SSY280 Model predictive control

Assignment 1 Inverted Pendlum

Group 20

Dandan Ge
Fabian Melvås

January 2017

1 Introduction

This report is about to balance an inverted pendulum using Model Predictive Control.

2 Pendulum model

$$x(k+1) = Ax(k) + Bu(k) = \begin{bmatrix} 1 & h \\ \alpha h & 1 \end{bmatrix} x(k) + \begin{bmatrix} \beta h^2/2 \\ \beta h \end{bmatrix} u(k) \tag{1a}$$

$$y(k) = Cx(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) \tag{1b}$$

where $u(k) = \tau(k)$, $x_1 = \theta$ and $x_2 = \dot{\theta}$, θ is the angle relative to the vertical direction.

3 Control design

The control of the pendulum will be based on the minimization of a quadratic cost function with a horizon N, at each time instant k.

$$\min_{u(k:k+N-1)} \quad q \left\| x(k+N) \right\|^2 + \sum_{i=0}^{N-1} (qy^2(k+i) + ru^2(k+i)) \tag{2a}$$

$$\text{s.t.} \quad x(k+i+1) = Ax(k+i) + Bu(k+i), \quad i = 0, 1, ..., N-1 \tag{2b}$$

4 Tasks

4.1 Without actuator constraints

(a) Rewrite the optimization problem on:

$$\min_z \frac{1}{2} z^T H z \tag{3a}$$

$$\text{s.t.} \quad A_{eq} z = b_{eq} \tag{3b}$$

Given that

$$z = \begin{bmatrix} x^T(k+1) & x^T(k+2) & \cdots & x^T(k+N) & u(k) & \cdots & u(k+N-1) \end{bmatrix}^T$$

Our cost function has the form

$$V_N(x(k), u(k:k+N-1)) = \sum_{k=0}^{N-1} (x^T(k) Q x(k) + u^T(k) R u(k) + x^T(N) P x(N))$$

From the problem we get:

$$Q = qC^TC = \begin{bmatrix} q & 0 \\ 0 & 0 \end{bmatrix} \quad P = \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix} \quad R = r \quad \Rightarrow \quad H = 2 \underbrace{\begin{bmatrix} Q & 0 & & \cdots & & 0 \\ 0 & \ddots & & & & \\ & & Q & \ddots & & \vdots \\ \vdots & & & P & & \\ & & & & R & \\ & & & & \ddots & 0 \\ 0 & & & \cdots & 0 & R \end{bmatrix}}_{\text{Dimension of matrice } (3\cdot N \times 3\cdot N)}$$

Rewrite the problem to $A_{eq}z = b_{eq}$ gives:

$$\underbrace{\begin{bmatrix} -I_2 & 0 & \cdots & B & 0 & \cdots & 0 & 0 & 0 & 0 \\ A & -I_2 & 0 & \cdots & B & 0 & \cdots & 0 & 0 & 0 \\ 0 & A & -I_2 & 0 & \cdots & B & 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & & & & \ddots & & & \\ & \ddots & & & & & & & & \\ & & & & & & & & & \\ 0 & & \cdots & & 0 & A & -I_2 & & \cdots & B \end{bmatrix}}_{A_{eq} \text{ Dimension of matrice } (2\cdot N \times 3\cdot N)} \underbrace{\begin{bmatrix} x(k) \\ \\ \vdots \\ \\ x(k+N) \\ u(k) \\ \vdots \\ u(k+N-1) \end{bmatrix}}_{z \ (3\cdot N \times 1)} = \underbrace{\begin{bmatrix} -A \\ 0 \\ \\ \vdots \\ \\ 0 \end{bmatrix} x(k)}_{b_{eq} \ (2\cdot N \times 1)}$$

I_2 is a 2×2 identity matrix

(b)

