Lab 2: The Nonlinear Pendulum

Computational Labs
Kevin Grainger
20333336
02/03/2022

Contents:

1) Introduction

The purpose of this lab was to investigate the dynamics of the pendulum, both simple,damped and driven forms. By translating the equations of motion of both systems into python functions we can track the motion graphically. This lab involves five exercises each changing the pendulum system, and thus comparing the motion and variable relations graphically.

(1.1) Exercise 1

This exercise involves a linear pendulum. The equations of motion for a linear pendulum are approximated as such:

$$\frac{d\theta}{dt} = \omega$$

$$\frac{d\omega}{dt} = \frac{g}{l}\theta$$

We used the trapezoidal rule to solve ODE's.

Using these we can graph the variables theta and omega against time.

(1.2)Exercise 2

This exercise involves solving the nonlinear pendulum equation. This is achieved by using the full pendulum equation.

$$F(\theta, \omega, t) = -(g/l)Sin\theta - k\omega + Acos\phi t$$

As before it is solved using the trapezoidal rule. The aim of this section is simply to compare the linear and nonlinear pendulum. This is done in graphs 2[a] & 2[b]

(1.3)Exercise 3

All that differs in this exercise is the method for solving the ODE's. In this exercise I used the Runge-Kutta method. This is simpler to the Trapezoidal method but the midpoint is used for the expansion, this means the Runge-Kutt method is accurate to the order of Δt^2.

(1.4)Exercise 4

A damping force was added to the pendulum. By graphing theta and omega once again we can determine the dampenings' effects on the dynamics.

I graphed theta and omega against time to demonstrate the motion of the system.

(1.5)Exercise 5

I introduce a driving force to the system. This affects the periodicity of the pendulum depending on the harmonics/frequency of the driving force. If the driving force frequency is a multiple of the

original frequency, it will give periodic motion. This is known as period doubling. If the driving force and pendulum motion are asynchronous chaotic motion may occur.
To analyse this complex motion we use phase portraits, this maps the value of theta against omega. This is done as it is easier to spot periodic motion on these graphs.
Lastly, with driven oscillators it can take a period of time for periodic motion to be reached. Often there is a beginning period of time where the motion is random, this is called transient motion, transient as it only lasts a finite period of time until it collapses into periodic motion.
In this exercise I introduced various driving forces (separately) and examined the dynamics of the pendulum using phase portraits.

2) Methodology

2.1) Exercise 1 - Linear Pendulum Analysis

1) Using the simplified version of the pendulum EOQ's, we alter the equation:
   $F(\theta, \omega, t) \; =- \; (g/l)Sin\theta \; - \; k\omega \; + \; Acos\phi t$
   Replacing Sinθ with θ, k=0 , phi=0.6667, and A=0. This gives us a simple pendulum without a dampening or driving force.
2) Next I created a nsteps variable and a time step of 0.1
3) The lab manual provides us already with the python transcription of the Trapezoidal rule for solving ODE's.
   ```
   k1a = dt * omega k1b = dt * f(theta, omega, t)
   k2a = dt * (omega + k1b)
   k2b = dt * f(theta + k1a, omega + k1b, t + dt)
   theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
   ```
4) Using a for loop we can iterate this code 1000 times.
5) In this section I produced 2 graphs (1[a],1[b])
   1[a] shows Theta and Omega graphed against nsteps, this is to verify that these variables are sinusoidal as the function iterations progress.
   1[b] shows Theta and Omega graphed against time.
6) By changing the starting values we can produce some interesting graphs as shown in the results section

2.2) Exercise 2 -Solving the nonlinear pendulum

1) By using the same code as in Ex1 but changing the θ to sinθ, we can create a nonlinear pendulum.
2) I ran this nonlinear pendulum function along with the linear pendulum function and graph each in order to compare their dynamics.

