1. **Introduction:**
   For this homework, the objective was to simulate a system where the mass is unknown. For the simulation, an adaptive control system was used. The adaptive control system was not given the mass but instead estimates the mass. While the mass is estimated, the controller provides control to the system.
2. **Results and Discussion:**

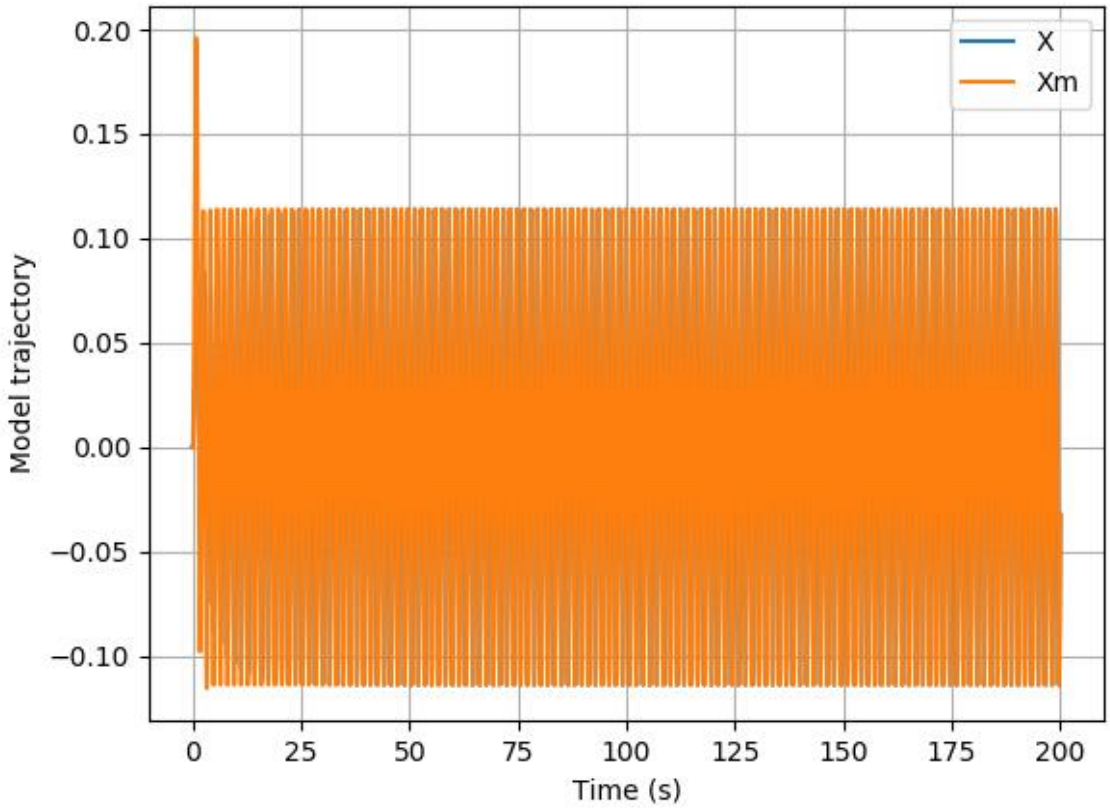


*Figure 1: The desired model trajectory and actual model trajectory plotted against time*

In Fig. 1, the model trajectory of the system was tracked in order to see how the actual model trajectory measured against the desired model trajectory. The actual model trajectory changes as the mass is estimated, so until the mass is found, the actual model trajectory and the desired model trajectory will not line up. They will instead converge and line up when the mass is estimated.

