UNIVERSITY OF MARYLAND,
COLLEGE PARK

# Final Project

Saumil Shah - 116745338
Smriti Gupta - 116748612

ENPM 667

December 16, 2019

# Contents

# List of Figures

# 1 Problem Statement

Consider a crane that moves along an one-dimensional track. It behaves as a frictionless cart with mass $M$ actuated by an external force $F$ that constitutes the input of the system.There are two loads suspended from cables attached to the crane. The loads have mass $m1$ and $m2$, and the lengths of the cables are $l1$ and $l2$, respectively. The following figure depicts the crane and associated variables used throughout this project.

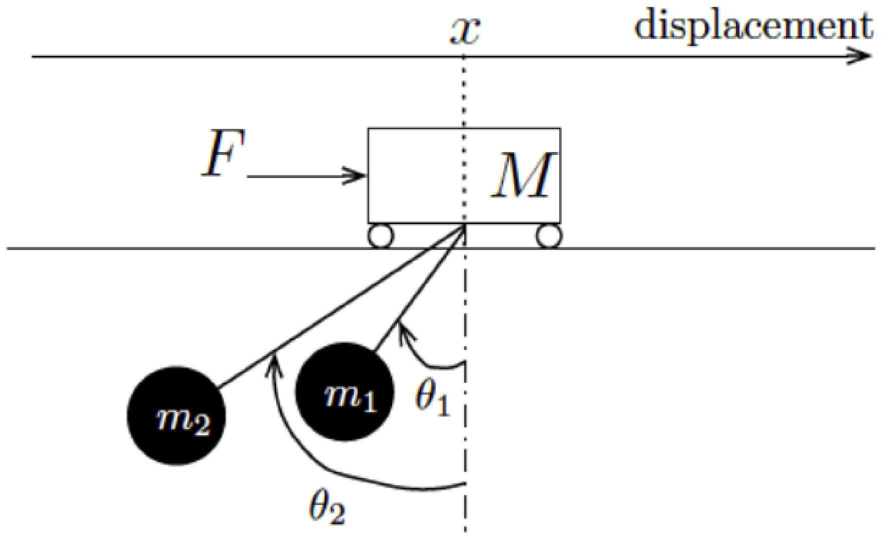


Figure 1.1: Dual pendulum on cart

A) Obtain the equations of motion for the system and the corresponding nonlinear state-space representation.

B) Obtain the linearized system around the equilibrium point specified by $x = 0$ and $\theta_1 = \theta_2 = 0$. Write the state-space representation of the linearized system.

C) Obtain conditions on $M, m1, m2, l1, l2$ for which the linearized system is controllable.

D) Choose $M = 1000Kg$, $m1 = m2 = 100Kg$, $l1 = 20m$ and $l2 = 10m$. Check that the system is controllable and obtain an LQR controller. Simulate the resulting response to initial conditions when the controller is applied to the linearized system and also to the original nonlinear system. Adjust the parameters of the LQR cost until you obtain a suitable response. Use Lyapunov's indirect method to certify stability (locally or globally) of the closed-loop system.

E) Suppose that you can select the following output vectors:$x(t)$, $(\theta_1(t), \theta_2(t))$, $(x(t), \theta_2(t))$ or $(x(t), \theta_1(t), \theta_2(t))$.Determine for which output vectors the linearized system is observable.

F) Obtain your "best" Luenberger observer for each one of the output vectors for which the system is observable and simulate its response to initial conditions and unit step input. The simulation should be done for the observer applied to both the linearized system and the original nonlinear system.

G) Design an output feedback controller for your choice of the "smallest" output vector. Use the LQG method and apply the resulting output feedback controller to the original nonlinear system. Obtain your best design and illustrate its performance in simulation. How would you reconfigure your controller to asymptotically track a constant reference on $x$? Will your design reject constant force disturbances applied on the cart ?

# 2 Equations of Motion

## 2.1 Velocity of each moving body

position of the mass $m1$ and $m2$, in the $x-y$ plane can be given by following equations,

$$P1 = (x - l_1 sin\theta_1 \ , \ -l_1 cos\theta_1)$$

$$P2 = (x - l_2 sin\theta_2 \ , \ -l_1 cos\theta_2)$$

Differentiating them with respect to time gives the velocity of that mass,

$$V_{1x} = \frac{d}{dt}(x - l_1 sin\theta_1) \quad = \quad \dot{x} - l_1 cos\theta_1 \dot{\theta}_1 \tag{2.1}$$

$$V_{1y} = \frac{d}{dt}(l_1 cos\theta_1) \quad = \quad -l_1 sin\theta_1 \dot{\theta}_1 \tag{2.2}$$

$$V_{2x} = \frac{d}{dt}(x - l_2 sin\theta_2) \quad = \quad \dot{x} - l_2 cos\theta_2 \dot{\theta}_2 \tag{2.3}$$

$$V_{2y} = \frac{d}{dt}(l_2 cos\theta_2) \quad = \quad -l_1 sin\theta_2 \dot{\theta}_2 \tag{2.4}$$

Velocity of the cart can be taken as

$$V = \dot{x} \tag{2.5}$$

## 2.2 Kinetic and potential energy of the system

Kinetic energy of the system can be calculated by using velocities derived in earlier section.

$$K = \frac{1}{2}MV^2 + \frac{1}{2}m_1(V_{1x}^2 + V_{1y}^2) + \frac{1}{2}m_2(V_{2x}^2 + V_{2y}^2)$$

$$K = \frac{1}{2}(M + m_1 + m_2)\dot{x}^2 + \frac{1}{2}m_1 l_1^2 \dot{\theta}_1^2 + \frac{1}{2}m_2 l_2^2 \dot{\theta}_2^2 - m_1 l_1 \dot{x}\dot{\theta}_1 cos\theta_1 - m_2 l_2 \dot{x}\dot{\theta}_2 cos\theta_2 \tag{2.6}$$

Since cart is moving on flat surface, it's potential energy will not change. We can find potential energy of hanging masses by considering flat plane as reference. Equations of potential energy can be written as follows,

$$U = -m_1 l_1 cos\theta_1 g - m_2 l_2 cos\theta_2 g \tag{2.7}$$

## 2.3 Lagrangian

Lagrangian for this system can be defined as

$$L = K - U$$

Following three equations will give the equations of motion.

$$\frac{\partial}{t}(\frac{\partial L}{\dot{x}}) - \frac{\partial L}{\partial x} = F$$

$$\frac{\partial}{t}(\frac{\partial L}{\dot{\theta}_1}) - \frac{\partial L}{\partial \theta_1} = 0$$

$$\frac{\partial}{t}(\frac{\partial L}{\dot{\theta}_2}) - \frac{\partial L}{\partial \theta_2} = 0$$

We will get the following equations after partial differentiation,

$$\ddot{x} = \frac{1}{M + m_1 + m_2}[m_1 l_1 \ddot{\theta}_1 cos\theta_1 + m_2 l_2 \ddot{\theta}_2 cos\theta_2 - m_1 l_1 \dot{\theta}_1^2 sin\theta_1 - m_2 l_2 \dot{\theta}_2^2 sin\theta_2 + F] \tag{2.8}$$

$$\ddot{\theta}_1 = \frac{\ddot{x} cos\theta_1}{l_1} - \frac{g sin\theta_1}{l_1} \tag{2.9}$$

$$\ddot{\theta}_2 = \frac{\ddot{x} cos\theta_2}{l_2} - \frac{g sin\theta_2}{l_2} \tag{2.10}$$

Taking $x$ , $\dot{x}$ , $\theta_1$ , $\dot{\theta}_1$ , $\theta_2$ , $\dot{\theta}_2$ as system states, we can write state space form of nonlinear system as follows,

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{-m_1 g sin\theta_1 cos\theta_2 - m_2 g sin\theta_2 cos\theta_2 - m_1 l_1 \dot{\theta}_1^2 sin\theta_1 - m_2 l_2 \dot{\theta}_2^2 sin\theta_2 + F}{M + m_1 + m_2 - m_1 cos^2\theta_1 - m_2 cos^2\theta_2} \\ \dot{\theta}_1 \\ \frac{-m_1 g sin\theta_1 cos\theta_2 - m_2 g sin\theta_2 cos\theta_2 - m_1 l_1 \dot{\theta}_1^2 sin\theta_1 - m_2 l_2 \dot{\theta}_2^2 sin\theta_2 + F}{(M + m_1 + m_2 - m_1 cos^2\theta_1 - m_2 cos^2\theta_2) l_1} - \frac{g sin\theta_1}{l_1} \\ \dot{\theta}_2 \\ \frac{-m_1 g sin\theta_1 cos\theta_2 - m_2 g sin\theta_2 cos\theta_2 - m_1 l_1 \dot{\theta}_1^2 sin\theta_1 - m_2 l_2 \dot{\theta}_2^2 sin\theta_2 + F}{(M + m_1 + m_2 - m_1 cos^2\theta_1 - m_2 cos^2\theta_2) l_2} - \frac{g sin\theta_2}{l_2} \end{bmatrix} \tag{2.11}$$

# 3 Linear State Space Representation of System

In the previous section we have derived the equation of motion of the cart system with two pendulum and it was represented in the state space form. Because of sine and cosine components, the equations are non linear and it becomes difficult to solve non linear equations.

