1 Problem 1

Consider the following system:

$$\begin{aligned}\dot{x} &= \begin{bmatrix} 0 & 1 \\ -20 & -1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 20 & 0 \end{bmatrix} x(t)\end{aligned} \tag{1}$$

a) Design a model predictive controller (MPC) such that the overshoot of its step response is less than 15% and the settling time is less than two seconds. (Determine all parameters of your choice).

b) Design the controller again considering the following constraints:

$$0 \leqslant y(t) \leqslant 1.2, \qquad |u(t)| \leqslant 2, \qquad \left|\frac{du(t)}{dt}\right| \leqslant 3 \tag{2}$$

c) What is the effect of changes in the control parameters on the output? (Investigate all four parameters)

1.1 Solution Approach

I implemented MATLAB code to address both parts a and b, offering flexibility in evaluating different scenarios. The code structure comprises defining the system model, specifying MPC parameters, and incorporating constraints and weights as required for each part.

```
close all; clear; clc;

%% Define Plant Model
A = [0 1; -20 -1];
B = [0; 1];
C = [20 0];
D = 0;

% Create state-space representation of the system
LTI_system = ss(A, B, C, D);

%% Define MPC Parameters and Settings
Ts = 0.1;          % Sampling time
Np = 10;           % Prediction horizon
Nc = 3;            % Control horizon

%% Choose between part a or b
type = 'a'; % Change this to 'b' for part b

if type == 'a'
    % Define constraints and weights for part a
    inputConstraints = struct('Min', [], 'Max', [], 'RateMin', [], 'RateMax', []);
    outputConstraints = struct('Min', 0, 'Max', 1.15);

elseif type == 'b'
    % Define constraints and weights for part b
    inputConstraints = struct('Min', -2, 'Max', 2, 'RateMin', -3, 'RateMax', 3);
    outputConstraints = struct('Min', 0, 'Max', 1.2);

else
    error('Invalid type. Choose between ''a'' or ''b''.');
```

```
end

% Define weights for the cost function
R = 0.05;   % Manipulated variable weight
Q = 2;      % Output variable weight
S = 0.1;    % Manipulated variable rate weight

% Set up the weights structure
Weights = struct('ManipulatedVariables', R, 'ManipulatedVariablesRate', S, 'OutputVariables', Q);

%% MPC object
mpcobj = mpc(LTI_system, Ts, Np, Nc, Weights,inputConstraints, outputConstraints);

%% Perform Simulation
Reference = 1;                      % Reference value for the simulation
SimulationTime = 5;                 % Simulation time in seconds
Samples = SimulationTime / Ts;      % Number of simulation samples

% Simulate MPC controller
sim(mpcobj, Samples, Reference);
```

1.2 Results

1.2.1 Part a: Overshoot and Settling Time Analysis

The initial simulation for part a showcased satisfactory results in meeting overshoot and settling time criteria. Figures 1 and 2 demonstrate the control effort and system output, respectively.

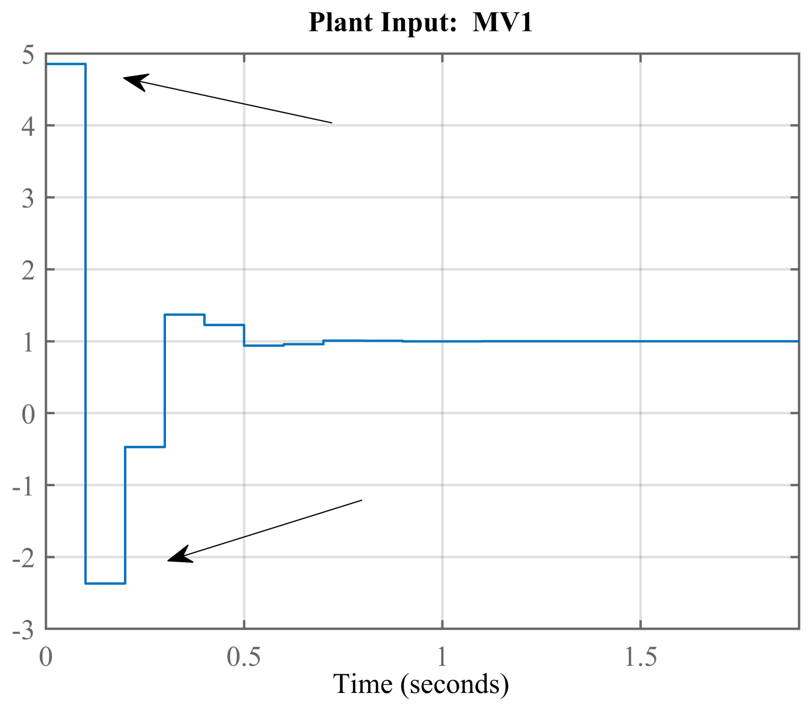

Figure 1: Control Effort - Part a: Ranging from -2.3 to 5

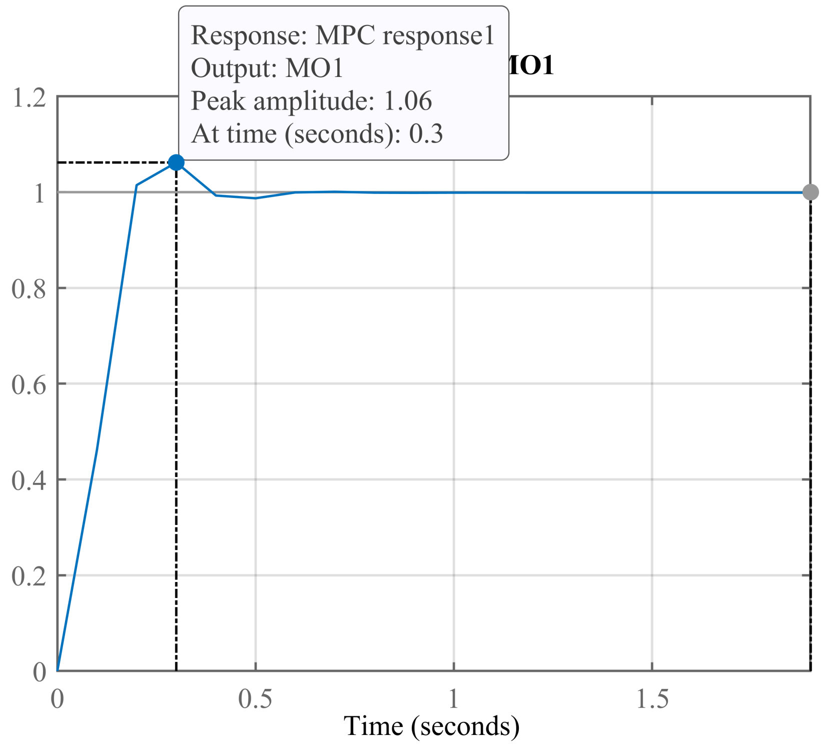

Figure 2: System Output - Part a

1.2.2 Part b: Incorporating Additional Constraints

Incorporating the new constraints altered the controller behavior, as depicted in Figures 3 and 4, showcasing control effort and system output for part b.