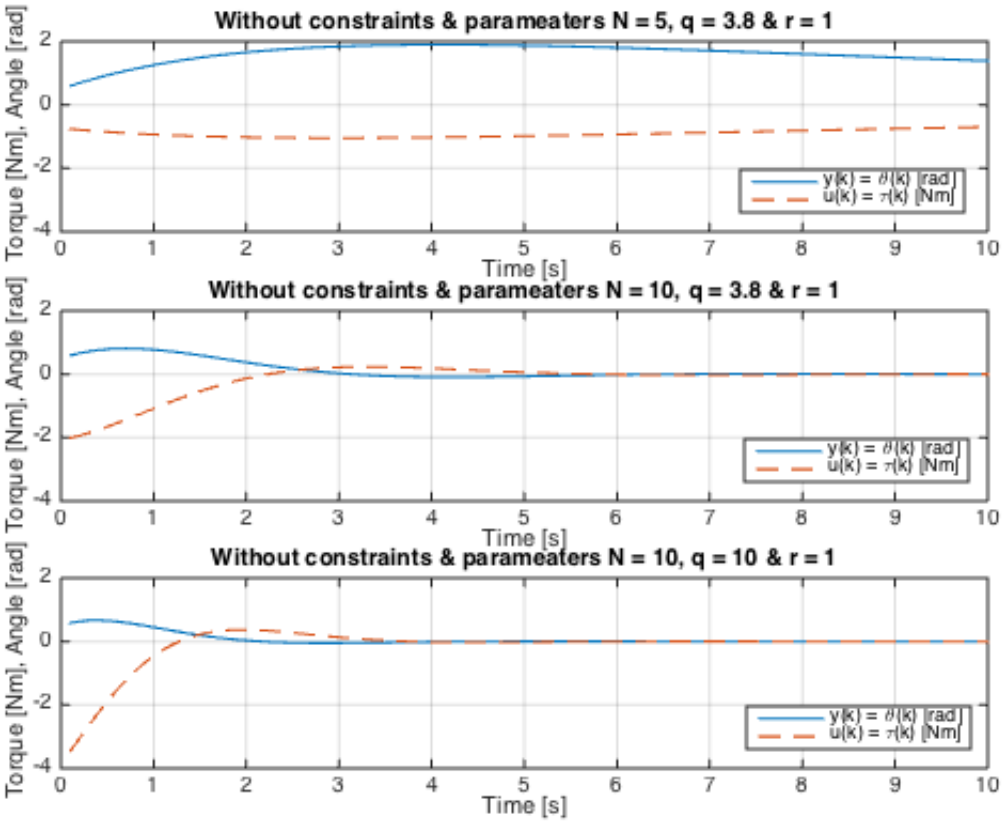

Figure 1: Resulting input and output without constraints for different parameters.

When comparing the two upper graphs in Figure 1 the difference is that the first one with prediction horizon N=5 does not reach the settling point, while the other one with N=10 does.The reason for this is that the second one uses a longer prediction horizon for each optimization, and can therefore see that a higher negative input will affect the output to first increase and then decrease to the settling point. While the first one can not see that fare and therefore is more careful with the high input.
The difference between the second and the third plot in Figure 1 is that the third has a higher state weighting coefficient q, state weighting coefficient reflecting the relative importance of x, a bigger weighting coefficient for the states gives less importance of the control input. This affect the optimization so that the system can increase the control input without increasing the cost. Then the controller can

have a stronger action to drive the output to the settling point, thus increases the converge rate.

In conclusion:
The longer the receding horizon, the faster the convergence speed.
The larger weighting coefficient of the states and the inputs, the better convergence performance. This is because when solving the optimization problem, larger weighting coefficient means less importance of the input signal, which means the cost energy of the control action is cheap,then the controller can take a stronger action to drive the output to the settling point.

(c)

