PHYS 2310H
Assignment 4
David Stothers
Mar. 23 2020

(1a) Inner Function: L E G
Outer Function: E E G
Main Program: G G G

The variables have no local value when they are used without being declared in a function, so each variable used like this retains its value from the layer above the current one.

(1b) The global tells outer to use the global memory address of x, so when x is changed in outer, it changes for the main function too.

Inner Function: L E G
Outer Function: E E G
Main Program: E G G

(1c) This time, x has its global address inside inner but not outer, so once x changes to L, it's like that in the main program too.

Inner Function: L E G
Outer Function: E E G
Main Program: L G G

(1d)

(2) x = x+10 assigns a value to a variable, and assigns it only. The new x takes the value of the argument x added to 10. Meanwhile, += takes a variable's address and increments it, so the old address of x is used and then added to.

(3d)

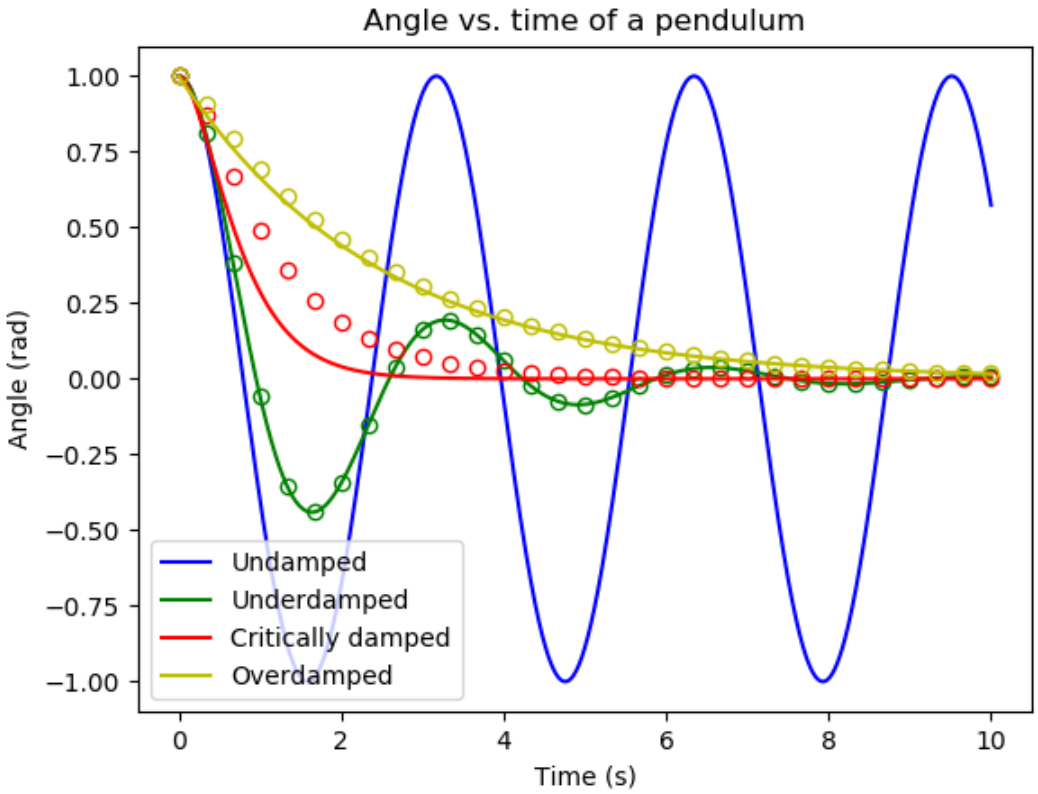


Figure 1: The graph of a pendulum's various cases. The assumptions in (3a) were used in the calculations. In addition, the undamped case of the pendulum is shown. The lines of the damped cases represent the exact solution curves, and the dots represent the numerical solutions found. Each case has been colour coded as well.

