Lab 2: The Pendulum

Valentina Watanabe

February 2024

Abstract

This computational project served as a study of pendulum motion through the simulation of undamped and damped, and linear and nonlinear oscillations. We used the Trapezoid method and the Runge-Kutta method to solve for the corresponding ordinary differential equations. We first created a script for an undamped linear pendulum and created graphs for various values of omega and theta in comparison to time. We then modified the script to simulate linear motion and then compared this with the previous results. To finalize the analysis of undamped motion, we compared the two methods used to solve for ODEs.

For studying damped oscillatory motion, the damping coefficient and the driven force were taken into consideration for the motion equations. Phase portraits to study the change of periodic motion for a set of amplitude values were created.

1 Introduction and Theory

1.1 Undamped Linear and Nonlinear Pendulum

The equation of motion for a nonlinear pendulum is,

$$\frac{d^2s}{dt^2} = -\frac{g}{l}\sin(\theta).$$

For the motion of a pendulum with a small enough angle to establish $\sin(\theta) \simeq \theta$, for a linear undamped oscillator we can use the linearised equation,

$$\frac{d^2s}{dt^2} = -\frac{g}{l}\theta.$$

This has the solution,

$$\theta = B\sin(\omega_0 t + \delta).$$

1.2 Solving ODEs

By solving for ordinary differential equations numerically we can describe the dynamics of the pendulum system over time. To do so, we can first use the

trapezoid rule. Its key idea is to divide the area under the curve into trapezoids and use the Taylor series expansion about the midpoint of the time step, making it more accurate than other methods, like Euler's. We get,

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

and

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

The second method we use is the fourth-order Runge-Kutta Method (or RK4) for the nonlinear undamped pendulum. It provides an enhanced accuracy of the previous method by including numerous slope estimates at various places within each time step. It uses the Taylor expansion about the mid-point of the time step and with the time step divided into quarters.

1.3 Damped Linear and Nonlinear Pendulum

For real-life cases, we have to take friction and dampness into consideration. We then use the motion equation while including the friction coefficient:

$$\frac{d^2 s}{dt^2} = -\frac{g}{l}\sin(\theta) - k\omega + A\cos(\theta t).$$

This is derived from including the friction coefficient, F_f, and a periodically driven force, F_{driven} into the mentioned undamped nonlinear motion equation.

$$F_F = -mk\omega$$

$$F_{driven} = A\cos(\theta t)$$

2 Experimental Method

2.1 Undamped Linear and Nonlinear Pendulum

For the first part of our code, we defined the function

$$F = -\theta - k\omega + A\cos(\phi t),$$

where k = 0, which indicates that is undamped, A = 0.0, and $\phi = 0.66667$. The objective is to solve for the motion of a linear pendulum. The trapezoid rules were also added for numerical integration. By enclosing this in a loop that takes 1000 steps, it solves the pendulum equation by updating the values of theta and omega at each time step. A command is given to plot omega and for theta at each time step, to finally get a final plot that helps visualize the motion of the pendulum over time. The code then continues by plotting theta versus t and omega versus t for a set of different values.

For the nonlinear pendulum, the same code was used but modified accordingly, so

$$F = -\sin(\theta) - k\omega + A\cos(\phi t).$$

The same plots are graphed with the values of the previous exercise, and with the additional plots that compare linear and nonlinear values of omega and theta. For the last part of the nonlinear pendulum analysis, the fourth-order Runge-Kutta method was used. This method uses several slope intervals to calculate the first-order differential equation of motion. For this part, values theta = 3.1415, and omega = 0.0 were used, and then the theta versus time for both mentioned methods were plotted.

2.2 Damped Linear and Nonlinear Pendulum

A similar script was used as in the previous section, with the modification to account for the damping effect. Like previously, plots for theta and omega with time were obtained. The function was set to,

$$F = -\frac{g}{L}\sin(\theta) - k\omega + A\cos(\phi t).$$

The final part of the analysis consisted of constructing phase portraits of the system for certain initial values. Phase portraits are useful to determine if the motion is periodic or not. It will be a closed loop for a periodic motion and it won't be closed in the opposite case.

3 Results

3.1 Exercise 1

These are the following graphs for displacement θ and angular velocity ω for a set of initial values.

