THE UNIVERSITY OF TEXAS AT SAN ANTONIO
EE 5243
INTRODUCTION TO CYBER-PHYSICAL SYSTEMS

HOMEWORKS # 6 + 7
SOLUTIONS

Ahmad F. Taha
November 8, 2015

READ Homework Instructions:

1. Type your solutions. This homework is mainly a programming assignment.
2. This is a very long problem set. You have to start early and ask questions when (if) in doubt.
3. **Due date:** Tuesday, October, 27th, @ 5:00pm on Blackboard, **AND** drop off a copy of your solutions (slip it under the office door if I'm away).
4. **Collaboration policy:** you cannot collaborate with anyone on this homework. It's mainly a programming exercise; you have to write your own codes.
5. Solutions that are unclear won't be graded.
6. Before you start with this homework assignment, make sure that you have grasped the content of Module 06.
7. This homework weighs twice as much a typical homework, as it's a bit more time consuming and you're given more time to complete it.
8. You should include the codes in the submitted PDF as well as commented m-files, to be uploaded on Blackboard.

The objective of this homework assignment is to teach you basic implementation of a model predictive control (MPC) engine for a nonlinear system. That is, using **real-time measurements (outputs) from the nonlinear dynamics of a system**, the objective is to design an MPC that uses the **linearized**, discretized version of the nonlinear system (read this sentence again and again).

This assignment is mainly a programming assignment. Thus, you will have to present well-commented codes. You are also required to upload your codes to Blackboard, in addition to a `ReadMe.txt` file that explains the structure of your files. Poorly developed codes **WILL** be *poorly graded* — whatever that means!

The $\theta - R$ Robotic Manipulator

In this problem set, the dynamics of a nonlinear dynamical system — the $\theta - R$ robotic manipulator — are summarized from [1]. The below figure shows a representation of the manipulator with a simplistic lumped mass representation of the system. The dynamics of the manipulator are given as

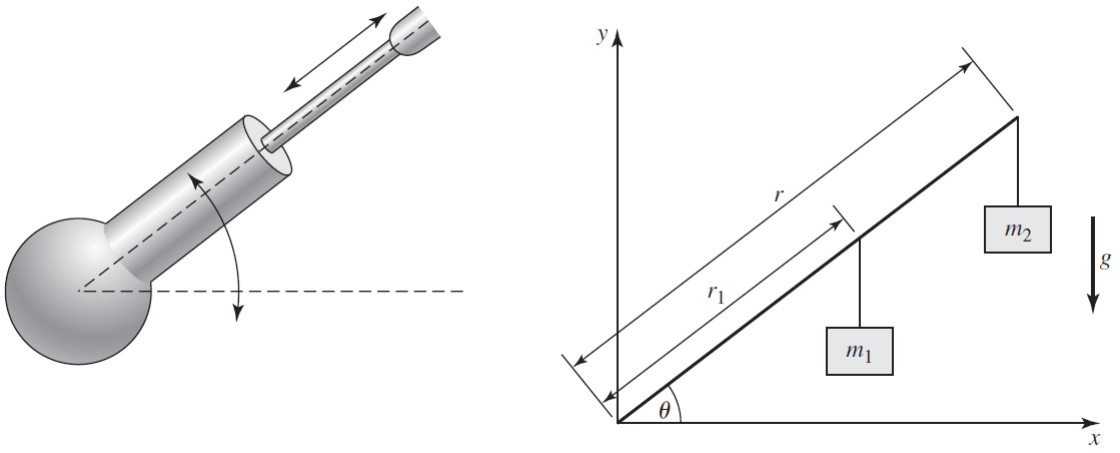

Figure 1: Figure to the left shows the $\theta - R$ manipulator, whereas the figure to the right illustrate the lumped mass representation; figures are taken from [1]. For more on the derivation of the state-space representation of this dynamical system, the reader is referred to [1].

follows:

$$\dot{x} = f(x,u) = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ \dfrac{-2m_2x_2x_3x_4 - g\,(m_1r_1 + m_2x_2)\cos(x_1) + u_1}{m_1r_1^2 + m_2x_2^2} \\ x_3^2x_2 - g\sin(x_1) + \dfrac{u_2}{m_2} \end{bmatrix}, \tag{1}$$

where:

- $x_1 = \theta$ is the angle (see Figure 1);
- $x_2 = r$ is the varying radius (see Figure 1);
- $x_3 = \dot{\theta}$ is the derivative of θ;
- $x_4 = \dot{r}$ is the derivative of x_2;
- $u_1 = T_\theta$ is the first control which is the torque;
- $u_2 = F_r$ is the second control input which is the translational force.

One of this assignment's objective is to compute optimal control trajectories for u_1 and u_2 such that a certain cost function is minimized, system dynamics are satisfied, and the controls are bounded. The problems in this assignment are all related.

Problem 1 — Linearization and System Properties

Answer the following questions:

1. The following is given for this problem and subsequent ones:
   (a) $m_1 = 10, m_2 = 3, r_1 = 1, g = 9.81$
   (b) Equilibrium point (that is generated by solving $\dot{x} = 0$) is: $x_e^\top = \begin{bmatrix} \frac{\pi}{4} & 2 & 0 & 0 \end{bmatrix}^\top$.

   Given the above, obtain $u_e = \begin{bmatrix} u_{e_1} \\ u_{e_2} \end{bmatrix}$ — the equilibrium control. Recall that $\dot{x} = f(x_e, u_e) = 0$. You can do this analytically or on MATLAB. The command `solve` can be useful.

2. For x_e and the computed u_e, obtain a linearized version of the nonlinear dynamics given in (1), i.e., find matrices A and B. Your dynamics should have the following form:

$$\Delta\dot{x}(t) = A\Delta x(t) + B\Delta u(t),\ \Delta x(t) = x(t) - x_e, \Delta u(t) = u(t) - u_e.$$

   You can do this analytically or on MATLAB (which is preferred). The command `jacob` and `subs` can be useful. With that in mind, we will allow ourselves to abuse this notation and use $\dot{x} = Ax + Bu$ instead.

3. Assume that the output of the linearized system is simply x_1 and x_2 (**not** $x_1 + x_2$). What is your C-matrix?

4. Write a MATLAB function that takes **ANY** A, B, C as inputs and generates a sequence of 1's and 0's that answers the following questions:
   (a) Is the system asymptotically stable?
   (b) Is the system controllable?
   (c) Is the system observable?
   (d) Is the system stabilizable?
   (e) Is the system detectable?

   Your function should work with any linearized system, i.e., for any dimensions and any state-space matrices. You should use the PBH-test to get some of the answers above for your m-file. If the answers to the above five questions are all "Yes", the output should be an array of 1's (`answer = [1 1 1 1 1]`). If the system satisfies all the properties besides detectability, your output should be `[1 1 1 1 0]`.

