"""

Created by: Sebastian Vergara Palacio,

Latest revision: 2/05/2021

"""

import pandas as pd

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

from scipy import signal

from sklearn import preprocessing

from scipy.fft import fft, fftfreq

def smoothTriangle(data, degree, dropVals=False):

triangle=np.array(list(range(degree)) + [degree] + list(range(degree)[::-1])) + 1

smoothed=[]

for i in range(degree, len(data) - degree \* 2):

point=data[i:i + len(triangle)] \* triangle

smoothed.append(sum(point)/sum(triangle))

if dropVals:

return smoothed

smoothed=[smoothed[0]]\*int(degree + degree/2) + smoothed

while len(smoothed) < len(data):

smoothed.append(smoothed[-1])

return smoothed

class TFBG\_proc(object):

""" Class which reads the sensed data from the TFBG sensor

Arguments:

path\_data: folder direction to data ubication. csv file with a specific format: separation:",". header=0. decimal:".".

"""

def \_\_init\_\_(self, path\_data):

self.data= pd.read\_csv(path\_data,sep=',',header=0,decimal=".",

names=["Tr A(WL)","Tr A(LEVEL)[LINEAR]","Tr B(WL)",

"Tr B(LEVEL)[LINEAR]","Tr C(WL)","Tr C(LEVEL)[LINEAR]",

"Tr D(WL)","Tr D(LEVEL)[LINEAR]","Tr E(WL)","Tr E(LEVEL)[LINEAR]",

"Tr F(WL)","Tr F(LEVEL)[LINEAR]","Tr G(WL)","Tr G(LEVEL)[LINEAR]"])

self.data.set\_index(self.data['Tr A(WL)'].values,inplace=True)

self.norm1=None # Norm1 calculated with some reference signal

self.norm2=None # Norm2 calculated with some reference signal after making a rolling mean for both signals.

self.MD=None

self.CMD\_short=None #CMD calculated from short wavelengths

self.CMD\_long=None #CMD calculated from long wavelengths

self.cut\_off\_wvl=None #cut-off wavelength

self.yyft\_pd=None # fft computed

self.xft=None # x fft computed components

self.rolling\_mean=None

self.chknorm1=0 #to check if norm1 have been computed

self.chknorm2=0 #to check if norm2 have been computed

self.rows=np.shape(self.data)[0] #number of rows in the dataframe

self.Tr\_fltr=None #transmission with triangle smooth

def norm1\_calc(self,name):

"""

name: name of the column to be used as signal reference

"""

norm=pd.DataFrame([self.data[colum]/self.data[name].values for colum in self.data.columns if '[LINEAR]' in colum])

norm=norm.T

norm.set\_index(self.data['Tr A(WL)'].values,inplace=True)

self.norm1= norm

self.chknorm1=1 #norm1 computed

def norm2\_calc(self,name,wd=3):

"""

name: name of the column to be used as signal reference

wd= windows for the rolling mean

"""

# rolling\_windows = df.rolling(2, min\_periods=1)

# rolling\_mean = rolling\_windows.mean()

self.rolling\_mean= self.data.rolling(wd).mean().iloc[wd-1:] #compute the rolling mean

aux=pd.DataFrame(np.array(np.repeat([self.rolling\_mean.iloc[0].values],wd-1,axis=0)),

columns=self.rolling\_mean.columns,

index=self.data.index[0:wd-1].values) # repeat the first value the windows minus 1 size

rollin\_mean\_aument=pd.concat([aux, self.rolling\_mean])# to concat the first wd rows with the first row

norm\_wt\_rllmean=pd.DataFrame([self.data[colum].loc[:]/rollin\_mean\_aument[name].loc[:].values

for colum in self.data.columns if '[LINEAR]' in colum]) #compute the norm with the mean

norm\_wt\_rllmean= norm\_wt\_rllmean.T

norm\_wt\_rllmean.set\_index(self.data['Tr A(WL)'].values,inplace=True)

self.norm2=norm\_wt\_rllmean

self.chknorm2=1 #norm2 computed

def MD\_compute(self,windows=7,norm=1,lim\_low=0,lim\_up=-1):