2.3) Exercise 3- Runge-Kutta integration scheme

1) The lab manual provided me with the Python code for the variables involved in the integration scheme. (ref.Tcd Computation Lab Manual #2)
   ```
   k1a = dt * omega k1b = dt * f(theta, omega, t)
   ```

```
k2a = dt * (omega + k1b/2)
k2b = dt * f(theta + k1a/2, omega + k1b/2, t + dt/2)
k3a = dt * (omega + k2b/2)
k3b = dt * f(theta + k2a/2, omega + k2b/2, t + dt/2)
k4a = dt * (omega + k3b)
k4b = dt * f(theta + k3a, omega + k3b, t + dt)
theta = theta + (k1a + 2*k2a + 2*k3a + k4a) / 6
omega = omega + (k1b + 2*k2b + 2*k3b + k4b) / 6
t = t + dt
```

2) In the same way as the previous exercise I implemented this code to graph our Omega and theta against time.

(2.4)Exercise 4- Damped Pendulum

1) The code for a damped pendulum was already included by coding the function: $F(\theta, \omega, t) \ =- \ (g/l)Sin\theta - k\omega + Acos\phi t$
   (k is already included in the integration schemes seen above, so no new code needed)
2) Previously we had k=0 which nulled our damping term.
3) By changing to k=0.5 the oscillator is dampened. I set the initial value of theta to 3.0

(2.5)Exercise 5- Driven & Damped Pendulum

1) The driving force was added by changing 0.0 to 0.9. As the term is $Acos(\phi t)$ we get a periodic driving force and thus ‘interference’ with our pendulum motion depending on A’s value
2) In order to eliminate the transient motion I needed to keep track of the number of iterations the function has taken, so as to skip the for 1000 or so.
3) This was done by introducing an iteration_number
4) By using a while loop I started my graph once the function reached iteration number 5000
5) I graphed the phase portrait for different values of A
6) k=0.5 and phi=0.667

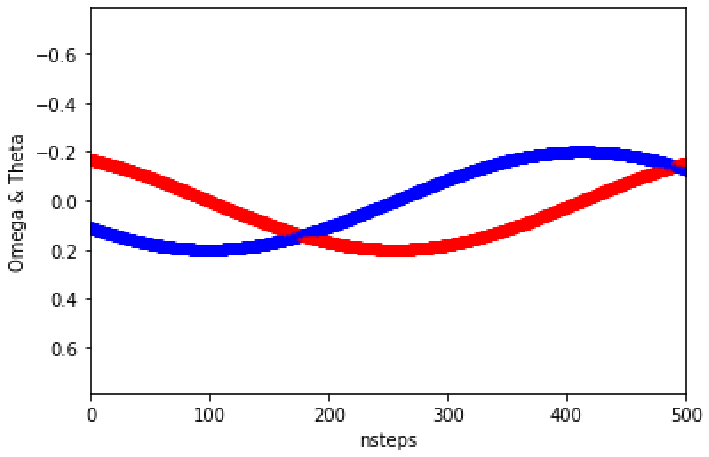
3) Results

All programs ran successfully.

3.1) Exercise 1 - Linear Pendulum Analysis

-1[a] (below) shows the value of the variables θ & ω as the number of iterations of our function (nsteps) increases.

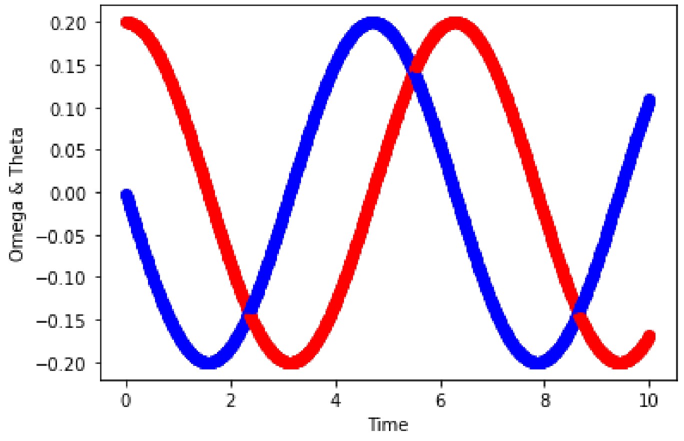
We can verify from the graph that these values are sinusoidal.