Solutions:

```
%% EE 5243: Optimization and Control of CPS
%% Author: Ahmad F. Taha
%% Date: 10/15/2015
%% Title: Homework 6+7 Solutions

% Clearing all
close all;
clear all;
clc;

%% Probelm 1: Linearization and System Properties

%% Problem 1-1: Obtain u_e
```

```
% Write down differential equations modeling the system.
% Represent the nonlinear model in the state-space format.

% Define variables

% Constants first

syms m1 m2 g r1 xe1 x_e1 x_e2 x_e3 x_e4;

% States

syms x1 x2 x3 x4 x_1dot x_2dot x_3dot x_4dot x xdot;

% Inputs & Outputs

syms u1 u2;

syms y1 y2;

% Loading constants

m1=10;
m2=3;
r1=1;
g=9.81;

x_e1=pi/4;
x_e2=2;
x_e3=0;
x_e4=0;


% Nonlinear State equations: state-space format;

x_1dot = x3;
x_2dot = x4;
x_3dot = (-2*m2*x3*x2*x4-g*cos(x1)*(m1*r1+m2*x2)+u1)/(m1*r1^2+m2*x2^2);
x_4dot = x3^2*x2-g*sin(x1)+(u2)/(m2);

x=[x1;x2;x3;x4];
u=[u1;u2];
xdot=[x_1dot;x_2dot;x_3dot;x_4dot];

%% Finding u_equilbirium

xe=[x_e1;x_e2;x_e3;x_e4];

feq = subs(xdot,x,xe);

ue1 = solve(feq(3) == 0);
ue2 = solve(feq(4) == 0);

ue = [ue1;ue2];
ue = double(ue)

%% Problem 1-2: Obtain A,B: the state-space matrices
```

```
%% Perform linearization & construct a state-space model of the linear model

A_jacob = jacobian(xdot,x);
B_jacob = jacobian(xdot,u);

A = subs(A_jacob,x,xe);
A = subs(A,u,ue);
B = subs(B_jacob,u,ue);
B = subs(B,x,xe);


A = double(A)
B = double(B)

%% Problem 1-3: Obtain C if the outputs are x1 and x2
% Clearly, the C-matrix can be written as:

C = [1 0 0 0; 0 1 0 0]
D =  zeros(2,2)

%% Problem 1-4: LTI system properties indicator

bool =LTISysProps(A,B,C)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% LTISysPropos m-file

function bool =LTISysProps(A,B,C)

eigval=eig(A);

flag_asystab=[];
flag_cont=[];
flag_stab=[];
flag_obs=[];
flag_det=[];

for i=1:size(A,1)
if (real(eigval(i))>=0  )
flag_asystab=0;
else
flag_asystab=1;
end
end

if(rank(ctrb(A,B))==size(A,1))   %need check
flag_cont=1;
else
flag_cont=0;
end

if(rank(obsv(A,C))==size(A,1))   %need check
flag_obs=1;
else
```

```
flag_obs=0;
end



for i=1:size(A,1)
if (rank([eigval(i)*eye(size(A, 1))-A,B])~=size(A,1) && real(eigval(i))>=0  )
flag_stab=0;
else
flag_stab=1;
end

end

for i=1:size(A,1)
if (rank([eigval(i)*eye(size(A, 1))-A;C])~=size(A,1) && real(eigval(i))>=0  )
flag_det=0;
else
flag_det=1;
end
end


bool=[flag_asystab flag_cont flag_stab flag_obs flag_det];
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Outputs:
ue =

110.9875
20.8102

A =

0         0    1.0000         0
0         0         0    1.0000
5.0449   -0.9459         0         0
-6.9367         0         0         0

B =

0         0
0         0
0.0455         0
0    0.3333

C =

1     0     0     0
0     1     0     0

D =
```

```
0     0
0     0

bool =

0     1     1     1     1
```

Problem 2 — Design of LSF and Observer Matrix Gains

1. If the linearized dynamical system is stabilizable, design a linear state-feedback matrix gain K such that the eigenvalues of the closed-loop system are located at: $[-2-3-4-5]$.
2. If the linearized dynamical system is detectable, design a linear observer gain L such that the eigenvalues of the closed-loop system are located at: $[-2-3-4-5]$.

Solutions:

```
%% Problem 2: Design of LSF and Observer Matrix Gains

lam_new=[-2,-3,-4,-5];

%% Problem 2-1

K=place(A,B,lam_new)

%% Problem 2-2

L=place(A',C',lam_new)'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%Outputs:

K =

550.9875  -20.8102  198.0000         0
-20.8102   18.0000         0   15.0000

L =

9.0000         0
0    5.0000
25.0449   -0.9459
-6.9367    6.0000
```

Problem 3 — Dynamical Simulation Using an ODE Solver, Control & Estimation

You will now simulate the performance of your system given a control law.

1. Develop a MATLAB file that depicts the trajectory or the behavior of the closed-loop system comprised of the state-feedback **controller driving the nonlinear model of the robot for different initial conditions. You will have to follow this procedure:**

   (a) Your given data should be: matrices A, B, C, K, L, vectors x_e, u_e, x_0, and given system constants (m_1, m_2, g, r_1). You should have these quantities in the beginning of your m-file.

   (b) Read about the `ode45` solver on MATLAB, and know how to simulate the dynamics of any nonlinear system. Here's a Mathworks link: http://www.mathworks.com/help/matlab/ref/ode45.html?refresh=true.

   (c) Note that your control law is given by: $\Delta u = -K\Delta x$, hence: $u = -K\Delta x + u_e$

   (d) Write a MATLAB function that represents the **nonlinear dynamics** of the manipulator. Name the function '`thetaRdynamics.m`'. If your control is constant, the function should look like that:

```
function [dx]= thetaRdynamics(t,x)
 % constants here
 % matrix gains here
 % constant vectors here
 % Deltax = x - xe;
 % Your control law here: u=-K Deltax + ue
 % Your nonlinear dynamics here: dx = f(x,u)
end
```

   (e) Specify your time-span to be: `tspan=0:0.01:5` and call the `ode45` solver when your initial plant conditions are all zero.

   (f) Plot the states trajectories (x_1, x_2, x_3, x_4) in addition to the two controls (u_1, u_2) for zero initial conditions. Your plots should have lables, titles, legends. Ugly plots often receive ugly reviews.

   (g) Repeat (f) when you change your initial conditions. Start from random ones in this case.

   (h) Using the computed matrix gain L, simulate the observer of the nonlinearized dynamical system, that is:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}).$$

   To do that, you will have to write a new m-file for the nonlinear dynamics, i.e., you'll have to write a file which you can call `thetaRdynamics_observer(t,x)` (the size of x here is 8, as it includes states of the nonlinear dynamics and the estimator) that dynamically simulates the observer and then uses it for the `ode45` solver.

      i. Start from the following initial conditions: $x_0^\top = [0.5\pi \quad 1.5 \quad 0 \quad 0]$ and zero estimator conditions.

      ii. Illustrate through your plots that the estimator's states are approaching the actual states, generated from the `ode45` solver.

      iii. Repeat this experiment for different initial conditions of your choice.

