```
from PIL import Image
import numpy as np
import os
import time as tps
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from mpl_toolkits.mplot3d import Axes3D
from math import *

NumImD=1
NumImF=500
NomDossier='12-phase'
mpixel=7*10**(-2)/300.5
fe=25
M=(240.52+138.25)/2    # Masse de l'ensemble d'un des 2 côtés = Plaque + ressort/2 + visses + Metronome
m2=13.5     # masse contrepoids
l=2.5*10**(-2)   # longueur du pendule (contrepoids => axe de rotation)
k=0.06*10**3  # Constante de raideur ressort

PathB='C:/Users/Alex 1/Documents/Travail/ESPCI/PSE/PSE2A/'+NomDossier
Path=os.path.join(PathB,NomDossier+str(NumImD).zfill(4)+'.png')

im=Image.open(Path)
im=np.array(im)


### INITIALISATION (à ne faire qu'une seule fois)

H=449  #408
L=1280
Cv=600 #640  #On coupe l'image en deux pour les recherches !
Ch=350 #300
mg=[]
md=[]
pg=[]
pd=[]

for i in range(NumImD,NumImF+1):
    A=str(i).zfill(4)
    P=NomDossier+A+'.png'
    Path=os.path.join(PathB,P)
    M1=np.array(Image.open(Path))
    L1=np.where((M1[0:Ch,0:Cv])==255)
    L2=np.where((M1[0:Ch,Cv:L])==255)
    L3=np.where((M1[Ch:H,0:Cv])==255)
    L4=np.where((M1[Ch:H,Cv:L])==255)
    mg.append([np.mean(L1[1]),H-np.mean(L1[0])]) # Metronome gauche
    md.append([np.mean(L2[1])+Cv,H-np.mean(L2[0])]) # Metronome droit
    pg.append([np.mean(L3[1]),H-np.mean(L3[0])-Ch]) # Plaque gauche
    pd.append([np.mean(L4[1])+Cv,H-np.mean(L4[0])-Ch]) # Plaque droite
```

```
### ANALYSE


mg=np.array(mg)
md=np.array(md)
pg=np.array(pg)
pd=np.array(pd)
X=np.array(range(NumImD,NumImF+1))
Xt=X/fe
ax=Axes3D(plt.figure())
ax.plot3D(X,mg[:,0],mg[:,1],'black',linewidth=2.0)
ax.plot3D(X,md[:,0],md[:,1],'black',linewidth=2.0)
ax.plot3D(X,pg[:,0],pg[:,1],'black',linewidth=2.0)
ax.plot3D(X,pd[:,0],pd[:,1],'black',linewidth=2.0)

N=np.size(X)

D=pd[:,0]-pg[:,0] # Elongation totale entre les points g et d des deux plaques
L0=np.mean(D)     # Longueur à vide entre les points g et d des deux plaques

Dp=D-L0     # Elongation du ressort
Thetag=-np.arctan((mg[:,0]-pg[:,0])/(mg[:,1]-pg[:,1]))
Thetad=-np.arctan((md[:,0]-pd[:,0])/(md[:,1]-pd[:,1]))
Dephasage=Thetad-Thetag
Thetaptg=(Thetag[1:N]-Thetag[0:N-1])*fe
Thetaptd=(Thetad[1:N]-Thetad[0:N-1])*fe
vg=(pg[:,0][1:N]-pg[:,0][0:N-1])*mpixel*fe # Vitesse plaque gauche
vmg=(mg[:,0][1:N]-mg[:,0][0:N-1])*mpixel*fe # Vitesse horizontale tige gauche
vd=(pd[:,0][1:N]-pd[:,0][0:N-1])*mpixel*fe # Vitesse plaque droite
vmd=(md[:,0][1:N]-md[:,0][0:N-1])*mpixel*fe # Vitesse horizontale tige droite

Thetad=Thetad[1:N]
Thetag=Thetag[1:N]
Dephasage=Dephasage[1:N]
Dp=Dp[1:N]*mpixel
xg=pg[:,0]
xd=pd[:,0]
xg=xg[1:N]*mpixel
xd=xd[1:N]*mpixel

### CREATION MATRICE GENERALE

N2=N-1  # Nouvelle taille des différents tableaux
X2=X[1:N]
G=[Dp,vg,vd,Thetag,Thetad,Thetaptg,Thetaptd]
G=np.array(G)


# G de la forme suivante :
# [   D1        ,   D2        , ...     0
#     vg1       ,   vg2       , ...     1
```

```
#    vd1        ,   vd2        , ...      2
#    Thetag1    ,   Thetag2    , ...      3
#    Thetad1    ,   Thetad2    , ...      4
#    Thetaptg1  ,   Thetaptg2  , ...      5
#    Thetaptd1  ,   Thetaptd2  , ... ]    6
#
# [   G[:,1]     ,   G[:,2]     , ... ]

m=[0,1,2,3,4,5,6]
m[0]=np.mean(abs(Dp[0:N2-1]-Dp[1:N2]))
m[1]=np.mean(abs(vg[0:N2-1]-vg[1:N2]))
m[2]=np.mean(abs(vd[0:N2-1]-vd[1:N2]))
m[3]=np.mean(abs(Thetag[0:N2-1]-Thetag[1:N2]))
m[4]=np.mean(abs(Thetad[0:N2-1]-Thetad[1:N2]))
m[5]=np.mean(abs(Thetaptg[0:N2-1]-Thetaptg[1:N2]))
m[6]=np.mean(abs(Thetaptd[0:N2-1]-Thetaptd[1:N2]))


def sousnorme(G1,G2,i):
    if (abs(G1[i])/2+abs(G2[i])/2)!=0:
        return(((G1[i]-G2[i])/m[i])**2)
    else:
        return(0)

def norme(G1,G2):

A=sqrt(sousnorme(G1,G2,0)+sousnorme(G1,G2,1)+sousnorme(G1,G2,2)+sousnorme(G1,G2,3)+so
usnorme(G1,G2,4)+sousnorme(G1,G2,5)+sousnorme(G1,G2,6))
    return(A)

Normes=np.zeros([N-1,N-1])
for i in range(N-1):
    for j in range(N-1):
        Normes[i,j]=norme(G[:,i],G[:,j])
        if i==j:
            Normes[i,j]=0

def calculR(Normes):
    R=0
    Corr=np.percentile(Normes,10)
    #print(Corr)
    for i in range(N-1):
        for j in range(N-1):
            if Normes[i,j]<=Corr:
                if (i+1<N-1) and (j+1<N-1):
                    if Normes[i+1,j+1]>Corr:
                        R+=1
    return(R)


indices=np.nonzero(Normes<0.3)
indices=np.array(indices)     # indices[:,i] renvoie les coodonnées du ième couple de point
```

respectant la condition

```
plt.figure()
plt.imshow(Normes)
plt.colorbar()
print(calculR(Normes))

# ATTENTION : ecart d'indices de +2 avec les vraies images. Ex: G[:,25] correspond  à l'image 27 car commence à 0 + troncage de la première image pour pouvoir calculer les dérivées

# Définition du centre de masse ainsi que de la vitesse du point différence, cela sera utilisé pour la figure2
xcm=(xd*M+md[:,0][1:N]*mpixel*m2+xg*M+mg[:,0][1:N]*mpixel*m2)/(2*m2+2*M)
vcm=(vd*M+vmd*m2+vg*M+vmg*m2)/(2*m2+2*M)
vDp=vd-vg
```