Spectral Reflectance and Transmittance

```
"""
@author: Paula Andrea Triana Guerrero
"""
import numpy as np
import matplotlib.pyplot as plt
from PyTMM.refractiveIndex import *

database = "../../../database"
catalogo = RefractiveIndex(database)

def VectorPropagacionX(n,b,w,c):

    kx = np.sqrt((n*w/c)**2 - b**2,dtype = np.complex128)
    return kx

def Angulo (Theta, n0, ni):
    Horizontal = np.sqrt(1 - ((n0/ni)* np.sin(Theta,dtype = np.complex128))**2,
dtype=np.complex128)
    return Horizontal

def Dinamica(pol, n0, ni, Theta):
    if pol is "P":
        Di = np.matrix([[Angulo(Theta,n0,ni), Angulo(Theta,n0,ni)],[ni, -ni]], dtype=np.complex128)
    elif pol is "S":
        Di = np.matrix([[1, 1],[ni * Angulo(Theta,n0,ni), -ni * Angulo(Theta,n0,ni)]],
dtype=np.complex128)
    return Di

def pro(phi):
    Pi = np.matrix([[np.e**(-1j*phi), 0],[0, np.e**(1j*phi)]], dtype=np.complex128)
    return Pi


def reflectancia (NumeroMateriales,Thet, IndiceRefraccion, pol, espesor,l):
    MatrizDinamica = [None]*(NumeroMateriales)
    VectorPropagacion = [None]*NumeroMateriales
    phi = [None]*(NumeroMateriales-2)
    InversaDinamica = [None]*(NumeroMateriales)
    Productoria = np.matrix([[1,0],[0,1]],dtype = np.complex128)
    MatrizPropagacion = [None]*(NumeroMateriales-2)
    A_0 = 1
    B_s = 0
    Theta = (Thet)*np.pi/180
    c= 3*(10**17)
    w = (2*np.pi*c)/l
    k0 = (IndiceRefraccion [0] *2*np.pi)/l
    b = k0* np.sin(Theta)

    #Hallando los vectores de Propagación.
    for medio in range(NumeroMateriales):
```

```
        VectorPropagacion [medio] = VectorPropagacionX(IndiceRefraccion[medio],b,w,c)
    #Definiendo las matrices dinámicas de cada medio.

    for medio in range(NumeroMateriales):
        if pol is "P" and medio==0:
            MatrizDinamica[medio] = np.matrix([[np.cos(Theta,dtype = np.complex128), np.cos(Theta,
dtype = np.complex128)],[IndiceRefraccion[0], -1*IndiceRefraccion [0]]], dtype = np.complex128)
        elif pol is "S" and medio==0:
            MatrizDinamica[medio] = np.matrix([[1,1],[IndiceRefraccion[0] * np.cos(Theta,dtype =
np.complex128), -IndiceRefraccion[0] * np.cos(Theta,dtype = np.complex128)]])
        else:
            MatrizDinamica[medio] = Dinamica(pol, IndiceRefraccion[0], IndiceRefraccion[medio],
Theta)
    #Definiendo las matrices de propagación en cada capa.

    for capa in range (NumeroMateriales-2):
        phi [capa] = (VectorPropagacion[capa+1] * espesor[capa+1])
        MatrizPropagacion [capa] = pro(phi[capa])
    #Hallando las inversas de las matrices dinámicas.
