One-dimensional photonic bound states in the continuum

Nabeel Ahmed, Harshit Agarwal, Kasi Reddy Sreeman Reddy, Jai Anil Israni

Open in Colab

Abstract

We develop some theory and reproduce some of the calculations and graphs presented in [1] using python code. First, we deal with an electron toy-model and look at the bound states in the continuum for a one-dimensional quantum potential well. We then exploit the one-to-one correspondence between electronic spin and polarisation states of light and develop the theory corresponding to bound states in the continuum for a one-dimensional photonic system, which consists of a one-dimensional photonic crystal conjugated with a liquid-crystalline defect layer and covered by metal film. We perform numerical calculations and plot graphs of physical quantities of interest for both these systems.

1) Introduction to Bound States in the Continuum

Waves are ubiquitous in nature, and are found in different flavours: Some waves are propogating, or 'travelling' waves. In the parlance of quantum mechanics, the states which correspond to such waves are called *scattering states.* In contrast to this, some waves tend to be spatially localised. Eigenstates of the wave equation that correspond to such waves which are confined to a particular region of space are known as *bound states.*

There are some very popular examples of bound states in both classical as well as quantum mechanics. A standing wave obtained on a plucked guitar string is spatially localised, and can thus be called a bound solution of the mechanical wave equation. The eigenstates obtained as solutions to the quantum mechanical harmonic oscillator problem are also all bound states.

Generally, scattering states form a continuum whereas bound states tend to be discrete. However, a *bound state in the continuum* (or *BIC* as we shall call it in the rest of this paper) is a unique case which defies conventional wisdom, and it occurs when the solution to a given wave equation is a variety of continuum scattering states, and there happen to be certain bound states embedded within this spectrum of continuum states.

As defined in [2], BICs are waves that remain localised even though they coexist with a continuous spectrum of radiating waves that can carry energy away.

One may wonder how these BICs are any different from the types of bound states described earlier, for instance say the bound states of the harmonic oscillator. The difference lies in the fact that the eigenstates of the (Schrödinger) wave equation for the harmonic oscillator are *all* discrete and *all* bound. Any state with any intermediate energy won't be allowed, and all the allowed energies correspond to bound states.

On the other hand, a BIC would typically look like a continuum of states, with any value of energy allowed(hence the continuum), but for very specific values of E, one happens to get states which are spatially localised. Thus, BICs are sometimes referred to as *embedded eigenstates* or *embedded trapped modes*.

This is a classically paradoxical phenomenon. For instance, a particle may have enough energy to escape an attractive potential well, but may still remain spatially confined. One can imagine throwing a tennis ball across some potential well. A BIC would mean that all values of energy are allowed, and that for (almost) all values of its energy, the tennis ball is able to pass through unimpeded. However, for certain very specific values of energy, the ball happens to be spatially confined!

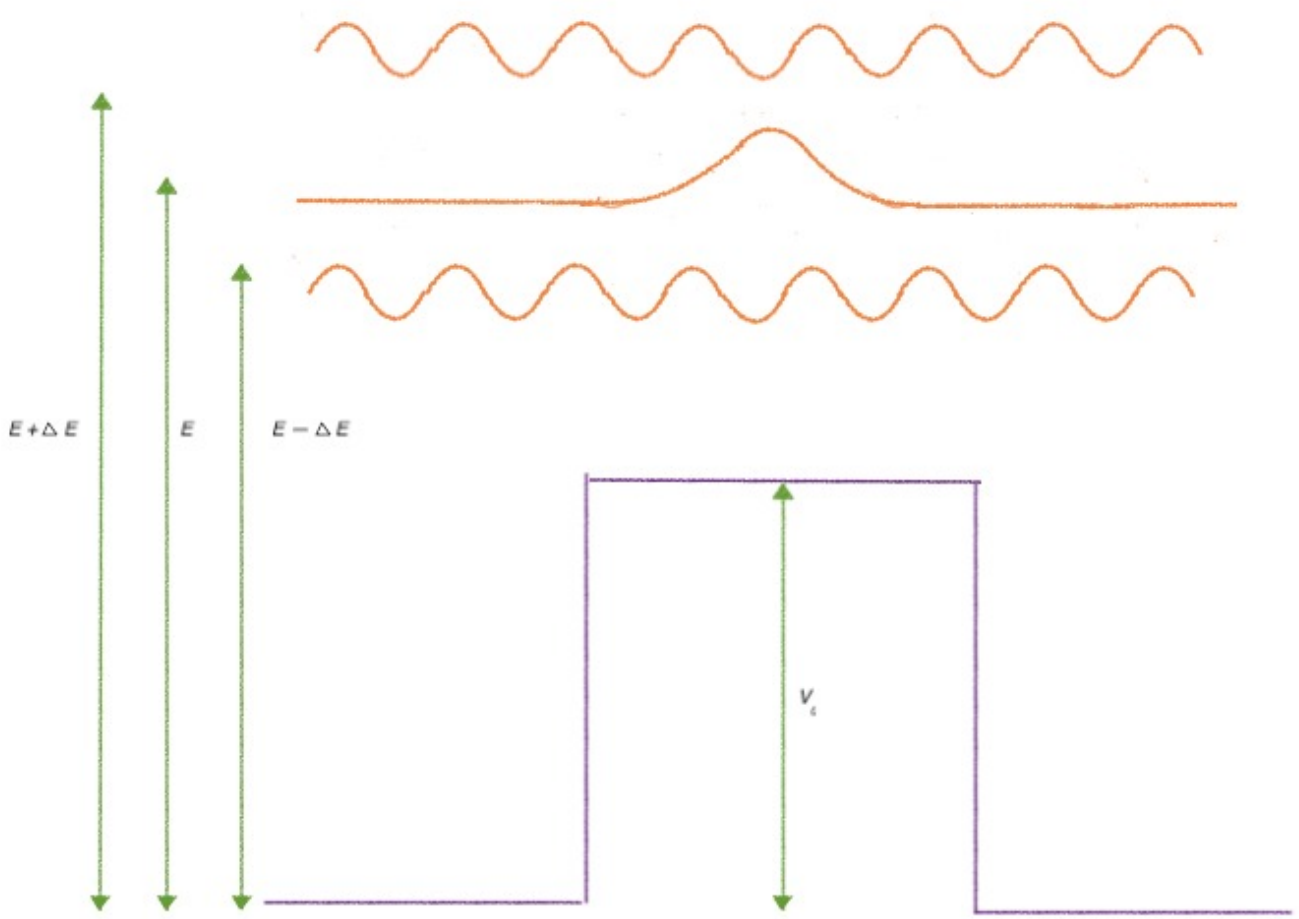


Figure 1: Schematic representation of a BIC, and how it is classically paradoxical

The unintuitiveness of the situation disappears once we replace the tennis ball with a photon or an electron, and that is precisely what we shall do for the remainder of the paper. Both electrons and photons are capable of showing BICs in a host of different situations. For this paper, we shall be looking at one-dimensional BICs. These BICs are not just a mathematical curiosity, but have been experimentally verified, especially for photons, and we shall develop the theory corresponding to one such experimental setup in this paper. However, before exploring the photonic system, we first take a look at an electron toy-model, and the corresponding BICs, for there are some useful insights that one can get from the mathematics of the problem.

2) The Electron Toy-Model

As far as the physics of this section goes, we are doing nothing special apart solving the Schrödinger wave equation for a single electron in a very specific system. This system may seem awfully abstract but the point of going through this exercise will become clear soon.

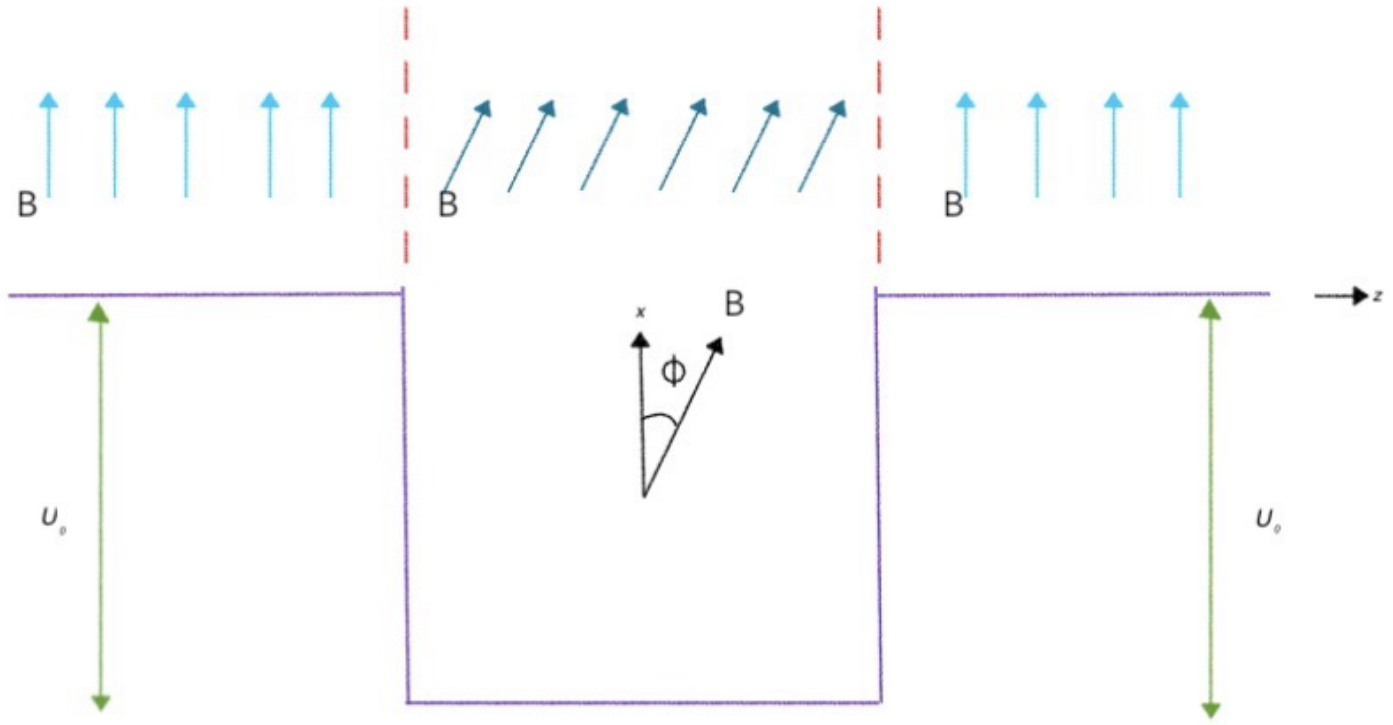


Figure 2: One-dimensional potentials and fields for the electron toy-model

The one-dimensional model, as illustrated in Figure 2, is along the z-axis. It consists of two semi-infinite regions containing a stationary magnetic field B pointing in the x-direction, and between these two regions is sandwiched a central layer in which the magnetic field is tilted by some angle ϕ. We also assume that the inner layer has the potential shifted by some value U_0.

The Schrödinger equation for an electron in magnetic field is given by:

$$\left[\frac{1}{2m}(i\hbar\nabla + e\mathbf{A})^2 + U_0(z) - \boldsymbol{\sigma}.\boldsymbol{B}(z)\right]\Psi = E\Psi$$

We state without proof, the following result: Let L be the length of the central layer described in Figure 2. If $L << \sqrt{\frac{\hbar c}{eB}}$, then we can ignore the orbital contribution in the previous equation. Since the choice of L is up to us, we can always take it to be small enough to be able to write the wave equation as

$$\left[\nabla^2 - U_0(z) + \boldsymbol{\sigma}.\boldsymbol{B}(z) + E\right]\Psi = 0$$

The Hamiltonian now takes the form:

$$\hat{H} = \begin{cases} -\frac{d^2}{dz^2} - \sigma_x B & |z| > \frac{L}{2} \\ -\frac{d^2}{dz^2} + U_0 - \sigma_x B cos(\phi) - \sigma_z B sin(\phi) & |z| < \frac{L}{2} \end{cases}$$

So for outer layers, we have the eigenstates of the hamiltonian as:

$$\Psi_{\mathbf{k}_\sigma}(z) = e^{i\mathbf{k}_\sigma \mathbf{z}|\sigma\rangle}$$

where

$$|\uparrow\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |\downarrow\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

And for the inner layer, we have the eigenstates as:

$$\Psi_{\mathbf{k}_s}(z) = e^{i\mathbf{k}_s \mathbf{z}|s\rangle}$$

where

$$|1\rangle = \begin{pmatrix} cos(\phi/2) \\ sin(\phi/2) \end{pmatrix}, |2\rangle = \begin{pmatrix} -sin(\phi/2) \\ cos(\phi/2) \end{pmatrix}$$

Here $\sigma = \uparrow, \downarrow$ and $s = 1, 2$

The expression for the incident electronic energy E is given by:

$$E = k_\sigma^2 \mp B \textit{ in the outer layers}$$
$$E = k_s^2 + U_0 \mp B \textit{ in the inner layer}$$

and these clearly form a continuum, as they are continuous functions of the variables k_σ and k_s

We choose E such that only the spin-up channel is open in the outer layers. Then for $z < -\frac{L}{2}$, we have

$$\Psi_L(z) = (e^{ik_\uparrow \mathbf{z}} + r_\uparrow e^{-ik_\uparrow \mathbf{z}})|\uparrow\rangle + r_\downarrow e^{-ik_\downarrow \mathbf{z}}|\downarrow\rangle$$

In the defect layer $|z| < \frac{L}{2}$, both channels are open, so we have:

$$\Psi_B(z) = \sum_{s=1,2} (a_s e^{ik_\uparrow \mathbf{z}} + b_s e^{-ik_\uparrow \mathbf{z}})|s\rangle$$

And for $z > \frac{L}{2}$, we have

$$\Psi_R(z) = t_\uparrow e^{ik_\uparrow \mathbf{z}}|\uparrow\rangle + t_\downarrow e^{ik_\downarrow \mathbf{z}}|\downarrow\rangle$$

Here, $r_\uparrow, r_\downarrow$ and $t_\uparrow, t_\downarrow$ are reflection and transmission coefficients of both spin states.