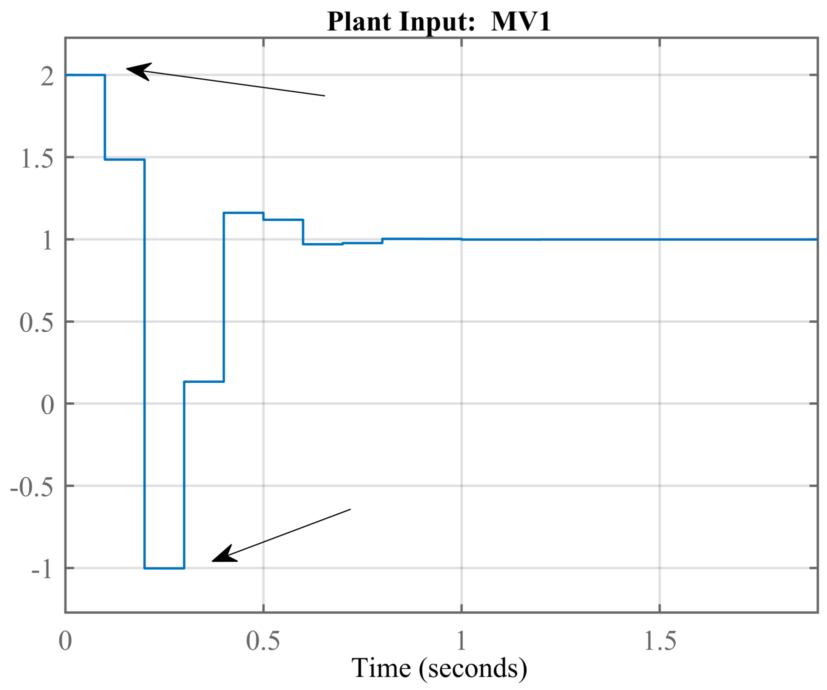

Figure 3: Control Effort - Part b: Ranging from -1 to 2

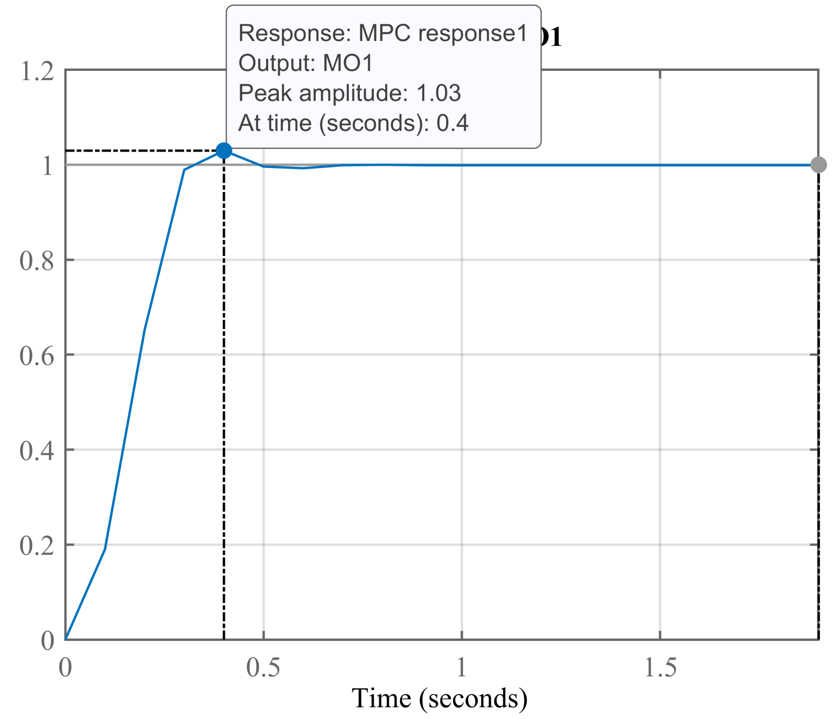

Figure 4: System Output - Part b

1.2.3 Parameter Variation Analysis

By systematically varying individual parameters while keeping others constant, their impact on system performance became evident. For instance, an increase in R constrained control signal magnitude, while increasing Q reduced steady-state error, as shown in Figure 5. The summarized effects of parameter changes are presented in Table 1.

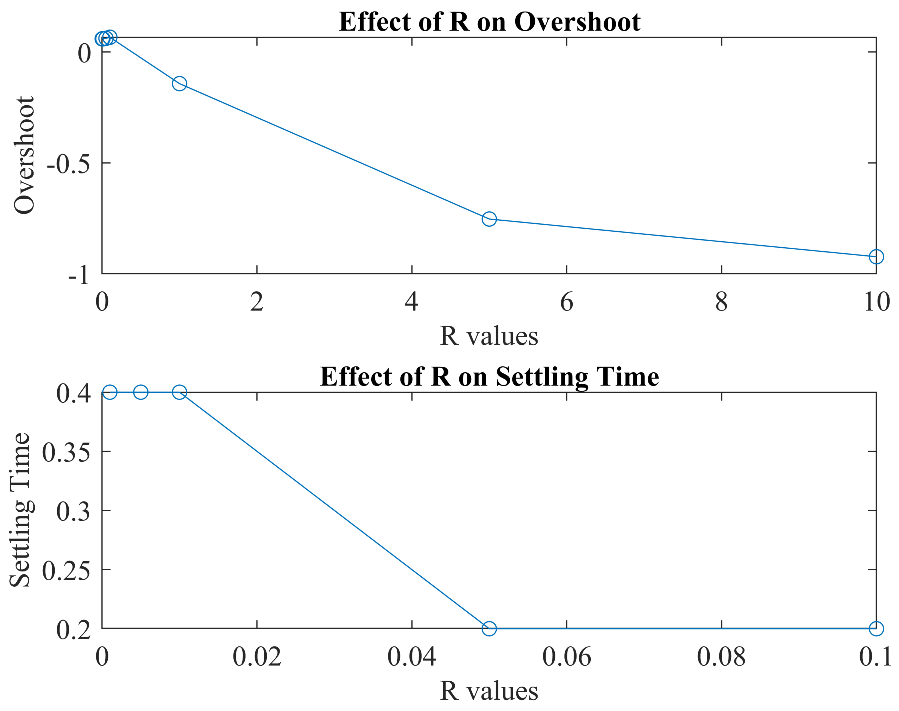

Figure 5: Effect of R on Overshoot and setteling time

finally we can summerize reults in the table below:

Parameter	**Overshoot**	**Settling Time**
Increase in Np	Decrease	Increase
Increase in Nc	Increase	Decrease
Increase in R	Decrease	Decrease
Increase in Q	Decrease	Increase

Table 1: Effects of Parameters on System Performance

2 Problem 2

Consider the following system:

$$\begin{aligned}\dot{x} &= \begin{bmatrix}5 & 0\\ 0 & -3\end{bmatrix} x(t) + \begin{bmatrix}1 & 0\\ 4 & 0\end{bmatrix} u(t)\\ y(t) &= \begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix} x(t)\end{aligned} \tag{3}$$

a) Design an unconstrained model predictive controller (MPC) without terminal cost and without terminal constraints for the above system.

b) Design a constrained MPC without terminal cost and without terminal constraints. (considering appropriate constraints)

c) Design a constrained MPC with terminal constraints for the system.

2.0.1 Part a:

To design an unconstrained model predictive controller (MPC) without terminal cost and constraints, the first step was to analyze the controllability and observability of the given system. A MATLAB function was written to compute the controllability and observability matrices. The results showed that the system is fully controllable and observable, which is a prerequisite for implementing MPC.

```
function [sys, controllable, uncontrollableStates, observable, unobservableStates] =
    Controllability_Observability_Analysis(sys)

controllabilityMatrix = ctrb(sys);
observabilityMatrix = obsv(sys);

uncontrollableStates = length(sys.A) - rank(controllabilityMatrix);
unobservableStates = length(sys.A) - rank(observabilityMatrix);

controllable = uncontrollableStates == 0;
observable = unobservableStates == 0;

if ~controllable
    disp(['System has ', num2str(uncontrollableStates), ' uncontrollable state(s).']);
else
    disp('System is fully controllable.');
end

if ~observable
    disp(['System has ', num2str(unobservableStates), ' unobservable state(s).']);
else
    disp('System is fully observable.');
end
end
```

The results are printed as shown below in command window:

```
System is fully controllable.
System is fully observable.
>>
```

The subsequent step involved a trial-and-error approach to optimize the MPC parameters. Different values for Np, Nc, R, Q, and S were considered to find the best combination. MATLAB code was used to iterate through various parameter settings and simulate the MPC control for each configuration. Metrics like overshoot and control effort were calculated and compared to identify the optimal parameter combination.

```
Ts = 0.01;
Np = [5, 10, 15, 20];
Nc = [5, 10, 15, 20];
R = {0.05 * eye(2), 0.1 * eye(2)};
Q = {1 * eye(2), 5 * eye(2), 10 * eye(2)};
S = {0.1 * eye(2), 0.5 * eye(2), 1 * eye(2)};
```

```
function best_combination = findBestParameters(sys, Ts, Np_values, Nc_values,
    R_values, Q_values, S_values)
% Initialization
best_overshoot = Inf;
best_control_effort = Inf;
best_combination = struct('Np', 0, 'Nc', 0, 'R', [], 'Q', [], 'S', []);

for i = 1:length(Np_values)
for j = 1:length(Nc_values)
for k = 1:length(R_values)
for l = 1:length(Q_values)
for m = 1:length(S_values)

Np = Np_values(i);
Nc = Nc_values(j);
R = R_values{k};
Q = Q_values{l};
S = S_values{m};

Weights = struct('ManipulatedVariables', R, 'ManipulatedVariablesRate', S, '
    OutputVariables', Q);

mpcobj = mpc(sys, Ts, Np, Nc, Weights);

Reference = ones(2);
SimulationTime = 5;
Samples = SimulationTime / Ts;

[~, ~, ~, ~, a, ~] = sim(mpcobj, Samples, Reference);
Plant_Inputs = a.LastMove;
Plant_OutPut = a.Plant;

% Compute performance metrics
overshoot_metric = max(max(abs(Plant_OutPut - ones(1, 2))));
control_effort_metric = sum(sum(abs(Plant_Inputs)));

% Compare metrics to find the best combination
if overshoot_metric < best_overshoot && control_effort_metric <
    best_control_effort
best_overshoot = overshoot_metric;
best_control_effort = control_effort_metric;
```

```
best_combination.Np = Np;
best_combination.Nc = Nc;
best_combination.R = R;
best_combination.Q = Q;
best_combination.S = S;
end end end end end end end
```

After exhaustive evaluation, the ideal settings were determined as follows: $Np = 20, Nc = 15, R = 0.05I_2, Q = 10I_2, S = 0.1I_2$

Implementing these parameters into the system, the control efforts and outputs were graphically depicted in Figures 6 and 7. These visualizations provided a clear representation of the system's response under the MPC control, showcasing its effectiveness in managing the given dynamics.

