Laboratory 2: The Pendulum

ismisebrendan

March 14, 2023

1 Introduction

This lab uses the trapezoid and Runge-Kutta methods to solve the ordinary differential equations of motion for a linear and nonlinear pendulum in python.

The two-dimensional position of a pendulum bob such as that shown in Figure 1.1 can be described by its angular position θ, the angle subtended by the pendulum and a vertical line, it is zero when the pendulum is vertical and at rest. The bob moves along the arc of a circle with radius L, travelling a distance s along the the tangent to the arc. s is given by Eq. 1.1.

$$s = L\theta \tag{1.1}$$

As L is constant the acceleration of the bob along the arc, $\frac{d^2s}{dt^2}$ is related to the angular acceleration by:

$$\frac{d^2s}{dt^2} = L\frac{d^2\theta}{dt^2} \tag{1.2}$$

The equation of motion for the pendulum can be found now by equating the mass of the pendulum times its acceleration to the force (its weight) acting on it in its direction of motion according to Newton's second law.

$$mL\frac{d^2\theta}{dt^2} = -mgsin\theta \tag{1.3}$$

Giving the equation of motion

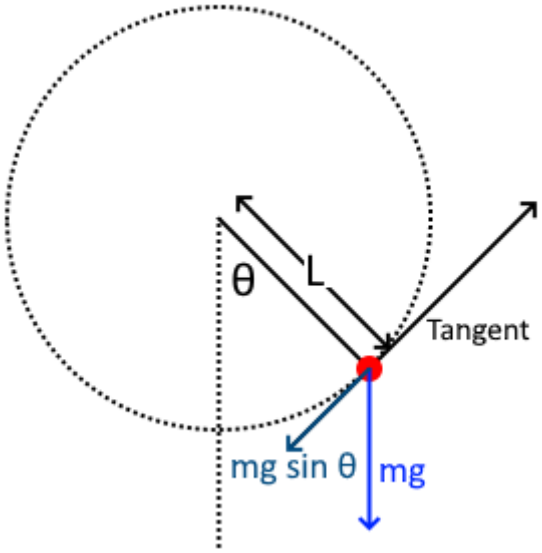

Figure 1.1: The simple pendulum, with length L at an angle θ from the vertical. The weight of the pendulum as well as its component tangential to the motion of the pendulum are shown as is the tangent vector to the arc at the bob.

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}sin\theta \tag{1.4}$$

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\theta \tag{1.5}$$

Eq. 1.4 can be approximated to Eq. 1.5 for small values of θ. Eq. 1.5 can be solved analytically to give

$$\theta = Bsin(\omega_0 t + \delta) \tag{1.6}$$

Where B and ϕ are the amplitude of the motion and the initial phase. ω_0 is the natural frequency of oscillation and is equal to $\sqrt{\frac{g}{L}}$.

In reality there is always some damping force acting against a swinging pendulum opposite to the direction of motion. Here it is taken to be proportional to the pendulum's velocity, ω, and is expressed as $-k\omega$.

There can also be another force driving the oscillation of the pendulum. In this case it is chosen to be sinusoidal in time and have an amplitude of A and a frequency ϕ. It is expressed as $Acos(\phi t)$

This gives an expanded equation of motion,

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}sin\theta - k\omega + Acos(\phi t) \tag{1.7}$$

The equation of the non-linear pendulum, Eq. 1.4, must be solved numerically. In order to do this more easily the angular velocity of the bob $\omega = \frac{d\theta}{dt}$ is introduced

to transform the equation from a second order differential equation to a first order differential equation. The equation of motion now becomes

$$\frac{d\omega}{dt} = f(\theta, \omega, t) \tag{1.8}$$

Where the function f is introduced for convenience and is given by Eq. 1.9 or in the case of the small angle approximation, Eq. 1.10

$$f(\theta, \omega, t) = -\frac{g}{L} sin\theta - k\omega + Acos(\phi t) \tag{1.9}$$

$$f(\theta, \omega, t) = -\frac{g}{L}\theta - k\omega + Acos(\phi t) \tag{1.10}$$

In numerically solving ordinary differential equations an initial value for the variables ad then move forward in time. A method of numerically solving ordinary differential equations is the trapezoid method and is related to the trapezoid rule method of approximate integration.

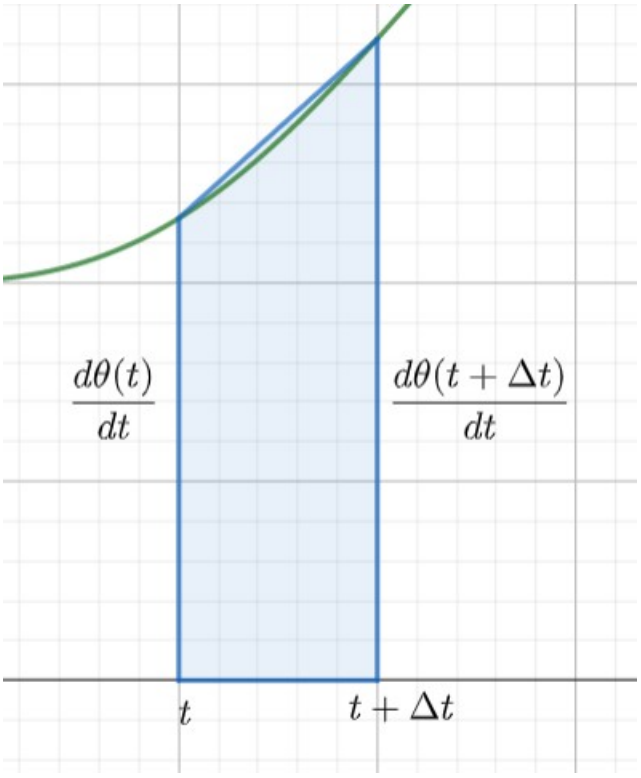

Figure 1.2: The trapezoid rule method of integration, with the graph of $\frac{d\theta}{dt}$ against t.

In Figure 1.2 the area, A, of the trapezoid is

$$A = ((t + \Delta t) - t)\left(\frac{\frac{d\theta(t)}{dt} + \frac{d\theta(t+\Delta t)}{dt}}{2}\right) \tag{1.11}$$

As Δt tends towards 0,

$$\Delta t \frac{\frac{d\theta(t)}{dt} + \frac{d\theta(t+\Delta t)}{dt}}{2} \cong \int_t^{t+\Delta t} \frac{d\theta}{dt} dt = \theta_{n+1} - \theta_n \tag{1.12}$$

Rearranging this to give the position, θ, at the next time step, θ_{n+1} given the current position, θ_n.

$$\theta_{n+1} \cong \theta_n + \frac{\Delta t}{2} \left(\frac{d\theta(t)}{dt} + \frac{d\theta(t+\Delta t)}{dt} \right) \tag{1.13}$$

This can be further simplified by recognising that $\omega = \frac{d\theta}{dt}$ as above, and by taking the Taylor expansion of $\frac{d\theta(t+\Delta t)}{dt}$

$$\frac{d\theta(t+\Delta t)}{dt} = \frac{d\theta(t)}{dt} + \frac{d^2\theta(t)}{dt^2}\Delta t + O(\Delta t)^2 \cong \omega(t) + f(\theta_n, \omega_n, t)\Delta t \tag{1.14}$$

To give

$$\theta_{n+1} \cong \theta_n + \frac{\Delta t}{2} \left(\omega_n + \left(\omega_n + f(\theta_n, \omega_n, t) \right) \Delta t \right) \tag{1.15}$$

Similarly for the angular velocity, ω_{n+1} can be calculated given ω_n

$$\omega_{n+1} \cong \omega_n + \frac{\Delta t}{2} (f(\theta_n, \omega_n, t) + f(\theta_{n+1}, \omega_n + f(\theta_n, \omega_n, t), t_{n+1}) \tag{1.16}$$

This can be implemented in python and iterated over for many time steps relatively easily.

Another method of numerically solving ordinary differential equations is through the Runge-Kutta method. The second-order algorithm is similar to the trapezoid method but the Taylor expansion occurs around the midpoint rather than the beginning of the time step. The fourth-order Runge-Kutta algorithm is again very similar except that the time step is divided in quarters. This again can be implemented in python and iterated over for many time steps relatively easily.

Pendulums driven by a periodic force with frequency ϕ can have periodic or aperiodic motion. If the motion remains periodic its period is an integer multiple of the period of the driving force, $\frac{2\pi}{\phi}$. If the motion becomes aperiodic it is referred to as chaotic.

The motion of a driven pendulum can conveniently be represented by a graph of the angle, θ, against the angular velocity, ω. This type of graph is referred to as a phase portrait.

The phase portrait of damped driven nonlinear pendulum becomes a closed loop (the limit cycle) if the pendulum is in periodic motion. For nearly every set of initial conditions the initial motion of the pendulum is off of the limit cycle, however after a period of transient motion it does reach the limit cycle.

For certain initial conditions the motion of the pendulum becomes circulation rather than oscillation, so θ appears to increase without bound, however in practice a pendulum angle of $\theta + 2\pi$ is indistinguishable from a pendulum angle of θ.

2 Methodology

2.1 Solving the Linear Pendulum Equation

The code for this section can be found in *Appendix A: Code, 5.1.1 Solving the Linear Pendulum Equation* below.

A function f was initialised corresponding to Eq. 1.10 above to model the linear pendulum. To remove the damping and driving forces the values of k and A were set to 0.0 initially. The $\frac{g}{L}$ term was set to 1 for convenience also.

The initial values of θ, ω, the time, the time step, and the number of steps were set. Different values of f for different θ, ω and t inputs were printed, to ensure that the function worked as expected and the variables were initialised correctly i.e., it was checked that the function printed $-\theta$.

The trapezoid method was implemented to find the values of θ and ω over 1000 time steps. Plots of θ against *nsteps* and ω against *nsteps* were plotted separately and together, as well as a plot of θ and ω together against time.

Two arrays of the initial values for θ and ω were initialised, their values are shown in Table 2.1. The values were looped over in a for loop as the initial values of θ and ω for the trapezoid method procedure.

