

SSY280 Model predictive control  
Assignment 1 Inverted Pendulum  
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

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

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

where  $u(k) = \tau(k)$ ,  $x_1 = \theta$  and  $x_2 = \dot{\theta}$ ,  $\theta$  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  $k$ .

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

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

## 4 Tasks

### 4.1 Without actuator constraints

(a) Rewrite the optimization problem on:

$$\min_z \frac{1}{2} z^T H z \quad (3a)$$

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

Given that

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

Our cost function has the form

$$V_N(x(k), u(k:k+N-1)) = \sum_{k=0}^{N-1} (x^T(k)Qx(k) + u^T(k)Ru(k) + x^T(N)Px(N))$$

From the problem we get:

$$Q = qC^T C = \begin{bmatrix} q & 0 \\ 0 & 0 \end{bmatrix} \quad P = \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix} \quad R = r \quad \Rightarrow \quad H = 2$$

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

Dimension of matrix  $(3 \cdot N \times 3 \cdot N)$

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

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

$I_2$  is a  $2 \times 2$  identity matrix

(b)

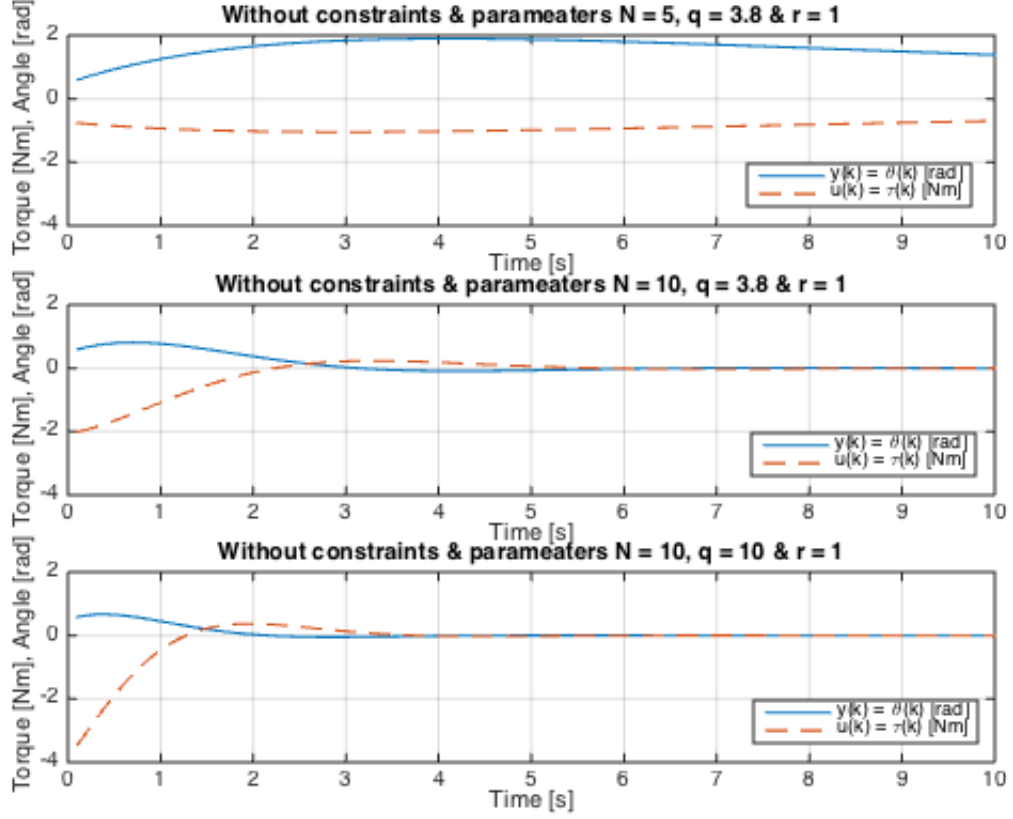


Figure 1: Resulting input and output without constraints for different parameters.