The electron with spin σ is incident with wave vector $\boldsymbol{k_\sigma} = (k_{x\sigma}, k_{z\sigma})$ and is reflected with reflection amplitude r_σ

Also, transverse component of momentum is conserved i.e. $k_{x\sigma} = k_{xs} = k_x$

We apply the boundary conditions, i.e the continuity of the wavefunction and its derivative at $+\frac{L}{2}$ and $-\frac{L}{2}$, which gives us the following eight equations:

$$1 + r_\uparrow = (a_1 + b_1)cos(\phi/2) - (a_2 + b_2)sin(\phi/2)$$
$$k_{z\uparrow}(1 - r_\uparrow) = k_{z1}(a_1 - b_1)cos(\phi/2) - k_{z2}(a_2 - b_2)sin(\phi/2)$$

$$r_\downarrow = (a_1 + b_1)sin(\phi/2) + (a_2 + b_2)cos(\phi/2)$$
$$-k_{z\downarrow}r_\downarrow = k_{z1}(a_1 - b_1)sin(\phi/2) + k_{z2}(a_2 - b_2)cos(\phi/2)$$
$$t_\uparrow e^{ik_{z\uparrow}L} = (a_1 e^{ik_{z1}L} + b_1 e^{-ik_{z1}L})cos(\phi/2) - (a_2 e^{ik_{z2}L} + b_2 e^{-ik_{z2}L})sin(\phi/2)$$
$$k_{z\uparrow}t_\uparrow e^{ik_{z\uparrow}L} = k_{z1}(a_1 e^{ik_{z1}L} + b_1 e^{-ik_{z1}L})cos(\phi/2) - k_{z2}(a_2 e^{ik_{z2}L} + b_2 e^{-ik_{z2}L})sin(\phi/2)$$
$$t_\downarrow e^{ik_{z\downarrow}L} = (a_1 e^{ik_{z1}L} + b_1 e^{-ik_{z1}L})sin(\phi/2) + (a_2 e^{ik_{z2}L} + b_2 e^{-ik_{z2}L})cos(\phi/2)$$
$$k_{z\downarrow}t_\downarrow e^{ik_{z\downarrow}L} = k_{z1}(a_1 e^{ik_{z1}L} + b_1 e^{-ik_{z1}L})sin(\phi/2) + k_{z2}(a_2 e^{ik_{z2}L} + b_2 e^{-ik_{z2}L})cos(\phi/2).$$

Combining these equations with the analytic equations for E, we find that for a given angle of incidence θ and a given choice of E and L, we can find a unique eigenfunction which satisfies Schrödinger's equation, and thus we find a continuum of solutions.

We now proceed to visually look at this continuum. We first fix a particular θ and vary E and L, and plot $r_\uparrow$ as a function of (E, L). We then fix E and plot $r_\uparrow$ as a function of (θ, L).

```
#CODE BLOCK 1: Generating data by varying E and L
#All equations and calculation parameters taken from [1]

import mpmath as mp
import numpy as np
import matplotlib.pyplot as plt
# initializing reflectence and transmittance as zero 2d array
ruparray1 = np.zeros((101, 81))
tuparray1 = np.zeros((101, 81))
rdownarray1 = np.zeros((101, 81))
tdownarray1 = np.zeros((101, 81))
# variables
Larray=np.arange(0,101)/20
Earray=np.arange(0,81)/2
# constants
phi=mp.pi/4
theta=mp.pi/3
B=10
U=-20

for lcounter in range(0,101):
  for ecounter in range(0, 81):
    L=Larray[lcounter]
    E=Earray[ecounter]
    def step_1(kx, kzup, kzdown, kz1, kz2):
        return [kx - mp.sqrt(mp.fabs(E+B) )*mp.sin(theta),
                kzup - mp.sqrt(mp.fabs(E+B))*mp.cos(theta),
                kzdown - mp.sqrt(mp.fabs( E  -  B    -  (E*(mp.sin(theta)**2))  - (
                kz1 - mp.sqrt(mp.fabs(  E  - U  + B    -  (E*(mp.sin(theta)**2))  -
                kz2 - mp.sqrt(mp.fabs( E  - U  - B    -  (E*(mp.sin(theta)**2))  -
                ]

    def step1(phi=mp.pi/4, theta=mp.pi/3, L=L, E=E, B=B, U=U):
      sol_step1 = mp.findroot(step_1, [10.5+0.5j, 10.5+0.5j, 10.5+0.5j, 10.5+0.5j,1
      return sol_step1
```

```
    k_set=step1()
    kx=k_set[0]
    kzup=k_set[1]
    kzdown=k_set[2]
    kz1=k_set[3]
    kz2=k_set[4]

    def step_2(rup, rdown, tup, tdown, a1, a2, b1, b2):
      return [1+rup -( (a1 + b1)*mp.cos(phi/2) - (a2 + b2)*mp.sin(phi/2)),

              kzup*(1-rup) -( kz1*(a1 - b1)*mp.cos(phi/2)-kz2*(a2-b2)*mp.sin(phi/2)
              rdown-((a1 + b1)*mp.sin(phi/2) + (a2 + b2)*mp.cos(phi/2)),
              - kzdown*rdown - (kz1*(a1-b1)*mp.sin(phi/2) + kz2*(a2-b2)*mp.cos(phi/
              tup*mp.exp(1j*kzup*L)-((a1*mp.exp(1j*kz1*L)+b1*mp.exp(-1j*kz1*L))*mp.
              kzup*tup*mp.exp(1j*kzup*L)-(kz1*(a1*mp.exp(1j*kz1*L)-b1*mp.exp(-1j*kz
              tdown*mp.exp(1j*kzdown*L)-((a1*mp.exp(1j*kz1*L)+b1*mp.exp(-1j*kz1*L))
              kzdown*tup*mp.exp(1j*kzdown*L)-(kz1*(a1*mp.exp(1j*kz1*L)-b1*mp.exp(-1

    def step2(phi=mp.pi/4, theta=mp.pi/3, L=Larray[lcounter], E=Earray[ecounter], E
      sol_step2 = mp.findroot(step_2, [10.5+0.5j, 10.5+0.5j, 10.5+0.5j, 10.5+0.5j,1
      return sol_step2

    f = step2()

    ruparray1[lcounter][ecounter]=mp.fabs(f[0])
    tuparray1[lcounter][ecounter]=mp.fabs(f[2])
    rdownarray1[lcounter][ecounter]=mp.fabs(f[1])
    tdownarray1[lcounter][ecounter]=mp.fabs(f[3])

print(ruparray1)
```

```
[[3.20081785e-23 7.17190326e-24 1.37590109e-21 ... 7.09042818e-25
  5.42675315e-22 6.33752709e-24]
 [1.55690908e-01 1.51987238e-01 1.48534948e-01 ... 7.20350265e-02
  7.17353593e-02 7.14431136e-02]
 [2.98811026e-01 2.91935504e-01 2.85506260e-01 ... 1.45796460e-01
  1.45410398e-01 1.45041506e-01]
 ...
 [1.47853672e+00 1.66336984e+00 2.45685464e+00 ... 8.18788086e-01
  8.13286052e-01 8.00700683e-01]
 [1.36442385e+00 1.34476737e+00 1.47880485e+00 ... 6.93828269e-01
  6.87223023e-01 6.74575262e-01]
 [1.20119130e+00 1.09665926e+00 9.71533296e-01 ... 5.34161381e-01
  5.30803060e-01 5.22926188e-01]]
```

```
#CODE BLOCK 2: Generating data by varying θ and L
#All equations and calculation parameters taken from [1]

import mpmath as mp
ruparray2 = np.zeros((101, 60))
tuparray2 = np.zeros((101, 60))
rdownarray2 = np.zeros((101, 60))
tdownarray2 = np.zeros((101, 60))
```

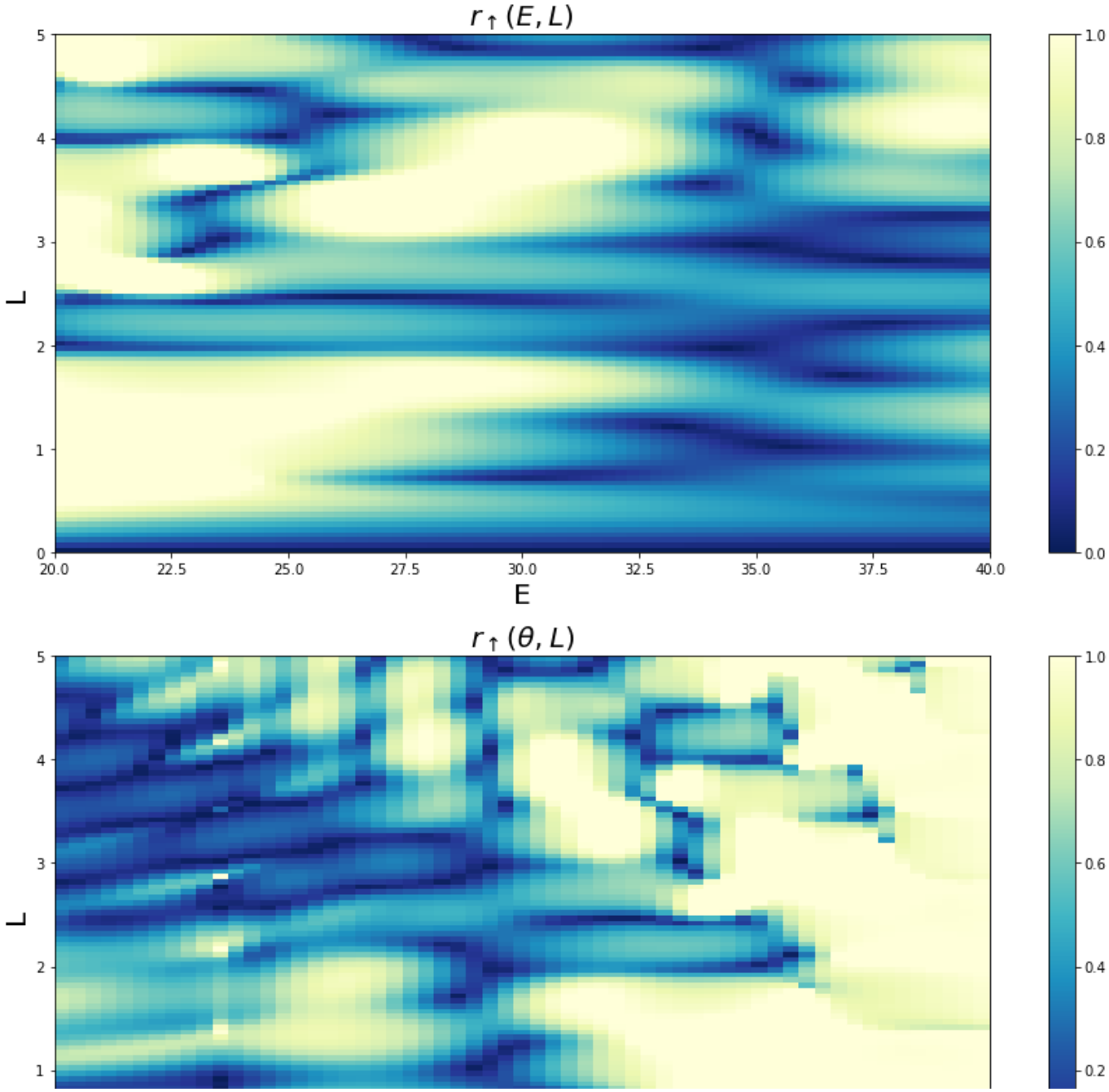
```
tdownarray2 = np.zeros((101, 60))
# variables
Larray=np.arange(0,101)/20
thetaarray=(0.2+(np.arange(0,60)/200))*mp.pi
# constants
phi=mp.pi/4
E=30
B=10
U=-20
for lcounter in range(0,101):
  for thetacounter in range(0, 60):
    L=Larray[lcounter]
    theta=thetaarray[thetacounter]
    def step_1(kx, kzup, kzdown, kz1, kz2):
        return [kx - mp.sqrt(mp.fabs(E+B) )*mp.sin(theta),
                kzup - mp.sqrt(mp.fabs(E+B))*mp.cos(theta),
                kzdown - mp.sqrt(mp.fabs( E  -  B    -  (E*(mp.sin(theta)**2))  - (
                kz1 - mp.sqrt(mp.fabs(  E  - U  + B    -  (E*(mp.sin(theta)**2))  -
                kz2 - mp.sqrt(mp.fabs( E  - U  - B    -  (E*(mp.sin(theta)**2))  -
                ]
    def step1(phi=phi, theta=thetaarray[thetacounter], L=Larray[lcounter], E=E, B=B
      sol_step1 = mp.findroot(step_1, [10.5+0.5j, 10.5+0.5j, 10.5+0.5j, 10.5+0.5j,1
      return sol_step1
    k_set=step1()
    kx=k_set[0]
    kzup=k_set[1]
    kzdown=k_set[2]
    kz1=k_set[3]
    kz2=k_set[4]
    def step_2(rup, rdown, tup, tdown, a1, a2, b1, b2):
      return [1+rup -( (a1 + b1)*mp.cos(phi/2) - (a2 + b2)*mp.sin(phi/2)),

              kzup*(1-rup) -( kz1*(a1 - b1)*mp.cos(phi/2)-kz2*(a2-b2)*mp.sin(phi/2)
              rdown-((a1 + b1)*mp.sin(phi/2) + (a2 + b2)*mp.cos(phi/2)),
              - kzdown*rdown - (kz1*(a1-b1)*mp.sin(phi/2) + kz2*(a2-b2)*mp.cos(phi/
              tup*mp.exp(1j*kzup*L)-((a1*mp.exp(1j*kz1*L)+b1*mp.exp(-1j*kz1*L))*mp.
              kzup*tup*mp.exp(1j*kzup*L)-(kz1*(a1*mp.exp(1j*kz1*L)-b1*mp.exp(-1j*kz
              tdown*mp.exp(1j*kzdown*L)-((a1*mp.exp(1j*kz1*L)+b1*mp.exp(-1j*kz1*L))
              kzdown*tup*mp.exp(1j*kzdown*L)-(kz1*(a1*mp.exp(1j*kz1*L)-b1*mp.exp(-1

    def step2(phi=phi, theta=thetaarray[thetacounter], L=L, E=E, B=B, U=U): #k_set
      sol_step2 = mp.findroot(step_2, [1+1j, 1+1j, 1+1j, 1+1j,1+1j, 1+1j, 1+1j, 1+1
      return sol_step2
    f = step2()
    ruparray2[lcounter][thetacounter]=mp.fabs(f[0])
    tuparray2[lcounter][thetacounter]=mp.fabs(f[2])
    rdownarray2[lcounter][thetacounter]=mp.fabs(f[1])
    tdownarray2[lcounter][thetacounter]=mp.fabs(f[3])

print(ruparray2)
```

```
[[7.47499379e-23 5.00885619e-23 2.42798412e-23 ... 3.04910146e-23
```

```
 1.02281536e-23 3.18878501e-22]
[5.44140687e-02 5.53383008e-02 5.63623875e-02 ... 6.55343777e-01
 7.99246060e-01 9.43789966e-01]
[1.17591845e-01 1.19565164e-01 1.21713713e-01 ... 8.91805687e-01
 9.58550590e-01 9.98645160e-01]
...
[3.60220983e-01 5.26976882e-01 6.07726740e-01 ... 1.17731078e+00
 1.49028815e+00 3.11471386e+00]
[3.18885834e-01 5.16733298e-01 6.38122131e-01 ... 1.01936701e+00
 1.01938105e+00 1.01186828e+00]
[3.20857143e-01 4.89563091e-01 6.42089054e-01 ... 9.97729570e-01
 9.98412343e-01 9.98772643e-01]]
```

```
#CODE BLOCK 3: REFLECTANCE SPECTRA
#All equations and calculation parameters taken from [1]

L=np.arange(0,101)/20
E=20+np.arange(0,81)/4
thetabypi=0.2+(np.arange(0,60)/200)

fig, axs = plt.subplots(2, 1,figsize=(15,15))
## for first plot:
ax = axs[0]
ax.set_xlabel('E',fontsize= 20)
ax.set_ylabel('L',fontsize= 20)
ax.grid(True)
c = ax.pcolor(E, L, ruparray1, cmap='YlGnBu_r', vmin=0, vmax=1)
ax.set_title(r"$r_{\uparrow}(E,L)$",fontsize= 20)
fig.colorbar(c, ax=ax)
## for second plot:
ax = axs[1]
ax.set_xlabel(r"${\theta / \pi}$",fontsize= 20)
ax.set_ylabel('L',fontsize= 20)
ax.grid(True)
c = ax.pcolor(thetabypi, L, ruparray2, cmap='YlGnBu_r', vmin=0, vmax=1)
ax.set_title(r"$r_{\uparrow}(\theta,L)$",fontsize= 20)
fig.colorbar(c, ax=ax)
```

```
<ipython-input-55-3fcacdde3c47>:14: MatplotlibDeprecationWarning: shading='flat'
  c = ax.pcolor(E, L, ruparray1, cmap='YlGnBu_r', vmin=0, vmax=1)
<ipython-input-55-3fcacdde3c47>:22: MatplotlibDeprecationWarning: shading='flat'
  c = ax.pcolor(thetabypi, L, ruparray2, cmap='YlGnBu_r', vmin=0, vmax=1)
<matplotlib.colorbar.Colorbar at 0x1e034c14f70>
```