Table 2.1: The initial values of θ and ω for this code

Initial Values	
θ_0	ω_0
0.2	0.0
1.0	0.0
3.14	0.0
0.0	1.0

2.2 Solving the Nonlinear Pendulum Equation

The code for this section can be found in *Appendix A: Code, 5.1.2 Solving the Noninear Pendulum Equation* below.

The code from *2.1 Solving the Linear Pendulum Equation* was copied, f was renamed to fL (f linear) and a new function fNL (f nonlinear) was created to correspond to Eq. 1.9, all other initial conditions were kept identical. The functions f and fNL were run simultaneously and the graphs of θ and ω against time for each method were compared.

2.3 Using the Runge-Kutta Algorithm

The code for this section can be found in *Appendix A: Code, 5.1.3 Using the Runge-Kutta Algorithm* below.

The code from *2.1 Solving the Linear Pendulum Equation* was copied again, and f was changed to correspond to Eq. 1.9, all other initial conditions were kept identical.

The Runge-Kutta algorithm was implemented as shown in the lab manual and it and the trapezoid method were used in parallel to plot a graph of θ against time for the two methods on the one graph for intial conditions, $\theta_0 = 3.14$ and $\omega_0 = 0.0$.

2.4 Solving the Damped Nonlinear Pendulum Equation

The code for this section can be found in *Appendix A: Code, 5.1.4 Solving the Damped Nonlinear Pendulum Equation* below.

The code from *2.3 Using the Runge-Kutta Algorithm* was copied and the value of k was changed to 0.5 to enable damping, and the code for the trapezoid method was removed.

Graphs of θ against time and ω against time were plotted for initial conditions, $\theta_0 = 3.0$ and $\omega_0 = 0.0$ over 2000 time steps to better show the nature of the graphs.

2.5 Solving the Damped Driven Nonlinear Pendulum Equation

The code for this section can be found in *Appendix A: Code, 5.1.5 Solving the Damped Driven Nonlinear Pendulum Equation* below.

The code from *2.4 Solving the Damped Nonlinear Pendulum Equation* was copied and the value of A was changed to 0.9 to enable the driving force. An array, A_list was created to allow the program loop over the different values of A required.

A variable called *iteration_number* was initialised to keep track of the number of times the script looped over itself. Another array *transient_list* was initialised with appropriate values for wach value of A such that when the graph of the function was plotted after a number of steps greater than that number the transient motion had disappeared. The array *start_graph_list* was created to graph the phase portrait of the data after a suitable number of steps.

For the values of A shown in Table 2.2 the value of θ increases without bound, however as θ is an angle in practice it cannot be distinguished from an angle of $\theta + 2\pi$ so the value of θ is restricted to lie in the range $[-\pi + b, \pi + b]$ where b was a value determined separately for each value of A to ensure the graph does not get cut in two.

Table 2.2: The initial amplitudes, A, of the driving force

A
0.90
1.07
1.35
1.47
1.5

3 Results

The graphs are available in *5.2 Appendix B: Graphs* below.

3.1 Solving the Linear Pendulum Equation

Table 2.1 shows the initial values of θ and ω used in this exercise, the graphs of θ against t and ω against t for each of these pairs of initial conditions are included below in Figures 5.1 to 5.8.

As can be seen they are all sinusoidal in time, and as expected from the fact that ω, the radial velocity is the time derivative of the position, θ there is a $\frac{\pi}{2}$ radians phase difference between the graphs of θ and ω.

3.2 Solving the Nonlinear Pendulum Equation

Using the same initial conditions as shown in Table 2.1 the graphs of θ against t and ω against t for each of these pairs of initial conditions were plotted and compared for both linear and non-linear pendulums. The graphs are shown in Figures 5.9 to 5.16.

As seen in Figures 5.9 and 5.10 the linear pendulum is a very good approximation for small initial angels and angular velocities (θ_0 and ω_0). However for even moderately large values of θ_0 the linear and non-linear pendulum motions deviate quickly as shown in Figures 5.11 and 5.12, where $\theta_0 = 1.0$.

When θ_0 becomes very large, such as in Figures 5.13 and 5.14 ($\theta_0 = 3.14$) the linear and non-linear pendulum motions are wildly different and the linear approximation is wildly different to the true motion.

It is also obvious from Figures 5.15 and 5.16 that the true motion deviates quickly from the idealised motion with a large initial angular velocity, ω_0 despite a small initial angle, θ_0.

3.3 Using the Runge-Kutta Algorithm

Figure 5.17 shows the graph of θ against time for a non-linear pendulum for initial conditions $\theta_0 = 3.14$ and $\omega_0 = 0.0$ as calculated by both the fourth order Runge-Kutta and Trapezoidal methods. As can be seen by the extremely close overlap, the Trapezoidal method is a very good approximation for the fourth order Runge-Kutta method at this level of precision.

3.4 Solving the Damped Nonlinear Pendulum Equation

The graphs in Figures 5.18 and 5.19 show the behaviour of a damped pendulum. It oscillates sinusoidally with an amplitude which decays exponentially, as is expected from knowledge of the nature of damped harmonic oscillators which this pendulum is an example of.

3.5 Solving the Damped Driven Nonlinear Pendulum Equation

Figures 5.20 to 5.24 show the phase portraits for the pendulum with driving forces of initial amplitude as shown in Table 2.2 above.

As seen in Figures 5.20 and 5.21, when the driving force has an initial amplitude of 0.9 or 1.07 the phase portrait of the motion is a closed loop, indicating both periodic motion and that the pendulum oscillates back and forth and does not circulate.

Figures 5.22 and 5.23, initial amplitudes of the force are 1.35 and 1.47 respectively, show examples of periodic motion which behaves in a circulatory manner. The graph in Figure 5.22 was cut at the point they were in order not to cut through the loop and clearly show the limit cycle as a single piece. In the case of the graph in Figure 5.23, the limit cycle required four separate rotations before starting again, so the decision was made to cut the graph at a position which did not interfere with any loops visible.

Figure 5.24 shows a graph with an example of chaotic, aperiodic motion. This is very unlikely to be a transient motion which persisted as the motion is not graphed until after 100,000 steps and is plotted over 50,000 steps, and there is clearly no periodic motion in this range. As such it is determined that this is an example of chaotic motion.

4 Conclusions

As expected a linear pendulum oscillates with its angle, θ, and angular velocity ω varying sinusoidally with time. The linear pendulum is a good approximation for the actual motion of the nonlinear pendulum for small initial values of θ and ω, however when either of them reaches even 1.0 the two methods deviate noticeably from each other, and for even larger initial values the linear approximation is wildly inaccurate in comparison to the nonlinear equation.

While the fourth order Runge-Kutta method may be more accurate than the Trapezoidal method in general, it appears that at least for the desired precision and the initial conditions used in this case, $\theta_0 = 3.14$ and $\omega_0 = 0.0$, they do not deviate significantly from one another.

A damped nonlinear pendulum oscillates sinusoidally with an exponentially decreasing amplitude of motion as shown by the exponential decrease in θ as time progressed and corresponding decrease in ω.

The motion of a damped driven nonlinear pendulum was shown to be periodic in some instances and aperiodic or chaotic in others. In the cases in which it was periodic in some instances the pendulum oscillated back and forth consistently, while in other cases its motion was circulatory. In the case in which its motion was chaotic the pendulum appeared to both oscillate back and forth as well as move in a circulatory manner. Over 50,000 iterations it did not appear to develop a consistent pattern of motion, as is expected from chaotic motion.

Both the fourth order Runge-Kutta and Trapezoidal methods can be used to solve ordinary differential equations in python through relatively simple implementation, how-

ever the fourth order Runge-Kutta method is likely more accurate than the Trapezoidal method as it more accurately calculates integrals than the Trapezoidal method.

5 Appendices

5.1 Appendix A: Code

5.1.1 Solving the Linear Pendulum Equation

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 13:59:32 2023

@author: wattersb
"""

'''
Model the motion of a linear pendulum
'''

import numpy as np
import matplotlib.pyplot as plt

# INITIAL VALUES FOR K PHI AND A
k = 0.0
phi = 0.66667
A = 0.0

# INITIAL VALUES OF VARIABLES
theta = 0.2
omega = 0.0
t = 0.0
dt = 0.01
nsteps = 0

# DEFINE EQUATION 8c FOR LINEAR PENDULUM
def f(theta, omega, t):
    return -theta -k*omega + A*np.cos(phi*t)

# CHECK FUNCTION BY PRINTING f FOR DIFFERENT THETA, OMEGA, T VALUES -> EFFECTIVLY PRINT -THETA
print(f(theta, omega, t))
print(f(0.1, 0.56, 2))
print(f(theta, 20, t))

##############################################
#                                            #
#    DIFFERENT VALUES OF THETA AND OMEGA     #
#                                            #
##############################################

```

```
# LIST OF INITIAL CONDITIONS
theta_init = np.array([0.2, 1.0, 3.14, 0.0])
omega_init = np.array([0.0, 0.0, 0.0, 1.0])

for i in range(len(theta_init)):

    # SET VARIABLES
    theta0 = theta_init[i]
    omega0 = omega_init[i]
    theta = theta0
    omega = omega0
    t = 0.0
    nsteps = 0

    # INITIALISE LISTS
    omega_list = np.array([])
    theta_list = np.array([])
    step_list = np.array([])
    time_list = np.array([])

    # TRAPEZOIDAL RULE
    for nsteps in range(1000):
        fndt = f(theta, omega, t) * dt
        theta = theta + omega * dt/2 + (omega+ fndt)*dt/2
        omega = omega + fndt/2 + f(theta, omega + fndt, t + dt) * dt/2

        # INCREASE TIME
        t = t + dt

        # APPEND TO ARRAYS
        omega_list = np.append(omega_list, omega)
        theta_list = np.append(theta_list, theta)
        step_list = np.append(step_list, nsteps)
        time_list = np.append(time_list, t)

