```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 28 15:17:49 2020

@author: davidstothers

COIS 2310H Assignment 3 Question 3

This program's goal is to plot the path of a swinging pendulum using the Euler-C
to compare the results with the numerical solution to the problem.
"""
import numpy as np
import matplotlib.pyplot as plt

def Fnet(theta,omega,q,dfreq,damp,t):
    """
    Calculates the force on a pendulum given its angle from the vertical and its
    Arguments: (theta,omega,q,dfreq,damp,t)
    Returns the net force for the given angle and angular speed.
    """

    Fdamp = -1.0*m*q*omega
    Fg = -1.0*(1.0*theta)*m*g/L # This line can be replaced with sin(theta)*m*g/
    Fdrive = damp*m*np.sin(dfreq*t)

    Fnet = 1.0*(Fg+Fdamp+Fdrive)
    return Fnet

def motion(theta0=1.0,omega0=0.0,forceq=0.0,drivefreq=1.0,driveamp=0.0):
    """
    Uses the Euler-Cromer method to approximate the motion of a pendulum,
    to about 0.08 radians of accuracy.
    Arguments: (theta0 = 1.0, omega0 = 0.0, forceq=0.0, drivefreq=1.0, driveamp=
    Returns arrays for time, position, and speed.
    """
    # Error currently sits at about 0.08 radians, or about 4.58 degrees.
    # It is not sensitive to dt.
    # I cannot find any ways to optimize the code further.

    N = 150001
    tmax = 10.0
    t=np.linspace(0,tmax,N,dtype=float)
    dt=1.0*(t[1]-t[0])

    theta=np.arange(0,N,dtype=float)
    omega=np.arange(0,N,dtype=float)

    theta[0]=theta0
    omega[0]=omega0
```

```
    for i in np.arange(0,N-1):
        omega[i+1] = 1.0*(omega[i]+(Fnet(theta[i],omega[i],forceq,drivefreq,driv
        theta[i+1] = 1.0*(theta[i]+omega[i+1]*dt)

        #if(theta[i+1]>np.pi):
        #    theta[i+1]-=2.0*np.pi
        #elif(theta[i+1]<-1.0*np.pi):
        #    theta[i+1]+=2.0*np.pi

    return t[:],theta[:],omega[:]

def exactUndDamSolRest (t,q=1.0,initialangle=1.0):
    """
    Takes a time and calculates the position of an underdamped pendulum based on
    Arguments: (t, tinitialangle = 1.0)
    Returns the theta corresponding to the given time.
    """
    ROOT=np.sqrt((1.0*g/L)-((q**2)/4.0))
    phi=np.arctan((2.0*ROOT)/q)

    theta0 = initialangle/np.sin(phi) # Thanks to prof. Bill Atkinson for helpin

    theta = theta0
    theta*= np.e**(-0.5*q*t)
    theta*= np.sin(phi+t*ROOT)
    return theta

def exactOvrDamSolRest (t,q=10.0,initialangle=1.0):
    """

    """
    BETA = np.sqrt((q**2)/4 - g/L)-q/2
    BETA/= q/2 + np.sqrt((q**2)/4 - g/L)

    theta2 = initialangle*1.0/(BETA+1)
    theta1 = initialangle-theta2

    theta = theta1*np.e**(-(q/2-np.sqrt(((q**2)/4.0)-g/L))*t)

    theta+= theta2*np.e**(-(q/2 - np.sqrt((q**2)/4 - g/L))*t)

    return theta

def exactCrtDamSolRest (t,q=5.0,initialangle=1.0):
    """

    """
    theta0=initialangle
    C = q*theta0/2.0
```

```python
    theta = theta0+C*t
    theta*= np.e**(-q*t/2.0)

    return theta

#The exact driven method currently does not work.
