Maxwell Cobar
Homework #1
Non-Linear Controls

Simple Pendulum Python Code without Control:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as S
plt.close("all")
g = 9.81
L = 1.0
m = 5.0
K = 10.0
b = 10.0
c = 0.0
theta0 = 90*np.pi/180.
thetac = 0*30*np.pi/180.
def controller(x):
    thetacdot = 0.0
    theta = x[0]
    thetadot = x[1]
    eint = x[2]
    e = thetac - theta
    edot = thetacdot - thetadot
    #T = K(s^2 + b*s + c)/s
    #T = K*s + K*b + K*c/s
    T = 0.0 #K*b*e + K*edot + K*c*eint
    return T,e
def DerivativesNL(x,t):
    theta = x[0]
    thetadot = x[1]
    eint = x[2]
```

```
    T,e = controller(x)
    thetaddot = -g/L*np.sin(theta) + T/(m*L**2)
    eintdot = e
    xdot = np.asarray([thetadot,thetaddot,eintdot])
    return xdot
def Derivatives(x,t):
    theta = x[0]
    thetadot = x[1]
    eint = x[2]
    T,e = controller(x)
    xdot = np.matmul(A,x) + B*T + B2*e
    return xdot
A = np.asarray([[0,1,0],[-g/L,0,0],[0,0,0]])
B = np.asarray([0,1/(m*L**2),0])
B2 = np.asarray([0,0,1])
#eigs = np.linalg.eig(A)
#print(eigs)
xinitial = np.asarray([theta0,0,0])
tout = np.linspace(0,20,1000)
xout = S.odeint(Derivatives,xinitial,tout)
xoutNL = S.odeint(DerivativesNL,xinitial,tout)
plt.plot(tout,xout[:,0]*180/np.pi,label="Linear")
plt.plot(tout,xoutNL[:,0]*180./np.pi,'r-',label="Nonlinear")
plt.xlabel('Time (sec)')
plt.ylabel('Angle (deg)')
plt.legend()
plt.grid()
plt.show()
```

Maxwell Cobar
Homework #1
Non-Linear Controls