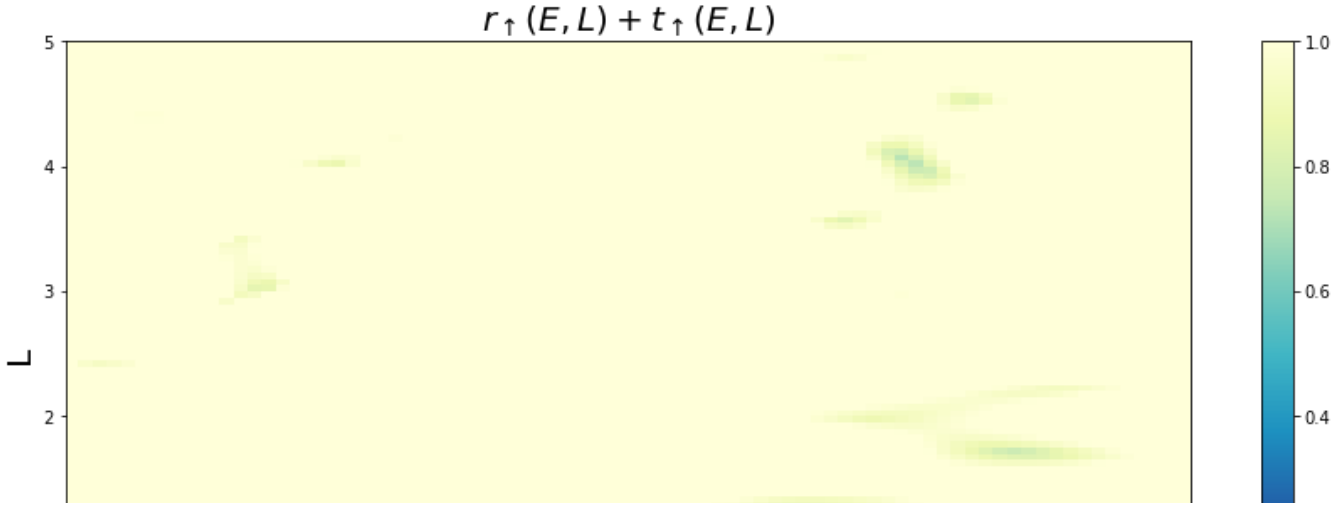


Naturally, a continuum is the first step towards finding a BIC. Now that we have the continuum, for the same model, we fix θ and plot $r_\uparrow + t_\uparrow$ as a function of (E, L). We then fix E and plot $r_\uparrow + t_\uparrow$ as a function of (θ, L). Naively, we may expect the value of $r_\uparrow + t_\uparrow$ to always be unity. Let us see if this is what we get.

```
#CODE BLOCK 4: REFLECTANCE + TRANSMITTANCE SPECTRA
#All equations and calculation parameters taken from [1]

fig, axs = plt.subplots(2, 1,figsize=(15,15))
## for first plot:
```

```
ax = axs[0]
ax.set_xlabel('E',fontsize= 20)
ax.set_ylabel('L',fontsize= 20)
ax.grid(True)
c = ax.pcolor(E, L, ruparray1+tuparray1, cmap='YlGnBu_r', vmin=0, vmax=1)
ax.set_title(r"$r_{\uparrow}(E,L)+t_{\uparrow}(E,L)$",fontsize= 20)
fig.colorbar(c, ax=ax)
## for second plot:
ax = axs[1]
ax.set_xlabel(r"$\theta / \pi}$",fontsize= 20)
ax.set_ylabel('L',fontsize= 20)
ax.grid(True)
c = ax.pcolor(thetabypi, L, ruparray2+tuparray2, cmap='YlGnBu_r', vmin=0, vmax=1)
ax.set_title(r"$r_{\uparrow}(\theta,L)+t_{\uparrow}(\theta,L)$",fontsize= 20)
fig.colorbar(c, ax=ax)
```

```
<ipython-input-54-057d7253450f>:10: MatplotlibDeprecationWarning: shading='flat'
  c = ax.pcolor(E, L, ruparray1+tuparray1, cmap='YlGnBu_r', vmin=0, vmax=1)
<ipython-input-54-057d7253450f>:18: MatplotlibDeprecationWarning: shading='flat'
  c = ax.pcolor(thetabypi, L, ruparray2+tuparray2, cmap='YlGnBu_r', vmin=0, vmax=
<matplotlib.colorbar.Colorbar at 0x1e0246c2bb0>
```



We see that the value of $r_\uparrow + t_\uparrow$ is not always unity, and there are certain regions where it is less than one.

To understand this, let us look at Figure 3. From the left, a wave of unity modulus is incident on the central layer. Some of it is reflected, and some of it is transmitted. If these reflection and transmisison coefficients don't add up to unity, then there is something peculiar which is happening here. Naturally, one may wonder where the wave goes, if it is not fully reflected or transmitted.

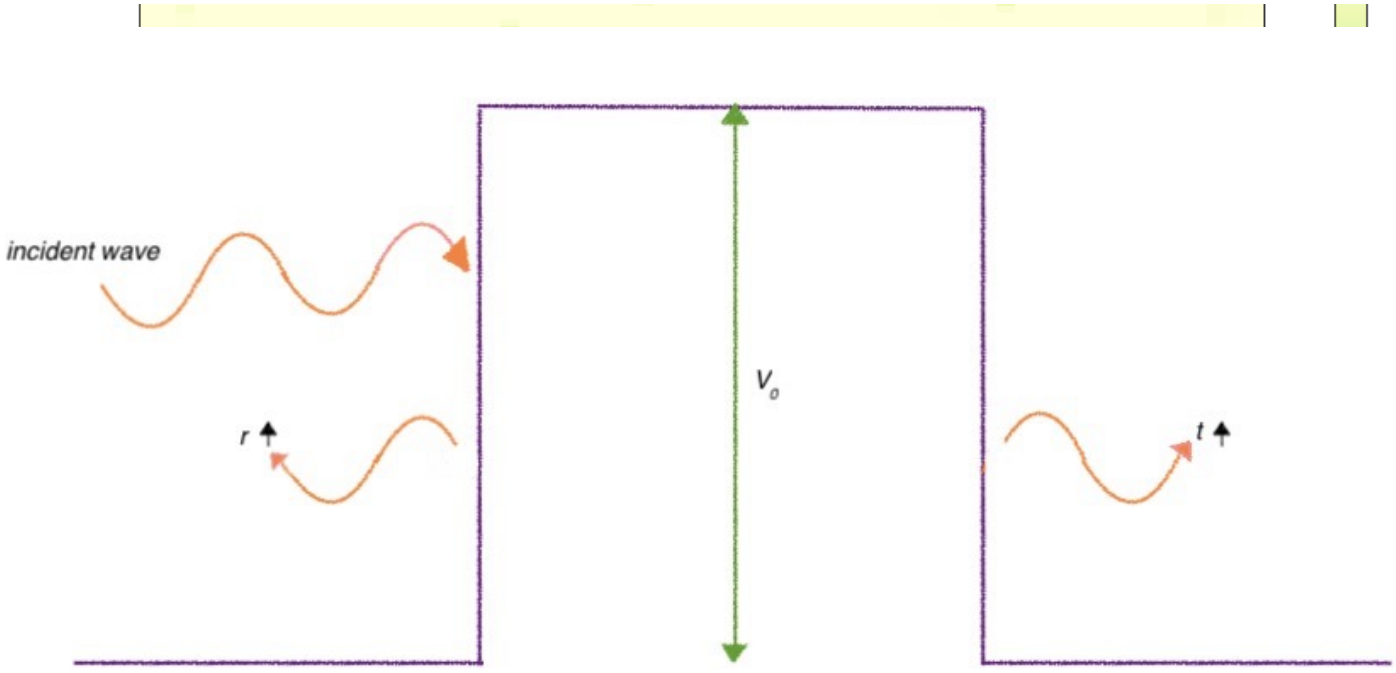


Figure 3: Schematic representation of reflection and transmission coefficients

The answer is, the spin-up wave *mixes* with the spin-down wave in the central layer. This feature is a very classic footprint of a BIC. For certain reasons (that we shall discuss in the end of the paper), a BIC is only possible when such a 'mixing' of channels occurs.

Now that we know that there is a certain mixing of the spin-up and spin-down continua happening, we may feel more confident in looking for a BIC.

In fact, when we plug in the very specific values of $(E, \theta, L) = (30, \pi/3, 2.5)$ and $(E, \theta, L) = (30, 0.036\pi, 2)$, we *do get* a symmetric and antisymmetric bound state respectively.

It may seem that these specific tuples which give us BICs are values that we have randomly stumpled upon. This is not true. There is a method for calculating exactly which values of (θ, L) will give us a BIC, but that is an unnecessary detour so we shall skip it.

What matters is that within the continuum, there do exist these states which are spatially confined. The representative wavefunctions for these BICs are plotted in the next code block:

```
#CODE BLOCK 5A: BIC WAVEFUNCTIONS
#All equations and calculation parameters taken from [1]

import mpmath as mp
import numpy as np
import matplotlib.pyplot as plt

phi=mp.pi/4
theta=mp.pi/3
E=30
B=10
U=-20
Lv=2.5

def step_1(kx, kzup, kzdown, kz1, kz2):
    return [kx - mp.sqrt(mp.fabs(E+B) )*mp.sin(theta),
            kzup - mp.sqrt(mp.fabs(E+B))*mp.cos(theta),
            kzdown - mp.sqrt(mp.fabs( E  -  B    -  (E*(mp.sin(theta)**2))  - (B*(m
            kz1 - mp.sqrt(mp.fabs(  E  - U  + B    -  (E*(mp.sin(theta)**2))  - (B*
            kz2 - mp.sqrt(mp.fabs( E  - U  - B    -  (E*(mp.sin(theta)**2))  - (B*(
            ]

def step1(phi=phi, theta=theta, E=E, B=B, U=U):
    sol_step1 = mp.findroot(step_1, [10.5+0.5j, 10.5+0.5j, 10.5+2j, 10.5+0.5j,10.5+
    return sol_step1


k_set=step1()
kx=k_set[0]
kzup=k_set[1]
kzdown=k_set[2]
kz1=k_set[3]
```

