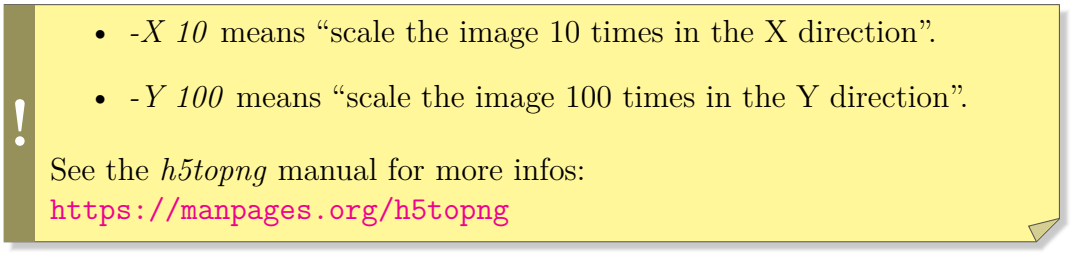
11. This *"example_DBR-epsilon.h5"* file contains the epsilon distribution inside a single unit-cell. In order to visualize 3 unit-cells, we create a new dataset *data-new* in it by running the following command:

```
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 1.10.

Figure 1.10: 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 1.11.

Figure 1.11: Three unit-cells of the DBR using resolution=64.

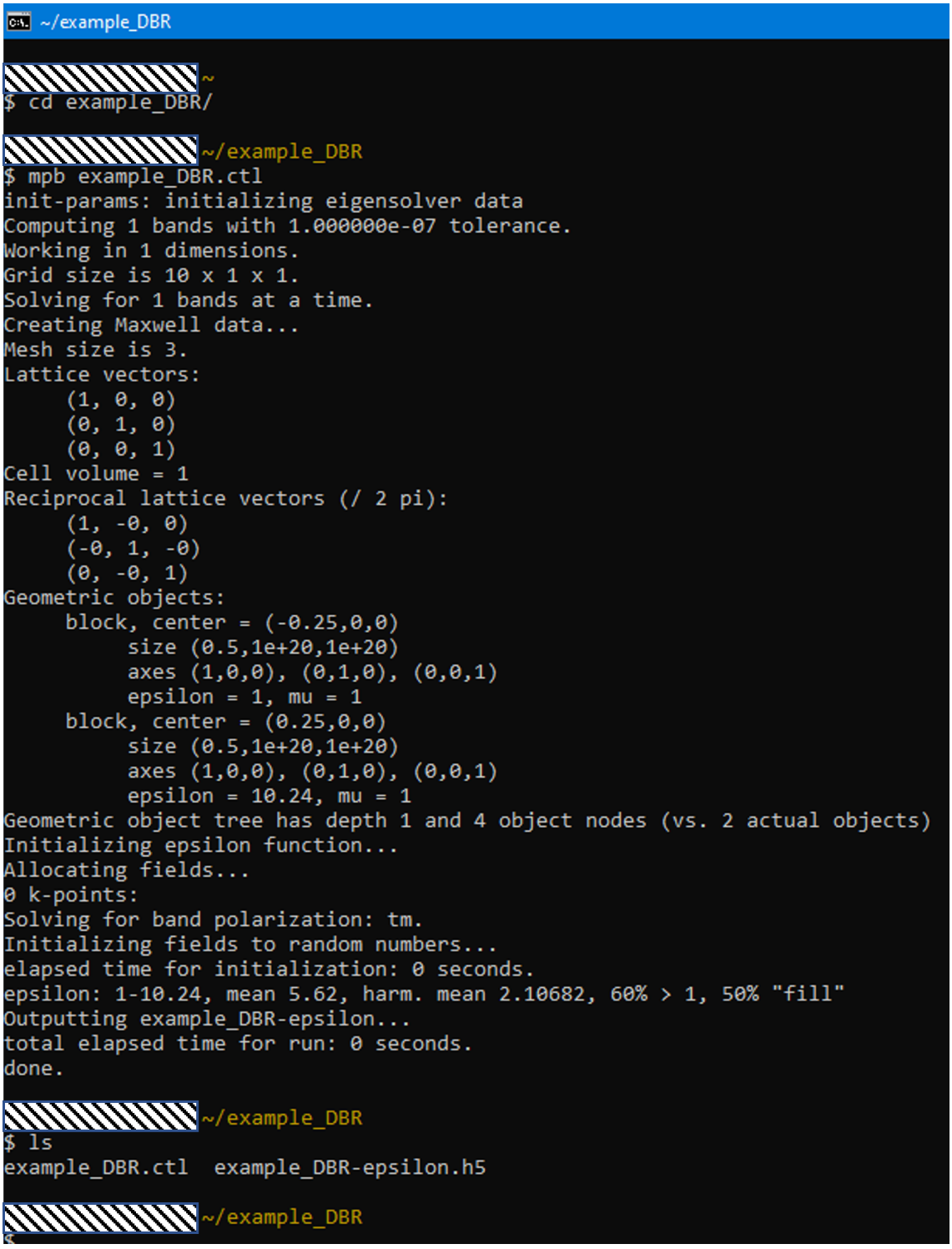


Figure 1.9: Output after the first run without k-points.

1.4.3 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:

$$\vec{k} = k_1\vec{b_1} + k_2\vec{b_2} + k_3\vec{b_3} \tag{1.1}$$

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 $k_1 = \frac{k_x}{2\pi/a}$.

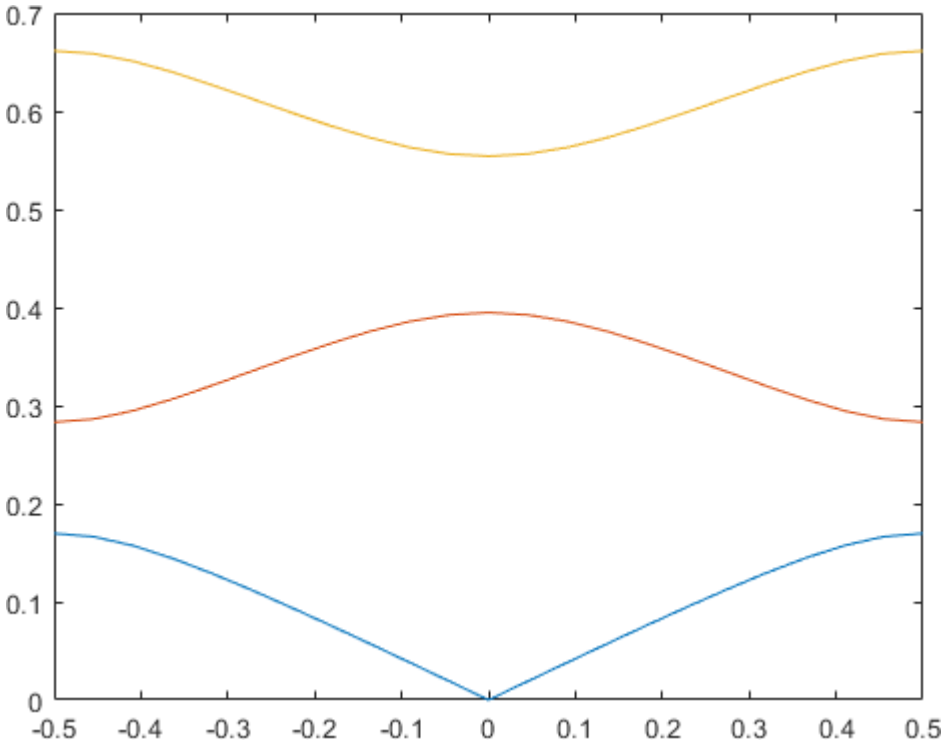


Figure 1.12: Our first photonic bandstructure!

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 $\frac{\Delta\omega}{\omega_0} \simeq 50.085\%$.

1.4.4 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.

1.4.4.1 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*

1.4.4.2 The workflow:

Once you created those files, you can create a plot like the one in figure 1.13 (an improved version of figure 1.12) 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.

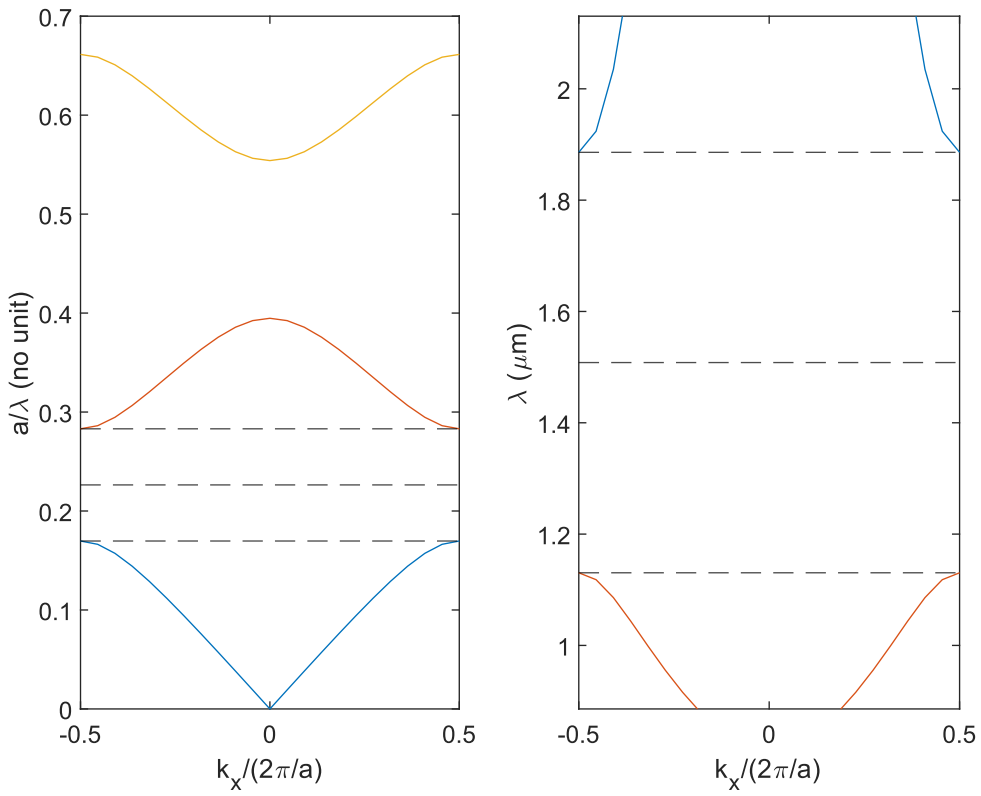


