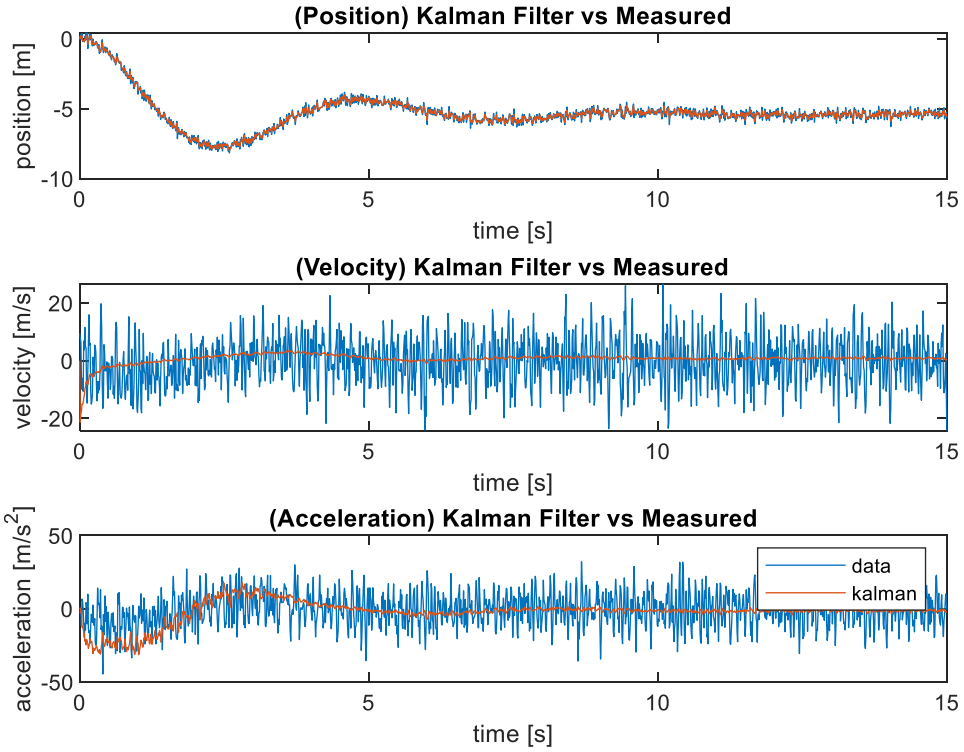
ME 494/5594 – Robotics Systems Identification

HW #8: Due Monday, November 28th, 2022

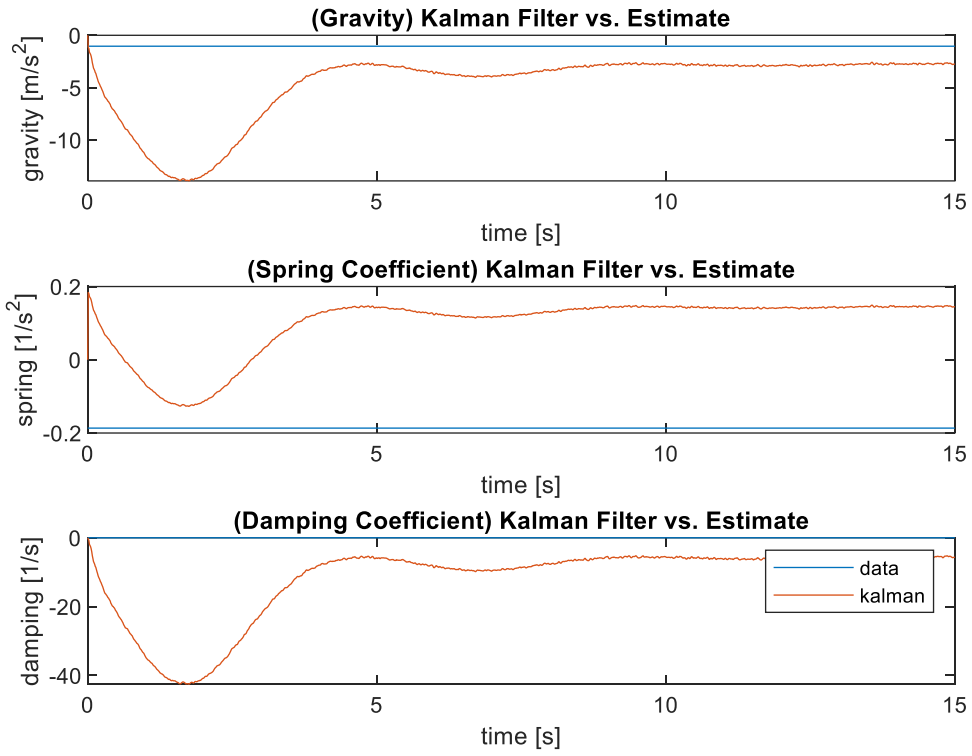
1. Using the Mass-Spring-Damper data on Canvas, create an Extended Kalman Filter that can be used to estimate the damping coefficient, spring coefficient, and gravity constant. Show the three state estimates (y, y_d, y_{dd}) as in HW #6 Problem 2. In a separate subplot (3 subplots in one figure) that show the three estimated coefficients/parameters vs. time.

   Start with poor initial parameter estimates (i.e. ~10% of the values in HW #6 Problem 2). Use the same Q (now a 6 x 6, same values in diagonal) and R matrices as in Problem 2.

   Provide a brief comment on your results (particularly on the x, xd, and xdd state estimation and the convergence/accuracy of the parameter estimates). Do your parameter estimates converge, or do they continue to vary? Explain what may be going on if they vary.



Using Q matrix values of .001 the position model follows the measured data nicely but not great for the velocity and acceleration models. The velocity and acceleration model does track nicely after about 5 seconds.



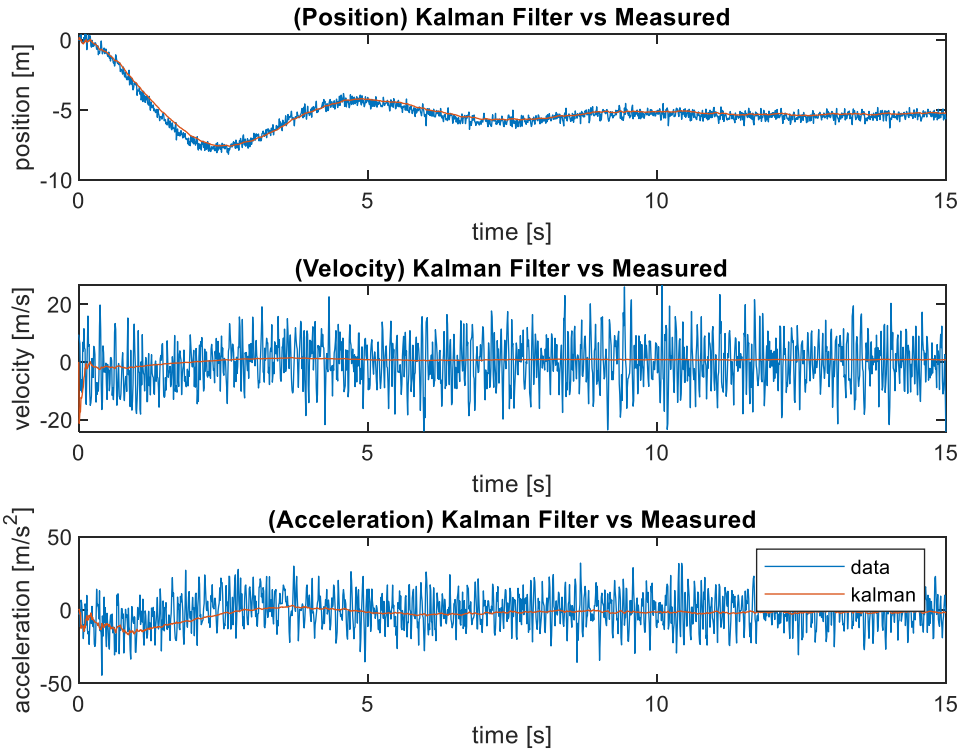
The estimates do converge but are not very accurate. The extended Kalman Filter trusted the poor initial guesses too much.