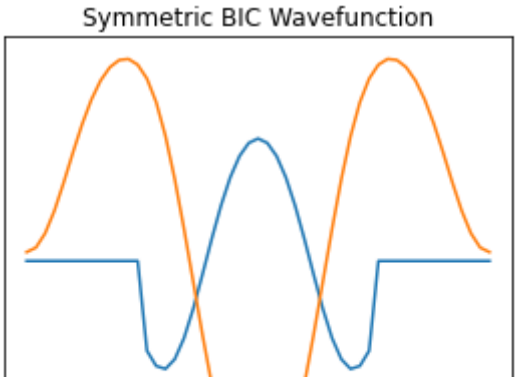
```
kz1=k_set[3]
kz2=k_set[4]

#phi = 2*mp.atan((mp.fabs(kzdown) - kz2*mp.tan(kz2*L/2))/(kz1*mp.tan(kz1*L/2) - mp.
print(k_set)

Lv=int(Lv*10)

symvaluetop = np.zeros(2*Lv + 1)
symvaluebot = np.zeros(2*Lv + 1)
asymvaluetop = np.zeros(2*Lv + 1)
asymvaluebot = np.zeros(2*Lv + 1)
```

```
[(5.47722557505166 + 0.0j)]
[(3.16227766016838 + 0.0j)]
[(3.16227766016838 + 0.0j)]
[(5.47722557505166 + 0.0j)]
[(3.16227766016838 + 0.0j)]
```

```
#CODE BLOCK 5B: BIC WAVEFUNCTIONS
#All equations and calculation parameters taken from [1]

for lcounter in range(-Lv,Lv+1):
    L=lcounter/20

    def step_2(rup, rdown, tup, tdown, a1, a2, b1, b2):
      return [1+rup -( (a1 + b1)*mp.cos(phi/2) - (a2 + b2)*mp.sin(phi/2)),
              kzup*(1-rup) -( kz1*(a1 - b1)*mp.cos(phi/2)-kz2*(a2-b2)*mp.sin(phi/2)
              rdown-((a1 + b1)*mp.sin(phi/2) + (a2 + b2)*mp.cos(phi/2)),
              - kzdown*rdown - (kz1*(a1-b1)*mp.sin(phi/2) + kz2*(a2-b2)*mp.cos(phi/
              tup*mp.exp(1j*kzup*L)-((a1*mp.exp(1j*kz1*L)+b1*mp.exp(-1j*kz1*L))*mp.
              kzup*tup*mp.exp(1j*kzup*L)-(kz1*(a1*mp.exp(1j*kz1*L)-b1*mp.exp(-1j*kz
              tdown*mp.exp(1j*kzdown*L)-((a1*mp.exp(1j*kz1*L)+b1*mp.exp(-1j*kz1*L))
              kzdown*tup*mp.exp(1j*kzdown*L)-(kz1*(a1*mp.exp(1j*kz1*L)-b1*mp.exp(-1

    def step2(phi=phi, theta=mp.pi*0.36, L=Lv/10, E=E, B=10, U=-20): #k_set enterec
      sol_step2 = mp.findroot(step_2, [1+1j, 1+1j, 1+1j, 1+1j,1+1j, 1+1j, 1+1j, 1+1
      return sol_step2

    f = step2()
    if (lcounter<-12 or lcounter>12):
      symvaluetop[lcounter+Lv] = -0.2
    else:
      symvaluetop[lcounter+Lv]=  + mp.fabs(f[4])*mp.cos(mp.fabs(phi/2))*mp.cos(L*mp

    if (lcounter<-9 or lcounter>9):
      asymvaluetop[lcounter+Lv] = 0
    else:
      asymvaluetop[lcounter+Lv]= - mp.fabs(f[4])*mp.cos(mp.fabs(phi/2))*mp.sin(L*mp

    if (lcounter<-16 or lcounter>16):
      asymvaluebot[lcounter+Lv] = 0
```

```
    else:
      asymvaluebot[lcounter+Lv]= mp.fabs(f[4])*mp.sin(mp.fabs(kz1)*L)*mp.sin(mp.fab

    symvaluebot[lcounter+Lv]=  - mp.fabs(f[4])*mp.cos(mp.fabs(kz1)*L)*mp.sin(mp.fab

ypoints = symvaluetop
xpoints = []
for lcounter in range(-Lv,Lv + 1):
  xpoints.append(lcounter/20)
fig, axs = plt.subplots(2, 1,figsize=(10,10))
ax=axs[0]
plt.subplot(2,2,1)
plt.plot(xpoints, ypoints, label="spin up")
plt.subplot(2,2,1)
plt.plot(xpoints, symvaluebot, label="spin down")
plt.yticks([])
plt.legend()
plt.title(r"Symmetric BIC Wavefunction")
ax=axs[1]
ax.set_title(r"Asymmetric BIC Wavefunction")
plt.subplot(2,2,3)
plt.plot(xpoints, asymvaluetop, label="spin up")
plt.subplot(2,2,3)
plt.plot(xpoints, asymvaluebot, label="spin down")
plt.yticks([])
plt.title(r"Antisymmetric BIC Wavefunction")
plt.legend()
plt.show()
```

```
<ipython-input-38-6669ade166ef>:47: MatplotlibDeprecationWarning: Adding an axes
  plt.subplot(2,2,1)
<ipython-input-38-6669ade166ef>:56: MatplotlibDeprecationWarning: Adding an axes
  plt.subplot(2,2,3)
```

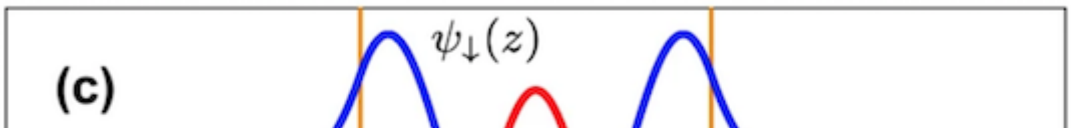


The wavefunctions for the Symmetric and Antisymmetric BICs look as follows:

$$\psi_{BIC,sym}(z) = \begin{cases} a_1 cos(k_{z_1} z)|1\rangle + b_2 cos(k_{z_2} z)|2\rangle & |z| < \frac{L}{2} \\ ce^{(-k_{z\downarrow})|z|} & |z| > \frac{L}{2} \end{cases}$$

$$\psi_{BIC,asym}(z) = \begin{cases} a_1 sin(k_{z_1} z)|1\rangle + b_2 sin(k_{z_2} z)|2\rangle & |z| < \frac{L}{2} \\ sign(z)ce^{(-k_{z\downarrow})|z|} & |z| > \frac{L}{2} \end{cases}$$

There is one interesting observation to be made here: even though the solution wavefunctions were complex, the bound state wavefunctions happen to be purely real. This fact is going to be particularly useful for the photonic case, which is discussed in the next section.



3) The Theory of Photonic BICs

One may wonder why we even considered the electron toy-model in the first place. We could have directly written the equations for the photon and proceeded from there. However, if we were to do this, by a clever change of variables, we realise that the photonic (EM wave) equation is exactly isomeric to the Schrödinger equation solved in the previous part, and there is a one-to-one correspondence with the following physical quantities:

Table 1: Quantum/Optical correspondence

Quantum mechanics	Optics
Electron	Photon
Electron wave function ψ	Electric field **E**
$\frac{\partial \psi}{\partial z}$	Magnetic field **H**
Spin	Polarization
Energy	Frequency
Spin-down electron $\lvert \downarrow \rangle$	Transverse electric wave
Spin-up electron $\lvert \uparrow \rangle$	Transverse magnetic wave
External magnetic field	Anisotropy axis

One may be familiar with a scenario where a similar one-to-one correspondence works: the *LC Circuit* and the *Spring Mass Oscillator*. Our present electron-to-photon conversion is precisely the same in letter and spirit. A rigorous derivation of this follows subsequently.

Before that, a word of caution: Even though we have used the word photon many times already, we shall in fact be treating light electromagnetically throughout this paper. The waves that we are concerned with are the Maxwellian Electric and Magnetic Fields, and *not* the photonic wavefunction. Thus, even though the word photon is a misnomer, we shall stick to this nomenclature throughout the remainder of the paper and the meaning will be clear from the context.

First, we consider the optical analogue of the tilted B-field setup as seen in Section 2. This is acheived by sandwiching an anisotropic defect layer (ADL) between two semi-infinite photonic crystal (PhC) arms, as seen in Figures 5 and 6:

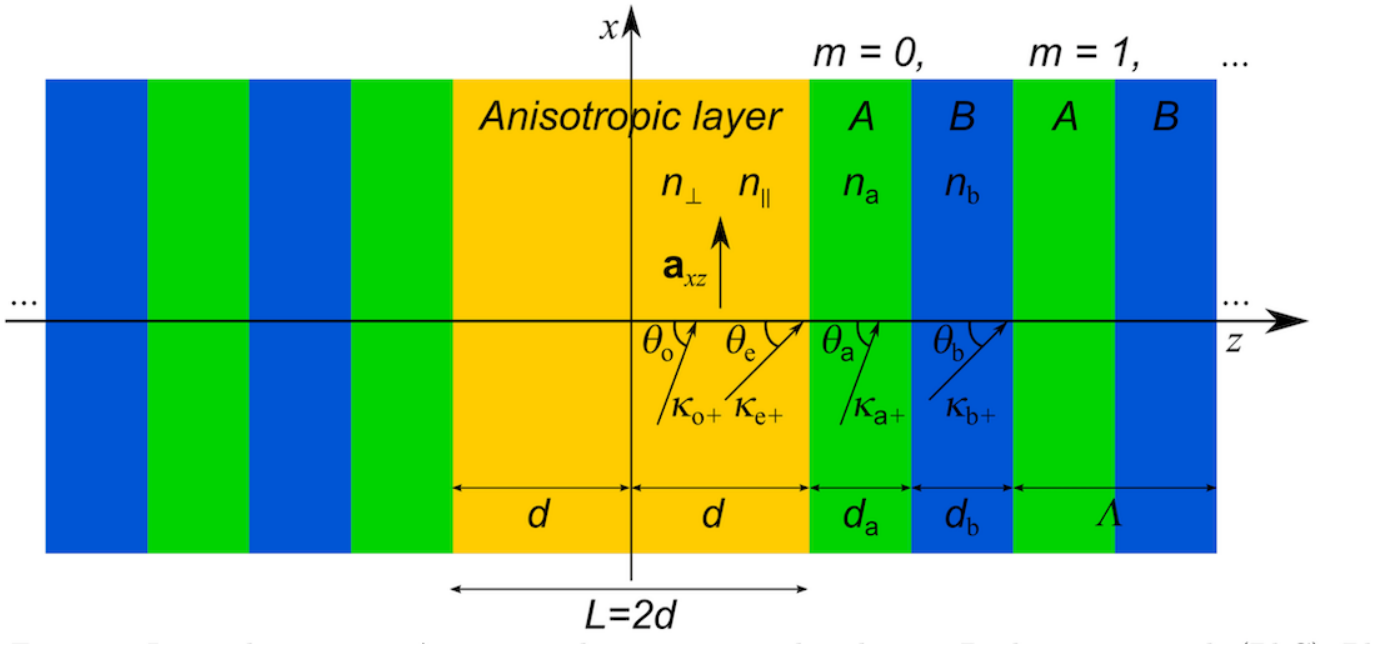


Figure 5 (taken from Supplementary Reading of [1]): One-dimensional optical setup for the photon which is equivalent to the toy-model for the electron

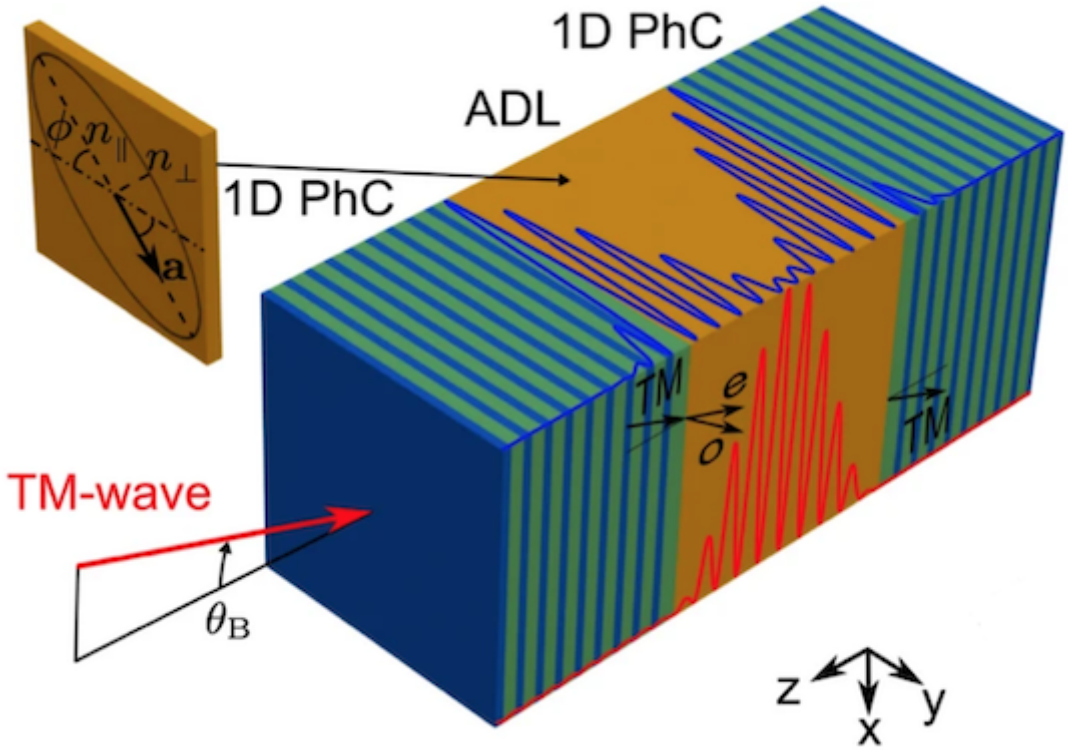


Figure 6 (taken from [1]): Physical setup corresponding to Figure 5

The unit cell of the PhC consists of alternating layers A and B, having refractive indices and thicknesses $n_a = \sqrt{\varepsilon_a}, d_a$ and $n_b = \sqrt{\varepsilon_b}, d_b$ respectively. The lattice period is $\Lambda = d_a + d_b$

The ADL, as the name suggests is anisotropic, i.e the refractive index has one value along one particular direction(given by what is known as the optical axis of the material) and a different value perpendicular to this axis.

Let $n_\perp = \sqrt{\varepsilon_0}$ and $n_{||} = \sqrt{\varepsilon_0 + \varepsilon'}$

We define the dielectric permittivity tensor $\hat{\epsilon}$ as

$$\epsilon_{ij} = \varepsilon_0 \delta_{ij} + \varepsilon' a_i a_j$$

With these notions, we can write the modified Maxwell's Equations as

$$\begin{pmatrix} 0 & \nabla\times \\ \nabla\times & 0 \end{pmatrix} \begin{pmatrix} \mathbf{E} \\ \mathbf{H} \end{pmatrix} = \frac{1}{c}\frac{\partial}{\partial t}\begin{pmatrix} \hat{\epsilon}\mathbf{E} \\ \mathbf{H} \end{pmatrix} = -ik_0 \begin{pmatrix} \hat{\epsilon}\mathbf{E} \\ \mathbf{H} \end{pmatrix}$$

Maxwell's equations decouple into two mutually orthogonal sets of equations: the TE-wave (transverse electric) corresponding to E_y, H_x, H_z and the TM-wave (transverse magnetic) corresponding to H_y, E_x, E_z.

In the anisotropic layer, two orthogonal types of solutions are determined by the optical axis $\mathbf{a}$. There is an extraordinary wave (e-wave) for which $\mathbf{E}_e . \mathbf{a} \neq 0$ and an ordinary wave (o-wave) for which $\mathbf{E}_o . \mathbf{a} = 0$.

Consider plane light waves of the form $e^{i(\mathbf{k}.\mathbf{r}-\omega t)}$, where $\mathbf{k}$ is the wave-vector, ω is the frequency, $\mathbf{r}$ is the radius vector, t is the time and $k_0 = \frac{\omega}{c}$. When a plane wave propogates in a layered medium, the projection of the wave parallel to the interfaces k_x has a common constant value in all layers, and so we omit the factor of $e^{i(k_x x-\omega t)}$ that is common to all field vectors.

This gives us the wave equation in the Helmholtz form, which is isostructural to the Schrödinger equation solved in Section 2:

Since we have already solved the Schrödinger equation for the electron, we can be assured that the algorithm to be followed in the succeeding photonic section will be the same, and the solutions will have identical properties. Most notably, for the photon, we will obtain a spectrum as a function of (ω, k_x). Out of this spectrum, we will have certain tuples which will give us BICs.

Now that we have already seen the solutions to the electron toy-model, one may ask whether it is possible to predict the nature of the tuples for the photonic BICs.

First, we can be assured of the fact that the BIC electric and magnetic fields(which correspond to the BIC wavefunction and its derivative in the electron model) will be real. This was shown in Section 2. Thus, if we get a solution in which either one of these fields becomes complex, then we can confidently reject that solution and claim that it will *certainly not* correspond to a BIC.

