```
# -*- coding: utf-8 -*-
"""
Created on Sat Jan  2 17:30:58 2016

@author: Dom
"""

from math import *
import matplotlib.pyplot as plt
import os
from pylab import*
from scipy.io import wavfile
import numpy as np

path =r'C:\Users\Admin2\Desktop\Present_ISN' # Attention
# si il y a des valeurs numériques derrière \ elles sont lues comme des codes
# de la table utf-8, il faut ajouter la lettre r devant les quotes : raw

# on teste le répertoire courant de travail
retval = os.getcwd() # cwd veut dire current work directory
print ("le répertoire de travail est : %s" % retval)

# On change de répertoire
os.chdir(path) # chdir pour change directory

# On reteste
retval = os.getcwd()

print("Le répertoire de travail a été modifié selon : %s" % retval)

#*****************************************************************************
# Lecture du fichier txt
# *****************************************************************************

def lecture(fichier):
    f = open(fichier) # on ouvre le fichier et on le met dans f
    temps,tensions = [],[] # on prépare deux listes temps et tensions
    for line in f: # on parcourt les lignes du fichier
        if line[0:5] != 'Temps': # si les 5 premiers caract. de la ligne sont
        # différents de "Temps" on continue
            line = line.replace(',','.') # on remplace les , par des .
            t,u = line.split(';') # on met les valeurs séparées par ; dans t et u
            temps.append(float(t)) # on remplit la liste temps avec t que l'on conve
            tensions.append(float(u))
    f.close()
    return temps,tensions

# On attribue les noms aux deux listes contenues dans le fichier
# t,u=(lecture('dia_1.txt'))

#*****************************************************************************
# Lecture du fichier wav
# *****************************************************************************
freq_ech,son = wavfile.read('diap_3.wav')
# La fonction scipy.io.wavfile.read permet de lire des fichiers .wav comme des
# fichiers contenant des entiers codés sur 16 bits (pour les formats 16-bit wavs)
# ou des entiers codés sur 32 bits (pour les formats 32-bit wavs).
# Les fichiers 24-bit ne sont pas supportés.
# Il est possible de tester le type de codage du fichier wav
son.dtype
```

```
#  la réponse renvoyée est 'int16', cela signifie que les valeurs
# de la surpression acoustique sont codées en
# valeurs entières pourvant aller de -2^15 à (2^15)-1.
# Il est possible de convertir le fichier son en tableau contenant des réels
# compris entre -1 et +1
son=real(son/(2.**15)) # conversion en réels compris entre -1 et +1

nbre_ech=son.shape # format et taille du fichier son
# ce fichier présente une voies (mono) et 186874 échantillons
# Le son est échatilloné à 22050 Hz, la durée de l'enregistrement est donc
# de 8,45 s.


delta_t=1/freq_ech # pas temporel d'enregistrement
t=np.arange(len(son))*delta_t # on construit l'axe du temps
u=son[:] # on copie son dans u

# ****************************************************************************
# Tracé du graphes bruts
# ****************************************************************************
plt.plot(t,u)
plt.grid()
plt.title("son d'un diapason")
plt.legend(['p(t)'],loc = "upper right")
plt.show()

# ****************************************************************************
# Tracé du graphes des fichiers limités
# ****************************************************************************
# au début et à la fin ce n'est pas propre, donc on coupe
deb=7000
fin=100000
t=np.arange(fin-deb)*delta_t # on reconstruit le vecteur temps
u=son[deb:fin] # on recopie le nouveau fichier dans u

plt.plot(t,u)
plt.grid()
plt.title("son d'un diapason")
plt.legend(['p(t)'],loc ="upper right")
plt.show()

# ****************************************************************************
# Calcul d'une valeur moyenne et d'un ecart type
# ****************************************************************************
def valeur_moyenne(L):
