



- 1 Choosing:  $I = S$ ,  $BR^{-1}B^{\top}P = UTV$ ,  $R^{-1} = T \implies T^{-1} = R$ ,  $VS^{-1}U = B^{\top}PB \implies U = B \wedge V = B^{\top}P$ . These were chosen to match it with the expected result (6). By the matrix inversion lemma this gives us:

$$(I + BR^{-1}B^{\top}P)^{-1} = I - IB(R + B^{\top}PIB)^{-1}B^{\top}PI$$

Inserting this into (5) gives us:

$$\begin{aligned} P &= Q + A^{\top}P(I - IB(R + B^{\top}PIB)^{-1}B^{\top}PI)A \\ &= Q + A^{\top}PA - A^{\top}PB(R + B^{\top}PB)^{-1}B^{\top}PA \\ 0 &= Q + A^{\top}PA - A^{\top}PB(R + B^{\top}PB)^{-1}B^{\top}PA - P \end{aligned}$$

Which is (6).

- 2 a) The code used is shown below.

```
1 Q = 1/2 * [  
2     4 0  
3     0 4  
4  ];  
5  
6 A = [  
7     1 0.1  
8    -0.1 1-0.1  
9  ];  
10  
11 R = 1/2 * 1;  
12  
13 B = [  
14     0  
15     0.1  
16  ];  
17  
18 [K, S, e] = dlqr(A, B, Q, R, []);  
19 disp(K);  
20 disp(S);  
21 disp(e);
```

Results in  $K = \begin{bmatrix} 1.0373 \\ 1.6498 \end{bmatrix}$  and  $e = \text{eig}(A - BK) = 0.8675 \pm 0.0531i$

b) The code used is shown below, I used the poles suggested by the task.

```
1  Q = 1/2 * [  
2      4  0  
3      0  4  
4  ];  
5  
6  A = [  
7      1  0.1  
8     -0.1  1-0.1  
9  ];  
10  
11 R = 1/2 * 1;  
12  
13 B = [  
14      0  
15      0.1  
16  ];  
17  
18 C = [  
19      1  
20      0  
21  ];  
22  
23 p = [  
24      0.5 + 0.03i  
25      0.5 - 0.03i  
26  ];  
27 K_f = place(A', C, p);  
28  
29 [K, S, e] = dlqr(A, B, Q, R, []);  
30  
31 x_0 = [5 1]';  
32 x_0_e = [6 0]';  
33  
34 x = zeros(2,50);  
35 x(:,1) = x_0;  
36 x_e = zeros(2,50);  
37 x_e(:,1) = x_0_e;  
38 u = zeros(50);  
39  
40 for t=2:50  
41     u(t-1) = -K*x_e(:,t-1);  
42     y = C'*x(:,t-1);  
43     y_e = C'*x_e(:,t-1);  
44     x(:,t) = A*x(:,t-1) + B*u(t-1);  
45     x_e(:,t) = A*x_e(:,t-1) + B*u(t-1)+K_f'*(y-y_e);  
46 end  
47  
48 figure(1);
```

```
49 subplot(2, 1, 1);
50 t = 0:49;
51 plot(t, x(1,:));
52 hold on;
53 plot(t, x_e(1,:));
54 hold off;
55 xlabel('t');
56 ylabel('x_1');
57 legend('$x_1(t)$', '$\hat{x}_1(t)$', 'Interpreter', 'latex
    ');
58 grid('on');
59
60 figure(1);
61 subplot(2, 1, 2);
62 t = 0:49;
63 plot(t, x(2,:));
64 hold on;
65 plot(t, x_e(2,:));
66 hold off;
67 xlabel('t');
68 ylabel('x_2');
69 legend('$x_2(t)$', '$\hat{x}_2(t)$', 'Interpreter', 'latex
    ');
70 grid('on');
```

This resulted in the graph shown in Figure 1.

