Senior Fresh Computational Physics Laboratory Report

# Laboratory 2: The Pendulum

Conor Kirby

October 28, 2023

# Contents

# 1 Abstract

The purpose of this experiment is to explore the dynamics of systems with nonlinear force laws and use Python to solve ordinary differential equations (ODEs). Initially we start off by solving the simple linear pendulum using the trapezoidal method.

Plotting theta and omega vs time for the damped nonlinear pendulum we can immediately see the effect of the damping force. In seconds the motion of the pendulum is almost negligible. We solve the nonlinear pendulum equations using both the trapezoidal method and the Runge-Kutta integration algorithm. We could conclude that the Runge-Kutta numerical integration system was more effective especially in solving higher order differential equations (in this case second and forth order). This method was more efficient in solving both the nonlinear pendulum equations and the damped, driven nonlinear pendulum equations.

However things start to get interesting once we start to investigate the damped, driven nonlinear pendulum. Up until a driving force amplitude of about 1.45 it's motion is predictable however once we hit an amplitude of 1.5 chaotic motion incurs. We can conclude this from the phase portrait graphs; plotting values of theta vs corresponding values of omega. Omega being the angular velocity and theta being the angular displacement.

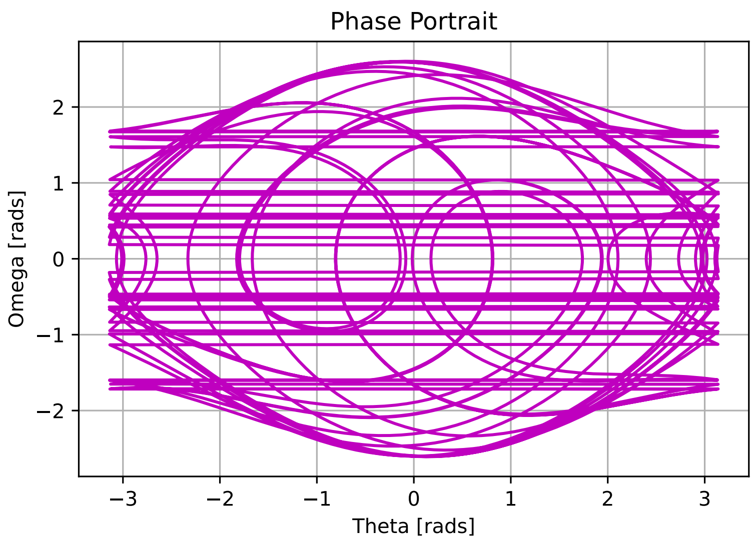

Figure 1.1: Phase Portrait for a nonlinear damped driven pendulum with a driving force amplitude of 1.5. Chaotic motion can clearly be seen.

# 2 Introduction and Theory

Pendulums have intrigued me and scientists around the world for years because of their elegant oscillatory behaviour. In this lab experiment we investigate three types of pendulums; the linear pendulum, the nonlinear pendulum, and the damped, driven nonlinear pendulum.

The equations of motion of the pendulum turn linear for small angles, i.e. we take the small angle approximation of $sin\theta \approx \theta$. However if this is not the case then the equations of motion are nonlinear because sin theta is nonlinear.

The dynamics of the nonlinear pendulum are more complex than that of the linear pendulum. And even more complex once we add the components which account for the driving and damping forces. The first order equations of motion for the damped, driven nonlinear pendulum are as follows: [1]

$$\frac{d\theta}{dt} = \omega \tag{2.1}$$

$$\frac{d\omega}{dt} = f(\theta, \omega, t) \tag{2.2}$$

$$f(\theta, \omega, t) = -\frac{g}{L} sin\theta - k\omega + Acos\phi t \tag{2.3}$$

Where A is the amplitude of the driving force, $\phi$ is the frequency and -kw is the damping term. For simplicity we choose g/L to equal 1. K and A are zero until we are investigating when the pendulum is damped and driven.

In this experiment we have a driving force acting on our pendulum. This is like a continuously acting pushing force that can act in the positive or negative directions. Our main focus are the plots of omega vs theta, otherwise known as the phase portrait. Phase portraits are an invaluable tool in understanding complex dynamical systems allowing us to visualise the behaviour of ordinary differential equations.

The driving force has a frequency of $\phi$ and a period of $2\pi/\phi$. The motion of the pendulum will remain periodic if it has the same period of the driving force or an integer multiple of the driving force. However if this is not the case the motion of the pendulum will become aperiodic and we will observe chaotic motion. Chaotic motion means that the motion is deterministic however it cannot be predicted.

