LQG Observers

Recall that if you have a dynamic system

$$sX = AX + BU$$

$$Y = CX$$

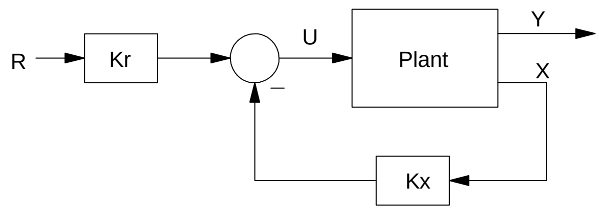
you can stabilize the system using full-state feedback

$$U = -K_x X + K_r R$$

resulting in the closed-loop system being

$$sX = (A - BK_x)X + (BK_r)R$$

The feedback gains, Kx, can be computed using Bass-Gura (pole placement) or LQR methods.



Full-State Feedback to Stabilize the System

One problem with these methods is you need to measure all of the states. If you can't measure certain states, you can estimate them using a full-order observer

$$s\hat{X} = A\hat{X} + BU + H\left(Y - \hat{Y}\right)$$

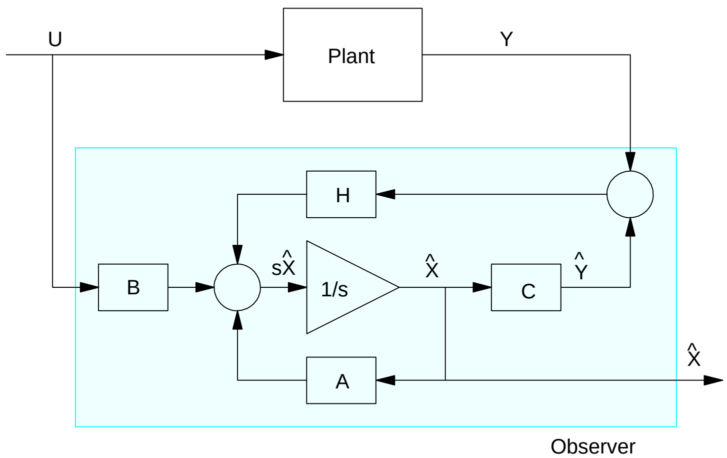
$$\hat{Y} = C\hat{X}$$

where H is the observer gain matrix choses to stabilize the observer dynamics

$$E = X - \hat{X}$$

$$sE = (A - HX)E$$

The observer gain, H, can then be found using Bass-Gura (pole placement) to stabilize (A - HC).



If the states are not measured, they can be estimated with a full-order observer

A strength of Bass-Gura is you can place the poles wherever you like. This is also its weakness - you don't know where the poles *should* be placed. One way around this is to determine H using LQR methods.

LQR methods find the gain, Kx, to place the poles of

$$A - BK_x$$

Here, in contrast, we are finding H to place the poles of

$$A - HC$$

If you transpose a matrix, the eigenvalues don't change. If you transpose (A-HC), you get

$$A^T - C^T H^T$$

This is then in the same form as (A - BKx) only

A is replaced with A^T

B is replaced with C^T, and

The gain you compute is H^T

However you come up with the observer gains, the resulting plant plus observer dynamics become:

$$s\begin{bmatrix} X \\ \hat{X} \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix}\begin{bmatrix} X \\ \hat{X} \end{bmatrix} + \begin{bmatrix} B \\ B \end{bmatrix}U + \begin{bmatrix} 0 \\ H \end{bmatrix}\left(Y - \hat{Y}\right)$$

or

$$s\begin{bmatrix} X \\ \hat{X} \end{bmatrix} = \begin{bmatrix} A & 0 \\ HC & A - HC \end{bmatrix}\begin{bmatrix} X \\ \hat{X} \end{bmatrix} + \begin{bmatrix} B \\ B \end{bmatrix}U$$

Example: Metal Bar (4th order RC filter). Design a full-order observer for the 4th-order heat equation.