    # SEPARATE GRAPHS
    plt.plot(time_list, theta_list)
    plt.xlabel("time")
    plt.ylabel("$\\theta$")
    plt.title("Graph of $\\theta$ v time for $\\theta_0$ =" + str(
   theta0) + ", $\omega_0$ = " + str(omega0) + " \n for a linear
   pendulum")
    plt.ylim([-np.pi, np.pi])
    plt.savefig("images/ThetaVtime"+str(theta0)+"."+str(i)+".png", dpi
   =300)
    plt.show()

    plt.plot(time_list, omega_list)
    plt.xlabel("time")
    plt.ylabel("$\omega$")
    plt.title("Graph of $\omega$ v time for $\\theta_0$ =" + str(theta0
   ) + ", $\omega_0$ = " + str(omega0) + " \n for a linear pendulum")
```

```
    plt.ylim([-np.pi, np.pi])
    plt.savefig("images/OmegaVtime"+str(omega0)+"."+str(i)+".png", dpi
    =300)
    plt.show()

    # GRAPHS TOGETHER
    plt.plot(step_list, theta_list, label='$\\theta$')
    plt.plot(step_list, omega_list, label='$\omega$')
    plt.legend()
    plt.xlabel("nsteps")
    plt.ylabel("$\\theta$, $\omega$")
    plt.title("Graph of $\\theta$ and $\omega$ v nsteps for $\\theta_0$
    =" + str(theta0) + ", $\omega_0$ = " + str(omega0) + " \n for a
    linear pendulum")
    plt.axis([0, 500, -np.pi, np.pi])
    plt.savefig("images/Exercise1TogetherSteps"+"."+str(i)+".png", dpi
    =300)
    plt.show()

    # GRAPH V TIME
    plt.plot(time_list, theta_list, label='$\\theta$')
    plt.plot(time_list, omega_list, label='$\omega$')
    plt.legend()
    plt.xlabel("time")
    plt.ylabel("$\\theta$, $\omega$")
    plt.title("Graph of $\\theta$ and $\omega$ v time for $\\theta_0$ =
    " + str(theta0) + ", $\omega_0$ = " + str(omega0) + " \n for a
    linear pendulum")
    plt.ylim([-np.pi, np.pi])
    plt.savefig("images/Exercise1TogetherTime"+"."+str(i)+".png", dpi
    =300)
    plt.show()
```

5.1.2 Solving the Nonlinear Pendulum Equation

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 15:01:42 2023

@author: wattersb
"""

'''
Compare the motion of a linear and nonlinear pendulum
'''

import numpy as np
import matplotlib.pyplot as plt

# INITIAL VALUES FOR K PHI AND A
k = 0.0
phi = 0.66667
A = 0.0
```

```
19
20 # INITIAL VALUE OF dt
21 dt = 0.01
22
23 # DEFINE EQUATION 8c FOR NON-LINEAR PENDULUM
24 def fNL(theta, omega, t):
25     return -np.sin(theta) -k*omega + A*np.cos(phi*t)
26
27 # DEFINE EQUATION 8c FOR LINEAR PENDULUM
28 def fL(theta, omega, t):
29     return -theta -k*omega + A*np.cos(phi*t)
30
31 ##############################################
32 #                                            #
33 #    DIFFERENT VALUES OF THETA AND OMEGA     #
34 #                                            #
35 ##############################################
36
37 # LIST OF INITIAL CONDITIONS
38 theta_init = np.array([0.2, 1.0, 3.14, 0.0])
39 omega_init = np.array([0.0, 0.0, 0.0, 1.0])
40
41 for i in range(len(theta_init)):
42
43     # SET VARIABLES
44     theta0 = theta_init[i]
45     omega0 = omega_init[i]
46     t = 0.0
47     nsteps = 0
48
49     # SET NONLINEAR AND LINEAR VERSIONS OF VARIABLES
50     NL_theta = theta0
51     NL_omega = omega0
52
53     L_theta = theta0
54     L_omega = omega0
55
56     # INITIALISE LISTS FOR LINEAR AND NONLINEAR
57     NL_omega_list = np.array([])
58     NL_theta_list = np.array([])
59
60     L_omega_list = np.array([])
61     L_theta_list = np.array([])
62
63     time_list = np.array([])
64
65     for nsteps in range(1000):
66         # TRAPEZOIDAL RULE
67
68         # NONLINEAR
69         NL_fndt = fNL(NL_theta, NL_omega, t) * dt
70         NL_theta = NL_theta + NL_omega * dt/2 + (NL_omega+ NL_fndt)*dt
```

```
    /2
        NL_omega = NL_omega + NL_fndt/2 + fNL(NL_theta, NL_omega +
    NL_fndt, t + dt) * dt/2

        # LINEAR
        L_fndt = fL(L_theta, L_omega, t) * dt
        L_theta = L_theta + L_omega * dt/2 + (L_omega+ L_fndt)*dt/2
        L_omega = L_omega + L_fndt/2 + fL(L_theta, L_omega + L_fndt, t
    + dt) * dt/2

        # INCREASE TIME
        t = t + dt

        # APPEND TO ARRAYS
        NL_omega_list = np.append(NL_omega_list, NL_omega)
        NL_theta_list = np.append(NL_theta_list, NL_theta)

        L_omega_list = np.append(L_omega_list, L_omega)
        L_theta_list = np.append(L_theta_list, L_theta)

        time_list = np.append(time_list, t)

    # GRAPHs V TIME
    plt.plot(time_list, NL_theta_list, label='$\\theta$ Non linear')
    plt.plot(time_list, L_theta_list, label='$\\theta$ Linear')
    plt.legend()
    plt.xlabel("time")
    plt.ylabel("$\\theta$")
    plt.title("Graph of $\\theta$ v time for $\\theta_0$ =" + str(
    theta0) + ", $\omega_0$ = " + str(omega0) + " \n for non-linear and
     linear pendula")
    plt.ylim([-np.pi, np.pi])
    plt.savefig("images/ComparisonTheta"+"."+str(i)+".png", dpi=300)
    plt.show()

    plt.plot(time_list, NL_omega_list, label='$\omega$ Non linear')
    plt.plot(time_list, L_omega_list, label='$\omega$ Linear')
    plt.legend()
    plt.xlabel("time")
    plt.ylabel("$\\theta$, $\omega$")
    plt.title("Graph of $\omega$ v time for $\\theta_0$ =" + str(theta0
    ) + ", $\omega_0$ = " + str(omega0) + " \n for non-linear and
    linear pendula")
    plt.ylim([-np.pi, np.pi])
    plt.savefig("images/ComparisonOmega"+"."+str(i)+".png", dpi=300)
    plt.show()
```

5.1.3 Using the Runge-Kutta Algorithm

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 15:19:59 2023

```

```
@author: wattersb
"""

'''
Model the motion of a non-linear pendulum
'''

import numpy as np
import matplotlib.pyplot as plt

# INITIAL VALUES FOR K PHI AND A
k = 0.0
phi = 0.66667
A = 0.0

# INITIAL VALUE OF dt
dt = 0.01

# DEFINE EQUATION 8c FOR NON-LINEAR PENDULUM
def f(theta, omega, t):
    return -np.sin(theta) -k*omega + A*np.cos(phi*t)

# SET VARIABLES
theta = 3.14
omega = 0.0
t = 0.0

RK_theta = theta
RK_omega = omega

TR_theta = theta
TR_omega = omega

# INITIALISE LISTS
RK_theta_list = np.array([])
TR_theta_list = np.array([])
time_list = np.array([])

for i in range(1000):
    # RUNGE-KUTTA
    k1a = dt * RK_omega
    k1b = dt * f(RK_theta, RK_omega, t)
    k2a = dt * (RK_omega + k1b/2)
    k2b = dt * f(RK_theta + k1a/2, RK_omega + k1b/2, t + dt/2)
    k3a = dt * (RK_omega + k2b/2)
    k3b = dt * f(RK_theta + k2a/2, RK_omega + k2b/2, t + dt/2)
    k4a = dt * (RK_omega + k3b)
    k4b = dt * f(RK_theta + k3a, RK_omega + k3b, t + dt)
    RK_theta = RK_theta + (k1a + 2*k2a + 2*k3a + k4a) / 6
    RK_omega = RK_omega + (k1b + 2*k2b + 2*k3b + k4b) / 6

    # TRAPEZOIDAL RULE
```

```
    fndt = f(TR_theta, TR_omega, t) * dt
    TR_theta = TR_theta + TR_omega * dt/2 + (TR_omega+ fndt)*dt/2
    TR_omega = TR_omega + fndt/2 + f(TR_theta, TR_omega + fndt, t + dt)
    * dt/2

    # INCREASE TIME
    t = t + dt

    # APPEND TO ARRAYS
    RK_theta_list = np.append(RK_theta_list, RK_theta)
    TR_theta_list = np.append(TR_theta_list, TR_theta)
    time_list = np.append(time_list, t)

# GRAPH V TIME
plt.plot(time_list, RK_theta_list, label='$\\theta$ for Runge-Kutta')
plt.plot(time_list, TR_theta_list, label='$\\theta$ for Trapezoidal
   Rule')
plt.legend()
plt.xlabel("time")
plt.ylabel("$\\theta$")
plt.title("Graph of $\\theta$ and $\omega$ v time for $\\theta_0$ =
   3.14, $\omega_0$ = 0.0 for a non-linear \n pendulum calculated
   using the RK Method and Trapezoidal Rule")
plt.ylim([-np.pi, np.pi])
plt.savefig("images/Exercise3.png", dpi=300)
plt.show()
```

5.1.4 Solving the Damped Nonlinear Pendulum Equation

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 15:35:00 2023

@author: wattersb
"""

'''
Model the motion of a damped non-linear pendulum
'''

import numpy as np
import matplotlib.pyplot as plt

# INITIAL VALUES FOR K PHI AND A
k = 0.5
phi = 0.66667
A = 0.0

# INITIAL VALUE OF dt
dt = 0.01

# DEFINE EQUATION 8c FOR NON-LINEAR PENDULUM
def f(theta, omega, t):
```

```
    return -np.sin(theta) -k*omega + A*np.cos(phi*t)

# SET VARIABLES
theta = 3.0
omega = 0.0
t = 0.0

# INITIALISE LISTS
omega_list = np.array([])
theta_list = np.array([])
time_list = np.array([])

for i in range(2000):
    # RUNGE-KUTTA METHOD
    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

    # INCREASE TIME
    t = t + dt

    # APPEND TO ARRAYS
    omega_list = np.append(omega_list, omega)
    theta_list = np.append(theta_list, theta)
    time_list = np.append(time_list, t)

# GRAPHS V TIME
plt.plot(time_list, theta_list)
plt.xlabel("time")
plt.ylabel("$\\theta$")
plt.title("Graph of $\\theta$ v time for $\\theta_0$ = 3.0, $\omega_0$
    = 0.0 \n for a damped non-linear pendulum")
plt.ylim([-np.pi, np.pi])
plt.savefig("images/Exercise4Theta.png", dpi=300)
plt.show()

plt.plot(time_list, omega_list)
plt.xlabel("time")
plt.ylabel("$\omega$")
plt.title("Graph of $\omega$ v time for $\\theta_0$ = 3.0, $\omega_0$ =
    0.0 \n for a damped non-linear pendulum")
plt.ylim([-np.pi, np.pi])
plt.savefig("images/Exercise4Omega.png", dpi=300)
plt.show()
```