For this homework, the objective was to simulate a simple pendulum 30°, 60°, and 90°. For the simulation, both a linear and non-linear model were used. Both models were used to show how linear systems are only applicable in ideal conditions and not in the real world, whereas non-linear models can be used for most real-world applications.

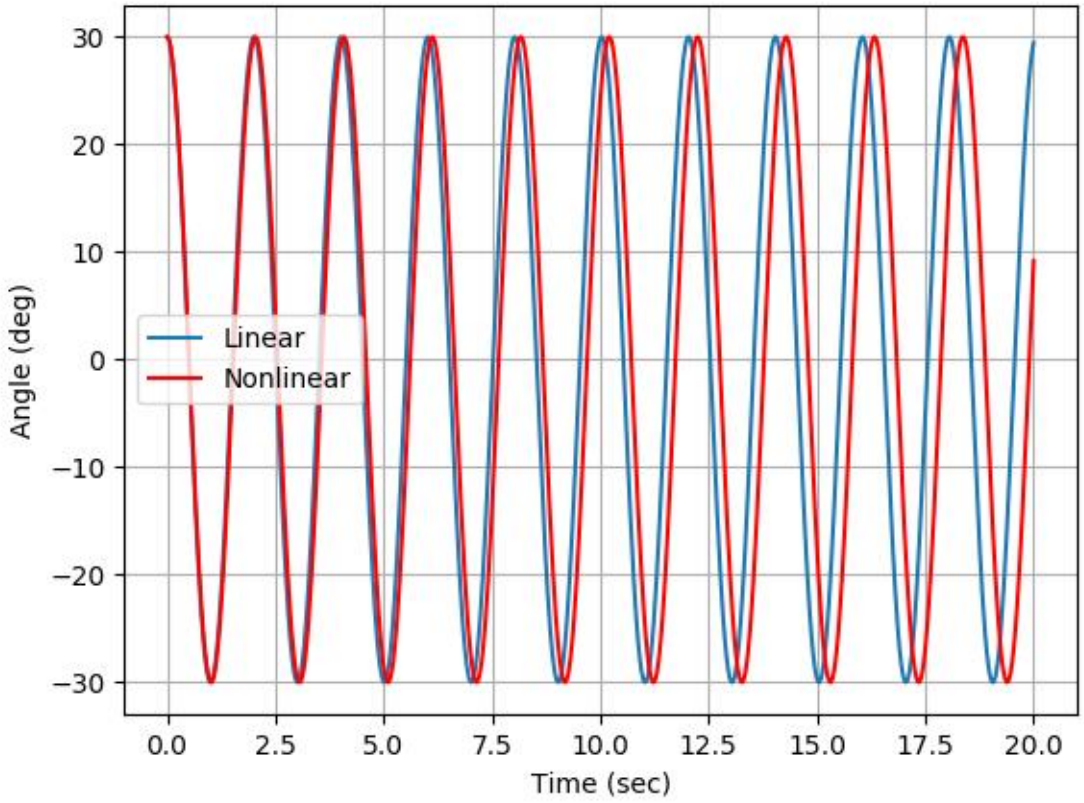

Figure 1: Simple Pendulum modeled at 30 degrees

For Fig. 1, we can see that the linear and non-linear model start out the same, but as the model progress the linear and non-linear models diverge. This divergence is due to the small angle approximation used to compute the linear model. At 30°, the angle is no longer small, and the model becomes inaccurate. However, the non-linear model does not depend on a small angle approximation and remains accurate through the duration of the simulation. Because there is no dampening term, the model does not decay to zero during the simulation.

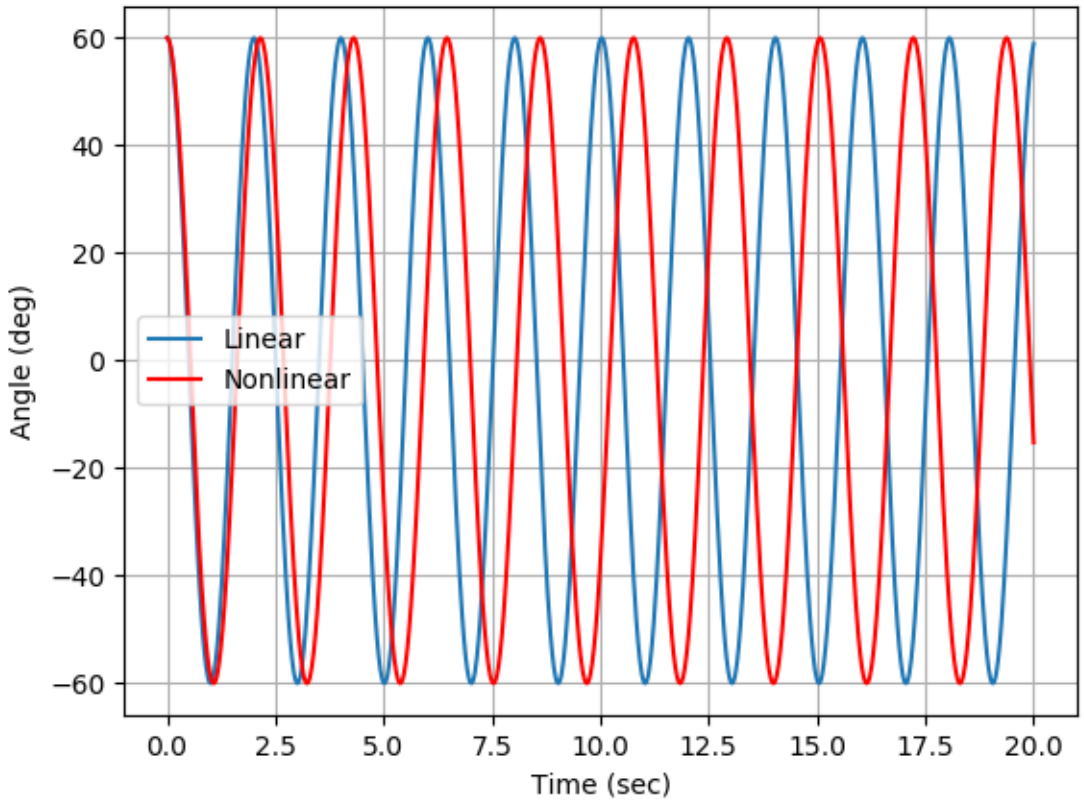

Figure 2: Simple Pendulum modeled at 60 degrees

For Fig. 2, we can see that the linear and non-linear model start out the same, but as the model progress the linear and non-linear models diverge. This divergence is due to the small angle approximation used to compute the linear model. At 60°, the angle is no longer small and is twice as large as the 30° model. This twice larger angle makes the linear and non-linear model diverge more quickly since the small angle approximation does not hold. However, the non-linear model does not depend on a small angle approximation and remains accurate through the duration of the simulation. Because there is no dampening term, the model does not decay to zero during the simulation.