When we are plotting our phase portraits we want to neglect the motion before the steady state, periodic motion. This unwanted motion is called the *transient* motion. To do this we must start collecting data only after a certain number of iterations... 5000 for instance. Then we want to start collecting the values of $\theta$ and $\omega$. How do we know if we have no transient motion? The transient motion on the phase portrait is the tail on the loop, so we skip 5000 iterations and there is still a ’tail’, then we must begin collecting data at a higher iteration number.

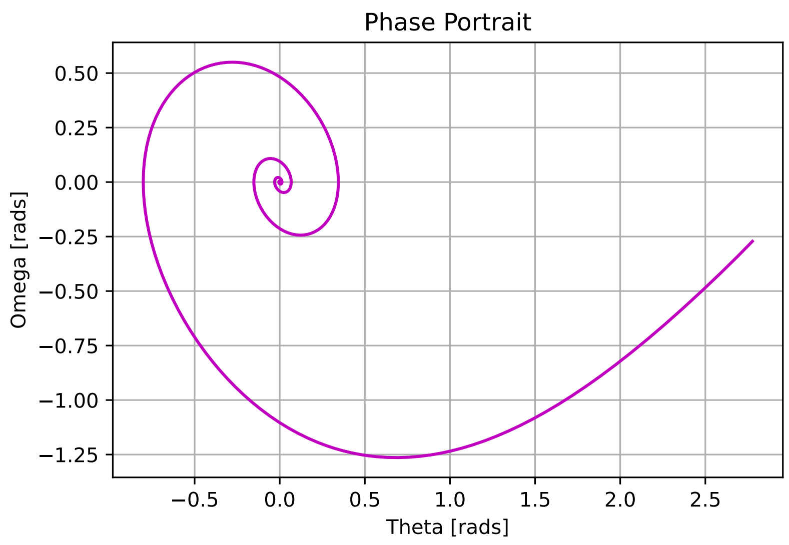

Figure 2.1: The tail in this phase portrait of a damped pendulum is the transient motion we are trying to eliminate.

Our first numerical integration system is using the trapezoidal rule which works by approximating the region below the graph in a certain interval using the area of a trapezoid.

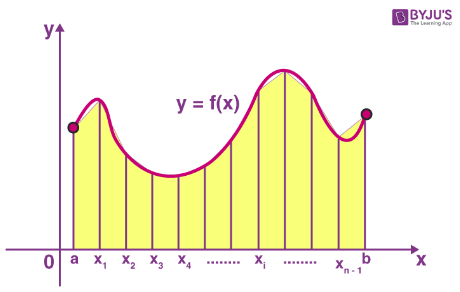

Figure 2.2: Trapeziodal Rule [3]

By this method we get ODEs for theta and omega which are as follows: [1]

$$\theta_{n+1} = \theta_n + \omega_n \Delta t/2 + (\omega_n + f(\theta_n, \omega_n, t)\Delta - t)\Delta t/2 \tag{2.4}$$

$$\omega_{n+1} = \omega_n + f(\theta_n, \omega_n, t)\Delta t/2 + f(\theta_{n+1}, \omega_n + f(\theta_n, \omega_n, t)\Delta t, t + \Delta t)\Delta t/2 \tag{2.5}$$

The second numerical integration system we use is the Runge-Kutta, which behaves similarly to the trapezoidal rule up until the second order. However the main difference is that it carries out the Taylor Series Expansion about the middle of the time step which allows it to be more accurate.

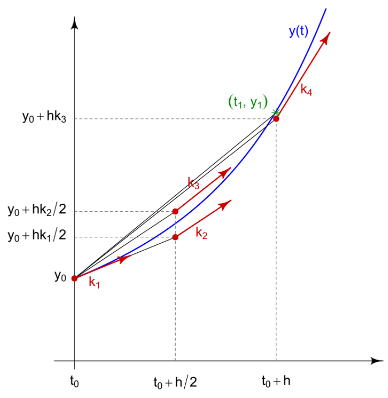

Figure 2.3: Runge-Kutta Integration Method [4]

## 3 Methodology

In this lab we use our numerical analysis skills to attempt to understand the complex motion of the pendulum under different circumstances. Lucky for us, python is fantastic for understanding and visualising data. In this case we will be using Spyder in Anaconda as the programming environment and using numpy and matplotlib.pyplot [2] libraries for mathematical expressions and plotting.