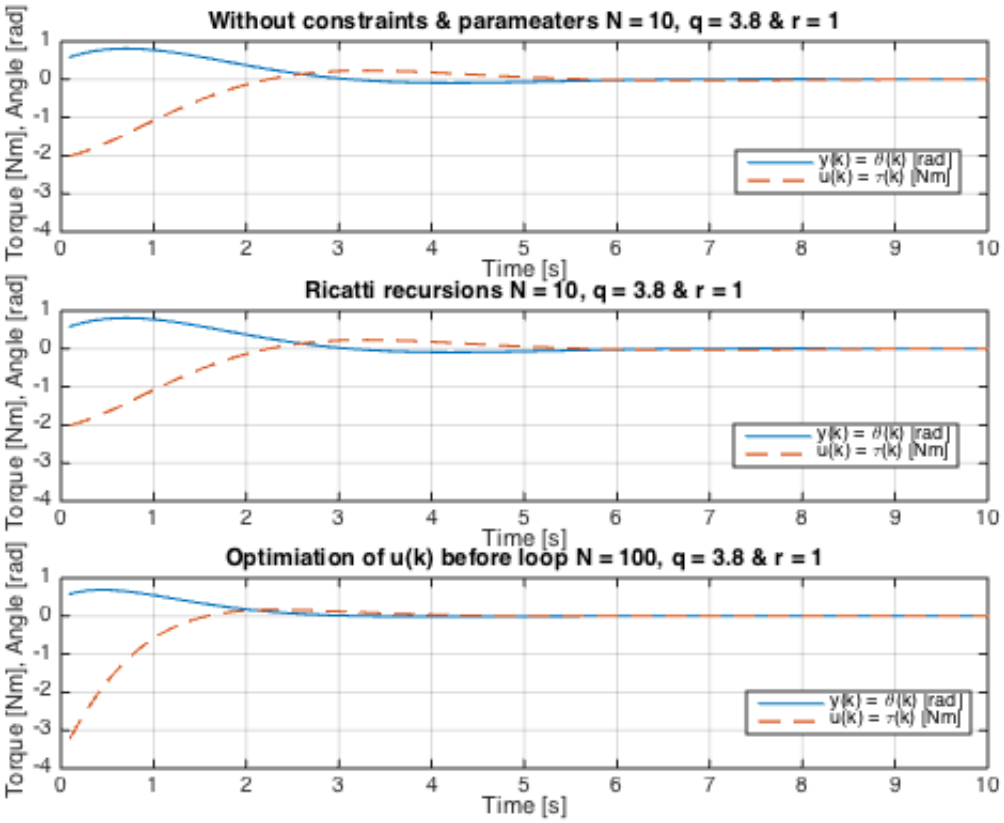

Figure 2: Comparision of different methods to compute controllers.

In Figure 2 for the first and second graph, it can be seen that the same result ap-

pear with the same prediction horizon N. This is obviously since the only difference between the graphs is the method to calculate the same optimization problem, the first use the MATLAB function quadprog and the second use Ricatti recursions. From the third graph, we only calculate the optimization problem once, there is no feedback compare to the other two cases, the system has a faster convergence performance and starts with a higher input.

Solving a finite time LQ problem over [0 N] using Riccati recursions, the solution to the LQ problem is in the form of a time-varying feedback control law. The solution can be precomputed based on the problem input data, but because of the feedback nature of the solution, the actual actions taken will make use of the updated state information from the controlled system. Sequence of optimal control laws:

$$u^0(k;x) = K(k)x, \quad k = 0, \ ... \ , N-1$$

$$K(k) = -(R + B^T P(k+1)B)^{-1} B^T P(k+1)A$$

Riccati equation:

$$P(k-1) = Q + A^T P(k)A - A^T P(k)B(R + B^T P(k)B)^{-1} B^T P(k+1)A, P(N) = P_f$$

Solving a finite time LQ control problem over [0 T] (T is the simulation time). In this case a simulation of the control problem is computed for T steps ahead directly and get a sequence of controllers, there is no feedback. It is a good method if there is no disturbances in the process, once there is any disturbance during the simulation, the output we get will not be accurate as we expected. But for the first two methods, since for every sampling instant, there is a state feedback into a new step, even though there is any disturbances, the result will always tend towards to the right direction.

4.2 With actuator constraints

Based on the previous unconstraint case, here with a constrain to control input:

$$-1 \leqslant u(k+i) \leqslant 1, \qquad i = 0, 1, ..., N-1 \tag{4}$$

(d) Rewrite the optimization problem on the form:

$$\min_{z} \frac{1}{2} z^T H z \tag{5a}$$

$$\text{s.t.} \quad A_{eq} z = b_{eq} \tag{5b}$$

$$A_{in} z \leq b_{in} \tag{5c}$$

Rewrite the optimization problem and get A_{in} and b_{in}

$$\underbrace{\begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \ddots & \ddots & & \vdots \\ 0 & & \cdots & & 0 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \\ 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \ddots & \ddots & & \vdots \\ 0 & & & & & -1 & 0 \\ 0 & & \cdots & & & 0 & -1 \end{bmatrix}}_{A_{in}\ (2\cdot N \times 3\cdot N)} \underbrace{\begin{bmatrix} x(k) \\ \\ \vdots \\ \\ x(k+N) \\ u(k) \\ \vdots \\ \\ u(k+N-1) \end{bmatrix}}_{z\ (3\cdot N \times 1)} \leq \underbrace{\begin{bmatrix} 1 \\ 1 \\ \\ \\ \vdots \\ \\ 1 \end{bmatrix}}_{b_{in}\ (2\cdot N \times 1)}$$