Figure 1.13: Bandstructure of the DBR. The dashed lines show the edges and centre of the fundamental bandgap.

1.4.5 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 1.14 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 1.4.4, 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.)*

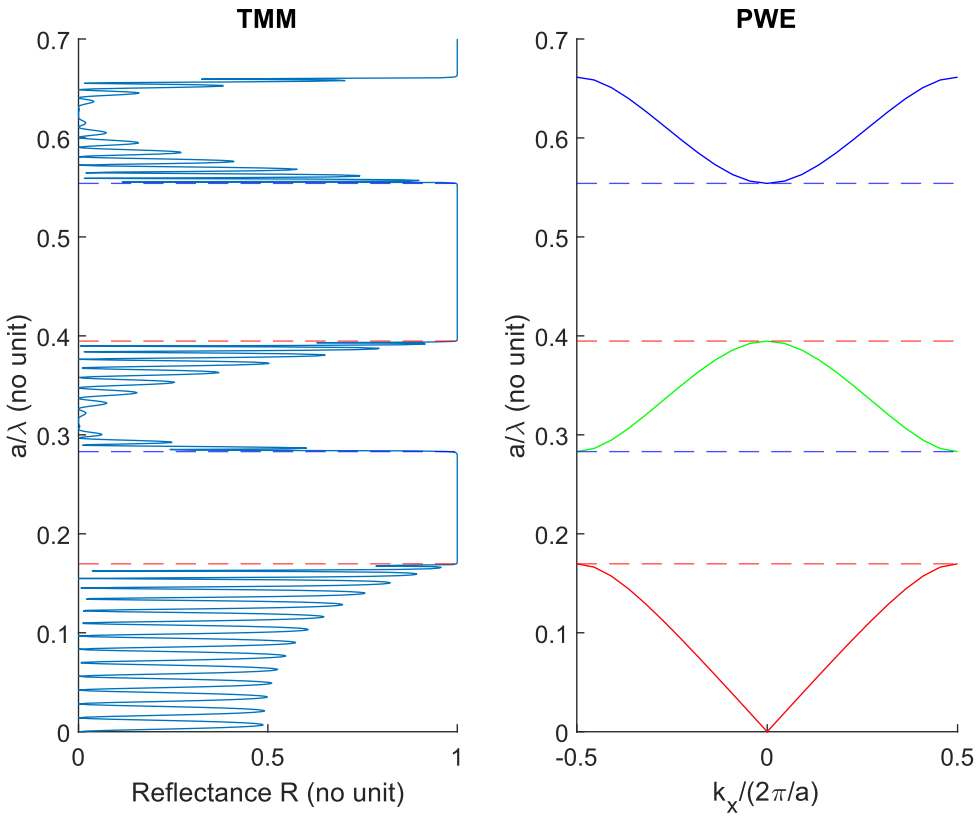


Figure 1.14: Comparison of the reflectance obtained using TMM and the photonic bandstructure. The red and blue dashed lines indicate the bottom and top of the photonic bandgaps respectively.

From figure 1.14, 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 (2.10), 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 §2.1), 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');

% 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*

1.5 Homework: Extended DBR study

We will now consider the same 1D photonic crystal as in the last session, as illustrated in Figure 1.15. We define the period a as $a = d_1 + d_2$.

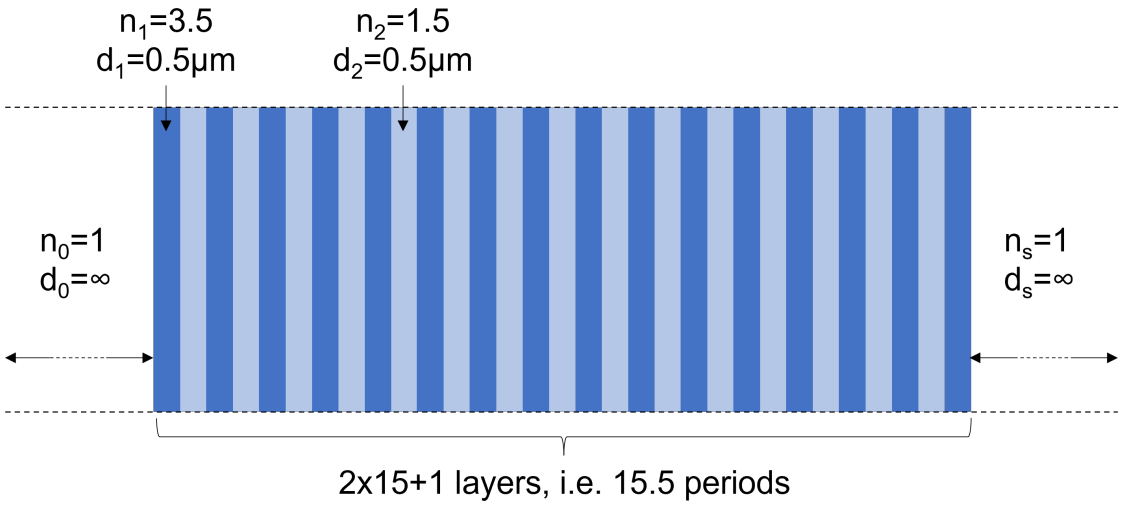


Figure 1.15: Geometry of the DBR example.

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 1.15).
   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 1.14 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?

2 Appendix

2.1 The Plane-Wave Expansion (PWE) method

The following is based on chapters 2 and 3 from [3]. 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:

$$\begin{cases} \vec{\nabla} \times \vec{H} - \frac{\partial \vec{D}}{\partial t} &= \vec{J} \\ \vec{\nabla} \times \vec{E} + \frac{\partial \vec{B}}{\partial t} &= \vec{0} \\ \vec{\nabla} \cdot \vec{D} &= \rho \\ \vec{\nabla} \cdot \vec{B} &= 0 \end{cases} \tag{2.1}$$

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:

$$\frac{D_i}{\varepsilon_0} = \sum_j \varepsilon_{ij} E_j + \sum_{j,k} \chi^{(2)}_{ijk} E_j E_k + \mathcal{O}(E^3) \tag{2.2}$$

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[5]: $\mu_r(\vec{r}) = 1$

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:

$$\vec{D}(\vec{r}, t) = \varepsilon_0 \varepsilon_r(\vec{r})\vec{E}(\vec{r}, t) \tag{2.3}$$

$$\vec{H}(\vec{r}, t) = \frac{1}{\mu_0}\vec{B}(\vec{r}, t) \tag{2.4}$$

And the Maxwell equations simplify to:

$$\begin{cases} \vec{\nabla} \times \vec{H}(\vec{r}, t) - \varepsilon_0 \varepsilon_r(\vec{r}) \frac{\partial \vec{E}(\vec{r}, t)}{\partial t} & = \vec{0} \\ \vec{\nabla} \times \vec{E}(\vec{r}, t) + \mu_0 \frac{\partial \vec{H}(\vec{r}, t)}{\partial t} & = \vec{0} \\ \vec{\nabla} \cdot \left[\varepsilon_r(\vec{r})\vec{E}(\vec{r}, t)\right] & = 0 \\ \vec{\nabla} \cdot \vec{H}(\vec{r}, t) & = 0 \end{cases} \tag{2.5}$$

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