### 3.1 Solving The Linear Pendulum Equation

Firstly we must define a function to using equation 2.3. For simplicity we set our variable $\phi$ to $\frac{2}{3}$. To neglect the driving force we set A to 0.0 and to get rid of the damping force we set k to 0.0. Also as we have mentioned in the theory section we choose that $\frac{g}{L}$ are in unison (equal to 1).
Initialise our variables as such:

```
theta = 0.2
omega = 0.0
t = 0
dt = 0
nsteps = 0
```

Listing 1: Initialising Variables

***Sidenote:*** *Keep using print commands throughout the lab so that you can catch a mistake and fix it early.*

Create a for loop to run for 1000 iterations using the following code to implement equations 2.4 and 2.5: [1]

```
    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
```

Listing 2: Trapezoidal Rule

**Number of Iterations:** Use the variable nsteps that we initialised earlier to count the number of iterations in the for loop by adding one each time the loop runs.

**Plotting:** Now plot $\omega$ and $\theta$ vs time on the same graph. To fit the plot nicely on the graph use the `plt.axis()` [2] to adjust the range if values on the x and y axes.

Now to attempt to gain a greater understanding of the effect of theta and omega on the plots try varying theta between 0 and 3.14 while omega is 0, and then set omega to 1 and theta to 0 and see what happens.

Make sure you put `plt.show()` [2] at the end of your code to see the final plot.

## 3.2 The Nonlinear Pendulum Equation

Create a new project file naming it appropriately and create a copy of your code from exercise 1. As mentioned in the theory, the motion becomes non linear if the angle $\theta$ is not small enough for us to take the small angle approximation.

Therefore we should change our function such that it is identical to equation 2.3 so we change $\theta \rightarrow sin\theta$ in our code.

Now plot $\theta$ and $\omega$ vs time using the same values that we used in the last section and compare the graphs.

## 3.3 Using The Runge-Kutta Integration Algorithm

Create a copy of the nonlinear pendulum code naming and commenting it appropriately. To change the numerical integration algorithm we are going to have to change the code within the for loop.
To use the Runge-Kutta method replace the code implement the trapezoidal rule with this code: [1]

```
    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
```

Listing 3: Runge-Kutta Integration Algorithm

To check that our code is working properly we should use both the trapezoidal rule and Runge-Kutta algorithm and plot theta vs time for each method on the same graph. If our new method is correct the plots should exactly overlap each other.

## 3.4 The Damped Nonlinear Pendulum

Create a copy of the code from exercise 4 and again we should name and comment the file appropriately. To introduce the damping term we must assign a non-zero value to k. Let us set k to 0.5. This damping force will subject the pendulum to friction forces which opposes the motion and cause it to slow down and ultimately stop oscillating.

Use the same methods as previous sections to plot theta vs time and omega vs time. Observe the effect the damping force has on the motion. If it is not immediately obvious to you, increase the number of iterations in the for loop and widen the range of values on the x-axis using the `plt.axis()` [2] function.

Set $\theta$ to 3.0 and $\omega$ to 0.0 and execute your code. Analyse the graphs and try to explain the shapes of the theta and omega vs time curves.

## 3.5 The Damped, Driven Nonlinear Pendulum

Start off by creating a copy of your code from you damped nonlinear pendulum, changing the file name and comments to avoid confusion. To introduce the driving force change the value of A(the amplitude of the driving force) from 0.0 to 0.9. Keeping k at 0.5 and

$\phi$ at $\frac{2}{3}$.

Initialise variables `iteration_number = 0` and `transient = 5000` at the start of the script. These will be used to tell the code to only start collecting values of theta and omega once `iteration_number` hits the transient value. Do this by incrementing `iteration_number` by one each time the for loop runs through. Use an if loop to start collecting data once `iteration_number` hits the transient value.

Run the code for the following parameters and in each case try to understand what the phase portrait says about the pendulum motion. `A = 0.90, 1.07, 1.35, 1.47 and 1.5` while keeping k and $\phi$ fixed at 0.5 and $\frac{2}{3}$ respectively.