2. Using the EKF from Problem 1, modify the Q matrix to be 0.01 in the diagonal elements. Rerun your EKF and show the plots.

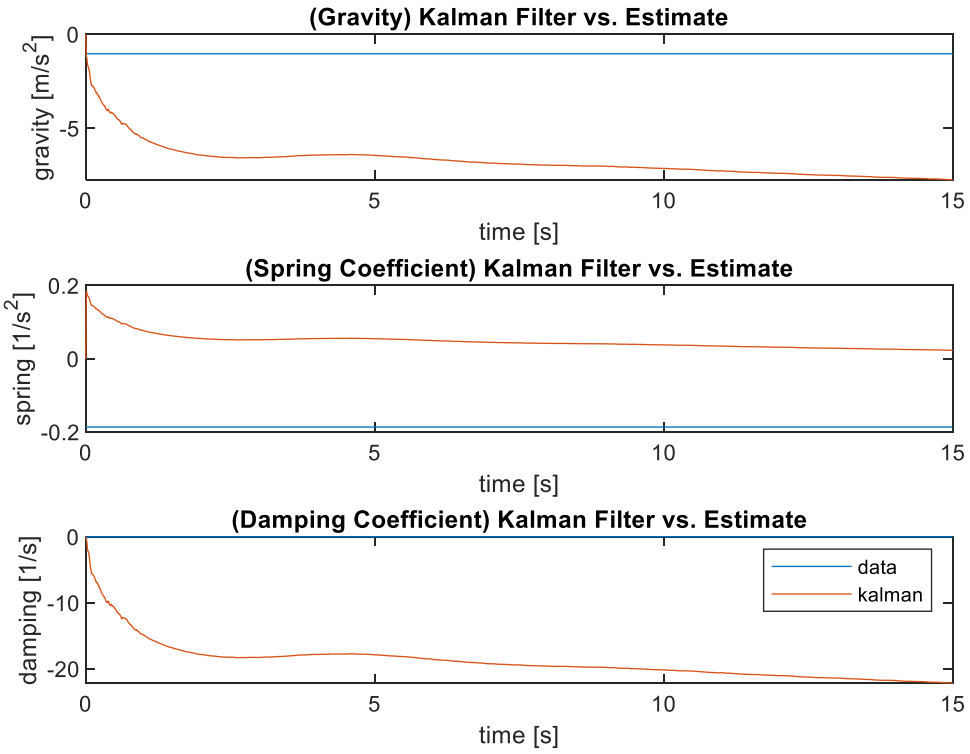
   Use a Q matrix with 0.000001 in the diagonal elements. Rerun your EKF and show the plots.

   Explain what you see in your state and parameter estimates compared to Problem 1.

   What is the purpose of Q? What is the purpose of R? This should be a short paragraph in length.



The state estimate tracks the position data ok but not as good as problem 1. The velocity estimate does seemingly track the data better but still poorly. The acceleration estimate tracks the measured data much better than the problem 1 estimate does.



The parameter estimates do not converge but are much closer to what they should be at 15 seconds when compared to problem 1. This is expected because the Q values were much smaller than problem one. The Q and R matrix maps the measured/estimated/modeled values to the Kalman Filter output. Smaller values in the R matrix makes the Kalman Filter trust the measured/estimated values more. Smaller values in the Q matrix makes the Kalman Filter trust the model more. Since the Q values were significantly smaller it makes sense the that estimates didn't quite converge. I would suspect that they would converge if the dataset was slightly bigger.

3. Using Qube_OpenLoop_HW7.mat, you need create the simplest, yet usable transfer function model. Use a discrete transfer function model. The data file contains an open loop (no control) frequency sweep for 120s (at 100 Hz).

   Use the output angular velocity (OL_Data(:,4)) and the input motor commands (OL_Data(:,2)) in your estimation.

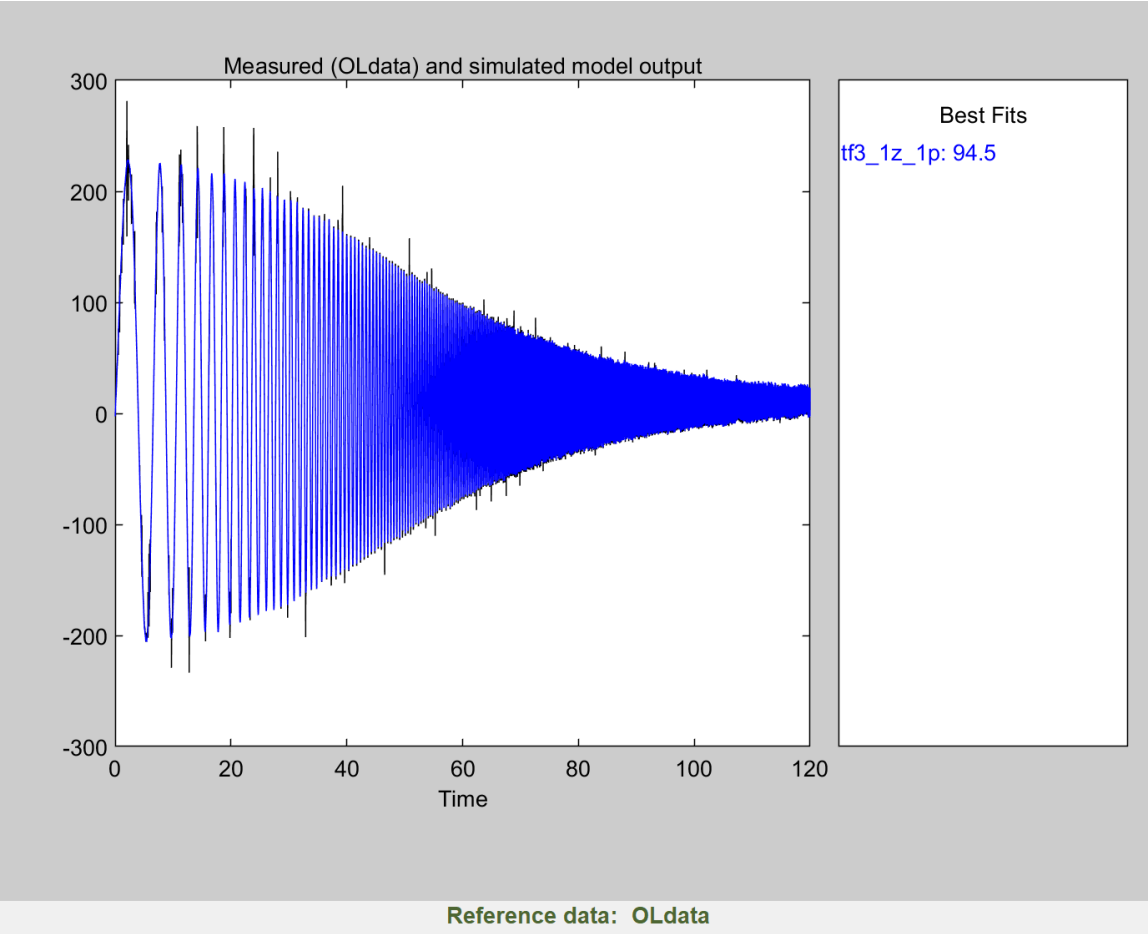
   Show the model fit, the transfer function, and the frequency response of the model.