## 3.1 Linearization of system

We will linearize the system around equilibrium point $x = 0$, $\theta_1 = 0$ and $\theta_2 = 0$, which is mentioned in the problem statement.

In the Eq 2.8, 2.9 and 2.10, for the limiting condition at equilibrium, we can replace,
$sin\theta_1 \approx \theta_1$
$sin\theta_2 \approx \theta_2$
$cos\theta_1 \approx 1$
$cos\theta_2 \approx 1$
$\dot{\theta}_1^2 = \dot{\theta}_2^2 \approx 0$

After taking this approximations, we will get the following equations,

$$\begin{aligned}
\ddot{x} &= \frac{1}{M}(-m_1 g\theta_1 - m_2 g\theta_2 + F) \\
\ddot{\theta}_1 &= \frac{1}{Ml_1}(-m_1 g\theta_1 - m_2 g\theta_2 - Mg\theta_1 + F) \\
\ddot{\theta}_2 &= \frac{1}{Ml_2}(-m_1 g\theta_1 - m_2 g\theta_2 - Mg\theta_2 + F)
\end{aligned} \tag{3.1}$$

## 3.2 Linear State Space Representation

System of equations represented in Eq 3.1 can be represented in state space form. Taking $x$, $\dot{x}$, $\theta_1$, $\dot{\theta}_1$, $\theta_2$ and $\dot{\theta}_2$ as state variables, state space representation can be written as shown below,

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{gm_1}{M} & 0 & -\frac{gm_2}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{g(-M-m_1)}{Ml_1} & 0 & -\frac{gm_2}{Ml_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{gm_1}{Ml_2} & 0 & \frac{g(-M-m_2)}{Ml_2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml_1} \\ 0 \\ \frac{1}{Ml_2} \end{bmatrix} u \tag{3.2}$$

$$y = CX + DU \tag{3.3}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad D = 0$$

Which is similar to $\dot{X} = AX + Bu$. Input force on cart $F$ is represented as $u$.

## 3.3 Controllability

We can check the controllability of the derived system by checking the rank of controllbility matrix. Controllability matrix can be defined as below,

$$R = \begin{bmatrix} B & AB & A^2B & A^3B & A^4B & A^5B \end{bmatrix}$$

Determinant of this matrix should not be equal to zero for the system to be controllable,

$$Det(R) = -\frac{g^6 l_1^2 - 2g^6 l_1 l_2 + g^6 l_2^2}{M^6 l_1^6 l_2^6} = -\frac{g^6 (l_1 - l_2)^2}{M^6 l_1^6 l_2^6}$$

Which gives us the condition that $l_1 \neq 0, l_2 \neq 0$ and $l_1 \neq l_2$.

## 4 Part-D LQR

Putting $M = 1000Kg$, $m_1 = m_2 = 100Kg$, $l_1 = 20m$ and $l_2 = 10m$ in Eq 3.2 , we get

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.98 & 0 & -0.98 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -0.539 & 0 & -0.049 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -0.098 & 0 & -1.078 & 0 \end{bmatrix} \tag{4.1}$$

$$B = \begin{bmatrix} 0 \\ 0.001 \\ 0 \\ 0.00005 \\ 0 \\ 0.0001 \end{bmatrix} \tag{4.2}$$

Controllability condition needs to be checked to verify if the system is controllable or not. Using the python ode given below, we can check the controllability condition.

```
##########################################################################
import numpy as np
import control

m1 = 100
m2 = 100
M = 1000
l1 = 20
l2 = 10
g = 9.81

A = np.array([[0, 1, 0, 0, 0, 0],
        [0, 0, -m1*g/M, 0, -m2*g/M, 0],
        [0, 0, 0, 1, 0, 0],
        [0, 0, -(M+m1)*g/(M*l1), 0, -m2*g/(M*l1), 0],
        [0, 0, 0, 0, 0, 1],
        [0, 0, -m1*g/(M*l2), 0, -(M+m2)*g/(M*l2),0]])
B = np.array([[0], [1/M], [0], [1/(M*l1)], [0], [1/(M*l2)]])
C = np.eye(6)
D = np.zeros((6,1))

print( np.linalg.matrix_rank(control.ctrb(A,B)) )
##########################################################################
```

Code above, when run, gives the rank of matrix 6, which shows that matrix is of full rank and thus the system is controllable.

Different values of Q and R shall be taken to control the system in optimum way and using the graphs we can decide which value of Q and R gives good results and stabilizes the system as soon as possible. Q can be decided on how much approximate time the system should take to stabilize the particular state and what is the error band within which it should work.

$$Q_i = \frac{1}{(time) * (error)^2}$$

After several experiments we have finalized this values of Q and R,

$$Q = \begin{bmatrix} \frac{1}{12*1^2} & 0 & 10 & 0 & 0 & 0 \\ 0 & \frac{1}{12*0.1^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{8*0.02^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{12*0.01^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{8*0.02^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{12*0.01^2} \end{bmatrix} \tag{4.3}$$

$R = 0.00001$

The code below will simulate both linear and non linear models and will plot the results which are shown. "stateValues.py", "cartPlotter.py" and "responsePlot.py" files are given in index.

```
########################################################
import numpy as np
from cartPlotter import cartPlotter
import control
from scipy.integrate import odeint
from stateValues import A,B,C,D,Q
from responsePlot import responsePlot
#########################################################
run_animation = 0 ### 0 if no animation is required
m1, m2, M, l1 , l2 , g = 100, 100, 1000, 20, 10, 9.81

Q = np.array([[1/(12*1*1), 0, 10, 0, 0, 0],
        [0, 1/(12*0.1*0.1), 0, 0, 0, 0],
        [0, 0, 1/(8*0.02*0.02), 0, 0, 0],
        [0, 0, 0, 1/(12*0.01*0.01), 0, 0],
        [0, 0, 0, 0, 1/(8*0.02*0.02), 0],
        [0, 0, 0, 0, 0, 1/(12*0.01*0.01)]])
```

```
R = 0.00001

############## LQR gain ##################################
K = control.lqr(A,B,Q,R)

############## ODE solver for linear model #############
def pend_linear(y, t, A, B, K_):
        dydt = np.matmul((A-np.matmul(B,K_)),(y.reshape((-1,1))))
        return dydt.reshape(-1)

######################### ODE solver for non linear model ###############
def pend_non_linear(y, t, A, B, K_):
        u = np.matmul(K_,y.reshape((-1,1)))[0,0]
        d1 = y[1]
        DD = M + m1 + m2 - m1*(np.cos(y[2])**2) - m2*(np.cos(y[4])**2)
        d2 = -(1/DD)*(m1*g*np.sin(2*y[2])/2 + m2*g*np.sin(2*y[4])/2 +
        m1*l1*y[3]*y[3]*np.sin(y[2]) + m2*l2*y[5]*y[5]*np.sin(y[4]) + u)
        d3 = y[3]
        d4 = (1/l1)*(d2*np.cos(y[2]) - g*np.sin(y[2]))
        d5 = y[5]
        d6 = (1/l2)*(d2*np.cos(y[4]) - g*np.sin(y[4]))
        return np.array([d1,d2,d3,d4,d5,d6])

############################## Initial conditions ############################
y0 = np.array([[0],[0],[-0.2],[0],[0.3],[0]])
t = np.arange(0, 100, 0.001)

############ Solves linear model with initial condition ####################
sol_linear = odeint(pend_linear, y0.reshape(-1), t, args=(A, B, K[0]))

############ Solves non linear model with initial condition ##################
sol_non_linear = odeint(pend_non_linear, y0.reshape(-1), t,
                        args=(A, B, K[0]))

######################### shows the Animation ################################
if run_animation:
        cartAnimatter = cartPlotter()
        cartAnimatter.run_animation(sol_linear, t, "Linear␣model")
        cartAnimatter.run_animation(sol_non_linear,t, "Non␣linear␣model")

response = responsePlot()
response.plotResponse(sol_linear, t, K, "Response␣of␣linear␣system")
response.plotResponse(sol_linear, t, K, "Response␣of␣non␣linear␣system")
##############################################################################
```