For some of the above parameters you'll notice that theta increases indefinitely, hence we have circular motion. The problem is that the angle of theta cannot be distinguished from the angle of theta + $2\pi$. To fix this we must restrict theta to between $-\pi$ and $\pi$ the following code must be included: [1]

```
    if (np.abs(theta) > np.pi):
        theta -= 2 * np.pi * np.abs(theta) / theta
```

You will find that in some cases you may need to change the range by a scalar `a` so that we can obtain a limit cycle as a single piece. ie. we must find a way to adjust the range from $[-\pi, \pi] \rightarrow [-\pi + a, \pi + a]$.

## 4 Results

### 4.1 The Linear Pendulum

In exercise 1 we investigated the motion of the linear pendulum. The equations of motion became linear when we take a small angle approximation for theta. Plotting $\theta$ vs time and $\omega$ vs time for a range of parameters we can see the effect of each on the pendulums motion.

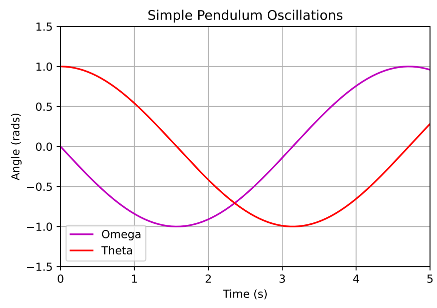

Figure 4.1: Linear Pendulum - $\theta = 1.0$, $\omega = 0.0$

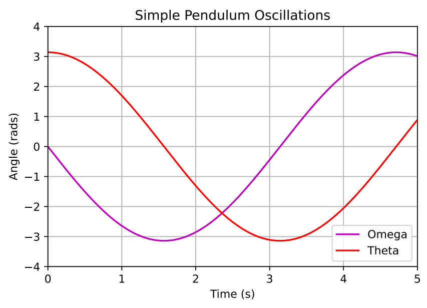

Figure 4.2: Linear Pendulum - $\theta = 3.14$, $\omega = 0.0$

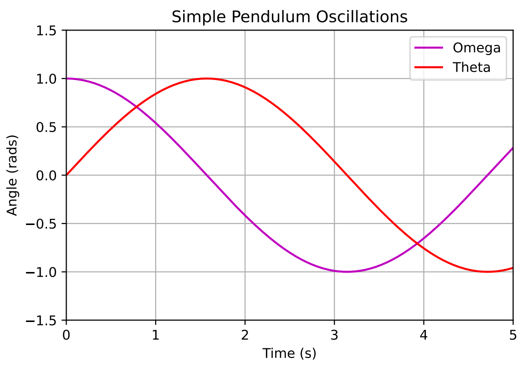

Figure 4.3: Linear Pendulum - $\theta = 0.0$, $\omega = 1.0$

From these plots we can see the sinusoidal periodic motion of the linear pendulum when there are no driving or damping forces. In exercise 2 we compare these plots with the plots for the nonlinear pendulum under the same parameters.

## 4.2 The Nonlinear Pendulum

In this exercise we change our function so that the motion of the pendulum is nonlinear and plot the same graphs as we did above to see the effect of the nonlinearity.

