```
# -*- coding: utf-8 -*-
"""
Cap 4

@author: campioni
"""

#EQUAZIONI DI LORENZ
#USO DEL METODO ITERATIVO PER IL CALCOLO DEL MASSIMO ESPONENTE DI LYAPUNOV
import numpy as np
import matplotlib.pyplot as plt
import math
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from pylab import figure, show, setp
from termcolor import colored

def intero_num(x):
  while True:
     try:
       a=int(input(x))
       return a
     except ValueError:
         print()
         print (colored('Non è un numero valido (intero). Riprova...','red'))

def virgola_num(x):
  while True:
     try:
       a=float(input(x))
       return a
     except ValueError:
         print()
         print (colored('Non è un numero valido (virgola mobile).
Riprova...','red'))

print()
h=virgola_num('passo h =  ')
n=intero_num('numero massimo dei passi =  ')
tmax=n*h
print()
print(' Immissione dei valori iniziali di x , y , z')
print()
x10=virgola_num('Valore iniziale x1 =  ' )
print()
y10=virgola_num('Valore iniziale y1 =  ' )
print()
z10=virgola_num('Valore iniziale z1 =  ' )
print()
print()
e=virgola_num('Perturbazione epsilon =  ' )
x20=x10+e
y20=y10+e
z20=z10+e
print()
print()

A ,  B , C =float(10), float(28), float(8/3)  # valori scelti

# definisce le 3 equazioni di Lorenz
def fx(x,y,z,t): return A*(y-x)
def fy(x,y,z,t): return (B*x-y-x*z)
def fz(x,y,z,t): return (x*y-C*z)
#a) Definisce il metodo classico di Runge-Kutta del 4° ordine
```

```
def RK45(x,y,z,fx,fy,fz,t,h):
    k1x,k1y,k1z=h*fx(x,y,z,t),h*fy(x,y,z,t),h*fz(x,y,z,t)
    k2x,k2y,k2z=h*fx(x+k1x/2,y+k1y/2,z+k1z/2,t+h/2),h*fy(x+k1x/2,y+k1y/2,z+k1z/
2,t+h/2),h*fz(x+k1x/2,y+k1y/2,z+k1z/2,t+h/2)
    k3x,k3y,k3z=h*fx(x+k2x/2,y+k2y/2,z+k2z/2,t+h/2),h*fy(x+k2x/2,y+k2y/2,z+k2z/
2,t+h/2),h*fz(x+k2x/2,y+k2y/2,z+k2z/2,t+h/2)

k4x,k4y,k4z=h*fx(x+k3x,y+k3y,z+k3z,t+h),h*fy(x+k3x,y+k3y,z+k3z,t+h),h*fz(x+k3x,y
+k3y,z+k3z,t+h)
    return x+(k1x+2*k2x+2*k3x+k4x)/6,y+(k1y+2*k2y+2*k3y+k4y)/6,z+
(k1z+2*k2z+2*k3z+k4z)/6
tmax=n*h
t=np.linspace(0 ,tmax , n)  #per diagrammi nel tempo
x1=np.zeros(n)
y1=np.zeros(n)
z1=np.zeros(n)
x2=np.zeros(n)
y2=np.zeros(n)
z2=np.zeros(n)
V=np.zeros(n)
V[0]=math.sqrt(3)*e
#Condizioni iniziali del punto 1
x1[0],y1[0],z1[0] = x10, y10,z10
#Condizioni iniziali del punto 2
x2[0],y2[0],z2[0] = x20, y20, z20
for i in range(1, n):
    x1[i],y1[i],z1[i] = RK45(x1[i-1],y1[i-1],z1[i-1], fx,fy,fz, t[i-1], h)
    x2[i],y2[i],z2[i] = RK45(x2[i-1],y2[i-1],z2[i-1], fx,fy,fz, t[i-1], h)
    a=(x2[i]-x1[i])
    b=(y2[i]-y1[i])
    c=(z2[i]-z1[i])
    a1=math.sqrt (a**2+b**2+c**2)
    V[i]=a1       #lista della distanza tra le due traiettorie
    x2[i]=x1[i]+a*V[0]/V[i]    #normalizza ad ogni passo
    y2[i]=y1[i]+b*V[0]/V[i]
    z2[i]=z1[i]+c*V[0]/V[i]
Ls=0
for i in range (2000,n):      #calcola dopo 2000 passi
    Lf=math.log(abs(V[i]/V[0]))
    Ls +=Lf
Lyapunov=Ls/(n+1-2000)/h
print()
plt.subplots(facecolor='olive')
plt.rcParams['axes.facecolor'] = 'azure'
print(colored("-L'esponente maggiore di Lyapunov è  %.5f "%(Lyapunov),'red'))
plt.title("DISTANZA TRA I DUE PUNTI. Lorenz")         # diagramma x , y
plt.xlabel(' tempo')
plt.ylabel('Distanze')
plt.grid()
plt.plot(t,V,linewidth=.5, color='r' )
plt.show()
```