def exactDrvUndSolRest (t,q=1.0,initialangle=1.0,driveamp=0.0,drivefreq=1.0):

    omega0 = g/L
    OMEGA = np.sqrt(omega0**2 - (q/2)**2)

    theta2 = driveamp
    theta2/= np.sqrt(((OMEGA**2)-(drivefreq**2))**2+(q*drivefreq)**2)

    ROOT=np.sqrt((1.0*g/L)-((q**2)/4.0))
    phi=np.arctan((2.0*ROOT)/q)

    theta1 = initialangle/np.sin(phi)

    theta = theta1
    theta*= np.e**(-0.5*q*t)
    theta*= np.sin(phi+t*ROOT)

    theta+= theta2*np.sin(drivefreq*t)

    return theta

global g
global L
global m

g = 9.81 # m/s^2
L=2.5 # m
m=5.0 # kg

#Switching the 0.5*np.pi for 1.0 removed 7/9 of the error in the approximation.
#Welp. Time to root out other stuff.
inittheta=1.0

q=0.0
pt,pth,po = motion(forceq=q,theta0=inittheta)
pte=pt.copy()

q=1.0
p2t,p2th,p2o = motion(forceq=q,theta0=inittheta)
p2te=p2t.copy()
p2the=exactUndDamSolRest(p2te[:])

q=2*np.sqrt(g/L)
```

```
p3t,p3th,p3o = motion(forceq=q,theta0=inittheta)
p3te=p3t.copy()
p3the=exactCrtDamSolRest(p3te[:])

q=10.0
p4t,p4th,p4o = motion(forceq=q,theta0=inittheta)
p4te=p4t.copy()
p4the=exactOvrDamSolRest(p4te[:])

plt.figure(0)
plt.clf()

plt.plot(pt,pth,'b',label="Undamped")

plt.plot(p2te,p2the,'g',label="Underdamped")
plt.plot(p2t[::5000],p2th[::5000],'go',mfc="none")

plt.plot(p3te,p3the,'r',label="Critically damped")
plt.plot(p3t[::5000],p3th[::5000],'ro',mfc="none")

plt.plot(p4te,p4the,'y',label="Overdamped")
plt.plot(p4t[::5000],p4th[::5000],'yo',mfc="none")

plt.legend()
plt.xlabel("Time (s)")
plt.ylabel("Angle (rad)")
plt.title("Angle vs. time of a pendulum")
plt.show()

plt.figure(1)
plt.clf()

plt.plot(p2t,p2th,'b',label="Euler-Cromer")
plt.plot(p2te,p2the,'g',label="Exact")
err = np.abs(p2th[:]-p2the[:])
plt.plot(p2t,err,'r',label="error")

plt.title("Comparison of Euler-Cromer Method vs. \nDifferential Solution of an U
plt.xlabel("Time (s)")
plt.ylabel("Angle (rad)")
plt.legend()
plt.show()

print(str(np.argmax(err[:])))

N=150001
td = np.linspace(0,20,N)
FD = 1.0

th1e = exactDrvUndSolRest(td[:],driveamp = FD,drivefreq = g/L - 1)
```

```python
th2e = exactDrvUndSolRest(td[:],driveamp = FD,drivefreq = g/L)
th3e = exactDrvUndSolRest(td[:],driveamp = FD,drivefreq = g/L + 1)

t1,th1,o1 = motion(drivefreq = g/L-1, driveamp = FD)
t2,th2,o2 = motion(drivefreq = g/L, driveamp = FD)
t3,th3,o3 = motion(drivefreq = g/L+1, driveamp = FD)

plt.figure(2)
plt.clf()

plt.plot(td,th1e,'b',label = "OMEGAD<OMEGA")
plt.plot(td,th2e,'g',label = "OMEGAD=OMEGA")
plt.plot(td,th3e,'r',label = "OMEGAD>OMEGA")

plt.plot(td,th1[::],'bo',mfc="none")
plt.plot(td,th2[::],'go',mfc="none")
plt.plot(td,th3[::],'ro',mfc="none")

plt.legend()
plt.xlabel("Time (s)")
plt.ylabel("Theta (rad)")
plt.title("Three different cases of an underdamped, driven pendulum")
plt.show()
```