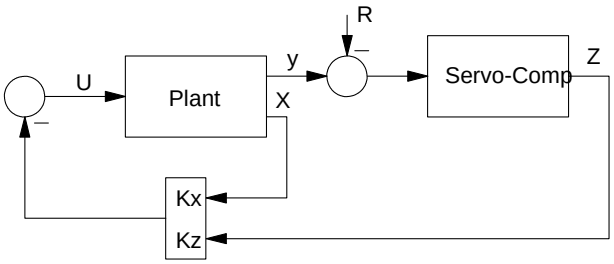
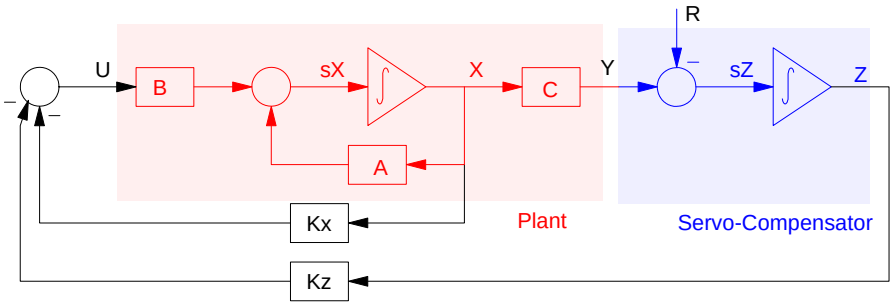
LQG Control with Servo Compensators

Servo Compensator:



Track a constant set-point:



$$s\begin{bmatrix} X \\ Z \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & 0 \end{bmatrix}\begin{bmatrix} X \\ Z \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix}U + \begin{bmatrix} 0 \\ -1 \end{bmatrix}R$$

$$U = -\begin{bmatrix} K_x & K_z \end{bmatrix}\begin{bmatrix} X \\ Z \end{bmatrix}$$

$$s\begin{bmatrix} X \\ Z \end{bmatrix} = \begin{bmatrix} A - BK_x & -BK_z \\ C & 0 \end{bmatrix}\begin{bmatrix} X \\ Z \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}R$$

Find the feedback gains so that the system has

- No error for a step input (assured with the use of a servo compensator)
- A 2% settling time of 4 seconds, and
- <4% overshoot for a step input

Since the servo-compensator state is just a dummy state of the controller, which you don't care that much about, make its weighting zero. Since you care about the output, y, design a LQR controller where

$$y = x_4 = \left[\begin{array}{cccc:c} 0 & 0 & 0 & 1 & 0 \end{array}\right] \left[\begin{array}{c} X \\ \cdots \\ Z \end{array}\right]$$

$$y = C_x X$$

$$Q = C_x^T C_x$$

In SciLab, this results in an error: the system is not observable from that output.

```
-->Q = 1 * Qy;
-->R = 1;
-->Kx = lqr(A, B, Q, R)
      !--error 998
      internal error, info=4.
```

When you get an error like that, the math is trying to tell you something. The challenge is trying to figure out what the math is trying to say....

Since the LQR method is complaining about Q, let's add an identity matrix to keep it happy. Make the weightings small since we don't care too much about the other states.

$$Q = C_x^T C + 10^{-3} I$$

The resulting control gains and step response are as follows:

```
-->Q = C5'*C5 + eye(5,5) * 1e-3

    0.001    0.       0.       0.       0.
    0.       0.001    0.       0.       0.
    0.       0.       0.001    0.       0.
    0.       0.       0.       1.001    0.
    0.       0.       0.       0.       0.001

-->Kx = lqr(A, B, Q, R)

    0.0641290    0.1298143    0.1958329     0.2527241    0.0316228
```

Note that the gains are pretty small. The closed-loop dominant pole is also way too slow:

```
-->eig(A5 - B5*Kx)
```

```
  - 3.5319727
  - 2.3486196
  - 0.9901871
  - 0.1708104
  - 0.0225392      ( dominant pole:  should be around -1 )
```