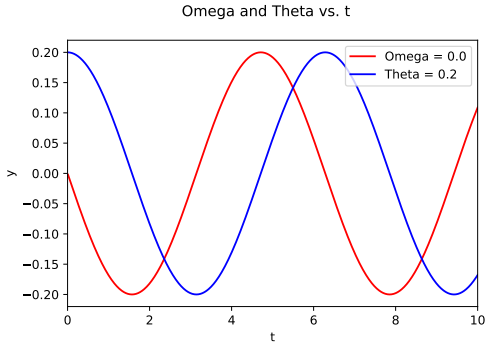

Figure 1: $\omega = 0.0$ *and* $\theta =$ 0.2 *vs. time*

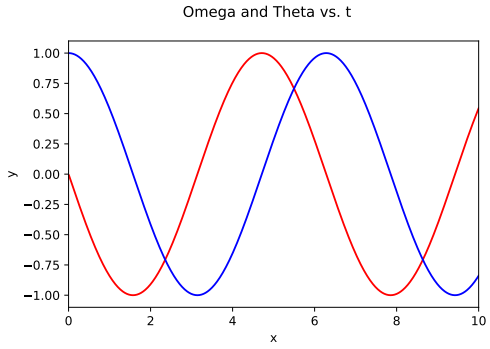

Figure 2: $\omega = 0.0$ *and* $\theta =$ 1.0 *vs. time*

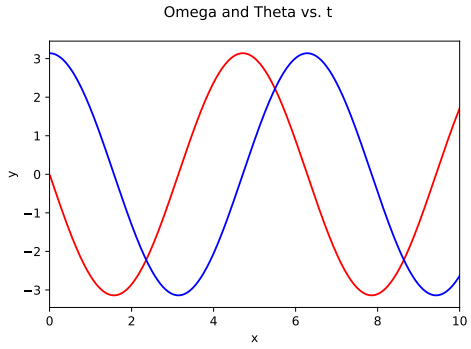

Figure 3: $\omega = 0.0$ *and* $\theta =$ 3.14 *vs. time*

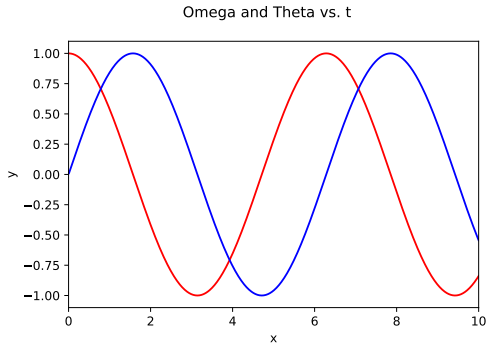

Figure 4: $\omega = 1.0$ *and* $\theta =$ 0.0 *vs. time*

3.2 Exercise 2

The following set of graphs compares the omega and theta concerning time for linear and nonlinear pendulum values.

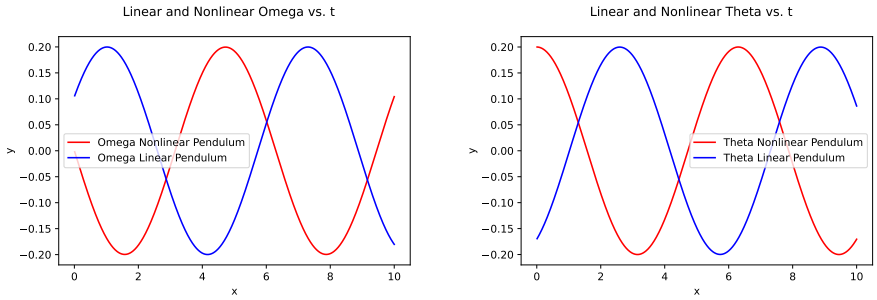

Figure 5: $\omega = 0.0$ *and* $\theta =$ 0.2 *vs. time*, Linear and Nonlinear.

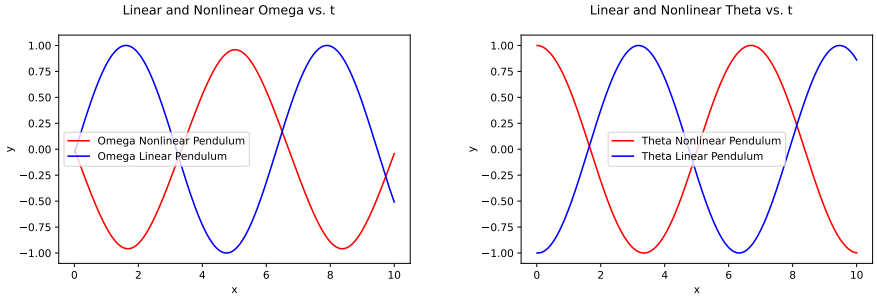

Figure 6: $\omega = 0.0$ *and* $\theta =$ 1.0 *vs. time*, Linear and Nonlinear.