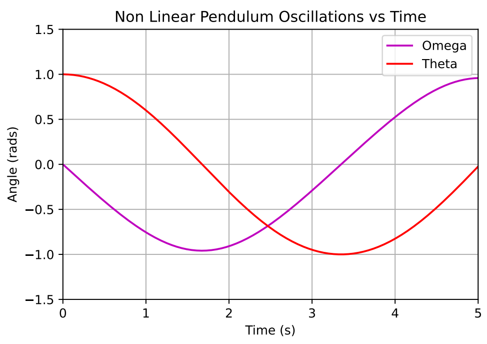

Figure 4.4: Nonlinear Pendulum - $\theta = 1.0$, $\omega = 0.0$

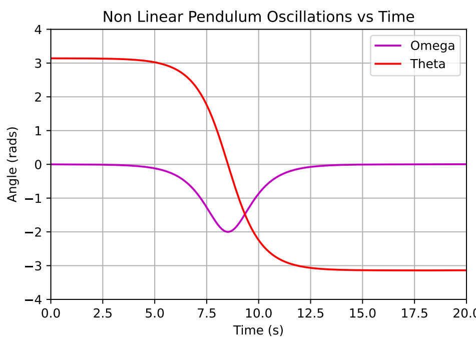

Figure 4.5: Nonlinear Pendulum - $\theta = 3.14$, $\omega = 0.0$

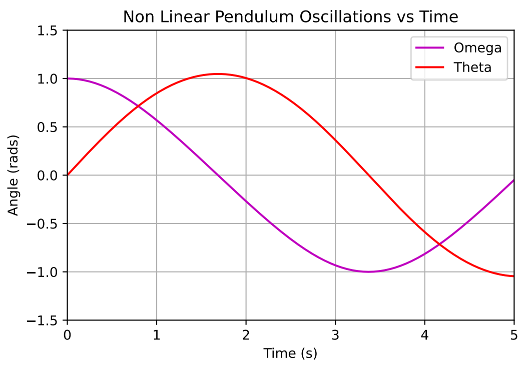

Figure 4.6: Nonlinear Pendulum - $\theta = 0.0$, $\omega = 1.0$

Ignoring the elephant in the room, all that changes with the plots are the periods of the functions and the corresponding amplitude of the omega function is ever so slightly smaller than above. It is clear that something changes for figure 4.5 but what is the cause? Since $sin\pi = 0$, $k = 0$ and $A = 0$ we can see that equation 2.3 is zero. So when the $\theta = \pi$, $\omega = 0$ and when $\theta = 0$, $\omega$ is at a max.

## 4.3 Changing to the Runge-Kutta Numerical Integration System

We do not determine much from this exercise however we must ensure that our new numerical integration method. Below is our plot to test the Runge-Kutta Algorithm.

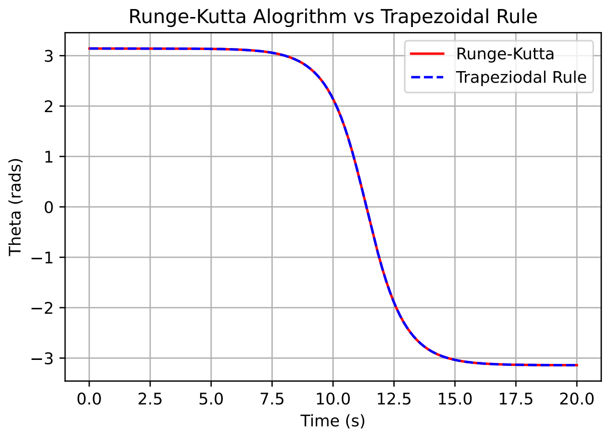

Figure 4.7: Runge-Kutta and Trapezoidal Rule vs Time

As you can see our plots overlap concluding that our new method works properly.

## 4.4 The Damped Nonlinear Pendulum

Turning on the damping force we plot theta vs time and omega vs time with initial values of theta = 3 and omega = 0.

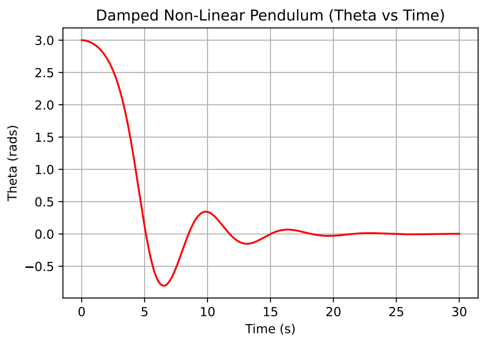

Figure 4.8: Damped Nonlinear Pendulum - $\theta_0 = 3.0$

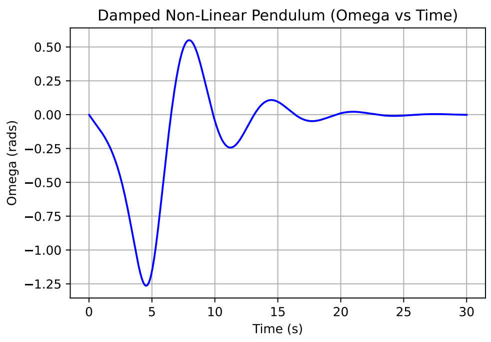

Figure 4.9: Damped Nonlinear Pendulum - $\omega_0 = 0.0$

In figure 4.9 we can see the bit of sinusoidal periodic motion however the damping force disturbs this motion immediately. with every period there maximums and minimums of theta become smaller and smaller until theta is almost negligible.

Figure 4.9 shows a large initial max of omega however the effect on theta directly affects omega and we can easily see the dampening of the omega vs time wave very quickly.

## 4.5 The Damped, Driven Nonlinear Pendulum

Plotting the phase diagrams for multiple values of A and hence changing the intensity of the driving force, we can see it's effect on the motion of this nonlinear damped driven pendulum.