Red = Theta
Blue = Omega

1[a]

The graph 1[b] (below) shows the values of θ & ω over time. We see that the variable Omega is Sinusoidal whereas Theta follows a cosine function.

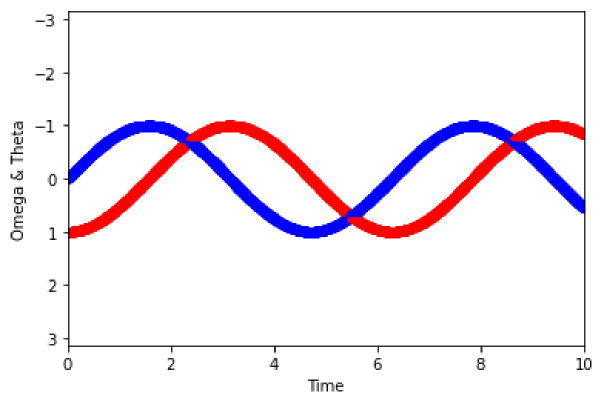


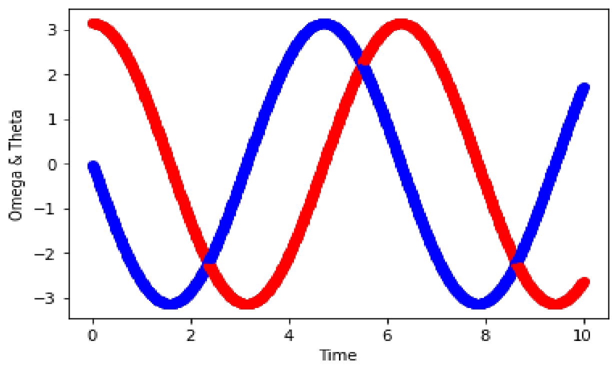
Red = Theta
Blue= Omega

Theta(0)=0.2
Omega(0)=0

1[b]

Red line is theta and Blue is Omega for all below:
The starting values for ω & θ are altered as described under each graph

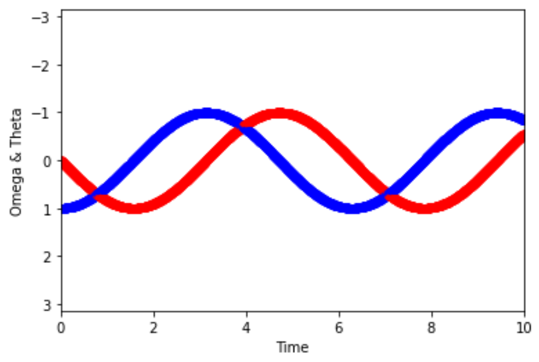




-

1[c] Theta(0)=1 Omega(0)=0

1[d] Theta(0)=3.14 Omega(0)=0



1[e] Theta(0)=3.14 Omega(0)=1

3.2)Exercise 2- Nonlinear Pendulum:

Nonlinear Pendulum: 2[a]