   Create a Bode plot of the data (see Canvas for example code) with coherence.

   Comment on your results (including coherence from the Bode plot), are they reasonable for a DC motor? What is the cutoff frequency (where the attenuation

starts)? Is this reasonable? How does the model compare to the Bode plot (compare frequency response plots).

Model Fit:



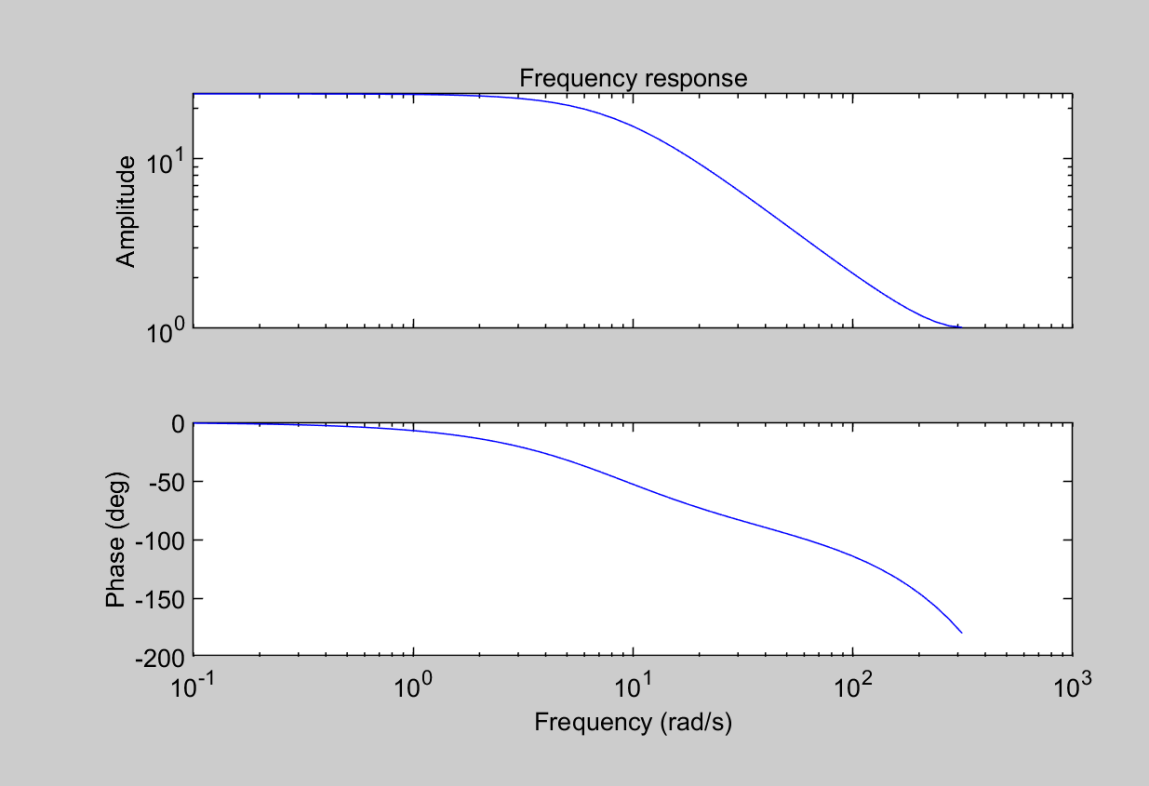
The model fit of 94.5% is great. I would suspect to have a good fit since there is such a simple system with minimal noise.

Transfer Function:

$$\frac{1.946\ z^{-1}}{1 - 0.9195\ z^{-1}}$$

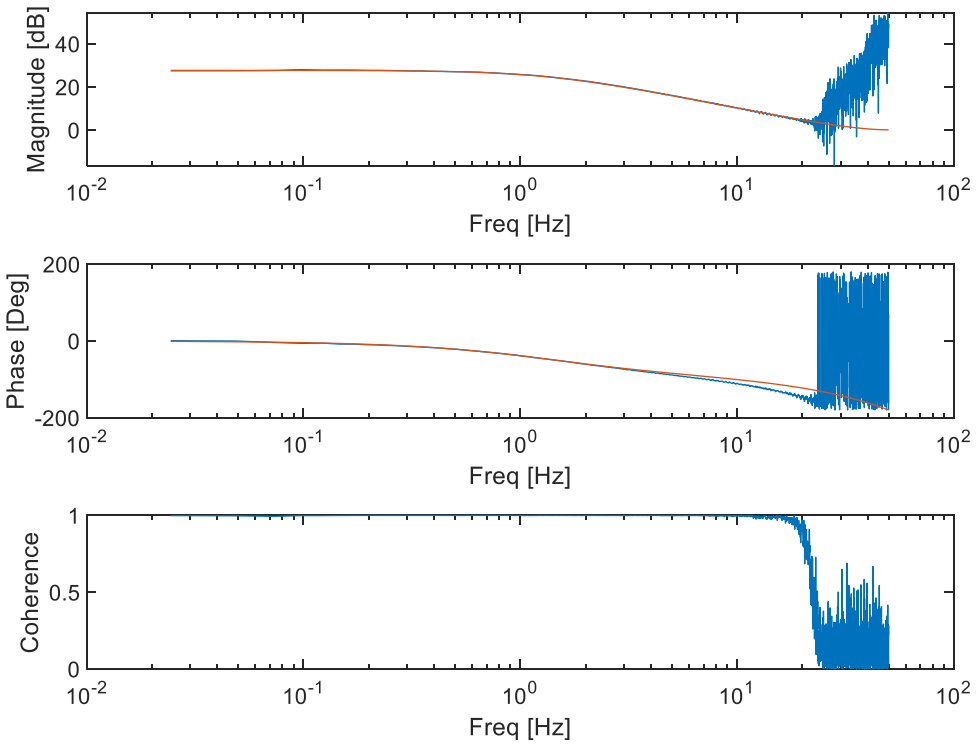
Having one pole and one zero. Creating a transfer function with 1 pole and no zeros produces a fit of 86.43 which is not great but might be considered to be sufficient depending on the application.

