```
import matplotlib.pyplot as plt
import numpy as np
import csv
from scipy.optimize import curve_fit
from scipy.constants import mu_0


#
================================================================================
==========
# Fonctions pour lire un fichier csv et en extraire les données
#
================================================================================
==========

def conversion_B(voltage):
    """
    Returns the magnetic field, in Tesla, from the output voltage of the sensor
    10**-4 to convert from G to Tesla
    """
    return ((voltage - 1.75)*2000/3.5)*10**-4


def read_csv(filename):
    """
    Fonction pour lire un fichier csv extrait de l'oscillo,
    temps, voie 1, voie 2
    """

    #initialisation des arrays qui seront renvoyés par la fonction
    data_times = []
    data_U = []
    data_B = []

    with open(filename, newline='') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')

        #parcourt les lignes du fichier
        for row in spamreader:
            try: #on essaie de convertir les données de la ligne en nombres
                data_times.append(np.float64(row[0]))
                data_U.append(np.float64(row[1]))
                data_B.append(np.float64(row[2]))
            except: #si premières lignes (pas des chiffres), erreur que l'on ignore
                pass

    return np.asarray(data_times), np.asarray(data_U), conversion_B(np.asarray(data_B))


filename = 'data/tournevis_800uF_3ème_essai.csv'
```

```
data_times, data_U, data_B = read_csv(filename) #in s, V, G


#indexes des temps entre lesquels on va vouloir fitter la décroissance de U
#à ajuster à la main pour se restreindre au régime exponentielle pour la suite
#♠en regardant le tracé de data_U

index_t1 = 165

index_t2 = len(data_times)-1

#Fonctions pour tracer U et B extraites du csv, si besoin (pour repérer où
#faire le fit par exemple


plt.figure('Tension')
plt.plot(data_times*10**3, data_U)
plt.vlines(data_times[index_t1]*10**3, ymin = np.min(data_U), ymax = np.max(data_U),
        linestyle = 'dashed', lw = 0.5, color = 'red')
plt.vlines(data_times[index_t2]*10**3, ymin = np.min(data_U), ymax = np.max(data_U),
        linestyle = 'dashed', lw = 0.5, color = 'red')
plt.xlabel('Time (ms)')
plt.ylabel('U (V)')
plt.ylim(ymin = 0)


plt.figure('Champ magnétique')
plt.plot(data_times*10**3, data_B)
plt.xlabel('Time (ms)')
plt.ylabel('B (G)')
plt.title('champ magnétique expérimental')

#
==========
# Fonctions pour fitter la décroissance exponentielle de U, en tirer C
#
==========


#on restreint les données à fitter au régime exponentiel définit plus haut
fit_times = data_times[index_t1:index_t2]
fit_U = data_U[index_t1:index_t2]



def exp(t,  A, tau, offset=0, t0 = fit_times[0]):
    """
```

```
    Fitting function of an exponential decay
    Redondance entre A et le décalage t0 mais plus pratique pour fitter
    """
    return A*np.exp(-(t- t0)/tau) + offset


def fit_exp(t_array, data, p0):
    """
    Function to process the fit of the exponential decay of the autocorrelation
    p0 : initial guess for the fitting parameters (t0, A, tau)
    """

    popt, pcov = curve_fit(exp, t_array, data, p0)
    return popt #liste des paramètres trouvés [A, tau]


U0_fit, tau_fit = fit_exp(fit_times, fit_U, p0 = [fit_U[0], 0.0002])

#on décale les temps pour prendre l'origine au début de la décroissance exp
data_times = data_times - fit_times[0]
fit_times = fit_times - fit_times[0]

plt.figure('Tension et son fit')

plt.plot(data_times*10**3, data_U, label = 'mesure')

plt.vlines(data_times[index_t1]*10**3, ymin = np.min(data_U), ymax = np.max(data_U),
        linestyle = 'dashed', lw = 0.5, color = 'red')
plt.vlines(data_times[index_t2]*10**3, ymin = np.min(data_U), ymax = np.max(data_U),
        linestyle = 'dashed', lw = 0.5, color = 'red')

plt.ylim(ymin = 0)

plt.xlabel('Time (ms)')
plt.ylabel('U (V)')

plt.plot(fit_times*10**3, U0_fit*np.exp(-fit_times/tau_fit), label = 'fit')
plt.legend()

#Input pour la capacité, afin de déduire la résistance à partir de tau
C = 800*10**-6

R_fit = (tau_fit*10**-3)/C

print('Résultats du fit: ', np.round(U0_fit, 3), np.round(tau_fit*10**3, 3))

print('Estimation de la résistance du railgun: ', np.round(R_fit*10**3, 3), 'milli-Ohms')
```

```
#
# ============================================================================
# Fonctions pour simulation dynamique du système
#
# ============================================================================

#on prend comme temps initial le premier point de la décroissance
t0 = fit_times[0]

#on prend comme temps final le dernier point des mesures
tf = data_times[-1]

#nombres d'itérations de la simulation
steps = 1000

dt = (tf - t0 )/steps

#array avec tous les instants discrets
simu_times = np.linspace(t0, tf, steps)

#on initialize à des arrays vides les résultats de la simu
simu_U = np.empty(steps)
simu_B = np.empty(steps)



#e=float(input("épaisseur du rail (en mm)="))/1000
#l=float(input("largeur du projectile (en mm)="))/1000
#h=float(input("hauteur du rail(en mm)="))/1000
#U0=float(input("tension aux bornes du condensateur à t0 (en V)="))
#C=float(input("capacité du condensateur (en microF)="))/1000000
#R=float(input("résistance du circuit (ohm)="))

e = 10*10**(-3) #in m
l = 10*10**(-3) #in m
h = 10*10**(-3) #in m
U0 = U0_fit #on peut prendre la valeur issue du fit
C = 800*10**(-6) #in F
R = R_fit #on peut prendre la valeur issue du fit

tau = tau_fit #RC ou la valeur issue du fit



s=e*h #in m^2
```

```
temps=[]
champ=[]
force=[]
vitesse=[]
joule=[]


t = t0

for i in range (1000):

    U=U0*np.exp(-t/tau)
    I=U/R     #I=C*U0*math.exp(-t/tho)*(-1/tho)
    j0=I/s     #calcul du champ
    B=(mu_0*I/(4*l))*(np.log((e/2+l)/(e/2)))    #en Gauss
    F=B*l*I
    dv=F*dt/0.003
    P=(U*I)*dt
    temps.append(t*1000)
    champ.append(B)
    force.append(F)
    vitesse.append(dv)
    joule.append(P)
    t=t+dt

E=(C*U0**2)/2 #intégration des variations de vitesse sur t

V=sum(vitesse)
J=sum(joule)

print("La vitesse est de " + str(V) + " m/s." + "\n"
"L'énergie initiale est de " + str(E) + " J." + "\n"
"Lénergie perdue par effet Joule est de " + str(J) + " J." + "\n"
"L'énergie cinétique est de " + str(0.5*0.003*V**2) + " J." + "\n") #affichage des graphes
plt.figure("B (T)")

plt.figure()
plt.plot(data_times*10**3, data_B)
plt.plot(temps,champ)
plt.xlabel("Temps (ms)")
plt.ylabel("B")
plt.title("modélisation de l'évolution de B en fonction de t ")
plt.figure("F (N)")


plt.plot(temps,force)
plt.xlabel("Temps (ms)")
plt.ylabel("F")
plt.title("évolution de F en fonction de t")
```