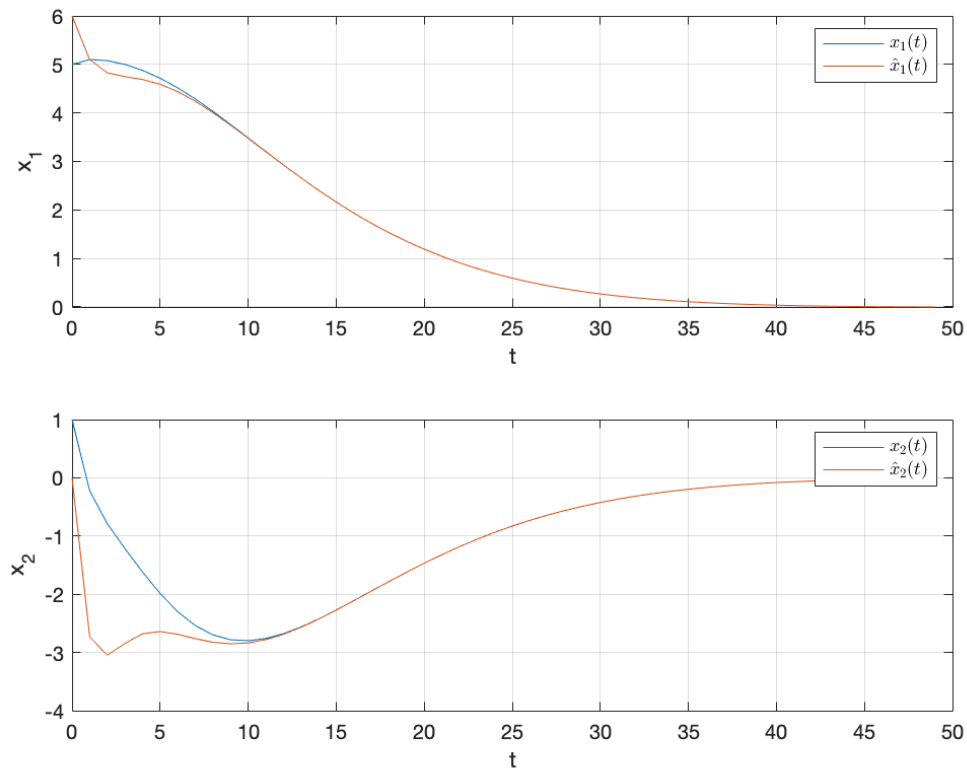


Figure 1: LQR with output feedback.

As we can see the estimated state converges pretty quickly towards the real state. The controller uses some time to converge towards the origin, but manages to do it within 50 time steps.

c) We have that:

$$\begin{aligned}
 BK &= \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} [1.0373 \quad 1.6498] = \begin{bmatrix} 0 & 0 \\ 0.10373 & 0.16498 \end{bmatrix} \\
 A - BK &= \begin{bmatrix} 1 & 0.1 \\ -0.1 & 0.9 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0.10373 & 0.16498 \end{bmatrix} = \begin{bmatrix} 1 & 0.1 \\ -0.2037 & 0.7350 \end{bmatrix} \\
 A - K_F C &= \begin{bmatrix} 1 & 0.1 \\ -0.1 & 0.9 \end{bmatrix} - \begin{bmatrix} 0.9 \\ 1.509 \end{bmatrix} [1 \quad 0] = \begin{bmatrix} 0.1 & 0.1 \\ -1.609 & 0.9 \end{bmatrix} \\
 \Rightarrow \Phi &= \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ -0.2037 & 0.7350 & 0.10373 & 0.16498 \\ 0 & 0 & 0.1 & 0.1 \\ 0 & 0 & -1.609 & 0.9 \end{bmatrix}
 \end{aligned}$$

Using `eig` in Matlab to find the eigen values of  $\Phi$ :

$$\lambda_1 = 0.8675 \pm 0.0530i$$

$$\lambda_2 = 0.5 \pm 0.03i$$

Which are the poles we got earlier.