5.1.5 Solving the Damped Driven Nonlinear Pendulum Equation

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 15:48:34 2023

@author: wattersb
"""

'''
Model the motion of a damped driven non-linear pendulum
'''

import numpy as np
import matplotlib.pyplot as plt

# INITIAL VALUES FOR K PHI AND A
k = 0.5
phi = 0.66667
A = 0.9

# INITIAL VALUE OF dt
dt = 0.01

# INITIAL VALUE OF ITERATION NUMBER
iteration_number = 0

# DEFINE EQUATION 8c FOR NON-LINEAR PENDULUM
def f(theta, omega, t):
    return -np.sin(theta) -k*omega + A*np.cos(phi*t)

################################
#                              #
#    DIFFERENT VALUES OF A     #
#                              #
################################

# LIST OF INITIAL CONDITIONS
A_list = np.array([0.90, 1.07, 1.35, 1.47, 1.5])
b_list = np.array([0, 0.1, 0, -0.5, 0])
transient_list = np.array([5000, 5000, 5000, 10000, 100000])
start_graph_list = np.array([10000, 10000, 25000, 30000, 150000])

for x in range(len(A_list)):

    # SET VARIABLES
    theta = 3.0
    omega = 0.0
    t = 0.0
    A = A_list[x]
    b = b_list[x]

```

```
    # INITIALISE LISTS
    omega_list = np.array([])
    theta_list = np.array([])
    time_list = np.array([])

    # RESET ITERATION NUMBER
    iteration_number = 0

    # SET TRANSIENT
    transient = transient_list[x]

    for iteration_number in range(start_graph_list[x]):
        # RUNGE-KUTTA METHOD
        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

        # INCREASE TIME
        t = t + dt

        # CHECK -PI < THETA < PI
        if (theta > np.pi + b) or (theta < -np.pi + b):
            theta -= 2 * np.pi * np.abs(theta) / theta

        # APPEND TO ARRAYS IF ITERATION_NUMBER >= TRANSIENT
        if iteration_number >= transient:
            omega_list = np.append(omega_list, omega)
            theta_list = np.append(theta_list, theta)
            time_list = np.append(time_list, t)

