```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 25 08:19:07 2019

@author: Publico
"""
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as scp
# Funciones utiles para el curso de Laboratorio de Fisica 1 (fisicos)
# Catedra P. Cobelli - Segundo cuatrimestre de 2017

def leermd(filename):
    result = []
    from pandas import read_csv
    import numpy as np
    # Leer el archivo, separado por tabulaciones, salteando 3 lineas de
header
    #     y empleando la coma como separador decimal
    df = read_csv(filename, sep='\t', skiprows=3, decimal=',')
    # Convertir el dataframe de pandas a un array de numpy
    nparray = df.as_matrix()
    # Remover los posibles nans del archivo
    nparray = nparray[~np.isnan(nparray).any(axis=1)]
    # Asignar las columnas a variables de salida
    for cols in range(nparray.shape[1]):
        out = nparray[:,cols]
        result.append(out)

    return result

datos1= leermd('datos1.csv') #Acá importamos los datos que medimos usando
la función de arriba
dife1=np.diff(datos1[:,1]) #Nos quedamos con la parte de voltaje de los
datos, porque nos interesan las transiciones entre destapado y obturación
indices_picos = scp.find_peaks(dife1) #Pedimos los índices de estos
saltos
tiempos_picos = datos1[indices_picos,0] #Nos quedamos con los tiempos
correspondientes a esos saltos
periodos=np.diff(tiempos_picos) #Calculamos todos los períodos, como
diferencia entre tiempos de saltos consecutivos

#Ajuste lineal
#Supongamos que tenemos medidas dos magnitudes, una en función de la otra
y(x)
y = datos
x = tiempo
#Y supongamos que conocemos los errores asociados a ambas magnitudes
erry = error_datos
errx = error_tiempo
#Entonces queremos plotear y en función de x pero incluyendo las barras
de error
plt.figure() #Creamos una figura en blanco
plt.plot(x,y) #Graficamos y en función de x
plt.errorbar(x,y,erry,errx) #Al gráfico generado inmediatamente arriba,
le agrega las barras de error
plt.show() #Mostramos el gráfico resultante
```

```
# Ahora, queremos ajustar esta función con un polinomio de grado 1 (veáse
una función linea),
# hacer esto se reduce a encontrar cuáles son los coeficientes de dicho
polinomio. Una forma de encontrar
# cuáles son los coeficientes que llevan al polinomio a ajustar de forma
razonable a los datos es utilizando
# el método de cuadrados mínimos. Éste método devuelve dichos
coeficientes. Entonces la idea de ajustar
# datos con un polinomio de un dado grado (1 en este caso), se reduce a
hallar los coeficientes (pendiente
# ordenada al origen en este caso), que mejor ajusten. (mejor en el
sentido que minimicen los cuadrados).
pendiente, ordenada, cov_matrix = np.polyfit(x,y,1,full=False,cov=True)
diagonal = np.diagonal(cov_matrix)
error_pendiente = diagonal[0]
error_ordenanda = diagonal[1]
#Ya tenemos los coeficientes con sus erores
f_lineal = pendiente*x + ordenada*np.ones(len(x))
plt.figure() #Creamos una figura nueva en blanco
plt.plot(x,y) #Graficamos y en función de x
plt.errorbar(x,y,erry,errx) #Al gráfico generado inmediatamente arriba,
le agrega las barras de error
plt.plot(x,f_lineal)
plt.show()
```