**3** a) The code used is shown below.

```
1 Qt = [  
2     4 0  
3     0 4  
4 ];  
5  
6 A = [  
7     1 0.1  
8    -0.1 1-0.1  
9 ];  
10  
11 B = [  
12     0  
13    0.1  
14 ];  
15  
16 C = [  
17     1  
18     0  
19 ];  
20  
21 x0 = [5 1]';  
22 x0_e = [6 0]';  
23  
24 N = 10;  
25 nx = 2;  
26 nu = 1;  
27 r = 1;  
28  
29 % Set G (objective function)  
30 Q = kron(eye(N),Qt);  
31 Rt = r;  
32 R = kron(eye(N),Rt);  
33 G = blkdiag(Q, R);  
34  
35  
36 % Set equality constraints (Aeq and Beq)  
37 Beq = zeros(N*nx, 1);  
38  
39 Aeq_1 = eye(N*nx);  
40 Aeq_2 = kron(diag(ones(N-1,1),-1),-A);  
41 Aeq_3 = kron(eye(N), -B);  
42 Aeq = [Aeq_1 + Aeq_2, Aeq_3];  
43  
44 u_low = -4;  
45 u_high = 4;  
46  
47 x_high = inf;  
48 x_low = -inf;
```

```
49
50 lb = [x_low*ones(N*nx, 1); u_low*ones(N*nu, 1)];
51 ub = [x_high*ones(N*nx, 1); u_high*ones(N*nu, 1)];
52
53 p = [
54     0.5 + 0.03i
55     0.5 - 0.03i
56 ];
57 K_f = place(A', C, p);
58
59
60 x = zeros(nx,N);
61 x(:,1) = x0;
62 x_e = zeros(nx,N);
63 x_e(:,1) = x0_e;
64
65 u = zeros(nu, 51);
66
67 for t = 1:50
68     Beq(1:nx) = A*x_e(:,t);
69     z = quadprog(G, [], [], [], Aeq, Beq, lb, ub);
70     u(t) = z(N*nx+1); % get first element of u as actual
        control input
71     disp(u(t));
72     x(:,t+1) = A*x(:,t) + B*u(t); % state space update
73     y_e = C'*x_e(:,t);
74     y = C'*x(:,t);
75     x_e(:,t+1) = A*x_e(:,t) + B*u(t) + K_f'*(y-y_e);
76 end
77
78 figure(1);
79 subplot(3, 1, 1);
80 t = 0:50;
81 plot(t, x(1,:));
82 hold on;
83 plot(t, x_e(1,:));
84 hold off;
85 xlabel('t');
86 ylabel('x_1');
87 legend('$x_1(t)$', '$\hat{x}_1(t)$', 'Interpreter', 'latex
    ');
88 grid('on');
89
90 figure(1);
91 subplot(3, 1, 2);
92 plot(t, x(2,:));
93 hold on;
94 plot(t, x_e(2,:));
95 hold off;
```

```

96 xlabel('t');
97 ylabel('x_2');
98 legend('$x_2(t)$', '$\hat{x}_2(t)$', 'Interpreter','latex
    ');
99 grid('on');
100
101 disp(size(u));
102 disp(size(t));
103 figure(1);
104 subplot(3, 1, 3);
105 plot(t, u);
106 xlabel('t');
107 ylabel('u');
108 legend('$u(t)$', 'Interpreter','latex');
109 grid('on');

```

The results are given in Figure 2.