    # GRAPH
    plt.scatter(omega_list, theta_list, s=0.05)
    plt.ylabel("$\\theta$")
    plt.xlabel("$\omega$")
    plt.title("Phase portrait of the damped driven non-linear pendulum
   for A = " + str(A))
    plt.savefig("images/Exercise5"+"."+str(x)+".png", dpi=300)
    plt.show()
```

5.2 Appendix B:Graphs

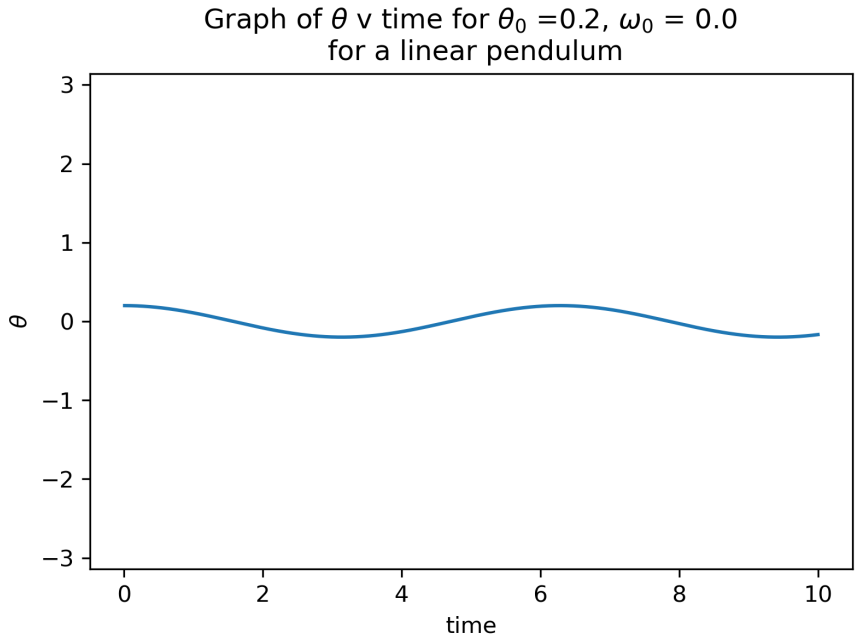

Figure 5.1: The graph of θ against time for initial conditions $\theta_0 = 0.2$ and $\omega_0 = 0.0$

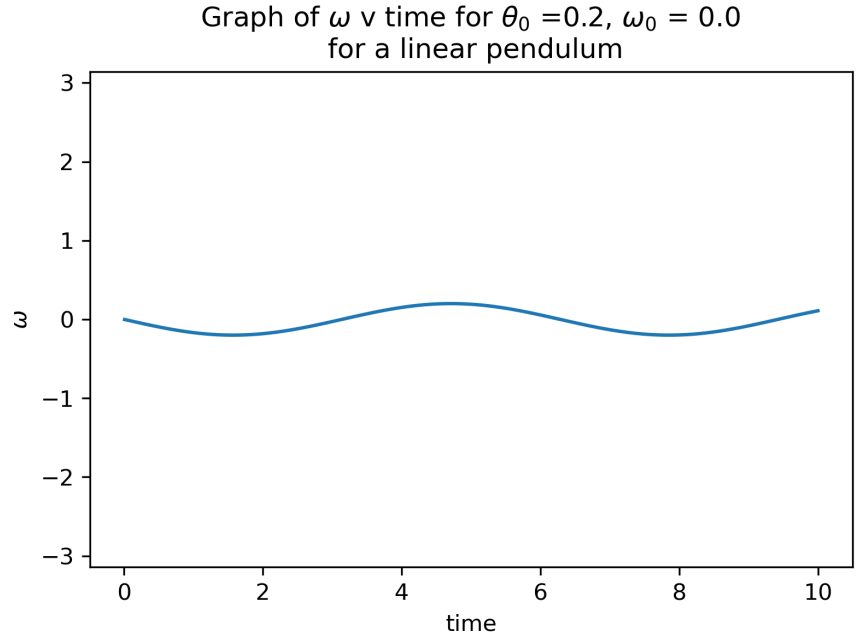

Figure 5.2: The graph of ω against time for initial conditions $\theta_0 = 0.2$ and $\omega_0 = 0.0$

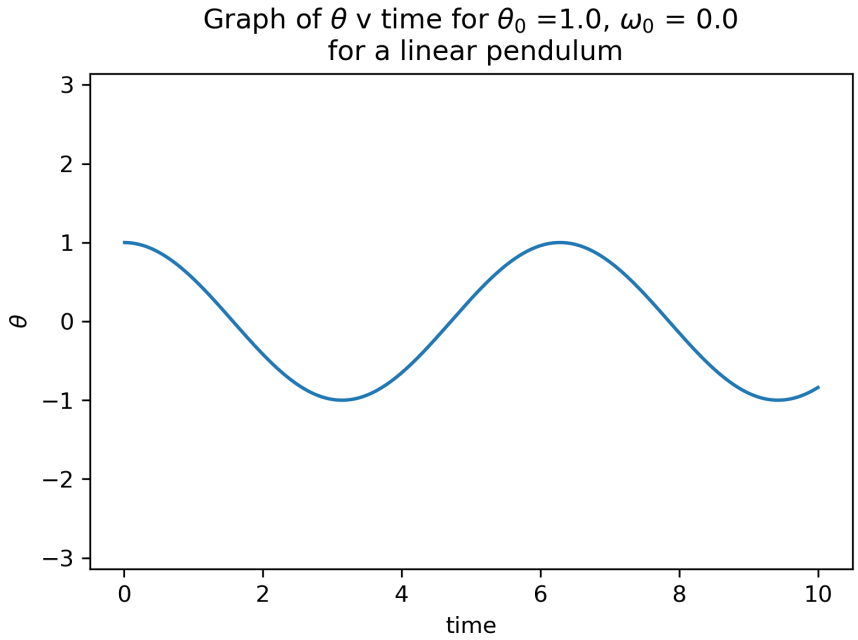

Figure 5.3: The graph of θ against time for initial conditions $\theta_0 = 1.0$ and $\omega_0 = 0.0$

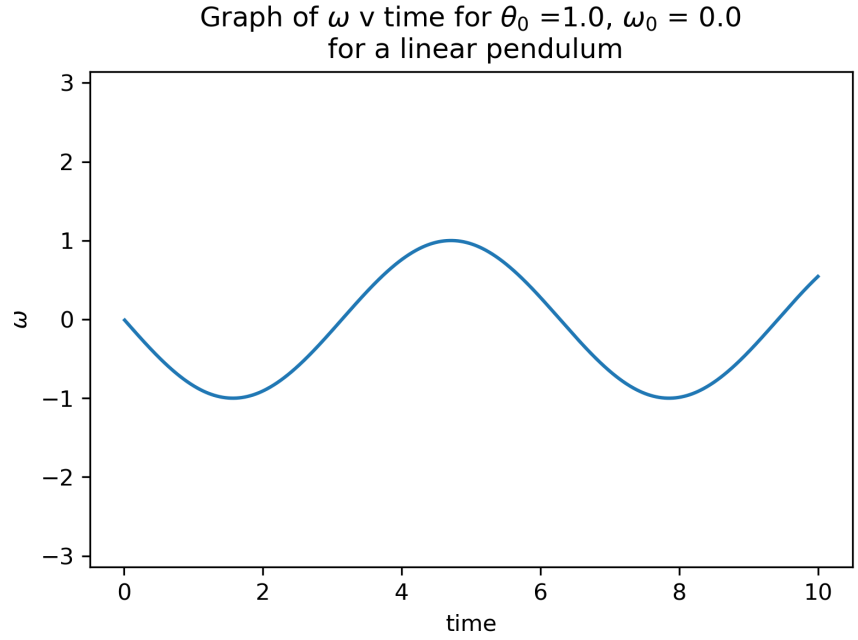

Figure 5.4: The graph of ω against time for initial conditions $\theta_0 = 1.0$ and $\omega_0 = 0.0$

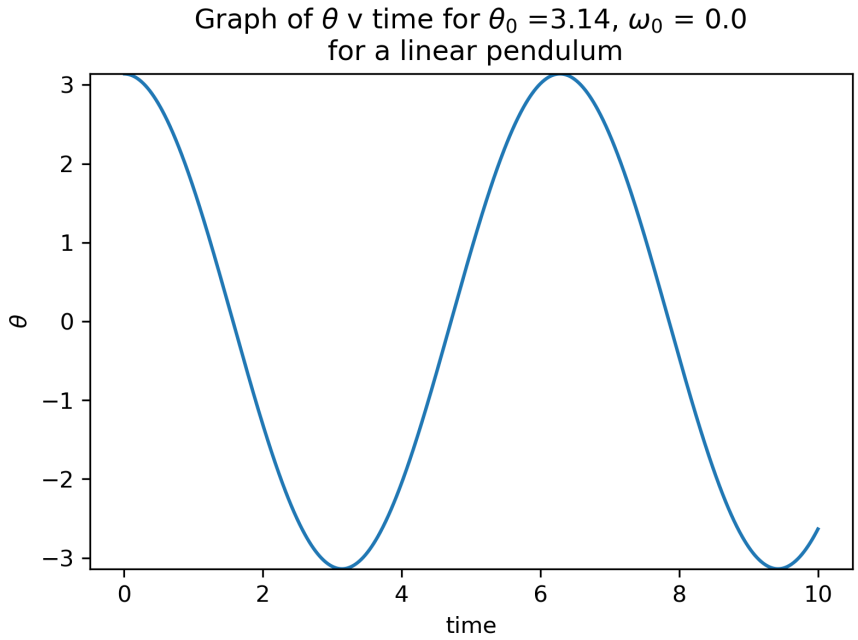

Figure 5.5: The graph of θ against time for initial conditions $\theta_0 = 3.14$ and $\omega_0 = 0.0$

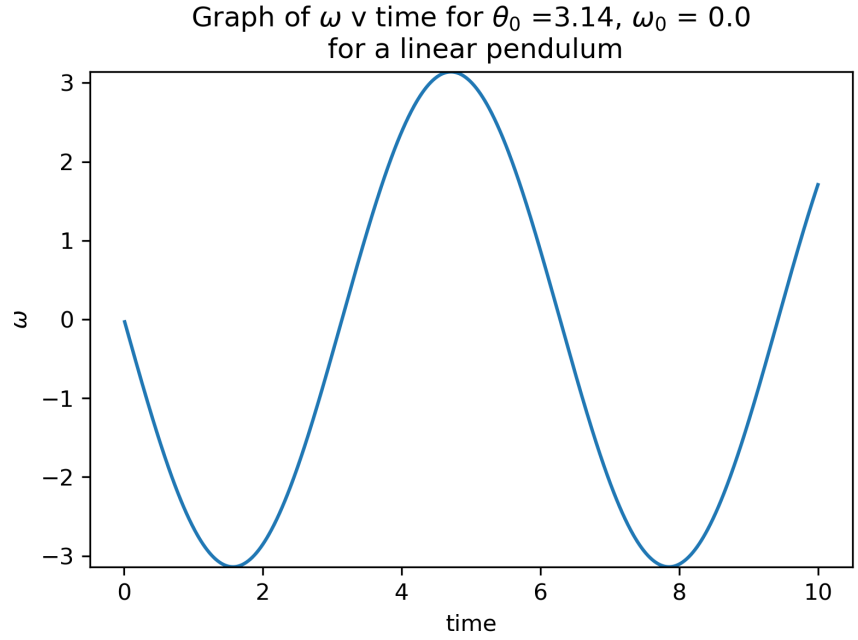