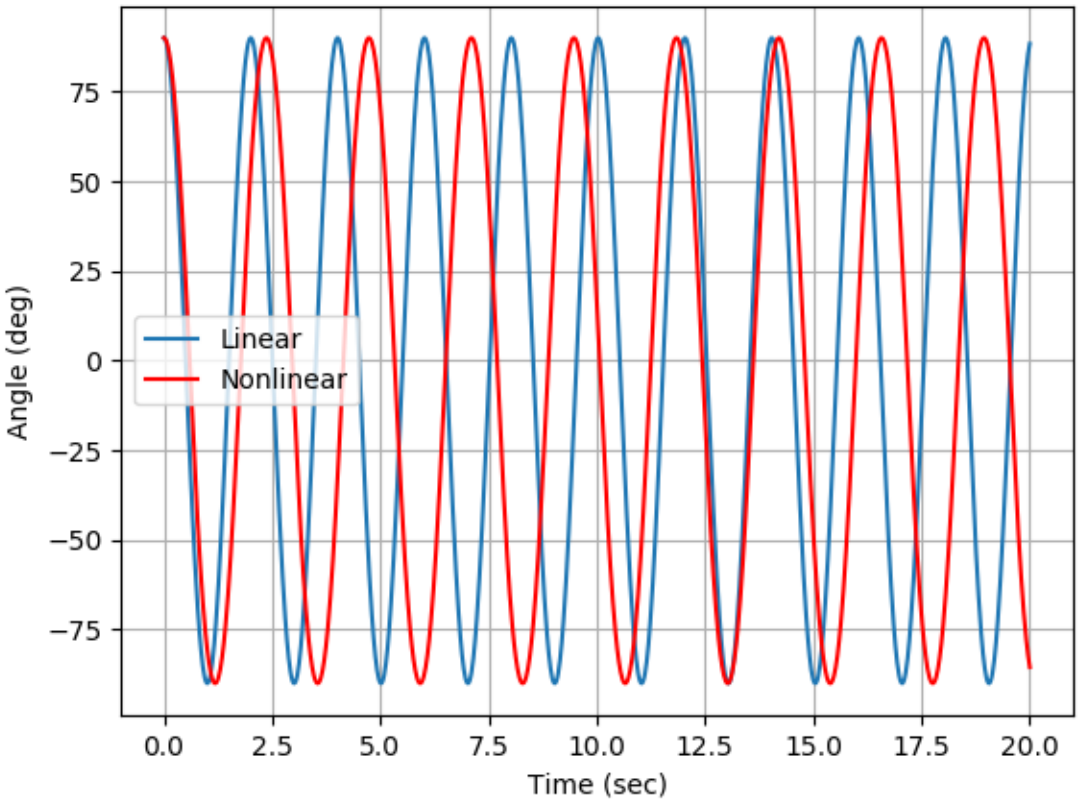

Figure 3: Simple Pendulum modeled at 90 degrees

For Fig. 3, we can see that the linear and non-linear model initially start out the same, but the linear and non-linear models diverge rather quickly. This divergence is due to the small angle approximation used to compute the linear model. At 90°, the angle is very large. This larger angle makes the linear and non-linear model diverge more quickly since the small angle approximation does not hold. However, the non-linear model does not depend on a small angle approximation and remains accurate through the duration of the simulation. Because there is no dampening term, the model does not decay to zero during the simulation.