## 4.1 Lyapunov's Indirect Stability

When we see the eigen values of $(A-BK)$, it should be in left half plane for the system to be stable. Eigen values of $(A-BK)$ are as follows,

$$\begin{bmatrix} -0.82946549+0.j \\ -0.11715611+0.j \\ -0.38393634+0.81505088j \\ -0.38393634-0.81505088j \\ -0.14447721+0.73211482j \\ -0.14447721-0.73211482j \end{bmatrix}$$

Real parts of all the eigen values are negative, and thus with the help of Lyapunov's indirect stability criterion, we can say that the system is stable.

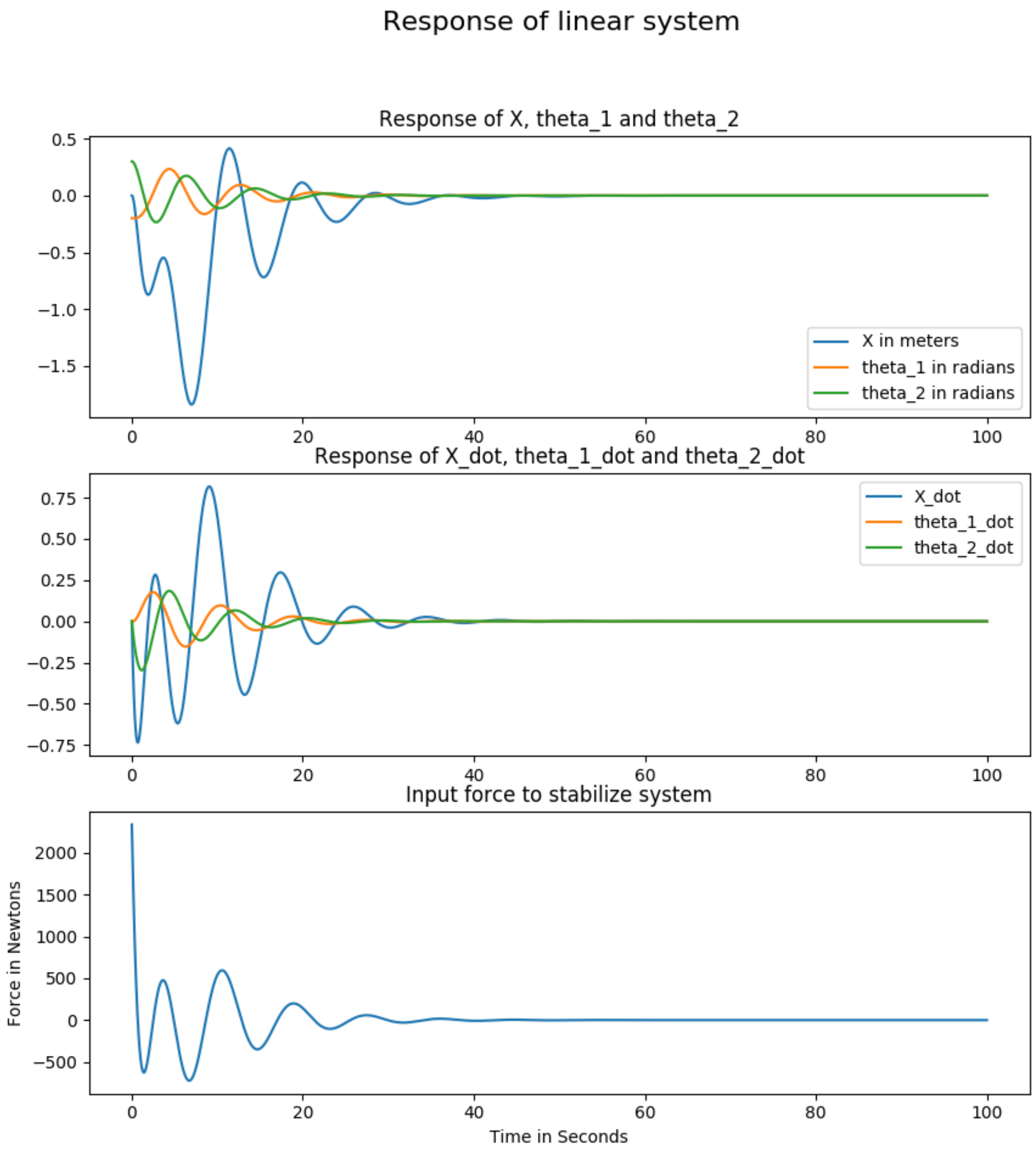


Figure 4.1: Response with linear model

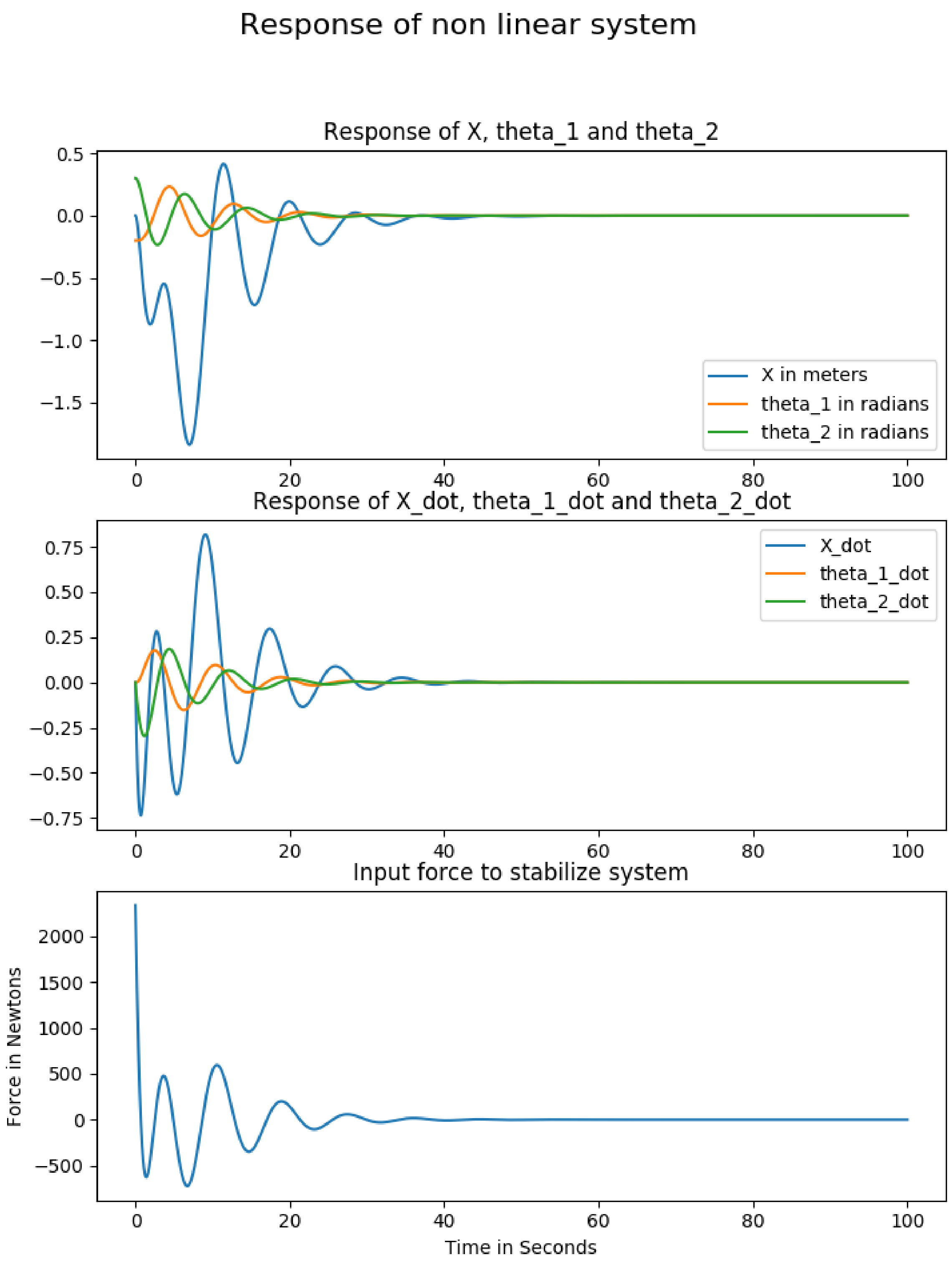


Figure 4.2: Response with non linear model

# 5 Part-E Observability

We have four sets of output values that needs to be checked for the observability condition.

1. $x(t)$
2. $\theta_1(t)$ , $\theta_2(t)$
3. $x(t)$ , $\theta_2(t)$
4. $x(t)$ , $\theta_1(t)$, $\theta_2(t)$

We will check the rank of following matrix and it should be 6, for the system to be observable with those states.

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \\ CA^4 \\ CA^5 \end{bmatrix} \qquad and \qquad rank(O) = 6$$

## 5.1 Observability for $x(t)$

We will take,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

When we check the rank condition with python's numpy module, we get the rank of $O$ is 6, which means that we can observer the full system with the $x(t)$ as our system output.

## 5.2 Observability for $\theta_1(t)$ and $\theta_2(t)$

We will take,

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

When we check the rank condition with python's numpy module, we get the rank of $O$ is 4, which means that we can not observer the full system with the $\theta_1(t)$ and $\theta_2(t)$ as our system output.

## 5.3 Observability for $x(t)$ and $\theta_2(t)$

We will take,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

When we check the rank condition with python's numpy module, we get the rank of $O$ is 6, which means that we can observer the full system with the $x(t)$ and $\theta_2(t)$ as our system output.

## 5.4 Observability for $x(t)$ , $\theta_1(t)$ and $\theta_2(t)$

We will take,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

When we check the rank condition with python's numpy module, we get the rank of $O$ is 6, which means that we can observer the full system with the $x(t)$, $\theta_1(t)$ and $\theta_2(t)$ as our system output.

## 6 Part-F Luenberger Observer

We have linear system which is represented in state space form,

$$\dot{X}(t) = AX(t) + Bu(t)$$

$$y(t) = CX(t) + Du(t)$$

For this system we can create a Luenberger observer for the input vectors with which system is observable and the observer system can be presented as below,

$$\dot{\hat{X}}(t) = A\hat{X}(t) + Bu(t) + L(y(t) - \hat{y}(t))$$

$$\hat{y}(t) = C\hat{X}(t) + Du(t)$$

Where $L$ is observer gain and $\hat{X}(t)$ is our estimated state from output $y(t)$. We can find $L$ by placing our poles for $A - LC$ in negative half plane. For better convergence we have put it at 3 times far from eigen values of $A - BK$, where $K$ is our feedback gain.

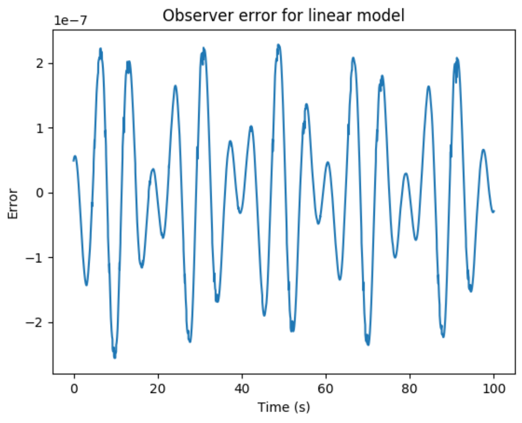
Following code will simulate the non linear and linear system given with the initial condition and a step input at time $t = 20s$.

```
######################### Start #######################
import numpy as np
from cartPlotter import cartPlotter
import control
from scipy.integrate import odeint
from stateValues import A,B,C,D,Q