First, pick Q and R. After some trial and error, let

- Q = I
- R = 1

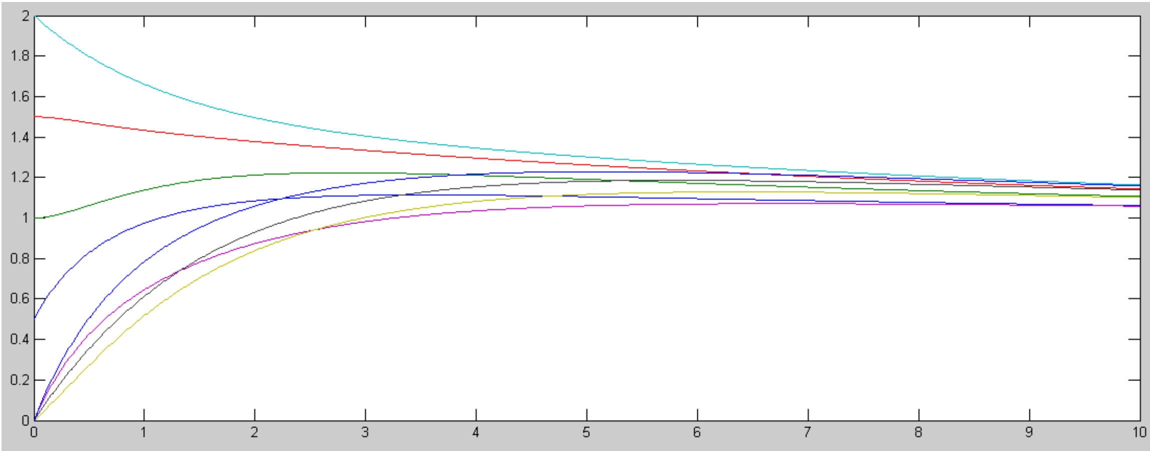
```
>> Q = diag([1,1,1,1]);
>> R = 1;
>> H = lqr(A', C', Q, R)'

    0.3713
    0.3116
    0.2826
    0.2705

>> eig(A - H*C)

   -3.5606
   -2.4418
   -1.1441
   -0.2248           Dominant pole for the observer
```

The augmented system (plant plus observer) is then

```
>> A8 = [A, zeros(4,4);H*C, A-H*C]
>> B8 = [B; B];
>> C8 = eye(8,8);
>> D8 = zeros(8,1);
```

Add initial conditions so you can see the observer states converge to the plant states:

```
>> X0 = [0.5;1;1.5;2; 0;0;0;0]

    0.5000
    1.0000     initial conditions of the plant
    1.5000
    2.0000
  - - - - - -
         0
         0     initial conditions of the observer
         0
         0

>> y8 = step2(A8, B8, C8, D8, X0, t);
>> plot(t,y8)
```



Plant and Observer States for Q = I, R = 1. Note that the states converge

To speed up the observer, increase the weightings. Seeing which state works best:

Weight on X1 = 1000:

```
>> Q = diag([1000,1,1,1]);
>> R = 1;
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -1.1826 + 1.3055i        Dominant pole for the observer:  6x faster
  -1.1826 - 1.3055i
  -3.2132 + 0.1366i
  -3.2132 - 0.1366i
```

Weight on X2 = 1000:

```
>> Q = diag([1,1000,1,1]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -4.1205
  -2.0341
  -1.8726 + 2.0161i        Dominant pole:  8x faster
  -1.8726 - 2.0161i
```

Weight on X3 = 1000:

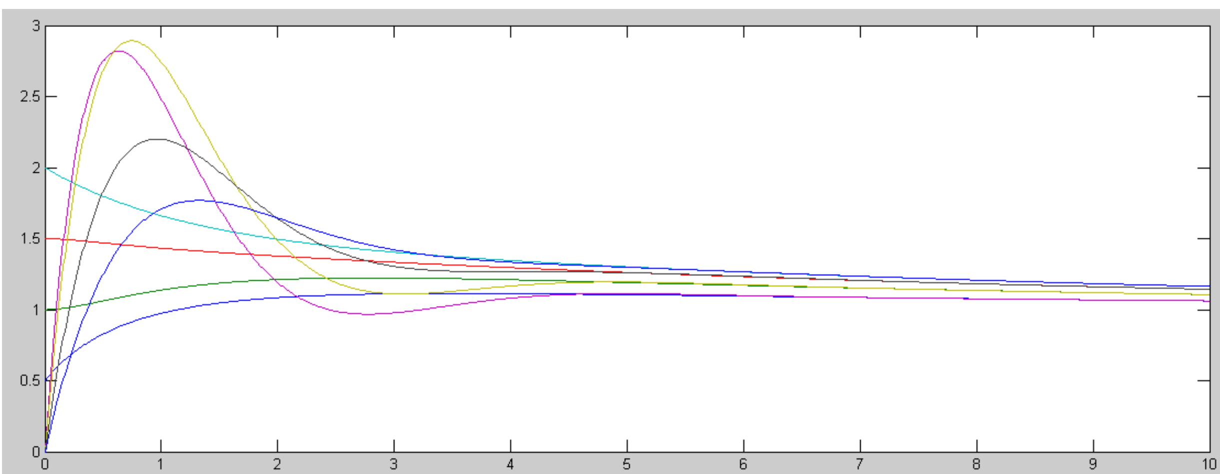
```
>> Q = diag([1,1,1000,1]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -0.9989                  Dominant pole:  4.5x faster
  -3.0160
  -4.2708 + 3.6451i
  -4.2708 - 3.6451i
```

Weight on X4 = 1000:

```
>> Q = diag([1,1,1,1000]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

   -0.5857                  dominant pole:  2x faster
   -1.9988
   -3.4132
  -31.6704
```

Looks like weighting X2 works best, so go with that. (Q is somewhat arbitrary - whatever seems to work)

```
>> Q = diag([1,1000,1,1]);
>> R = 1;
>> H = lqr(A', C', Q, R)'

    5.4325
    4.9030
    2.8963
    1.7915

>> A8 = [A, zeros(4,4);H*C, A-H*C];
>> B8 = [B; B];
>> C8 = eye(8,8);
>> D8 = zeros(8,1);
>> y8 = step2(A8, B8, C8, D8, X0, t);
>> plot(t,y8);
```



Step Response with Iniital Conditions: Q = diag(1 1000 1 1) R = 1

With this method, Q and R are just tools to use to adjust the observer dynamics. The resulting gains tend to be smaller than you'd get with Bass-Gura (good). It's somewhat difficult to get a specific response, however.

Example 2: Gantry System: Output = Position

```
>> A = [0,0,1,0;0,0,0,1;0,-19.6,0,0;0,29.4,0,0]

         0         0    1.0000         0
         0         0         0    1.0000
         0  -19.6000         0         0
         0   29.4000         0         0

>> B = [0;0;1;-1]

     0
     0
     1
    -1

>> C = [1,0,0,0]

C =

     1     0     0     0
```

Q and R are tools you can adjust to get the response you want. Trying different weightings to see which ones speed up the system the most:

All Weights 1.000:

```
>> Q = diag([1,1,1,1]);
>> R = 1;
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -0.9872 + 0.4962i        slowest pole
  -0.9872 - 0.4962i
  -5.0521
  -5.7288
```

Weight position (x) with 1000:

```
>> Q = diag([1e3,1,1,1]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

 -31.6228
  -0.0380                  slowest pole - got worse
  -5.4224 + 0.0570i
  -5.4224 - 0.0570i
```

Weight angle with 1000:

```
>> Q = diag([1,1e3,1,1]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -9.6826
  -4.8534 + 6.3666i
  -4.8534 - 6.3666i
  -0.0569                  slowest pole - got worse
```

Weight velocity (sx) with 1000:

```
>> Q = diag([1,1,1e3,1]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -4.0134 + 3.9521i        slowest pole:  better!
  -4.0134 - 3.9521i
  -5.6431
  -5.1940
```

Weight angular velocity with 1000:

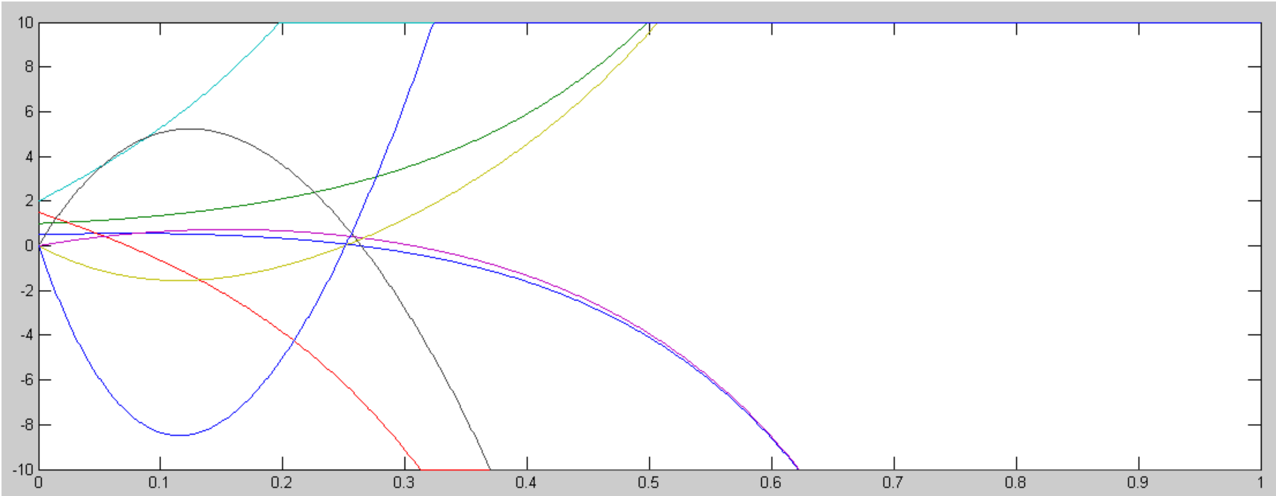
```
>> Q = diag([1,1,1,1e3]);
>> H = lqr(A', C', Q, R)';
>> eig(A - H*C)

  -2.2718 + 3.3054i        slowest pole
  -2.2718 - 3.3054i
  -6.0924 + 1.2055i
  -6.0924 - 1.2055i
```

Weighting velocity seems to work best, so let Q = diag([1, 1, 1000, 1])

```
>> Q = diag([1,1,1e3,1]);
>> R = 1;
>> H = lqr(A', C', Q, R)'

   18.8640
  -57.8416
  177.4249
 -313.5820

>> eig(A - H*C)

  -4.0134 + 3.9521i
  -4.0134 - 3.9521i
  -5.6431
  -5.1940
```

The plant is unstable, so the open-loop response will take off to infinity. Regardless, the observer states do converge - even if the plant isn't.

```
>> A8 = [A, zeros(4,4);H*C, A-H*C];
>> B8 = [B; B];
>> C8 = eye(8,8);
>> D8 = zeros(8,1);
>> X0 = [0.5;1;1.5;2; 0;0;0;0];
>> t = [0:0.001:1]';
>> y8 = step2(A8, B8, C8, D8, X0, t);
>> plot(t,min(10,max(-10, y8)))
```



Plant and Observer States for an Open-Loop Step Response:

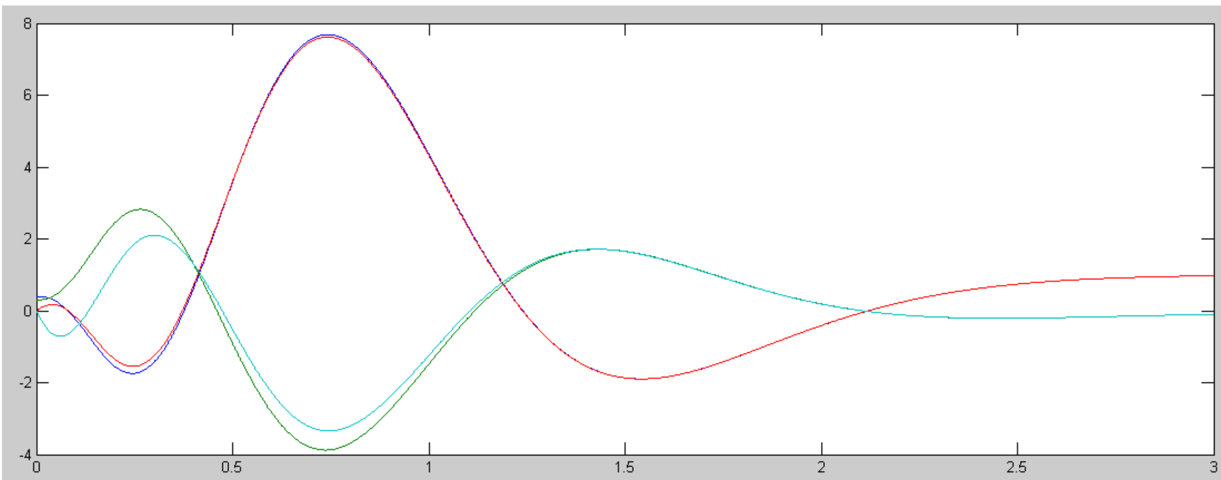
If you add full-state feedback, the system will be stabile and you can watch the observer converge:

```
>> Q = diag([1000,0,0,0]);
>> R = 1;
>> Kx = lqr(A, B, Q, R);
>> eig(A-B*Kx)

  -5.3273 + 2.4049i
  -5.3273 - 2.4049i
  -2.8173 + 1.0648i
  -2.8173 - 1.0648i

>> Kx

  -31.6228 -164.2922   -29.5050   -45.7942

>>
>> A8 = [A, -B*Kx   ;   H*C, A-B*Kx-H*C];
>> DC = -C*inv(A - B*Kx)*B

   -0.0316

>> Kr = 1/DC

  -31.6228
```

The combined system is then

$$\begin{bmatrix} sX \\ s\hat{X} \end{bmatrix} = \begin{bmatrix} A & -BK_x \\ HC & A - BK_x - HC \end{bmatrix} \begin{bmatrix} X \\ \hat{X} \end{bmatrix} + \begin{bmatrix} BK_r \\ BK_r \end{bmatrix} R$$

```
>> A8 = [A, -B*Kx;H*C, A-H*C-B*Kx];
>> B8 = [B*Kr; B*Kr];
>> C8 = [1,0,0,0,0,0,0,0;0,1,0,0,0,0,0,0;0,0,0,0,1,0,0,0;0,0,0,0,0,1,0,0];

     1     0     0     0     0     0     0     0     position
     0     1     0     0     0     0     0     0     angle
     0     0     0     0     1     0     0     0     position estimate
     0     0     0     0     0     1     0     0     angle estimate

>> D8 = zeros(4,1);
>> X0 = [0.4;0.3;0.2;0.1; 0;0;0;0];
>> y8 = step2(A8, B8, C8, D8, X0, t);
>> plot(t,min(10,max(-10, y8)))>>
```