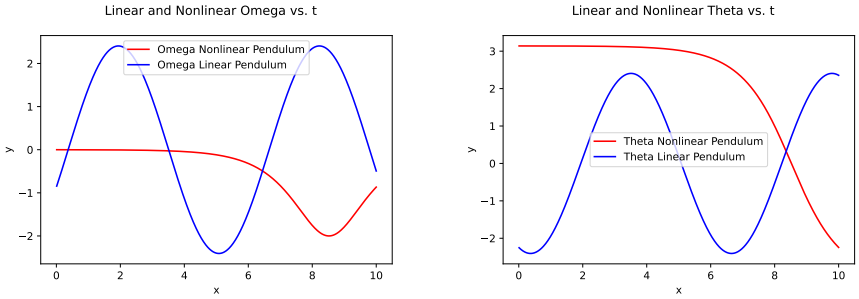

Figure 7: $\omega = 0.0$ *and* $\theta =$ 3.14 *vs. time*, Linear and Nonlinear.

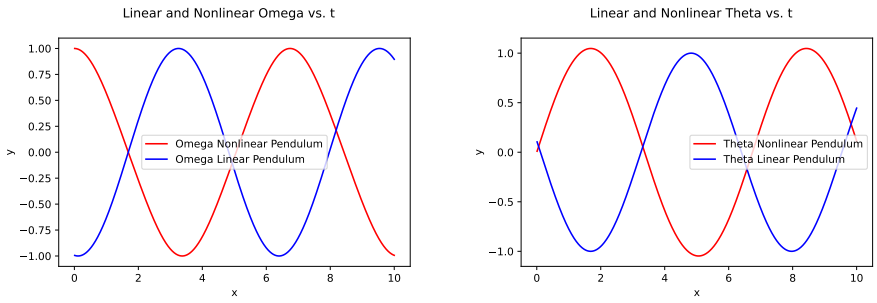

Figure 8: $\omega = 1.0$ *and* $\theta =$ 0.0 *vs. time*, Linear and Nonlinear.

3.3 Exercise 3

The following two graphs show the Trapezoid and RK4 methods with theta = 3.1415, and omega = 0.0. From the graph, we can see that the values of theta are the same but the curves start shifting out of phase as time increases.

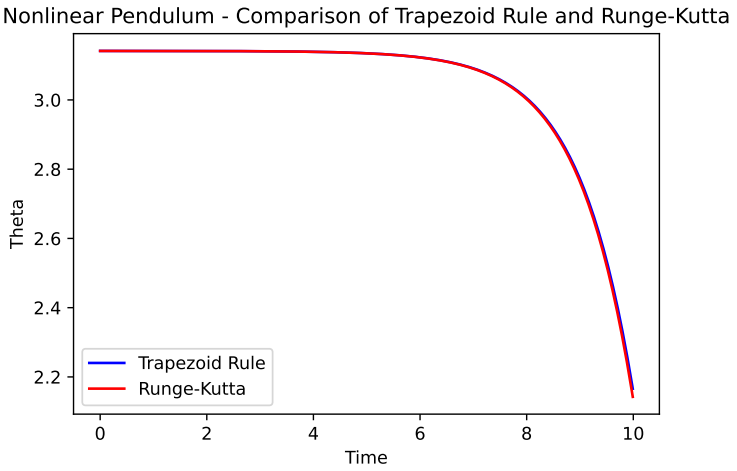

Figure 9: RK4 and Trapezoid Method

3.4 Exercise 4

The following graphs are for a damped nonlinear pendulum, with $\omega = 0.0$, $\theta = 3.0$, and k = 0.5. My section's graphs illustrate how damping works over time. They demonstrate how oscillation amplitude gradually decreases over time. Light damping is the term used to describe this level of damping.