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 far 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 affects 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)

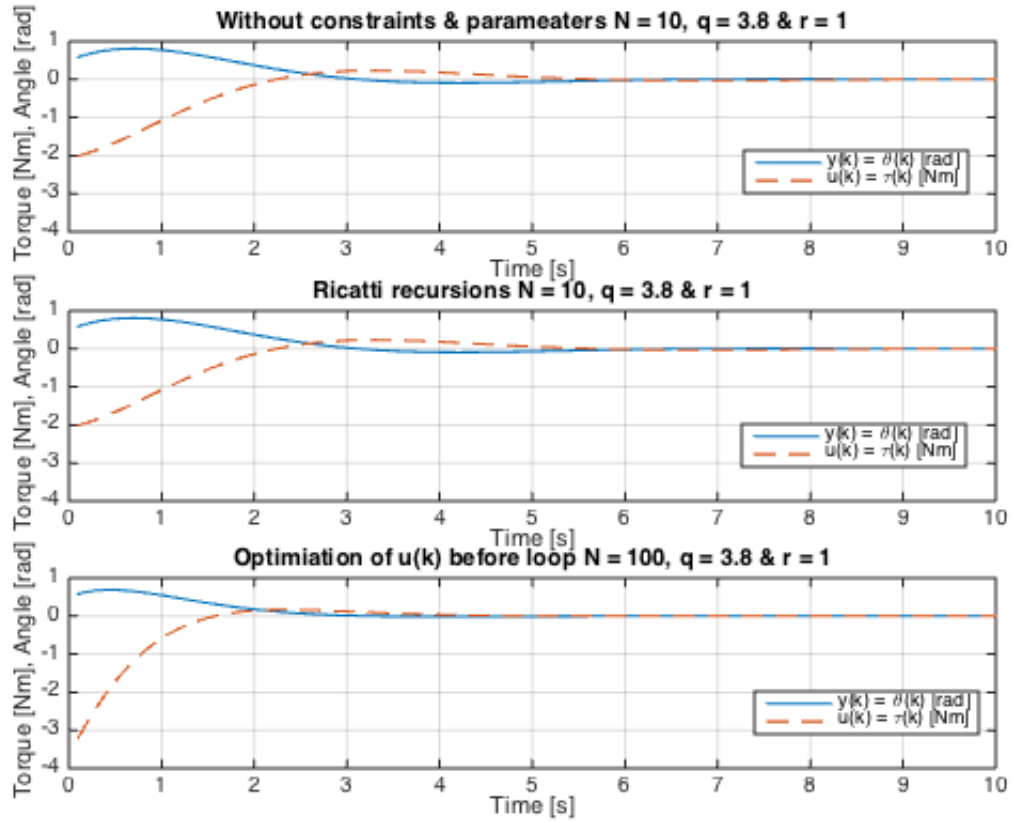


Figure 2: Comparision of different methods to compute controllers.

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 Riccati 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:

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

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

Riccati equation:

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

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:

$$-1 \leq u(k+i) \leq 1, \quad i = 0, 1, \dots, N-1 \quad (4)$$

(d) Rewrite the optimization problem on the form:

$$\min_z \frac{1}{2} z^T H z \quad (5a)$$

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

$$A_{in} z \leq b_{in} \quad (5c)$$

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

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

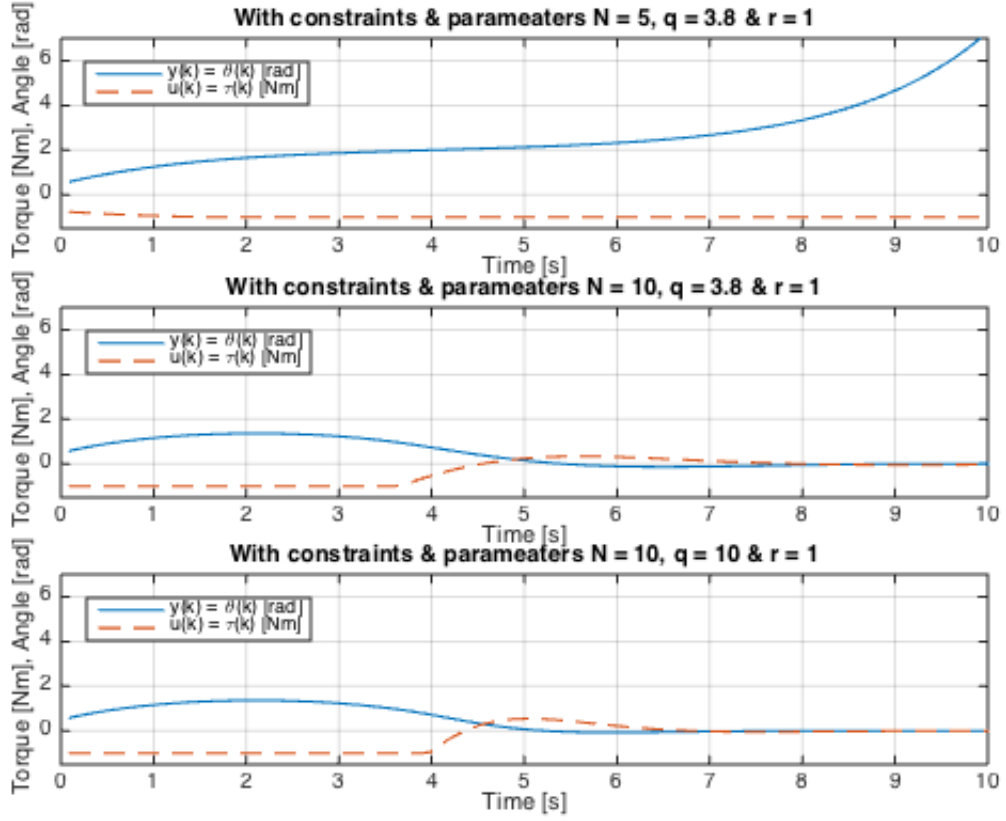


Figure 3: Resulting input and output with constraints for different parameters.

In graph one from 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 from  $N=5$  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 weighting 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)

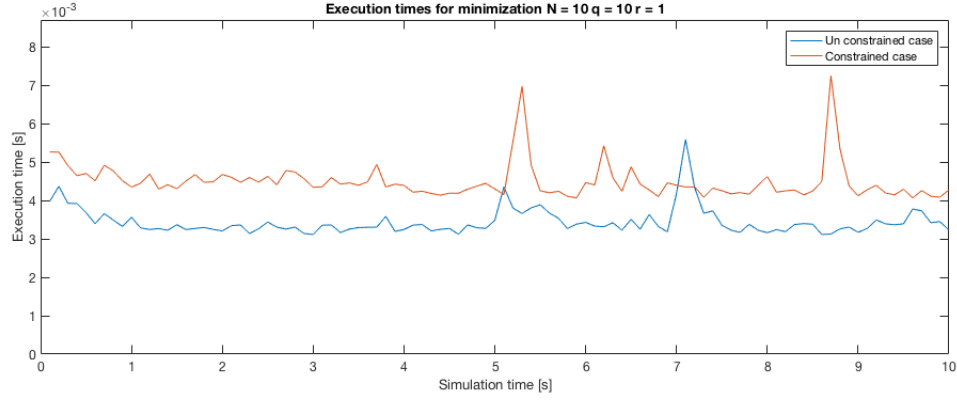


Figure 4: Comparison of execution time for the optimization between constrained and unconstrained case.

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 perform during the optimization for the algorithm.