Solutions:

```
%% Problem 3: Dynamical Simulation Using an ODE Solver

%% Problem 3-1-(a)--(f)

% Define constants
```

```
xe1=pi/4;
xe2=2;
xe3=0;
xe4=0;
xe=[xe1;xe2;xe3;xe4];
ue1=ue(1);
ue2=ue(2);
tol=1e-7;
options=odeset('RelTol',tol);

% Initial conditions
x0 = [0;0;0;0];

% Timespan
tspan = 0:0.01:5;


% Call the solver
[t,X] = ode45(@thetaRdynamics,tspan,x0,options);
x1 = X(:,1);
x2 = X(:,2);
x3 = X(:,3);
x4 = X(:,4);
Dx=[];
u=[];
y=[];
for i = 1:1:length(x1)
Dx=X(i,:)-xe';
Dx=Dx';
u(:,i)=-K*Dx+ue;
y(:,i)=C*X(i,:)';
end

% Plot the results for zero initial conditions

figure
hold on
plot(t,x1,'LineWidth',3);
plot(t,x2,'r','LineWidth',3);
l=legend('$\theta$','$R$');
grid;
set(l,'interpreter','latex','fontsize',16);
ylabel('$\theta,R$','interpreter','latex','fontsize',16);
xlabel('Time (seconds)','interpreter','latex','fontsize',16);
title('$\theta,R$ vs. Time for $x_0
 = [ 0 \hspace{0.2cm} 0 ... \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $','interpreter','latex','fontsize',16);




figure
hold on
plot(t,x3,'LineWidth',3);
plot(t,x4,'r','LineWidth',3);
grid;
l=legend('$\dot{\theta}$','$\dot{R}$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$\dot{\theta},\dot{R}$','interpreter','latex','fontsize',16)
xlabel('Time (seconds)','interpreter','latex','fontsize',16)
```

```
title('$\dot{\theta},\dot{R}$ vs. Time for $x_0 = [0 \hspace{0.2cm} 0 \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $'


figure
hold on
plot(t,u(1,:),'LineWidth',3);
plot(t,u(2,:),'r','LineWidth',3);
grid;
l=legend('$T_{\theta}$','$F_r$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$T_{\theta},F_r$','interpreter','latex','fontsize',16);
xlabel('Time (seconds)','interpreter','latex','fontsize',16);
title('$T_{\theta},F_r$ vs. Time for $x_0 = [ 0 \hspace{0.2cm} 0 \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $','int
%% ODE Solver:
function [dx]= thetaRdynamics(t,x)

m1=10; m2=3; g=9.81; r1=1; ue1=110.98; ue2=20.81; xe1=pi/4;
xe2=2; xe3=0; xe4=0;

K =[550.9875  -20.8102  198.0000         0
-20.8102   18.0000         0   15.0000];
xe=[xe1;xe2;xe3;xe4];
ue=[ue1;ue2];
Dx=x-xe;
u=-K*Dx+ue;

dx(1) = x(3);
dx(2) = x(4);
dx(3) = (-2*m2*x(2)*x(3)*x(4)-g*cos(x(1))*(m1*r1+m2*x(2))+u(1))/(m1*r1^2+m2*x(2)^2);
dx(4) = (x(3))^2*x(2)-g*sin(x(1))+u(2)/m2;
dx = dx';
end
%%%%%%%%%%%%%%%%%%%%%
%% Problem 3-1-(g)--Simulating for different initial conditions

clear X
clear t

% Random Initial conditions
x0 = randn(4,1);

% Timespan
tspan = 0:0.01:5;


% Call the solver
[t,X] = ode45(@thetaRdynamics,tspan,x0,options);
x1 = X(:,1);
x2 = X(:,2);
x3 = X(:,3);
x4 = X(:,4);
Dx=[];
u=[];
y=[];
for i = 1:1:length(x1)
Dx=X(i,:)-xe';
Dx=Dx';
u(:,i)=-K*Dx+ue;
y(:,i)=C*X(i,:)';
end
```

```
% Plot the results for zero initial conditions

figure
hold on
plot(t,x1,'LineWidth',3);
plot(t,x2,'r','LineWidth',3);
l=legend('$\theta$','$R$');
grid;
set(l,'interpreter','latex','fontsize',16);
ylabel('$\theta,R$','interpreter','latex','fontsize',16);
xlabel('Time (seconds)','interpreter','latex','fontsize',16);
title('$\theta,R$ vs. Time for random $x_0$','interpreter','latex','fontsize',16);




figure
hold on
plot(t,x3,'LineWidth',3);
plot(t,x4,'r','LineWidth',3);
grid;
l=legend('$\dot{\theta}$','$\dot{R}$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$\dot{\theta},\dot{R}$','interpreter','latex','fontsize',16)
xlabel('Time (seconds)','interpreter','latex','fontsize',16)
title('$\dot{\theta},\dot{R}$ vs. Time for random $x_0$','interpreter','latex','fontsize',16)


figure
hold on
plot(t,u(1,:),'LineWidth',3);
plot(t,u(2,:),'r','LineWidth',3);
grid;
l=legend('$T_{\theta}$','$F_r$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$T_{\theta},F_r$','interpreter','latex','fontsize',16);
xlabel('Time (seconds)','interpreter','latex','fontsize',16);
title('$T_{\theta},F_r$ vs. Time for random $x_0$','interpreter','latex','fontsize',16);

%% Problem 3-1-(h)--Simulating the behavior of the observer

x0 = [pi/2;1.5;0;0;0;0;0;0];
%timespan
tspan =0:0.01:5;
%call the solver
[t,X] = ode45(@thetaRdynamics_observer,tspan,x0,options);
x1 = X(:,1);
x2 = X(:,2);
x3 = X(:,3);
x4 = X(:,4);
z1 = X(:,5);
z2 = X(:,6);
z3 = X(:,7);
z4 = X(:,8);
Dx=[];
u=[];
y=[];
for i = 1:1:length(x1)
u(:,i) = -K*X(i,5:8)'+ue;
y(:,i) = C*X(i,1:4)';
```

```
end


% The Objective in these simulations is to see how the estimated state z is
% tracking $$ \Delta x, \textbf{z} \rightarrow \Delta \textbf{x} $$
figure
hold on
plot(t,x1-xe1,'b','LineWidth',3);
plot(t,x2-xe2,'r','LineWidth',3);
plot(t,z1,'b--','LineWidth',3);
plot(t,z2,'r--','LineWidth',3);
l=legend('$\Delta \theta$','$\Delta R$','$\hat{\theta}$','$\hat{R}$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$\Delta \theta,\Delta R,\hat{\theta},\hat{R}$','fontsize',16,'interpreter','latex');
xlabel('Time (seconds)','fontsize',16,'interpreter','latex');
title('$\Delta \theta ,\Delta R ,\hat{\theta},\hat{R}$ vs. Time for $x_0 = [ pi/2 \hspace{0.2cm} 1.5 \hspace{0.2cm}

figure
hold on
plot(t,x3-xe3,'b','LineWidth',3);
plot(t,x4-xe4,'r','LineWidth',3);
plot(t,z3,'b--','LineWidth',3);
plot(t,z4,'r--','LineWidth',3);
l=legend('$\Delta \dot{\theta} $','$\Delta \dot{R} $','$\hat{\dot{\theta}}$','$\hat{\dot{R}}$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$\Delta \dot{\theta },\Delta \dot{R },\hat{\dot{\theta}},\hat{\dot{R}}$','fontsize',16,'interpreter','latex
xlabel('Time (seconds)','fontsize',16,'interpreter','latex');
title('$\Delta \dot{\theta} ,\Delta \dot{R} ,\dot{\hat{\theta}},\dot{\hat{R}}$ vs. Time for $x_0 = [ pi/2 \hspace{0

% Observer succeeds in tracking $\Delta x$

set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,u(1,:),'LineWidth',3);
plot(t,u(2,:),'LineWidth',3);
l=legend('$T_{\theta}$','$F_r$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$T_{\theta},F_r$','fontsize',16,'interpreter','latex');
xlabel('Time (seconds)','fontsize',15,'interpreter','latex');
title('$T_{\theta},F_r$ vs. Time for $x_0 = [ pi/2 \hspace{0.2cm} 1.5 \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [dx]= thetaRdynamics_observer(t,x)

m1=10; m2=3; g=9.81; r1=1; ue1=110.98;
ue2=20.81; xe1=pi/4; xe2=2; xe3=0; xe4=0;
A =[
0         0    1.0000         0
0         0         0    1.0000
5.0449   -0.9459         0         0
-6.9367         0         0         0];

B =[
0         0
0         0
0.0455         0
0    0.3333];

C = [1 0 0 0; 0 1 0 0];
```

```
L =[9.0000         0
0    5.0000
25.0449   -0.9459
-6.9367    6.0000];

K =[550.9875  -20.8102  198.0000         0
-20.8102   18.0000         0   15.0000];

xe=[xe1;xe2;xe3;xe4];

ue=[ue1;ue2];

u  = -K*x(5:end)+ue;

Du=u-ue;

AA=A-L*C; % 4by4
Dy1 = C(1,1)*(x(1)-xe(1))+C(1,2)*(x(2)-xe(2))+C(1,3)*(x(3)-xe(3))+C(1,4)*(x(4)-xe(4));
Dy2 = C(2,1)*(x(1)-xe(1))+C(2,2)*(x(2)-xe(2))+C(2,3)*(x(3)-xe(3))+C(2,4)*(x(4)-xe(4));


dx(1) = x(3);
dx(2) = x(4);
dx(3) = (-2*m2*x(2)*x(3)*x(4)-g*cos(x(1))*(m1*r1+m2*x(2))+u(1))/(m1*r1^2+m2*x(2)^2);
dx(4) = (x(3))^2*x(2)-g*sin(x(1))+u(2)/m2;
dx(5) = AA(1,1)*x(5)+AA(1,2)*x(6)+AA(1,3)*x(7)+AA(1,4)*x(8)+L(1,1)*Dy1+L(1,2)*Dy2+B(1,1)*Du(1)+B(1,2)*Du(2);
dx(6) = AA(2,1)*x(5)+AA(2,2)*x(6)+AA(2,3)*x(7)+AA(2,4)*x(8)+L(2,1)*Dy1+L(2,2)*Dy2+B(2,1)*Du(1)+B(2,2)*Du(2);
dx(7) = AA(3,1)*x(5)+AA(3,2)*x(6)+AA(3,3)*x(7)+AA(3,4)*x(8)+L(3,1)*Dy1+L(3,2)*Dy2+B(3,1)*Du(1)+B(3,2)*Du(2);
dx(8) = AA(4,1)*x(5)+AA(4,2)*x(6)+AA(4,3)*x(7)+AA(4,4)*x(8)+L(4,1)*Dy1+L(4,2)*Dy2+B(4,1)*Du(1)+B(4,2)*Du(2);
dx = dx';
end
```