(3e)

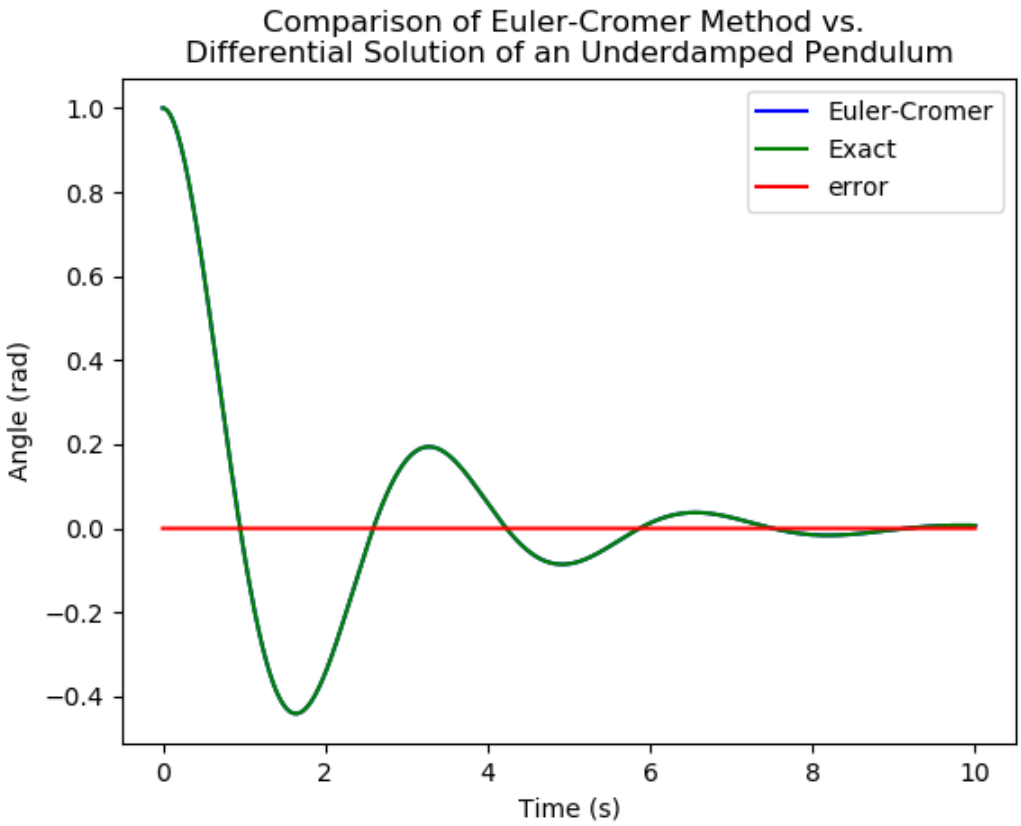


Figure 2: The exact and numerical solutions of the underdamped pendulum from (3d), graphed side by side. The error on this graph was so small at N = 150001 (the default value I used) that it might as well have been 0.

(3f)