Frequency Response:



Input to output is equal until about 10 rad/s. At about 100 rad/s the input to output ratio is .1 and the signal is about 100 deg. Out of phase.

Bode Plot With Coherence:



The model is quite good as it follows the data nicely. The coherence plot shows that output is explained by the input up to about 20 hz. The cutoff frequency is about 20hz.

Coherence: Metric to examine how well output data is explained by input. We want Coh>0.7, if it drops below it, then cutoff data right at 0.7 in example, cutoff at beginning of descent.

4. Using Qube_ClosedLoop_HW7.mat, you need to create the simplest, yet usable transfer function. Use a discrete transfer function model. The data file contains a closed loop (active control) frequency sweep of the desired angle to actual angle for 120s (at 100 Hz).

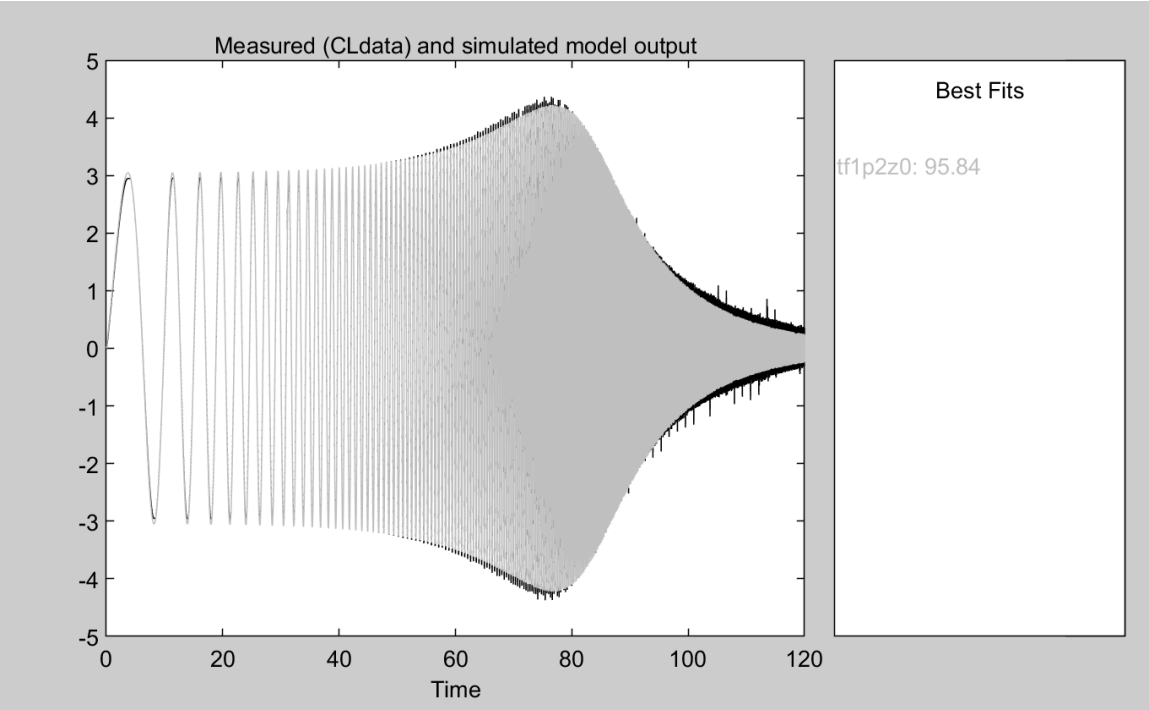
   Use the output angle (CL_Data(:,3)) and the input desired angle (CL_Data(:,6)) in your estimation.

   Create a Bode plot of the data (see Canvas for example code) with coherence.

   Show the model fit, the transfer function, and the frequency response of the model. Comment on your results, are they reasonable for an active control system on a DC motor? How do the results compare with Problem #1? What is the cutoff frequency

(where the attenuation starts)? Is this reasonable? How does the model compare to the Bode plot (compare frequency response plots).

Model Fit:



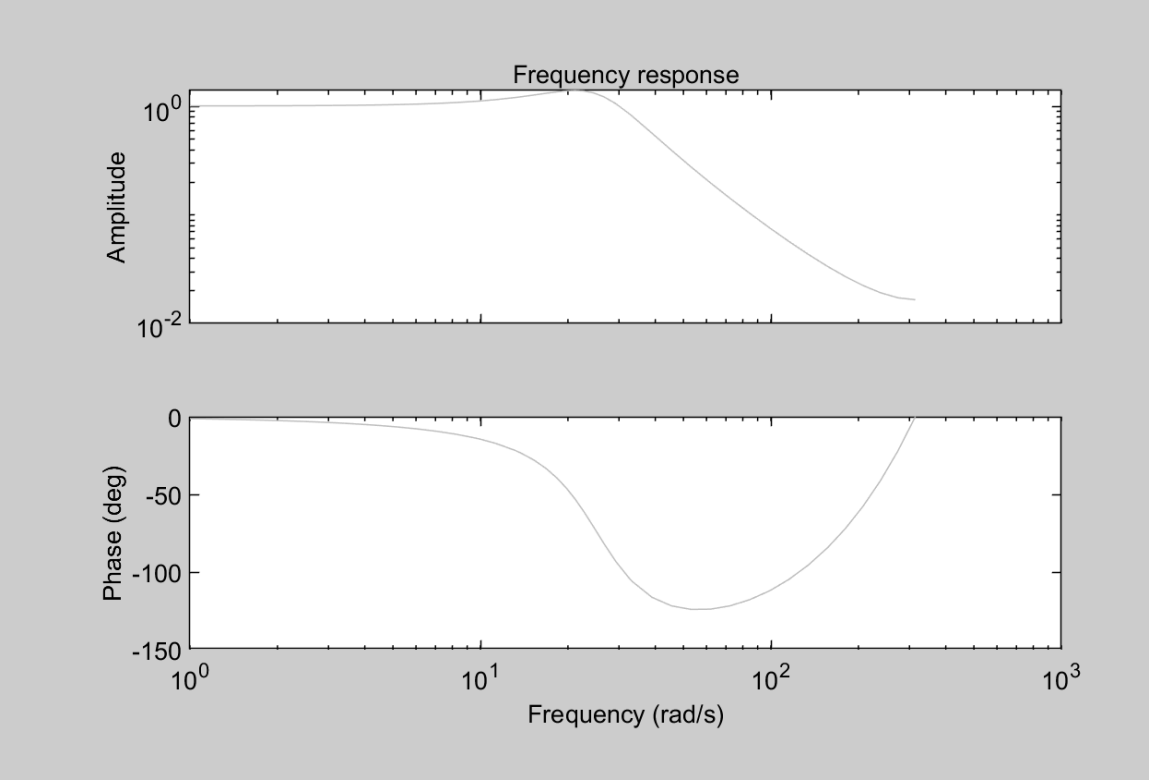
The model fit of 94.84% is great. I would suspect to have a good fit since there is such a simple system with minimal noise.