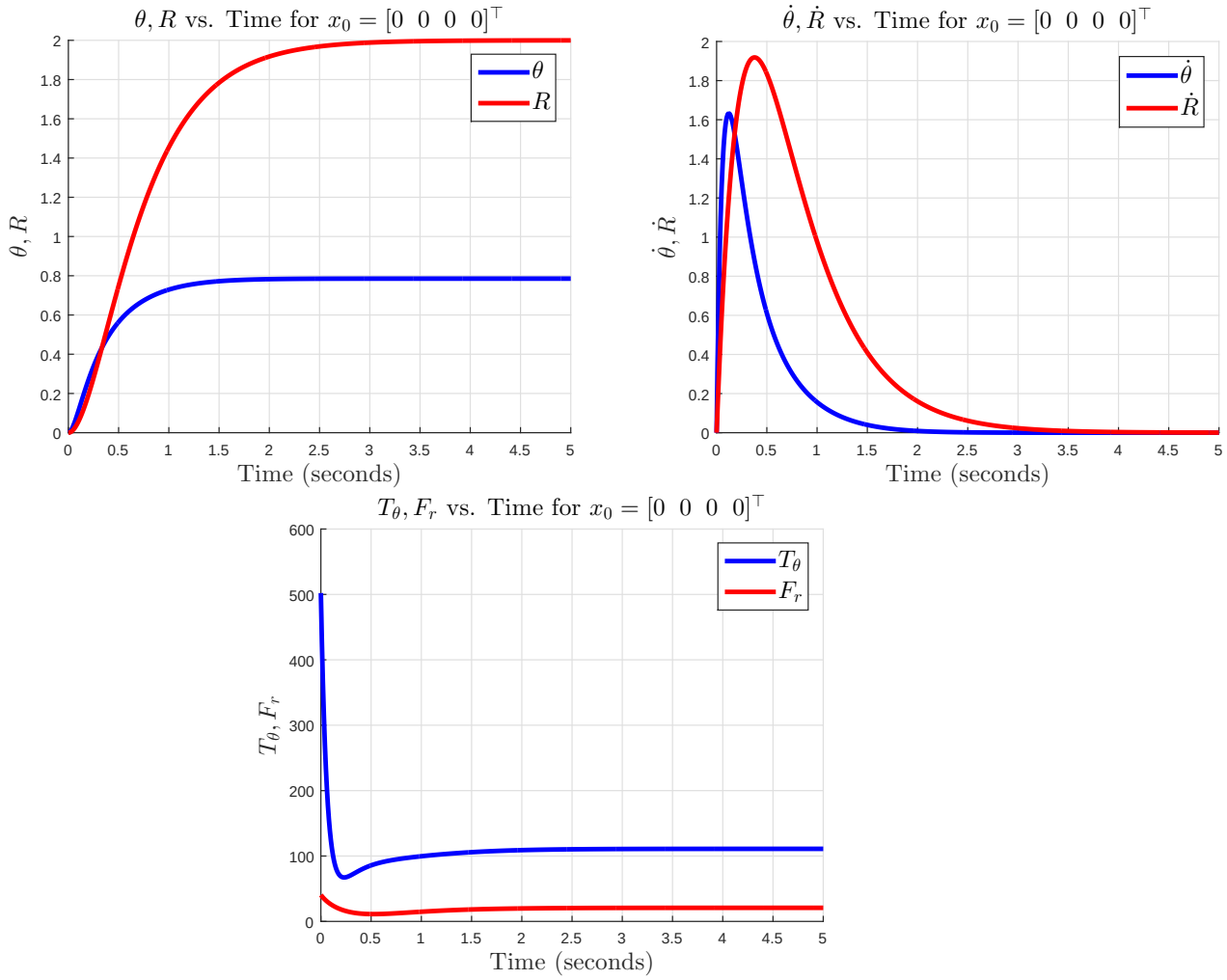

Figure 2: The above three figures show the state-response (top figures) and the control trajectory, given zero initial conditions.

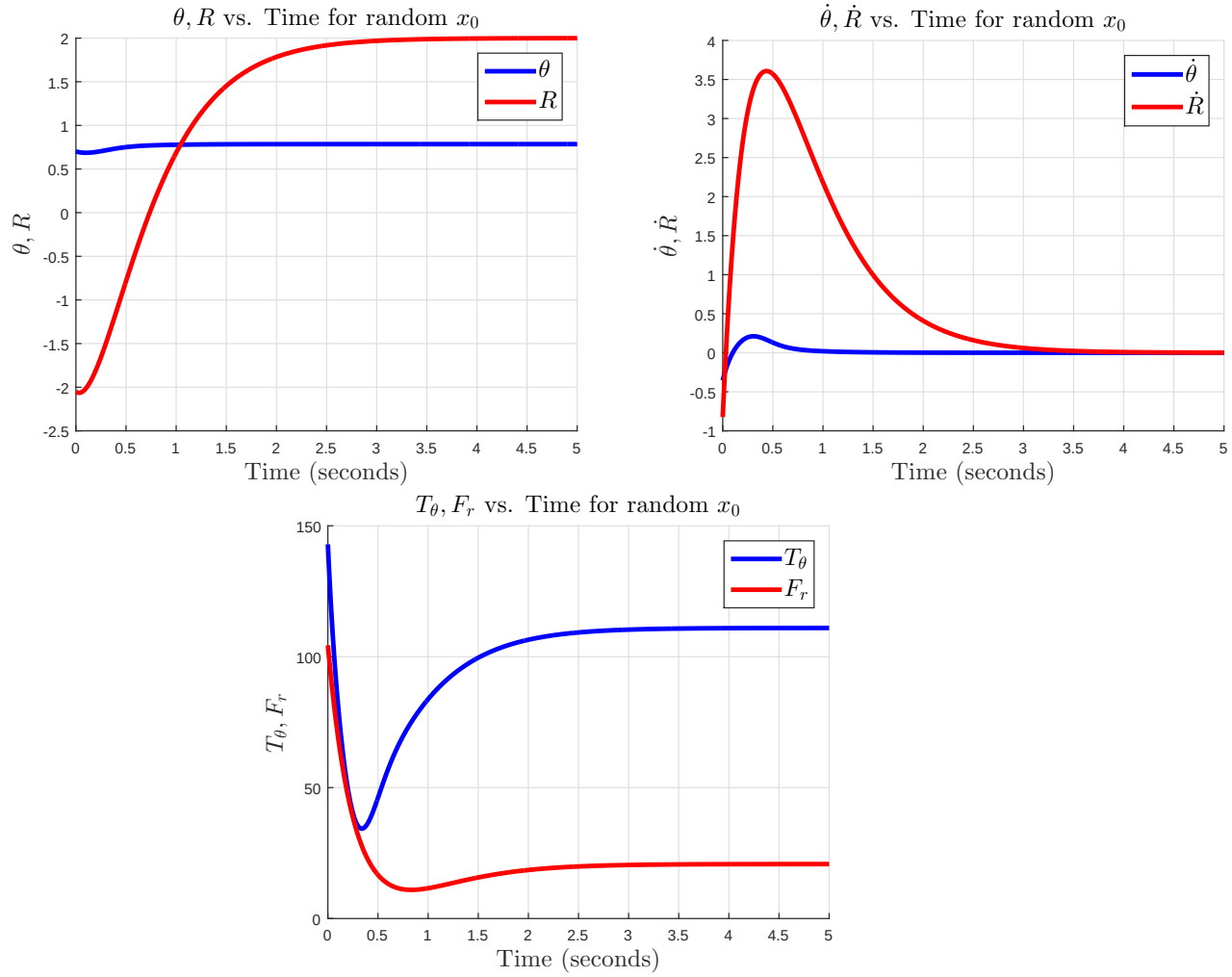

Figure 3: The above three figures show the state-response and the control trajectory, given randomly generated initial conditions.