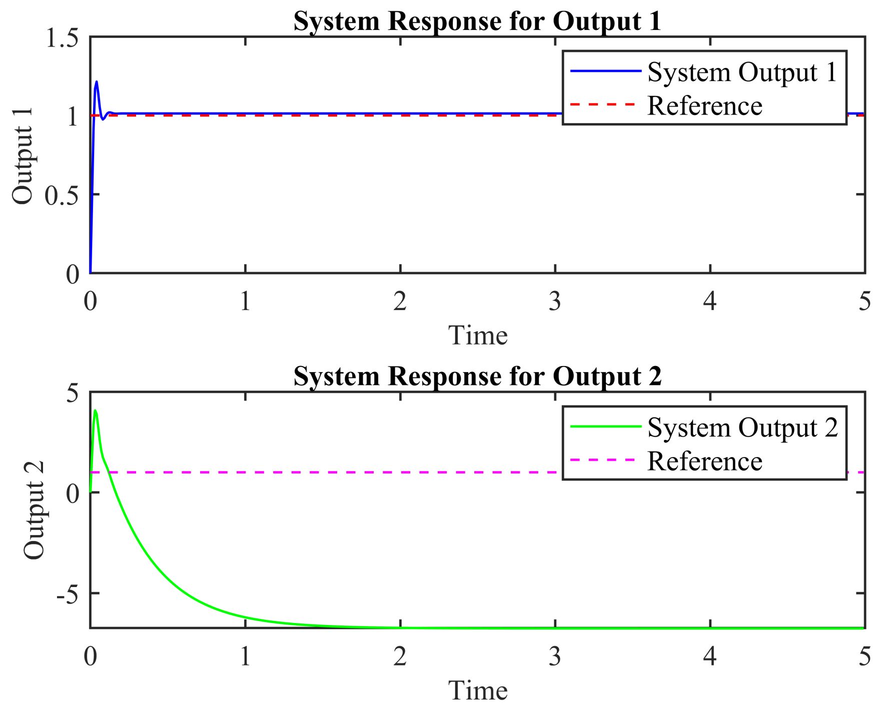

Figure 6: system output

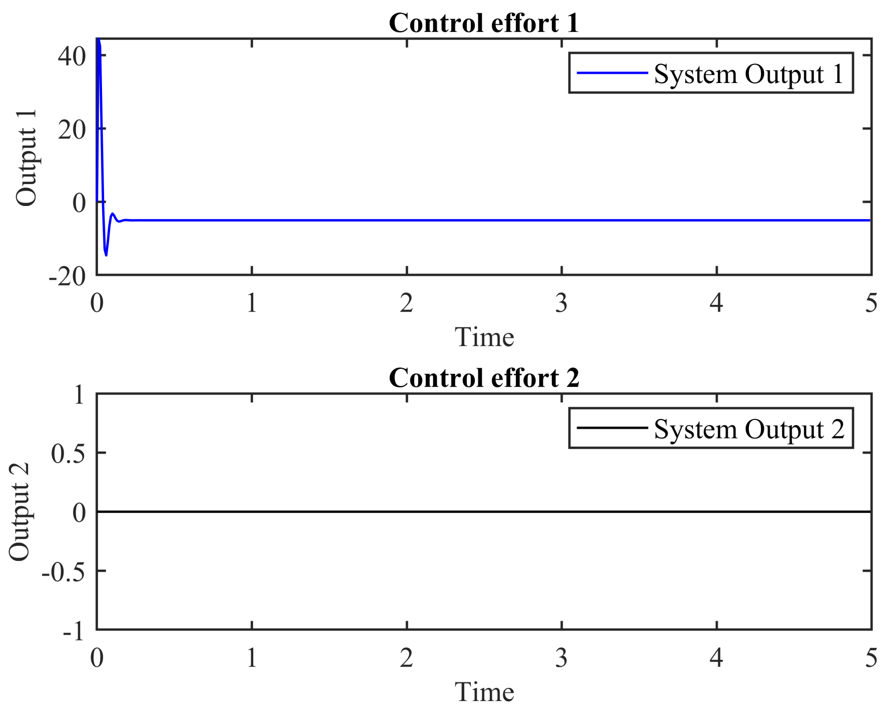

Figure 7: control effort

2.0.2 Part b:

In this section, I analyze three distinct sets of constraints to demonstrate their impact on the design of MPC, categorized as: **easy**, **moderate**, and **hard** constraints.

```
close all; clear; clc;

%% Define Plant Model
% Define system matrices
A = [5 0; 0 -3];
B = [1 0; 4 0];
C = [1 0; 0 1];
D = 0;

% Create state-space representation of the system
sys = ss(A, B, C, D);

%% Define MPC Parameters and Settings
Ts = 0.01;          % Sampling time
Np = 20;            % Prediction horizon
Nc = 15;            % Control horizon

% Define output constraints
outputConstraints = struct('Min', 0, 'Max', 1.2);

% Define weights for the cost function
R = 0.05*eye(2);    % Manipulated variable weight
```

```
Q = 10*eye(2);      % Output variable weight
S = 0.1*eye(2);     % Manipulated variable rate weight

% Set up the weights structure
Weights = struct('ManipulatedVariables', R, 'ManipulatedVariablesRate', S, '
    OutputVariables', Q);

%% Define Different Sets of Input Constraints
% Define "Easy" input constraints
EasyInputConstraints(1) = struct('Min', -10, 'Max', 10, 'RateMin', -20, '
    RateMax', 20);
EasyInputConstraints(2) = struct('Min', -20, 'Max', 20, 'RateMin', -15, '
    RateMax', 15);
EasyOutputConstraints = struct('Min', 0, 'Max', 5);

% Define "Moderate" input constraints
ModerateInputConstraints(1) = struct('Min', -6, 'Max', 6, 'RateMin', -9, '
    RateMax', 9);
ModerateInputConstraints(2) = struct('Min', -8, 'Max', 8, 'RateMin', -10, '
    RateMax', 10);
ModerateOutputConstraints = struct('Min', 0, 'Max', 1.5);

% Define "Hard" input constraints
HardInputConstraints(1) = struct('Min', -1, 'Max', 1, 'RateMin', -2, 'RateMax'
    , 2);
HardInputConstraints(2) = struct('Min', -0.5, 'Max', 0.5, 'RateMin', -1, '
    RateMax', 1);
HardOutputConstraints = struct('Min', 0, 'Max', 1.2);

%% Create MPC objects for each constraint set
mpcobjEasy = mpc(sys, Ts, Np, Nc, Weights, EasyInputConstraints,
    EasyOutputConstraints);
mpcobjModerate = mpc(sys, Ts, Np, Nc, Weights, ModerateInputConstraints,
    ModerateOutputConstraints);
mpcobjHard = mpc(sys, Ts, Np, Nc, Weights, HardInputConstraints,
    HardOutputConstraints);

%% Perform Simulations for each MPC object
Reference = ones(2);                    % Reference value for the simulation
SimulationTime = 10;                   % Simulation time in seconds
Samples = SimulationTime / Ts;          % Number of simulation samples

% Simulate MPC controllers for each constraint set
%% Perform Simulations for each MPC object
[~, t1, ~, ~, aEasy, ~] = sim(mpcobjEasy, Samples, Reference);
[~, t2, ~, ~, aModerate, ~] = sim(mpcobjModerate, Samples, Reference);
[~, t3, ~, ~, aHard, ~] = sim(mpcobjHard, Samples, Reference);


%% Plot Results for Plant Output Variable 1
figure;

subplot(3,1,1);
plot(t1, aEasy.Plant(:, 1),'k');
```

```
title('Easy Constraints - Output');
hold on
plot(t1, aEasy.Plant(:, 2),'m--');


subplot(3,1,2);
plot(t2, aModerate.Plant(:, 1),'k');
title('Moderate Constraints - Output');
hold on
plot(t2, aModerate.Plant(:, 2),'m--');

subplot(3,1,3);
plot(t3, aHard.Plant(:, 1),'k');
title('Hard Constraints - Output');
hold on
plot(t3, aHard.Plant(:, 2),'m--');
%%
figure;
subplot(3,1,1);
plot(t1, aEasy.LastMove(:, 1),'r');
hold on;
plot(t1, aEasy.LastMove(:, 2),'b');
title('Easy Constraints - Control effort');

subplot(3,1,2);
plot(t2, aModerate.LastMove(:, 1),'r');
hold on
plot(t2, aModerate.LastMove(:, 2),'b');

title('Moderate Constraints - Control effort');

subplot(3,1,3);
plot(t3, aHard.LastMove(:, 1),'r');
hold on;
plot(t3, aHard.LastMove(:, 2),'b');
title('Hard Constraints - Control effort');
```

