Table of Contents


```
clear; close all
```

Define the system

Let's use the model of a motor with a flexible shaft connected to a load, with bearing damping.

```
J = 0.1;
k = 400;
B = 0.41e-3;
L = 10e-3;
R = 3;
km = 3.5;

A = ...
    [ ...
        -B/J                1/J; ...
        -k/(1+k*L/km^2)     -k*R/(km^2 + k*L); ...
    ];

B = ...
    [ ...
        0; ...
        k*km/(km^2 + k*L) ...
    ];

C = ...
    [ ...
        1   0; ...
        0   1 ...
    ];

D = ...
    [ ...
        0; ...
        0 ...
    ];
```

Define the system model

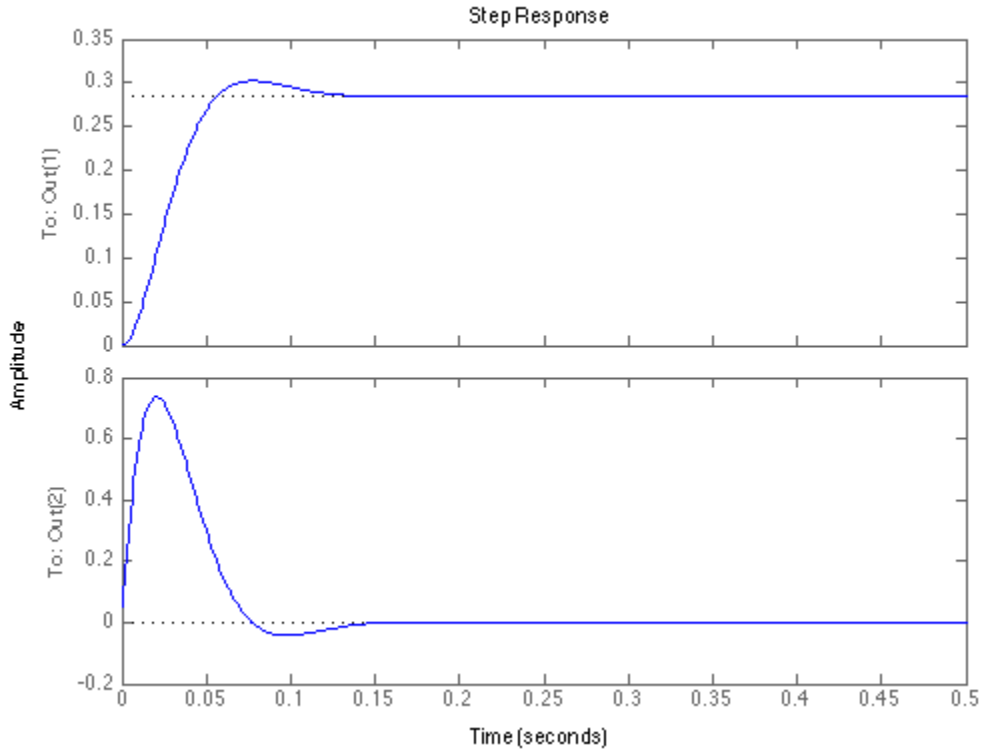
In matlab, we can define a system model with the ss command as follows.

```
sys = ss(A,B,C,D);

% Note: if you are unsure how to use this or any command, type
% "help <command>" to see the documentation.
```

Step response

Let's say we are interested in finding the step response for the system. We can use the step command to simulate this.

