```
#!/usr/bin/env python
# -*- coding:Utf-8 -*-


import numpy as np
from math import *
from scipy import interpolate
from uncertainties import *
from uncertainties import unumpy
from uncertainties.umath import *
import matplotlib.pyplot as plt
from numpy.linalg import eig, inv
from tkinter.filedialog import *
import tkinter
import tkinter.filedialog


def fct_analyse(seuil1,
seuil2,
PWD,
name_file_ref,
name_file_signal,
plot_fit=1,
plot_abcd=1,
plot_bandes=1,
normalize = 1,):
## root = Tkinter.Tk()
## name_file_ref = askopenfilename(filetypes=[('Text', '.txt')], initialdir=PWD)
## name_file_signal = askopenfilename(filetypes=[('Text', '.txt')], initialdir=PWD)


omega = 200*np.pi


def fitEllipse(x,y):
x = x[:,np.newaxis]
y = y[:,np.newaxis]
D = np.hstack((x*x, x*y, y*y, x, y, np.ones_like(x)))
S = np.dot(D.T,D)
C = np.zeros([6,6])
C[0,2] = C[2,0] = 2; C[1,1] = -1
E, V = eig(np.dot(inv(S), C))
n = np.argmax(np.abs(E))
a = V[:,n]
return(a)


def ellipse_center(a):
b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a[4]/2, a[5], a[0]
num = b*b-a*c
x0=(c*d-b*f)/num
y0=(a*f-b*d)/num
return(np.array([x0,y0]))
```

```
def ellipse_angle_of_rotation( a ):
b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a[4]/2, a[5], a[0]
return(0.5*np.arctan(2*b/(a-c)))



def ellipse_axis_length( a ):
b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a[4]/2, a[5], a[0]
up = 2*(a*f*f+c*d*d+g*b*b-2*b*d*f-a*c*g)
down1=(b*b-a*c)*( (c-a)*np.sqrt(1+4*b*b/((a-c)*(a-c)))-(c+a))
down2=(b*b-a*c)*( (a-c)*np.sqrt(1+4*b*b/((a-c)*(a-c)))-(c+a))
res1=np.sqrt(up/down1)
res2=np.sqrt(up/down2)
return(np.array([res1, res2]))

def ellipse_angle_of_rotation2( a ):
b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a[4]/2, a[5], a[0]
if b == 0:
if a > c:
return(0)
else:
return(np.pi/2)
else:
if a > c:
return(np.arctan(2*b/(a-c))/2)
else:
return(np.pi/2 + np.arctan(2*b/(a-c))/2)



def moyenne(tableau):
return(sum(tableau,0.0)/len(tableau))

def variance(tableau):
m=moyenne(tableau)
return(moyenne([(x-m)**2 for x in tableau]))

def ecartype(tableau):
return(variance(tableau)**0.5)



def alternance(tableau1,tableau2):
pluslong=-1
if len(tableau1)==len(tableau2)+1:
pluslong=0
for i in range (len(tableau1)-1):
if not(tableau1[i]<tableau2[i] and tableau2[i]<tableau1[i+1]):
return(-1)
```

```
elif len(tableau2)==len(tableau1)+1:
pluslong=1
for i in range (len(tableau2)-1):
if not(tableau2[i]<tableau1[i] and tableau1[i]<tableau2[i+1]):
return(-1)


elif len(tableau2)==len(tableau1):
if tableau1[0]<tableau2[0]:
pluslong=0
for i in range (len(tableau1)-1):
if not(tableau1[i]<tableau2[i] and tableau2[i]<tableau1[i+1]):
return(-1)
else:
pluslong=1
for i in range (len(tableau2)-1):
if not(tableau2[i]<tableau1[i] and tableau1[i]<tableau2[i+1]):
return(-1)
return(pluslong)
plt.clf()
seuil_bande = seuil1
seuil_maximini = seuil2


file_ref = np.loadtxt(name_file_ref)
file_signal = np.loadtxt(name_file_signal)
Iref = file_ref[:,1]
Is = file_signal[:,1]


Is=Is-moyenne(Is)
Iref=Iref-moyenne(Iref)


if normalize == 1:
Is = Is/max(abs(Is))
Iref = Iref/max(abs(Iref))
arc = 2.0
R = np.arange(0,arc*np.pi, 0.01)
a = fitEllipse(Iref,Is)
center = ellipse_center(a)
phi = ellipse_angle_of_rotation(a)
#phi = ellipse_angle_of_rotation2(a)
axes = ellipse_axis_length(a)


a, b = axes
xx = center[0] + a*np.cos(R)*np.cos(phi) - b*np.sin(R)*np.sin(phi)