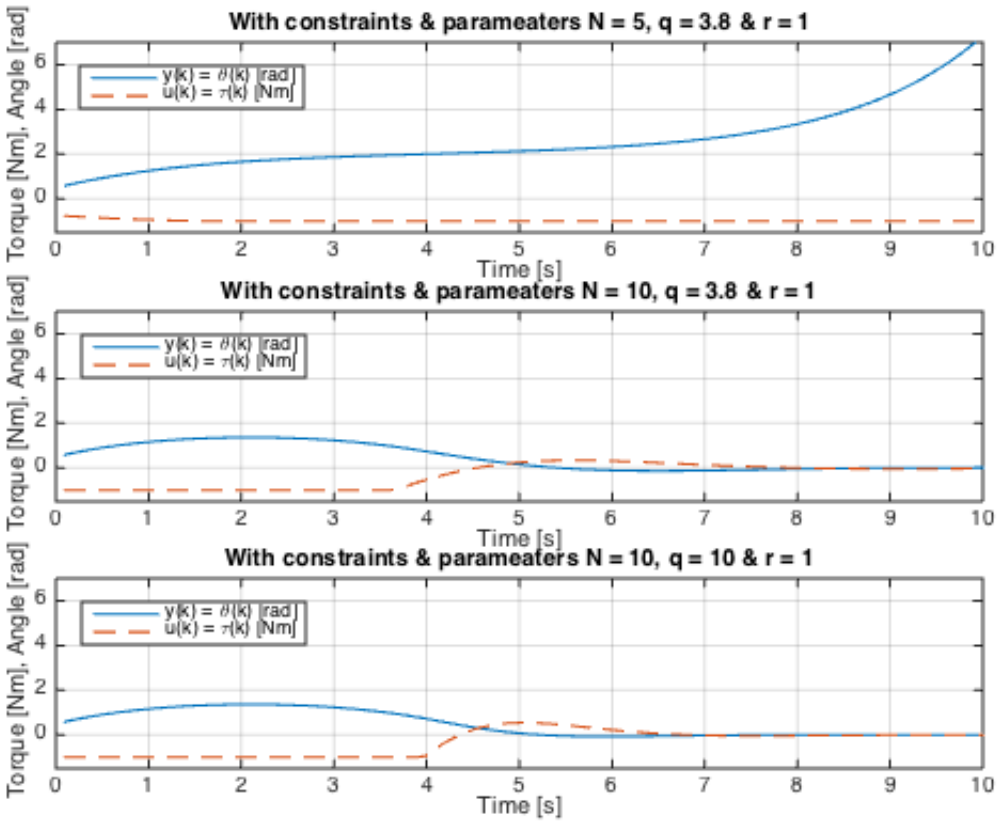

Figure 3: Resulting input and output with constraints for different parameters.

In graph one form Figure 3, it is easy to see that since the system has both constraints and a short prediction horizon the input torque is too small and the pendulum starts falling, this can be seen since the output angle for the pendulum starts to diverge with increasing time. One way to make it stable when the prediction horizon is short is to increase state penalty Q. When increasing the prediction horizon form N=5 to N=10 the controller is able to stabilize the pendulum, even though the input now have constraint. Compared with the unconstrained case but using the same parameters the time for finding stability is longer which is reasonable since it now is a limit for the maximal torque. One way to handle the problem with long stabilization time is to increase the state wighting coefficient, q. This is done in the third graph and then the system converge faster and the controller become more aggressive.

4.3 Optimization algorithm performance

(e)