To speed up the system, increase the weighting on y by 1000x

```
-->Q = 1000*C5'*C5 + eye(5,5) * 1e-3

    0.001    0.       0.       0.          0.
    0.       0.001    0.       0.          0.
    0.       0.       0.001    0.          0.
    0.       0.       0.       1000.001    0.
    0.       0.       0.       0.          0.001

-->Kx = lqr(A, B, Q, R)

    1.6015116    4.4849429    9.4975698     15.097135    0.0316228
```

The gains are bigger - which is expected when you increase the weightings on Q. The dominant pole got *worse* however:

```
-->eig(A5 - B5*Kx)

  - 3.1945132
  - 3.1689866
  - 1.1185062 + 1.3690332i
  - 1.1185062 - 1.3690332i
  - 0.0009995         ( dominant pole:  worse than before )
```

As you increase the weightings on the output (y), it keeps getting slower and slower - exactly opposite of what you'd expect.

Compensator Design (take 2)

What's happening is this. The math is dumb: it doesn't know what the system output is. What the system *looks* like mathematically is a 5th order system with the states being

something - something - something - $\frac{dz}{dt}$ - z

Initially, when we set the weight on the servo-state (z) to zero, the system was unobservable through that output. That makes some sense: if you just watch the speed of your car, you can't determine your position.

With Q, we were telling the control law that you want the system to stabilize while keeping $\frac{dz}{dt}$ small. By increasing Q, we were penalizing $\frac{dz}{dt}$ more and more, resulting in a slower system (meaning $\frac{dz}{dt}$ was made smaller and smaller like it was supposed to.)

So, instead of weighting the actual system's output, treat the servo-state as the system's output. Then,

- Increasing $Q = C_z^T C_z$ should act like a stiffer spring, speeding up the system
- Increasing $Q = C_x^T C_x$ should act like more friction, reducing the overshoot and slowing down the system

In Matlab: Start with Cz and Cx:

```
-->Cz = [0,0,0,0,1]

    0.    0.    0.    0.    1.

-->Cx = [0,0,0,1,0]

    0.    0.    0.    1.    0.
```

and the corresponding Q matrices:

```
-->Qz = Cz'*Cz;
-->Qx = Cx'*Cx
```

First guess, let Q = Qz

```
-->Q = Qz
-->Kx = lqr(A, B, Q, R)
-->eig(A5 - B5*Kx)

  - 3.5321003
  - 2.3470723
  - 1.0094503
  - 0.2426534 + 0.2462031i     dominant poles
  - 0.2426534 - 0.2462031i
```

The dominant poles are too slow (the real part is smaller than -1). To speed up the system, increase Q to 100:

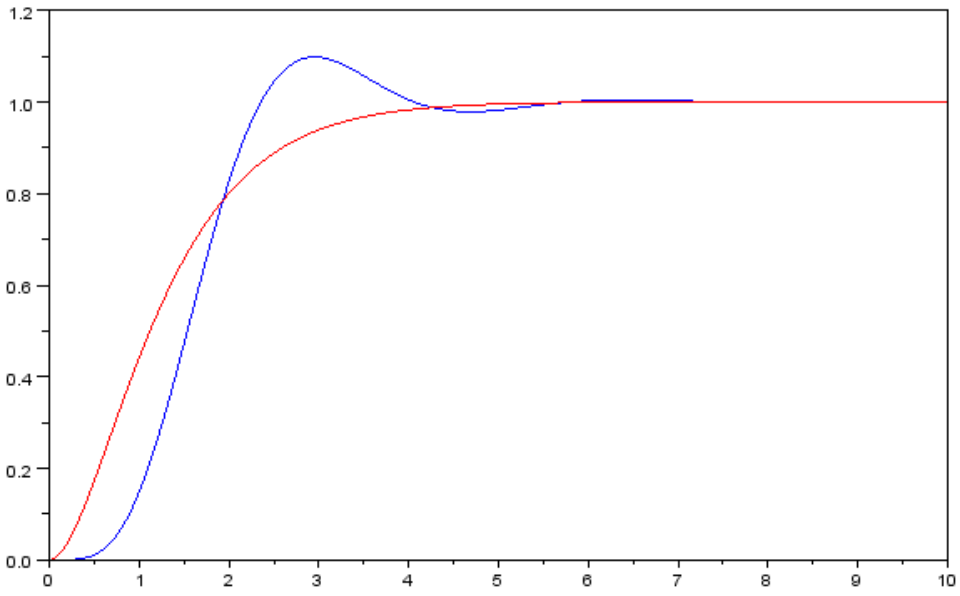
```
-->Q = Qz*100;
-->Kx = lqr(A, B, Q, R)
-->eig(A5 - B5*Kx)

  - 3.5332243
  - 2.3236428
  - 1.3382872
  - 0.5353237 + 0.7896661i
  - 0.5353237 - 0.7896661i
```