    for medio in range (NumeroMateriales):

        InversaDinamica[medio] = np.linalg.inv(MatrizDinamica[medio])
        #Multiplicatoria total
    for medio in range (1, NumeroMateriales-1):
        Productoria = Productoria*(MatrizDinamica[medio]*MatrizPropagacion[medio-
1]*InversaDinamica[medio])
    #Matriz de Transferecia del sistema
    MT = InversaDinamica[0] * Productoria * MatrizDinamica[-1]

    TMT = np.asarray(MT)
    #Extracción de elementos para cálculo de B_0 Y A_s
    B_0 = (TMT[1][0])/(TMT[0][0])
    R = np.abs(B_0**2)
    A_s = 1/(TMT[0][0])
    T = (VectorPropagacion[-1]/VectorPropagacion[0])*(np.abs(A_s**2))
    return (R, T)


#Pidiendo datos al usuario
NumeroMateriales = int (raw_input("Introduzca el número de materiales: "))
#lo = int (raw_input("Introduzca la Longitug de Onda (nm): "))
Thet = int (raw_input("Introduzca Ángulo de Incidencia (°): "))
pol = raw_input("Introduzca Tipo de Polarización(P o S): ")

#Definiendo el tamaño de cada matriz
materiales = []
Refractive = [None]*NumeroMateriales
Extinction = [None]*NumeroMateriales
Extinction [0] = 0*1j
IndiceRefraccion = [None]*NumeroMateriales
```

```
MultiDinamica = 1
MultiPropagacion = 1
espesor = [None]*NumeroMateriales
espesor[0] = float ("inf")
espesor[-1] = float ("inf")

MatrizSistema = [None]*(NumeroMateriales-2)
Parcial = np.matrix([[1,0],[0,1]],dtype = np.complex128)
InversaSistema = [None]*(NumeroMateriales-2)
A_i = [None]*(NumeroMateriales-2)
B_i = [None]*(NumeroMateriales-2)
TotalE = []

#Solicitando datos del sistema al usuario
for medio in range(NumeroMateriales):
    materiales.append (raw_input("Introduzca el material (shelf, Book, Page):").split(", "))
    if espesor [medio] != float ("inf"):
        espesor [medio] = (int(raw_input("Introduzca el espesor: ")))


reflactancias = []
transmitancias =[]
Longitudes = np.linspace(370, 1100, 2000)

for item in Longitudes:
    for medio in range(NumeroMateriales):
        Material_Catalogo = catalogo.getMaterial(materiales [medio][0], materiales [medio]
[1],materiales [medio][2])
        Refractive [medio] = Material_Catalogo.getRefractiveIndex(item)
        try:
            Extinction [medio] = Material_Catalogo.getExtinctionCoefficient(item)*1j
            Extinction [0] = 0*1j
            IndiceRefraccion[medio] = np.sum([Refractive[medio],Extinction[medio]], axis =0)
        except NoExtinctionCoefficient:
            Extinction[medio] = -0*1j
            IndiceRefraccion[medio] = np.sum([Refractive[medio],Extinction[medio]], axis =0)

    reflactancia_parcial, transmitancia_parcial = reflectancia (NumeroMateriales,Thet,
IndiceRefraccion, pol, espesor,item)

    transmitancias.append(transmitancia_parcial)
    reflactancias.append(reflactancia_parcial)

#Definiendo constantes
plt.plot (Longitudes,reflactancias)
plt.plot (Longitudes,transmitancias)
plt.title("Reflectance and Transmittance")
plt.ylabel("(a.u.)")
plt.xlabel("Wavelenght (nm)")
plt.legend(['R','T'], loc='best')
plt.savefig("RTLamdaBK7AuAirP.png")
plt.figure()
```

Angular Reflectance and Transmittance

```
"""
@author: Paula Andrea Triana Guerrero
"""
import numpy as np
import matplotlib.pyplot as plt
from PyTMM.refractiveIndex import *

database = "../../../database"
catalogo = RefractiveIndex(database)

def VectorPropagacionX(n,b):

    kx = np.sqrt((n*w/c)**2 - b**2,dtype = np.complex128)
    return kx

def Angulo (Theta, n0, ni):
    Horizontal = np.sqrt(1 - ((n0/ni)* np.sin(Theta,dtype = np.complex128))**2,
dtype=np.complex128)
    return Horizontal

def Dinamica(pol, n0, ni, Theta):
    if pol is "P":
        Di = np.matrix([[Angulo(Theta,n0,ni), Angulo(Theta,n0,ni)],[ni, -ni]], dtype=np.complex128)