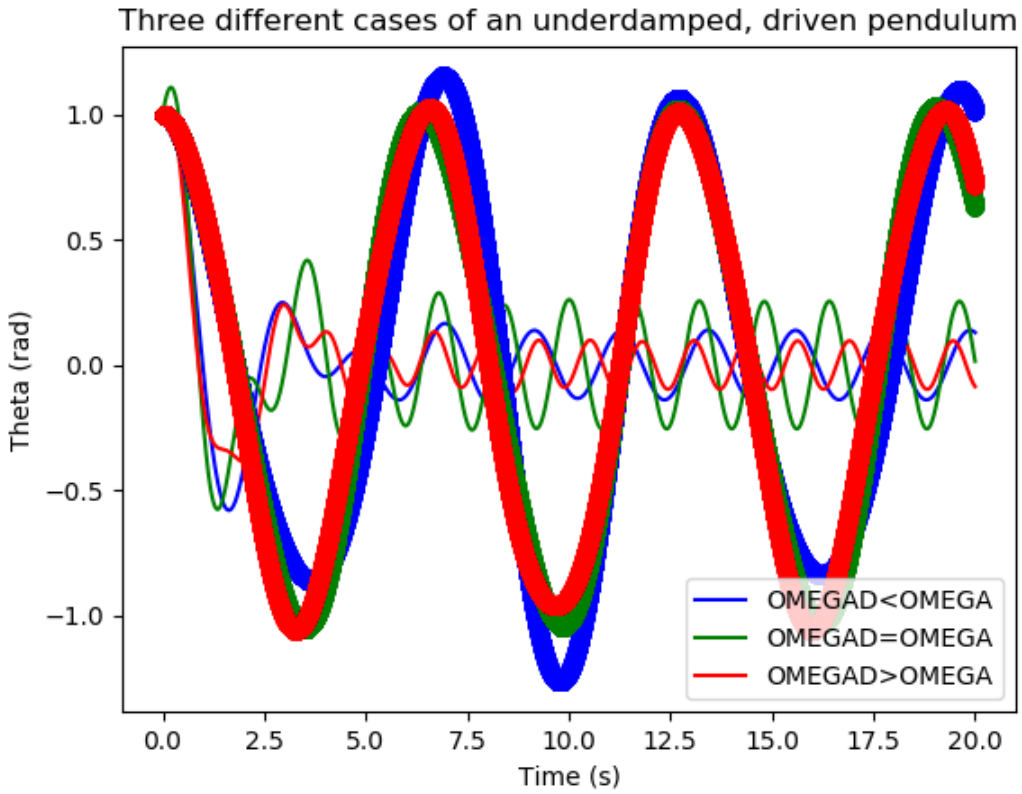


Figure 3: Three exact solutions of an underdamped, driven pendulum. Note the green case being of higher amplitude than the rest, showing a resonance at OMEGAD = OMEGA. The thick lines are the numerical solutions, which do not appear to follow the exact solutions.

(4b) For the following figures, the textbook's initial parameters were taken – both pendulums start from rest and from an angle of 0.2 radians. Their masses were both 1 kg and their lengths were both 9.81 metres. The textbook differed here in that it used only 9.8 m/s^2 for gravitational acceleration, but I erred in favour of the string lengths and the gravitational acceleration having the "same" magnitudes, if one can call it that.