from responsePlot import responsePlot
import matplotlib.pyplot as plt

########## For output state x(t) ##################
C = np.array([[1,0,0,0,0,0]])

####### For outout states x(t) and theta_2(t) ####
# C = np.array([[1,0,0,0,0,0],
#                        [0,0,0,0,1,0]])

## For outout states x(t), theta_1(t) and theta_2(t) ####
# C = np.array([[1,0,0,0,0,0],
#                        [0,0,1,0,0,0],
#                        [0,0,0,0,1,0]])

run_animation = 0 ### 0 if no animation is required

m1, m2, M, l1 , l2 , g = 100, 100, 1000, 20, 10, 9.81
```

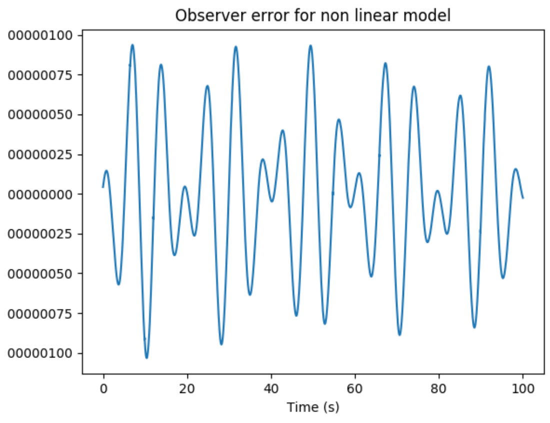
```
Q = np.array([[1/(12*1*1), 0, 10, 0, 0, 0],
        [0, 1/(12*0.1*0.1), 0, 0, 0, 0],
        [0, 0, 1/(8*0.02*0.02), 0, 0, 0],
        [0, 0, 0, 1/(12*0.01*0.01), 0, 0],
        [0, 0, 0, 0, 1/(8*0.02*0.02), 0],
        [0, 0, 0, 0, 0, 1/(12*0.01*0.01)]])
R = 0.00001

############## LQR gain ################################
K = control.lqr(A,B,Q,R)

############## ODE solver for linear model #############
def pend_linear(y, t, A, B, K_):
        dydt = np.matmul(A,(y.reshape((-1,1))))
        return dydt.reshape(-1)

############### ODE solver for non linear model ##########
def pend_non_linear(y, t, A, B, K_):
        d1 = y[1]
        DD = M + m1 + m2 - m1*(np.cos(y[2])**2) - m2*(np.cos(y[4])**2)
        d2 = -(1/DD)*(m1*g*np.sin(2*y[2])/2 + m2*g*np.sin(2*y[4])/2
        + m1*l1*y[3]*y[3]*np.sin(y[2]) + m2*l2*y[5]*y[5]*np.sin(y[4]))
        d3 = y[3]
        d4 = (1/l1)*(d2*np.cos(y[2]) - g*np.sin(y[2]))
        d5 = y[5]
        d6 = (1/l2)*(d2*np.cos(y[4]) - g*np.sin(y[4]))
        return np.array([d1,d2,d3,d4,d5,d6])

############## ODE solver for observer #################
def obser(y, t, L_, y_):
        u = 0
        dxdt = np.matmul((A - np.matmul(L_,C)),y.reshape(-1,1)) + B*u
        + np.matmul(L_,y_.reshape((-1,1)))
        return dxdt.reshape(-1)

##############  Initial conditions ###################
x0 = np.array([[0],[0],[-0.2],[0],[0.3],[0]])
t = np.arange(0, 100, 0.001)

############# Placing the poles of L at 3 times of poles of K #######
L = control.place(np.matrix.transpose(A), np.matrix.transpose(C), 3*K[2])
L_ = np.matrix.transpose(L)
x_hat0 = x0.reshape(6)
u = -np.matmul(K[0],x_hat0)
```

```
observed = []
error = []
trueStates = []
observed.append(x_hat0)

########### Loop to solve two differential equations ###########
########### One for plant and one for observer ###############
for i in range(len(t) - 1):
        if t[i] == 20:  ### Step input at t = 20 s
                x_hat0[0] = 1
        else:
                pass
        ############### Plant linear or non linear ######
        X = odeint(pend_linear, x_hat0, t[i:i+2], args=(A, B, K[0]))
        trueStates.append(X[1])
        y = np.matmul(C,(X[1]).reshape(-1,1)).reshape(-1)
        ############# Observer ###############
        x_hat_ = odeint(obser , x_hat0 , t[i:i+2], args=(L_,y))
        x_hat0 = x_hat_[1]  ## Initial condition for next step
        observed.append(x_hat0)
        error.append(X[1] - x_hat_[1])

trueStates = np.array(trueStates)
error = np.array(error)
states = np.array(observed)

if run_animation:
        cartAnimatter = cartPlotter()
        cartAnimatter.run_animation(states, t[:states.shape[0]], "Linear␣model")

plt.plot(t[:error.shape[0]],error[:,0])
plt.title("Observer␣error␣for␣linear␣model")
plt.xlabel("Time␣(s)")
plt.ylabel("Error")
plt.show()
############################## End #####################################
```

