Assignment

Pendulum With Higher Oszillation

Name TN 1: Tim Peinkofer

tim.peinkofer@student.uni-tuebingen.de

Name TN 2: Fabian Kostow

fabian.kostow@student.uni-tuebingen.de

Tutor: Jose Carlos Olvera Meneses

Date: January 26, 2025

Contents

1 Theory **2**

2 Code **3**

3 Results **6**

1 Theory

The ODE of a pendulum is given by

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = 0$$

we can rewrite this second order ODE into a system of 2 first order ODE:

$$\begin{aligned} \theta' &= \omega \\ \omega' &= -\frac{g}{l} sin(\theta) \end{aligned}$$

This system of equations can be solved using Runge-Kutta.

2 Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation


#1 second order DEQ intwo 2 first order DEQ
def ThetaPrime(omega):
    return omega

def OmegaPrime(theta):
    return -(g / l) * np.sin(theta)

# Runge-Kutta 4th-order
def runge_kutta(theta_0, omega_0, t0, t_end, dt):
    t = np.arange(t0, t_end + dt, dt)
    theta = np.zeros(len(t))
    omega = np.zeros(len(t))
    theta[0] = theta_0
    omega[0] = omega_0

    for i in range(1, len(t)):
        k1_theta = ThetaPrime(omega[i - 1])
        k1_omega = OmegaPrime(theta[i - 1])

        k2_theta = ThetaPrime(omega[i - 1] + dt * k1_omega /
                                   2)
        k2_omega = OmegaPrime(theta[i - 1] + dt * k1_theta /
                                   2)

        k3_theta = ThetaPrime(omega[i - 1] + dt * k2_omega /
                                   2)
        k3_omega = OmegaPrime(theta[i - 1] + dt * k2_theta /
                                   2)

        k4_theta = ThetaPrime(omega[i - 1] + dt * k3_omega)
        k4_omega = OmegaPrime(theta[i - 1] + dt * k3_theta)

        theta[i] = theta[i - 1] + (dt / 6) * (k1_theta + 2 *
                                   k2_theta + 2 *
                                   k3_theta + k4_theta)
        omega[i] = omega[i - 1] + (dt / 6) * (k1_omega + 2 *
                                   k2_omega + 2 *
                                   k3_omega + k4_omega)

    return t, theta, omega
```

```
#Constants
g = 9.81
l = 1.0
# Initial conditions
dt = 0.01
t_max = 15
t0=0
theta_0 = np.pi / 1.1
omega_0 = 0

# Solve the system
t, theta, omega = runge_kutta(theta_0, omega_0, t0, t_max, dt
                              )

# Plot theta(t)
plt.figure(figsize=(8, 4))
plt.plot(t, theta, label='Theta (rad)')
plt.title('Pendulum Motion')
plt.xlabel('Time (s)')
plt.ylabel('Theta (rad)')
plt.legend()
plt.grid()
plt.show()
#phasendiagramm
plt.figure(figsize=(6, 6))
plt.plot(theta, omega, label='Phase Space')
plt.title('Phase Diagram')
plt.xlabel('Theta (rad)')
plt.ylabel('Omega (rad/s)')
plt.legend()
plt.grid()
plt.show()

# Animation
x = l * np.sin(theta)
y = -l * np.cos(theta)

fig, ax = plt.subplots(figsize=(6, 6))
ax.set_xlim(-l-0.1, l+0.1)
ax.set_ylim(-l-0.1, l+0.1)
ax.set_aspect('equal')
ax.grid()

line, = ax.plot([], [], 'o-', lw=2)
trace, = ax.plot([], [], 'r-', lw=1, alpha=0.6)
trajectory_x, trajectory_y = [], []
```