Figure 5.6: The graph of ω against time for initial conditions $\theta_0 = 3.14$ and $\omega_0 = 0.0$

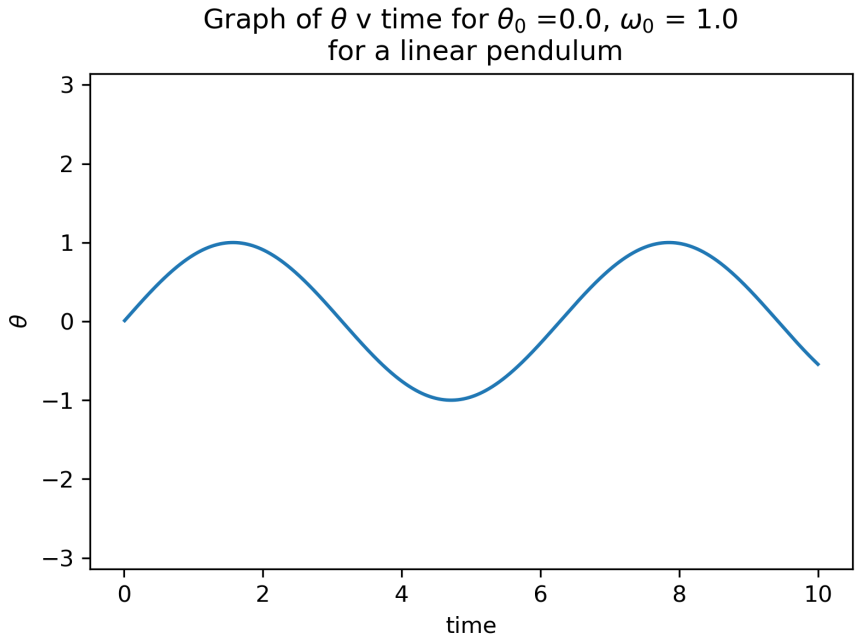

Figure 5.7: The graph of θ against time for initial conditions $\theta_0 = 0.0$ and $\omega_0 = 1.0$

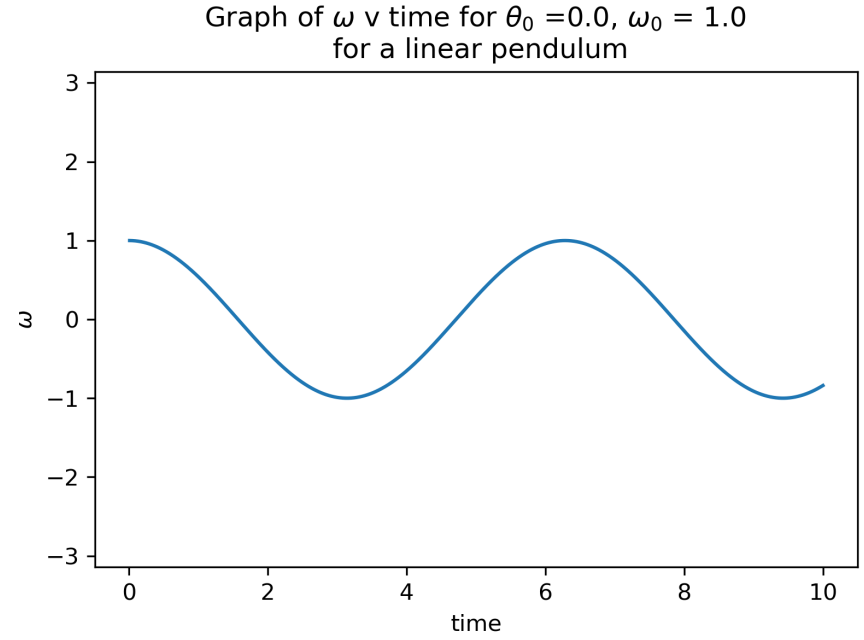

Figure 5.8: The graph of ω against time for initial conditions $\theta_0 = 0.0$ and $\omega_0 = 1.0$

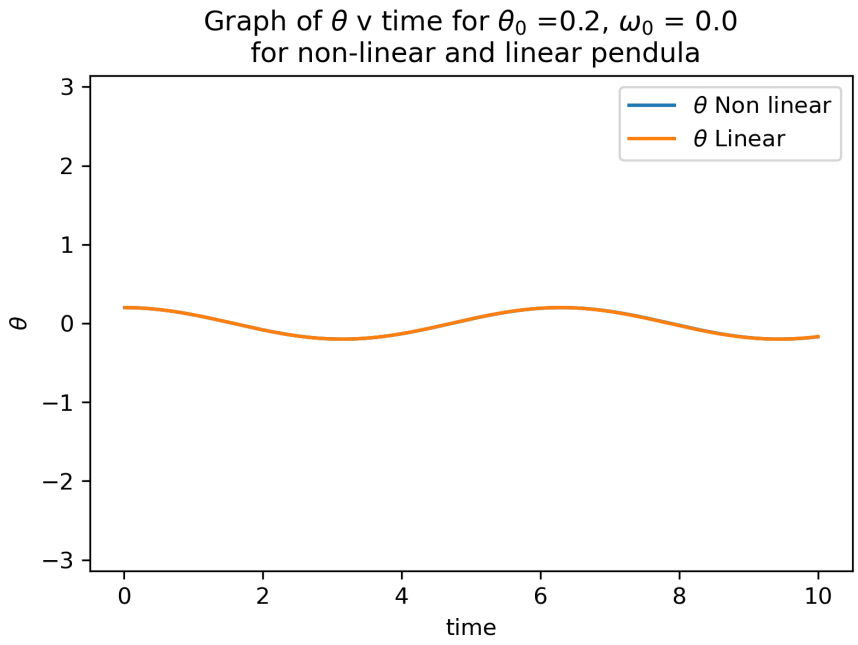

Figure 5.9: The graph of θ against time for linear and non-linear pendulums for initial conditions $\theta_0 = 0.2$ and $\omega_0 = 0.0$

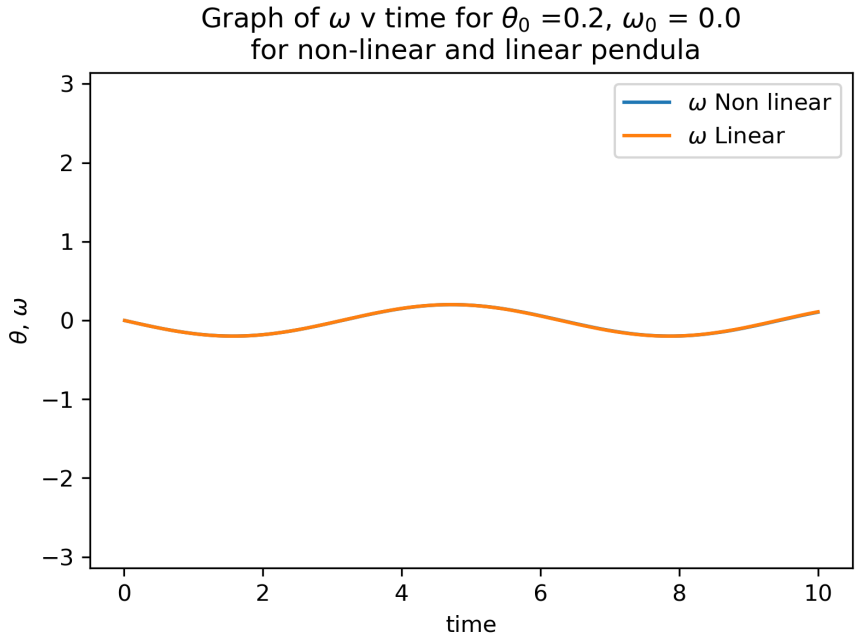

Figure 5.10: The graph of ω against time for linear and non-linear pendulums for initial conditions $\theta_0 = 0.2$ and $\omega_0 = 0.0$

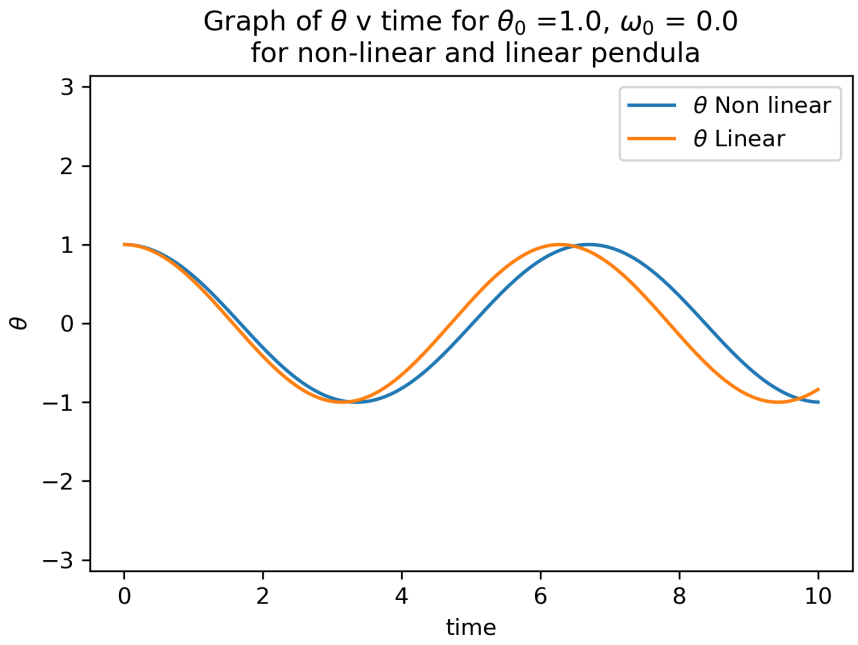

Figure 5.11: The graph of θ against time for linear and non-linear pendulums for initial conditions $\theta_0 = 1.0$ and $\omega_0 = 0.0$

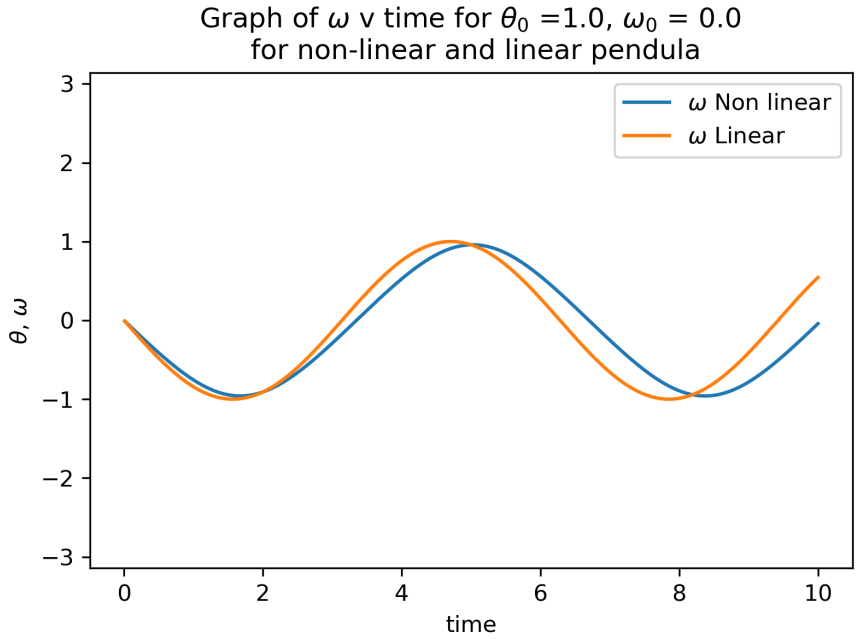

Figure 5.12: The graph of ω against time for linear and non-linear pendulums for initial conditions $\theta_0 = 1.0$ and $\omega_0 = 0.0$