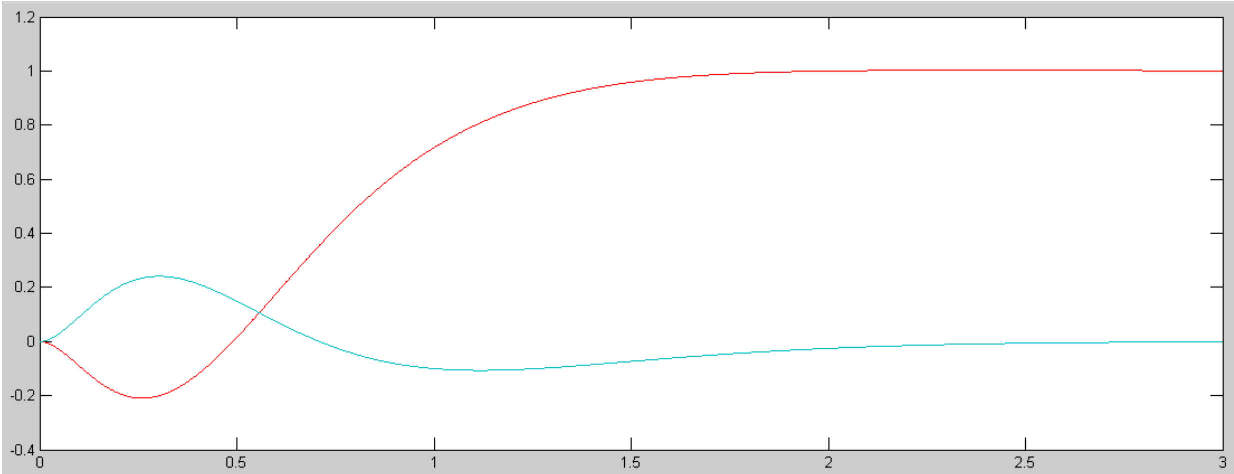


Step Response for the Plant, Observer, and Full-State Feedback with Errors in the Inital Estimates of the States

Note that the state estiamtes converge to the actual states. The system doesn't behave well, but it's at least not unstable.

Once the state estimates converge, (X0 is zero meaning no error in the state estimate) the system behaves much better:

```
>> y8 = step2(A8, B8, C8, D8, 0*X0, t);
>> plot(t,min(10,max(-10, y8)))
```



Step Response for the Plant, Observer, and Full-State Feedback with No Error in the State Estimates

Observers with Multiple Outputs:

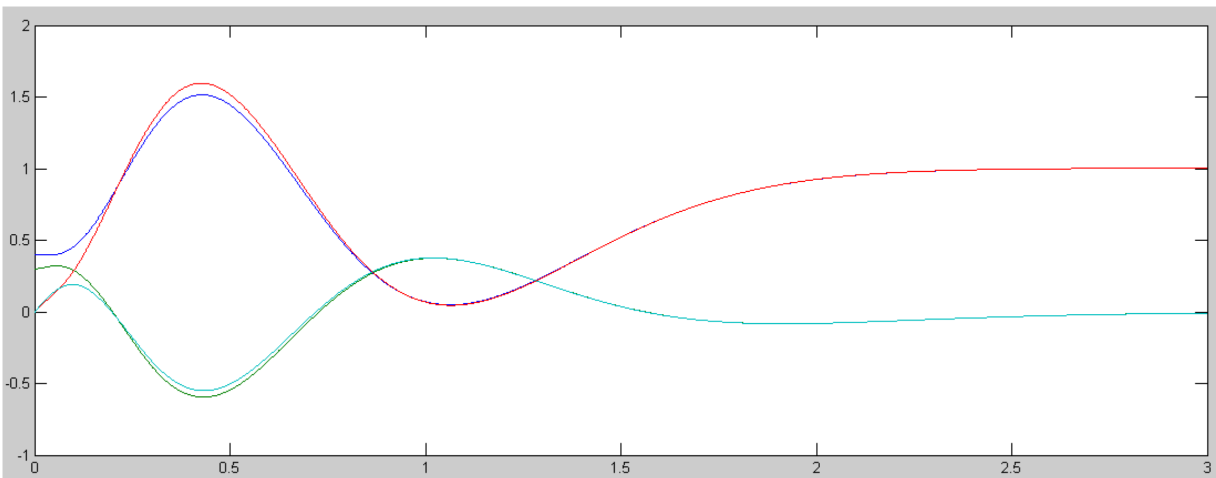
With LQR, you can use multiple outputs just as well. For example, assume you can measure poistion and angle:

```
>> C = [1,0,0,0 ; 0 1 0 0]

C =

     1     0     0     0         position (x)
     0     1     0     0         angle (q)

>> Q = diag([1,1,1e3,1e3]);
>> R = diag([1,1]);
>> H = lqr(A', C', Q, R)'

       x         q
    8.4497   -1.6848
   -1.6848   11.9071
   36.6179  -26.1793
   -8.1168   71.8085

>> eig(A - H*C)

  -3.9395 + 4.1662i
  -3.9395 - 4.1662i
  -6.2388 + 2.5857i
  -6.2388 - 2.5857i
```