### 4.4 Reducing the number of optimization variables

(g) Formulate the optimization problem on the condensed form

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

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

$$A_{in} u_N \leq b_{in} \quad (6c)$$

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

```
1 %% Basic system model
2 %% Without constraints
3 clear all
4 clc
5 close all
6 fig_num =1;
7 textsize = 10;
8 %
9 h=0.1;
10 A=[1 h;0.5*h 1];
11 B=[h^2/2; h];
12 C=[1 0];
13 n=size(A,1);
14 m=size(B,2);
15 %
16 % Parameters
17 %
18 x0=[0.5 1]';
19 q_vec = [3.8 3.8 10];
20 r_vec = [1 1 1];
21 N_vec = [5 10 10];
22 %
23 for l = 1:length(q_vec)
24 q = q_vec(l);
25 r = r_vec(l);
26 N = N_vec(l);
27 % Define matrices for the QP
28 Q=q*(C'*C);
29 R=r;
30 P=eye(n);
31 Q_bar=kron(eye(N 1),Q);
32 R_bar=kron(eye(N),R);
33 %
34 H=2*(blkdiag(Q_bar,P,R_bar));
35 f=[];
36 % For the case with actuator constraints
37 %w2
38 Ain=[]; % Use empty matrices for the first case without actuator...
39 bin=[]; % ...constraints and change for the case with constraints!
40 %
41 % Cost function
42 Nn = kron(eye(N 1), A);
43 nn = zeros(2,2*N 2);
44 nN = [nn;Nn];
45 NN = [nN,zeros(2*N,2)];
46 Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
47 %
48 % MPC algorithm
49 %
50 T=100; % simulation time
51 %
52 xk=x0; % initialize state vector
53 yvec=[];
54 uvec=[];
55 options = optimset('Algorithm','interior point convex','Display','off');
56 %
57 AA = [A;zeros(2*N 2,2)];
58 %
59 tic;
60 for k=1:T
61 beq=AA*xk; % The matrix AA defines how the last measured state xk
62 % determines the right hand side of the inequality condition.
63 tic;
64 z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
65 t.uncon(k,1) = toc;
66 uk=z(n*N+1);
67 xk=A*xk+B*uk;
68 yvec=[yvec; C*xk];
69 uvec=[uvec; uk];
70 end
71 t_loop_uncon(1) = toc;
72 tvec=h*(1:1:T);
73 fig_num =1;
74 figure(fig_num)
75 fig_num = 1 + fig_num;
76 subplot(3,1,1) % For the other two sets of parameters you should change
```

```

77 % the third index to 2 and 3, respectively.
78 plot(tvec,yvec,' ',tvec,uvec,' '); grid
79 axis([0 10 4 2])
80 set(gca,'fontsize',textsize)
81 legend('y(k) = \theta(k) [rad]', 'u(k) = \tau(k) [Nm]');
82 title(['Without constraints & parameters N = ',num2str(N),', q = ',...
83       num2str(q), ' & r = ',num2str(r)]);
84 xlabel('Time [s]');
85 ylabel('Torque [Nm], Angle [rad]');
86
87 if l == 2
88 figure(fig_num)
89 subplot(3,1,1) % For the other two sets of parameters you should change
90 % the third index to 2 and 3, respectively.
91 plot(tvec,yvec,' ',tvec,uvec,' '); grid
92 axis([0 10 4 1])
93 set(gca,'fontsize',textsize)
94 legend('y(k) = \theta(k) [rad]', 'u(k) = \tau(k) [Nm]');
95 title(['Without constraints & parameters N = ',num2str(N),', q = ',...
96       num2str(q), ' & r = ',num2str(r)]);
97 xlabel('Time [s]');
98 ylabel('Torque [Nm], Angle [rad]');
99 end
100
101 end
102
103 % Part C
104 % Parameters
105 %
106 %
107 q = 3.8;
108 r = 1;
109 N = 10;
110 % Define matrices for the QP
111 Q=q*(C'*C);
112 R=r;
113 P=q*eye(n);
114 Q_bar=kron(eye(N 1),Q);
115 R_bar=kron(eye(N),R);
116 %
117 H=2*(blkdiag(Q_bar,P,R_bar));
118 f=[];
119 % For the case with actuator constraints
120 %w2
121 Ain=[]; % Use empty matrices for the first case without actuator...
122 bin=[]; % ...constraints and change for the case with constraints!
123 %
124 % Cost function
125 Nn = kron(eye(N 1), A);
126 nn = zeros(2,2*N 2);
127 nN = [nn;Nn];
128 NN = [nN,zeros(2*N,2)];
129 Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
130 %
131 % Ricatti algorithm
132 %
133 T=100; % simulation time
134 p=P;
135 for i=flip(1:N 1)
136 p=Q+A'*p*A - (A'*p*B)*(R+B'*p*B)\(B'*p*A);
137 end
138 K = (R+B'*p*B)\(B'*p*A);
139 xk=x0; % initialize state vector
140 yvec=[];
141 uvec=[];
142 for k=1:T
143 uk=K * xk;
144 xk=A*xk+B*uk;
145 yvec=[yvec; C*xk];
146 uvec=[uvec; uk];
147 end
148
149 figure(fig_num)
150
151 subplot(3,1,2) % For the other two sets of parameters you should change
152 % the third index to 2 and 3, respectively.
153 plot(tvec,yvec,' ',tvec,uvec,' '); grid
154 axis([0 10 4 1])
155 set(gca,'fontsize',textsize)
156 legend('y(k) = \theta(k) [rad]', 'u(k) = \tau(k) [Nm]');
157 title(['Ricatti recursions N = ',num2str(N),', q = ',num2str(q),...
158       ' & r = ',num2str(r)]);
159 xlabel('Time [s]');