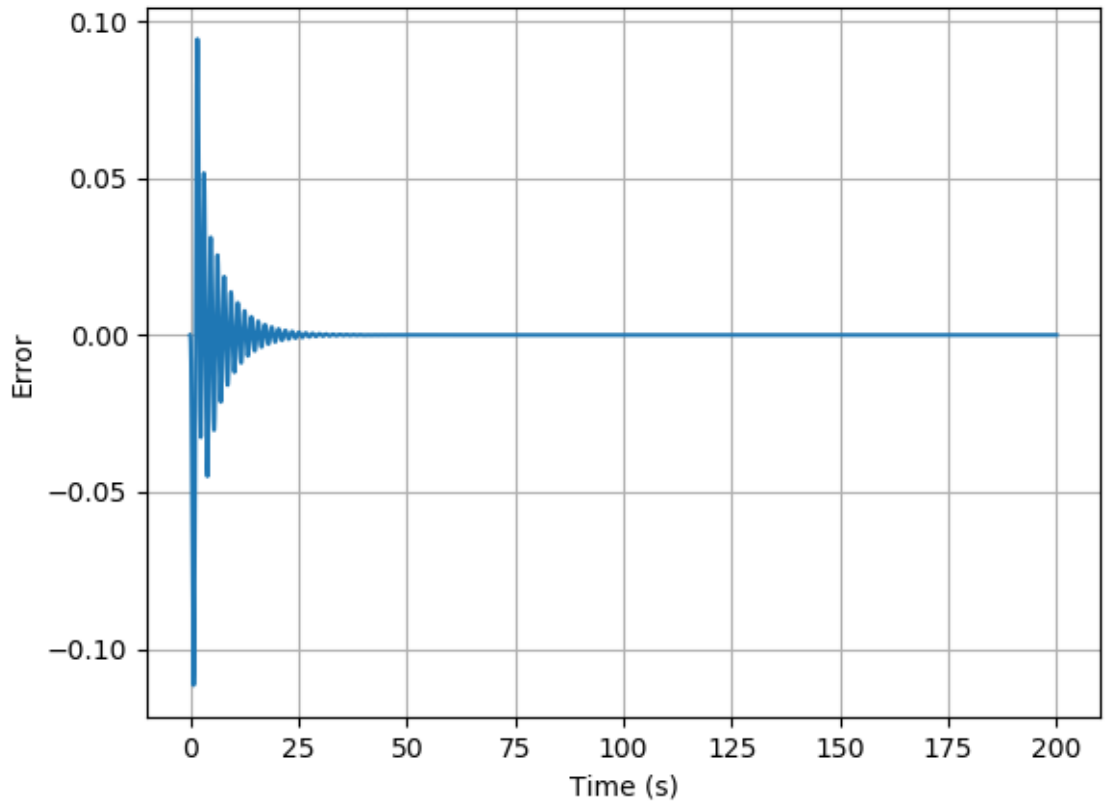


*Figure 2: The error position signal plotted against time*

In Fig. 2, the error signal of the system was plotted against time to ensure the system has no steady state error. The error signal is the desired model trajectory subtracted from the actual model trajectory. As the actual model trajectory changes with the mass estimation, the error will converge to zero. This is because the actual model trajectory will estimate the actual mass and will equal the desired model trajectory.