    elif pol is "S":
        Di = np.matrix([[1, 1],[ni * Angulo(Theta,n0,ni), -ni * Angulo(Theta,n0,ni)]],
dtype=np.complex128)
    return Di

def pro(phi):
    Pi = np.matrix([[np.e**(-1j*phi), 0],[0, np.e**(1j*phi)]], dtype=np.complex128)
    return Pi
#Pidiendo datos al usuario
NumeroMateriales = int (raw_input("Introduzca el número de materiales: "))
l = int (raw_input("Introduzca la Longitug de Onda (nm): "))
#Thet = int (raw_input("Introduzca Ángulo de Incidencia (°): "))
pol = raw_input("Introduzca Tipo de Polarización(P o S): ")

#Definiendo el tamaño de cada matriz
materiales = []
Refractive = [None]*NumeroMateriales
Extinction = [None]*NumeroMateriales
Extinction [0] = 0*1j
IndiceRefraccion = [None]*NumeroMateriales
c= 3*(10**17)
w = (2*np.pi*c)/l
```

```
MultiDinamica = 1
MultiPropagacion = 1
espesor = [None]*NumeroMateriales
espesor[0] = float ("inf")
espesor[-1] = float ("inf")

MatrizSistema = [None]*(NumeroMateriales-2)
Parcial = np.matrix([[1,0],[0,1]],dtype = np.complex128)
InversaSistema = [None]*(NumeroMateriales-2)
A_i = [None]*(NumeroMateriales-2)
B_i = [None]*(NumeroMateriales-2)
TotalE = []

#Solicitando datos del sistema al usuario
for medio in range(NumeroMateriales):
    materiales.append (raw_input("Introduzca el material (shelf, Book, Page):").split(", "))
    if espesor [medio] != float ("inf"):
        espesor [medio] = (int(raw_input("Introduzca el espesor: ")))
    Material_Catalogo = catalogo.getMaterial(materiales [medio][0], materiales [medio]
[1],materiales [medio][2])
    Refractive [medio] = Material_Catalogo.getRefractiveIndex(l)
    try:
        Extinction [medio] = Material_Catalogo.getExtinctionCoefficient(l)*1j
        Extinction [0] = 0*1j
        IndiceRefraccion[medio] = np.sum([Refractive[medio],Extinction[medio]], axis =0)
    except NoExtinctionCoefficient:
        Extinction[medio] = -0*1j
        IndiceRefraccion[medio] = np.sum([Refractive[medio],Extinction[medio]], axis =0)
#Definiendo constantes


def reflectancia (NumeroMateriales,Thet, IndiceRefraccion, pol, espesor):
    MatrizDinamica = [None]*(NumeroMateriales)
    VectorPropagacion = [None]*NumeroMateriales
    phi = [None]*(NumeroMateriales-2)
    InversaDinamica = [None]*(NumeroMateriales)
    Productoria = np.matrix([[1,0],[0,1]],dtype = np.complex128)
    MatrizPropagacion = [None]*(NumeroMateriales-2)
    A_0 = 1
    B_s = 0
    Theta = (Thet)*np.pi/180
    c= 3*(10**17)
    w = (2*np.pi*c)/l
    k0 = (IndiceRefraccion [0] *2*np.pi)/l
    b = k0* np.sin(Theta)

    #Hallando los vectores de Propagación.
    for medio in range(NumeroMateriales):
        VectorPropagacion [medio] = VectorPropagacionX(IndiceRefraccion[medio],b)
    #Definiendo las matrices dinámicas de cada medio.
```

```
    for medio in range(NumeroMateriales):
        if pol is "P" and medio==0:
            MatrizDinamica[medio] = np.matrix([[np.cos(Theta,dtype = np.complex128), np.cos(Theta,
dtype = np.complex128)],[IndiceRefraccion[0], -1*IndiceRefraccion [0]]], dtype = np.complex128)
        elif pol is "S" and medio==0:
            MatrizDinamica[medio] = np.matrix([[1,1],[IndiceRefraccion[0] * np.cos(Theta,dtype =
np.complex128), -IndiceRefraccion[0] * np.cos(Theta,dtype = np.complex128)]])
        else:
            MatrizDinamica[medio] = Dinamica(pol, IndiceRefraccion[0], IndiceRefraccion[medio],
Theta)
    #Definiendo las matrices de propagación en cada capa.

    for capa in range (NumeroMateriales-2):
        phi [capa] = (VectorPropagacion[capa+1] * espesor[capa+1])
        MatrizPropagacion [capa] = pro(phi[capa])
    #Hallando las inversas de las matrices dinámicas.