$$\begin{cases} \vec{H}(\vec{r}, t) = & \vec{H}(\vec{r})e^{-i\omega t} \\ \vec{E}(\vec{r}, t) = & \vec{E}(\vec{r})e^{-i\omega t} \end{cases} \tag{2.6}$$

The set of equations 2.5 can then be reduced to the following *master equation*:

$$\boxed{\vec{\nabla} \times \left(\frac{1}{\varepsilon_r(\vec{r})} \vec{\nabla} \times \vec{H}(\vec{r}) \right) = \left(\frac{\omega}{c_0} \right)^2 \vec{H}(\vec{r})} \tag{2.7}$$

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*):

$$\boxed{\vec{H}_k(\vec{r}) = e^{i\vec{k}\cdot\vec{r}} \cdot \vec{u}_k(\vec{r})} \tag{2.8}$$

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 2.7 now becomes:

$$\vec{\nabla} \times \left(\frac{1}{\varepsilon_r(\vec{r})} \vec{\nabla} \times \vec{H}_k(\vec{r}) \right) = \left(\frac{\omega}{c_0} \right)^2 \vec{H}_k(\vec{r}) \tag{2.9}$$

By inserting equation (2.8) into it, we get another form of the *master equation* that depends on $\vec{k}$:

$$\boxed{(i\vec{k} + \vec{\nabla}) \times \left(\frac{1}{\varepsilon_r(\vec{r})} (i\vec{k} + \vec{\nabla}) \times \vec{u}_k(\vec{r}) \right) = \left(\frac{\omega}{c_0} \right)^2 \vec{u}_k(\vec{r})} \tag{2.10}$$

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

$$(i\vec{k} + \vec{\nabla}) \cdot \vec{u}_k(\vec{r}) = 0 \tag{2.11}$$

Given a real wavevector $\vec{k}$, solving equation (2.10) 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.