Problem 4 — Unconstrained MPC for the Nonlinear System

In this problem, you will design an MPC for the nonlinear system, using the linearized dynamics of the system. Read the instructions carefully.

1. First, write a MATLAB function that takes the following inputs: matrices A, B, C of a CT LTI system, N_p (prediction horizon, which we assume to be equal to the control horizon), and h (the sampling period needed for discretization). This function should:
   (a) Discretize the continuous system.
   (b) Construct the augmented MPC dynamics (from Module 6).
   (c) Compute matrices W, Z (Module 6).
   (d) This function should be something like this:
   `function [W,Z,Ca,PhiA,GammaA] = MPCMATs(A,B,C,Np,h).`
   (e) Your function should work for **any MIMO dynamical system of all sizes**.
2. Using `MPCMATs` function, write another function that finds the optimal control ΔU:
   (a) This function should be something like this:
   `function DU = optimizer(xa,A,B,C,R,Q,r,Np,h)`

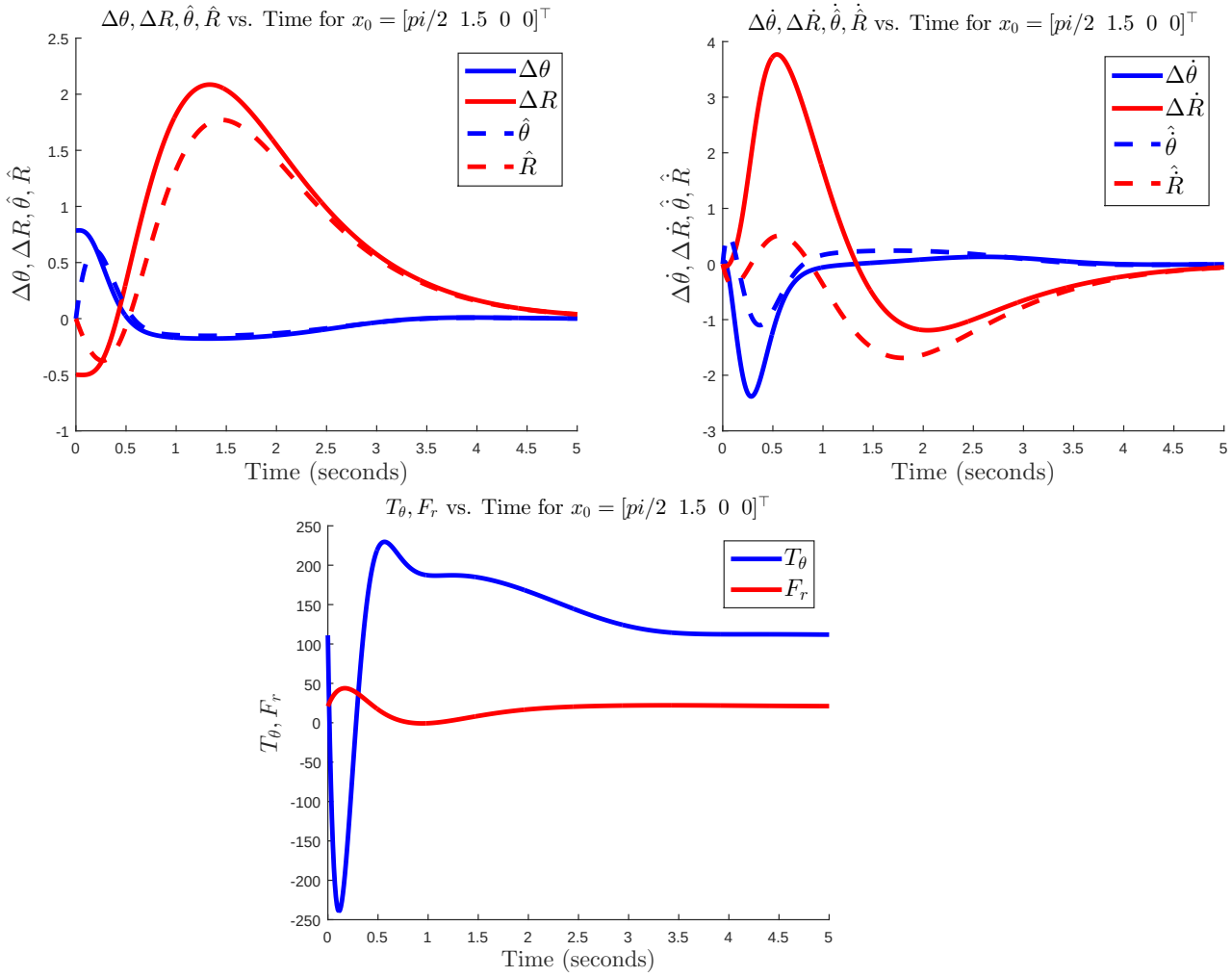

Figure 4: The above figures show the state-response with the estimated generated from the Luenberger observer for the nonlinear system (top two figures), as well as the control trajectories. The observer succeeds in tracking the controlled system behavior, as illustrated in the figures.

(b) This function obviously calls `MPCMATs` to generate Z, W

(c) Matrices Q, R are weight matrices discussed in Module 6, x_a is the initial vector for the MPC augmented system, and r is the reference signal.

(d) The output of this function is the optimal control for a predicted horizon.

(e) You have basically designed an MPC now without constraints on the control or states.

3. You will have to simulate the unconstrained MPC control law **on the nonlinear continuous model of the system.** Do the following:

(a) The following is given:

$$N_p = 40, h = 0.101, x_0^\top = \begin{bmatrix} \pi/8 & 1 & 0 & 0 \end{bmatrix}, y_0 = Cx_0, u_0 = 0, T_{final} = 20$$

(b) The weight matrices and reference signals are:

$$R = \text{diag}(0.1, 0.1), Q = \text{diag}(4, 0.1), r = \begin{bmatrix} -0.1 \\ -0.2 \end{bmatrix}$$

Of course, the given reference signals and weight matrices will have much bigger sizes as you should do consider the predicted horizon, and hence the given costs are for only one

time-step. The MATLAB commands `kron` and `repmat` should be helpful in generating the actual Q, R and r — quantities that will be used for the optimizer function.

(c) Your simulation file should have the following structure (this is only a pseudo-code):

```
% All the constants
% All the matrices and initial conditions
% UU(:,1)=u0;
% for k=1:1:(Tfinal/h)
% tspan(:,k)=(k-1)*h:0.001:k*h;
% tt(k)=tspan(1,k);
% [t,X] = ode45(@thetaRdynamics,[(k-1)*h k*h],XX(:,k),options,ue+UU(:,k));
% Find x_a for the new iteration
% Find DU optimal for the new horizon using the optimizer function
% Extract UU (or u(k))
% end
```

(d) Plot the states trajectories (x_1, x_2, x_3, x_4) in addition to the two controls (u_1, u_2). Your plots should have lables, titles, legends. Again, nasty plots get nasty reviews ;).

(e) Change your initial conditions and R, Q, r values and explain the corresponding outputs and plots.