```

```

160 ylabel('Torque [Nm], Angle [rad]');
161 %
162 q = 3.8;
163 r = 1;
164 N = 100;
165 % Define matrices for the QP
166 Q=q*(C'*C);
167 R=r;
168 P=q*eye(n);
169 Q_bar=kron(eye(N 1),Q);
170 R_bar=kron(eye(N),R);
171 %
172 H=2*(blkdiag(Q_bar,P,R_bar));
173 f=[];
174 % For the case with actuator constraints
175 %w2
176 Ain=[]; % Use empty matrices for the first case without actuator...
177 bin=[]; % ...constraints and change for the case with constraints!
178 %
179 % Cost function
180 Nn = kron(eye(N 1), A);
181 nn = zeros(2,2*N 2);
182 nN = [nn;Nn];
183 NN = [nN,zeros(2*N,2)];
184 Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
185 %
186 % MPC algorithm
187 %
188 T=100; % simulation time
189 %
190 xk=x0; % initialize state vector
191 yvec=[];
192 uvec=[];
193 options = optimset('Algorithm','interior point convex','Display','off');
194 %
195 AA = [A;zeros(2*N 2,2)];
196 beq=AA*x0; % The matrix AA defines how the last measured state xk
197 % determines the right hand side of the inequality condition.
198 %
199 z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
200 for k=1:T
201 uk=z(n*N+k);
202 xk=A*xk+B*uk;
203 yvec=[yvec; C*xk];
204 uvec=[uvec; uk];
205 end
206 %
207 subplot(3,1,3) % For the other two sets of parameters you should change
208 % the third index to 2 and 3, respectively.
209 plot(tvec,yvec,' ',tvec,uvec,' '); grid
210 axis([0 10 4 1])
211 set(gca,'fontsize',textsize)
212 legend('y(k) = \theta(k) [rad]', 'u(k) = \tau(k) [Nm]');
213 title(['Optimization of u(k) before loop N = ',num2str(N),', q = ',num2str(q),...
214 ' & r = ',num2str(r)]);
215 xlabel('Time [s]');
216 ylabel('Torque [Nm], Angle [rad]');
217 %
218 fig_num = 1 + fig_num;
219 %
220 %% With constraints
221 exist fig_num
222 if ans == 1
223 else
224 clear all
225 clc
226 close all
227 fig_num =1; %use if run only this section
228 textsize = 12;
229 end
230 %
231 h=0.1;
232 A=[1 h;0.5*h 1];
233 B=[h^2/2; h];
234 C=[1 0];
235 n=size(A,1);
236 m=size(B,2);
237 % Parameters
238 x0=[0.5 1]';
239 q_vec = [3.8 3.8 10];
240 r_vec = [1 1 1];
241 N_vec = [5 10 10];
242 %