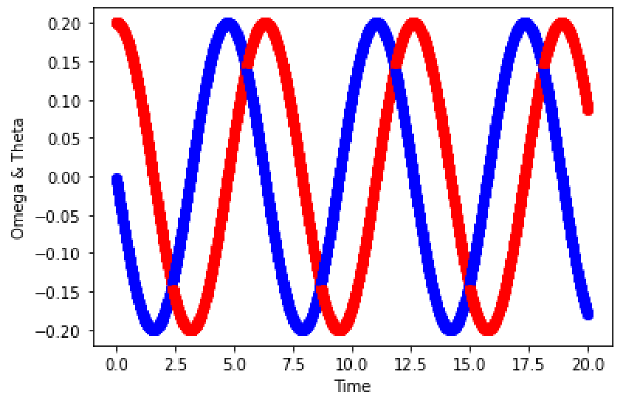
Blue-Omega

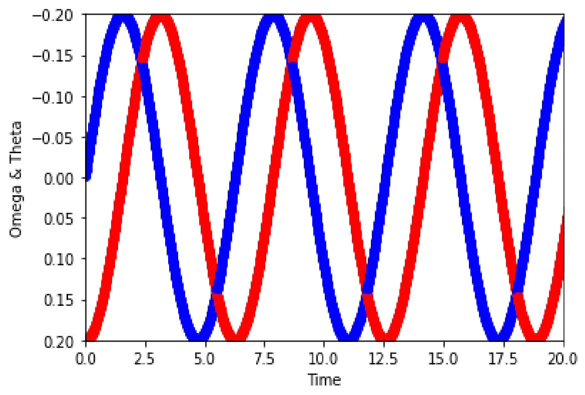
Red-Theta

Linear Pendulum: 2[b]

Blue-Omega

Red-Omega

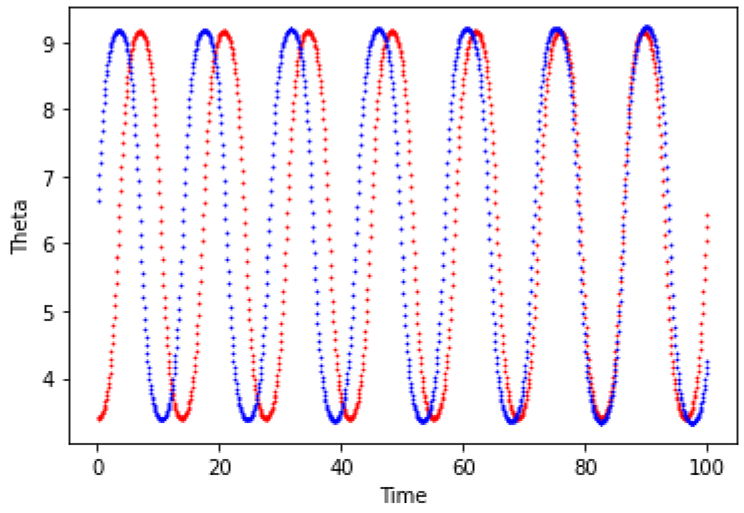




We can see clearly the Linear and nonLinear values are out of phase. Increasingly so as time progresses.

3.3)Exercise 3

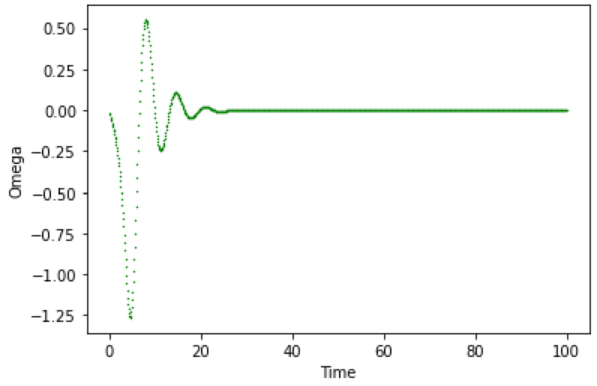
Theta vs Time for the Runge-Kutt method and the Trapezoidal method
Blue- Trapezoidal Method
Red- RK method

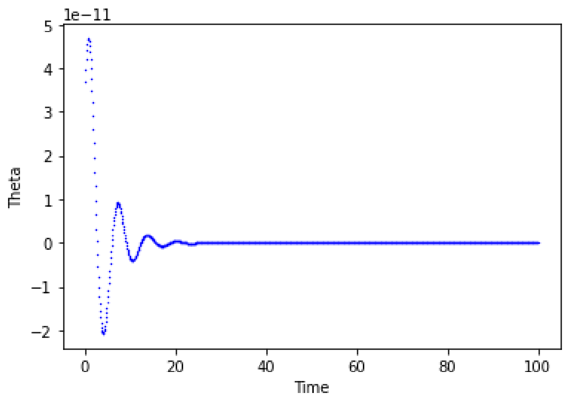


We see the difference between the integration methods garner similar results, yet the functions are still out of phase.