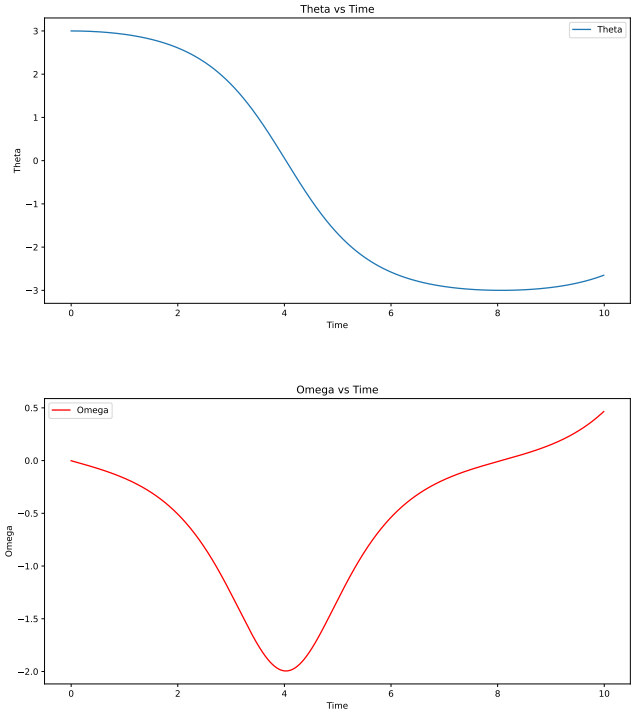

Figure 10: Graphs of θ(t) and ω(t) of the damped pendulum.

3.5 Exercise 5

The following phase portraits are for the set of amplitudes A = 0.90, 1.07, 1.35, 1.47 and 1.5 with fixed values k = 0.5, phi = 0.66667.

The first phase portrait is a representation of periodic motion, as we can see a single loop. The system experiences oscillatory motion before reaching a steady-state periodic behaviour. As the amplitude increases, this pattern starts to break down. The second portrait still shows a periodic motion but switches as it oscillates, resulting in period doubling. The third phase portrait breaks out of periodic oscillation but follows a clear path. The same for the fourth portrait, although the motion is increasingly more complex. In the final one, we can see a chaotic behaviour.