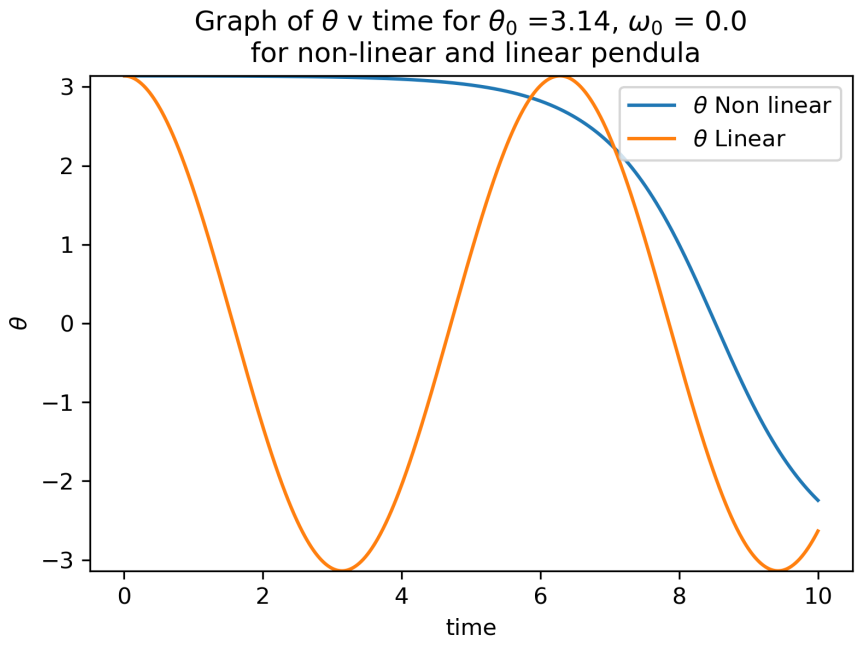

Figure 5.13: The graph of θ against time for linear and non-linear pendulums for initial conditions $\theta_0 = 3.14$ and $\omega_0 = 0.0$

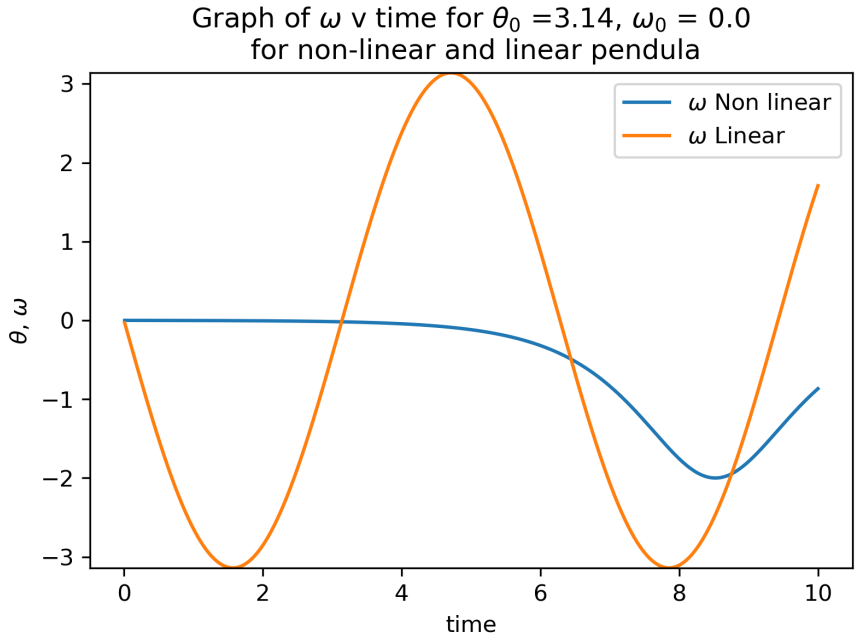

Figure 5.14: The graph of ω against time for linear and non-linear pendulums for initial conditions $\theta_0 = 3.14$ and $\omega_0 = 0.0$

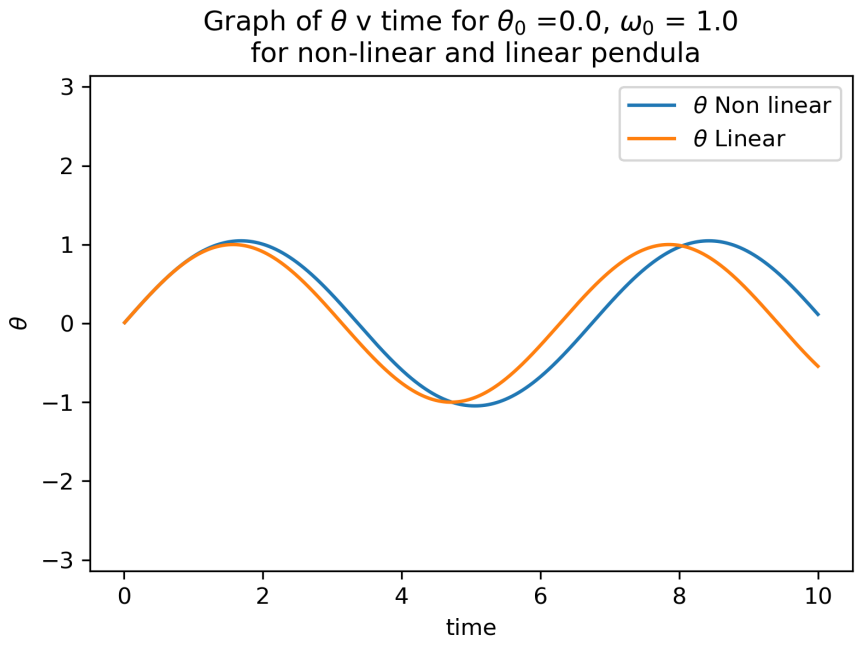

Figure 5.15: The graph of θ against time for linear and non-linear pendulums for initial conditions $\theta_0 = 0.0$ and $\omega_0 = 1.0$

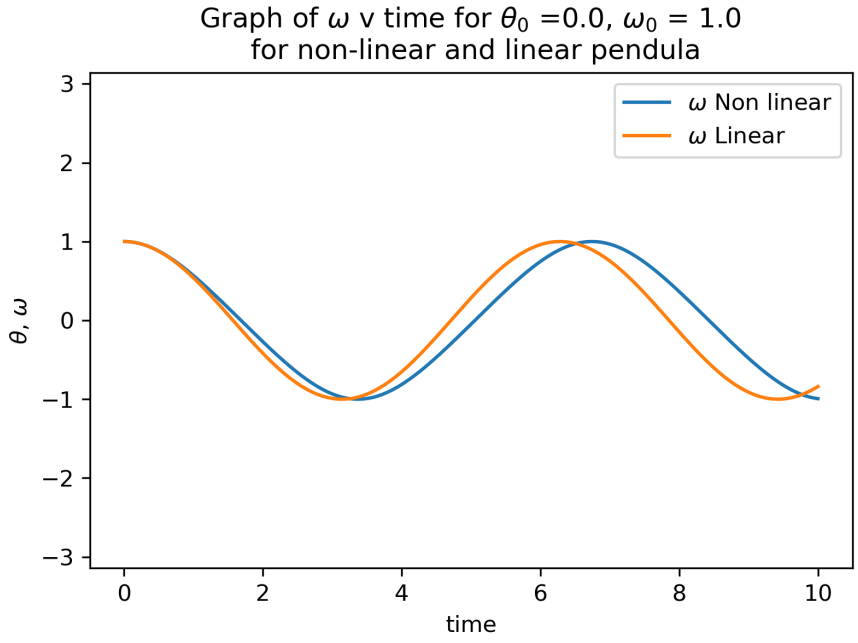

Figure 5.16: The graph of ω against time for linear and non-linear pendulums for initial conditions $\theta_0 = 0.0$ and $\omega_0 = 1.0$

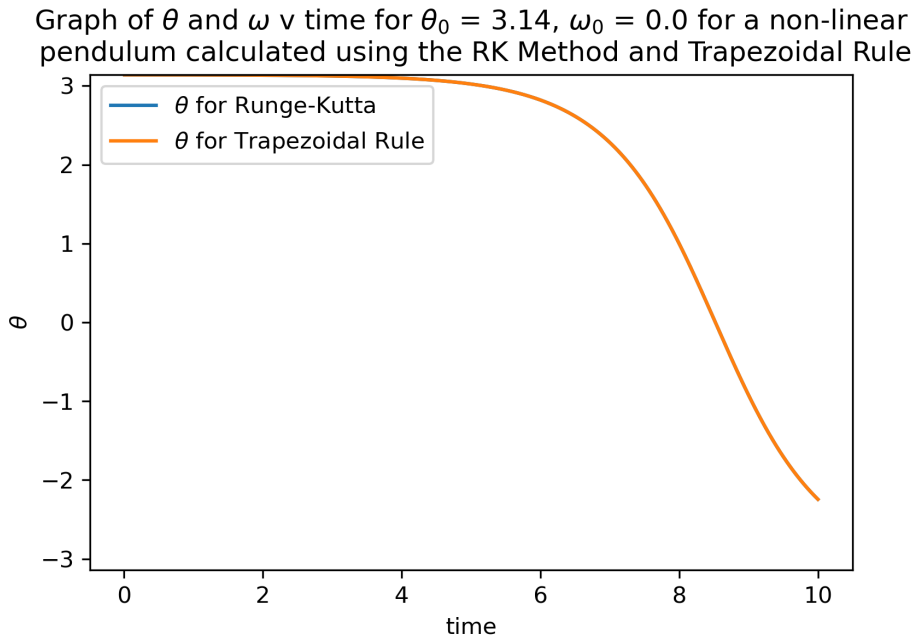

Figure 5.17: The graph of θ against time for a non-linear pendulum for initial conditions $\theta_0 = 3.14$ and $\omega_0 = 0.0$ calculated by both the Runge-Kutta and Trapezoidal methods

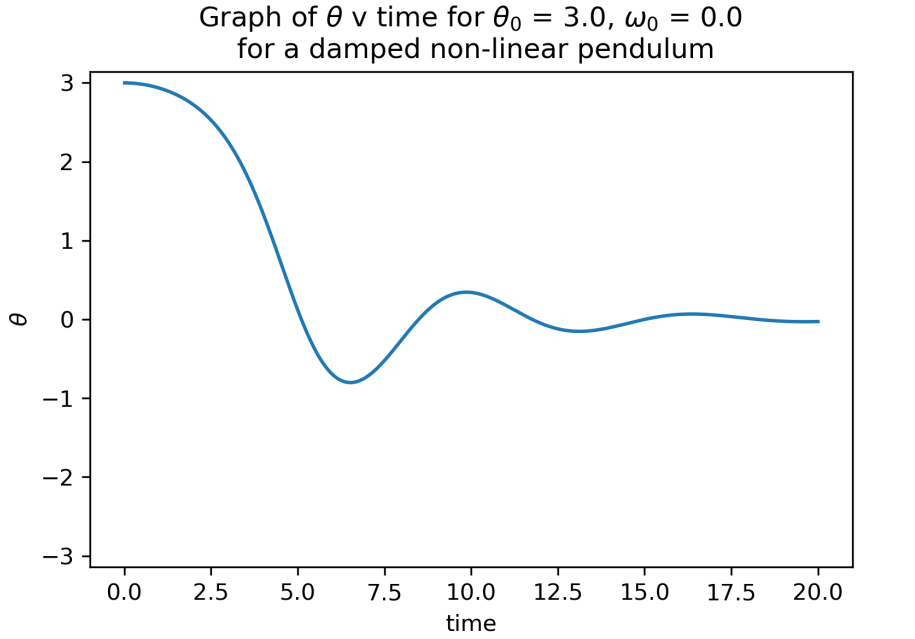