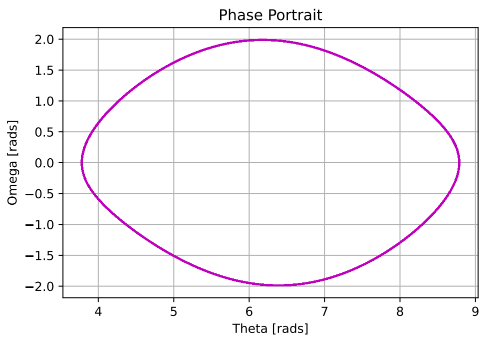

Figure 4.10: Phase Diagram for A = 0.9

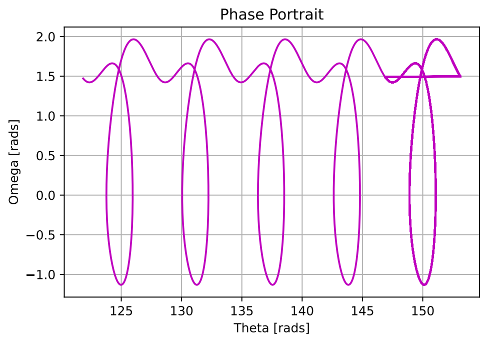

Figure 4.11: Phase Diagram for A = 1.35

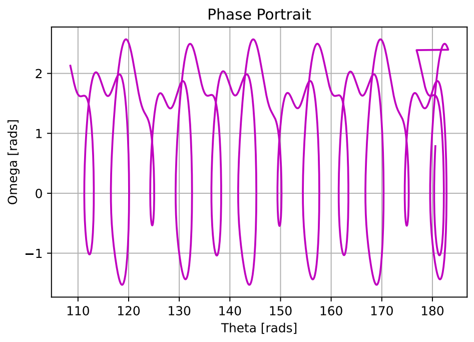

Figure 4.12: Phase Diagram for A = 1.47, even though it is starting to look aperiodic we can actually see that the pattern repeats about every 25 radians in the angle theta.

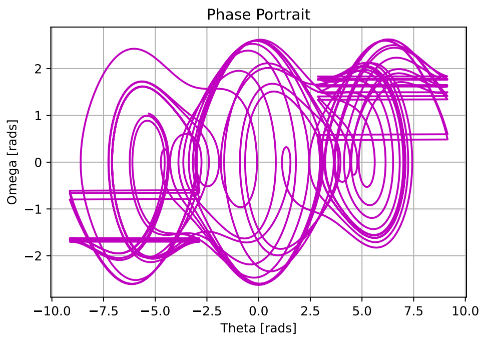

Figure 4.13: Phase Diagram for A = 1.5, the motion has transitioned to a state of chaotic motion.

In figure 4.10 simple harmonic motion is clear because of the closed loop and repetitive periodic motion. Figures 4.11 and 4.12 is still periodic however the effects of the driving and damping are clear.

Initially we can see the predictable periodic motion in the phase portraits however once we reach a driving amplitude of 1.5 chaotic motion ensues and becomes completely unpredictable. This chaotic behavior arises due to the interplay between the periodic driving force and the dissipative effects of damping.

## 5 Conclusions

In this experiment we set out to gain further insight into the motion of the pendulum in four different situations; small angle linear motion, nonlinear motion, nonlinear motion with damping forces, and nonlinear motion with driving and damping forces.

Comparing linear motion to nonlinear motion the main differences were the change in period and the special case when theta was approximately equal to $\pi$ which is explained in the results.

Adding the damping force had an immediate effect on our pendulums motion, decreasing the maximum value of theta with ever oscillation until the motion was almost negligible. With the driving force the motion is still predictable for lower values of A, however in this case once the amplitude of the driving force reaches 1.5 the motion becomes chaotic and completely unpredictable.

Another key observation we can take away from this experiment is the efficiency of the Runge-Kutta integration scheme over the trapezoidal rule. For the second order, while the trapezoidal method takes the Taylor Expansion about the beginning, the Runge-Kutta method take it about the midpoint allowing for accuracy of $\Delta t^2$ rather than $\Delta t$. When we use the fourth order Runge-Kutta method it divides the time step into four rather than two.

This experiment clearly shows the functionality of numerical algorithms in python to gain further insight into the complex motion of a pendulum under many different circumstances.