Transfer Function:

$$\frac{0.05918}{1 - 1.761\ z^{-1} + 0.8189\ z^{-2}}$$

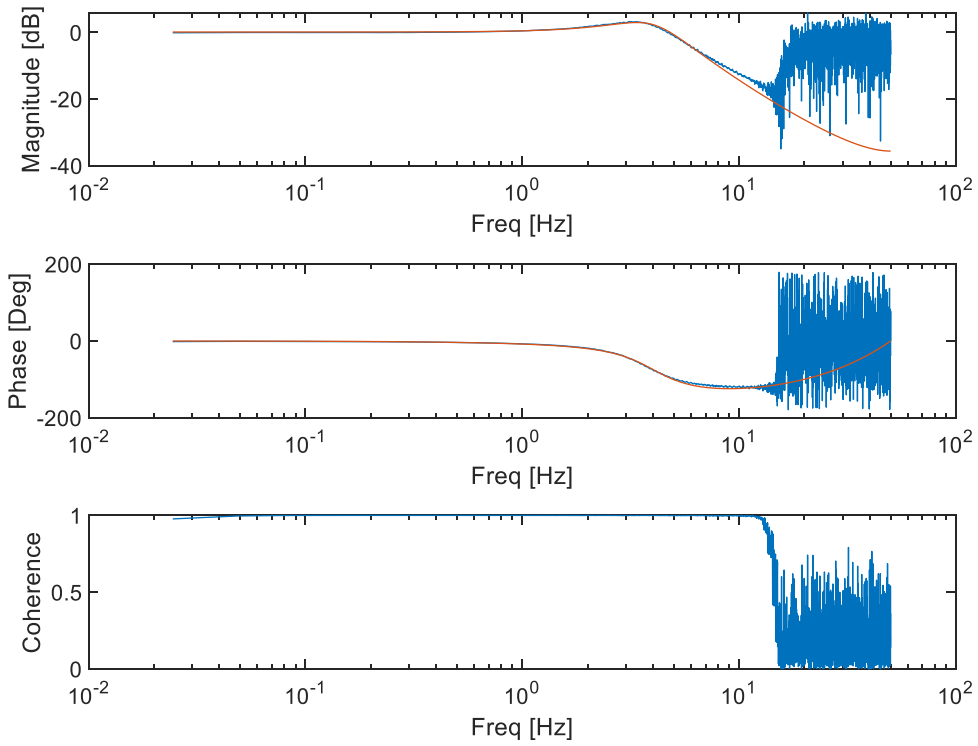
Having two poles and no zeros. Creating a transfer function with 1 pole and 1 zero produces a fit of 68.84 which is insufficient. Therefore, the simplest usable transfer function above was chosen.

Frequency Response:



Input to output is equal until about 25 rad/s. 20 rad/s is near resonance. At about 300 rad/s the input to output ratio is .01. The signal is out of phase between 10 and 300rad/s.

Bode Plot With Coherence:



The model is quite good as it follows the data nicely. The coherence plot shows that output is explained by the input up to about 15 hz. The cutoff frequency is about 15hz.

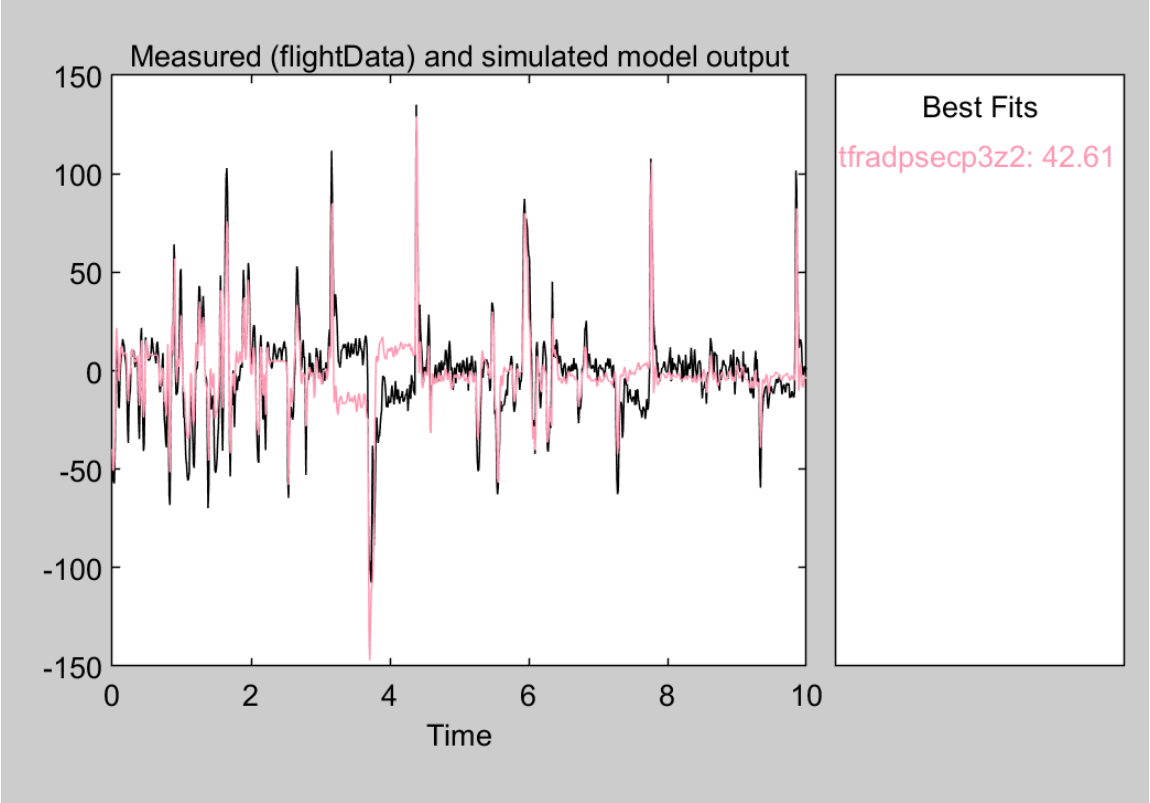
5. Using HW1_Timber.mat, you need to create the simplest, yet usable transfer function. Use a discrete transfer function model. The data file contains actual flight test data from a small fixed wing aircraft (100 Hz).

   Use the output roll rate(timber.rollrate) and the input aileron deflection (timber.aileron) in your estimation.

   Create a Bode plot of the data (see Canvas for example code) with coherence.

   Show the model fit, the transfer function, and the frequency response of the model (and Bode plot). Comment on your results, how do they compare in both fit and model complexity compared to the open loop model (Problem #1)? What is the cutoff frequency (where the attenuation starts)? Is this reasonable?