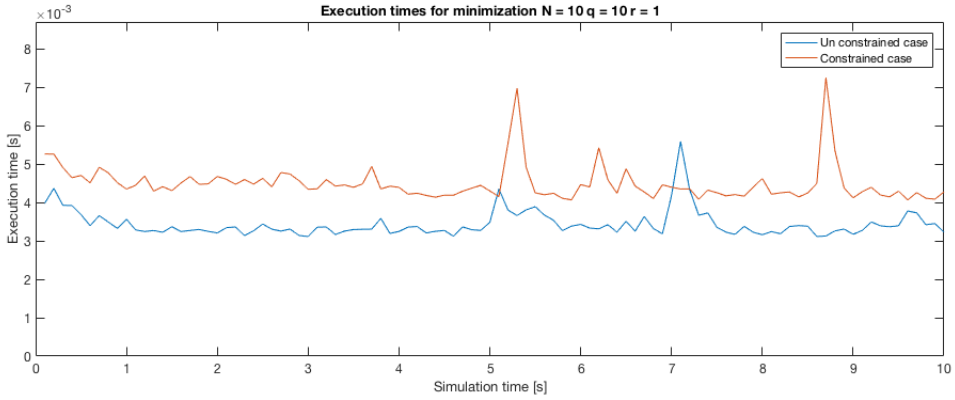

Figure 4: Comparison of execution time for the optimization between constrained and unconstrained case.

As it shows in Figure 4, the execution time for the constrained cases takes longer time than unconstrained cases. This can be that the constraints introduce more calculation to preform during the optimization for the algorithm.

4.4 Reducing the number of optimization variables

(g) Formulate the optimization problem on the condensed form

$$\min_{u_N} \frac{1}{2} u_N^T H u_N + f^T u_N \tag{6a}$$

$$\text{s.t.} \quad A_{eq} u_N = b_{eq} \tag{6b}$$

$$A_{in} u_N \leqslant b_{in} \tag{6c}$$

Repeated use of the pendulum model equations gives:

$$\underbrace{\begin{bmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+N) \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{\Omega} x(k) + \underbrace{\begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}}_{\Gamma} \underbrace{\begin{bmatrix} u(k)) \\ u(k+1)) \\ \vdots \\ u(k+N-1) \end{bmatrix}}_{u_N}$$

with a more compact notation:

$$X = \Omega x(k) + \Gamma u_N$$

Our cost original function can now be written as:

$$\begin{aligned} V_N(x(k), u_N) &= x^T(k)Qx(k) + X^T\bar{Q}X + u_N^T\bar{R}u_N = \\ &= u_N^T(\Gamma^T\bar{Q}\Gamma + \bar{R})u_N + 2x^T(k)\Omega^T\bar{Q}\Gamma u_N + x^T(k)(Q + \Omega^T\bar{Q}\Omega)x(k) \end{aligned}$$

with

$$\bar{Q} = \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & Q & 0 \\ 0 & 0 & 0 & P \end{bmatrix}, \bar{R} = \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & R \end{bmatrix}$$

The initial state term in this function can we ignore because it is constant and we dont need to optimize it. Thus can we see

$$H = 2(\Gamma^T\bar{Q}\Gamma + \bar{R})$$

$$f^T = 2x^T(k)\Omega^T\bar{Q}\Gamma$$

$$A_{in} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 1 \\ -1 & 0 & \cdots & 0 \\ 0 & -1 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & -1 \end{bmatrix}_{(2\cdot N \times N)}, b_{in} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{(2\cdot N \times 1)}$$

A_{eq} and b_{eq} are empty.