    for medio in range (NumeroMateriales):

        InversaDinamica[medio] = np.linalg.inv(MatrizDinamica[medio])
        #Multiplicatoria total
    for medio in range (1, NumeroMateriales-1):
        Productoria = Productoria*(MatrizDinamica[medio]*MatrizPropagacion[medio-
1]*InversaDinamica[medio])
    #Matriz de Transferecia del sistema
    MT = InversaDinamica[0] * Productoria * MatrizDinamica[-1]

    TMT = np.asarray(MT)
    #Extracción de elementos para cálculo de B_0 Y A_s
    B_0 = (TMT[1][0])/(TMT[0][0])
    R = np.abs(B_0**2)
    A_s = 1/(TMT[0][0])
    T = (VectorPropagacion[-1]/VectorPropagacion[0])*(np.abs(A_s**2))
    return (R, T)

reflactancias = []
transmitancias =[]
Angles = np.linspace(0, 89.99, 1000)
for item in Angles:
    reflactancia_parcial, transmitancia_parcial = reflectancia (NumeroMateriales,item,
IndiceRefraccion, pol, espesor)
    transmitancias.append(transmitancia_parcial)
    reflactancias.append(reflactancia_parcial)

plt.plot (Angles,reflactancias)
plt.plot (Angles,transmitancias)
plt.title("Reflectance and Transmittance")
plt.ylabel("(a.u.)")
plt.xlabel("$\Theta (^{\circ}$)")
plt.legend(['R','T'], loc='best')
plt.savefig("RTH2OAir.png")
plt.show()
```

Electric Field Profile

```
"""
@author: Paula Andrea Triana Guerrero
"""

import numpy as np
import matplotlib.pyplot as plt
from PyTMM.refractiveIndex import *

database = "../../../database"
catalogo = RefractiveIndex(database)

def VectorPropagacionX(n):
    kx = np.sqrt((n*w/c)**2 - b**2,dtype = np.complex128)
    return kx

def Angulo (Theta, n0, ni):
            Horizontal = np.sqrt(1 - ((n0/ni)* np.sin(Theta,dtype = np.complex128))**2,
dtype=np.complex128)
    return Horizontal

def Dinamica(pol, n0, ni, Theta):
    if pol is "P":
        Di = np.matrix([[Angulo(Theta,n0,ni), Angulo(Theta,n0,ni)],[ni, -ni]], dtype=np.complex128)
    elif pol is "S":
                Di = np.matrix([[1, 1],[ni * Angulo(Theta,n0,ni), -ni * Angulo(Theta,n0,ni)]],
dtype=np.complex128)
    return Di

def pro(phi):
    Pi = np.matrix([[np.e**(-1j*phi), 0],[0, np.e**(1j*phi)]], dtype=np.complex128)
    return Pi
#Pidiendo datos al usuario
NumeroMateriales = int (raw_input("Introduzca el número de materiales: "))
l = int (raw_input("Introduzca la Longitug de Onda (nm): "))
Thet = int (raw_input("Introduzca Ángulo de Incidencia (°): "))
pol = raw_input("Introduzca Tipo de Polarización(P o S): ")

#Definiendo el tamaño de cada matriz
materiales = []
Refractive = [None]*NumeroMateriales
Extinction = [None]*NumeroMateriales
Extinction [0] = 0*1j
IndiceRefraccion = [None]*NumeroMateriales
VectorPropagacion = [None]*NumeroMateriales
MatrizDinamica = [None]*(NumeroMateriales)
phi = [None]*(NumeroMateriales-2)
MatrizPropagacion = [None]*(NumeroMateriales-2)
InversaDinamica = [None]*(NumeroMateriales)
MultiDinamica = 1
MultiPropagacion = 1
```

```
espesor = [None]*NumeroMateriales
espesor[0] = float ("inf")
espesor[-1] = float ("inf")
Productoria = np.matrix([[1,0],[0,1]],dtype = np.complex128)
MatrizSistema = [None]*(NumeroMateriales-2)
Parcial = np.matrix([[1,0],[0,1]],dtype = np.complex128)
InversaSistema = [None]*(NumeroMateriales-2)
A_i = [None]*(NumeroMateriales-2)
B_i = [None]*(NumeroMateriales-2)
TotalE = []

#Solicitando datos del sistema al usuario
for medio in range(NumeroMateriales):
   materiales.append (raw_input("Introduzca el material (shelf, Book, Page):").split(", "))
   if espesor [medio] != float ("inf"):
      espesor [medio] = (int(raw_input("Introduzca el espesor: ")))
         Material_Catalogo = catalogo.getMaterial(materiales [medio][0], materiales [medio]
[1],materiales [medio][2])
   Refractive [medio] = Material_Catalogo.getRefractiveIndex(l)
   try:
      Extinction [medio] = Material_Catalogo.getExtinctionCoefficient(l)*1j
      Extinction[0] = 0*1j
      IndiceRefraccion[medio] = np.sum([Refractive[medio],Extinction[medio]], axis =0)
   except NoExtinctionCoefficient:
      Extinction[medio] = 0*1j
      IndiceRefraccion[medio] = np.sum([Refractive[medio],Extinction[medio]], axis =0)
#Definiendo constantes
A_0 = 1
B_s = 0
Theta = (Thet)*np.pi/180
c= 3*(10**17)
w = (2*np.pi*c)/l
k0 = (IndiceRefraccion [0] *2*np.pi)/l
b = k0* np.sin(Theta)

#Hallando los vectores de Propagación.