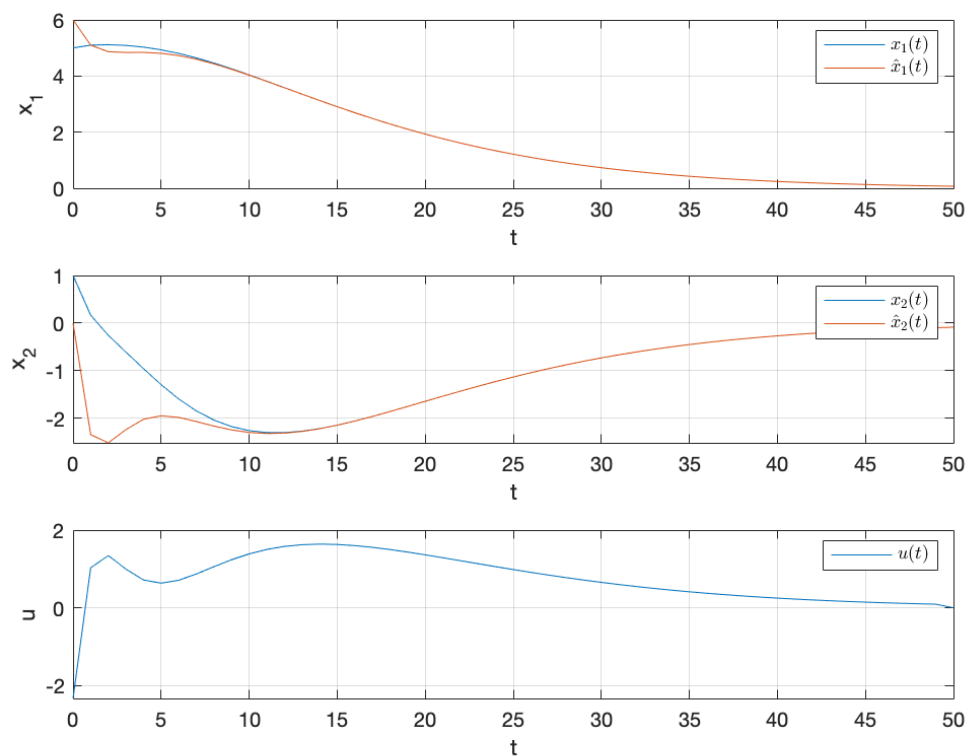


Figure 2: MPC with output feedback.

The code used is shown below.

```

1 Qt = [
2     4 0
3     0 4
4 ];

```

```
5
6 A = [
7       1  0.1
8      -0.1  1-0.1
9    ];
10
11 B = [
12       0
13      0.1
14    ];
15
16 C = [
17       1
18       0
19    ];
20
21 x0 = [5  1]';
22 x0_e = [6  0]';
23
24 N = 10;
25 nx = 2;
26 nu = 1;
27 r = 1;
28
29 % Set G (objective function)
30 Q = kron(eye(N),Qt);
31 Rt = r;
32 R = kron(eye(N),Rt);
33 G = blkdiag(Q, R);
34
35
36 % Set equality constraints (Aeq and Beq)
37 Beq = zeros(N*nx, 1);
38
39 Aeq_1 = eye(N*nx);
40 Aeq_2 = kron(diag(ones(N-1,1),-1),-A);
41 Aeq_3 = kron(eye(N), -B);
42 Aeq = [Aeq_1 + Aeq_2, Aeq_3];
43
44 u_low = -4;
45 u_high = 4;
46
47 x_high = inf;
48 x_low = -inf;
49
50 lb = [x_low*ones(N*nx, 1); u_low*ones(N*nu, 1)];
51 ub = [x_high*ones(N*nx, 1); u_high*ones(N*nu, 1)];
52
53 p = [
```



```
54      0.5 + 0.03i
55      0.5 - 0.03i
56  ];
57  K_f = place(A', C, p);
58
59
60  x = zeros(nx,N);
61  x(:,1) = x0;
62  x_e = zeros(nx,N);
63  x_e(:,1) = x0_e;
64
65  u = zeros(nu, 51);
66
67  for t = 1:50
68      Beq(1:nx) = A*x_e(:,t);
69      z = quadprog(G, [], [], [], Aeq, Beq, lb, ub);
70      u(t) = z(N*nx+1); % get first element of u as actual
        control input
71      disp(u(t));
72      x(:,t+1) = A*x(:,t) + B*u(t); % state space update
73      y_e = C'*x_e(:,t);
74      y = C'*x(:,t);
75      x_e(:,t+1) = A*x_e(:,t) + B*u(t) + K_f'*(y-y_e);
76  end
77
78  figure(1);
79  subplot(3, 1, 1);
80  t = 0:50;
81  plot(t, x(1,:));
82  hold on;
83  plot(t, x_e(1,:));
84  hold off;
85  xlabel('t');
86  ylabel('x_1');
87  legend('$x_1(t)$', '$\hat{x}_1(t)$', 'Interpreter','latex');
88  grid('on');
89
90  figure(1);
91  subplot(3, 1, 2);
92  plot(t, x(2,:));
93  hold on;
94  plot(t, x_e(2,:));
95  hold off;
96  xlabel('t');
97  ylabel('x_2');
98  legend('$x_2(t)$', '$\hat{x}_2(t)$', 'Interpreter','latex');
99  grid('on');
100
101  disp(size(u));
```

```

102 disp(size(t));
103 figure(1);
104 subplot(3, 1, 3);
105 plot(t, u);
106 xlabel('t');
107 ylabel('u');
108 legend('$u(t)$', 'Interpreter', 'latex');
109 grid('on');

```

b) The results are given in Figure 4.