Secondly, the energy of the electron corresponding to the BIC tuples was also always real, and this implies that the frequency of the photonic BIC *can not be complex*.

Thus for the upcoming section, we have a neat idea: we may get a whole continuum of complex frequencies which act as solutions, but the ones which correspond to the BICs will be purely real. Thus, if we start from the whole complex frequency spectrum and start narrowing in on subsets for which the frequency is purely real, we can be assured that the BICs will lie somewhere in these subsets.

This narrowing down can be done effectively with the help of *dispersion relations*. We can simply reject the complex frequencies and try looking for solutions in the subsets of the dispersion spectrum which correspond to real frequencies.

In the next section, we implement this procedure and try to hunt for BICs by solving the photonic equations for the exact setup.

4) The Solutions of Photonic BICs

4.1) Modified setup

First, for fabricational simplicity, we replace the setup described in Figure 5 with the one described in Figure 7:

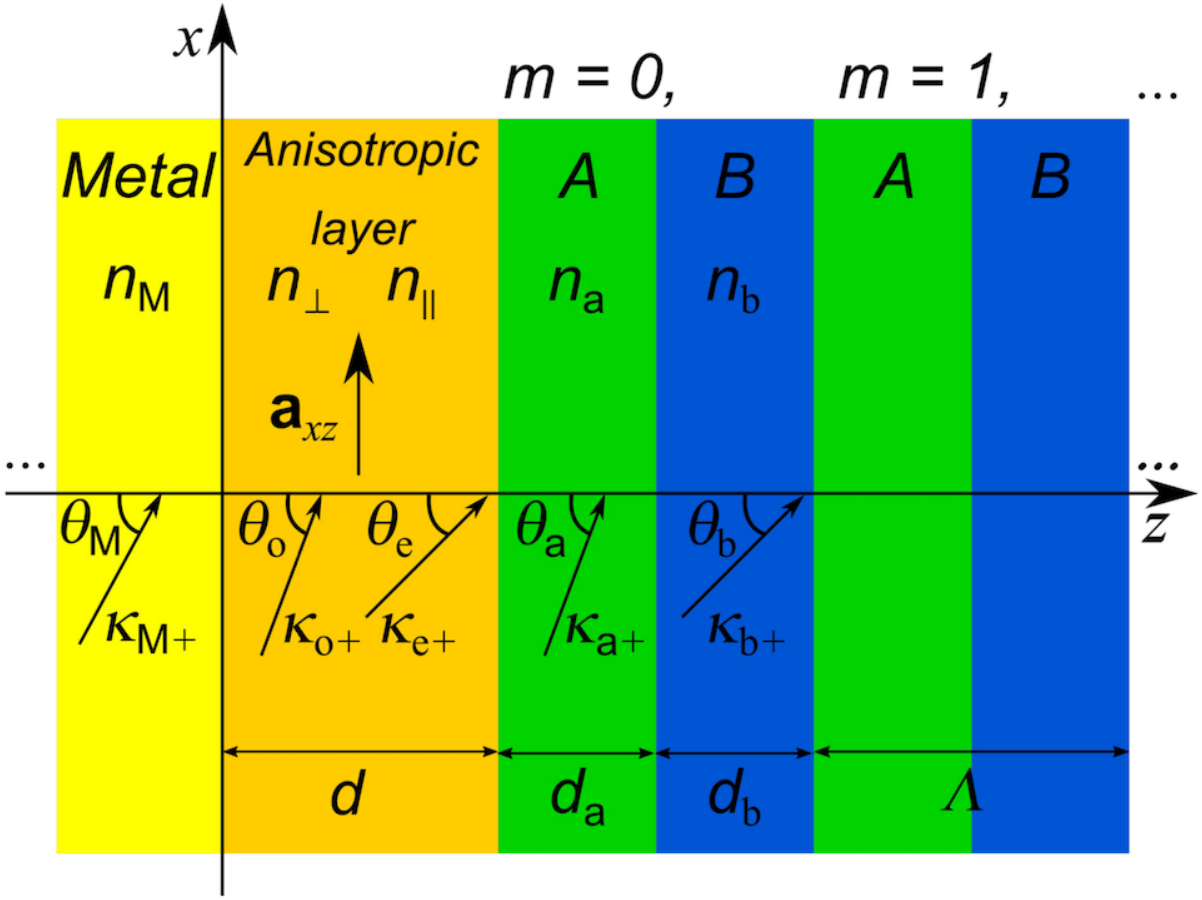


Figure 7 (taken from Supplementary Reading of [1]): Modified setup corresponding to one-dimensional optical setup for the photon

This setup is mathematically equaivalent to the one described in Section 3, and one can visualise this equivalence by considering the metal as a reflecting mirror as shown in Figure 8:

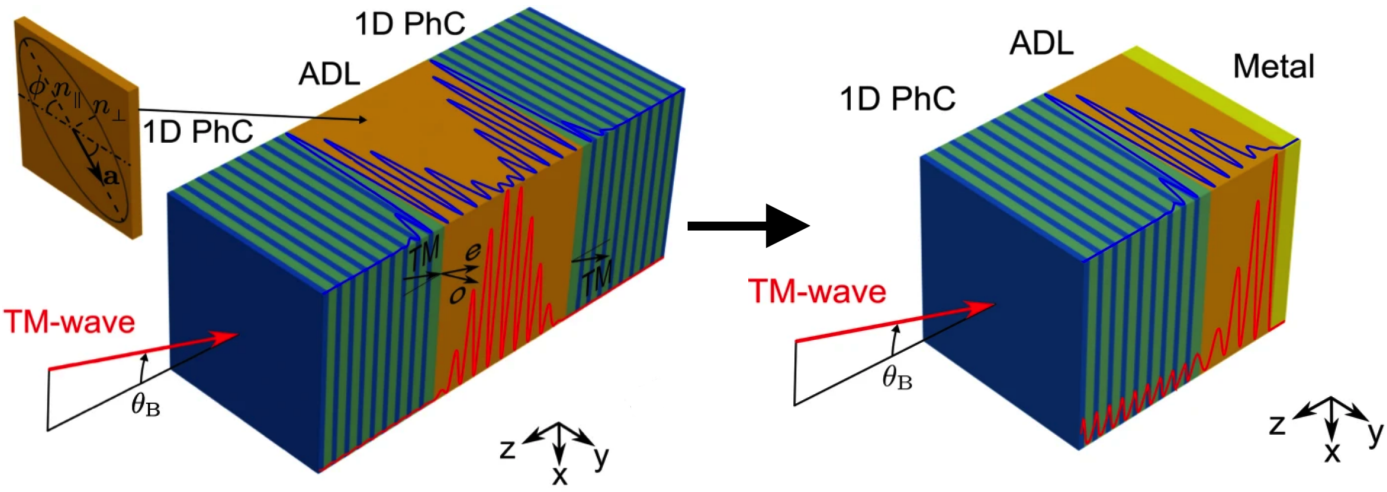


Figure 8 (taken from [1]): Correspondence between the original setup and the modified setup; the

We now have all the tools and ingredients ready to solve the wave equation in the three regions: in the PhC, inside the ADL and in the metal.

4.2) In the PhC

Layer A corresponds to Silicon Dioxide (SiO_2) and Layer B corresponds to Titanium Dioxide (TiO_2).

For TE-Waves, we write the electric and magnetic field components in the m^{th} unit cell as the sum of 'forward' (+) and a 'backward (-) propogating waves, with coefficients A_+, A_-, B_+, B_-:

$$\begin{aligned} E_{ay}^{(m)} &= e^{iK\Lambda m}(A_+ e^{ik_{az}(z-(d+m\Lambda))} + A_- e^{-ik_{az}(z-(d+m\Lambda))}) \\ H_{ax}^{(m)} &= \frac{k_{az}}{k_0} e^{iK\Lambda m}(-A_+ e^{ik_{az}(z-(d+m\Lambda))} + A_- e^{-ik_{az}(z-(d+m\Lambda))}) \\ E_{by}^{(m)} &= e^{iK\Lambda m}(B_+ e^{ik_{bz}(z-(d+m\Lambda+d_a))} + B_- e^{-ik_{bz}(z-(d+m\Lambda+d_a))}) \\ H_{bx}^{(m)} &= \frac{k_{bz}}{k_0} e^{iK\Lambda m}(-B_+ e^{ik_{bz}(z-(d+m\Lambda+d_a))} + B_- e^{-ik_{bz}(z-(d+m\Lambda+d_a))}) \end{aligned}$$

By the m^{th} unit cell we mean the regions $(d + m\Lambda < z < d + m\Lambda + d_a)$ for A and $(d + m\Lambda + d_a < z < d + (m + 1)\Lambda)$ for B.

Here, the extra factor of $e^{iK\Lambda m}$ is due to Bragg diffraction and is a consequence of the periodicity of the structure, K is the Bloch wave number, and the projection of the wave vector on the z axis is given by:

$$k_{jz} = \sqrt{k_0^2 \varepsilon_j - k_x^2}, j = a, b$$

By equating the tangential field components at the m^{th} boundary between the layers A and B, we get the following system of equations:

$$\begin{cases} A_+(e^{ik_{az}d_a} - e^{iK\Lambda}e^{-ik_{bz}d_b}) - A_- r_{ab\perp}(e^{ik_{az}d_a} - e^{iK\Lambda}e^{ik_{bz}d_b}) = 0 \\ -A_+ r_{ab\perp}(e^{ik_{az}d_a} - e^{iK\Lambda}e^{-ik_{bz}d_b}) + A_-(e^{-ik_{az}d_a} - e^{iK\Lambda}e^{ik_{bz}d_b} = 0 \end{cases}$$

where $r_{ab\perp}$ is the Fresnel reflection coefficient given by:

$$r_{ab\perp} = \frac{k_{az} - k_{bz}}{k_{bz} + k_{az}}$$

By solving for A_+, A_-, we get the dispersion relation for the TE-waves in the PhC:

$$cos(K\Lambda) = cos(k_{az}d_a)cos(k_{bz}d_b) - sin(k_{az}d_a)sin(k_{bz}d_b)\frac{1 + r_{ab\perp}^2}{1 - r_{ab\perp}^2}$$

For TM-Waves, we repeat the same procedure:

$$E_{ax}^{(m)} = e^{iK\Lambda m}(A'_{+}e^{ik_{az}(z-(d+m\Lambda))} + A'_{-}e^{-ik_{az}(z-(d+m\Lambda))})$$
$$H_{ay}^{(m)} = \frac{k_0\epsilon_a}{k_{az}}e^{iK\Lambda m}(-A'_{+}e^{ik_{az}(z-(d+m\Lambda))} + A'_{-}e^{-ik_{az}(z-(d+m\Lambda))})$$
$$E_{bx}^{(m)} = e^{iK\Lambda m}(B'_{+}e^{ik_{bz}(z-(d+m\Lambda+d_a))} + B'_{-}e^{-ik_{bz}(z-(d+m\Lambda+d_a))})$$
$$H_{by}^{(m)} = \frac{k_0\epsilon_b}{k_{bz}}e^{iK\Lambda m}(-B'_{+}e^{ik_{bz}(z-(d+m\Lambda+d_a))} + B'_{-}e^{-ik_{bz}(z-(d+m\Lambda+d_a))})$$
$$r_{ab||} = \frac{\varepsilon_b k_{az} - \varepsilon_a k_{bz}}{\varepsilon_a k_{bz} + \varepsilon_b k_{az}}$$

and obtain the dispersion relation as:

$$cos(K\Lambda) = cos(k_{az}d_a)cos(k_{bz}d_b) - sin(k_{az}d_a)sin(k_{bz}d_b)\frac{1 + r_{ab||}^2}{1 - r_{ab||}^2}$$

We see that the RHS is a function of k_x and ω. For each k_x and ω, we obtain a certain $cos(K\Lambda)$. The condition $|cos(K\Lambda)| < 1$ corresponds to real K, and therefore bands with propogating light. For $|cos(K\Lambda)| > 1$, the Bloch Wave Number becomes complex, and we get a *photonic bandgap*.

Thus, we have already zeroed in substantially on a subset of the huge ω spectrum: only those frequencies which do not lie in the bandgap will be supported, and hence we need to look only in these supported bands for the BICs.

In the following section, we have plotted the dispersion spectrum for both TE and TM Waves. The coloured regions correspond to the propogating bands, and the white regions represent the bandgaps.

Thus for a choice of k_x, only a certain set of frequencies are supported.

```
#CODE BLOCK 6A: Generating data for TE
#All caclulation parameters and equations taken from [1]
#Values of refractive indices taken from [3]

import mpmath as mp
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import colors

epsilona = 1.4776*1.4776
epsilonb = 2.65*2.65
da = 145.0 * 10**-9
db = 94.0 * 10**-9
def kazreturner(omega, kx):
    return [mp.sqrt(  (epsilona*omega*omega)- (kx*kx) )]
```

```
17
18 def kbzreturner(omega, kx):
19   return [mp.sqrt(   (epsilonb*omega*omega) - (kx*kx)   )]
20
21 def rperpreturner(kaz, kbz):
22   return [(kaz-kbz)/(kaz+kbz)]
23
24 def cosklambdareturner(kaz,kbz,rperp):
25   return [   (mp.cos(kaz*da)*mp.cos(kbz*db)) - (   (1+(rperp*rperp))/(1-(rperp*rpe
26
27 zvalue = np.zeros((501,201))
28 kx = 0 + np.linspace(0,201,201) * 10**5
29 omega = (80 + np.linspace(0,501,501)/10) * 10**5
30 for omegacounter in range(0, 501):
31   for kxcounter in range(0, 201):
32     kaz = kazreturner(omega[omegacounter], kx[kxcounter])[0]
33     kbz = kbzreturner(omega[omegacounter], kx[kxcounter])[0]
34     rperp = rperpreturner(kaz, kbz)[0]
35     cosklambda = cosklambdareturner(kaz, kbz, rperp)[0]
36
37     if (mp.fabs(cosklambda)<=1) :
38       zvalue[omegacounter][kxcounter] = 1
39     else :
40       zvalue[omegacounter][kxcounter] = 0
41 print(zvalue)
```

```
    [[0. 0. 0. ... 0. 0. 0.]
     [0. 0. 0. ... 0. 0. 0.]
     [1. 1. 1. ... 0. 0. 0.]
     ...
     [1. 1. 1. ... 0. 0. 0.]
     [1. 1. 1. ... 0. 0. 0.]
     [1. 1. 1. ... 0. 0. 0.]]
```

```
 1 #CODE BLOCK 6B: Generating data for TM
 2 #All caclulation parameters and equations taken from [1]
 3 #Values of refractive indices taken from [3]
 4
 5
 6 import mpmath as mp
 7 import numpy as np
 8 import matplotlib.pyplot as plt
 9 import matplotlib as mpl
10 from matplotlib import colors
11
12 epsilona = 1.4776*1.4776
13 epsilonb = 2.65*2.65
14 da = 145.0 * 10**-9
15 db = 94.0 * 10**-9
16 def kazreturner(omega, kx):
17    return [mp.sqrt(  (epsilona*omega*omega)- (kx*kx) )]
18
```

