SSY281 Model Predictive Control

Assignment 1

Yongzhao Chen

February 8, 2023

1 Question 1

1.1 a

Following Lemma2.1, I used *exmp* to calculate the A, B, C, then verify the result using *c2d* method.
The code for Q1 is in $homework1_12.mlx$

```
disH=expm([A,B;zeros(1,4),0]*h)
Adh=disH(1:4,1:4)
Bdh=disH(1:4,5)

%verify
sysc=ss(A,B,C,D);
sysd=c2d(sysc,h)
```

The result shows:

```

Adh = ×44
    1.4421    0.1143         0   -0.0045
    9.4370    1.4421         0   -0.0908
    0.0237    0.0008    1.0000    0.0833
    0.4814    0.0237         0    0.6845

Bdh = ×41
    0.0677
    1.3715
    0.2530
    4.7660

C = ×24
     1     0     0     0
     0     0     1     0

sysd =

  A =
              x1          x2          x3          x4
   x1      1.442      0.1143           0   -0.004479
   x2      9.437       1.442           0    -0.09079
   x3    0.02375   0.0007924           1     0.08325
```

```
  x4      0.4814    0.02375          0      0.6845

 B =
           u1
  x1  0.06765
  x2    1.371
  x3    0.253
  x4    4.766

 C =
      x1  x2  x3  x4
  y1   1   0   0   0
  y2   0   0   1   0

 D =
      u1
  y1   0
  y2   0
```

1.2 b

Follow the *Lemma*2.2, change the code from question a

```
t=0.8*h;
disHt=expm([A,B;zeros(1,4),0]*(h-t))
Adht=disHt(1:4,1:4)
Bdht=disHt(1:4,5)
```

The output is :

```
Adht = ×44
    1.0166    0.0201         0   -0.0002
    1.6627    1.0166         0   -0.0182
    0.0010    0.0000    1.0000    0.0193
    0.0968    0.0010         0    0.9272

Bdht = ×41
    0.0028
    0.2757
    0.0111
```

```
    1.1002
```

The eigenvalues of A, A_a shows below:

```
ans(A) = ×41
    1.0000
    2.4781
    0.4008
    0.6899

ans(A_a) = ×41
    1.0000
    1.1990
    0.8329
    0.9284
```

We can see the time delay are able to change the eigenvalue of the system, which may change the stability in some situation.

2 Question 2: Dynamic Programming solution of the LQ problem

$DPfunctio, mBatchfunction, Dp_con$ function are all attached as files.

2.1 a

Set a loop until the system is stable,the criterion is $\lambda(A - B * K) < 1$.
Because in the book equation and in my code, the signs of K are all negative, so in the code I use + instead.

```
Na = 33
eiga = ×41
    0.5584
    0.9236
    0.9524
    0.9996
Pend = ×44
1.0e+04 *
```

```

    3.0780    0.3401   -0.0201   -0.0731
    0.3401    0.0382   -0.0023   -0.0082
   -0.0201   -0.0023    0.0042    0.0006
   -0.0731   -0.0082    0.0006    0.0020
Kend =

  -46.1566   -5.2278    0.0155    0.4031
```

2.2 b

Comparing with a, I add the constrain to DP.m shows below:

```
if norm(P{i+1}-P{i}) <= 1e-1
    break;
end
```

And the output of *idare* function is:

```
P =

   1.0e+04 *

    4.8730    0.5323   -0.1038   -0.1154
    0.5323    0.0587   -0.0114   -0.0127
   -0.1038   -0.0114    0.0125    0.0027
   -0.1154   -0.0127    0.0027    0.0030
```

The output of my DP_con is:

```
Pcon =

   1.0e+04 *

    4.8726    0.5322   -0.1037   -0.1154
    0.5322    0.0587   -0.0114   -0.0127
   -0.1037   -0.0114    0.0125    0.0027
   -0.1154   -0.0127    0.0027    0.0030

timecon =

```

```
    426
```

The iteration times are 426.

2.3 c

Take the P_con above into my DP function:

```
stable = false;
Nc = 0;
while ~stable
    Nc = Nc+1;
    [Kend1,Pend1] = DP(A,B,Q,R,Pcon,Nc);
    if all(abs(eig(A+B*Kend))<=1)
        stable = true;
    end
end
Nc
eigc=eig(A+B*Kend1)
Pend1
Kend1
```

Which says $Nc = 1$ and the $Pend1$ is the same with my $Pcon$.
The result is natural that in section 2b, we already calculate the stationary solution of the Ricatti equation. In this case, no matter N, the result will always be the same, which is equal to the infinite-horizon solution. So just 1 step we reach the solution and no matter how many steps the solution P will always stay the same with P_con above.

3 3

mBatch function is attached.

```
stable = false;
N5 = 0;
while ~stable
    N5 = N5+1;
    [K0,Pend] = mBatch(A,B,Q,R,Pf,N5);
    if all(abs(eig(A+B*K0))<=1)
        stable = true;
```

```
    end
end

Pf
N5
eig(A+B*K0)
Pend
K0
```

And as expected, the result of Batch solution matches exactly with the DP solution.

```
N5 =

    33


ans =

    0.5584
    0.9236
    0.9524
    0.9996


Pend =

   1.0e+04 *

    3.0780    0.3401   -0.0201   -0.0731
    0.3401    0.0382   -0.0023   -0.0082
   -0.0201   -0.0023    0.0042    0.0006
   -0.0731   -0.0082    0.0006    0.0020


K0 =

  -46.1566   -5.2278    0.0155    0.4031
```