"""

windows: windows to make the triangle smooth

norm: to define with which norm to compute the MD. norm=1:norm1, norm=2:norm2

lim\_low and lim\_up: lower and upper wavelenght to perform the MD calculation. These limits must be consistent

with future or additional computations. Mut be the logic positions

"""

if lim\_up==-1: #in case lim\_up is not indicated

lim\_up=self.rows

if norm==1:

if self.chknorm1==0:

return(print('Norm1 not calculated yet, please use .norm1\_calc'))

else:

N= np.shape(self.norm1)[0]

norm=self.norm1.copy(deep=True)

else:

if self.chknorm2==0:

return(print('Norm2 not calculated yet, please use .norm2\_calc'))

else:

N= np.shape(self.norm2)[0]

norm=self.norm2.copy(deep=True)

MD\_vec=[]

MD=pd.DataFrame()

Tr\_fltr=pd.DataFrame()

for column in norm.columns:

Tr\_fltr[column]= smoothTriangle(norm[column].iloc[lim\_low:lim\_up].values,degree=windows)

MD\_vec.append( (np.sum(np.abs(norm[column].iloc[lim\_low:lim\_up].values - Tr\_fltr[column].values)))/N )

self.Tr\_fltr= Tr\_fltr

MD=pd.DataFrame(MD\_vec, index=norm.columns)

MD=MD.T

self.MD=MD

def CMD\_compute(self,norm\_pck=1,lim\_low=0,lim\_up=-1,direction=1):

"""

norm: to define with which norm to compute the MD. norm=1:norm1, norm=2:norm2

lim\_low and lim\_up: lower and upper wavelenght to perform the MD calculation. These limits must be consistent

with future or additional computations. Mut be the logic positions

direction: if CMD is calculated from short or long wavelengths (1 or -1)

"""

if lim\_up==-1: #in case lim\_up is not indicated

lim\_up=self.rows

if norm\_pck==1:

if self.chknorm1==0:

return(print('Norm1 not calculated yet, please use .norm1\_calc'))

else:

N= np.shape(self.norm1)[0]

norm=self.norm1.copy(deep=True)

else:

if self.chknorm2==0:

return(print('Norm2 not calculated yet, please use .norm2\_calc'))

else:

N= np.shape(self.norm2)[0]

norm=self.norm2.copy(deep=True)

abs\_dif=np.abs(norm.iloc[lim\_low:lim\_up].values - self.Tr\_fltr)

k\_vec=np.arange(1,abs\_dif.shape[0]+1,1)

CMD=pd.DataFrame()

for column in norm.columns:

CMD[column]=(np.cumsum( abs\_dif[column].iloc[::direction].values))/k\_vec

CMD=CMD.iloc[::direction]

CMD.set\_index(self.data.index.values[lim\_low:lim\_up],inplace=True)

x\_try = CMD.values #returns a numpy array

min\_max\_scaler = preprocessing.MinMaxScaler()

x\_scaled = min\_max\_scaler.fit\_transform(x\_try)

df = pd.DataFrame(x\_scaled,columns=norm.columns,index=self.data.index.values[lim\_low:lim\_up])

if direction==1:

self.CMD\_short= df

else:

self.CMD\_long= df

def cutoff\_compute(self,CMD,name,norm\_pck=1,lim\_low=0,lim\_up=-1,ref\_line= 0.8):

"""

CMD: the self.CMD\_short or self.CMD\_long wavelength

norm: to define with which norm to compute the MD. norm=1:norm1, norm=2:norm2

lim\_low and lim\_up: lower and upper wavelenght to perform the MD calculation. These limits must be consistent

with future or additional computations. Mut be the logic positions

direction: if CMD is calculated from short or long wavelengths (1 or -1)

ref\_line: horizontal reference line to compute the cut-off

name: name of the column to be used as signal reference

"""

if lim\_up==-1: #in case lim\_up is not indicated

lim\_up=self.rows

if norm\_pck==1:

if self.chknorm1==0:

return(print('Norm1 not calculated yet, please use .norm1\_calc'))

else:

N= np.shape(self.norm1)[0]

norm=self.norm1.copy(deep=True)

else:

if self.chknorm2==0:

return(print('Norm2 not calculated yet, please use .norm2\_calc'))

else:

N= np.shape(self.norm2)[0]

norm=self.norm2.copy(deep=True)

norm['trace']=np.array(norm.shape[0] \* [ref\_line])

cut\_off\_wvl=pd.DataFrame()

for column in CMD.columns:

#print(CMD[column].values[:] - norm['trace'].iloc[lim\_low:lim\_up].values)

if norm\_pck==1:

if column==name:

None

else:

cut\_off\_wvl[column]=[CMD.iloc[np.argwhere(np.diff(np.sign(CMD[column].values[:] - norm['trace'].iloc[lim\_low:lim\_up].values)))

.flatten()[0]].name]

else:

cut\_off\_wvl[column]=[CMD.iloc[np.argwhere(np.diff(np.sign(CMD[column].values[:]-norm['trace'].iloc[lim\_low:lim\_up].values))).flatten()[0]].name]

self.cut\_off\_wvl=cut\_off\_wvl

def fft\_calc(self,norm\_wt\_rllmean,lim\_low=0,lim\_up=-1):

"""

Computes de fft of the input data

norm\_wt\_rlmean: Dataframe to be calculated

lim\_low and lim\_up: lower and upper wavelenght to perform the MD calculation. These limits must be consistent

with future or additional computations.

"""

if lim\_up==-1: #in case lim\_up is not indicated

lim\_up=self.rows

yft\_vec=[]

yft\_pd=pd.DataFrame()

for i in norm\_wt\_rllmean.columns:

yft\_vec.append(fft(norm\_wt\_rllmean[i].iloc[lim\_low:lim\_up].values))

yft\_pd=pd.DataFrame(yft\_vec,index=norm\_wt\_rllmean.columns)

yft\_pd=yft\_pd.T

self.yyft\_pd= yft\_pd

xft=fftfreq(norm\_wt\_rllmean.iloc[lim\_low:lim\_up].shape[0])

self.xft= xft

def figure\_general(self,x,data\_y,size):

""" Make a n-plot. data\_x and data\_y is a vector containing each vector to be plotted"""

fig, ax = plt.subplots(size,figsize=(15,15))

column=data\_y.columns

if size==1:

ax.plot(x,data\_y[column[0]])

ax.set\_xlabel(r'wavelenght (nm)')

else:

for i in range(size):

ax[i].plot(x,data\_y[column[i]].values,label=column[i])

ax[i].legend()

ax[i].set\_xlabel(r'wavelenght (nm)')

def figure\_norm(self,dataframe,name,n,ylim\_l=0.5,ylim\_up=1):

""" Make a plot. dataframe is the target to be plotted

name:the reference

n: refractive index vector

ylim\_l:plot lower limit

ylim\_up:plot upper limit

"""

fig, ax = plt.subplots(2,figsize=(15,15))

c=0

ax[0].plot(dataframe.index.values,dataframe[name].values,label=name)

ax[0].set\_title('REFERENCE')

ax[0].set\_ylabel(r'Power Spectral Density ($\mu W/ nm$)')

ax[0].legend()

for i in dataframe.columns:

ax[1].plot(dataframe.index.values,norm[i],label=n[c])

c+=1

#ax[1].set\_ylim([0,1])

ax[1].set\_title('Normalization')

ax[1].set\_ylabel(r'Normalized Power Spectral Density')

ax[1].set\_xlabel(r'wavelenght (nm)')

ax[1].set\_ylim([ylim\_l,ylim\_up])

ax[1].legend()

def figure\_CMD(self,df,name,n,low\_wlght=1545,up\_wvlght=1560):

""" Make a plot. df is the target to be plotted

name:the reference

n: refractive index vector

low\_wlght:plot lower limit wavelength

up\_wvlght:plot upper limit wavelength

"""

fig, ax = plt.subplots(1,figsize=(14,7))

for i,column in zip(range(df.shape[1]),df.columns):

if column==name:

None

else:

ax.plot(df[column].loc[low\_wlght:up\_wvlght],label=n[i])

if direction==1:

ax.set\_title('Calculated from the side with shorter wavelengths',fontsize=15)

else:

ax.set\_title('Calculated from the side with longer wavelengths',fontsize=15)

ax.set\_xlabel(r'wavelenght (nm)',fontsize=15)

ax.legend()

ax.set\_ylabel(r'CMD',fontsize=15);