Figure 5.18: The graph of θ against time for a damped non-linear pendulum for initial conditions $\theta_0 = 3.0$ and $\omega_0 = 0.0$

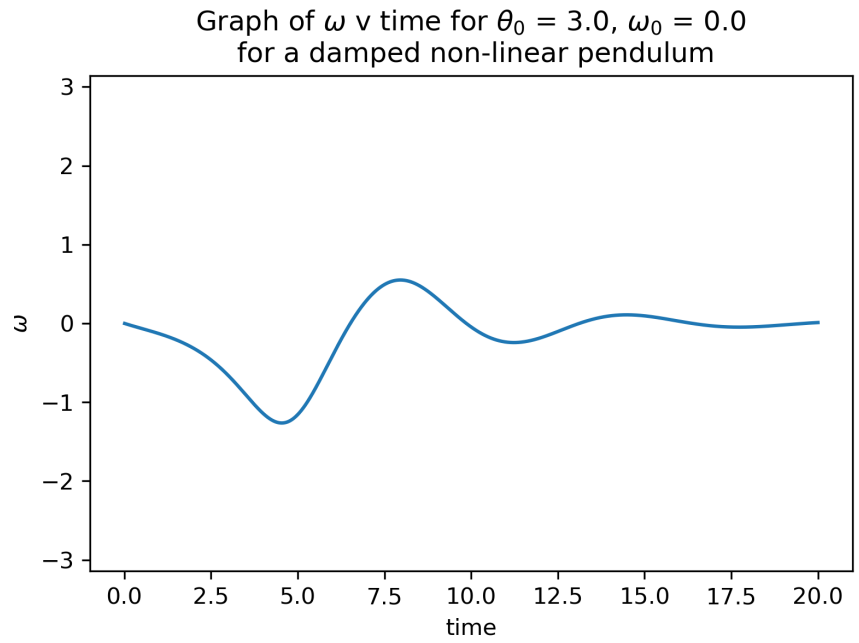

Figure 5.19: The graph of ω against time for a damped non-linear pendulum for initial conditions $\theta_0 = 3.0$ and $\omega_0 = 0.0$

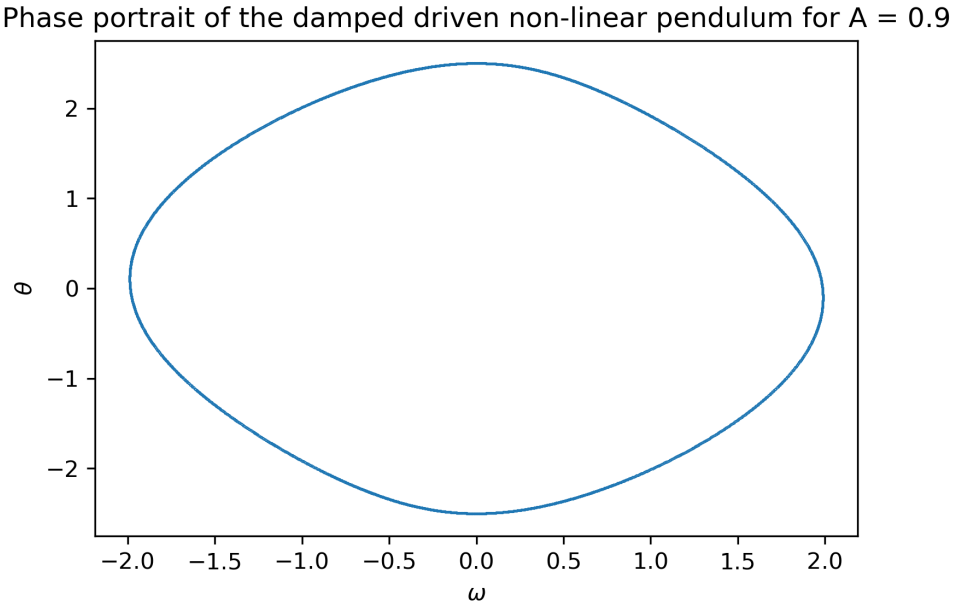

Figure 5.20: The graph of θ against ω for a damped driven non-linear pendulum with a driving force of initial amplitude 0.9

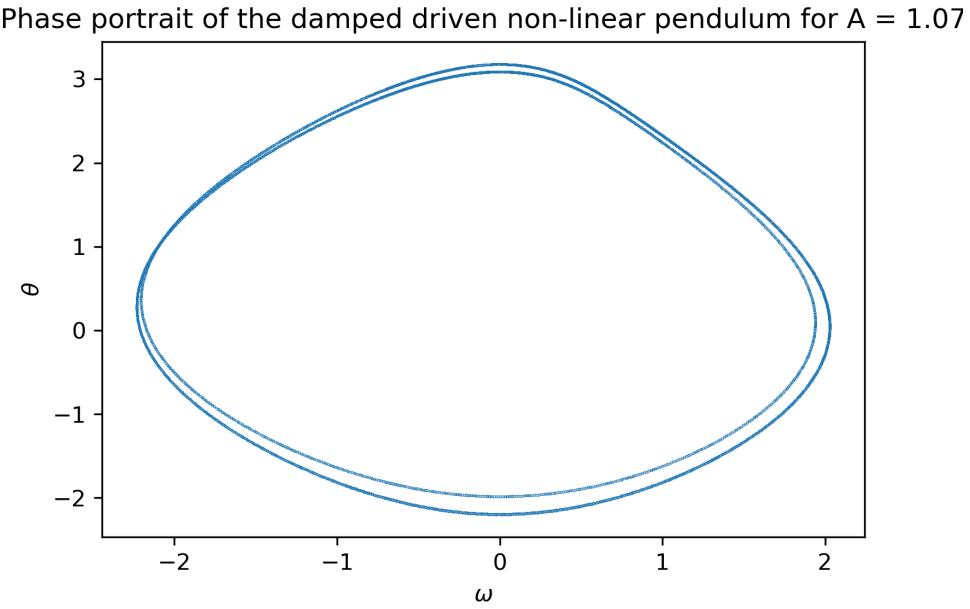

Figure 5.21: The graph of θ against ω for a damped driven non-linear pendulum with a driving force of initial amplitude 1.07

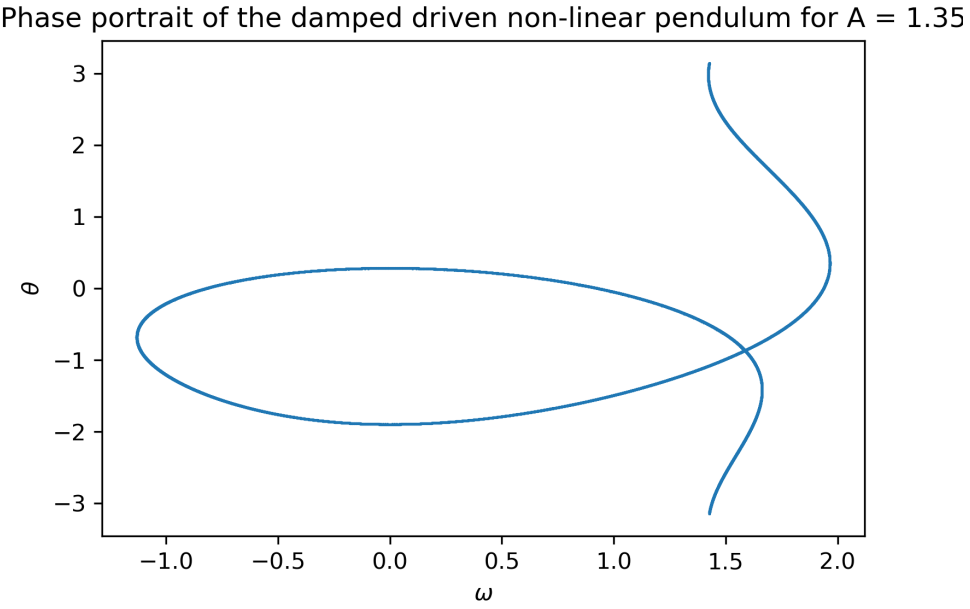

Figure 5.22: The graph of θ against ω for a damped driven non-linear pendulum with a driving force of initial amplitude 1.35

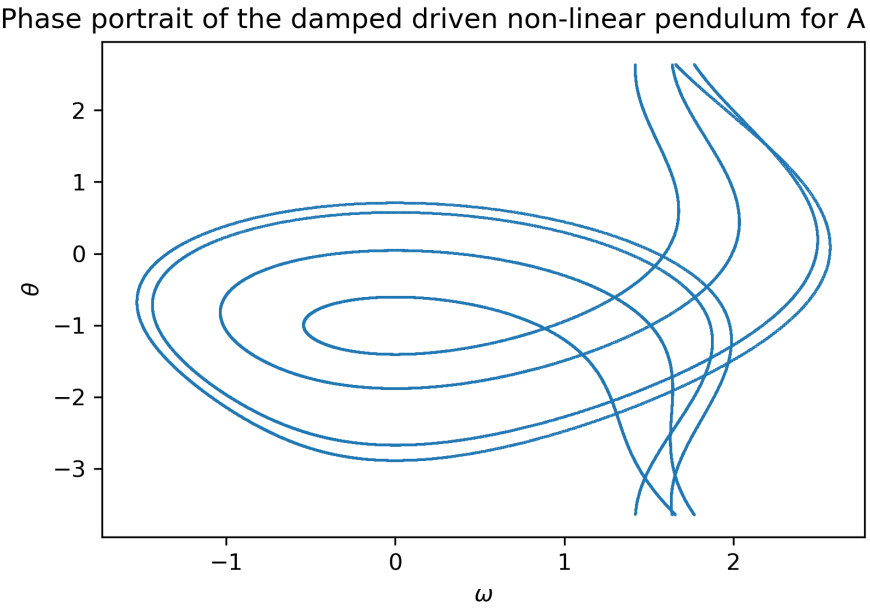

Figure 5.23: The graph of θ against ω for a damped driven non-linear pendulum with a driving force of initial amplitude 1.47

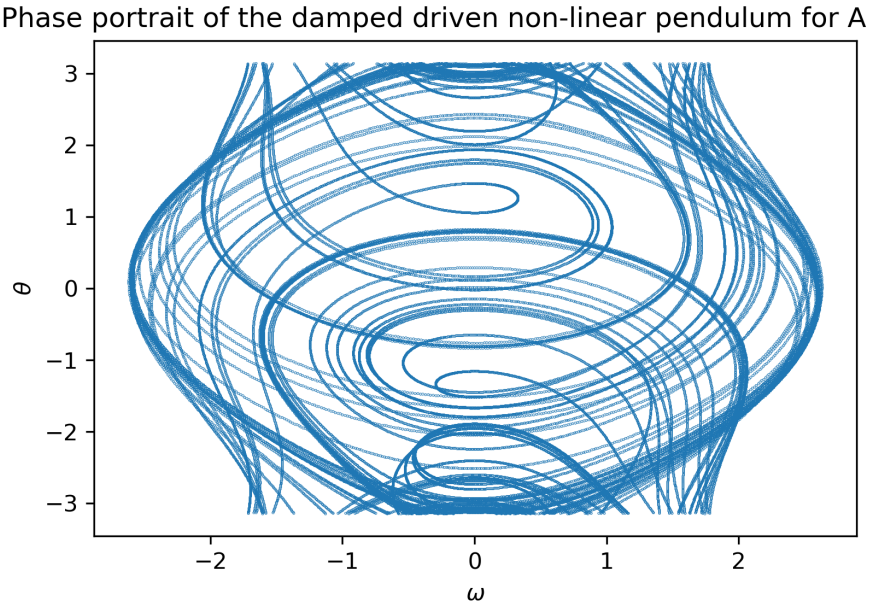

Figure 5.24: The graph of θ against ω for a damped driven non-linear pendulum with a driving force of initial amplitude 1.5