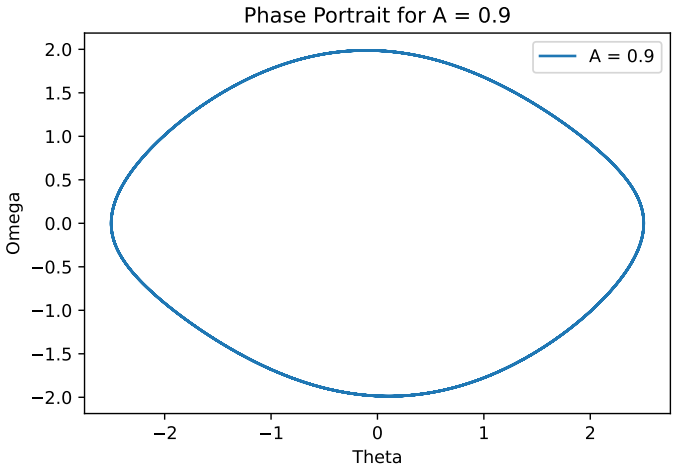

Figure 11: A = 0.9

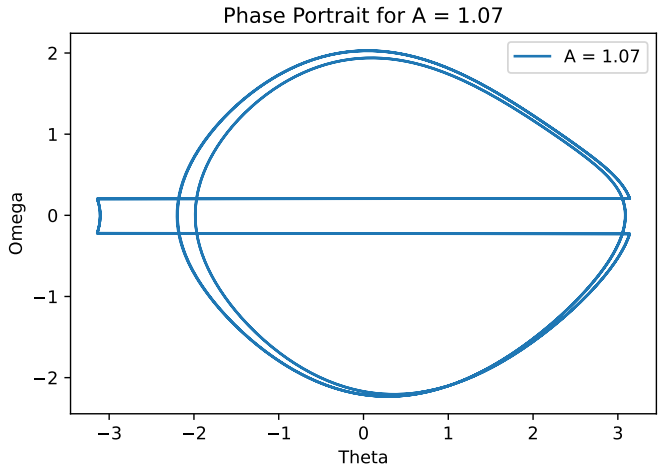

Figure 12: A = 1.07

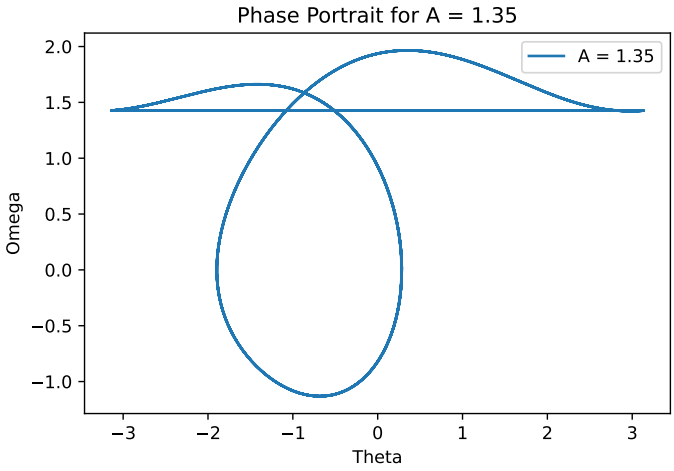

Figure 13: A = 1.35

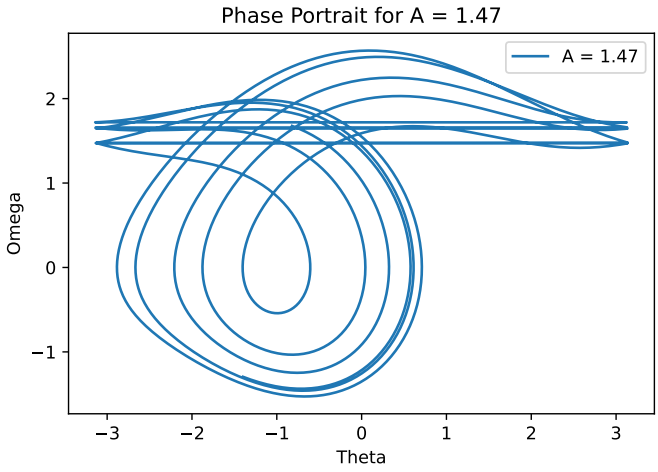

Figure 14: A = 1.47

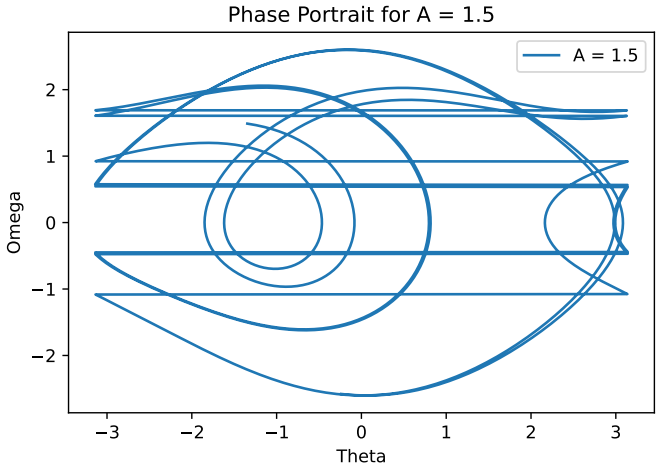

Figure 15: A = 1.5

4 Conclusion

In this computational lab, the dynamics of a nonlinear pendulum were investigated, and we used numerical techniques to solve the coupled ordinary differential equations (ODEs) that characterize its motion. Newton's second law served as the basis for the equation of motion for the pendulum, which took into account the effects of damping, gravity, and an outside driving force. For the initial undamped linear and nonlinear pendulum simulation, damping and the resisting force were discarded.

In order to demonstrate the resulting equations and their analytical solution, we started by simulating the motion of a linear pendulum at small angles. Next, we moved to a more complicated case where the pendulum equation was modified to include nonlinearity, damping, and a sinusoidal driving force. We were able to solve the nonlinear pendulum equations efficiently thanks to the numerical techniques, which included the trapezoid rule and the more precise RK4 method.

We then generated phase portraits for the pendulum motion, using a parameter set of amplitude values. The phase portraits visually represented the system's behaviour in phase space. This offered us insights into periodic and chaotic motion, showing that as the amplitude increased, the motion became more chaotic.

To conclude, this lab exercise studied the significance of using numerical techniques for solving ODEs in our scripts. We were able to include this for useful analysis of the periodic motion for nonlinear dynamical systems. The phase portraits demonstrated the complex dynamics that may arise from seemingly basic physical systems and were an effective tool for viewing and analyzing

the complex motion of the driven damped nonlinear pendulum.