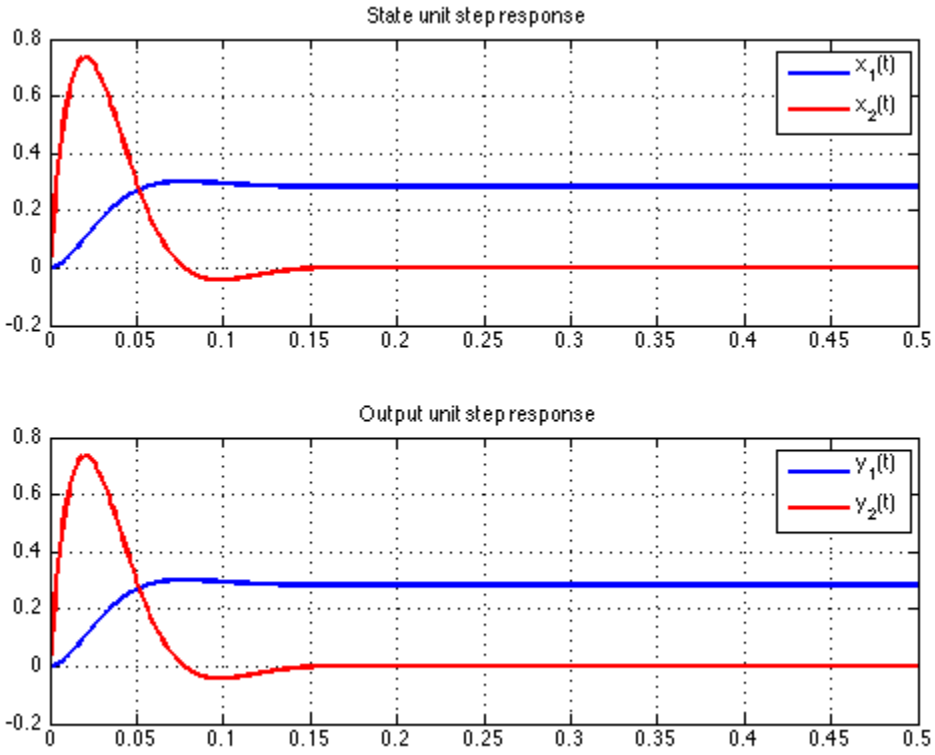
```
t = 0:0.001:0.5; % define a time vector
figure; step(sys,t)

% However, we might want to access the actual data from simulation. In this
% case, we might use the following call to step.
[ ys, tdum, xs ] = step(sys,t);

% This returns the time vector t,
% output matrix y (number of time steps by the output vector length), and
% state matrix x (number of time steps by the state vector length).
```



Plotting the step response

Let's plot the step response from the data.

```
figure;
subplot(2,1,1);
    plot( t, xs(:,1), 'b','Linewidth', 2 ); hold on
    plot( t, xs(:,2), 'r','Linewidth', 2 );
    title('State unit step response')
    legend('x_1(t)','x_2(t)')
    xlim([0 t(end)])
    grid on
subplot(2,1,2);
    plot( t, ys(:,1), 'b','Linewidth', 2 ); hold on
    plot( t, ys(:,2), 'r','Linewidth', 2 );
    title('Output unit step response')
    legend('y_1(t)','y_2(t)')
    xlim([0 t(end)])
    grid on
```