yy = center[1] + a*np.cos(R)*np.sin(phi) + b*np.sin(R)*np.cos(phi)


################################################################################
####################Calcul des a,b,c,d


indices_Zeros = np.where(np.abs(Is)<seuil_bande*max(Is))
```

```
Zer= Iref[indices_Zeros[0]]


Indices_b = np.where(np.array(Zer>0,dtype = int)>0)
B = Zer[Indices_b]
mu_b = np.mean(B)
sigma_b = np.std(B)



Indices_a = np.where(np.array(Zer<0,dtype = int)>0)
A = Zer[Indices_a]
mu_a = np.mean(A)
sigma_a = np.std(A)


maxi = Is[np.where(Iref == max(Iref))][0]
mini = Is[np.where(Iref == min(Iref))][0]


indices_Maxs = np.where(np.abs(Is-maxi)<seuil_bande*max(Is))
indices_Mins = np.where(np.abs(Is-mini)<seuil_bande*max(Is))
Maxs = Iref[indices_Maxs[0]]
Mins = Iref[indices_Mins[0]]


Indices_c = np.where(np.array(Mins<seuil_maximini*min(Iref),dtype = int)>0)
Indices_d = np.where(np.array(Maxs>seuil_maximini*max(Iref),dtype = int)>0)
C = Mins[Indices_c]
D = Maxs[Indices_d]


mu_c = np.mean(C)
sigma_c = np.std(C)
mu_d = np.mean(D)
sigma_d = np.std(D)


a= ufloat(mu_a, sigma_a)
b= ufloat(mu_b, sigma_b)
c= ufloat(mu_c, sigma_c)
d= ufloat(mu_d, sigma_d)


print("a =", mu_a, "±", sigma_a)
print("b =", mu_b, "±", sigma_b)
print("c =", mu_c, "±", sigma_c)
print("d =", mu_d, "±", sigma_d)


fig = plt.figure()
ax = plt.subplot(111)


ax.scatter(Iref,Is)
```

```
if plot_fit ==1:
ax.plot(xx,yy, color='black', linewidth=3, label = 'fit')
if plot_abcd ==1:
A= np.ones(len(Is))*mu_a
B= np.ones(len(Is))*mu_b
C= np.ones(len(Is))*mu_c
D= np.ones(len(Is))*mu_d
ax.plot(A, Is,label= 'a',color='blue')
ax.plot(B, Is,label= 'b',color='blue')
ax.plot(C, Is, label='c',color='green')
ax.plot(D, Is, label='d',color='green')
if plot_bandes ==1:
S1 = seuil_maximini*min(Iref)*np.ones(len(Is))
S2 = seuil_maximini*max(Iref)*np.ones(len(Is))
ax.plot(S1, Is,
label= 'seuil min',
color = 'red',
linestyle = 'dashed',
linewidth=0.5)


ax.plot(S2, Is,
label= 'seuil max',
color = 'red',
linestyle = 'dashed',
linewidth=0.5)


S3p = (maxi +seuil_bande*max(Is))*np.ones(len(Iref))
S3m = (maxi -seuil_bande*max(Is))*np.ones(len(Iref))


ax.plot(Iref, S3p,
linestyle = 'dashed',
color = 'red',
linewidth=0.5)


ax.plot(Iref, S3m,
linestyle = 'dashed',
color = 'red',
linewidth=0.5)


S4p = (mini +seuil_bande*max(Is))*np.ones(len(Iref))
S4m = (mini -seuil_bande*max(Is))*np.ones(len(Iref))


ax.plot(Iref, S4p,
linestyle = 'dashed',
color = 'red',
linewidth=0.5)


ax.plot(Iref, S4m,
linestyle = 'dashed',
```

```
color = 'red',
linewidth=0.5)


Z=seuil_bande*max(Is)*np.ones(len(Iref))
ax.plot(Iref, Z,
linestyle = 'dashed',
color = 'black',
linewidth=0.5)
ax.plot(Iref, -Z,
linestyle = 'dashed',
color = 'black',
linewidth=0.5)



ax.set_title("Ar as reference")
ax.set_xlabel('Ar Intensity (AU)')
ax.set_ylabel('Eu Intensity (AU)')
ax.grid(True)


# Shrink current axis by 20%
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])


# Put a legend to the right of the current axis
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))


# plt.savefig('ellipse60C.png')
plt.savefig('ellipse_0C_Ar_Eu.jpg')
x = (b-a)/(d-c)
Phi = abs(asin(x))
print("\n Phi =", Phi)
Tau = 1000*(tan(Phi)/omega)
print("\n Tau=", Tau, "ms")
retour= [unumpy.nominal_values((a,b,c,d,Phi,Tau)),unumpy.std_devs((a,b,c,d,Phi,Tau))]
return(retour)
```