From plots, you can see that error has same kind of trend but they are very less and their value changes with the selection of input vectors.

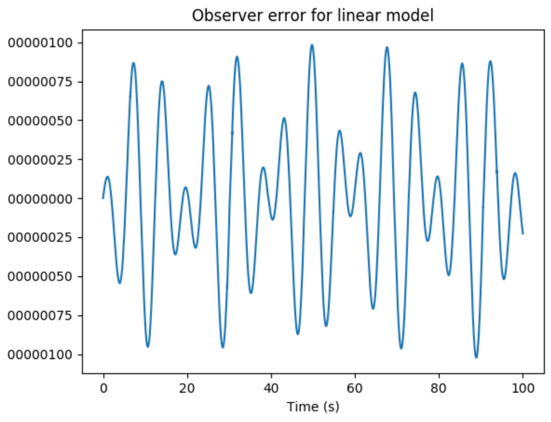


(a) Observer error with linear model

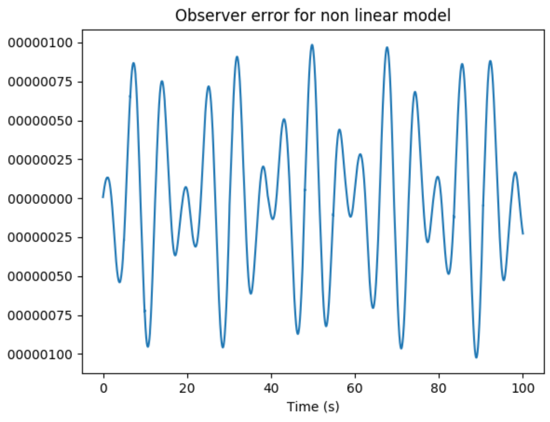


(b) Observer error with nonlinear model

Figure 6.1: Observer error with $x(t)$ output vector

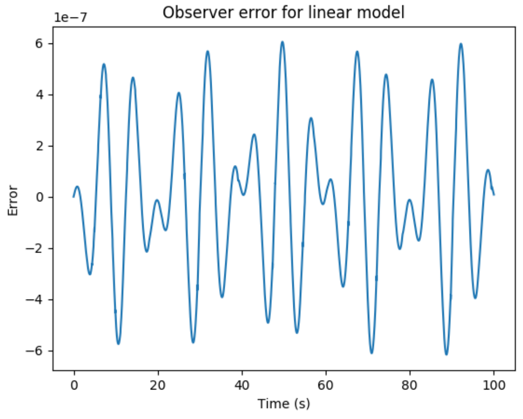


(a) Observer error with linear model

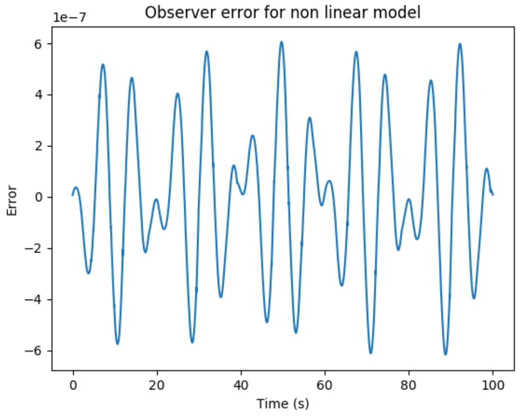


(b) Observer error with nonlinear model

Figure 6.2: Observer error with $x(t)$ and $\theta_2(t)$ output vector



(a) Observer error with linear model



(b) Observer error with nonlinear model

Figure 6.3: Observer error with $x(t)$, $\theta_1(t)$ and $\theta_2(t)$ output vector