5 Appendix

Exercise 1:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 19 14:02:07 2024

@author: watanabv
"""
#script to solve linear pendulum equation
import numpy as np
import matplotlib.pylab as plt

k = 0.0
phi = 0.66667
A = 0.0
theta = 0.2

omega = 0.0
t = 0.0
dt = 0.01
nsteps = 0

omegaval = []
thetaval = []
nstepsval =[]
tval = []
theta2 =[0.2,1.0,3.14,0.0]
omega2 =[0.0,0.0,0.0,1.0]

def force(theta, omega,t):
    return -theta - k*omega + A*np.cos(phi)*t

#prints value of force function
print(force(theta, omega, t))

for nsteps in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * force(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps += 1
    omegaval.append(omega)
```

```
    thetaval.append(theta)
    nstepsval.append(nsteps)
    tval.append(t)


plt.plot(tval, omegaval,'r', label = 'Omega')
plt.plot(tval, thetaval,'b', label = 'Theta')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Omega and Theta vs. t')
plt.legend()
plt.show()

plt.plot(nstepsval, thetaval, label = 'Theta')
plt.plot(nstepsval, omegaval, label = 'Omega')
plt.suptitle("Omega and Theta vs. nsteps")
plt.xlabel("nsteps")
plt.xlim(0,500)
plt.ylabel("y")
plt.ylim(-np.pi, np.pi)
plt.legend()
plt.show()

def replace(omega, theta):
    omegaval = []
    thetaval = []
    tval = []
    t = 0
    for i in range(1000):
        k1a = dt * omega
        k1b = dt * force(theta, omega, t)
        k2a = dt * (omega + k1b)
        k2b = dt * force(theta + k1a, omega + k1b, t + dt)
        theta = theta + (k1a + k2a) / 2
        omega = omega + (k1b + k2b ) / 2
        t = t + dt
        omegaval.append(omega)
        thetaval.append(theta)
        tval.append(t)
    plt.plot(tval, omegaval,'r', label = 'Omega')
    plt.plot(tval, thetaval,'b', label = 'Theta')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.suptitle('Omega and Theta vs. t')
    plt.xlim(0,10)
    plt.legend()
```

```
    plt.show()

replace(0.0,0.2)

replace(0.0,1.0)

replace(0.0,3.14)