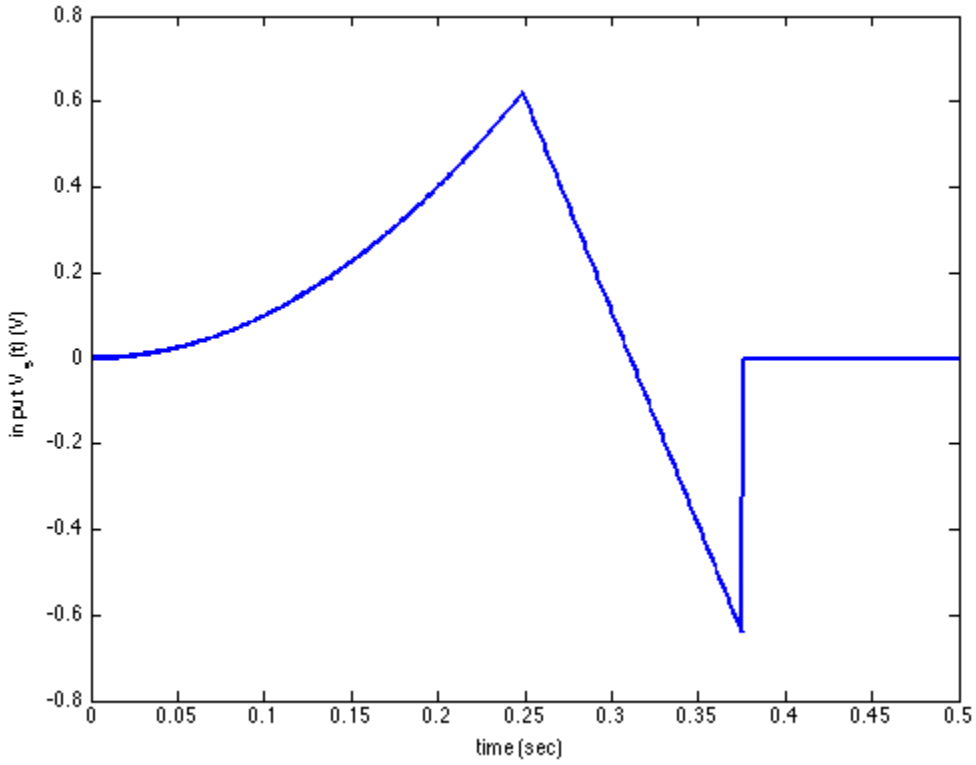


Response to an arbitrary input

Let's construct an input

```
u = zeros(size(t)); % initialize to zero
u(1:round(length(t)/2)) = 10*t(1:round(length(t)/2)).^2;
```

```
u(round(length(t)/2):round(3*length(t)/4)) = 3.11-10*t(round(length(t)/2):round(3*

% Plot the input
figure;
plot( t, u,'Linewidth', 2 );
xlabel('time (sec)')
ylabel('input V_s(t) (V)')

% Simulate the response
[ y, tdum, x ] = lsim( sys, u, t );

% Note: you can also add initial conditions to lsim.
```

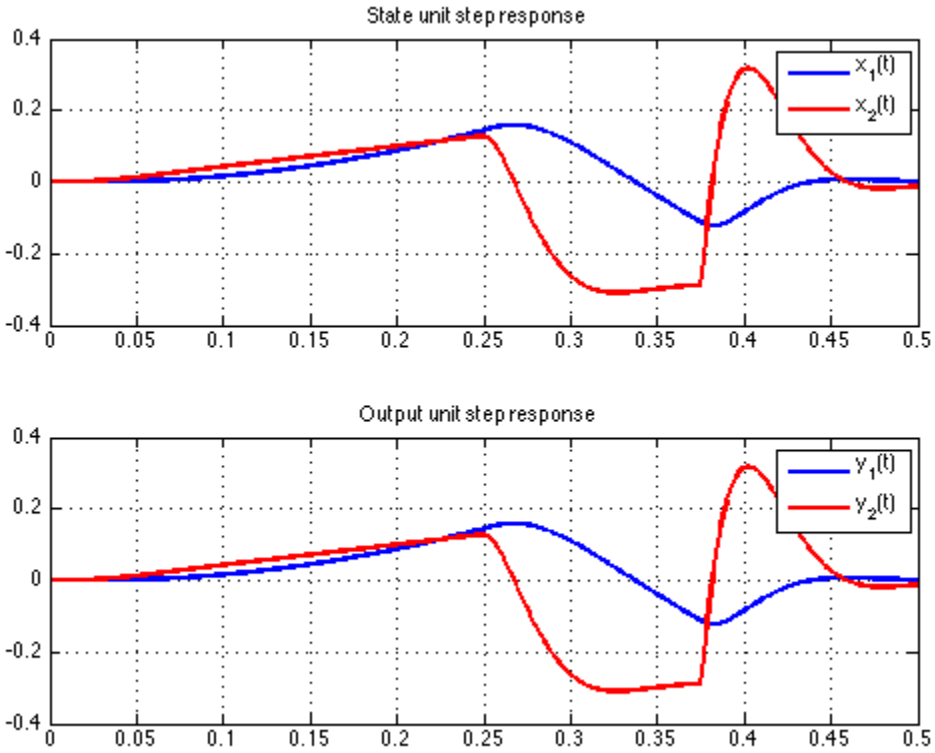


Plot the response to an arbitrary input

Since we've now called almost the same code in two places, we should probably just write a plotting function. I didn't want to make this confusing, though.

```
figure;
subplot(2,1,1);
    plot( t, x(:,1), 'b','Linewidth', 2 ); hold on
    plot( t, x(:,2), 'r','Linewidth', 2 );
    title('State unit step response')
    legend('x_1(t)','x_2(t)')
    xlim([0 t(end)])
    grid on
```

```
subplot(2,1,2);
    plot( t, y(:,1), 'b','Linewidth', 2 ); hold on
    plot( t, y(:,2), 'r','Linewidth', 2 );
    title('Output unit step response')
    legend('y_1(t)','y_2(t)')
    xlim([0 t(end)])
    grid on
```