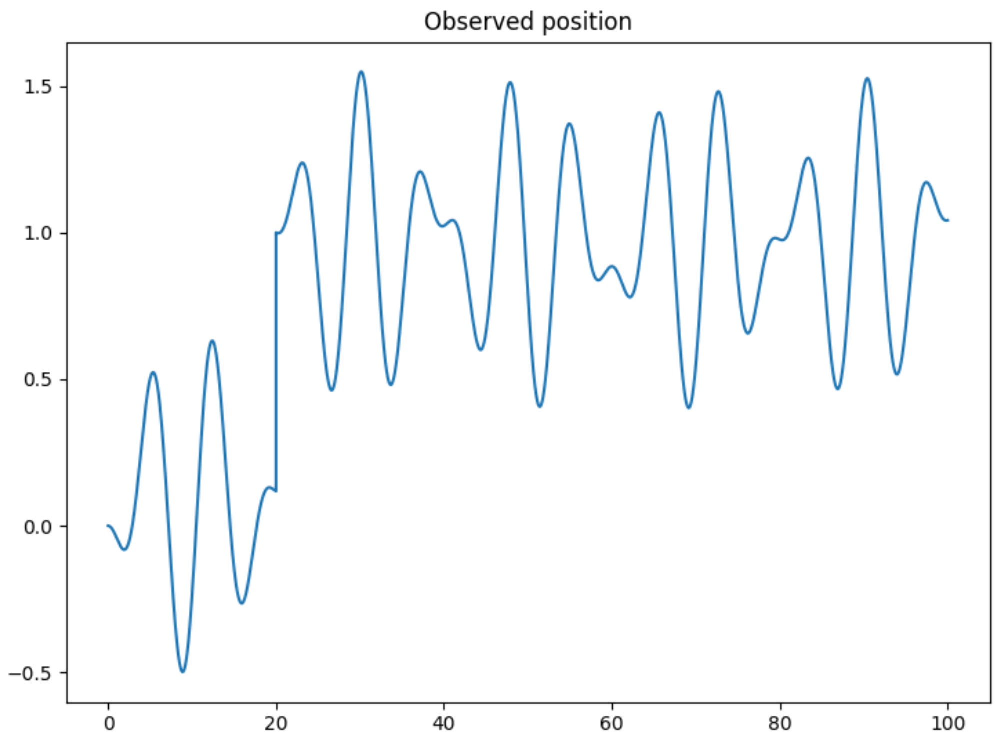


Figure 6.4: Initial condition $X(0)$ with step input at $t = 20s$

## 7 Part-G LQG Control

$x(t)$ is the minimum vector with which we can observe the entire system. So we will use $X(t)$ as system output and we will place poles of observer at the same place as $K$. We have added random Gaussian noise to system and measurement data, which are called system disturbances and measurement noise. Noise with mean 0 and standard deviation of 0.16 has been added. Code to simulate this control with non linear model has been provided below.

```
##############################   Start  ################################
import numpy as np
from cartPlotter import cartPlotter
import control
from scipy.integrate import odeint
from stateValues import A,B,C,D,Q
from responsePlot import responsePlot
import matplotlib.pyplot as plt

run_animation = 1 ### 0 if no animation is required
mean = 0  ### Mean od noise
std_dev = 0.16  ### Stdandard devication of noise
m1, m2, M, l1, l2, g = 100, 100, 1000, 20, 10, 9.81

C = np.array([[1,0,0,0,0,0]])  ### measure x(t)

Q = np.array([[1/(12*1*1), 0, 10, 0, 0, 0],
        [0, 1/(12*0.1*0.1), 0, 0, 0, 0],
        [0, 0, 1/(8*0.02*0.02), 0, 0, 0],
        [0, 0, 0, 1/(12*0.01*0.01), 0, 0],
        [0, 0, 0, 0, 1/(8*0.02*0.02), 0],
        [0, 0, 0, 0, 0, 1/(12*0.01*0.01)]])
R = 0.00001

############## LQR gain ###############################
K = control.lqr(A,B,Q,R)

############## ODE solver for non linear model ##########
def pend_non_linear(y, t, A, B, K_):
        u = np.matmul(K_,y.reshape((-1,1)))[0,0]
        d1 = y[1] + np.random.normal(mean, std_dev)
        DD = M + m1 + m2 - m1*(np.cos(y[2])**2) - m2*(np.cos(y[4])**2)
        d2 = -(1/DD)*(m1*g*np.sin(2*y[2])/2 + m2*g*np.sin(2*y[4])/2
                + m1*l1*y[3]*y[3]*np.sin(y[2]) + m2*l2*y[5]*y[5]*np.sin(y[4]) + u)
```

```
            + np.random.normal(mean, std_dev)
        d3 = y[3] + np.random.normal(mean, std_dev)
        d4 = (1/l1)*(d2*np.cos(y[2]) - g*np.sin(y[2]))
            + np.random.normal(mean, std_dev)
        d5 = y[5] + np.random.normal(mean, std_dev)
        d6 = (1/l2)*(d2*np.cos(y[4]) - g*np.sin(y[4]))
            + np.random.normal(mean, std_dev)
        return np.array([d1,d2,d3,d4,d5,d6])

############ ODE solver for Observer ########
def obser(x_hat, t, L_, y_):
        u = -np.matmul(K[0],x_hat.reshape(-1,1))
        dxdt = np.matmul((A - np.matmul(L_,C)),x_hat.reshape(-1,1))
            + B*u + np.matmul(L_,y_.reshape((-1,1)))
        return dxdt.reshape(-1)

############# Solving non linear equation ############################
x0 = np.array([[0],[0],[-0.2],[0],[0.3],[0]])
t = np.arange(0, 100, 0.1)
####################################################################
L = control.place(np.matrix.transpose(A), np.matrix.transpose(C), K[2])
L_ = np.matrix.transpose(L)
x_hat0 = x0.reshape(6)
u = -np.matmul(K[0],x_hat0)
observed = []
observed.append(x_hat0)

for i in range(len(t) - 1):
        print("solving␣"+str(i))
        X = odeint(pend_non_linear, x_hat0, t[i:i+2], args=(A, B, K[0]))
        y = np.matmul(C,(X[1]).reshape(-1,1)).reshape(-1)
            + np.random.normal(mean, std_dev)
        x_hat = odeint(obser , x_hat0 , t[i:i+2], args=(L_,y))
        x_hat0 = x_hat[1]
        u = -np.matmul(K[0],x_hat0)
        observed.append(x_hat0)

sol = np.array(observed)

if run_animation:
        cartAnimatter = cartPlotter()
        cartAnimatter.run_animation(sol, t[:sol.shape[0]], "Observer")

response = responsePlot()
```

```
response.plotResponse(sol, t, K, "Response␣of␣non␣linear␣system␣at␣noisy␣data")

############################ END ############################################
```

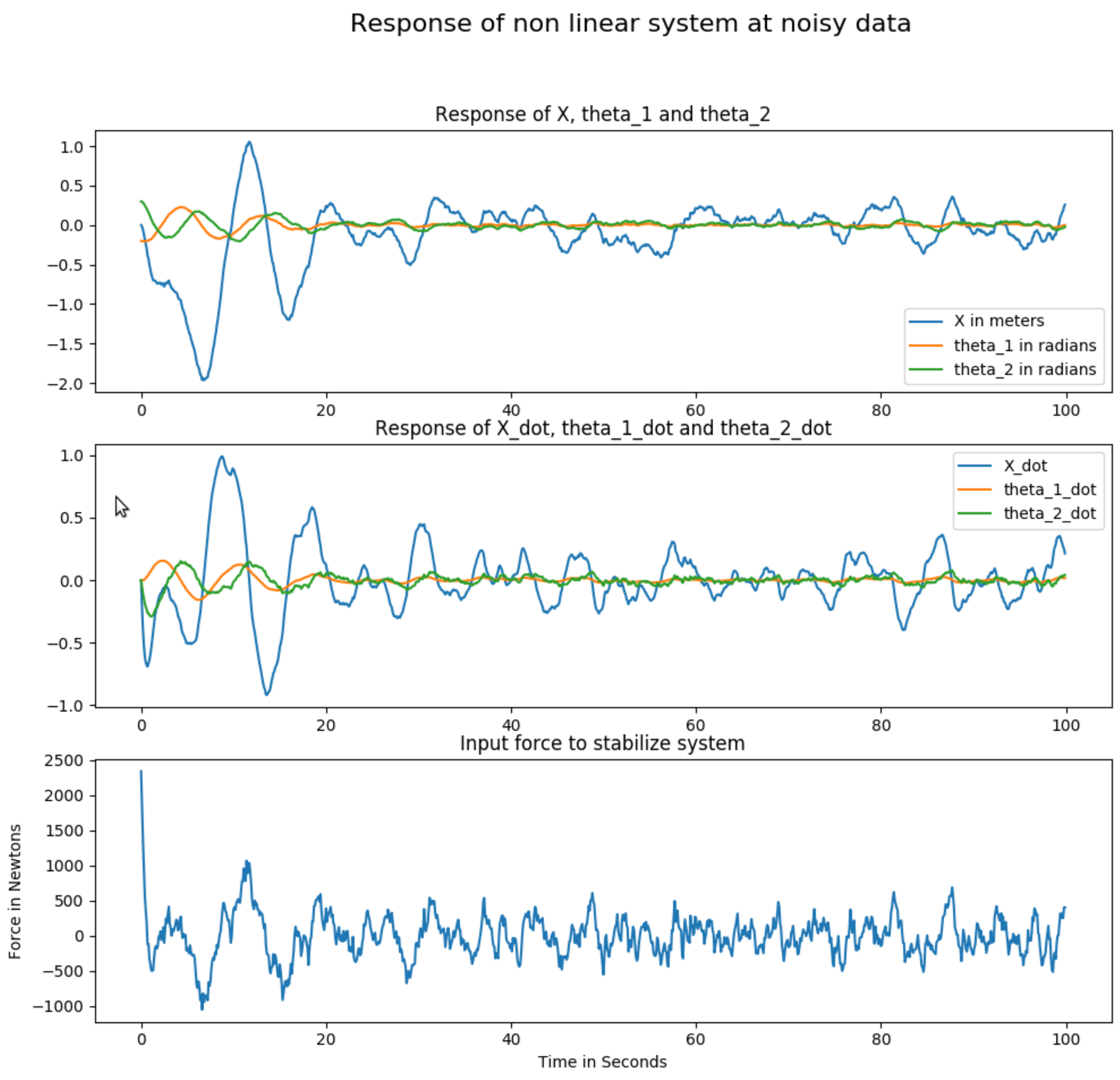


Figure 7.1: Response with LQG controller

From Figure 7.1, you can see that even with the constant disturbances it is trying to stabilize the system at the equilibrium point.

