```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 30 09:10:00 2022

@author: fabrice
"""

#système différentiel de lorentz résolution et représentation graphique animée
#de la variable x en fonction du temps, pour 2 conditions initiales

import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import matplotlib.animation as animation

##déclaration graphique
fig = plt.figure(figsize = (12, 6))
ax = fig.add_subplot()

from matplotlib import rc
font_properties = {'size' : 14}
rc('font' , **font_properties)

#constantes k1 = sigma k2=rho k3 =béta
k1 , k2 , k3 = 10 , 28 , 8/3

#conditions initiales (y1=x, y2=y , y3=z)
y10 , y20 , y30 = 0.1 , 0.1 , 0.1
y101 , y20 , y30 = 0.1001 , 0.1 , 0.1
y0 = y10 , y20 , y30
y1 = y101 , y20 , y30

cond_init = y0 , y1 #vecteur des conditions initiales qui sera utilisé dans la
#boucle

#axe des temps
tmax = 200

#définition des variables fonction de t et écriture des trois équations couplées
def dydt(t , y , k1 , k2, k3):
    y1 , y2 , y3 = y
    dy1dt = k1*(y2 - y1)
    dy2dt = k2*y1 - y2 - y1*y3
    dy3dt = y1*y2 - k3*y3
    return dy1dt , dy2dt , dy3dt

#fichier à 2 dimensions pour stocker les valeurs de x calculées
valeur = np.empty((2 ,  tmax))
j = 0

#intégration. boucle pour obtenir les valeurs de x qui seront affichées sur le graphi
for y0 in cond_init:
```

```
    solution = solve_ivp(dydt , (0 , tmax) , y0 , dense_output = True, args=(k1,k2,k3)

    #fichier avec les valeurs de l'axe horizontal et extraction des réponses
    #convection=x (dénomination de Lorentz)

    t = np.linspace(0 , tmax , tmax+1 , dtype=int)
    convection , delta_T, profil = solution.sol(t)

    #boucle pour remplir le fichier des valeurs
    for i in range(tmax):
        valeur[j , i] = convection[i]
    j+= 1

#partie animation
x , y1 , y2 = [] , [] , []

line1, = ax.plot([] , [],  lw=2 , c = "k" ,  label = "condition initiale, x0 = " +str(
                 linestyle="--")
line2, = ax.plot([] , [],  lw=2, c = "k" , label = "condition initiale, x0 = " +str(y1
                 linestyle="-")

ax.set_xlim(0 , tmax)

#recherche de la valeur la plus élevée de x pour les limites du graphique
if max(valeur[0]) < max(valeur[1]):
    maximum = max(valeur[1])

else:
    maximum = max(valeur[0])

if min(valeur[0]) < min(valeur[1]):
    minimum = min(valeur[0])

else:
    minimum = min(valeur[1])

#limites du graphique et titre des axes
ax.set_ylim(minimum , maximum)
ax.set_xlabel("temps (s)")
ax.set_ylabel("x / Convection (u.a.)")

def animate(t):
    global  x , y1 , y2
    x.append(t)
    y1.append(valeur[0][t]*0.5) #on peut insérer un facteur pour réduire les valeurs
    y2.append(valeur[1][t]*0.5) #on peut insérer un facteur pour réduire les valeurs
    line1.set_data(x, y1)
    line2.set_data(x, y2)

#commande de l'animation
ani = animation.FuncAnimation(fig , animate, frames = t,
                      interval = 20 , repeat = False)

plt.title("Equation de Lorentz et sensibilité aux conditions initiales, x en fonction
```

```
plt.legend()
plt.grid()

#sauvegarde fichier(optionnel)
ani.save('lorentz x versus t avec chaos.mp4',fps=10,extra_args=['-vcodec', 'libx264'])


plt.show()
```