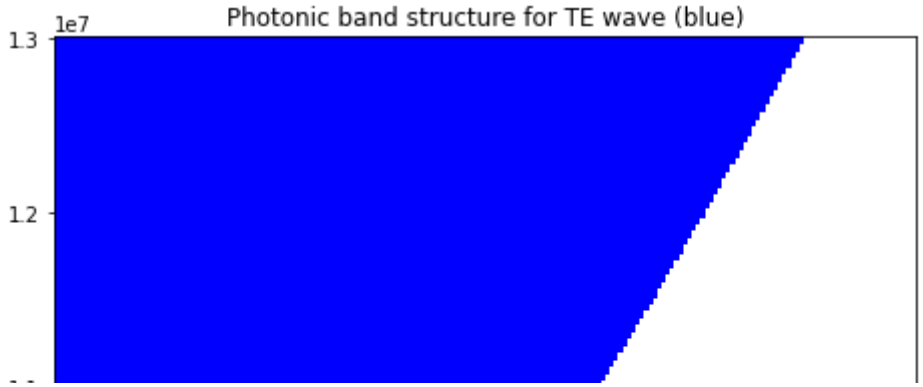
```
def kbzreturner(omega, kx):
  return [mp.sqrt(   (epsilonb*omega*omega) - (kx*kx)   )]

def rparallelreturner(kaz, kbz):
  return [(epsilonb*kaz-epsilona*kbz)/(epsilona*kbz+epsilonb*kaz)]

def cosklambdareturner(kaz,kbz,rparallel):
  return [   (mp.cos(kaz*da)*mp.cos(kbz*db)) - (   (1+(rparallel*rparallel))/(1-(rp

zvalue2 = np.zeros((501,201))
kx = 0 + np.linspace(0,201,201) * 10**5
omega = (80 + np.linspace(0,501,501)/10) * 10**5
for omegacounter in range(0, 501):
  for kxcounter in range(0, 201):
    kaz = kazreturner(omega[omegacounter], kx[kxcounter])[0]
    kbz = kbzreturner(omega[omegacounter], kx[kxcounter])[0]
    rparallel = rparallelreturner(kaz, kbz)[0]
    cosklambda = cosklambdareturner(kaz, kbz, rparallel)[0]
    if (mp.fabs(cosklambda)<=1) :
      zvalue2[omegacounter][kxcounter] = 1
    else :
      zvalue2[omegacounter][kxcounter] = 0
print(zvalue2)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 1. 1. ... 0. 0. 0.]
 [1. 1. 1. ... 0. 0. 0.]
 [1. 1. 1. ... 0. 0. 0.]]
```

```
#CODE BLOCK 6C: Photonic Band Structure
#All caclulation parameters and equations taken from [1]
#Values of refractive indices taken from [3]


fig, axs = plt.subplots(2, 1,figsize=(7,14))
ax = axs[0]
axs[0].set_xlabel(r"$k_x$ in $\mu m^{-1}$")
axs[0].set_ylabel(r"$\omega$ in $\mu m^{-1}$")
axs[0].grid(True)
cmap1 = mpl.colors.ListedColormap(['w', 'b'])
bounds = [0., 0.5, 1.]
norm = mpl.colors.BoundaryNorm(bounds, cmap1.N)
c = ax.pcolor(kx,omega,zvalue, cmap=cmap1, vmin=0, vmax=1)
ax.set_title('Photonic band structure for TE wave (blue)')
ax = axs[1]
axs[1].set_xlabel(r"$k_x$ in $\mu m^{-1}$")
axs[1].set_ylabel(r"$\omega$ in $\mu m^{-1}$")
axs[1].grid(True)
cmap2 = mpl.colors.ListedColormap(['w', 'r'])
```

```
cmap2 = mpl.colors.ListedColormap(['w', 'r'])
bounds = [0., 0.5, 1.]
norm = mpl.colors.BoundaryNorm(bounds, cmap2.N)
c = ax.pcolor(kx, omega,zvalue2, cmap=cmap2, vmin=0, vmax=1)
ax.set_title(r"Photonic band structure for TM wave (red) ")
fig.tight_layout()
plt.show()
```

```
<ipython-input-3-7a0dafe353cd>:14: MatplotlibDeprecationWarning: shading='flat' v
  c = ax.pcolor(kx,omega,zvalue, cmap=cmap1, vmin=0, vmax=1)
<ipython-input-3-7a0dafe353cd>:23: MatplotlibDeprecationWarning: shading='flat' v
  c = ax.pcolor(kx, omega,zvalue2, cmap=cmap2, vmin=0, vmax=1)
```



4.3) In the ADL

The ADL in our setup is an E7 Liquid Crystal Layer [1]

For a pair of media, Brewster's angle is the angle of incidence (θ_i) such that the angle of refraction (θ_r) is $\pi/2 - \theta_i$. Just like we injected only a spin-up electron beam in the toy-model, we inject our EM-wave at the Brewster's angle to ensure that only one polarisation passes through. In our PhC with alternating isotropic layers A and B, the light propagates at the Brewster's angle given by $\theta_a = \arctan(n_b/n_a)$. In the ADL however, mixing of polarizations of light results in entirely new wave vectors.

In our setup the anistropy axis vector $\mathbf{a}$ is given by $\mathbf{a} = (sin(\phi), cos(\phi), 0)$ when it makes an angle ϕ with the y-axis. Relative anisotropy $\eta = \varepsilon'/\varepsilon_o$ and ϕ determines the direction θ_e and length k_e of the wave vector of the e-wave

$$\theta_e = \arctan\left(\frac{k_x}{k_{ez}}\right) = \arctan\left(\frac{k_x}{\sqrt{\varepsilon_o k_0^2(1+\eta) - k_x^2(1+\eta sin^2(\phi))}}\right)$$

$$k_e = k_0 n_e = k_0\sqrt{\varepsilon_e}, \varepsilon_e = \frac{\varepsilon_o(1+\eta)}{1+\eta sin^2(\phi)sin^2(\theta_e)}$$

The direction and length of the wave vector for the o-wave does not depend on direction of the optical axis:

$$\theta_o = \arctan\left(\frac{k_x}{k_{oz}}\right) = \arctan\left(\frac{k_x}{\sqrt{\varepsilon_o k_o^2 - k_x^2}}\right), k_o = k_0 n_o$$

We define a unit vector along the direction of the optical axis:
$\kappa_{j+,-} = k_{j+,-}/k_j = (sin(\theta_j), 0, \pm cos(\theta_j)), j = e, o$

Using this unit vector, we find the e-waves and o-waves in the ADL:

$$\mathbf{E}_{e+,-} = E_{e+,-}\left(a - \kappa_{e+,-}(a\kappa_{j+,-})\frac{\varepsilon_o}{\varepsilon_e}\right)$$

$$\mathbf{H}_{e+,-} = \frac{k_e}{k_0}[\kappa_{e+,-} \times \mathbf{E}_{e+,-}]$$

$$\mathbf{E}_{o+,-} = E_{o+,-}[a \times \kappa_{o+,-}]$$

$$\mathbf{H}_{o+,-} = \frac{k_o}{k_0}[\kappa_{o+,-} \times \mathbf{E}_{o+,-}]$$

where $E_{o+,-}$, $E_{e+,-}$, are amplitudes. We now proceed to write a general solution in the ADL as the sum of forward and backward waves:

$$\mathbf{E}_{e,o} = \mathbf{E}_{e,o+}e^{ik_{e,oz}z} + \mathbf{E}_{e,o-}e^{-ik_{e,oz}z}$$

$$\mathbf{H}_{e,o} = \mathbf{H}_{e,o+}e^{ik_{e,oz}z} + \mathbf{H}_{e,o-}e^{-ik_{e,oz}z}$$

Using these general expressions, we can write the tangential components of the e-wave:

$$E_{ex} = (E_{e+}e^{ik_{ez}z} + E_{e-}e^{-ik_{ez}z})\left(1 - \frac{\varepsilon_e}{\varepsilon_o}sin^2(\theta_e)\right)sin(\phi)$$

$$E_{ey} = (E_{e+}e^{ik_{ez}z} + E_{e-}e^{-ik_{ez}z})cos(\phi)$$

$$H_{ex} = \frac{k_e}{k_0}(-E_{e+}e^{ik_{ez}z} + E_{e-}e^{-ik_{ez}z})cos(\theta_e)cos(\phi)$$

$$H_{ey} = \frac{k_e}{k_0}(E_{e+}e^{ik_{ez}z} - E_{e-}e^{-ik_{ez}z})cos(\theta_e)sin(\phi)$$

Tangential component of the o-wave:

$$E_{ox} = (E_{o+}e^{ik_{oz}z} - E_{o-}e^{-ik_{oz}z})cos(\theta_o)cos(\phi)$$

$$E_{oy} = (-E_{o+}e^{ik_{oz}z} + E_{o-}e^{-ik_{oz}z})cos(\theta_o)sin(\phi)$$

$$H_{ox} = \frac{k_o}{k_0}(E_{o+}e^{ik_{oz}z} + E_{o-}e^{-ik_{oz}z})cos^2(\theta_o)sin(\phi)$$

$$H_{oy} = \frac{k_o}{k_0}(E_{o+}e^{ik_{oz}z} + E_{o-}e^{-ik_{oz}z})cos(\phi)$$

4.4) In the Metal

So far we have mentioned the components of the light wave in the PhC (TE and TM) and the ADL (e-wave and o-wave). The last part of our setup consists of a metallic film (the experimental setup used a gold mirror film with refractive index $n_M = \sqrt{\varepsilon_M}$). While using this setup certainly has its

advantages, the imaginary coefficient of the refractive index of the metal brings unnecessary material losses and sets the limit of the Q-factor (discussed ahead)

An isotropic semi-infinite metal mirror is opaque to waves of any polarization – light is partially reflected from its surface and partially absorbed (Im(n_M) > 0) inside its bulk. We write the TE-wave field in the metal for z < 0 as follows

$$E_{My} = M_- e^{-ik_{Mz}z}$$

$$H_{Mx} = \frac{k_{Mz}}{k_0} M_- e^{-ik_{Mz}z}$$

Similarly the TM-wave field in the metal for z < 0 is

$$E_{Mx} = -M'_- e^{-ik_{Mz}z}$$

$$H_{My} = \frac{k_0 \varepsilon_M}{k_{Mz}} M'_- e^{-ik_{Mz}z}$$

$$k_{Mz} = \sqrt{k_0^2 \varepsilon_M - k_x^2}$$

Matching the fields in the ADL to the decaying fields in the metallic mirror gives us the final

4.5) Final solution

We combine the three sets of equations for PhC, ADL and metal, beginning by first finding the waves in the $m = 0$ unit cell adjacent to the ADL:

$$E_{ay}^{(0)} = A_+ e^{ik_{az}(z-d)} + A_- e^{-ik_{az}(z-d)}$$

$$H_{ax}^{(0)} = \frac{k_{az}}{k_0}(-A_+ e^{ik_{az}(z-d)} + A_- e^{-ik_{az}(z-d)})$$

$$E_{ax}^{(0)} = A'_+ e^{ik_{az}(z-d)}$$

$$H_{ay}^{(0)} = \frac{k_0 \varepsilon_a}{k_{az}} A'_+ e^{ik_{az}(z-d)}$$

We now apply boundary conditions at all the interfaces and equate the tangential components wherever necessary, to obtain a system of 10 equations in 10 unknowns. The solution to this system will give us the dispersion relation corresponding to the whole setup.

For a given k_x, we can solve this system of equations(i.e the dispersion relation) and find the complex frequency which is be given by

$$\omega = \omega_r - i\gamma$$

Now, from our previous considerations, we already know that for a BIC, ω is purely real. We state another fact here: the converse is also true, i.e if $\gamma = 0$, then the frequency obtained *must* correspond to a BIC.

The reason for this, as stated in [2], is that γ represents the leakage rate of the wave, and for a bound state that is spatially confined, there is no leakage.

Thus, to look for BICs, we can fix a k_x and change the orientation of the anisotropy axis, and find the complex frequency corresponding to that angle ϕ.

We then define a Quality factor of the resonance 'Q' as follows:

$$Q = \omega_r / 2\gamma$$

On plotting Q as a function of ϕ, we would expect that at the points at which BICs occur, the Q-factor becomes infinite. This is represented in Figure 9:

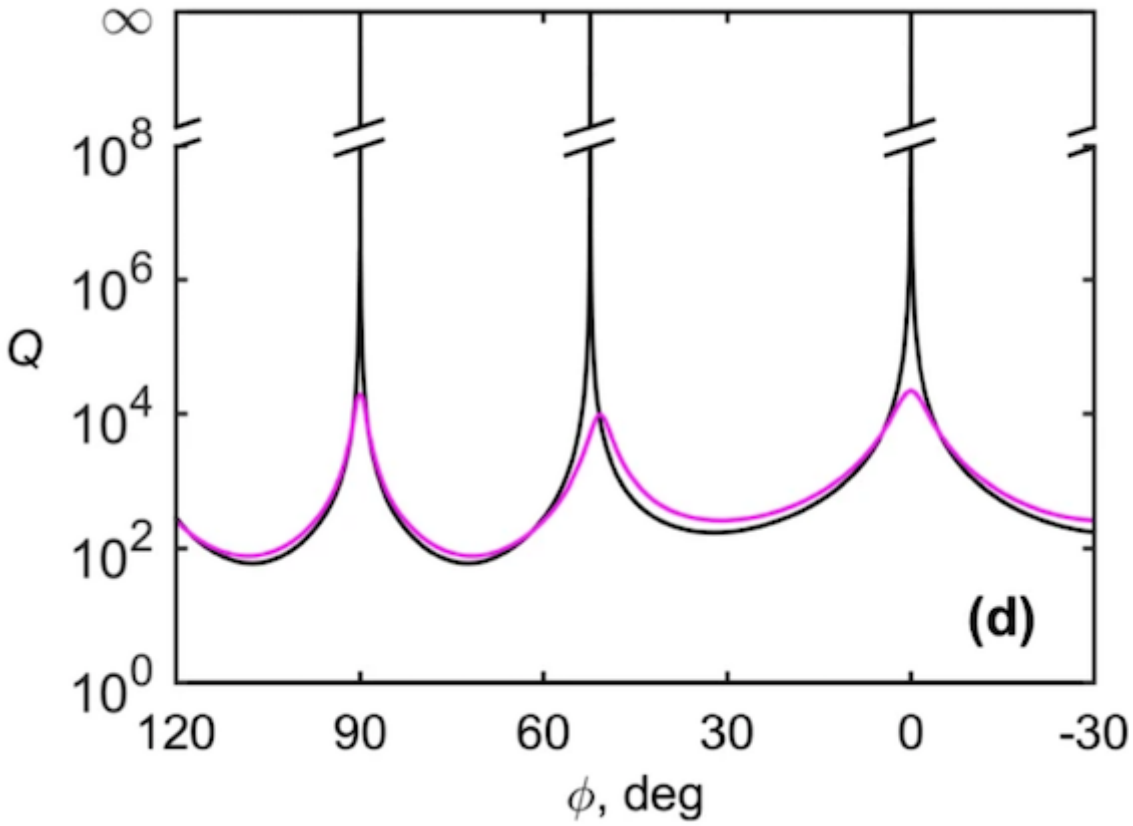


Figure 9 (taken from [1]): Q-factor as a function of the angle of orientation of the anisotropy axis

