```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 15 15:37:51 2020

@author: brage
"""
########### SLAB WAVEGUIDE -- TM / TE ###########
# A waveguide simplified as a thin-film model   #
# coord.system: x:up; y:out-of-page; z:right    #
#                                               #
#   n_2                                         #
#  -------------------------------              #
#   n_1      wave--->       b                   #
#  -------------------------------              #
#   n_2                                         #
#                                               #
#################################################
#
# where n_1 > n_2 to ensure total internal reflection
# b = height of n_1 region
#
# Derivation of TE Mode Dispersion Relation and E_y can be found in lecture notes T8C.pdf
# Derivation of TM Mode Dispersion Relation and H_y can be found in assignment submission Optikk_GK_Assignment_4.pdf

import numpy as np
import matplotlib.pyplot as plt
import sys

def intersectionsAmount(lhs, rhs, rhs_modes):
    # returns the amount of intersections between rhs and lhs(for a given b)
    # rhs_modes is decided by user in system constants
    if lhs[0] > rhs[0] + (rhs_modes - 1) * np.pi:
        return rhs_modes
    else:
        return intersectionsAmount(lhs, rhs, rhs_modes - 1)

P = 1000                                # data points

# SYSTEM CONSTANTS
pol = "TE"                              # polarization, either "TE" or "TM"
wl = 1.55                               # wavelength, micrometers
n_1 = 1.7                               # inner material refractive index
n_2 = 1.4                               # outer material refractive index
rhs_modes = 4                               # number of modes m to plot of right-hand-side mode-relation eq

# system variables
bwl = np.arange(.25, rhs_modes / 2, .25)        # b / wl
N = np.linspace(n_2 * 1.0001, n_1 * 0.999, P)   # Effective refractive index; avoid divide by zero
beta = 2 * np.pi * N / wl                       # propagation constant

if pol != "TE" and pol != "TM":
    sys.exit("Make sure parameter pol is _exactly_ equal to either TE or TM!")

############## MODE DISPERSION ##############
MD_lhs = []                             # left-hand-side of mode dispersion equation
for i in bwl:                           # calculate for various b/wl
    MD_lhs.append(np.sqrt(n_1**2 - N**2) * 2 * np.pi * i)
if pol == "TM":
    MD_rhs = 2 * np.arctan( (n_1 / n_2)**2 * np.sqrt((N**2 - n_2**2) / (n_1**2 - N**2))) # right hand side, valid for any multiplum of pi
elif pol == "TE":
    MD_rhs = 2 * np.arctan(np.sqrt((N**2 - n_2**2) / (n_1**2 - N**2)))

# Figure
plt.figure()

for i in range(rhs_modes):
    plt.plot(N, MD_rhs + i * np.pi, '#1f77b4')
for i in range(len(bwl)):
    plt.plot(N, MD_lhs[i], '#ff7f0e')
```

```
plt.grid(True)
plt.xlabel(r"$N$")
plt.ylabel(r"phase [rad]")
plt.title(r"Mode dispersion (" + pol + ") for $n_1=$" + str(n_1) + " $n_2=$"+ str(n_2) + " $\lambda=$"+ str(wl))
plt.show()

# User choose for which "b" to find modes (look at graph)
print("Note: mode dispersion MD_lhs is found for each value of b/wl, in total", len(MD_lhs), "cases.")
lhs_b = int(input("Choose which MD_lhs (red curves) to find intersect points (1 - " + str(len(MD_lhs)) + "): ")) - 1

# For each intersection, find their indices in MD_lhs&MD_rhs => can find N for each mode
numberOfModes = intersectionsAmount(MD_lhs[lhs_b], MD_rhs, rhs_modes)
idx = []
for i in range(numberOfModes):
    idx.append(np.argwhere(np.diff(np.sign(MD_lhs[lhs_b] - (MD_rhs + np.pi*i)))).flatten())
N_mode = []
for i in idx:
    N_mode.append(float(N[i]))
b = float(bwl[lhs_b] * wl)

print("\nb = ", b, "um")
print("Number of modes:", numberOfModes)
print("N =", N_mode)



############### TRANSVERSAL FIELD COMPONENT ###############
# Henceforth using the N and b found from the mode calculation
N2 = np.power(N_mode, 2)
K = np.sqrt(n_1**2 - N2) * 2 * np.pi / wl         # diff. eq. solution wave prop. constants
gamma = np.sqrt(N2 - n_2**2) * 2 * np.pi / wl     # diff. eq. solution wave prop. constants
A = 1                                             # diff. eq. solution constant; arbitrarily chosen for a fitting amplitude
if pol == "TM":
    C = gamma * A * n_1**2 / (K * n_2**2)                          # diff. eq. solution constant
    D = A * (K*n_2**2*np.sin(K*b) / (gamma*n_1**2)) - np.cos(K * b) # diff. eq. solution constant
elif pol == "TE":
    C = gamma * A / K
    D = A * (K * np.sin(K * b) / gamma - np.cos(K * b))

print("\nDifferential equation solution constants:")
print("K =", K, "\ngamma =", gamma, "\nA = B =", A, "; C =", C, "; D =", D)

# For each mode, solve fieldTransv and store in fieldTransv[mode]
x = np.linspace(-2*b, 3*b, P)                        # x position
fieldTransv = np.zeros((numberOfModes, len(x)))      # transversal field, E_y (if TE) or H_y (if TM)
for mode in range(numberOfModes):
    index = 0
    for X in x:
        if X < 0:
            fieldTransv[mode][index] = A*np.exp(gamma[mode]*X)
        elif X >= 0 and X <= b:
            fieldTransv[mode][index] = A*np.cos(K[mode]*X) + C[mode]*np.sin(K[mode]*X)
        elif X > b:
            fieldTransv[mode][index] = D[mode]*np.exp(-gamma[mode]*(X - b))
        else:
            print("Invalid x value when attempting to calculate fieldTransv: x =", i)
            sys.exit("Exiting program.")
        index = index + 1

# Normalize
fieldTransv = fieldTransv / np.max(fieldTransv)

# Figure
fieldlimit = [-np.max(fieldTransv[0]), np.max(fieldTransv[0])]
plt.figure()
for mode in range(numberOfModes):
    plt.plot(fieldTransv[mode], x, label="m = " + str(mode))
plt.plot(fieldlimit, [0, 0], 'k', linewidth=1.5)
plt.plot(fieldlimit, [b, b], 'k', linewidth=1.5)
plt.grid(True)
```

```
plt.xlim(fieldlimit)
if pol == "TM":
    plt.xlabel(r"$H_y$ [a.u]")
elif pol == "TE":
    plt.xlabel(r"$E_y$ [a.u]")
plt.ylabel(r"$x$ [$\mu$m]")
plt.title(r"b = " + str(b) + "\nN = " + str(np.around(N_mode, 4)))
plt.legend()
plt.show()
```