Transfer function

We could also derive the transfer functions for the two outputs. This is easy enough.

```
[ num, den ] = ss2tf( A, B, C, D );

% There are two num rows, corresponding to the two outputs.
% There's only one den row, which is the denominator for all transfer
% functions.

% We can create a Matlab system object with the transfer function, as well.
systf1 = tf( num(1,:), den )
systf2 = tf( num(2,:), den )
```

```
systf1 =

         861.5
```

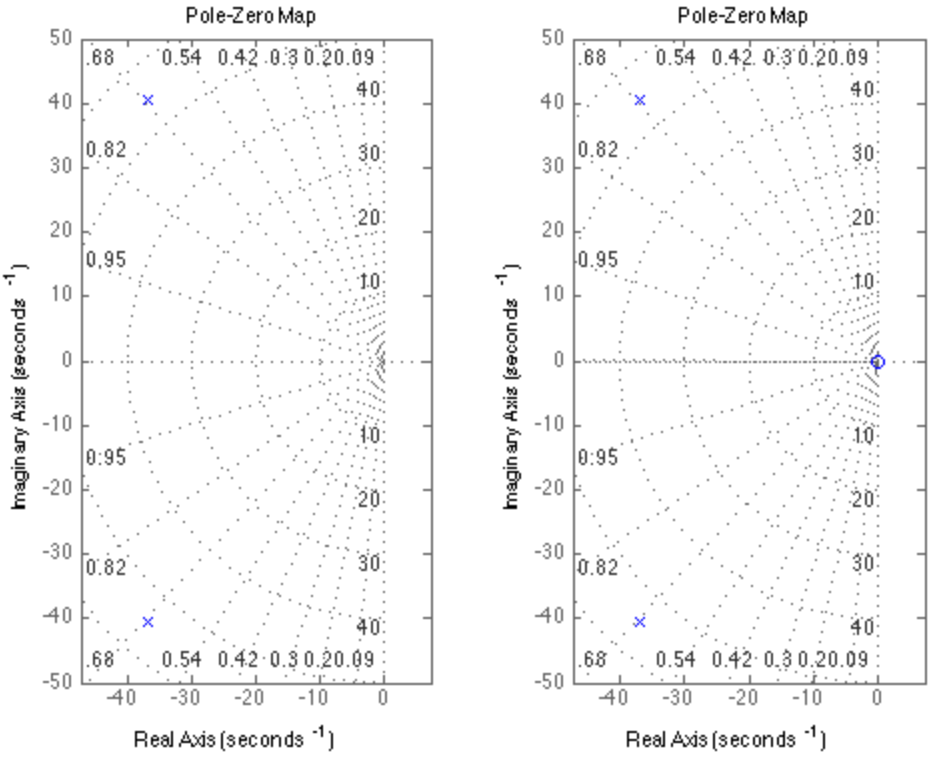
```
  --------------------
  s^2 + 73.85 s + 3016

Continuous-time transfer function.


systf2 =

    86.15 s + 0.3532
  --------------------
  s^2 + 73.85 s + 3016

Continuous-time transfer function.
```

Poles and zeros

Finding poles and zeros is easy, too.

```
% Poles for transfer functions:
polestf1 = pole(systf1)
polestf2 = pole(systf2)

% Poles for the state model:
polesss = pole(sys)

% Why are all three equal?

% Zeros and gains can be found in a similar way:
[ zerostf1, gtf1 ] = zero(systf1)
[ zerostf2, gtf2 ] = zero(systf2)
zerosss = zero(sys) % I'm not sure why this is only returning the (nonexistent) ze

% You may also do pole-zero plots
figure;
subplot(1,2,1)
    pzplot(systf1)
    axis equal
    zgrid % nice touch, shows natural frequency and damping ratio
subplot(1,2,2)
    pzplot(systf2)
    axis equal
    zgrid


    polestf1 =

     -36.9251 +40.6475i
     -36.9251 -40.6475i


    polestf2 =

     -36.9251 +40.6475i
```

```
 -36.9251 -40.6475i


polesss =

 -36.9251 +40.6475i
 -36.9251 -40.6475i


zerostf1 =

   Empty matrix: 0-by-1


gtf1 =

  861.5385


zerostf2 =

   -0.0041


gtf2 =

   86.1538

Warning: Use TZERO to compute the transmission zeros of a SISO or MIMO mod

zerosss =

   Empty matrix: 0-by-1
```