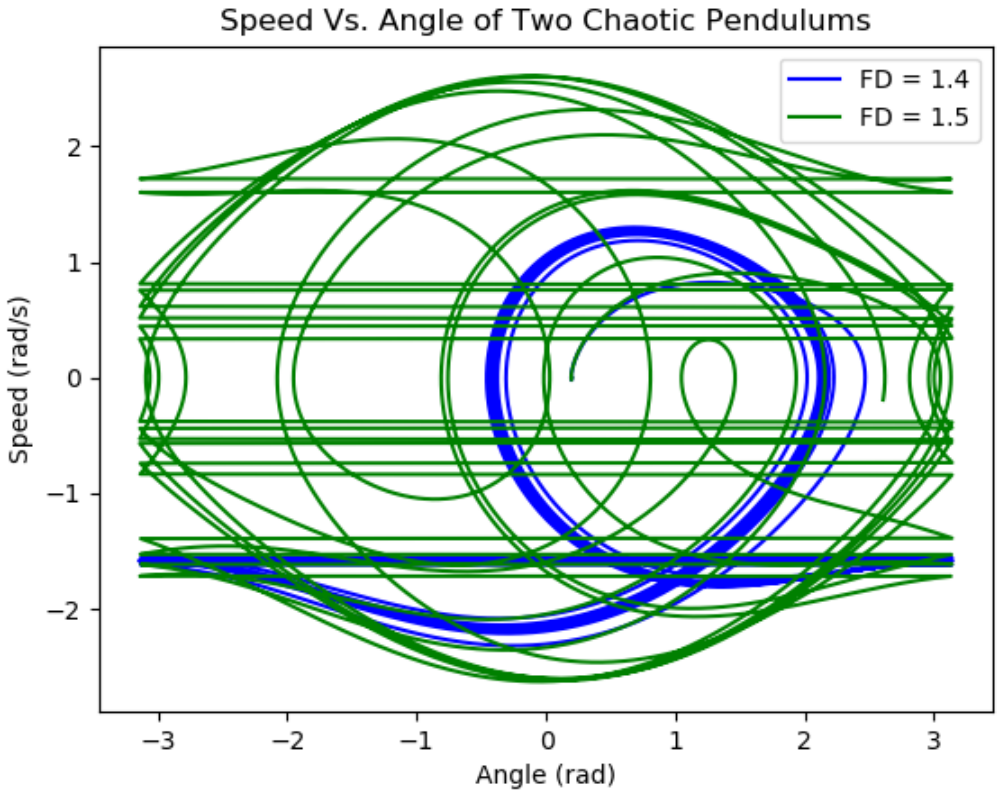


Figure 4: A plot of the pendulums' speeds as they change with angle. Here we can see – disregarding the horizontal lines corresponding to a remapping of the angle back into the interval [-pi,pi) – that the pendulum with FD = 1.5 has its maximum speed near theta = 0, but the pendulum with FD = 1.4 follows a cyclic, repetitive pattern. As an aside, I found these types of figures in the textbook to look incredibly cool.

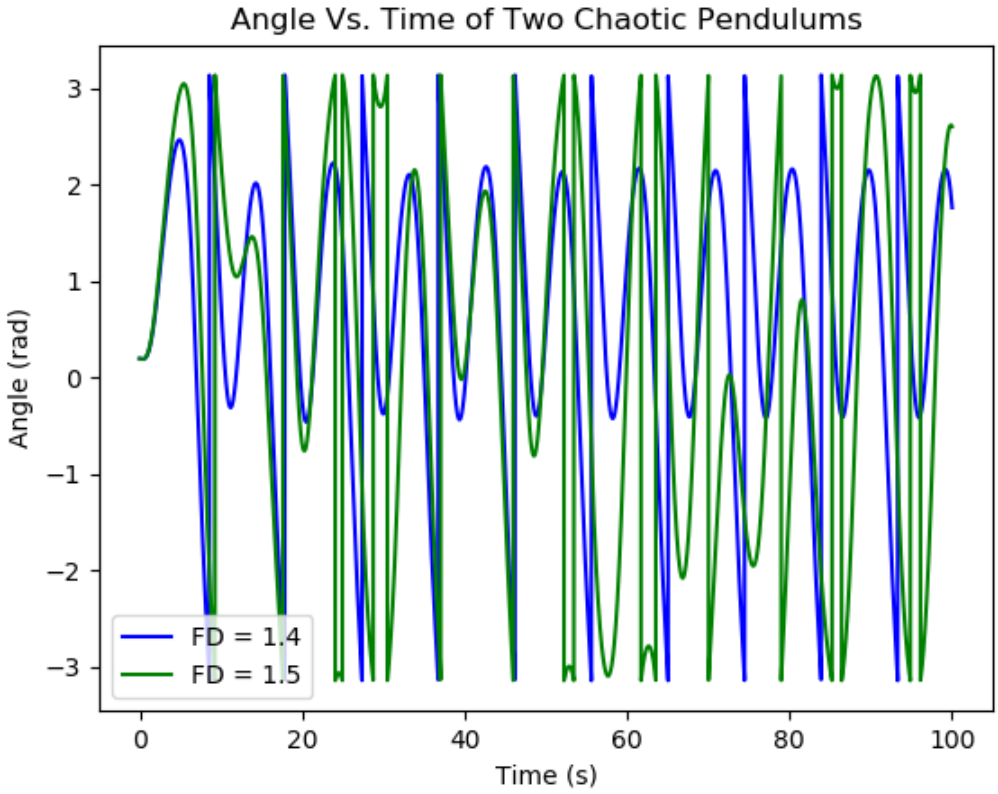


Figure 5: A plot of the two pendulums' angles with respect to time. Note the vertical lines corresponding to mapping, and the erratic behaviour of the pendulums.