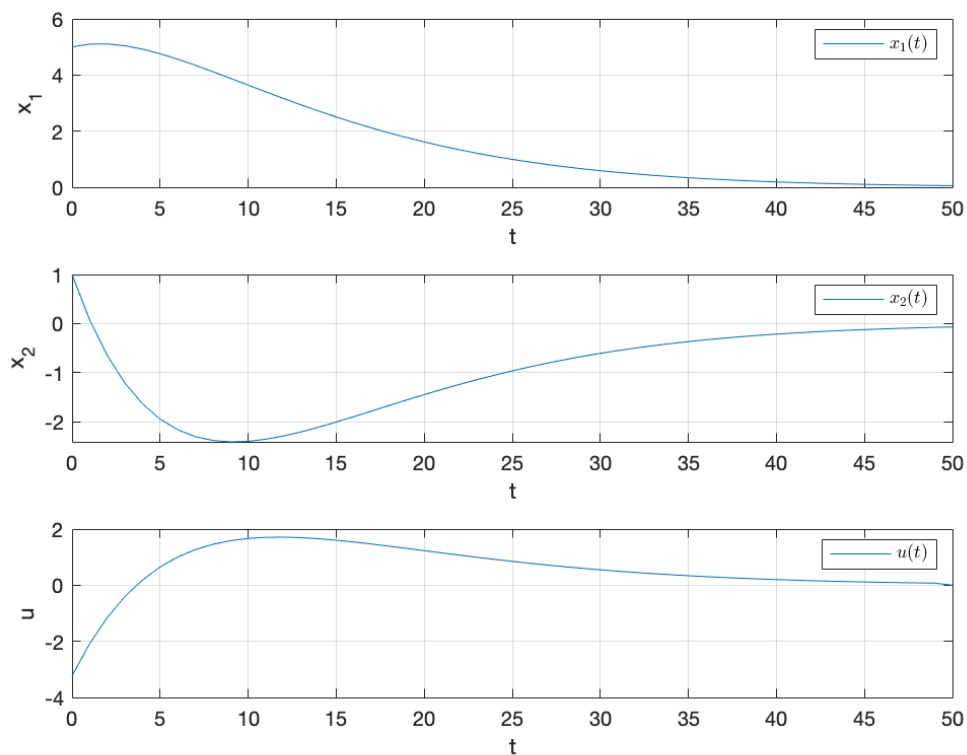


Figure 3: MPC with state feedback.

4 a)  $P$  was gotten in Problem 1:

$$P = \begin{bmatrix} 27.5170 & 7.2713 \\ 7.2713 & 10.2339 \end{bmatrix}$$

b) The code used is shown below.

```

1 Qt = [
2     4 0
3     0 4
4 ];
5

```

```
6 A = [  
7     1  0.1  
8    -0.1  1-0.1  
9  ];  
10  
11 B = [  
12     0  
13    0.1  
14  ];  
15  
16 C = [  
17     1  
18     0  
19  ];  
20 Rt = 1;  
21  
22 x0 = [5  1]';  
23  
24 N = 1;  
25 nx = 2;  
26 nu = 1;  
27 r = 1;  
28  
29 % Set G (objective function)  
30  
31 [K, S, e] = dlqr(A, B, Qt/2, Rt/2, []);  
32 disp(S);  
33 Q = kron(eye(N),Qt);  
34 Q(N*nx-1:N*nx, N*nx-1:N*nx) = S;  
35 disp(Q);  
36 Rt = r;  
37 R = kron(eye(N),Rt);  
38 G = blkdiag(Q, R);  
39  
40  
41  
42 % Set equality constraints (Aeq and Beq)  
43 Beq = zeros(N*nx, 1);  
44  
45 Aeq_1 = eye(N*nx);  
46 Aeq_2 = kron(diag(ones(N-1,1),-1),-A);  
47 Aeq_3 = kron(eye(N), -B);  
48 Aeq = [Aeq_1 + Aeq_2, Aeq_3];  
49  
50 u_low = -4;  
51 u_high = 4;  
52  
53 x_high = inf;  
54 x_low = -inf;
```

```
55
56 lb = [x_low*ones(N*nx, 1); u_low*ones(N*nu, 1)];
57 ub = [x_high*ones(N*nx, 1); u_high*ones(N*nu, 1)];
58
59 x = zeros(nx,N);
60 x(:,1) = x0;
61
62 u = zeros(nu, 51);
63
64 for t = 1:50
65     Beq(1:nx) = A*x(:,t);
66     z = quadprog(G, [], [], [], Aeq, Beq, lb, ub);
67     u(t) = z(N*nx+1); % get first element of u as actual
        control input
68     disp(u(t));
69     x(:,t+1) = A*x(:,t) + B*u(t); % state space update
70 end
71
72 figure(1);
73 subplot(3, 1, 1);
74 t = 0:50;
75 plot(t, x(1,:));
76 xlabel('t');
77 ylabel('x_1');
78 legend('$x_1(t)$', 'Interpreter', 'latex');
79 grid('on');
80
81 figure(1);
82 subplot(3, 1, 2);
83 plot(t, x(2,:));
84 xlabel('t');
85 ylabel('x_2');
86 legend('$x_2(t)$', 'Interpreter', 'latex');
87 grid('on');
88
89 disp(size(u));
90 disp(size(t));
91 figure(1);
92 subplot(3, 1, 3);
93 plot(t, u);
94 xlabel('t');
95 ylabel('u');
96 legend('$u(t)$', 'Interpreter', 'latex');
97 grid('on');
```