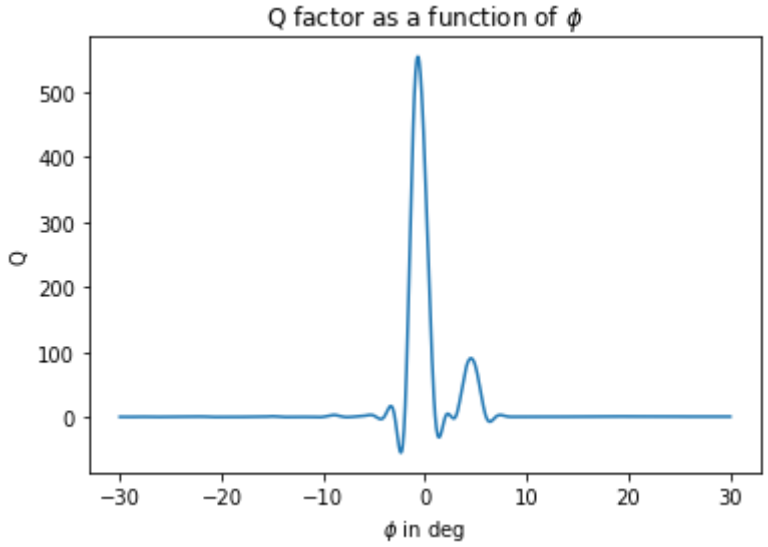
The magenta line in the figure corresponds to calculations made using an imperfect metal with evanescent attenuation. We pereform similar calculations for the Q-factor in the subsequent code blocks:

```
1 #CODE BLOCK 7A: Q-FACTOR FOR LOW ANGLES
2 #All caclulation parameters and equations taken from [1]
```

```
#Values of refractive indices taken from [3]


import mpmath as mp
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.interpolate import CubicSpline

na = 1.4776
nb = 2.65
nm = 0.14 + 20j
nperp = 1.57
npar = 1.7

epsilona = na*na
epsilonb = nb*nb
epsilonm = nm*nm

epsilon_o = nperp*nperp
epsilon_dash = npar*npar - epsilon_o
eta = epsilon_dash/epsilon_o

da = 145.0 * 10**-9
db = 94.0 * 10**-9
d = 1.375 * 10**-6

theta_a = mp.atan(nb/na)

qvalue = []
anglevalue = []

for angle in range(-30, 30):
  phi = np.radians(angle) ###

  def kreturner(k, kx, kaz, kbz, kmz, kez, koz, theta_e, theta_o, theta_m, epsilon_
    return[kx - 1.52 * k * mp.sin((53.1/180) * mp.pi),

          kaz -  mp.sqrt(  (epsilona*k*k)- (kx*kx) )  ,
          kbz -  mp.sqrt(   (epsilonb*k*k) - (kx*kx)   ),
          kmz -  mp.sqrt(   (epsilonm*k*k) - (kx*kx)   ),

          kez - mp.sqrt(   (epsilon_o * k* k * (1+eta)) - (kx * kx * (1 + eta * mp.
          koz - mp.sqrt(   (epsilon_o * k* k) - (kx*kx)   ),

          theta_e - mp.atan(kx/kez),
          theta_o - mp.atan(kx/koz),
          theta_m - mp.atan(kx/kmz), ##

          epsilon_e - (epsilon_o * (1 + eta))/(1 + eta*mp.sin(phi)**2 * mp.sin(thet

          rabperp - (kaz - kbz)/(kaz + kbz),
```

```
 55
 56          r1ae - (kaz - kez)/(kaz + kez),
 57          r1me - (kmz - kez)/(kmz + kez),
 58          t1ae - 2*kaz/(kaz + kez),
 59          t1me - 2*kmz/(kmz + kez),
 60
 61          r1ao - (kaz - koz)/(kaz + koz),
 62          r1mo - (kmz - koz)/(kmz + koz),
 63          t1ao - 2*kaz/((kaz + kez)*mp.cos(theta_o)),
 64          t1mo - 2*kmz/((kmz + kez)*mp.cos(theta_o)),
 65
 66          r2ae - (kaz*(1 - (epsilon_e/epsilon_o)*mp.sin(theta_e)**2) - kez*mp.cos(t
 67          r2me - (kmz*(1 - (epsilon_e/epsilon_o)*mp.sin(theta_e)**2) - kez*mp.cos(t
 68          t2ae - 2*kaz/(kaz*(1 - (epsilon_e/epsilon_o)*(mp.sin(theta_e))**2) + kez*
 69          t2me - 2*kmz/(kmz*(1 - (epsilon_e/epsilon_o)*(mp.sin(theta_e))**2) + kez*
 70
 71          r2ao - (  kaz*(mp.cos(theta_o))**2 - koz*(mp.cos(theta_a))**2    )/(  kaz
 72          r2mo - (  kmz*(mp.cos(theta_o))**2 - koz*(mp.cos(theta_m))**2    )/(  kmz
 73          t2ao - 2*kaz*mp.cos(theta_o)/(  kaz*(mp.cos(theta_o))**2 + koz*(mp.cos(th
 74          t2mo - 2*kmz*mp.cos(theta_o)/(  kmz*(mp.cos(theta_o))**2 + koz*(mp.cos(th
 75
 76          alpha1 - ( (t2mo*mp.sin(phi)*r2me)/(t2me*mp.cos(phi)) + (t1mo*mp.cos(phi)
 77          alpha2 - (  (t1mo*mp.cos(phi))/(t1me*mp.sin(phi)) + (t2mo*mp.sin(phi))/(t
 78
 79
 80          V - ( (  (mp.cos(phi)/t2ao)*( mp.exp(-1j*koz*d)  -  (   t1mo*mp.cos(phi)*
 81
 82          U - (  (-1*V*alpha1 + r2mo - r1mo)/(alpha2)  ),
 83          y - ( (t1mo*mp.cos(phi))*(U + V*r1me)/(t1me*mp.sin(phi)) + r1mo  ),
 84
 85          x - ( ( (mp.sin(phi)*(r1ao*mp.exp(-1j*koz*d) - y*mp.exp(1j*koz*d) ))/(t1a
 86          (x*mp.exp(1j*kaz*da) -  rabperp*mp.exp(-1j*kaz*da))*(mp.exp(1j*kbz*db) -
 87
 88          ]
 89
 90
 91  num = 0.1 + 0.1j
 92  inp = []
 93
 94  for i in range(34):
 95    inp.append(num)
 96
 97  def func():
 98    array = mp.findroot(kreturner, x0=inp, verbose = False,  verify = False)
 99    return array
100
101  try:
102    solution = func()
103    wavenumber = solution[0]
104    omega = 3 * 10**8 * wavenumber
105    Q = mp.fabs(omega.real)/(2*mp.fabs(omega.imag))
106    anglevalue.append(angle)
```

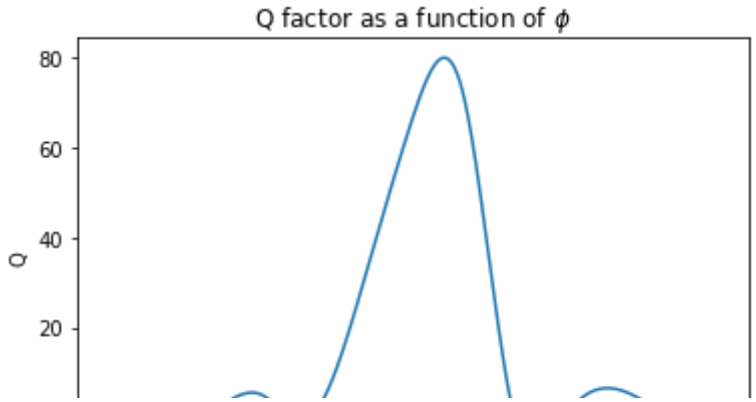
```
106     anglevalue.append(angle)
107     qvalue.append(Q)
108   except:
109     pass
110
111 cs = CubicSpline(anglevalue, qvalue, bc_type='natural')
112 xnew = np.linspace(-30, 30, 1000)
113 plt.plot(xnew, cs(xnew))
114 plt.xlabel(r"$\phi$ in deg")
115 plt.ylabel('Q')
116 plt.title(r"Q factor as a function of $\phi$")
117 plt.show()
```




```
 1 ODE BLOCK 7B: Q-FACTOR FOR INTERMEDIATE ANGLES
 2 ll caclulation parameters and equations taken from [1]
 3 alues of refractive indices taken from [3]
 4
 5
 6 port mpmath as mp
 7 port numpy as np
 8 port matplotlib.pyplot as plt
 9 port scipy
10 om scipy.interpolate import CubicSpline
11
12  = 1.4776
13  = 2.65
14  = 0.14 + 20j
15 erp = 1.57
16 ar = 1.7
17
18 silona = na*na
19 silonb = nb*nb
20 silonm = nm*nm
21
22 silon_o = nperp*nperp
23 silon_dash = npar*npar - epsilon_o
```

```
a = epsilon_dash/epsilon_o

 = 145.0 * 10**-9
 = 94.0 * 10**-9
= 1.375 * 10**-6

eta_a = mp.atan(nb/na)

alue = []
glevalue = []

 anglee in range(0, 12):
angle = 44 + anglee
phi = np.radians(angle) ###

def kreturner(k, kx, kaz, kbz, kmz, kez, koz, theta_e, theta_o, theta_m, epsilon_e,
  return[kx - 1.52 * k * mp.sin((53.1/180) * mp.pi),

        kaz -  mp.sqrt(  (epsilona*k*k)- (kx*kx) )  ,
        kbz -  mp.sqrt(   (epsilonb*k*k) - (kx*kx)   ),
        kmz -  mp.sqrt(   (epsilonm*k*k) - (kx*kx)   ),

        kez - mp.sqrt(   (epsilon_o * k* k * (1+eta)) - (kx * kx * (1 + eta * mp.si
        koz - mp.sqrt(   (epsilon_o * k* k) - (kx*kx)   ),

        theta_e - mp.atan(kx/kez),
        theta_o - mp.atan(kx/koz),
        theta_m - mp.atan(kx/kmz), ##


        epsilon_e - (epsilon_o * (1 + eta))/(1 + eta*mp.sin(phi)**2 * mp.sin(theta_
        rabperp - (kaz - kbz)/(kaz + kbz),


        r1ae - (kaz - kez)/(kaz + kez),
        r1me - (kmz - kez)/(kmz + kez),
        t1ae - 2*kaz/(kaz + kez),
        t1me - 2*kmz/(kmz + kez),


        r1ao - (kaz - koz)/(kaz + koz),
        r1mo - (kmz - koz)/(kmz + koz),
        t1ao - 2*kaz/((kaz + kez)*mp.cos(theta_o)),
        t1mo - 2*kmz/((kmz + kez)*mp.cos(theta_o)),

        r2ae - (kaz*(1 - (epsilon_e/epsilon_o)*mp.sin(theta_e)**2) - kez*mp.cos(the
        r2me - (kmz*(1 - (epsilon_e/epsilon_o)*mp.sin(theta_e)**2) - kez*mp.cos(the
        t2ae - 2*kaz/(kaz*(1 - (epsilon_e/epsilon_o)*(mp.sin(theta_e))**2) + kez*mp
        t2me - 2*kmz/(kmz*(1 - (epsilon_e/epsilon_o)*(mp.sin(theta_e))**2) + kez*mp

        r2ao - (  kaz*(mp.cos(theta_o))**2 - koz*(mp.cos(theta_a))**2    )/(  kaz*(
```