(3a) $E = \frac{mL^2\omega^2}{2} + mgL(1-\cos\theta)$

$\omega_{i+1} = \omega_i - \frac{g\theta_i \Delta t}{L}$

$\theta_{i+1} = \theta_i + \omega_i \Delta t$

Assuming low θ: $\cos\theta \cong 1$

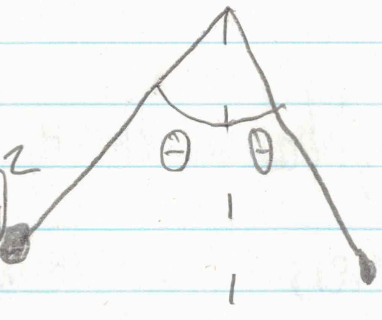
$E = \frac{mL^2\omega^2}{2}$

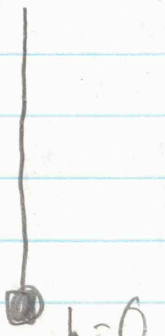
$\downarrow$

$E_{i+1} = \frac{mL^2(\omega_{i+1})^2}{2}$

$\downarrow$

$= mL^2\left(\omega_i - \frac{g\theta_i \Delta t}{L}\right)^2$





Crit:

$\Omega^2 = \frac{q^2}{4}$

$q = 2\Omega$

ie Eq. 3.8

In essence, the rightmost term can never be θ_i. This means energy is always added to the system. or negative.

$mL^2\left(\omega_i^2 - \frac{2g\theta_i\omega_i\Delta t}{L} + \left(\frac{g\theta_i\Delta t}{L}\right)^2\right)$

(36) This question was tackled assuming gravity, the pendulum mass, and the pendulum length would not change. They are global variables, set to 9.81 m/s^2, 5.0 kg, and 2.5 m respectively.

[In the program, q is a global variable as well, to avoid having to pass it through two layers of F arguments. This will change in Question 4.)

I'm also assuming a start from rest, to keep the exact solution simple. The pendulums will also start at one radian, to avoid having to round π.

Update; q has been made an argument of the EC and Fnet functions.

(3d) $\theta(t) = \theta_0 e^{\frac{-qt}{2}} \sin(\alpha t + \phi) \qquad \alpha = \sqrt{\frac{g}{L} - \frac{q^2}{4}}$

$\omega(t) = \theta_0\left(\frac{-q}{2}\right)e^{\frac{-qt}{2}} \sin(\alpha t + \phi) + \theta_0 e^{\frac{-qt}{2}}(\alpha)\cos(\alpha t + \phi)$

$\omega(t) = \theta_0 e^{\frac{-qt}{2}}\left(\frac{-q}{2}\sin(\alpha t + \phi) + \alpha\cos(\alpha t + \phi)\right)$

↓ start from rest

$\omega(0) = 0 = \theta_0\left(\frac{-q}{2}\sin(\alpha t + \phi) + \alpha\cos(\alpha t + \phi)\right)$

↓ $\theta_0 \neq 0$

$0 = \frac{-q}{2}\sin(\phi) + \alpha\cos\phi$

$\frac{q}{2}\sin\phi = \alpha\cos\phi$

$\tan\phi = \frac{2\alpha}{q}$

$\theta(t) = (\theta_0 + Ct)e^{\frac{-qt}{2}} \qquad \theta(0) = \theta_0$

$\omega(t) = Ce^{\frac{-qt}{2}} + (\theta_0 + Ct)\left(\frac{-q}{2}\right)\left(e^{\frac{-qt}{2}}\right)$

$\omega(0) = C + (\theta_0)\left(\frac{-q}{2}\right)$

$0 = C + \frac{-q\theta_0}{2}$

$C = \frac{q\theta_0}{2}$

(3e) 03/18/20: As of right now, the error is not dependent on dt. Until larger sources of error are found it shall remain this way.
03/22/20: Thanks to the prof's help I've fixed the error, and it's negligible now.