# 8 Appendix

## 8.1 stateValues.py

```
import numpy as np

m1, m2, M, l1, l2, g = 100, 100, 1000, 20, 10, 9.81

A = np.array([[0, 1, 0, 0, 0, 0],
        [0, 0, -m1*g/M, 0, -m2*g/M, 0],
        [0, 0, 0, 1, 0, 0],
        [0, 0, -(M+m1)*g/(M*l1), 0, -m2*g/(M*l1), 0],
        [0, 0, 0, 0, 0, 1],
        [0, 0, -m1*g/(M*l2), 0, -(M+m2)*g/(M*l2),0]])
B = np.array([[0], [1/M], [0], [1/(M*l1)], [0], [1/(M*l2)]])
C = np.eye(6)
D = np.zeros((6,1))

Q = np.array([[1/(12*1*1), 0, 10, 0, 0, 0],
        [0, 1/(12*0.1*0.1), 0, 0, 0, 0],
        [0, 0, 1/(8*0.02*0.02), 0, 0, 0],
        [0, 0, 0, 1/(12*0.01*0.01), 0, 0],
        [0, 0, 0, 0, 1/(8*0.02*0.02), 0],
        [0, 0, 0, 0, 0, 1/(12*0.01*0.01)]])
```

## 8.2 responsePlot.py

```
import matplotlib.pyplot as plt
import numpy as np


class responsePlot:
        def __init__(self):
                pass

        def plotResponse(self, sol, t, K, title):
                fig2 = plt.figure()
                fig2.suptitle(title, fontsize=16)
                ax_1 = fig2.add_subplot(311)
                ax_2 = fig2.add_subplot(312)
                ax_u = fig2.add_subplot(313)
                ax_u.set_xlabel("Time␣in␣Seconds")
                ax_u.set_ylabel("Force␣in␣Newtons")
                ax_u.set_title("Input␣force␣to␣stabilize␣system")
                ax_1.set_title("Response␣of␣X,␣theta_1␣and␣theta_2")
                ax_2.set_title("Response␣of␣X_dot,␣theta_1_dot␣and␣theta_2_dot")
                ax_1.plot(t,sol[:,0].reshape(−1), label="X␣in␣meters")
                ax_1.plot(t,sol[:,2].reshape(−1), label="theta_1␣in␣radians")
                ax_1.plot(t,sol[:,4].reshape(−1), label="theta_2␣in␣radians")
                ax_2.plot(t,sol[:,1].reshape(−1), label="X_dot")
                ax_2.plot(t,sol[:,3].reshape(−1), label="theta_1_dot")
                ax_2.plot(t,sol[:,5].reshape(−1), label="theta_2_dot")
                ax_1.legend()
                ax_2.legend()
                U_ = np.matmul(K[0],np.matrix.transpose(sol))
                ax_u.plot(t, U_.reshape(−1))
                plt.show()
```

## 8.3 cartPlotter.py

```
import matplotlib.pyplot as plt
import numpy as np

class cartPlotter:
        def __init__(self):
                pass

        def plot_cart(self,x_, mass, ax):
                den = 100
                l = 1
                ax.plot([x_ + l/2, x_ + l/2, x_ - l/2, x_ - l/2, x_ + l/2],
                        [0, l, l, 0, 0] )
                pass

        def ball(self,theta, l, x_, mass, ax):
                l = l/10
                ax.scatter(x_ - l*np.sin(theta), -l*np.cos(theta), s=mass*2)
                ax.plot([x_, x_ - l*np.sin(theta)], [0, -l*np.cos(theta)])
                pass

        def write_vals(self,sol, i, t, ax):
                ax.text(0.8,0.95,'x␣:␣'+str(round(sol[i,0],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.9,'dx/dt␣:␣'+str(round(sol[i,1],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.85,'t1␣:␣'+str(round(sol[i,2],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.8,'d(t1)/dt␣:␣'+str(round(sol[i,3],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.75,'t2␣:␣'+str(round(sol[i,4],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.7,'d(t2)/dt␣:␣'+str(round(sol[i,5],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.65,'time␣:␣'+str(round(t[i],2)),
                        horizontalalignment='left',verticalalignment='center',
                transform=ax.transAxes)

    def run_animation(self, sol, t, title):
        m1, m2, M, l1, l2, g = 100, 100, 1000, 20, 10, 9.81
        fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.ion()
        for i in range(0,sol.shape[0],100):
            if i%10==0:
                print("␣Number␣:␣"+str(i))
                pass
            self.plot_cart(sol[i,0], M, ax)
            self.ball(sol[i,2], l1, sol[i,0], m1, ax)
            self.ball(sol[i,4], l2, sol[i,0], m2, ax)
            self.write_vals(sol, i, t, ax)
            ax.plot([-50,50],[0,0],c="black")
            plt.xlim(-5, 5)
            plt.ylim(-10, 10)
            plt.title(title)
            plt.pause(0.0001)
            plt.cla()

        plt.ioff()
        self.plot_cart(sol[i,0], M, ax)
        self.ball(sol[i,2], l1, sol[i,0], m1, ax)
        self.ball(sol[i,4], l2, sol[i,0], m2, ax)
        self.write_vals(sol, i, t, ax)
        ax.plot([-5,5],[0,0],c="black")
        plt.xlim(-5, 5)
        plt.ylim(-10, 10)
        plt.title(title)
        plt.show()
```

## 8.4 lqr.py

```
import matplotlib.pyplot as plt
import numpy as np

class cartPlotter:
        def __init__(self):
                pass

        def plot_cart(self,x_, mass, ax):
                den = 100
                l = 1
                ax.plot([x_ + l/2, x_ + l/2, x_ - l/2, x_ - l/2, x_ + l/2],
                        [0, l, l, 0, 0] )
                pass

        def ball(self,theta, l, x_, mass, ax):
                l = l/10
                ax.scatter(x_ - l*np.sin(theta), -l*np.cos(theta), s=mass*2)
                ax.plot([x_, x_ - l*np.sin(theta)], [0, -l*np.cos(theta)])
                pass

        def write_vals(self,sol, i, t, ax):
                ax.text(0.8,0.95,'x␣:␣'+str(round(sol[i,0],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.9,'dx/dt␣:␣'+str(round(sol[i,1],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.85,'t1␣:␣'+str(round(sol[i,2],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.8,'d(t1)/dt␣:␣'+str(round(sol[i,3],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.75,'t2␣:␣'+str(round(sol[i,4],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.7,'d(t2)/dt␣:␣'+str(round(sol[i,5],2)),
                        horizontalalignment='left',verticalalignment='center',
                        transform=ax.transAxes)
                ax.text(0.8,0.65,'time␣:␣'+str(round(t[i],2)),
                        horizontalalignment='left',verticalalignment='center',
                transform=ax.transAxes)

    def run_animation(self, sol, t, title):
        m1, m2, M, l1, l2, g = 100, 100, 1000, 20, 10, 9.81
        fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.ion()
        for i in range(0,sol.shape[0],100):
            if i%10==0:
                print("␣Number␣:␣"+str(i))
                pass
            self.plot_cart(sol[i,0], M, ax)
            self.ball(sol[i,2], l1, sol[i,0], m1, ax)
            self.ball(sol[i,4], l2, sol[i,0], m2, ax)
            self.write_vals(sol, i, t, ax)
            ax.plot([-50,50],[0,0],c="black")
            plt.xlim(-5, 5)
            plt.ylim(-10, 10)
            plt.title(title)
            plt.pause(0.0001)
            plt.cla()

        plt.ioff()
        self.plot_cart(sol[i,0], M, ax)
        self.ball(sol[i,2], l1, sol[i,0], m1, ax)
        self.ball(sol[i,4], l2, sol[i,0], m2, ax)
        self.write_vals(sol, i, t, ax)
        ax.plot([-5,5],[0,0],c="black")
        plt.xlim(-5, 5)
        plt.ylim(-10, 10)
        plt.title(title)
        plt.show()
```