```
def init():
    line.set_data([], [])
    trace.set_data([], [])
    return line, trace

def update(frame):
    trajectory_x.append(x[frame])
    trajectory_y.append(y[frame])

    line.set_data([0, x[frame]], [0, y[frame]])
    trace.set_data(trajectory_x, trajectory_y)
    return line, trace

ani = FuncAnimation(fig, update, frames=len(t), init_func=
                               init, blit=True, interval=dt*
                               1000)
plt.show()
```

3 Results

The animation can be seen by running the code.

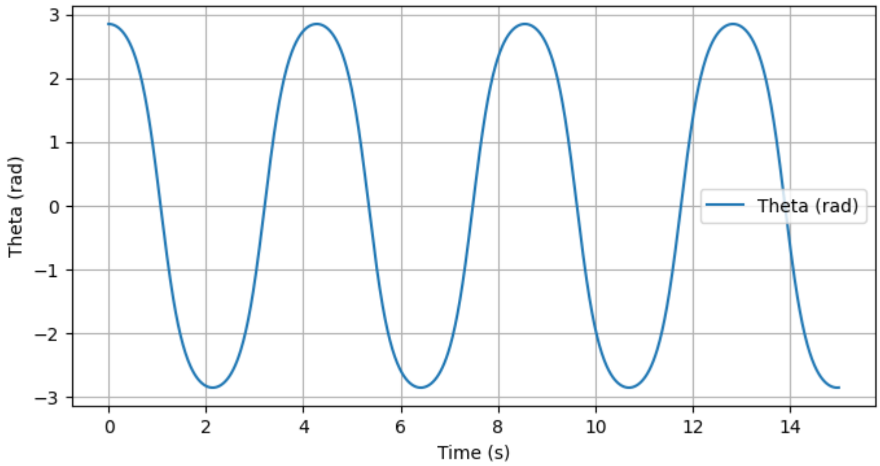

Figure 1:

Here we can see that the solution of this ODE is not a sin function, for high deflection.

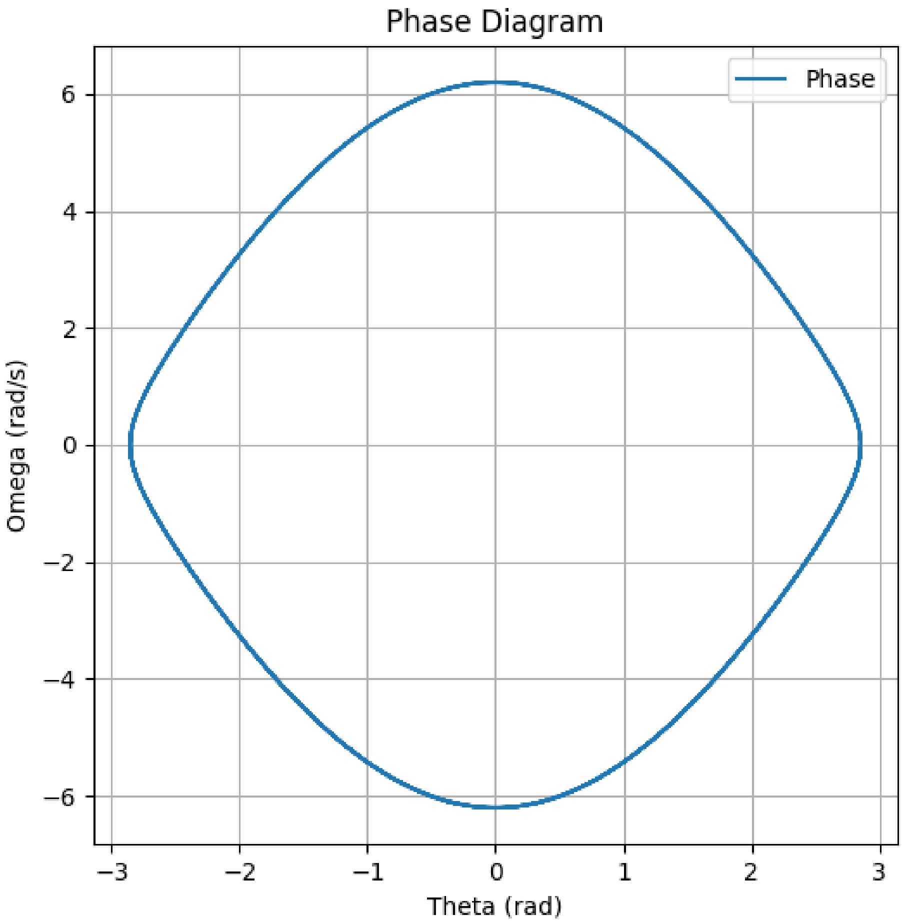

Figure 2: Result for every iteration

In the phase diagram, we see that the curve is closed; this is because energy is conserved and there is no rollover.