    s=0
    for i in range(len(L)):
        s=s+L[i]
    Moyenne=s/len(L)
    return Moyenne

def ecarttype(L):
    s=0
    for i in range(len(L)):
        s=s+(L[i]-valeur_moyenne(L))**2 # on somme les écarts au carré
    ecarttype=sqrt(s/len(L)) # on calcule la racine carré de la variance
    return ecarttype

def courbe_centree(L):
    L_moy=valeur_moyenne(L)
```

```
    for i in range(len(L)):
        L[i]=L[i]-L_moy
    return L

# ********************************************************************************
# Annulations
# ********************************************************************************
# Il faut veiller à ce que la courbe soit centrée
u=courbe_centree(u)
def points_d_annulation(L):
    par_zeros = [] # création d'une liste vide en attente
    for i in range(len(L)-1): # tous sauf le dernier
        if L[i]*L[i+1] < 0 : # si les signes sont différents
            par_zeros.append(i) # on ajoute l'index du point d'annulation à la liste
    return par_zeros # on a fabriqué une liste d'index des points d'annulation



# ********************************************************************************
# Maxima successifs
# ********************************************************************************
def maxima_successifs(L,N): # cherche les N max de la liste L
    M=[]
    tps=[]
    for i in range(1,len(L)-1):
        if L[i]>L[i-1] and L[i]>L[i+1]:
            if L[i]>0: # on ne veut que les max positifs
                M.append(L[i]) # on ajoute dans M les valeurs des max
                tps.append(i) # on ajoute dans tps les valeurs des index des max
            if len(M)==N: # quand on a le nombre N on s'arrête
                return M,tps

# ********************************************************************************
# Tracé du graphes des maxima
# ********************************************************************************
N=int((fin-deb)*delta_t/0.0023)
tmax=[i*delta_t for i in maxima_successifs(u,N)[1]]
max_succ=(maxima_successifs(u,N)[0])

plt.plot(tmax,max_succ)
plt.grid()
plt.title("évolution temporelle des maxima")
plt.show()

# ********************************************************************************
# Traitement de la courbe des maxima
# ********************************************************************************
# Traitement par moyenne mobile
def moyenne_mobile(L,Nm):
    Lmoy=[]
    mm=0
    for i in range(Nm, len(L)-Nm):
        mm=sum(L[i:i+Nm])
        Lmoy.append(mm)
    return Lmoy

Nm=20
tt=len(tmax)-Nm
tmax_moy=tmax[Nm:tt]
max_succ_moy=moyenne_mobile(max_succ,Nm)
```

```
# Tracé de la courbe lissée
plt.plot(tmax_moy,max_succ_moy)
plt.grid()
plt.title("évolution des maxima moyennés")
plt.show()
#
# Anamorphose
# la forme attendue est une exponentielle en fonction du temps exp(-mw0(t-t0))
def ana_log(L):
    ana_log=[]
    for i in range (len(L)):
        ana_log.append(log(L[i]))
    return ana_log
#
#
# Test et tracé
ana_log=ana_log(max_succ_moy)
plt.plot(tmax_moy,ana_log)
plt.grid()
plt.title("évolution du log des max")
plt.show()
#
from scipy import linalg #  on importe le module d'algèbre linéaire
from scipy import stats #  on importe le module d'algèbre de stat

ana_log_array=np.asarray(ana_log) # transforme une liste en un vecteur colonne
tmax_moy_array=np.asarray(tmax_moy) # transforme une liste en un vecteur colonne

nb=tmax_moy_array.size # pour avoir la longueur d'un des deux vecteurs colonne
Tps=np.vstack([tmax_moy_array,np.ones(nb)]).T # on fabrique une matrice avec
# la première colonne composée de 1 et la seconde des valeurs du temps (x)
# vstack fabrique une matrice verticale composée des deux mat. données en argument
solu=linalg.lstsq(Tps,ana_log_array) # renvoie toutes les infos de la régression
#
pente=solu[0][0]
ordo_orig=solu[0][1]