This is closer to a 4 second settling time. Increase Q even more:

```
-->Q = Qz*1e4;
-->Kx = lqr(A, B, Q, R)
-->eig(A5 - B5*Kx)

  - 3.6206066
  - 0.8799431 + 1.801827i
  - 0.8799431 - 1.801827i
  - 2.4786032 + 0.8518153i
  - 2.4786032 - 0.8518153i
```

Now the system is fast enough. The complex poles result in too much overshoot, however:

```
G = ss(A5-B5*Kx,Br,C5,0);
y = step(G,t);
plot(t,y,'b',t,Yd,'r');
```

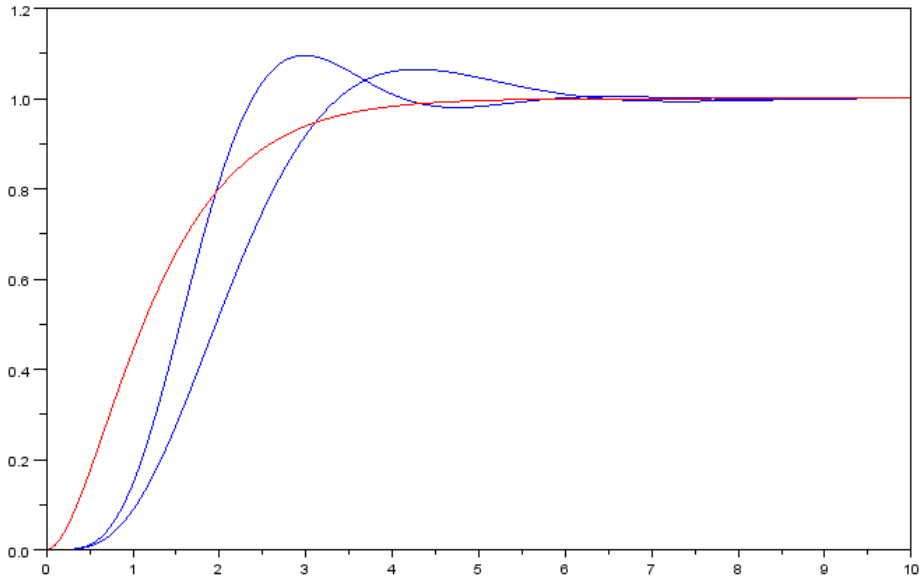


Desired Step Response (red) and Actual Step Response (blue)

To get rid of the overshoot, increase the weighting on $\frac{dz}{dt}$ (which is state y)

```
-->Q = Qz*1e4 + Qy*1e0;
-->Kx = lqr(A, B, Q, R)

-->eig(A5 - B5*Kx)
  - 3.6564395
  - 0.8868360 + 1.767468i
  - 0.8868360 - 1.767468i
  - 2.5146141 + 0.8188981i
  - 2.5146141 - 0.8188981i

-->G = ss(A5-B5*Kx,Br,C5,0);
-->y = step(G,t);
-->plot(t,y,'b',t,Yd,'r');

-->Q = Qz*1e4 + Qy*1e2;
-->Kx = lqr(A, B, Q, R)
-->eig(A5 - B5*Kx)

  - 10.304043
  - 3.2092893
  - 1.8753923
  - 0.7474964 + 1.0265077i
  - 0.7474964 - 1.0265077i

-->G = ss(A5-B5*Kx,Br,C5,0);
```

```
-->y = step(G,t);
-->plot(t,y,'b',t,Yd,'r');
```



Desired Step Response (red) and Actual Step Response (blue). Increasing the weight on dz/dt (Qy) reduces the overshoot

Iterating between

- Increasing the weight on z to speed up the system and
- Increasing the weight on dz/dt to reduce the overshoot

should eventually get closer and closer to the desired response.