Model Fit:



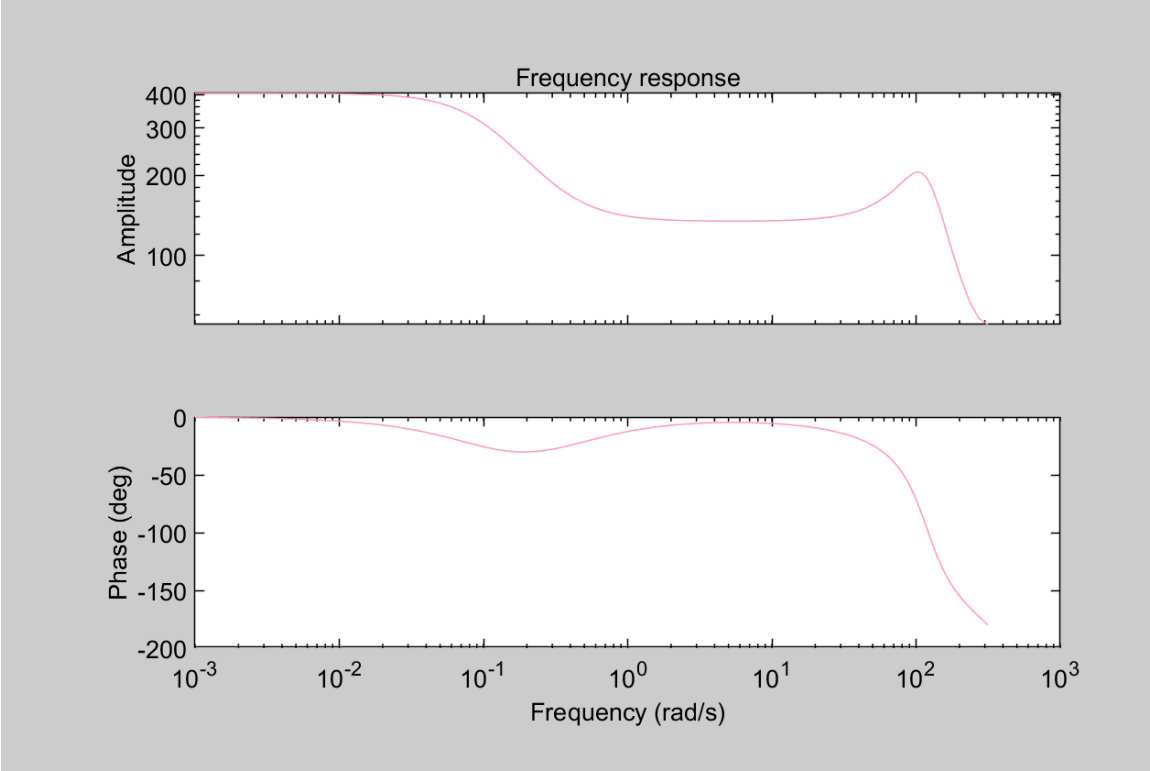
After applying a passband filter from 0 to 100 rad/s to create working data a transfer function with a 42.61% fit was achieved. This is the best fit I was able to create. I believe this transfer function is not good enough.

Transfer Function:

$$\frac{109.4\ z^{-1} - 109.1\ z^{-2}}{1 - 1.583\ z^{-1} + 0.9834\ z^{-2} - 0.3994\ z^{-3}}$$

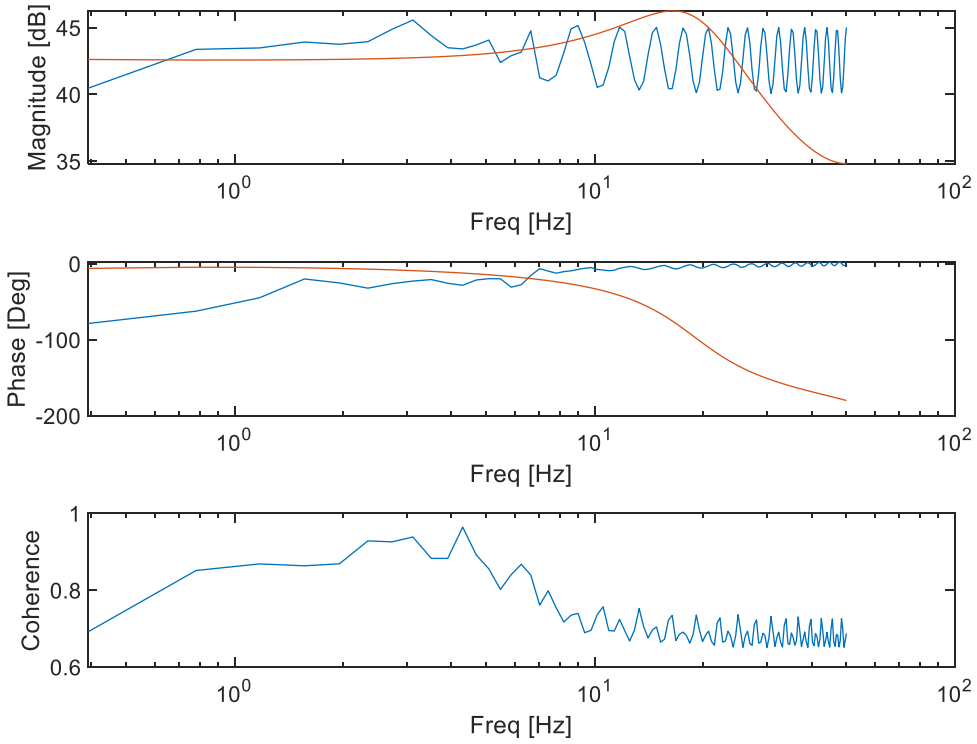
The above transfer function having 2 zeros and 3 poles is the best transfer function I was able to achieve. The model is more complex than the open loop model. I would have suspected an even more complex model.

Frequency Response:



Input to output is equal from .001 to .1 rad/s. After .1 rad/s the input to output ratio drops drastically. After 100 rad/s the signal starts to become out of phase.

Bode Plot With Coherence:



The model somewhat follows the frequency response. The coherence is good between about .5 hz and 5 hz.