# ou bien par polyfit qui est plus "pratique"
model,resid = np.polyfit(tmax_moy_array,ana_log_array,1)[:2]
coeff_corr=1-resid/(ana_log_array.size*ana_log_array.var())

# Tracés de la courbe et de la droite de régréssion
plt.figure()
plt.plot (tmax_moy_array,ana_log_array)
yy=solu[0][0]*tmax_moy_array+solu[0][1]
plt.plot(tmax_moy_array,yy)
plt.annotate('droite de régression', xy=(2.6,0.9), xytext=(3, 1.6),
             arrowprops=dict(facecolor='black',shrink=0.01,width=1),)
plt.title("courbe et modèle linéaire")
plt.xlabel("temps en s")
plt.ylabel("log de l'amplitude")
plt.show()

# ou bien par le module stats qui est encore plus "pratique"
solu_stat=stats.linregress(tmax_moy_array,ana_log_array)
print('la pente =',solu_stat[0])
print("l'ordonnée à l'origine =",solu_stat[1])
print("le coeff. de correlation =",abs(solu_stat[2]))
```

```
# ********************************************************************************
# Liste des décréments logarithmique et calcul de la valeur moyenne
# ********************************************************************************
def decrements(L):
    Liste_delta=[]
    for i in range(len(L)-1):
        Liste_delta.append(log(L[i]/L[i+1]))
    return Liste_delta

# print(decrements(max_succ)) # calcul les décréments des max successifs
# print (valeur_moyenne(decrements(max_succ)))

# ********************************************************************************
# Pseudo-periode par methode des zeros
# ********************************************************************************
def pseudo_periode_par_zeros(L1,L2):
    index_zeros = points_d_annulation(L2) # dans la liste zeros on a les index
    # des points d'annulation de L2
    Liste_T=[] # on prépare une liste vide pour les pseudo-periodes
    for i in range(len(index_zeros)-1):
        t1,t2=index_zeros[i],index_zeros[i+1]
        Liste_T.append(2*(L1[t2]-L1[t1])) # on ajoute les valeurs de pseudo-periode
        # par soustraction des valeurs de L1[i],
        # c'est pour ça qu'il faut L1 !
    return Liste_T

# print("les pseudo-periodes sont", pseudo_periode_par_zeros(t,u))

# ********************************************************************************
# Facteur de qualite et pulsation propre
# ********************************************************************************
def Q_et_w0(temps,tensions):
    delta = valeur_moyenne(decrements(max_succ))
    T = valeur_moyenne(pseudo_periode_par_zeros(t,u))
    Q = sqrt((pi/delta)**2+1/4) # par définition de Q en foncton de delta
    w0= 2*pi/T/sqrt(1-1/(4*Q**2))
    return Q,w0

# ********************************************************************************
# Récapitulatif
# ********************************************************************************
# On attribue les noms aux deux listes contenues dans le fichier
# t,u=(lecture('echantillon1.txt'))

# On détermine et affiche la liste des index des passage par 0 de la tension
zeros= points_d_annulation(u)
# print("les index de la position des passages par 0 est la suivante",zeros)

# On détermine et affiche la valeur moyenne de la pseudopériode
# print("liste des pseudo-périodes", pseudo_periode_par_zeros(t,u))

# On détermine et affiche la valeur moyenne de la pseudo-pulsation
w=2*pi/valeur_moyenne(pseudo_periode_par_zeros(t,u))
print("la valeur moyenne de la pseudo-pulsation est :",round(w,0),"rad/s" )

# On détermine et affiche la liste des n valeurs maximales de la tension
# print("les maxima de tension sont les suivants :", maxima_successifs[0](u,100))

# On détermine et affiche le facteur de qualité et la pseudopulsation
Q,wo=Q_et_w0(t,u)
```

```
print("le facteur de qualité vaut :", round(Q,0))
print("la pseudo-période vaut :",round(valeur_moyenne(pseudo_periode_par_zeros(t,u))
print("l'ecart-type sur la pseudopériode vaut :", round(ecarttype(pseudo_periode_par
print("la fréquence associée vaut : f=",round(1/valeur_moyenne(pseudo_periode_par_ze
```