## 8.5 observer.py

```
import numpy as np
from cartPlotter import cartPlotter
import control
from scipy.integrate import odeint
from stateValues import A,B,C,D,Q
from responsePlot import responsePlot
import matplotlib.pyplot as plt

########## For output state x(t) #################
C = np.array([[1,0,0,0,0,0]])

####### For outout states x(t) and theta_2(t) ####
# C = np.array([[1,0,0,0,0,0],
#                        [0,0,0,0,1,0]])

## For outout states x(t), theta_1(t) and theta_2(t) ####
# C = np.array([[1,0,0,0,0,0],
#                        [0,0,1,0,0,0],
#                        [0,0,0,0,1,0]])

run_animation = 0 ### 0 if no animation is required

m1, m2, M, l1, l2, g = 100, 100, 1000, 20, 10, 9.81

Q = np.array([[1/(12*1*1), 0, 10, 0, 0, 0],
        [0, 1/(12*0.1*0.1), 0, 0, 0, 0],
        [0, 0, 1/(8*0.02*0.02), 0, 0, 0],
        [0, 0, 0, 1/(12*0.01*0.01), 0, 0],
        [0, 0, 0, 0, 1/(8*0.02*0.02), 0],
        [0, 0, 0, 0, 0, 1/(12*0.01*0.01)]])
R = 0.00001

############## LQR gain ################################
K = control.lqr(A,B,Q,R)

############## ODE solver for linear model #############
def pend_linear(y, t, A, B, K_):
        dydt = np.matmul(A,(y.reshape((-1,1))))
        return dydt.reshape(-1)

############### ODE solver for non linear model ##########
```

```
def pend_non_linear(y, t, A, B, K_):
        d1 = y[1]
        DD = M + m1 + m2 - m1*(np.cos(y[2])**2) - m2*(np.cos(y[4])**2)
        d2 = -(1/DD)*(m1*g*np.sin(2*y[2])/2 +
                m2*g*np.sin(2*y[4])/2 + m1*l1*y[3]*y[3]*np.sin(y[2]) +
                m2*l2*y[5]*y[5]*np.sin(y[4]))
        d3 = y[3]
        d4 = (1/l1)*(d2*np.cos(y[2]) - g*np.sin(y[2]))
        d5 = y[5]
        d6 = (1/l2)*(d2*np.cos(y[4]) - g*np.sin(y[4]))
        return np.array([d1,d2,d3,d4,d5,d6])

############## ODE solver for observer #################
def obser(y, t, L_, y_):
        u = 0
        dxdt = np.matmul((A - np.matmul(L_,C)),y.reshape(-1,1)) +
        B*u + np.matmul(L_,y_.reshape((-1,1)))
        return dxdt.reshape(-1)

##############  Initial conditions ###################
x0 = np.array([[0],[0],[-0.2],[0],[0.3],[0]])
t = np.arange(0, 100, 0.001)

############# Placing the poles of L at 3 times of poles of K #######
L = control.place(np.matrix.transpose(A), np.matrix.transpose(C),
        3*K[2])
L_ = np.matrix.transpose(L)
x_hat0 = x0.reshape(6)
u = -np.matmul(K[0],x_hat0)
observed = []
error = []
trueStates = []
observed.append(x_hat0)

########### Loop to solve two differential equations ###########
########### One for plant and one for observer ###############
for i in range(len(t) - 1):
        if t[i] == 20:  ### Step input at t = 20 s
                x_hat0[0] = 1
        else:
                pass
        ############### Plant linear or non linear ######
        X = odeint(pend_linear, x_hat0, t[i:i+2], args=(A, B, K[0]))
        trueStates.append(X[1])
        y = np.matmul(C,(X[1]).reshape(-1,1)).reshape(-1)
        ############# Observer ###############
        x_hat_ = odeint(obser , x_hat0 , t[i:i+2], args=(L_,y))
        x_hat0 = x_hat_[1]  ## Initial condition for next step
        observed.append(x_hat0)
        error.append(X[1] - x_hat_[1])

trueStates = np.array(trueStates)
error = np.array(error)
states = np.array(observed)

if run_animation:
        cartAnimatter = cartPlotter()
        cartAnimatter.run_animation(states, t[:states.shape[0]],
                "Linear␣model")

plt.plot(t[:error.shape[0]],error[:,0])
plt.title("Observer␣error␣for␣linear␣model")
plt.xlabel("Time␣(s)")
plt.ylabel("Error")
plt.show()
```

## 8.6 lqg.py

```
################################  Start  ##################################
import numpy as np
from cartPlotter import cartPlotter
import control
from scipy.integrate import odeint
from stateValues import A,B,C,D,Q
from responsePlot import responsePlot
import matplotlib.pyplot as plt

run_animation = 1 ### 0 if no animation is required
mean = 0  ### Mean od noise
std_dev = 0.16  ### Stdandard devication of noise
m1, m2, M, l1 , l2 , g = 100, 100, 1000, 20, 10, 9.81

C = np.array([[1 ,0 ,0 ,0 ,0 ,0]])  ### measure x(t)

Q = np.array([[1/(12*1*1), 0, 10, 0, 0, 0],
        [0, 1/(12*0.1*0.1), 0, 0, 0, 0],
        [0, 0, 1/(8*0.02*0.02), 0, 0, 0],
        [0, 0, 0, 1/(12*0.01*0.01), 0, 0],
        [0, 0, 0, 0, 1/(8*0.02*0.02), 0],
        [0, 0, 0, 0, 0, 1/(12*0.01*0.01)]]])
R = 0.00001

############## LQR gain ##################################
K = control.lqr(A,B,Q,R)

############## ODE solver for non linear model ##########
def pend_non_linear(y, t, A, B, K_):
        u = np.matmul(K_,y.reshape((-1,1)))[0,0]
        d1 = y[1] + np.random.normal(mean, std_dev)
        DD = M + m1 + m2 - m1*(np.cos(y[2])**2) - m2*(np.cos(y[4])**2)
        d2 = -(1/DD)*(m1*g*np.sin(2*y[2])/2 + m2*g*np.sin(2*y[4])/2
                + m1*l1*y[3]*y[3]*np.sin(y[2]) +
                m2*l2*y[5]*y[5]*np.sin(y[4]) + u)
                + np.random.normal(mean, std_dev)
        d3 = y[3] + np.random.normal(mean, std_dev)
        d4 = (1/l1)*(d2*np.cos(y[2]) - g*np.sin(y[2]))
                + np.random.normal(mean, std_dev)
        d5 = y[5] + np.random.normal(mean, std_dev)
        d6 = (1/l2)*(d2*np.cos(y[4]) - g*np.sin(y[4]))
```

```
            + np.random.normal(mean, std_dev)
    return np.array([d1,d2,d3,d4,d5,d6])

############ ODE solver for Observer  ########
def obser(x_hat, t, L_, y_):
    u = -np.matmul(K[0],x_hat.reshape(-1,1))
    dxdt = np.matmul((A - np.matmul(L_,C)),x_hat.reshape(-1,1))
            + B*u + np.matmul(L_,y_.reshape((-1,1)))
    return dxdt.reshape(-1)

############# Solving non linear equation ############################
x0 = np.array([[0],[0],[-0.2],[0],[0.3],[0]])
t = np.arange(0, 100, 0.1)
#####################################################################
L = control.place(np.matrix.transpose(A),
    np.matrix.transpose(C), K[2])
L_ = np.matrix.transpose(L)
x_hat0 = x0.reshape(6)
u = -np.matmul(K[0],x_hat0)
observed = []
observed.append(x_hat0)

for i in range(len(t) - 1):
    print("solving␣"+str(i))
    X = odeint(pend_non_linear, x_hat0, t[i:i+2], args=(A, B, K[0]))
    y = np.matmul(C,(X[1]).reshape(-1,1)).reshape(-1)
            + np.random.normal(mean, std_dev)
    x_hat = odeint(obser , x_hat0 , t[i:i+2], args=(L_,y))
    x_hat0 = x_hat[1]
    u = -np.matmul(K[0],x_hat0)
    observed.append(x_hat0)

sol = np.array(observed)

if run_animation:
    cartAnimatter = cartPlotter()
    cartAnimatter.run_animation(sol, t[:sol.shape[0]], "Observer")

response = responsePlot()
response.plotResponse(sol, t, K,
    "Response␣of␣non␣linear␣system␣at␣noisy␣data")

############################ END ###########################################
```