CODE:

 # 1 & 2

```
load('msd_data_hw6.mat')
% import data %
time = msd.t;
y = msd.y;
yd = deriv(y,.01);
ydd = msd.ydd;
N = length(ydd);
bias = ones(N,1);
%%% OLS %%
x = [bias, y, yd];
T_hat = (x'*x)\x'*ydd;
Y_hat = x*T_hat;
R_sq = (T_hat'*x'*ydd - length(ydd)*mean(ydd)^2) / (ydd'*ydd -
length(ydd) * mean(ydd)^2)

%% Setting up the model matrices %%
```

```
% initial guesses %
for k=1:length(y)
    x4(k) = T_hat(1)*.1;
    x5(k) = T_hat(2)*.1;
    x6(k) = T_hat(3)*.1;
end
%%
Xk = [y(k); yd(k); ydd(k); x4(k); x5(k); x6(k)];

dt = 0.01;
k = 1;
m = 2.75;
N = 6;
F = [1 dt dt^2/2 0 0 0; 0 1 dt 0 0 0; x5(k)/m x6(k)/m 0 -x4(k) y(k)/m
yd(k)/m; 0 0 0 1 0 0; 0 0 0 0 1 0; 0 0 0 0 0 1];
G = [1 0 0 0 0 0;0 1 0 0 0 0;0 0 1 0 0 0;0 0 0 1 0 0; 0 0 0 0 1 0; 0 0
0 0 0 1];
P = eye(N)*1.0;
%QV = .001;
QV = 0.000001;
Q = [QV 0 0 0 0 0;
     0 QV 0 0 0 0;
     0 0 QV 0 0 0;
     0 0 0 QV 0 0;
     0 0 0 0 QV 0;
     0 0 0 0 0 QV];
R = [0.001 0 0 0 0 0;0 10 0 0 0 0;0 0 10 0 0 0, ;0 0 0 10 0 0;0 0 0 0
10 0;0 0 0 0 0 10];
P_diags(1,:) = diag(P);

%% Extended Kalman %%
for k = 2:length(time)
    %X_pred = F*Xk;
    X_pred(1,1) = Xk(1) + Xk(2)*dt + Xk(3)*(dt^2)/2;
    X_pred(2,1) = Xk(2) + Xk(3)*dt;
    X_pred(3,1) = -Xk(4)+Xk(5)/m*Xk(1) + Xk(6)/m*Xk(2);
    X_pred(4,1) = Xk(4);
    X_pred(5,1) = Xk(5);
    X_pred(6,1) = Xk(6);

    P_pred = F*P*F' + Q;
    Z = [y(k); yd(k); ydd(k); x4(k); x5(k); x6(k)];
    yk = Z - G*X_pred;
    Sk = G*P_pred*G' + R;
    Kk = P_pred*G'*Sk^-1;
    Xk = X_pred + Kk*yk;
    P = (eye(N) - Kk*G)*P_pred;
    P_diags(k,:) = diag(P);

    angle_kal(k) = Xk(1);
    rate_kal(k) = Xk(2);
    accel_kal(k) = -Xk(3);
    gravity_kal(k) = Xk(4);
    spring_over_mass_kal(k) = -Xk(5);
    damping_over_mass_kal(k) = Xk(6);
```

```
end

figure(1)
subplot(3,1,1)
plot(time, y, time, angle_kal)
title('(Position) Kalman Filter vs Measured')
xlabel('time [s]')
ylabel('position [m]')
subplot(3,1,2)
plot(time, yd, time, rate_kal)
title('(Velocity) Kalman Filter vs Measured')
xlabel('time [s]')
ylabel('velocity [m/s]')
subplot(3,1,3)
plot(time, ydd, time, accel_kal)
title('(Acceleration) Kalman Filter vs Measured')
xlabel('time [s]')
ylabel('acceleration [m/s^2]')
legend('data','kalman')

figure(2)
subplot(3,1,1)
plot(time, x4, time, gravity_kal)
title('(Gravity) Kalman Filter vs. Estimate')
xlabel('time [s]')
ylabel('gravity [m/s^2]')
subplot(3,1,2)
plot(time, x5, time, spring_over_mass_kal)
title('(Spring Coefficient) Kalman Filter vs. Estimate')
xlabel('time [s]')
ylabel('spring [1/s^2]')
subplot(3,1,3)
plot(time, x6, time, damping_over_mass_kal)
title('(Damping Coefficient) Kalman Filter vs. Estimate')
xlabel('time [s]')
ylabel('damping [1/s]')
legend('data','kalman')
```

#3

```
load('Qube_OpenLoop_HW8')
time = OL_Data(:,1);
t = time;
sampling_frequency = t(2);
fs = 1/sampling_frequency; %hz
input_motor_command = OL_Data(:,2);
u = input_motor_command;
angular_velocity = OL_Data(:,4);
vel = angular_velocity;

figure(1)
```

```
plot(time, vel)
title('Measured angular velocity')
xlabel('Time [sec]')
ylabel('ang. vel. [rad/s]')


input = u;
output = vel;
Tfinal = time(end); %seconds
% Doesn't work currently.
Y = fft(output); % FFT of OUTPUT data
X = fft(input); % FFT of INPUT data
Y = Y(1:round(end/2)); % Only 1st half is unique per Nyquist sampling
X = X(1:round(end/2));

f = fs*(1:length(time))/length(time); % Frequency vector, Hz
f = f(1:round(end/2)); % Grabs only the first half, as this is all that is
valid

Gyy = 1/(fs*length(input))*abs(Y).^2; % output PSD estimate
Gyy(2:end-1) = Gyy(2:end-1)*2;
Gxx = 1/(fs*length(input))*abs(X).^2; % input PSD estimate
Gxx(2:end-1) = Gxx(2:end-1)*2;
Gxy = 1/(fs*length(input))*conj(X).*Y; % Cross PSD
Gxy(2:end-1) = Gxy(2:end-1)*2;
%
% coherence = abs(Gxy).^2./((Gxx).*(Gyy)); % Coherence estimate
[matlab_coherence, ff] = mscohere(input,output,[],[],[],fs);
gxx = pwelch(input);
gyy = pwelch(output);
gxy = cpsd(input, output);
H2 = gxy./gxx;
H2db = 20*log10(H2);
phase2 = angle(H2)*180/3.14;
H = Gxy./Gxx; % Transfer Function estimate
HdB = 20*log10(H); % Converting to dB format
phase = angle(H)*180/3.14; % Phase lag in Degrees
figure(3)
ax1 = subplot(3,1,1);
semilogx(ff, real(H2db))
xlabel('Freq [Hz]')
ylabel('Magnitude [dB]')
ax2 = subplot(3,1,2);
semilogx(ff, phase2)
xlabel('Freq [Hz]')
ylabel('Phase [Deg]')
ax3 = subplot(3,1,3);
semilogx(ff,matlab_coherence)
xlabel('Freq [Hz]')
ylabel('Coherence')
linkaxes([ax1, ax2, ax3], 'x')
%% Combined Bode Plotter %%
ffRAD = ff*2*3.14; % Convert to rad/s
[mag, ph, w] = bode(tf1, ffRAD);
```

```
mag = squeeze(mag(1,1,:));
ph = squeeze(ph(1,1,:));
magDB = 20*log10(mag);
%wHZ = w*180/3.14;
figure(4)
ax1 = subplot(3,1,1);
semilogx(ff, real(H2db), ff, magDB)
xlabel('Freq [Hz]')
ylabel('Magnitude [dB]')
ax2 = subplot(3,1,2);
semilogx(ff, -phase2, ff, ph)%-360)
xlabel('Freq [Hz]')
ylabel('Phase [Deg]')
ax3 = subplot(3,1,3);
semilogx(ff,matlab_coherence)
xlabel('Freq [Hz]')
ylabel('Coherence')
linkaxes([ax1, ax2, ax3], 'x')
```