## References

[1] Trinity College Dublin SF Computational Lab Manual 2023

[2] Matplotlib, python plotting library, https://matplotlib.org/stable/tutorials/index

[3] Wikipedia, Trapezoidal Rule, https://en.wikipedia.org/wiki/Trapezoidal_rule

[4] Wikipedia, Runge-Kutta Integration Algorithm, https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods

## 6 Appendix

### 6.1 Python script to solve the linear pendulum equation

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

    g = 9.8 #m/s
    L = 9.8 #m
    k = 0.0
    A = 0.0

    phi = 2/3
    theta = 1.0
    omega = 0.0

    t = 0.0
    dt = 0.01
    nsteps = 0

    def f(theta, omega, t):
        return (-g/L)*theta - k*omega + A*np.cos(phi*t)

    print(f(1,2,3))

    nsteps_list = []
    nsteps = 0
    theta_list = []
    omega_list = []
    t_list = []

    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
        t_list.append(t)

        theta_list.append(theta)
        omega_list.append(omega)

        nsteps += 1
        nsteps_list.append(nsteps)
    '''
    plt.plot(nsteps_list, omega_list, '-m')
    plt.plot(nsteps_list, theta_list, '-r')
    plt.title("Simple Pendulum Oscillations")
    plt.legend(["Omega", "Theta"])
    plt.xlabel("Number of Steps")
    plt.ylabel("Angle (rads)")
    plt.axis([0, 500, -1.5, 1.5])
    plt.grid()
    #plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/0.0,1.0.pdf')
    plt.show()
    '''
    plt.plot(t_list, omega_list, '-m')
    plt.plot(t_list, theta_list, '-r')
    plt.title("Simple Pendulum Oscillations")
    plt.legend(["Omega", "Theta"])
    plt.xlabel("Time (s)")
    plt.ylabel("Angle (rads)")
    plt.axis([0, 5, -1.5, 1.5])
    plt.grid()
    plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/Computational
   Lab/Lab 2/Lab 2 Figures/lin1.0,0.0time.pdf')
    plt.show()
```

## 6.2 Solving the nonlinear pendulum equation

```python
''' Exercise 2: Solving the non linear pendulum'''

import numpy as np
import matplotlib.pyplot as plt

g = 9.8 #m/s
L = 9.8 #m
k = 0.0
A = 0.0

phi = 2/3
theta = 1.0
omega = 0.0

t = 0.0
dt = 0.01
nsteps = 0

def f(theta, omega, t): #-k*omega is the dampening term
    return (-g/L)*np.sin(theta) - k*omega + A*np.cos(phi*t)

print(f(1,2,3))

nsteps_list = []
nsteps = 0
theta_list = []
omega_list = []
t_list = []

for i in range(2000):

    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

    t_list.append(t)

    theta_list.append(theta)
    omega_list.append(omega)

    nsteps += 1
    nsteps_list.append(nsteps)


    '''
    plt.plot(nsteps_list, omega_list, '-m')
    plt.plot(nsteps_list, theta_list, '-r')
    plt.title("Non Linear Pendulum Oscillations")
    plt.legend(["Omega", "Theta"])
    plt.xlabel("Number of Steps")
    plt.ylabel("Angle (rads)")
    plt.axis([0, 500, -1.5, 1.5])
    plt.grid()
    #plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/nonlin0.0,1.0.
   pdf')
    plt.show()
    '''
    plt.plot(t_list, omega_list, '-m')
    plt.plot(t_list, theta_list, '-r')
    plt.title("Non Linear Pendulum Oscillations vs Time")
    plt.legend(["Omega", "Theta"])
    plt.xlabel("Time (s)")
    plt.ylabel("Angle (rads)")
    plt.axis([0, 20, -1.5, 1.5])
    plt.grid()
    #plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/Computational
   Lab/Lab 2/Lab 2 Figures/nonlin1.0,0.0 time.pdf')
    plt.show()
```

## 6.3 Change the numerical integration algorithm

```
''''Exercise 3: Using the Runge Kutta Integration System'''

import numpy as np
import matplotlib.pyplot as plt

g = 9.8 #m/s
L = 9.8 #m
k = 0.0
A = 0.0

phi = 2/3
theta = 3.1415
omega = 0.0

t = 0.0
dt = 0.01

def f(theta, omega, t): #-k*omega is the dampening term
    return (-g/L)*np.sin(theta) - k*omega + A*np.cos(phi*t)
#print(f(1,2,3))

'''Runge-Kutta Algorithm'''

nsteps_list = []
```

```
    nsteps = 0
    theta_list = []
    omega_list = []
    t_list = []

    for i in range(2000):