(3f) $F_D = A\sin(\Omega_D t)$ Note that this is actually for 3d

$$\theta(t) = \Theta_1 e^{-\left(\frac{q}{2}+\sqrt{\frac{q^2}{4}-\Omega^2}\right)t} + \Theta_2 e^{-\left(\frac{q}{2}-\sqrt{\frac{q^2}{4}-\Omega^2}\right)t}$$

$$\omega(t) = \Theta_1(-\alpha^+)e^{-\alpha^+ t} + \Theta_2(-\alpha^-)e^{-\alpha^- t}$$

$$= \Theta_2\left(\sqrt{\frac{q^2}{4}-\Omega^2}-\frac{q}{2}\right)e^{-\alpha^- t} - \Theta_1\alpha^+ e^{-\alpha^+ t}$$

$$\theta(0) = \Theta_1 + \Theta_2$$

$$\omega(0) = \Theta_2\left(\sqrt{\frac{q^2}{4}-\Omega^2}-\frac{q}{2}\right) - \Theta_1\left(\frac{q}{2}+\sqrt{\frac{q^2}{4}-\Omega^2}\right)$$

$$\Theta_1\left(\frac{q}{2}+\sqrt{\frac{q^2}{4}-\Omega^2}\right) = \Theta_2\left(\sqrt{\frac{q^2}{4}-\Omega^2}-\frac{q}{2}\right)$$

$$\Theta_1 = \frac{\left(\sqrt{\frac{q^2}{4}-\Omega^2}-\frac{q}{2}\right)}{\left(\frac{q}{2}+\sqrt{\frac{q^2}{4}-\Omega^2}\right)}\Theta_2 \qquad \theta(0) = (\beta+1)\Theta_2$$

β

(3f) $F_D = A\sin(\Omega_D t)$

Googling also nets the result

$$\theta(t) = A_h e^{-\gamma t}\sin(\omega' t + \varphi_h) + A\cos(\omega t - \varphi)$$

$$\therefore \omega(t) = A_h(-\gamma)e^{-\gamma t}\sin(\omega' t + \varphi_h) + A_h e^{-\gamma t}\omega'\cos(\omega' t + \varphi_h) - A_h \sin(\omega t - \varphi)$$

where:

$$A_h = \frac{x_0 - A\cos\varphi}{\sin\varphi_h} \qquad \gamma = \frac{c}{2m} \qquad \omega' = \sqrt{\omega_0^2 - \gamma^2}$$

$$\omega_0 = \sqrt{\frac{k}{m}}$$

$$\varphi_h = \arctan\left(\frac{\omega'(x_0 - A\cos\varphi)}{v_0 + \gamma(x_0 - A\cos\varphi) - A\omega\sin\varphi}\right)$$

$$A = \frac{F_0}{m\sqrt{(\omega_0^2 - \omega^2)^2 + 4\gamma^2\omega^2}} \qquad \varphi = \arctan\left(\frac{c\omega}{k - m\omega^2}\right) - \varphi_d$$

c is the damping coefficient

However

if the force is instead $F_0\cos(\Omega_D t)$

the solution is $x(t) = x_0\cos(\omega t + \phi)$

where $x_0 = \frac{F_0}{m\sqrt{(\omega^2 - \omega_0^2)^2 + (\omega\gamma)^2)}}$, $\phi = \arctan\left(\frac{\gamma\omega}{\omega^2 - \omega_0^2}\right)$

$\omega_0 = \frac{k}{m}$ $\gamma = q$

Hilroy

$$\theta = \theta_D + \theta_H$$

$$= A_D \sin(\Omega_D t) + A_H e^{-\frac{qt}{2}} \sin(\alpha t + \phi)$$

$$\omega = A_D \cos(\Omega_D t) + A_H\left(\frac{-q}{2}\right) e^{-\frac{qt}{2}} \sin(\alpha t + \phi)$$

$$+ A_H(\alpha) e^{-\frac{qt}{2}} \cos(\alpha t + \phi)$$

$$\theta_{(0)} = A_H \sin\phi$$

$$\omega_{(0)} = A_D + A_H\left(\frac{-q}{2}\right)\sin\phi + A_H(\alpha)\cos\phi$$