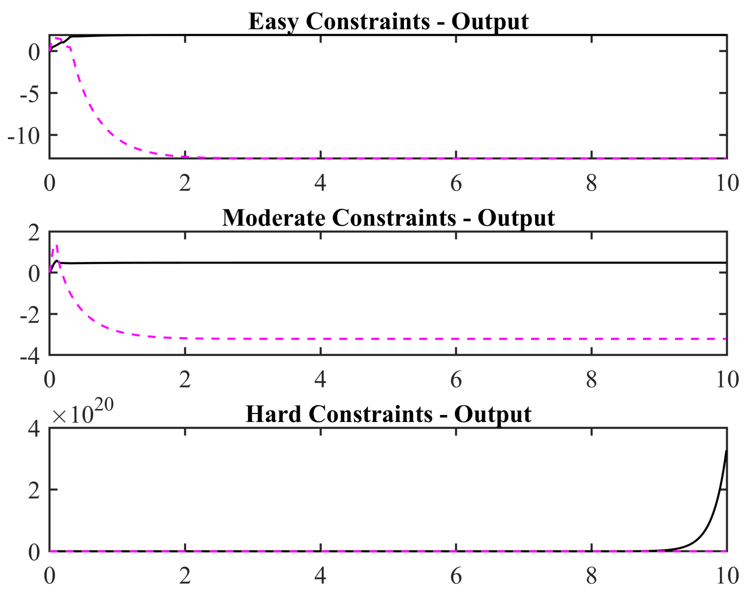

Figure 8: system outputs

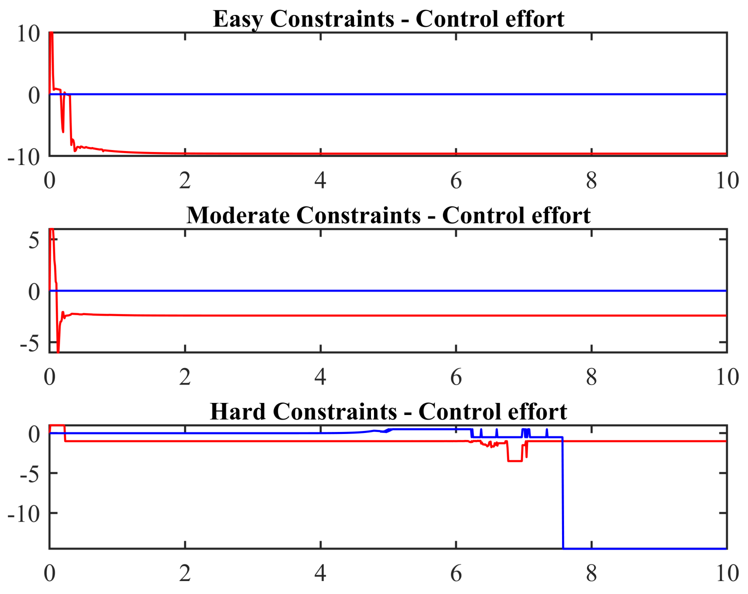

Figure 9: control efforts

We observed that when constraints are excessively stringent or challenging, the controller fails to meet them while maintaining system stability.

2.0.3 Part C:

The terminal constraint can be applied in the MPC controller as follows

```
close all; clear; clc;
%% Define Plant Model
A = [5 0; 0 -3];
B = [1 0; 4 0];
C = [1 0; 0 1];
D = 0;

sys = ss(A, B, C, D);
%% Define MPC Parameters and Settings
Ts = 0.01;        % Sampling time
Np = 20;          % Prediction horizon
Nc = 15;          % Control horizon

% Define weight
R = 0.05*eye(2);   % Manipulated variable weight
Q = 10*eye(2);     % Output variable weight
S = 0.1*eye(2);    % Manipulated variable rate weight

% Set up the weights structure
Weights = struct('ManipulatedVariables', R, 'ManipulatedVariablesRate', S, '
    OutputVariables', Q);

% Create MPC object
mpcobj = mpc(sys, Ts, Np, Nc, Weights);
Y = struct('Min',[0.9,0.9],'Max',[1.3,1.3]);
U = struct('Min',[-5,-inf]);

%% Perform Simulation
Reference = ones(2);                    % Reference value for the simulation
SimulationTime = 5;              % Simulation time in seconds
Samples = SimulationTime / Ts;   % Number of simulation samples

% Simulate MPC controller
sim(mpcobj, Samples, Reference);
```

Simulation results:

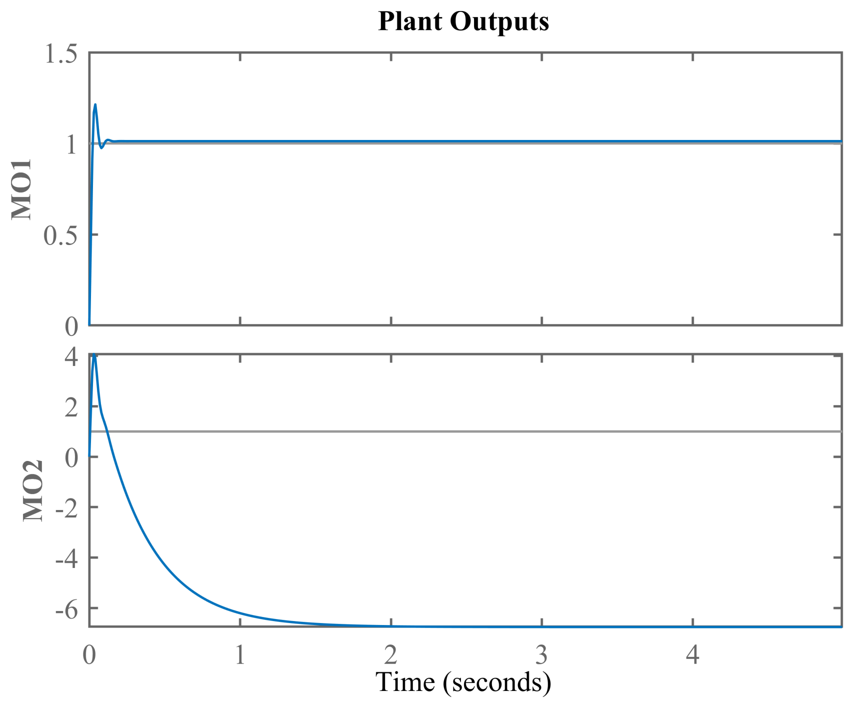

Figure 10: system outputs

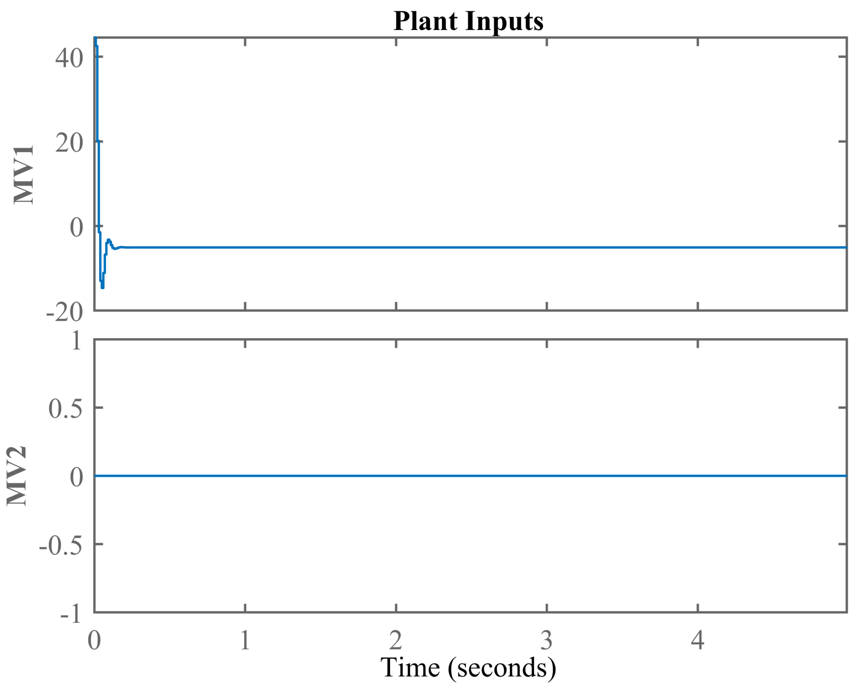

Figure 11: control efforts

3 Problem 3

Consider the following system:

$$\begin{aligned}\dot{x} &= \begin{bmatrix} 0 & 1 \\ 32 & -2.2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 32 & 0 \end{bmatrix} x(t)\end{aligned} \tag{4}$$

a) Taking $R = 0.1$ and $Q = 2$, find the terminal cost and control law such that stability of the system is guaranteed.

b) Design a MPC that can track a sinusoidal reference input.

c) If the system has a constant disturbance of magnitude 0.1, show the step response of the closed-loop system.

3.1 Solve

3.1.1 Part a:

MPC controller with infinite horizon is equivalent to LQR optimal controller. The above system They have two special values as follows: eig(A)= 4.6628, -6.8628 The above system has an unstable mode at s=-6.86.

First, we define the system parameters using the code below, and then use the lqry command State feedback, K, terminal penalty weight function and new special values of the system Controlled A-BK, denoted by en, results in:

```
% Define system matrices
A = [0 1; 32 -2.2];
B = [0; 1];
C = [32 0];
D = 0;

% Create state-space system
sys = ss(A, B, C, D);

% Set Q and R values for LQR controller
Q = 2;
R = 0.1;

% Design LQR controller
[K, S, e] = lqry(sys, Q, R);

% Solve the system using ODE45
[t, x] = ode45(@fun, [0 8], [0.3, 0.5]);

% Plot state variables x1 and x2
figure(1);
plot(t, x(:, 1), 'b', 'linewidth', 1);
hold on;
plot(t, x(:, 2), 'r', 'linewidth', 1);
```

```
legend('x_1', 'x_2');
xlabel('Time');
ylabel('State Variables');
title('State Variables x_1 and x_2');

% Calculate and plot control signal u
u = -K * x';
figure(2);
plot(t, u, 'k', 'linewidth', 1);
xlabel('Time');
ylabel('Control Signal u');
title('Control Signal');

% Calculate and plot output y
y = C * x';
figure(3);
plot(t, y, 'k', 'linewidth', 1);
xlabel('Time');
ylabel('Output y');
title('Output');


function dx= fun(t,x)
        A = [0 1; 32 -2.2];
        B = [0;1];
        K = [178.6424    16.8296];

        dx=A*x+B*(-K*x);
end
```

```
K =
  178.6424   16.8296

S =
  286.0943   17.8642
   17.8642    1.6830

e =
  -9.5148 + 7.4907i
  -9.5148 - 7.4907i
```

As you can see, the special values of the controlled system with the alternative controller The LQR mode feedbacks are all negative, so the system will be stable (until this part is answered is enough). Now the system can be controlled with the state feedback control law obtained from the LQR method