```

```

243 for l = 1:length(q_vec)
244     q = q_vec(l);
245     r = r_vec(l);
246     N = N_vec(l);
247     % Define matrices for the QP
248     Q=q*(C'*C);
249     R=r;
250     P=q*eye(n);
251     Q_bar=kron(eye(N 1),Q);
252     R_bar=kron(eye(N),R);
253     %
254     H=2*blkdiag(Q_bar,P,R_bar);
255     f=[];
256     % For the case with actuator constraints
257     %
258     Ain=[zeros(N,2*N),eye(N);zeros(N,2*N), eye(N)]; % Use empty matrices for...
259     bin=[ones(1,N)';ones(1,N)']; % ... the first case without actuator
260     % constraints ... and change for the case with constraints!
261     % Cost function
262     Nn = kron(eye(N 1), A);
263     nn = zeros(2,2*N 2);
264     nN = [nn;Nn];
265     NN = [nN,zeros(2*N,2)];
266     Aeq = [kron(eye(N),eye(2)) + NN,kron(eye(N), B)];
267     %
268     % MPC algorithm
269     %
270     T=100; % simulation time
271     %
272     xk=x0; % initialize state vector
273     yvec=[];
274     uvec=[];
275     options = optimset('Algorithm','interior point convex','Display','off');
276     %options = optimset('Algorithm','active set','Display','off');
277     %
278     AA = [A;zeros(2*N 2,2)];
279     tic;
280     for k=1:T
281         beq=AA*xk; % The matrix AA defines how the last measured state xk
282         % determines the right hand side of the inequality condition.
283         tic;
284         z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
285         t_con(k,1) = toc;
286         uk=z(n*N+1);
287         xk=A*xk+B*uk;
288         yvec=[yvec; C*xk];
289         uvec=[uvec; uk];
290     end
291     t_loop_con(1) = toc;
292     tvec=h*(1:1:T);
293     figure(fig_num)
294     %
295     subplot(3,1,1) % For the other two sets of parameters you should change
296     % the third index to 2 and 3, respectively.
297     plot(tvec,yvec,' ',tvec,uvec,' '); grid
298     axis([0 10 1.5 7])
299     legend('y(k) = \theta(k) [rad]', 'u(k) = \tau(k) [Nm]');
300     title(['With constraints & parameters N = ',num2str(N),', q = ', ...
301           num2str(q), ' & r = ',num2str(r)]);
302     set(gca,'fontsize',textsize)
303     xlabel('Time [s]');
304     ylabel('Torque [Nm], Angle [rad]');
305     end
306     fig_num = 1 + fig_num;
307     %
308     %% E To run first run With and without con
309     exist fig_num
310     if ans == 1
311         else
312         error('Run section with and without constraints first')
313     end
314     figure(fig_num)
315     fig_num = 1 + fig_num;
316     plot(tvec,t_uncon(:,3),tvec,t_con(:,3))
317     set(gca,'fontsize',textsize)
318     axis([0 10 0 1.2*max(t_con(:,3))])
319     legend('Un constrained case','Constrained case')
320     title(['Execution times for minimization N = ',num2str(N) ...
321           ', q = ',num2str(q), ' & r = ',num2str(r)]);
322     ylabel('Execution time [s]')
323     xlabel('Simulation time [s]')

```