2.2 Introduction to Scheme

2.2.1 Introduction

Both MPB [1] and MEEP [4] use a scripting language called Scheme. Scheme is a simplified derivative of Lisp, and is a small and beautiful dynamically typed, lexically scoped, functional language.

2.2.2 Using Scheme

2.2.2.1 Setting up Notepad++

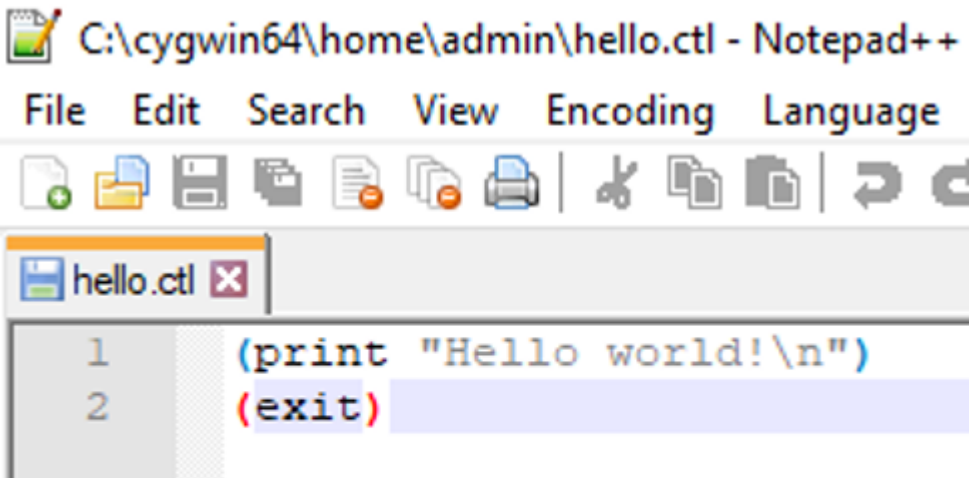
On Windows, a good text editor for Scheme is Notepad++. It is installed on the lab PCs already, but if you are using your own computer, you can get it here:
https://notepad-plus-plus.org/
See *KD6041-resources\Documentation\notepad++_tips.pdf* for a recommended setup.

2.2.2.2 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)
```

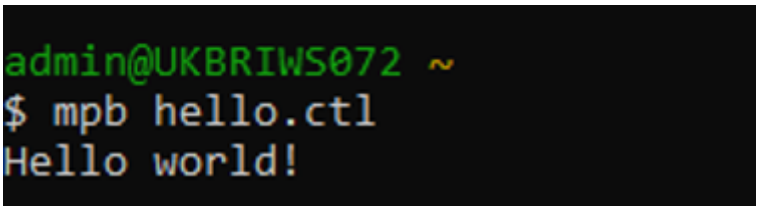
Listing 6: Also available from KD6041-resources\scheme-examples\hello.ctl



4. Now open a *Cygwin terminal* (see figure 1.8) and run the following command in it:

```
mpb hello.ctl
```

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



2.2.2.3 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/`

Bibliography

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

[2] 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

[3] 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/

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

[5] 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