4 Question 4:Receding horizon control

I achieved RHC using my own mBatch function.

```
R4=[1,1,0.1,0.1];
N4=[40,80,40,80];
Kend=cell(4,1);
%using mBatch to solve this
for i = 1:4
    [Kend{i},Pend]=mBatch(A,B,Q,R4(i),Pf,N4(i));
end

Kend
T=201;
x = cell(4,1);
u = cell(4,1);
for i = 1:4
    x{i} = zeros(T, 4);
    u{i} = zeros(T, 1);
    x{i}(1, :) = x0;
    for j = 2:T
        x{i}(j, :) = (A + B * Kend{i}) * x{i}(j-1, :)';
        u{i}(j) = Kend{i}*x{i}(j-1, :)';
    end
end
```

And the plot result shows below:

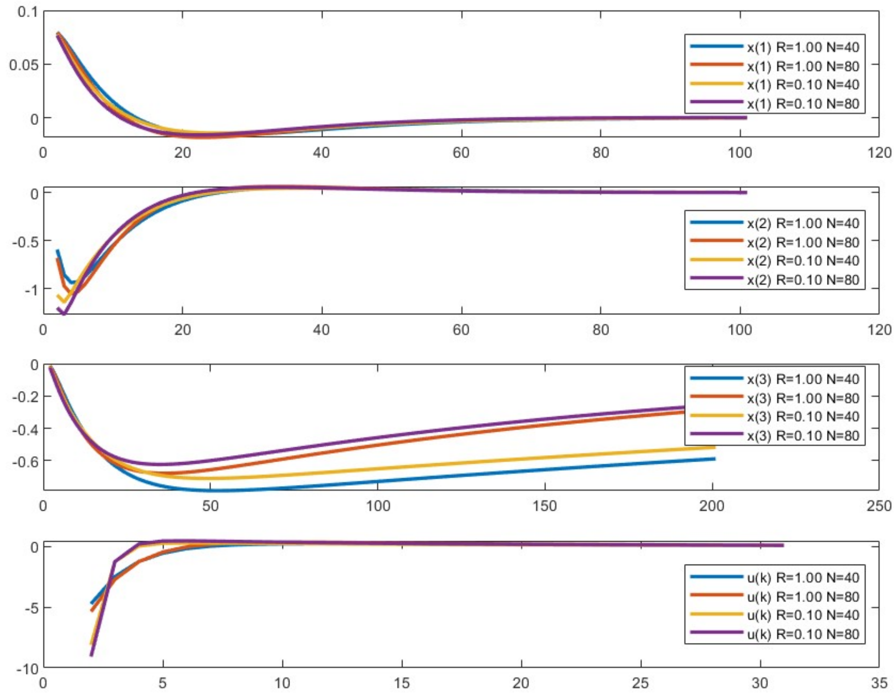

Figure 1: Receding horizon control

The Figure1 is plot without the X0 U0, this remains later in Question 5.

5 Question 5:Constrained receding horizon control

The code for question 5 is in *question5.mlx*. I used CRHC.m from PSS1, and the contraint is changed to this:

```
    F  = kron([eye(N); zeros(N)], [0 1 0 0;0 -1 0 0;0 0 0 0;0 0 0 0]);
    G  = kron([zeros(N); eye(N)], [1; -1; 1 ; -1]);
    h  = [x2_max*ones(n*N,1); u_max*ones(n*N,1)];
```

And the plot result shows below:

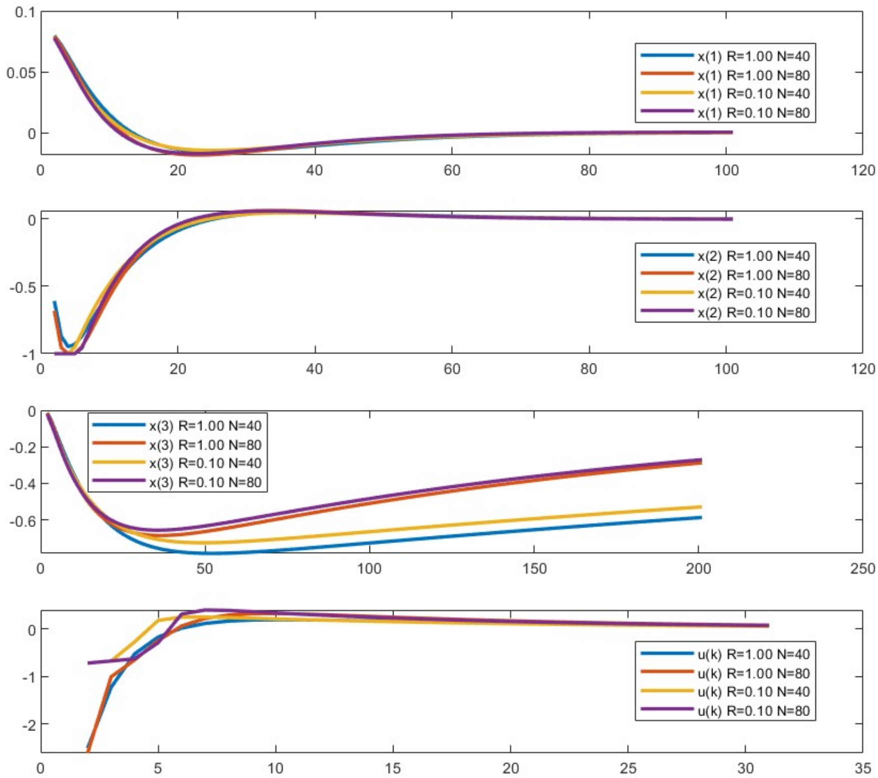

Figure 2: Constraint Receding horizon control

Comparing the Figure2 and Figure1, we can find the main difference is at the control signal u. Because the constraint exist that the u need to consider the range.
This appears apparently when R=0.1, which means u is kind of cheap. In unconstraint situation, we can see control signal u goes high while in the constraint situation control signal u runs more cautious.