The results are given in Figure ??.

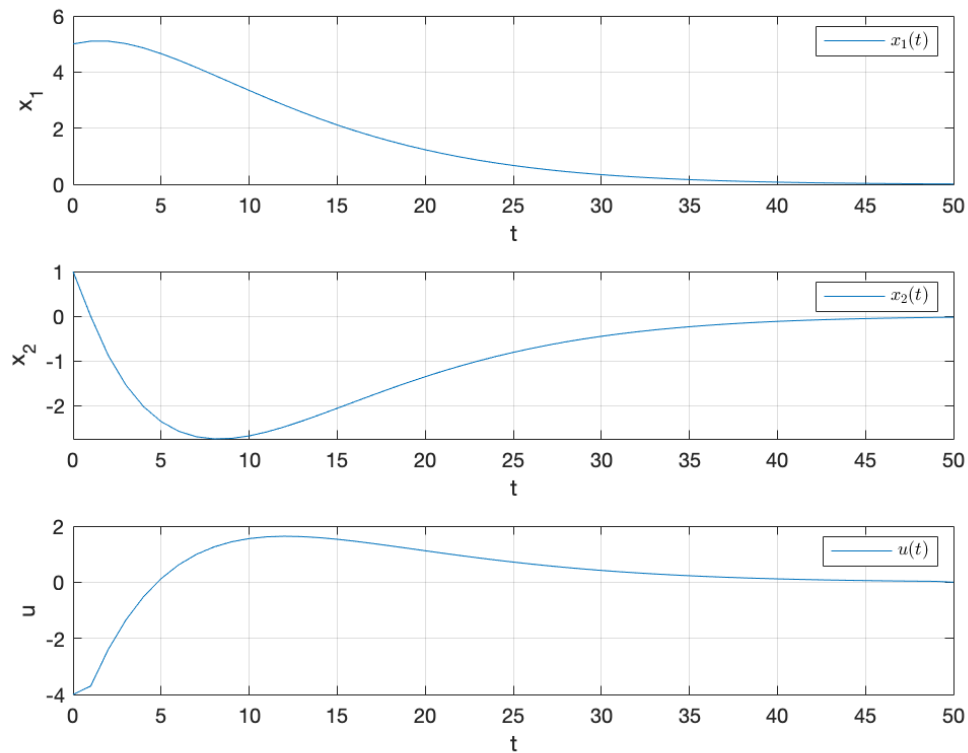


Figure 4: MPC with state feedback and infinite horizon.

We can see that this MPC converges slightly faster towards the origin than the MPC from Problem 3b. Changing  $N$  did not change the solution much, as far as my testing went (from values 1-30), the control input constraints were inactive except for the first few control inputs.  $N$  becomes more important for performance in realistic scenarios without state feedback and when there are clearer performance goals, like runtime or memory constraints.