$$= A_D + \theta_{(0)}\left(\frac{-q}{2}\right) + A_H \alpha \cos\phi$$

4a $\Delta\theta \simeq e^{\lambda t}$

$\frac{\ln(\frac{\Delta\theta}{c}) = \lambda t}{}$

$\frac{\ln(\frac{\Delta\theta}{c})}{t} = \lambda$

Assuming $c = 1$

$\frac{\ln(\Delta\theta)}{t} = \lambda$

```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
```

```
    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)
```

```
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()
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 18 15:11:46 2020

COIS 2310H Assignment 3 Question 4

@author: davidstothers
"""

import q4lib
import numpy as np
import matplotlib.pyplot as plt

N=15001

t = np.linspace(0,100,N)
m=1
L=9.81

theta01=0.2
theta02=0.2

th1,o1 = q4lib.pendulum_EC(t,mass=m,Length=L,forceq=0.5,driveamp=1.4,theta0=thet
th2,o2 = q4lib.pendulum_EC(t,mass=m,Length=L,forceq=0.5,driveamp=1.5,theta0=thet

plt.figure(0)
plt.clf()
plt.plot(th1,o1,'b',label="FD = 1.4")
plt.plot(th2,o2,'g',label="FD = 1.5")
plt.title("Speed Vs. Angle of Two Chaotic Pendulums")
plt.xlabel("Angle (rad)")
plt.ylabel("Speed (rad/s)")
plt.legend()
plt.show()

plt.figure(1)
plt.clf()
plt.plot(t,th1,'b',label="FD = 1.4")
plt.plot(t,th2,'g',label="FD = 1.5")
plt.title("Angle Vs. Time of Two Chaotic Pendulums")
plt.xlabel("Time (s)")
plt.ylabel("Angle (rad)")
plt.legend()
plt.show()

plt.figure(2)
plt.clf()
dth = np.abs(th2[:]-th1[:])
plt.plot(t,dth,'b')
```

```
plt.title('dtheta vs t')

print(str(np.argmax(q4lib.lyapunov(t[2:],th1[2:],th2[2:]))))
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 18 15:14:07 2020

COIS 2310H Assignment 3 Question 4 (Module)

@author: davidstothers
"""

import numpy as np


global g
g=9.81

def pendulum_EC(time,mass=1.0,Length=1.0,theta0=1.0,omega0=0.0,forceq=0.0,drivef
    """
    Uses the Euler-Cromer method to approximate the motion of a pendulum.

    time: numpy array of time locations.
    mass: mass of the pendulum.
    Length: length of the pendulum.
    theta0: the initial angle of the pendulum.
    omega0: the initial speed of the pendulum.
    forceq: the constant determining the damping force.
    drivefreq: the frequency of the driving force.
    driveamp: the amplitude of the driving force.

    Returns arrays for position and speed.
    """
    timelength=np.size(time)

    theta=np.arange(0,timelength,dtype=float)
    theta[0]=theta0
    omega=np.arange(0,timelength,dtype=float)
    omega[0]=omega0

    dt=np.arange(0,timelength,dtype=float)

    for i in np.arange(0,timelength-1):
        dt[i] = time[i+1]-time[i]
        omega[i+1]=omega[i]+(Fnet(mass,Length,theta[i],omega[i],forceq,drivefreq
        theta[i+1]=theta[i]+omega[i+1]*dt[i]

        while(theta[i+1]>=np.pi):
            theta[i+1]-=2.0*np.pi
        while(theta[i+1]<-1.0*np.pi):
            theta[i+1]+=2.0*np.pi
```

```
    return theta,omega

def Fnet(m,L,theta,omega,q,dfreq,damp,t):
    Fdamp = -1.0*m*q*omega
    Fg = -1.0*np.sin(1.0*theta)*m*g/L
    Fdrive = damp*np.sin(dfreq*t)

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

def lyapunov(time,theta1,theta2):
    dtheta=np.abs(theta2-theta1)
    ly = np.log(dtheta)*1.0/time

    return ly
```