replace(1.0,0.0)
```

Exercise 2:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 19 14:02:07 2024

@author: watanabv
"""
#script to solve linear pendulum equation
import numpy as np
import matplotlib.pylab as plt

k = 0.0
phi = 0.66667
A = 0.0



def force(theta, omega,t):
    return -np.sin(theta) - k*omega + A*np.cos(phi)*t

def force1(theta, omega,t):
    return -theta - k*omega + A*np.cos(phi)*t

theta = 0.2

omega = 0.0

t = 0.0
dt = 0.01
nsteps = 0

omegaval = []
thetaval = []
nstepsval =[]
```

```
tval = []

for nsteps in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * force(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps += 1
    omegaval.append(omega)
    thetaval.append(theta)
    nstepsval.append(nsteps)
    tval.append(t)

t1 = 0.0
dt1 = 0.01
nsteps1 = 0

omegaval1 = []
thetaval1 = []
nstepsval1 =[]
tval1 = []
for nsteps1 in range(1000):
    k1a = dt1 * omega
    k1b = dt1 * force1(theta, omega, t1)
    k2a = dt1 * (omega + k1b)
    k2b = dt1 * force1(theta + k1a, omega + k1b, t1 + dt1)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t1 = t1 + dt1
    nsteps1 += 1
    omegaval1.append(omega)
    thetaval1.append(theta)
    nstepsval1.append(nsteps1)
    tval1.append(t1)


plt.plot(tval, omegaval,'r', label = 'Omega Nonlinear Pendulum')
plt.plot(tval1, omegaval1,'b', label = 'Omega Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Omega vs. t')
plt.legend()
plt.savefig('EX2.2 1.0.pdf')
```

```
plt.show()

plt.plot(tval, thetaval,'r', label = 'Theta Nonlinear Pendulum')
plt.plot(tval1, thetaval1,'b', label = 'Theta Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Theta vs. t')
plt.legend()
plt.savefig('EX2.2 1.1.pdf')
plt.show()



theta = 1.0

omega = 0.0

t = 0.0
dt = 0.01
nsteps = 0

omegaval = []
thetaval = []
nstepsval =[]
tval = []

for nsteps in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * force(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps += 1
    omegaval.append(omega)
    thetaval.append(theta)
    nstepsval.append(nsteps)
    tval.append(t)

t1 = 0.0
dt1 = 0.01
nsteps1 = 0

omegaval1 = []
thetaval1 = []
```

```
nstepsval1 =[]
tval1 = []
for nsteps1 in range(1000):
    k1a = dt1 * omega
    k1b = dt1 * force1(theta, omega, t1)
    k2a = dt1 * (omega + k1b)
    k2b = dt1 * force1(theta + k1a, omega + k1b, t1 + dt1)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t1 = t1 + dt1
    nsteps1 += 1
    omegaval1.append(omega)
    thetaval1.append(theta)
    nstepsval1.append(nsteps1)
    tval1.append(t1)


plt.plot(tval, omegaval,'r', label = 'Omega Nonlinear Pendulum')
plt.plot(tval1, omegaval1,'b', label = 'Omega Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Omega vs. t')
plt.legend()
plt.savefig('EX2.2 1.2.pdf')
plt.show()

plt.plot(tval, thetaval,'r', label = 'Theta Nonlinear Pendulum')
plt.plot(tval1, thetaval1,'b', label = 'Theta Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Theta vs. t')
plt.legend()
plt.savefig('EX2.2 1.3.pdf')
plt.show()




theta = 3.14

omega = 0.0

t = 0.0
dt = 0.01
nsteps = 0
```

```
omegaval = []
thetaval = []
nstepsval =[]
tval = []

for nsteps in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * force(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps += 1
    omegaval.append(omega)
    thetaval.append(theta)
    nstepsval.append(nsteps)
    tval.append(t)

t1 = 0.0
dt1 = 0.01
nsteps1 = 0

omegaval1 = []
thetaval1 = []
nstepsval1 =[]
tval1 = []
for nsteps1 in range(1000):
    k1a = dt1 * omega
    k1b = dt1 * force1(theta, omega, t1)
    k2a = dt1 * (omega + k1b)
    k2b = dt1 * force1(theta + k1a, omega + k1b, t1 + dt1)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t1 = t1 + dt1
    nsteps1 += 1
    omegaval1.append(omega)
    thetaval1.append(theta)
    nstepsval1.append(nsteps1)
    tval1.append(t1)


plt.plot(tval, omegaval,'r', label = 'Omega Nonlinear Pendulum')
plt.plot(tval1, omegaval1,'b', label = 'Omega Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.suptitle('Linear and Nonlinear Omega vs. t')
plt.legend()
plt.savefig('EX2.2 1.4.pdf')
plt.show()

plt.plot(tval, thetaval,'r', label = 'Theta Nonlinear Pendulum')
plt.plot(tval1, thetaval1,'b', label = 'Theta Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Theta vs. t')
plt.legend()
plt.savefig('EX2.2 1.5.pdf')
plt.show()


theta = 0.0

omega = 1.0




t = 0.0
dt = 0.01
nsteps = 0

omegaval = []
thetaval = []
nstepsval =[]
tval = []

for nsteps in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * force(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps += 1
    omegaval.append(omega)
    thetaval.append(theta)
    nstepsval.append(nsteps)
    tval.append(t)
```

```
t1 = 0.0
dt1 = 0.01
nsteps1 = 0

omegaval1 = []
thetaval1 = []
nstepsval1 =[]
tval1 = []
for nsteps1 in range(1000):
    k1a = dt1 * omega
    k1b = dt1 * force1(theta, omega, t1)
    k2a = dt1 * (omega + k1b)
    k2b = dt1 * force1(theta + k1a, omega + k1b, t1 + dt1)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t1 = t1 + dt1
    nsteps1 += 1
    omegaval1.append(omega)
    thetaval1.append(theta)
    nstepsval1.append(nsteps1)
    tval1.append(t1)


plt.plot(tval, omegaval,'r', label = 'Omega Nonlinear Pendulum')
plt.plot(tval1, omegaval1,'b', label = 'Omega Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Omega vs. t')
plt.legend()
plt.savefig('EX2.2 1.6.pdf')
plt.show()

plt.plot(tval, thetaval,'r', label = 'Theta Nonlinear Pendulum')
plt.plot(tval1, thetaval1,'b', label = 'Theta Linear Pendulum')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Linear and Nonlinear Theta vs. t')
plt.legend()
plt.savefig('EX2.2 1.7.pdf')
plt.show()

Exercise 3:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  4 03:19:47 2024
```

```
@author: vwitch
"""

import numpy as np
import matplotlib.pylab as plt

k = 0.0
phi = 0.66667
A = 0.0
theta = 0.31415

omega = 0.0
t = 0.0
dt = 0.01
nsteps = 0

omegaval = []
thetaval = []
nstepsval =[]
tval = []


def force(theta, omega,t):
    return -np.sin(theta) - k*omega + A*np.cos(phi)*t

#prints value of force function
print(force(theta, omega, t))

for nsteps in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b)
    k2b = dt * force(theta + k1a, omega + k1b, t + dt)
    theta = theta + (k1a + k2a) / 2
    omega = omega + (k1b + k2b ) / 2
    t = t + dt
    nsteps += 1
    omegaval.append(omega)
    thetaval.append(theta)
    nstepsval.append(nsteps)
    tval.append(t)

omegaval1 = []
thetaval1 = []
nstepsval1 =[]
```

```
tval1 = []

for nsteps1 in range(1000):
    k1a = dt * omega
    k1b = dt * force(theta, omega, t)
    k2a = dt * (omega + k1b/2)
    k2b = dt * force(theta + k1a/2, omega + k1b/2, t + dt/2)
    k3a = dt * (omega + k2b/2)
    k3b = dt * force(theta + k2a/2, omega + k2b/2, t + dt/2)
    k4a = dt * (omega + k3b)
    k4b = dt * force(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
    omegaval1.append(omega)
    thetaval1.append(theta)
    nstepsval.append(nsteps1)
    tval1.append(t)




plt.plot(tval1, thetaval1,'r', label = 'RK4')
plt.plot(tval, thetaval,'b', label = 'Trapezoid Method')
plt.xlabel('t')
plt.ylabel('theta')
plt.suptitle('Theta vs. t')
plt.legend()
plt.savefig('Lab 2 Ex30.pdf')
plt.show()

Exercise 4:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 26 14:19:13 2024

@author: vwitch
"""
import numpy as np
import matplotlib.pyplot as plt

# Function definition for the damped, driven, nonlinear pendulum
def f(theta, omega, t, A=0.0, k=0.0, phi=0.0):
    g_over_L = 1.0  # gravitational constant over length
```