Solutions:

```
%% Problem 4: Unconstrained MPC Design for the NL System

tol=1e-7;
XX=[];
YY=[];
UU=[];
n=4;
p=2;
% Initial Conditions 1: Zero ICs:
x0 = [pi/8;1;0;0]; % x(0)
y0 = C*x0;
ye=C*xe;
u0=[0;0]; % u(0)
XX(:,1)=x0;
YY(:,1)=y0;
UU(:,1)=u0;
options=odeset('RelTol',tol);
h=0.101;
Np=40;
R = [0.1 0;0 0.1];
R = kron(eye(Np),R);
Q = [4 0; 0 0.1];
Q = kron(eye(Np),Q);
r = [-0.1;-0.2];
r = repmat(r,Np,1);
tspan=[];
Tfinal=20;
tt=[];
for k=1:1:(Tfinal/h)
tspan(:,k)=(k-1)*h:0.001:k*h;
tt(k)=tspan(1,k);
[t,X] = ode45(@thetaRdynamics_MPC,[(k-1)*h k*h],XX(:,k),options,ue+UU(:,k));
XX(:,k+1) = X(end,:)';
YY(:,k+1) = C*XX(:,k+1);
xa=[XX(:,k+1)-XX(:,k);YY(:,k+1)-ye];
DU_optimal = optimizer(xa,A,B,C,R,Q,r,Np,h);
Du=[eye(p) zeros(2,2*(Np-1))]*DU_optimal;
```

```
UU(:,k+1)=UU(:,k)+Du;
end
t=[tt Tfinal];

set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,XX(1,:),'LineWidth',3);
plot(t,XX(2,:),'r','LineWidth',3);
l=legend('$\theta$','$R$');
grid;
set(l,'interpreter','latex','fontsize',16,'interpreter','latex')
ylabel('$\theta,R$','fontsize',16,'interpreter','latex');
xlabel('Time (seconds)','fontsize',15,'interpreter','latex');
title('$\theta,R$ vs. Time for $x_0 = [ 0 \hspace{0.2cm} 0  \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $','fontsize


set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,XX(3,:),'LineWidth',3);
plot(t,XX(4,:),'r','LineWidth',3);
grid;
l=legend('$\dot{\theta}$','$\dot{R}$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$\dot{\theta},\dot{R}$','fontsize',16,'interpreter','latex')
xlabel('Time (seconds)','fontsize',15,'interpreter','latex')
title('$\dot{\theta},\dot{R}$ vs. Time for $x_0 = [0 \hspace{0.2cm} 0 \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $'


set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,ue(1)+UU(1,:),'LineWidth',3);
plot(t,ue(2)+UU(2,:),'r','LineWidth',3);
grid;
l=legend('$T_{\theta}$','$F_r$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$T_{\theta},F_r$','fontsize',16,'interpreter','latex')
xlabel('Time (seconds)','fontsize',15,'interpreter','latex')
title('$T_{\theta},F_r$ vs. Time for $x_0 = [ 0 \hspace{0.2cm} 0 \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top} $','fon

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% MPCMATs m-file

function [W,Z,Ca,PhiA,GammaA] = MPCMATs(A,B,C,Np,h)
Phi=expm(A*h);
gB = @(rho) expm(A*rho)*B;
Gamma = integral(gB,0,h,'ArrayValued',true);
p=2;
n=4;
Ca=[zeros(p,n) eye(p)];
GammaA=[Gamma;C*Gamma];
PhiA=[Phi zeros(n,p);C*Phi eye(p)];
[q,w]=size(Ca);

W=zeros(q*Np,w);
Z=zeros(q*Np,q*Np);

for i=1:1:Np
```

```
W(2*i-1:2*i,:)=Ca*(PhiA^(i));
end


for ii=1:1:Np
Z(2*ii-1:2*ii,2*ii-1:2*ii)=Ca*GammaA;
end

M=[];

for k=1:1:Np
Z(2*k-1:2*k,1:2)=Ca*(PhiA^(k-1))*GammaA;
end

L=Z(:,1:2);

for kk=1:1:Np-1
M(:,2*kk-1:2*kk)=[zeros(2*kk,2);L(1:end-2*kk,:)];
end
Z=[L M];
end

%% optimizer.m

function DU = optimizer(xa,A,B,C,R,Q,r,Np,h)
[W,Z] = MPCMATs(A,B,C,Np,h);
DU_optimal=((R+Z'*Q*Z)^-1)*Z'*Q*(r-W*xa);
DU=DU_optimal;
end

%% thetaRdynamics_MPC(t,x,u)
function [dx]= thetaRdynamics_MPC(t,x,u)
% Theta-R dynamics didn't change
m1=10; m2=3; g=9.81; r1=1;
dx(1) = x(3);
dx(2) = x(4);
dx(3) = (-2*m2*x(2)*x(3)*x(4)-g*cos(x(1))*(m1*r1+m2*x(2))+u(1))/(m1*r1^2+m2*x(2)^2);
dx(4) = (x(3))^2*x(2)-g*sin(x(1))+u(2)/m2;

dx = dx';
end
```

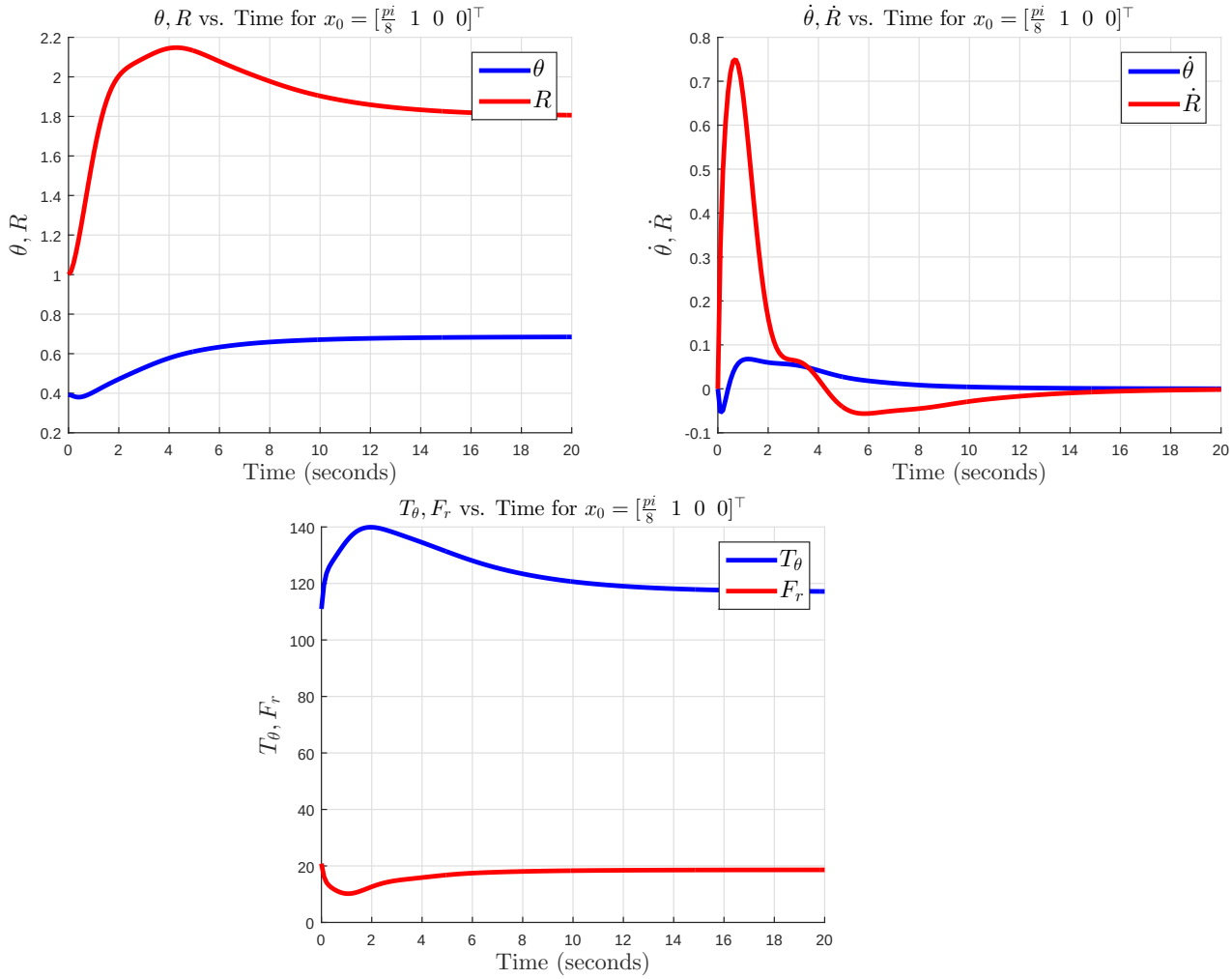

Figure 5: The above figures show the state-response with the estimated generated from the **unconstrained** MPC design for the nonlinear system (top two figures), as well as the optimal MPC control trajectories. Note the drastic difference in the optimal control trajectories for the MPC in comparison with the state-feedback control.