Note that with LQR methods, you can handle two outputs without any problems: the observer gain, H, has two columns - one for position (x) and one for angle (q)

The step response of the plant plus observer plus full-state feedback is then:

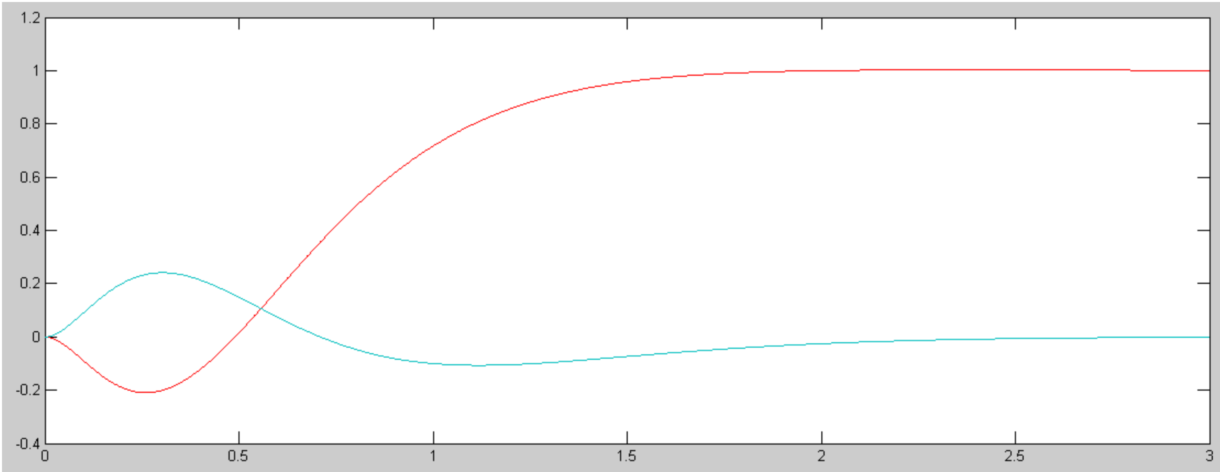
```
>> A8 = [A, -B*Kx;H*C, A-H*C-B*Kx];
>> B8 = [B*Kr; B*Kr];
>> C8 = [1,0,0,0,0,0,0,0;0,1,0,0,0,0,0,0;0,0,0,0,1,0,0,0;0,0,0,0,0,1,0,0]

C8 =

     1     0     0     0     0     0     0     0     position, x
     0     1     0     0     0     0     0     0     angle, q
     0     0     0     0     1     0     0     0     position estiamte xm
     0     0     0     0     0     1     0     0     angle estimate qm

>> D8 = zeros(4,1);
>> y8 = step2(A8, B8, C8, D8, X0, t);
>> plot(t,min(10,max(-10, y8)))
```



Position (blue and red) and angle (green teal) for the plant and observer with initial error in the state estimates:

If you remove the error, the tracking is better:

```
>> y8 = step2(A8, B8, C8, D8, 0*X0, t);
>> plot(t,min(10,max(-10, y8)))
```



Step response with no error in the initial state estiamtes: Position (red) and angle (green)

Note that H is

x	q	
8.4497	-1.6848	x update
-1.6848	**11.9071**	q update
36.6179	-26.1793	sx update
-8.1168	**71.8085**	sq update

As you would expect

- The position sensor mostly updates position and velocity estimates
- The angle sensor mostly updates angle and angular velocity estimates

Function Step2.m

```
function [ y ] = step2( A, B, C, D, X0, t )


T = t(2) - t(1);
[m, n] = size(C);

npt = length(t);

Az = expm(A*T);
Bz = B*T;

X = X0;

y = zeros(npt, m);

y(1,:) = (C*X + D)';

for i=2:npt
    X = Az*X + Bz;
    Y = C*X + D;

    y(i,:) = Y';

   end

end
```