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

# Initialize parameters and initial conditions
theta = 3.0
omega = 0.0
t = 0.0
dt = 0.01
nsteps = 1000

# Lists to store values for plotting
theta_values = []
omega_values = []

# Runge-Kutta integration
for _ in range(nsteps):
    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)

    # Update theta and omega using the Runge-Kutta method
    theta = theta + (k1a + 2*k2a + 2*k3a + k4a) / 6
    omega = omega + (k1b + 2*k2b + 2*k3b + k4b) / 6
    t = t + dt

    # Append values for plotting
    theta_values.append(theta)
    omega_values.append(omega)

# Plotting Theta versus Time
plt.figure(figsize=(12, 6))
plt.plot(np.arange(nsteps) * dt, theta_values, label='Theta')
plt.title('Theta vs Time')
plt.xlabel('Time')
plt.ylabel('Theta')
plt.legend()
plt.savefig('Ex4 final 1.pdf')
plt.show()

# Plotting Omega versus Time
plt.figure(figsize=(12, 6))
plt.plot(np.arange(nsteps) * dt, omega_values, 'r',label='Omega')
```

```
plt.title('Omega vs Time')
plt.xlabel('Time')
plt.ylabel('Omega')
plt.legend()
plt.savefig('Ex4 final 2.pdf')
plt.show()

Exercise 5:

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 26 15:16:28 2024

@author: vwitch
"""

import numpy as np
import matplotlib.pylab as plt

k = 0.5
phi = 0.66667
A_values = [0.90, 1.07, 1.35, 1.47, 1.5]
transient = 5000

for A in A_values:
    omega = 0.0
    theta = 3.0
    t = 0.0
    dt = 0.01
    nsteps = 0
    iteration_number = 0

    def force(theta, omega, t):
        return -np.sin(theta) - k * omega + A * np.cos(phi * t)

    omegaval = []
    thetaval = []
    nstepsval = []
    tval = []

    for nsteps in range(10000):
        k1a = dt * omega
        k1b = dt * force(theta, omega, t)
        k2a = dt * (omega + k1b / 2)
        k2b = dt * force(theta + k1a / 2, omega + k1b / 2, t + dt / 2)
```

```
        k3a = dt * (omega + k2b / 2)
        k3b = dt * force(theta + k2a / 2, omega + k2b / 2, t + dt / 2)
        k4a = dt * (omega + k3b)
        k4b = dt * force(theta + k3a, omega + k3b, t + dt)

        theta = theta + (k1a + 2 * k2a + 2 * k3a + k4a) / 6
        omega = omega + (k1b + 2 * k2b + 2 * k3b + k4b) / 6

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

        t = t + dt
        nsteps += 1
        iteration_number += 1

        if iteration_number > transient:
            omegaval.append(omega)
            thetaval.append(theta)
            nstepsval.append(nsteps)
            tval.append(t)

plt.plot(thetaval, omegaval, label=f'A = {A}')
plt.xlabel('Theta')
plt.ylabel('Omega')
plt.title(f'Phase Portrait for A = {A}')
plt.legend()
plt.show()
```