        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
        t_list.append(t)

        theta_list.append(theta)
        omega_list.append(omega)

        nsteps += 1
        nsteps_list.append(nsteps)

    plt.plot(t_list, theta_list, '-r')
    plt.title("Runge-Kutta Alogrithm vs Trapezoidal Rule")

    plt.xlabel("Time (s)")
    plt.ylabel("Theta (rads)")
    plt.grid()


    '''Trapeziodal Rule Algorithm'''
    phi = 2/3
    theta = 3.1415
    omega = 0.0

    t = 0.0
    dt = 0.01

    nsteps_list = []
    nsteps = 0
    theta_list = []
    omega_list = []
    t_list = []
```

```python

    for i in range(2000):

        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

        t_list.append(t)

        theta_list.append(theta)
        omega_list.append(omega)

        nsteps += 1
        nsteps_list.append(nsteps)

    plt.plot(t_list, theta_list, '--b')
    plt.legend(["Runge-Kutta", "Trapeziodal Rule"])
    plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/Computational Lab/Lab 2/Lab 2 Figures/RK vs TrapRule time.pdf')
    plt.show()

    '''The algorithms plot identically proving that the new Runge-Kutta method works the same as the trapezoidal'''
```

## 6.4 The damped nonlinear pendulum

```python
'''Exercise 4: Dampening the Non-Linear Pendulum'''
import numpy as np
import matplotlib.pyplot as plt

g = 9.8 #m/s
L = 9.8 #m
k = 0.5
A = 0.0

phi = 2/3
theta = 3.0
omega = 0.0

t = 0.0
dt = 0.01

def f(theta, omega, t): #-k*omega is the dampening term
    return (-g/L)*np.sin(theta) - k*omega + A*np.cos(phi*t)
#print(f(1,2,3))
```

```

    '''Runge-Kutta Algorithm'''

    nsteps_list = []
    nsteps = 0
    theta_list = []
    omega_list = []
    t_list = []

    for i in range(3000):

        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
        t_list.append(t)

        theta_list.append(theta)
        omega_list.append(omega)

        nsteps += 1
        nsteps_list.append(nsteps)

    plt.plot(t_list, omega_list, '-b')
    plt.title("Damped Non-Linear Pendulum (Omega vs Time)")
    plt.xlabel("Time (s)")
    plt.ylabel("Omega (rads)")
    plt.grid()
    #plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/Computational
   Lab/Lab 2/Lab 2 Figures/DampedNonLin__Theta.pdf')
    plt.show()

    plt.plot(t_list, theta_list, '-r')
    plt.title("Damped Non-Linear Pendulum (Theta vs Time)")
    plt.xlabel("Time (s)")
    plt.ylabel("Theta (rads)")
    plt.grid()
    #plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/Computational
   Lab/Lab 2/Lab 2 Figures/DampedNonLin__Omega.pdf')
    plt.show()
```

## 6.5 The damped, driven nonlinear pendulum

```python
'''Exercise 5: The damped, driven non-linear pendulum'''

import numpy as np
import matplotlib.pyplot as plt

g = 9.8 #m/s
L = 9.8 #m
k = 0.5
A = 1.47

phi = 2/3
theta = 3
omega = 0.0

t = 0.0
dt = 0.01

def f(theta, omega, t): #-k*omega is the dampening term
    return (-1)*np.sin(theta) - k*omega + A*np.cos(phi*t)
#print(f(1,2,3))

'''Runge-Kutta Algorithm'''

iteration_number = 0
theta_list = []
omega_list = []
t_list = []
q= 180

transient = 18000

for i in range(30000):

    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

```

```
        if (np.abs(theta) > np.pi + q):
            theta -= 2 * (np.pi)* np.abs(theta) / theta

        if iteration_number >= transient:

            theta_list.append(theta)
            omega_list.append(omega)
            t_list.append(t)

        iteration_number += 1

    #print(theta_list, omega_list, newt)



    plt.plot(theta_list, omega_list, '-m')
    plt.title("Phase Portrait")
    plt.xlabel("Theta [rads]")
    plt.ylabel("Omega [rads]")
    plt.grid()
    plt.savefig('C:/Users/Conor Kirby/OneDrive/Desktop/Computational
   Lab/Lab 2/Lab 2 Figures/phaseportrait1.47.pdf')
    plt.show()
```