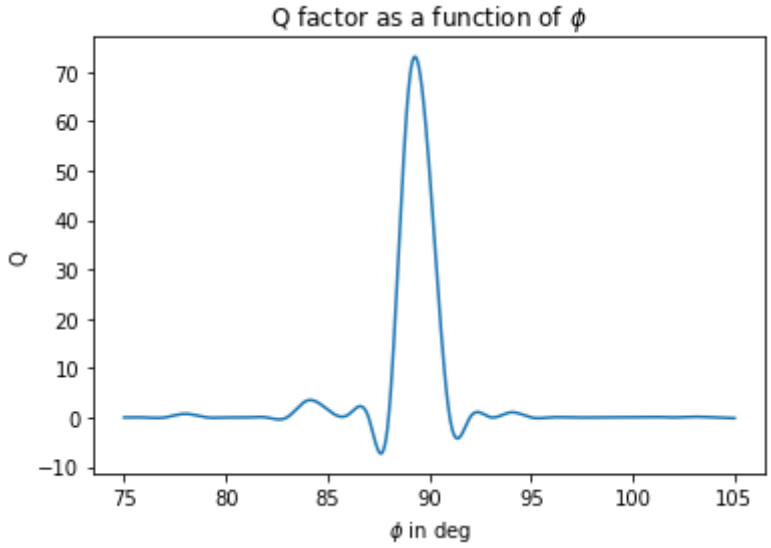
```
        r2mo - (  kmz*(mp.cos(theta_o))**2 - koz*(mp.cos(theta_m))**2    )/(  kmz*(
        t2ao - 2*kaz*mp.cos(theta_o)/(  kaz*(mp.cos(theta_o))**2 + koz*(mp.cos(thet
        t2mo - 2*kmz*mp.cos(theta_o)/(  kmz*(mp.cos(theta_o))**2 + koz*(mp.cos(thet

        alpha1 - ( (t2mo*mp.sin(phi)*r2me)/(t2me*mp.cos(phi)) + (t1mo*mp.cos(phi)*r
        alpha2 - (  (t1mo*mp.cos(phi))/(t1me*mp.sin(phi)) + (t2mo*mp.sin(phi))/(t2m


        V - ( (  (mp.cos(phi)/t2ao)*( mp.exp(-1j*koz*d)  -  (   t1mo*mp.cos(phi)*(r

        U - (  (-1*V*alpha1 + r2mo - r1mo)/(alpha2)  ),
        y - ( (t1mo*mp.cos(phi))*(U + V*r1me)/(t1me*mp.sin(phi)) + r1mo  ),

        x - ( ( (mp.sin(phi)*(r1ao*mp.exp(-1j*koz*d) - y*mp.exp(1j*koz*d) ))/(t1ao)
        (x*mp.exp(1j*kaz*da) -  rabperp*mp.exp(-1j*kaz*da))*(mp.exp(1j*kbz*db) -  x

        ]

num = 0.000001 + 0.0000001j
num = num + 57*(0.25 + 0.25j)
inp = []

for i in range(34):
  inp.append(num)

def func():
  array = mp.findroot(kreturner, x0=inp, verbose = False,  verify = False)
  return array

try:
  solution = func()
  wavenumber = solution[0]
  omega = 3 * 10**8 * wavenumber
  Q = mp.fabs(omega.real)/(2*mp.fabs(omega.imag))
  anglevalue.append(angle)
  qvalue.append(Q)
except:
  pass

 = CubicSpline(anglevalue, qvalue, bc_type='natural')
ew = np.linspace(44, 56, 1000)
t.plot(xnew, cs(xnew))
t.xlabel(r"$\phi$ in deg")
t.ylabel('Q')
t.title(r"Q factor as a function of $\phi$")
t.show()
```




```
ODE BLOCK 7C: Q-FACTOR FOR LARGE ANGLES
ll caclulation parameters and equations taken from [1]
alues of refractive indices taken from [3]


port mpmath as mp
port numpy as np
port matplotlib.pyplot as plt
port scipy
om scipy.interpolate import CubicSpline

 = 1.4776
 = 2.65
 = 0.14 + 20j
erp = 1.57
ar = 1.7

silona = na*na
silonb = nb*nb
silonm = nm*nm

silon_o = nperp*nperp
silon_dash = npar*npar - epsilon_o
a = epsilon_dash/epsilon_o

 = 145.0 * 10**-9
 = 94.0 * 10**-9
= 1.375 * 10**-6

eta_a = mp.atan(nb/na)
alue = []
glevalue = []

r anglee in range(0, 30):
angle = 75 + anglee
phi = np.radians(angle) ###

def kreturner(k, kx, kaz, kbz, kmz, kez, koz, theta_e, theta_o, theta_m, epsilon_e,
  return[kx - 1.52 * k * mp.sin((53.1/180) * mp.pi),

```

```
        kaz -  mp.sqrt(  (epsilona*k*k)- (kx*kx) )  ,
        kbz -  mp.sqrt(   (epsilonb*k*k) - (kx*kx)   ),
        kmz -  mp.sqrt(   (epsilonm*k*k) - (kx*kx)   ),

        kez - mp.sqrt(   (epsilon_o * k* k * (1+eta)) - (kx * kx * (1 + eta * mp.si
        koz - mp.sqrt(   (epsilon_o * k* k) - (kx*kx)   ),

        theta_e - mp.atan(kx/kez),
        theta_o - mp.atan(kx/koz),
        theta_m - mp.atan(kx/kmz), ##

        epsilon_e - (epsilon_o * (1 + eta))/(1 + eta*mp.sin(phi)**2 * mp.sin(theta_
        rabperp - (kaz - kbz)/(kaz + kbz),

        r1ae - (kaz - kez)/(kaz + kez),
        r1me - (kmz - kez)/(kmz + kez),
        t1ae - 2*kaz/(kaz + kez),
        t1me - 2*kmz/(kmz + kez),

        r1ao - (kaz - koz)/(kaz + koz),
        r1mo - (kmz - koz)/(kmz + koz),
        t1ao - 2*kaz/((kaz + kez)*mp.cos(theta_o)),
        t1mo - 2*kmz/((kmz + kez)*mp.cos(theta_o)),

        r2ae - (kaz*(1 - (epsilon_e/epsilon_o)*mp.sin(theta_e)**2) - kez*mp.cos(the
        r2me - (kmz*(1 - (epsilon_e/epsilon_o)*mp.sin(theta_e)**2) - kez*mp.cos(the
        t2ae - 2*kaz/(kaz*(1 - (epsilon_e/epsilon_o)*(mp.sin(theta_e))**2) + kez*mp
        t2me - 2*kmz/(kmz*(1 - (epsilon_e/epsilon_o)*(mp.sin(theta_e))**2) + kez*mp

        r2ao - (  kaz*(mp.cos(theta_o))**2 - koz*(mp.cos(theta_a))**2    )/(  kaz*(
        r2mo - (  kmz*(mp.cos(theta_o))**2 - koz*(mp.cos(theta_m))**2    )/(  kmz*(
        t2ao - 2*kaz*mp.cos(theta_o)/(  kaz*(mp.cos(theta_o))**2 + koz*(mp.cos(thet
        t2mo - 2*kmz*mp.cos(theta_o)/(  kmz*(mp.cos(theta_o))**2 + koz*(mp.cos(thet

        alpha1 - ( (t2mo*mp.sin(phi)*r2me)/(t2me*mp.cos(phi)) + (t1mo*mp.cos(phi)*r
        alpha2 - (  (t1mo*mp.cos(phi))/(t1me*mp.sin(phi)) + (t2mo*mp.sin(phi))/(t2m

        V - ( (  (mp.cos(phi)/t2ao)*( mp.exp(-1j*koz*d)  -  (   t1mo*mp.cos(phi)*(r

        U - (  (-1*V*alpha1 + r2mo - r1mo)/(alpha2)  ),
        y - ( (t1mo*mp.cos(phi))*(U + V*r1me)/(t1me*mp.sin(phi)) + r1mo  ),

        x - ( ( (mp.sin(phi)*(r1ao*mp.exp(-1j*koz*d) - y*mp.exp(1j*koz*d) ))/(t1ao)
        (x*mp.exp(1j*kaz*da) -  rabperp*mp.exp(-1j*kaz*da))*(mp.exp(1j*kbz*db) -  x

        ]

num = 0.000001 + 0.0000001j
num = num + 15*(0.25 + 0.25j)
inp = []

for j in range(34):
```

```
 93   inp.append(num)
 94
 95 def func():
 96   array = mp.findroot(kreturner, x0=inp, verbose = False,  verify = False)
 97   return array
 98
 99 try:
100   solution = func()
101   wavenumber = solution[0]
102   omega = 3 * 10**8 * wavenumber
103   Q = mp.fabs(omega.real)/(2*mp.fabs(omega.imag))
104   anglevalue.append(angle)
105   qvalue.append(Q)
106 except:
107   pass
108
109  = CubicSpline(anglevalue, qvalue, bc_type='natural')
110 ew = np.linspace(75, 105, 1000)
111 t.plot(xnew, cs(xnew))
112 t.xlabel(r"$\phi$ in deg")
113 t.ylabel('Q')
114 t.title(r"Q factor as a function of $\phi$")
115 t.show()
```



5) Some Interesting Features of the Photonic System

5.1) A physical interpretation of the BICs

We solved wave equations(Schrödinger and Maxwell) analytically. Now, *it just so happened* that some of the solutions were bound states embedded in a continuum.

One may wonder whether these BICs were random, or wether there was some underlying principle, using which we could have predicted when the BIC would occur.

The answer to this question is that BICs can be classified into different categories based on the mechanism by which they are formed, and these are discussed in [2]. The BICs that we have seen in this paper fall into two categories: Symmetry-Protected (SP) BICs and Friedrich-Wintgen (FW) BICs.

Symmetry-Protected (SP) BICs occur when a system exhibits rotational or reflection symmetry, and modes of different symmetry classes completely decouple. In such a scenario, we tend to find the bound state of one symmetry class embedded in the continuous spectrum of another symmetry class, and their coupling is prohibited as long as the symmetry is preserved.

In our photonic crystal system, the BICs corresponding to $\theta = 0^\circ$ and $\theta = 90^\circ$ correspond to the SP BICs, and in both these cases the symmetry is the 180° rotational symmetry along the z-axis.

Friedrich-Wintgen BICs arise as a result of complete destructive interference of two resonant states. Precise destructive interference in the outer regions result in a state that is bound inside the central layer. This is illustrated in Figure 10:

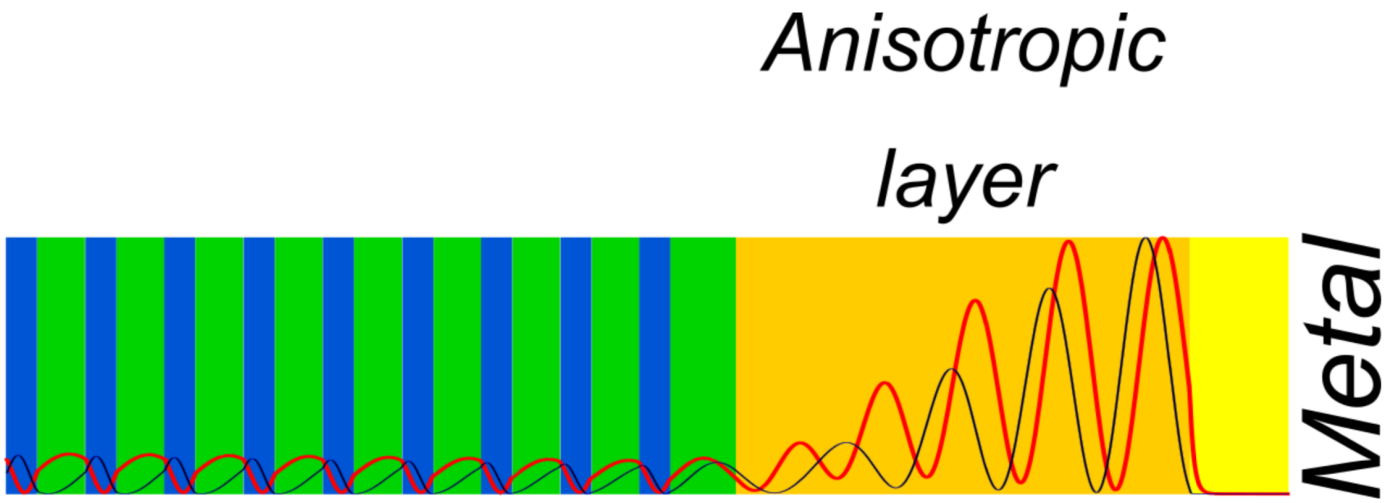


Figure 10 (taken from Supplementary Reading of [1]): Schematic showing the destructive interference of resonant modes leading to formation of BICs

However, it should be understood that this is not just any odd example of destructive interference; the two modes which interfere must have specific properties as discusses in the following section:

5.2) A note on the mixing of channels

In the 1D-layered PhC structure, FW BICs can't occur if the polarizsations (TE and TM) can be seperated. This is because in 1D systems, there are no transmittance zeroes, and respectively, there are no FW BICs because of the absence of degeneracy of avoided crossing resonances which could result in complete destructive interference.

Hence, the ADL is necessary, as it induces the mixing of the two-polarizations and consequently, FW BICs can occur because of complete destructive interference of the two channels with different polarisations in the defect layer

5.3) Advantages of this experimental setup

In this setup, we start with a huge spectrum of complex frequencies, and the first step to narrowing down our range to real frequencies is accomplished by the dispersion relation of the PhC. The ADL then mixes the polarisations for complete destructive interference of avoided crossing resonances, while the metal acts as a mirror.

This setup thus effectively eliminates a wide range of frequencies which do not give BICs, and allows us to physically observe BICs with real fields and real frequencies.

This setup is fabrication-friendly, as we have reduced two semi-infinite PhC arms to one semi-infinite arm. One can also easily control the length of the liquid-crystal ADL since it is poured in the cavity between the mirror and the PhC arm.

The BICs presented in the setup described above have been experimentally observed in [1], and this sort of system can pave the way for novel tunable high-quality devices both in spintronics and photonics [1].

Appendix

References

[1] Pankin, P.S., Wu, BR., Yang, JH. *et al*. One-dimensional photonic bound states in the continuum. *Commun Phys* **3**, 91 (2020). https://doi.org/10.1038/s42005-020-0353-z

[2] Hsu, C., Zhen, B., Stone, A. *et al*. Bound states in the continuum. *Nat Rev Mater* **1**, 16048 (2016). https://doi.org/10.1038/natrevmats.2016.48

[3] https://refractiveindex.info/

Bibliography

(1) Timofeev, I. V., Maksimov, D. N. & Sadreev, A. F. Optical defect mode with tunable Q factor in a one-dimensional anisotropic photonic crystal. *Phys. Rev. B* **97**, 24306 (2018). https://doi.org/10.1103/PhysRevB.97.024306

(2) Friedrich, H. & Wintgen, D. Interfering resonances and bound states in the continuum. *Phys. Rev. A* **32**, 3231–3242 (1985). https://doi.org/10.1103/PhysRevA.32.3231

About this project

This project was undertaken by undergraduate sophomore students at the Physics Department at IIT Bombay as a part of the course PH 202: Waves, Oscillations and Optics, instructed by Prof. Anshuman Kumar. The work presented in this paper is far from original and is merely a reproduction of the work done in [1].

The calculation parameters have been explicity mentioned in the code blocks as well as in the paper, whenever necessary, and the values have been taken from [1] and [3]. In cases where the values were not clear in these references, the values we have used have been explicitly declared as variables in the code blocks.

The graphs we produced using python may not match the ones present in [1] due to more than one of the following reasons:

- Python has a limited processing capability as opposed to other numerical analysis tools (MATLAB was used by the authors of [1])
- The granularity of data points supported on python is limited
- Certain systems of non-linear equations may not converge on python due to limited solving capability of the mpmath/scipy solvers, so we used finite data points with a certain degree of error(the value of which can be displayed in each code block by adding a few extra commands)

Few specific cases of discrepancies:

- For the BIC graphs, there is a discontinuity of the functions at the boundaries because python plots the graphs as a linear spline
- For the Q-factor graph, we found finite data points and used a cubic spline to interpolate the function, which have resulted in deviations
- In all graphs, we have used *average* values of refractive indices over small ranges instead of having a perfectly point-wise frequency-dependent refractive index. These average values are taken from [3] and the averages have been calculated over very small intervals, to ensure that the error is small.

However, as far as the qualitative physics of the problems is concerned, all graphs convey the requisite information as is desired.

Contribution of each team member is as follows:

1. Nabeel Ahmed *(19B030016)* : Codeblocks 6A, 7A, 7B & 7C and Theory Sections 1 & 5
2. Harshit Agarwal *(190260022)* : Codeblocks 5A & 5B and Theory Section 4
3. Kasi Reddy Sreeman Reddy *(190070029)* : Codeblocks 3, 6B & 6C and Theory Section 3
4. Jai Anil Israni *(190010033)* : Codeblocks 1,2 & 4 and Theory Section 2