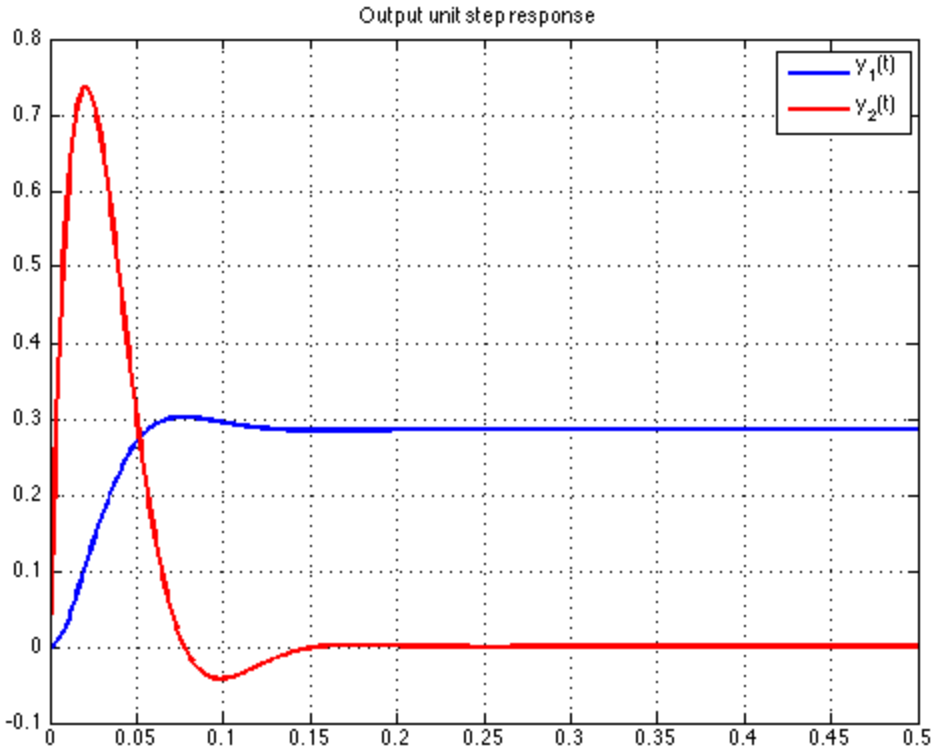
Diagonalization

We can also diagonalize the system and find it's state transition matrix.

```
% The eig function returns
% the eigenvalue matrix $\Lambda$ (which has the eigenvalues on the diagonal) and
% the modal matrix M (which has the corresponding eigenvectors as columns).
[ M, Lambda ] = eig(A)

% The state transition matrix is: (commented because it requires the
% symbolic toolbox, which I don't have atm.
% time = sym('time');
% Phi = expm( A * t )

% Transform to a basis that diagonalizes the A matrix:
Ap = inv(M)*A*M
Bp = inv(M)*B;
Cp = C*M;
Dp = D;

% Notice that $A' = \Lambda$:
Ap - Lambda % should be tiny

sysp = ss(Ap,Bp,Cp,Dp);

% Even though the system is defined by complex matrices, its model works
```

```
% just fine. For instance, let's examine the step response.
[ yps, tdum, xps ] = step(sysp,t);

figure;
plot( t, real(yps(:,1)), 'b','Linewidth', 2 ); hold on
plot( t, real(yps(:,2)), 'r','Linewidth', 2 );
title('Output unit step response')
legend('y_1(t)','y_2(t)')
xlim([0 t(end)])
grid on

% Note: taking the real parts was ok because the imaginary parts are
% virtually zero. The states are complex, though, so we cannot simply lop
% off the imaginary parts.
```

```
M =

  -0.1205 - 0.1326i  -0.1205 + 0.1326i
   0.9838 + 0.0000i   0.9838 + 0.0000i


Lambda =

 -36.9251 +40.6475i   0.0000 + 0.0000i
   0.0000 + 0.0000i -36.9251 -40.6475i


Ap =

 -36.9251 +40.6475i   0.0000 - 0.0000i
   0.0000 - 0.0000i -36.9251 -40.6475i


ans =

   1.0e-13 *

   0.0000 + 0.0000i   0.0711 - 0.0933i
   0.1066 - 0.0133i   0.0711 + 0.0711i
```



Published with MATLAB® R2013a