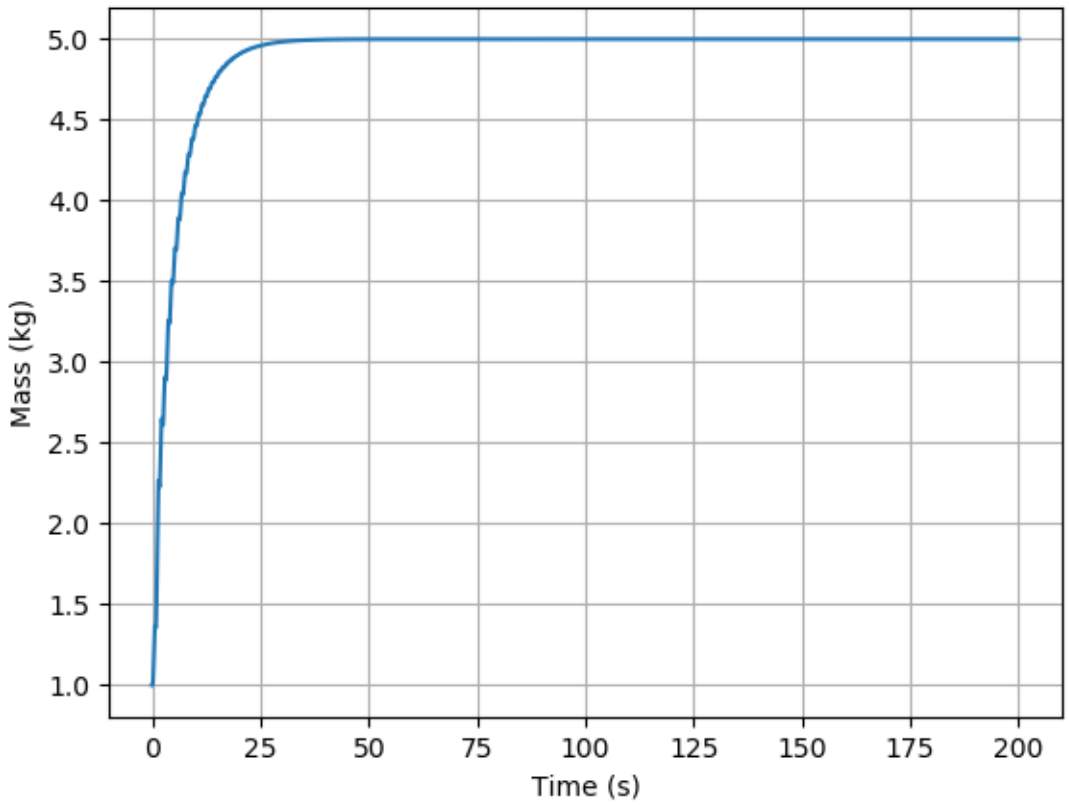


*Figure 3: The mass estimation plotted against time*

In Fig. 3, the mass estimation of the system was plotted against time to show how quickly the system can accurately estimate the mass. The controller does not initially know the mass of the system, so the initial mass estimated is zero. As the controller loops through iterations in order to reduce the error to zero, the mass is eventually found around 30 seconds.

## 3. Appendix:

### Code:

```
import numpy as np

import matplotlib.pyplot as plt

import scipy.integrate as ode

def Derivatives(state,t):

    ##States

    x = state[0]

    xdot = state[1]
```

```
xm = state[2]
xmdot = state[3]
mhat = state[4]
###Parameters
m = 5.0
lam = 3.0
zeta = 0.8
wn = 2.0
bm = 2.0
gam = 3.0
###reference signal
r = np.sin(4*t)
###Error Signals
xtilde = x - xm
xtildedot = xdot - xmdot
###Model Dynamics
xmddot = bm*r - 2*zeta*wn*xmdot - wn**2*xm
###Control
kd = 2*zeta*wn
kp = wn**2
v = xmddot - kd*xtildedot - kp*xtilde
u = mhat*v
###Plant Dynamics and Kinematics
#xdot already defined
xddot = u/m
###Adaptive Control Dynamics
s = xtildedot + lam*xtilde
#mhatdot = -v*s*gam
mhatdot = -gam*xtildedot*v
return np.asarray([xdot,xddot,xmdot,xmddot,mhatdot])
```

```
##Time vector
tout = np.linspace(0,200,10000)
##Initial Conditions
x0 = 0
xdot0 = 0
xm0 = x0
xmdot0 = xdot0
mhat0 = 1.0
state_initial = np.asarray([x0,xdot0,xm0,xmdot0,mhat0])
state_out = ode.odeint(Derivatives,state_initial,tout)
###Extract my states
xout = state_out[:,0]
xmout = state_out[:,2]
mhatout = state_out[:,4]
plt.figure()
plt.plot(tout,xout,label='X')
plt.plot(tout,xmout,label='Xm')
plt.xlabel('Time (s)')
plt.ylabel('Model trajectory')
plt.legend()
plt.grid()
plt.figure()
plt.plot(tout,xout-xmout)
plt.xlabel('Time (s)')
plt.ylabel('Error')
plt.grid()
plt.figure()
plt.plot(tout,mhatout)
plt.xlabel('Time (s)')
plt.ylabel('Mass (kg)')
```

```
plt.grid()
plt.show()
```