Problem 5 — Constrained MPC for the Nonlinear System

For this problem, we will resolve Problem 4 (with the exact same parameters, unless otherwise specified), but with constraints on one of the state variables and control inputs.

1. In this problem, you will us the Matlab function `quadprog` to solve the constrained MPC problem. First, teach yourself how `quadprog` works through any quadratic optimization example.

2. Assume that we require that the variable radius r (or $y = x_2$) to be bounded as follows:

$$1 \leq x \leq 2 \;\; \Rightarrow \;\; 1 \leq \Delta x_2 + x_{e_2} \leq 2 \Rightarrow \;\; 1 - x_{e_2} \leq \Delta x_2 = \Delta y \leq 2 - x_{e_2}.$$

   Start by formulating the bounded output problem in terms of ΔU, similar to the technique we used in class.

3. Also, we will assume that controls are both bounded as follows:

$$0 \leq U(k) \leq 200 \;\; \forall k,$$

   i.e., the torque and force can only be nonnegative quantities smaller or equal to 200 in magnitude.

4. What you have to change in this problem (from Problem 4) is the `optimizer` function (call the new function `optimizer_con.m`), as this function computes the optimal $\Delta U(k)$. In Problem 4, $U(k)$ and $Y(k)$ were both unconstrained. Using `quadprog`, update your `optimizer` function such that the bounds mentioned above are reflected in the MPC optimization. **You should be very careful when doing so, as we are operating on the equilibrium, linearized representation of the system.**

5. Your simulation file for the constrained MPC should have the following updated structure:

```
% All the constants
% All the matrices and initial conditions
% UU(:,1)=u0;
% for k=1:1:(Tfinal/h)
% tspan(:,k)=(k-1)*h:0.001:k*h;
% tt(k)=tspan(1,k);
% [t,X] = ode45(@thetaRdynamics,[(k-1)*h k*h],XX(:,k),options,ue+UU(:,k));
% Find x_a for the new iteration
% Find DU optimal for the new horizon using THE UPDATED CONSTRAINED OPTIMIZAR FILE
% Extract UU (or u(k))
% end
```

6. Plot your control trajectories and the corresponding states for the constrained MPC problem.

7. Change your initial conditions and R, Q, r values and explain the corresponding outputs and plots.

```
%% Problem 5-(1)--(6): Constrained MPC for the NL System for the given parameters

tol=1e-7;
XX=[];
YY=[];
UU=[];
n=4;
p=2;
% Initial Conditions
x0 = [pi/8;1;0;0]; % x(0)
y0 = C*x0;
u0=[0;0]; % u(0)
XX(:,1)=x0;
```

```
YY(:,1)=y0;
UU(:,1)=u0;
options=odeset('RelTol',tol);
h=0.101;
Np=40;
R = [0.1 0;0 0.1];
R = kron(eye(Np),R);
Q = [4 0; 0 0.1];
Q = kron(eye(Np),Q);
% The deviation is defined to be 0.1 for Theta and 0.2 for R.
r = [-0.1;-0.2];
r = repmat(r,Np,1);
tspan=[];
Tfinal=20;
tt=[];
% Constraints on the input 0 < U < 200
umin=0*ones(Np*p,1);
umax=200*ones(Np*p,1);
% Constraints on \Delta R , \Delta \Theta
% Since we want 1 < r < 2, hence, we want -1 < \Delta R = R - R_{eq} < 0
% This justifies the following bounds for the outputs (\theta and R).
ymin=repmat([-pi/2-pi/4 -1]',Np,1);
ymax=repmat([+pi/2-pi/4 0]',Np,1);

for k=1:1:(Tfinal/h)
tspan(:,k)=(k-1)*h:0.001:k*h;
tt(k)=tspan(1,k);
[t,X] = ode45(@thetaRdynamics_MPC,[(k-1)*h k*h],XX(:,k),options,ue+UU(:,k));
XX(:,k+1) = X(end,:)';
YY(:,k+1) = C*XX(:,k+1);
xa=[XX(:,k+1)-XX(:,k);YY(:,k+1)-ye];
[DU_optimal,GammaA,PhiA,W,Z,Qhat,H,E] = optimizer_con(xa,A,B,C,Np,h,UU(:,k)+ue,umin,umax,ymin,ymax,R,Q,r);
Du=[eye(p) zeros(2,2*(Np-1))]*DU_optimal;
UU(:,k+1)=UU(:,k)+Du;
end
t=[tt Tfinal];

% Plot the results for Pbm1-b, for different initial conditions
%
set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,XX(1,:),'b','LineWidth',3);
plot(t,XX(2,:),'r','LineWidth',3);
l=legend('$\theta$','$R$');
grid;
set(l,'interpreter','latex','fontsize',16)
ylabel('$\theta,R$','fontsize',16,'interpreter','latex');
xlabel('Time (seconds)','fontsize',15,'interpreter','latex');
title('$\theta,R$ vs. Time for $x_0 = [ \frac{\pi}{8} \hspace{0.2cm} 1  \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{\top}

set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,XX(3,:),'b','LineWidth',3);
plot(t,XX(4,:),'r','LineWidth',3);
grid;
l=legend('$\dot{\theta}$','$\dot{R}$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$\dot{\theta},\dot{R}$','fontsize',16,'interpreter','latex');
xlabel('Time (seconds)','fontsize',15,'interpreter','latex');
```

```
title('$\dot{\theta},\dot{R}$ vs. Time for $x_0 = [ \frac{\pi}{8} \hspace{0.2cm} 1 \hspace{0.2cm} 0 \hspace{0.2cm}


set(0,'defaulttextinterpreter','latex')
figure
hold on
plot(t,ue(1)+UU(1,:),'b','LineWidth',3);
plot(t,ue(2)+UU(2,:),'r','LineWidth',3);
grid;
l=legend('$T_{\theta}$','$F_r$');
set(l,'interpreter','latex','fontsize',16)
ylabel('$T_{\theta},F_r$','fontsize',16,'interpreter','latex');
xlabel('Time (seconds)','fontsize',15,'interpreter','latex')
title('$T_{\theta},F_r$ vs. Time for $x_0 = [  \frac{\pi}{8} \hspace{0.2cm} 1 \hspace{0.2cm} 0 \hspace{0.2cm}  0]^{

%%%%%%%%%%%%%%%%%

function [DU_optimal,GammaA,PhiA,W,Z,Qhat,H,E] = optimizer_con(xa,A,B,C,Np,h,u,umin,umax,ymin,ymax,R,Q,r)
[W,Z,Ca,PhiA,GammaA] = MPCMATs(A,B,C,Np,h);
[DU_optimal,Qhat,H,E]=MPCconopt(W,Z,Q,R,xa,r,Np,u,umin,umax,ymin,ymax);
end

%%%%%%%%%%%%%%%%%

function [dx]= thetaRdynamics_MPC(t,x,u)
% Theta-R dynamics didn't change
m1=10; m2=3; g=9.81; r1=1;
dx(1) = x(3);
dx(2) = x(4);
dx(3) = (-2*m2*x(2)*x(3)*x(4)-g*cos(x(1))*(m1*r1+m2*x(2))+u(1))/(m1*r1^2+m2*x(2)^2);
dx(4) = (x(3))^2*x(2)-g*sin(x(1))+u(2)/m2;

dx = dx';
end

%%%%%%%%%%%%%%%%%


function [DU_optimal,Qhat,H,E] = MPCconopt(W,Z,Q,R,xa,r,Np,u,umin,umax,ymin,ymax)

% min j(DU)=1/2 (r-Y)'*Q*(r-Y) + 1/2 (DU)'*R*(DU)
% min j(DU)=1/2 (r-Wxa-ZDU)'*Q(r-Wxa-ZDU) + 1/2 (DU)'*R*(DU);

% We need to write this equation in the quadratic form, after some
% manipulations, we arrive at this form:

% f(x) = j(x) = 1/2 x'Qx + x'b + c % the usual quadratic equation where:

% j(DU) = 1/2 DU'*Qhat*Du + DU'*bhat + c

Qhat=R+Z'*Q*Z;
Qhat=1/2*(Qhat+Qhat');
bhat=-Z'*Q*(r-W*xa);
c=0.5*(r-W*xa)'*Q*(r-W*xa);

% Now, formulate the constraints on the input first u(k):
% u_min <= u(k) <= u_max

% From the lecture notes, we arrive at this constraint:

% Subject To: U(k) = E u(k-1)+H DU ;
```

```
% Previous constraint can be written as:

% [-H;H] DU <= [-Umin+Eu(k-1);Umax-Eu(k-1)];

% Constructing H,E matrices
% H = [Im;Im;...;Im];
% E = [Im zeros; Im Im zeros; ...; Im Im Im ... Im];

m=2;
E=zeros(Np,m);
for i=1:Np
E(2*i-1:2*i,:)=eye(2);
end

for kk=1:1:Np-1
M(:,2*kk-1:2*kk)=[zeros(2*kk,2);E(1:end-2*kk,:)];
end
H=[E M];

% A       x <= b

% Hhat1 * DU <= e1;

Hhat1=[-H;H];
e1=[-umin+E*u;umax-E*u];

% Now, formulate the constraint on the raduis R (which is the second
% output) y(2):

%[-Z;Z] DU <= [-Ymin+Wxa;Ymax-Wxa];

% Hhat2 * DU <= e2;

Hhat2=[-Z;Z];

e2=[-ymin+W*xa;ymax-W*xa];

% Now, we have:
% Hhat1 * DU <= e1;
% Hhat2 * DU <= e2;

% Then:
% Hhat * Du <= e;

Hhat=[Hhat1;Hhat2];
e=[e1;e2];
DU_optimal = quadprog(Qhat,bhat,Hhat,e);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [W,Z,Ca,PhiA,GammaA] = MPCMATs(A,B,C,Np,h)
Phi=expm(A*h);
gB = @(rho) expm(A*rho)*B;
Gamma = integral(gB,0,h,'ArrayValued',true);
p=2;
n=4;
Ca=[zeros(p,n) eye(p)];
GammaA=[Gamma;C*Gamma];
PhiA=[Phi zeros(n,p);C*Phi eye(p)];
```

```
[q,w]=size(Ca);

W=zeros(q*Np,w);
Z=zeros(q*Np,q*Np);

for i=1:1:Np
W(2*i-1:2*i,:)=Ca*(PhiA^(i));
end


for ii=1:1:Np
Z(2*ii-1:2*ii,2*ii-1:2*ii)=Ca*GammaA;
end

M=[];

for k=1:1:Np
Z(2*k-1:2*k,1:2)=Ca*(PhiA^(k-1))*GammaA;
end

L=Z(:,1:2);

for kk=1:1:Np-1
M(:,2*kk-1:2*kk)=[zeros(2*kk,2);L(1:end-2*kk,:)];
end
Z=[L M];
end
```