The response close-loop system is as follows, the output signal is stable and state and control and output are limited and output regulated to zero:

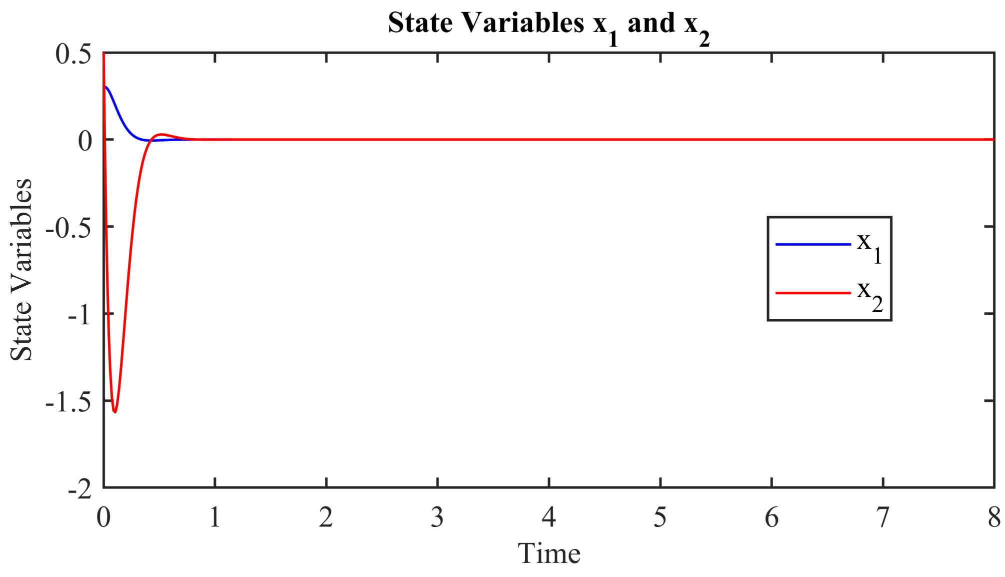

Figure 12: system state trajectory

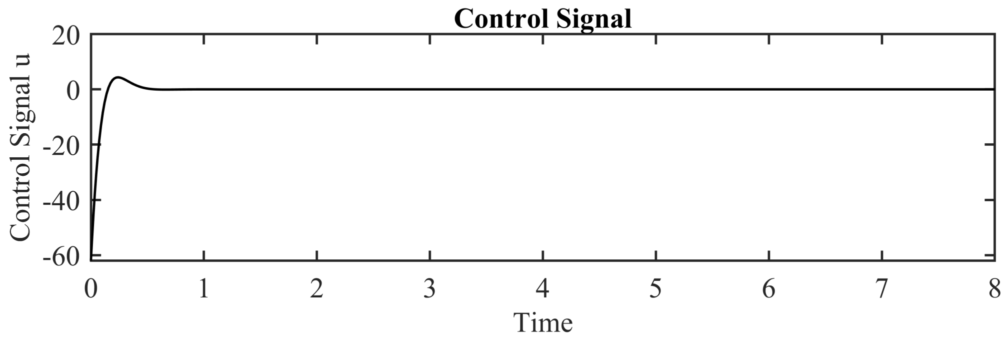

Figure 13: control effort

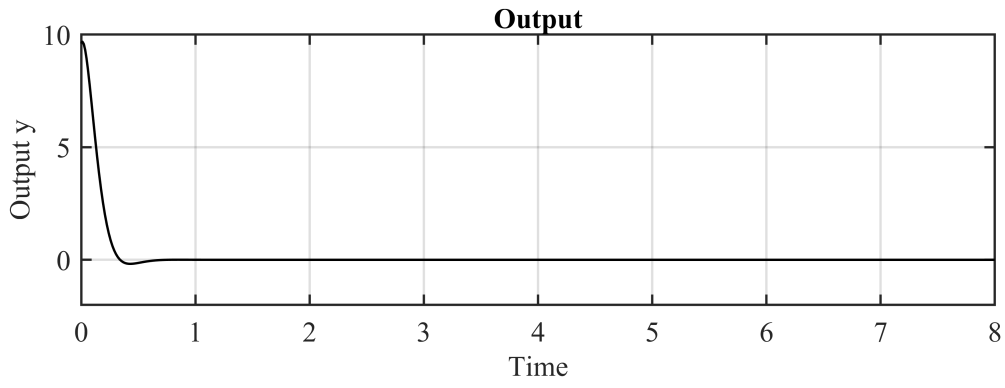

Figure 14: Output

3.1.2 Part b:

In this part and the next part I used the Simulink to design MPC. and every thing had saved to check in detail if it is necessary.

firstly i define system in MATLAB:

```
% Define system matrices
A = [0 1; 32 -2.2];
B = [0; 1];
C = [32 0];
D = 0;

% Create state-space system
SYS = ss(A, B, C, D);

% Create transfer-function system
transfer_function=tf(SYS);

mpcobject=mpc(SYS,0.01);
sim('Q3_b.slx')
```

then by some setting in MPC toolbox, we can obtain the following output from system that's show a good tracking of sinisy input singal.