for medio in range(NumeroMateriales):
   VectorPropagacion [medio] = VectorPropagacionX(IndiceRefraccion[medio])
#Definiendo las matrices dinámicas de cada medio.
for medio in range(NumeroMateriales):
   if pol is "P" and medio==0:
        MatrizDinamica[medio] = np.matrix([[np.cos(Theta,dtype = np.complex128), np.cos(Theta,
dtype = np.complex128)],[IndiceRefraccion[0], -1*IndiceRefraccion [0]]], dtype = np.complex128)
   elif pol is "S" and medio==0:
         MatrizDinamica[medio] = np.matrix([[1,1],[IndiceRefraccion[0] * np.cos(Theta,dtype =
np.complex128), -IndiceRefraccion[0] * np.cos(Theta,dtype = np.complex128)]])
   else:
         MatrizDinamica[medio] = Dinamica(pol, IndiceRefraccion[0], IndiceRefraccion[medio],
Theta)
#Definiendo las matrices de propagación en cada capa.
for capa in range (NumeroMateriales-2):
   phi [capa] = (VectorPropagacion[capa+1] * espesor[capa+1])
```

```
   MatrizPropagacion [capa] = pro(phi[capa])
#Hallando las inversas de las matrices dinámicas.
for medio in range (NumeroMateriales):
   InversaDinamica[medio] = np.linalg.inv(MatrizDinamica[medio])
   #Multiplicatoria total
for medio in range (1, NumeroMateriales-1):
                Productoria = Productoria*(MatrizDinamica[medio]*MatrizPropagacion[medio-
1]*InversaDinamica[medio])
#Matriz de Transferecia del sistema
MT = InversaDinamica[0] * Productoria * MatrizDinamica[-1]

TMT = np.asarray(MT)
#Extracción de elementos para cálculo de B_0 Y A_s
B_0 = (TMT[1][0])/(TMT[0][0])
A_s = 1/(TMT[0][0])

#Matriz de Transferencia de las capas
for capa in range(NumeroMateriales-2):
   if capa==0:
      MatrizSistema[capa] = InversaDinamica[capa]*MatrizDinamica[capa+1]
      InversaSistema [capa] = np.asarray (np.linalg.inv(MatrizSistema[capa]))
   else:
                        Parcial = Parcial * (MatrizDinamica[capa]*MatrizPropagacion[capa-
1]*InversaDinamica[capa])
      MatrizSistema[capa] = InversaDinamica[0]*Parcial*MatrizDinamica[capa+1]
      InversaSistema [capa] = np.asarray (np.linalg.inv(MatrizSistema[capa]))

#Amplitudes en capas
for capa in range (NumeroMateriales-2):
      A_i [capa] = InversaSistema[capa][0][0] * A_0 + (InversaSistema[capa][0][1] * TMT[1]
[0])/TMT[0][0]
      B_i [capa] = InversaSistema[capa][1][0] * A_0 + (InversaSistema[capa][1][1] * TMT[1]
[0])/TMT[0][0]

def CampoElectrico(z, A_i, B_i, kx):
   E = []
   for item in np.linspace(0,z,z*100):
      E.append (np.abs(A_i * np.e**(1j * kx *item) + B_i *  np.e**(-1j * kx *item)))
   return E

grosor = np.sum(espesor[1:-1])

for capa in range (NumeroMateriales-2):
            TotalE = TotalE + (CampoElectrico(espesor[capa+1],A_i[capa], B_i[capa],
VectorPropagacion[capa+1]))

plt.plot (np.linspace(0,grosor,grosor*100), TotalE)
for capa in range(len(espesor)):
   plt.axvline(np.sum(espesor[1:capa]), color='r', ls="dotted")
#    texto1 = text(2, 0.6, r'$\frac{\sin(x)}{x}$', fontsize=20)
plt.title("Electric Field Module")
plt.ylabel("|E(x)|")
```

```
plt.xlabel("x (nm)")
plt.savefig("EF7LayersP.png")
plt.show()
```