4

```
load('Qube_ClosedLoop_HW8')
time = CL_Data(:,1);
t = time;
sampling_frequency = t(2);
fs = 1/sampling_frequency; %hz
desired_angle = CL_Data(:,6);
u = desired_angle;
ang = CL_Data(:,3);


figure(1)
plot(time, ang)
title('Measured angle')
xlabel('Time [sec]')
ylabel('angle [rad]')
figure(2)
plot(time, u)
title('desired angle')
xlabel('Time [sec]')
ylabel('angle [rad]')

input = u;
output = ang;
Tfinal = time(end); %seconds
% Doesn't work currently.
Y = fft(output); % FFT of OUTPUT data
X = fft(input); % FFT of INPUT data
Y = Y(1:round(end/2)); % Only 1st half is unique per Nyquist sampling
X = X(1:round(end/2));

f = fs*(1:length(time))/length(time); % Frequency vector, Hz
f = f(1:round(end/2)); % Grabs only the first half, as this is all that is
valid
```

```
Gyy = 1/(fs*length(input))*abs(Y).^2; % output PSD estimate
Gyy(2:end-1) = Gyy(2:end-1)*2;
Gxx = 1/(fs*length(input))*abs(X).^2; % input PSD estimate
Gxx(2:end-1) = Gxx(2:end-1)*2;
Gxy = 1/(fs*length(input))*conj(X).*Y; % Cross PSD
Gxy(2:end-1) = Gxy(2:end-1)*2;
%
% coherence = abs(Gxy).^2./((Gxx).*(Gyy)); % Coherence estimate
[matlab_coherence, ff] = mscohere(input,output,[],[],[],fs);
gxx = pwelch(input);
gyy = pwelch(output);
gxy = cpsd(input, output);
H2 = gxy./gxx;
H2db = 20*log10(H2);
phase2 = angle(H2)*180/3.14;
H = Gxy./Gxx; % Transfer Function estimate
HdB = 20*log10(H); % Converting to dB format
phase = angle(H)*180/3.14; % Phase lag in Degrees
figure(3)
ax1 = subplot(3,1,1);
semilogx(ff, real(H2db))
xlabel('Freq [Hz]')
ylabel('Magnitude [dB]')
ax2 = subplot(3,1,2);
semilogx(ff, phase2)
xlabel('Freq [Hz]')
ylabel('Phase [Deg]')
ax3 = subplot(3,1,3);
semilogx(ff,matlab_coherence)
xlabel('Freq [Hz]')
ylabel('Coherence')
linkaxes([ax1, ax2, ax3], 'x')
%% Combined Bode Plotter %%
ffRAD = ff*2*3.14; % Convert to rad/s
[mag, ph, w] = bode(tf1p2z0, ffRAD);
mag = squeeze(mag(1,1,:));
ph = squeeze(ph(1,1,:));
magDB = 20*log10(mag);
%wHZ = w*180/3.14;
figure(4)
ax1 = subplot(3,1,1);
semilogx(ff, real(H2db), ff, magDB)
xlabel('Freq [Hz]')
ylabel('Magnitude [dB]')
ax2 = subplot(3,1,2);
semilogx(ff, -phase2, ff, ph)%-360)
xlabel('Freq [Hz]')
ylabel('Phase [Deg]')
ax3 = subplot(3,1,3);
semilogx(ff,matlab_coherence)
xlabel('Freq [Hz]')
ylabel('Coherence')
linkaxes([ax1, ax2, ax3], 'x')
```

5

```
load('HW1_Timber')
time = timber.t;
offset = time(1);
for k = 1:length(time)
    time(k) = time(k)- offset;
end
fs = 100; %hz
ang = timber.aileron;
u = lowpass(ang,.001);
rollrate = lowpass(timber.rollrate,.001);

figure(1)
plot(time, u)
title('Measured aileron angle')
xlabel('Time [sec]')
ylabel('angle [rad]')
figure(2)
plot(time, rollrate)
title('measured rollrate')
xlabel('Time [sec]')
ylabel('angle [rad/s]')

input = u;
output = rollrate;
Tfinal = time(end); %seconds
% Doesn't work currently.
Y = fft(output); % FFT of OUTPUT data
X = fft(input); % FFT of INPUT data
Y = Y(1:round(end/2)); % Only 1st half is unique per Nyquist sampling
X = X(1:round(end/2));

f = fs*(1:length(time))/length(time); % Frequency vector, Hz
f = f(1:round(end/2)); % Grabs only the first half, as this is all that is
valid

Gyy = 1/(fs*length(input))*abs(Y).^2; % output PSD estimate
Gyy(2:end-1) = Gyy(2:end-1)*2;
Gxx = 1/(fs*length(input))*abs(X).^2; % input PSD estimate
Gxx(2:end-1) = Gxx(2:end-1)*2;
Gxy = 1/(fs*length(input))*conj(X).*Y; % Cross PSD
Gxy(2:end-1) = Gxy(2:end-1)*2;
%
% coherence = abs(Gxy).^2./((Gxx).*(Gyy)); % Coherence estimate
[matlab_coherence, ff] = mscohere(input,output,[],[],[],fs);
gxx = pwelch(input);
gyy = pwelch(output);
gxy = cpsd(input, output);
H2 = gxy./gxx;
H2db = 20*log10(H2);
phase2 = angle(H2)*180/3.14;
H = Gxy./Gxx; % Transfer Function estimate
```

```
HdB = 20*log10(H); % Converting to dB format
phase = angle(H)*180/3.14; % Phase lag in Degrees
figure(3)
ax1 = subplot(3,1,1);
semilogx(ff, real(H2db))
xlabel('Freq [Hz]')
ylabel('Magnitude [dB]')
ax2 = subplot(3,1,2);
semilogx(ff, phase2)
xlabel('Freq [Hz]')
ylabel('Phase [Deg]')
ax3 = subplot(3,1,3);
semilogx(ff,matlab_coherence)
xlabel('Freq [Hz]')
ylabel('Coherence')
linkaxes([ax1, ax2, ax3], 'x')
%% Combined Bode Plotter %%
ffRAD = ff*2*3.14; % Convert to rad/s
[mag, ph, w] = bode(tfradpsecp3z2, ffRAD);
mag = squeeze(mag(1,1,:));
ph = squeeze(ph(1,1,:));
magDB = 20*log10(mag);
%wHZ = w*180/3.14;
figure(4)
ax1 = subplot(3,1,1);
semilogx(ff, real(H2db), ff, magDB)
xlabel('Freq [Hz]')
ylabel('Magnitude [dB]')
ax2 = subplot(3,1,2);
semilogx(ff, -phase2, ff, ph)%-360)
xlabel('Freq [Hz]')
ylabel('Phase [Deg]')
ax3 = subplot(3,1,3);
semilogx(ff,matlab_coherence)
xlabel('Freq [Hz]')
ylabel('Coherence')
linkaxes([ax1, ax2, ax3], 'x')
```