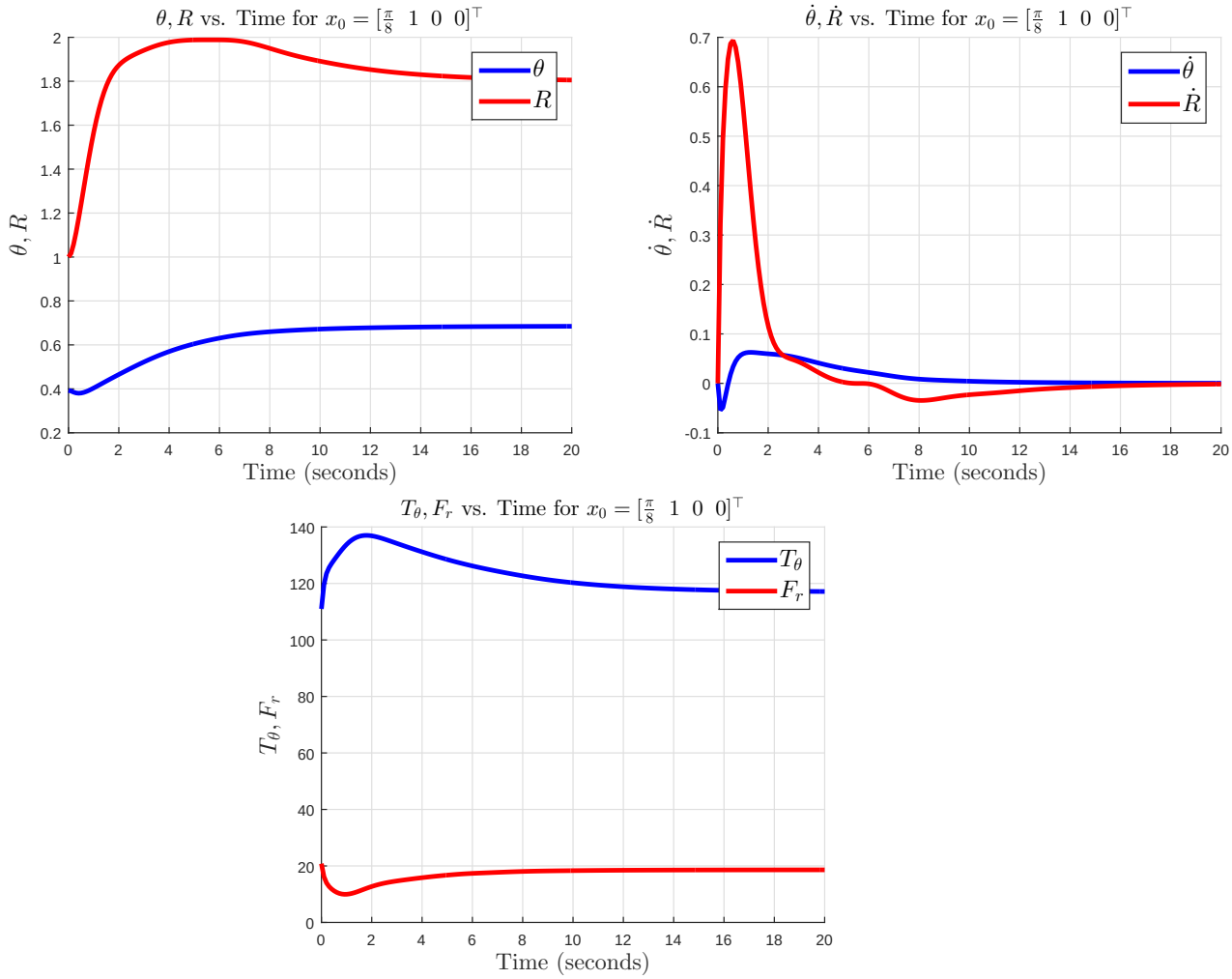

Figure 6: The above figures show the state-response with the estimated generated from the **constrained** MPC design for the nonlinear system (top two figures), as well as the optimal MPC control trajectories. Notice that R satisfies the given constraints, i.e., $1 < R < 2$, and the control input is bounded within the given ranges.

References

[1] S. H. Żak, *Systems and Control*. New York: Oxford University Press, 2003