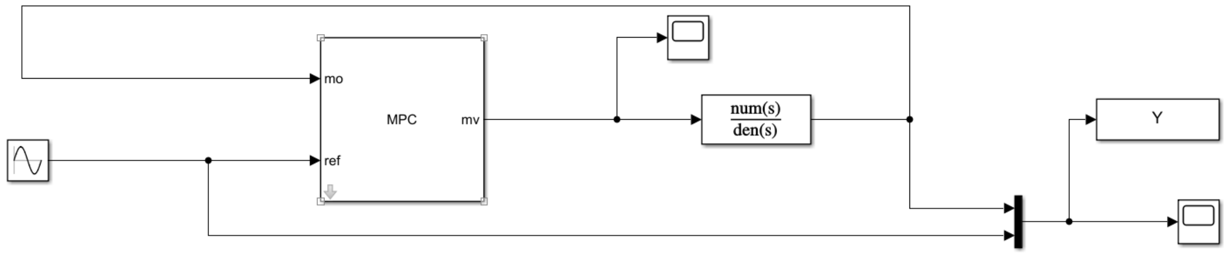

Figure 15: Simulink

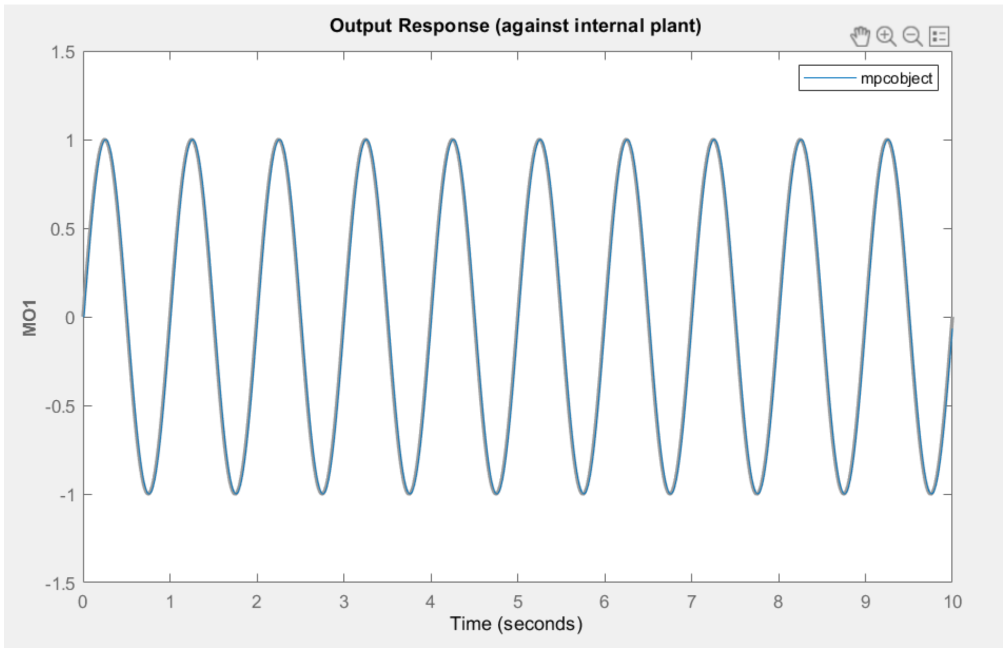

Figure 16: Simulink

3.1.3 Part C:

In this segment and the subsequent one, I utilized Simulink to implement MPC. All relevant data was saved for detailed analysis when necessary.

Initially, I defined the system in MATLAB using the provided code to establish the state-space and transfer-function systems. Then, leveraging the MPC toolbox, I created an MPC object and ran simulations using a sine input signal, showcasing the system's commendable tracking.

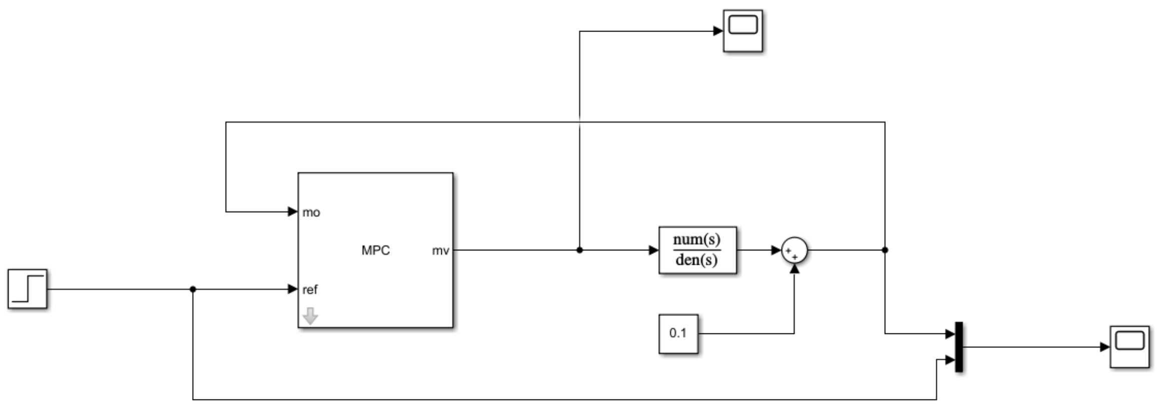

Figure 17: Simulink

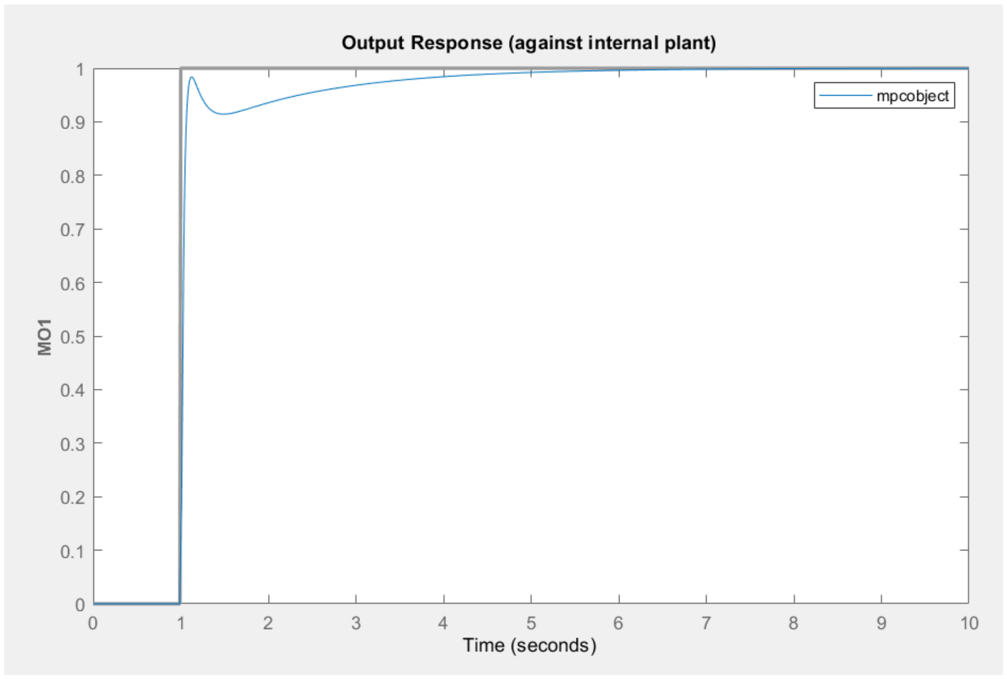

Figure 18: Simulink