5 Apendix Matlab code

5.1 Code for calculations and simulations

```
%%   Basic system model
%% Without constraints
clear all
clc
close all
fig_num =1;
textsize = 10;
%
h=0.1;
A=[1 h;0.5*h 1];
B=[h^2/2; h];
C=[1 0];
n=size(A,1);
m=size(B,2);
%
%    Parameters
%
x0=[0.5 1]';
q_vec = [3.8 3.8 10];
r_vec = [1 1 1];
N_vec = [5 10 10];
%
for l = 1:length(q_vec)
q = q_vec(l);
r = r_vec(l);
N = N_vec(l);
%    Define matrices for the QP
Q=q*(C'*C);
R=r;
P=q*eye(n);
Q_bar=kron(eye(N 1),Q);
R_bar=kron(eye(N),R);
%
H=2*(blkdiag(Q_bar,P,R_bar));
f=[];
%    For the case with actuator constraints
%w2
Ain=[]; % Use empty matrices for the first case without actuator...
bin=[]; % ...constraints and change for the case with constraints!
%
%    Cost function
Nn = kron(eye(N 1),  A);
nn = zeros(2,2*N 2);
nN = [nn;Nn];
NN = [nN,zeros(2*N,2)];
Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
%
%    MPC algorithm
%
T=100; % simulation time
%
xk=x0; % initialize state vector
yvec=[];
uvec=[];
options = optimset('Algorithm','interior point convex','Display','off');
%
AA = [A;zeros(2*N 2,2)];
%
tic;
for k=1:T
beq=AA*xk; % The matrix AA defines how the last measured state xk
% determines the right hand side of the inequality condition.
tic;
z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
t_uncon(k,l) = toc;
uk=z(n*N+1);
xk=A*xk+B*uk;
yvec=[yvec; C*xk];
uvec=[uvec; uk];
end
t_loop_uncon(l) = toc;
tvec=h*(1:1:T);
fig_num =1;
figure(fig_num)
fig_num = 1 + fig_num;
subplot(3,1,l) % For the other two sets of parameters you should change
```

```
% the third index to 2 and 3, respectively.
plot(tvec,yvec,' ',tvec,uvec,'  '); grid
axis([0 10  4  2])
set(gca,'fontsize',textsize)
legend('y(k) = \theta(k) [rad]','u(k) = \tau(k) [Nm]');
title(['Without constraints & parameaters N = ',num2str(N),', q = ',...
    num2str(q),' & r = ',num2str(r)]);
xlabel('Time [s]');
ylabel('Torque [Nm], Angle [rad]');

if l == 2
figure(fig_num)
subplot(3,1,1) % For the other two sets of parameters you should change
% the third index to 2 and 3, respectively.
plot(tvec,yvec,' ',tvec,uvec,'  '); grid
axis([0 10  4  1])
set(gca,'fontsize',textsize)
legend('y(k) = \theta(k) [rad]','u(k) = \tau(k) [Nm]');
title(['Without constraints & parameaters N = ',num2str(N),', q = ',...
    num2str(q),' & r = ',num2str(r)]);
xlabel('Time [s]');
ylabel('Torque [Nm], Angle [rad]');
end

end

% Part C
%   Parameters
%
%
q = 3.8;
r = 1;
N = 10;
%   Define matrices for the QP
Q=q*(C'*C);
R=r;
P=q*eye(n);
Q_bar=kron(eye(N 1),Q);
R_bar=kron(eye(N),R);
%
H=2*(blkdiag(Q_bar,P,R_bar));
f=[];
%   For the case with actuator constraints
%w2
Ain=[]; % Use empty matrices for the first case without actuator...
bin=[]; % ...constraints and change for the case with constraints!
%
%   Cost function
Nn = kron(eye(N 1),  A);
nn = zeros(2,2*N 2);
nN = [nn;Nn];
NN = [nN,zeros(2*N,2)];
Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
%
%   Ricatti algorithm
%
T=100; % simulation time
p=P;
for i=fliplr(1:N 1)
    p=Q+A'*p*A (A'*p*B)*((R+B'*p*B)\(B'*p*A));
end
K =  (R+B'*p*B)\(B'*p*A);
xk=x0; % initialize state vector
 yvec=[];
 uvec=[];
for k=1:T
uk=K * xk;
xk=A*xk+B*uk;
yvec=[yvec; C*xk];
uvec=[uvec; uk];
end

figure(fig_num)

subplot(3,1,2) % For the other two sets of parameters you should change
% the third index to 2 and 3, respectively.
plot(tvec,yvec,' ',tvec,uvec,'  '); grid
axis([0 10  4  1])
set(gca,'fontsize',textsize)
legend('y(k) = \theta(k) [rad]','u(k) = \tau(k) [Nm]');
title(['Ricatti recursions N = ',num2str(N),', q = ',num2str(q),...
    ' & r = ',num2str(r)]);
xlabel('Time [s]');
```

```
ylabel('Torque [Nm], Angle [rad]');
%
q = 3.8;
r = 1;
N = 100;
%   Define matrices for the QP
Q=q*(C'*C);
R=r;
P=q*eye(n);
Q_bar=kron(eye(N 1),Q);
R_bar=kron(eye(N),R);
%