3.4) Exercise 4

Omega vs. Time *4[a]* *Theta vs Time* *4[b]*

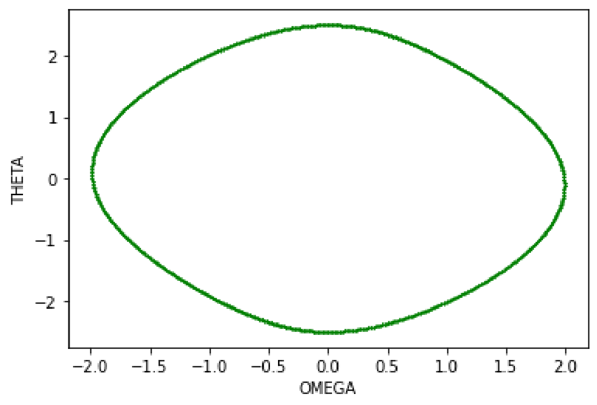




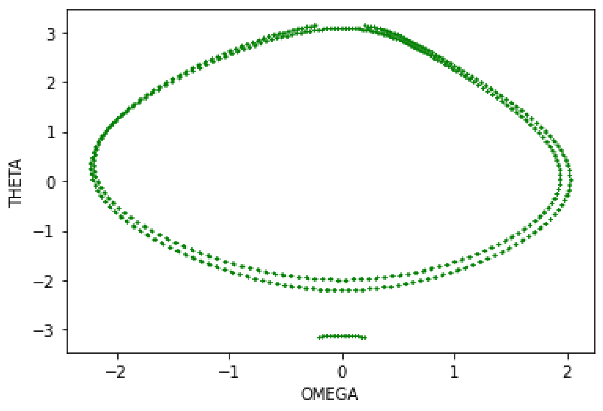
We see the damping term reduces the amplitude over time until the motion stops. This is consistent with real world motion where damping forces like air resistance bring pendulums to a stop.

3.5) Exercise 5- Driven Pendulum

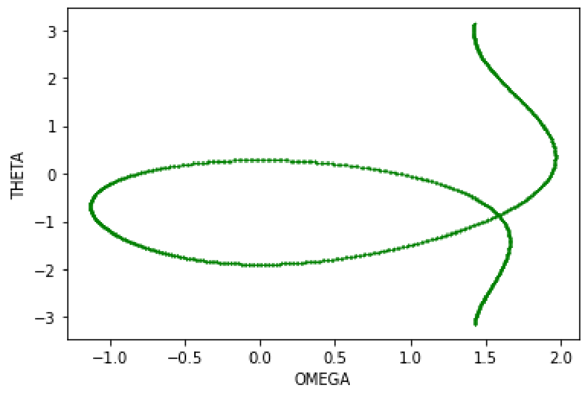
A=0.9 *5[a]*



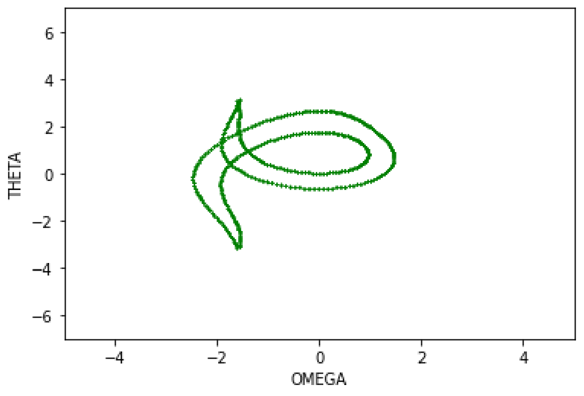
A=1.07 5[b]