H=2*(blkdiag(Q_bar,P,R_bar));
f=[];
%   For the case with actuator constraints
%w2
Ain=[]; % Use empty matrices for the first case without actuator...
bin=[]; % ...constraints and change for the case with constraints!
%
%   Cost function
Nn = kron(eye(N 1),  A);
nn = zeros(2,2*N 2);
nN = [nn;Nn];
NN = [nN,zeros(2*N,2)];
Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
%
%   MPC algorithm
%
T=100; % simulation time
%
xk=x0; % initialize state vector
yvec=[];
uvec=[];
options = optimset('Algorithm','interior point convex','Display','off');
%
AA = [A;zeros(2*N 2,2)];
beq=AA*x0; % The matrix AA defines how the last measured state xk
% determines the right hand side of the inequality condition.
%
z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
for k=1:T
uk=z(n*N+k);
xk=A*xk+B*uk;
yvec=[yvec; C*xk];
uvec=[uvec; uk];
end
%
subplot(3,1,3) % For the other two sets of parameters you should change
% the third index to 2 and 3, respectively.
plot(tvec,yvec,' ',tvec,uvec,'  '); grid
axis([0 10  4  1])
set(gca,'fontsize',textsize)
legend('y(k) = \theta(k) [rad]','u(k) = \tau(k) [Nm]');
title(['Optimiation of u(k) before loop N = ',num2str(N),', q = ',num2str(q),...
    ' & r = ',num2str(r)]);
xlabel('Time [s]');
ylabel('Torque [Nm], Angle [rad]');
%
fig_num = 1 + fig_num;
%
%% With constraints
exist fig_num
if ans == 1
else
clear all
clc
close all
fig_num =1; %use if run only this section
textsize = 12;
end
%
h=0.1;
A=[1 h;0.5*h 1];
B=[h^2/2; h];
C=[1 0];
n=size(A,1);
m=size(B,2);
%   Parameters
x0=[0.5  1]';
q_vec = [3.8  3.8  10];
r_vec = [1  1  1];
N_vec = [5  10  10];
%
```

```
for l = 1:length(q_vec)
q = q_vec(l);
r = r_vec(l);
N = N_vec(l);
%   Define matrices for the QP
Q=q*(C'*C);
R=r;
P=q*eye(n);
Q_bar=kron(eye(N 1),Q);
R_bar=kron(eye(N),R);
%
H=2*blkdiag(Q_bar,P,R_bar);
f=[];
%   For the case with actuator constraints
%
Ain=[zeros(N,2*N),eye(N);zeros(N,2*N), eye(N)]; % Use empty matrices for...
bin=[ones(1,N)';ones(1,N)']; % ... the first case without actuator
% constraints ... and change for the case with constraints!
%   Cost function
Nn = kron(eye(N 1),  A);
nn = zeros(2,2*N 2);
nN = [nn;Nn];
NN = [nN,zeros(2*N,2)];
Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
%
%   MPC algorithm
%
 T=100; % simulation time
%
 xk=x0; % initialize state vector
 yvec=[];
 uvec=[];
 options = optimset('Algorithm','interior point convex','Display','off');
%options = optimset('Algorithm','active set','Display','off');
%
 AA = [A;zeros(2*N 2,2)];
tic;
for k=1:T
beq=AA*xk; % The matrix AA defines how the last measured state xk
% determines the right hand side of the inequality condition.
tic;
z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
t_con(k,l) = toc;
uk=z(n*N+1);
xk=A*xk+B*uk;
yvec=[yvec; C*xk];
uvec=[uvec; uk];
end
t_loop_con(l) = toc;
 tvec=h*(1:1:T);
figure(fig_num)
%
subplot(3,1,l) % For the other two sets of parameters you should change
% the third index to 2 and 3, respectively.
plot(tvec,yvec,' ',tvec,uvec,'  '); grid
axis([0 10  1.5  7])
legend('y(k) = \theta(k) [rad]','u(k) = \tau(k) [Nm]');
title(['With constraints & parameaters N = ',num2str(N),', q = ', ...
    num2str(q),' & r = ',num2str(r)]);
set(gca,'fontsize',textsize)
xlabel('Time [s]');
ylabel('Torque [Nm], Angle [rad]');
end
fig_num = 1 + fig_num;
%
%% E To run first run With and withoute con
exist fig_num
if ans == 1
else
error('Run secton with and withoute constrains first')
end
figure(fig_num)
fig_num = 1 + fig_num;
plot(tvec,t_uncon(:,3),tvec,t_con(:,3))
set(gca,'fontsize',textsize)
axis([0 10 0 1.2*max(t_con(:,3))])
legend('Un constrained case','Constrained case')
title(['Execution times for minimization N = ',num2str(N) ...
    ', q = ',num2str(q),' & r = ',num2str(r)'])
ylabel('Execution time [s]')
xlabel('Simulation time [s]')
```