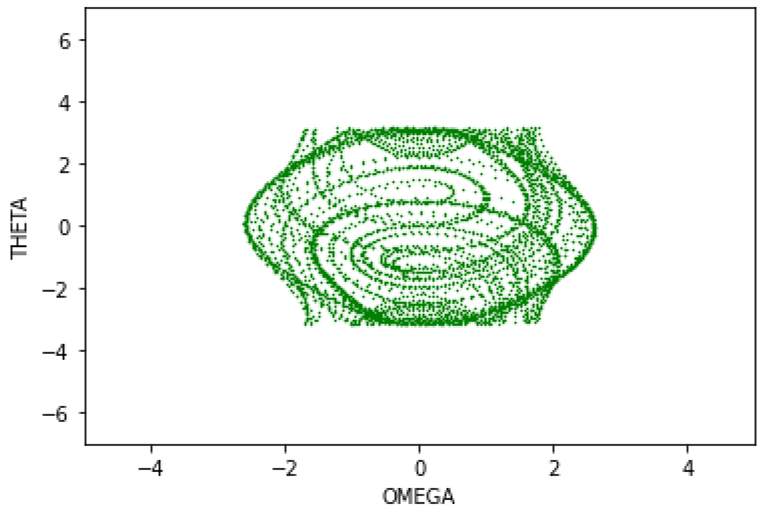
A=1.35 5[c]



A=1.47 5[d]



A=1.5 5[e]



We see the path depends heavily on the value of A.

4) Conclusion

From the first exercise the graphs for the linear pendulum showed us that in the progression of the pendulum the values of ω & θ were sinusoidal in shape. Yet was θ shifted by a factor of π . By altering the initial theta value we just change the amplitude of the function. Whereas is we change the omega value (eg.in 1[d]), we see theta is shifted by $\pi/2$ and omega is shifted by π .
Exercise 2 shows us that the linear approximation of a pendulum is not accurate especially as t grows larger. We found this by comparing the values of the two functions graphically.
In Exercise 3 I switch the trapezoidal method for the Runge-Kutt method for integration. The difference between the functions was striking especially for smaller values of t. Yet the difference was minimal as time progressed in my graph.
Exercise 4 introduces a damping force to our function. This affected my graphs by reducing the amplitude of the theta and omega functions. This corresponds to a slowing pendulum bob and a shortening of amplitude over time. This is consistent with real world pendulum motion where damping forces like air resistance or friction reduce amplitude of motion until the pendulum comes to a full stop. These results show my function is accurately emulating the SMH of a pendulum.
For Exercise 5 a driving force was introduced. As discussed in the introduction the driving force we introduce is periodic. Therefore there is ‘interference’ between it and the pendulum's regular motion. Depending on how the frequencies of the driving force and the pendulum line up, determines the dynamics of the system. If the frequencies match we get regular and periodic motion as seen in figures 5[a] & 5[b], these were likely cases of period doubling. But of these for figure 5[e] we can clearly see the motion was not periodic but rather chaotic.

-Apologies, the resolution of some of my graphs is poor as my computer is quite slow, so I had to lower the amount of data points.

References:
[1] Tcd Computational Lab manual, Lab #2, The Nonlinear Pendulum
[2] https://www.programiz.com/python-programming/methods/built-in/help
[3] https://en.wikipedia.org/wiki/Trapezoidal_rule_(differential_equations)
[4]https://matplotlib.org/stable/tutorials/index
[5]https://www.geeksforgeeks.org/runge-kutta-2nd-order-method-to-solve-differential-equations

Source Code:

Ex1:

```
import numpy as np
import matplotlib.pylab as plt
theta = 0.2
omega = 0.0
t = 0.0
dt = 0.01
nsteps = 0
g=9.81
l=1
#simplified linear pendulum
def f(theta ,omega ,t):
    k=0
    A=0
    phi=0.66667
    return
-theta-k*omega-k*omega+A*t
    #Cos(0.6667)=1

print (f(theta ,omega ,t))
nsteps=0
#for loop using the trap method from
lab manual
for i in range (10000):
    k1a = dt * omega
    k1b = dt * f(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * f(theta + k1a, omega +
k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt #time step iteration
    nsteps+=1
    plt.scatter(t,theta,color='red')
#points upon each iteration
    plt.scatter(t,omega,color='blue')
    plt.xlabel('Time')
    plt.ylabel('Omega & Theta')
    plt.xlim(0,20)
    plt.ylim(0.20,-(0.20))
plt.show()
#For loop for second plot
(theta&omega vs nsteps)
for nsteps in range (1000):
    k1a = dt * omega
    k1b = dt * f(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * f(theta + k1a, omega +
k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2

    plt.axis([0,500,-np.pi,np.pi])
    plt.scatter(nsteps,theta,color='red')

plt.scatter(nsteps,omega,color='blue'
)
    plt.xlabel('nsteps')
    plt.ylabel('Omega & Theta')
    plt.xlim(0,500)
    plt.ylim(0.25*np.pi,-0.25*(np.pi))
#to make graph clearer
    t = t + dt
plt.show()
```

Ex 2:

```
import numpy as np
import matplotlib.pylab as plt
theta = 0.2
omega = 0.0
t = 0.0
dt = 0.01
nsteps = 0
g=9.81
l=1
#we define both linear and non-linear for
comparision
def f(theta ,omega ,t):
    k=0
    A=0
    phi=0.66667
    return
-np.sin(theta)-k*omega+A*np.cos(phi*t)
    #Cos(0.6667)=1
    #linear
def f_linear(theta ,omega ,t):
    k=0
    A=0
    phi=0.66667
    return -theta-k*omega-k*omega+A*t

print (f(theta ,omega ,t))
nsteps=0
#for loop as in ex 1
for i in range (1000):
    k1a = dt * omega
    k1b = dt * f(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * f(theta + k1a, omega + k1b, t
+ dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps+=1
    plt.scatter(t,theta,color='red')
    plt.scatter(t,omega,color='blue')
    plt.xlabel('Time')
    plt.ylabel('Omega & Theta')

#second for loop for second plot

for i in range (1000):
    k1a = dt * omega
    k1b = dt * f_linear(theta, omega, t)
#referencing linear function instead
    k2a = dt * (omega + k1b)
    k2b = dt * f_linear(theta + k1a, omega +
k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps+=1
    plt.scatter(t,theta,color='red')
    plt.scatter(t,omega,color='blue')
    plt.xlabel('Time')
    plt.ylabel('Omega & Theta')
plt.show()
```

Ex 3:

```
import numpy as np
import matplotlib.pylab as plt
theta = 3.4
omega = 0.0
t = 0.0
t1=0.0
dt = 0.1
dt1=0.1
nsteps = 0
g=9.81
l=1
#non linear pendulum
def f(theta ,omega ,t):
    k=0
    A=0
    phi=0.66667
    return -np.sin(theta)-k*omega+A*np.cos(phi)*t
    #Cos(0.6667)=1
print (f(theta ,omega ,t)) #print our result given
intial values
nsteps=0
#ploting theta & omega vs time for nonlinear
for i in range (1000):
   k1a = dt * omega
   k1b = dt * f(theta, omega, t)
   k2a = dt * (omega + k1b/2)
   k2b = dt * f(theta + k1a/2, omega + k1b/2, t +
dt/2)
   k3a = dt * (omega + k2b/2)
   k3b = dt * f(theta + k2a/2, omega + k2b/2, t +
dt/2)
   k4a = dt * (omega + k3b)
   k4b = dt * f(theta + k3a, omega + k3b, t + dt)
   theta = theta + (k1a + 2*k2a + 2*k3a + k4a) / 6
   omega = omega + (k1b + 2*k2b + 2*k3b + k4b)
/ 6
   t = t + dt
   nsteps+=1 #unecessary for this ex
   plt.scatter(t,theta,color='red',s=0.75)
   #plt.scatter(t,omega,color='blue')

   #theta & omega vs time for linear
for ilinear in range(1000):
    k1a = dt * omega
    k1b = dt * f(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * f(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t1 = t1 + dt1
    nsteps+=1
    plt.scatter(t1,theta,color='blue',s=0.75)
    plt.xlabel('Time')
    plt.ylabel('Theta')
```

Ex 4:

```
import numpy as np
import matplotlib.pylab as plt
theta = 3.0
omega = 0.0
t = 0.0
t1=0.0
dt = 0.1
dt1=0.1
nsteps = 0
g=9.81
l=1
def f(theta ,omega ,t):
    k=0.5 #introduction of dampening force
    A=0
    phi=0.66667
    return -np.sin(theta)-k*omega+A*np.cos(phi*t)
    #Cos(0.6667)=1

print (f(theta ,omega ,t))
nsteps=0
#exact same as previously but our dampening for is not 0
for i in range (1000): #1000 data points
   k1a = dt * omega
   k1b = dt * f(theta, omega, t)
   k2a = dt * (omega + k1b/2)
   k2b = dt * f(theta + k1a/2, omega + k1b/2, t + dt/2)
   k3a = dt * (omega + k2b/2)
   k3b = dt * f(theta + k2a/2, omega + k2b/2, t + dt/2)
   k4a = dt * (omega + k3b)
   k4b = dt * f(theta + k3a, omega + k3b, t + dt)
   theta = theta + (k1a + 2*k2a + 2*k3a + k4a) / 6
   omega = omega + (k1b + 2*k2b + 2*k3b + k4b) / 6
   t = t + dt
   nsteps+=1
   #plt.scatter(t,theta,color='red')
   plt.scatter(t,omega,color='green',linewidths=0.7,s=0.5)
#theta on spearte graph
   #this time
   plt.xlabel('Time')
   plt.ylabel('Omega')
plt.show()
   #function to plot omega
for ilinear in range(1000): #1000 data points
    k1a = dt * omega
    k1b = dt * f(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * f(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t1 = t1 + dt1
    nsteps+=1
    plt.scatter(t1,theta,color='blue',linewidths=0.7,s=0.5)
    #plt.scatter(t1,omega,color='green')
    plt.xlabel('Time')
    plt.ylabel('Theta')
plt.show()
```

Ex 5:

```
import numpy as np
import matplotlib.pylab as plt
theta = 3.0
omega = 0.0
t = 0.0
t1=0.0
dt = 0.1
dt1=0.1
nsteps = 0
g=9.81
l=1
iteration_number=0
transient=5000
def f(theta ,omega ,t):
    k=0.5
    A=1.5 #driving for and dampening force this
time
    phi=0.66667
    return -np.sin(theta)-k*omega+A*np.cos(phi*t)
    #Cos(0.6667)=1

print (f(theta ,omega ,t))
nsteps=0
#only one grapgh being produced as we are only
interested in the
#phase portraits of each A value
for i in range (10000): #10,000 data points
   k1a = dt * omega
   k1b = dt * f(theta, omega, t)
   k2a = dt * (omega + k1b/2)
   k2b = dt * f(theta + k1a/2, omega + k1b/2, t +
dt/2)
   k3a = dt * (omega + k2b/2)
   k3b = dt * f(theta + k2a/2, omega + k2b/2, t +
dt/2)
   k4a = dt * (omega + k3b)
   k4b = dt * f(theta + k3a, omega + k3b, t + dt)
   theta = theta + (k1a + 2*k2a + 2*k3a + k4a) / 6
   omega = omega + (k1b + 2*k2b + 2*k3b + k4b)
/ 6
   t = t + dt
   iteration_number+=1
   #the code below was provided by the lab
maunal and it is to
   #ensure the program can regonise radian
values, such that
   # theta + 2pi = theta
   if (np.abs(theta) > np.pi):
      theta -= 2 * np.pi * np.abs(theta) / theta
      #this if function ensures the function only
starts ploting after
      #5000 iterations to elimate transient motion
   if iteration_number>transient:
       #plt.scatter(t,theta,color='red')

plt.scatter(omega,theta,color='green',linewidths=
0.7,s=0.5)
       plt.xlabel('OMEGA')
       plt.ylabel('THETA')
       plt.xlim(-5,5)
       plt.ylim(-7,7)
plt.show()
```