Project 2: System Identification of Fixed Wing Aircraft Pitch Rate

Bethany Calvert

May 8, 2023

1 Discrete Time Plant

1.1 Continuous-Time Plant Definition

The system used for analysis in this project is the pitch rate control loop for an A-4 fighter plane. A general block diagram for the system is displayed in Fig. 1.

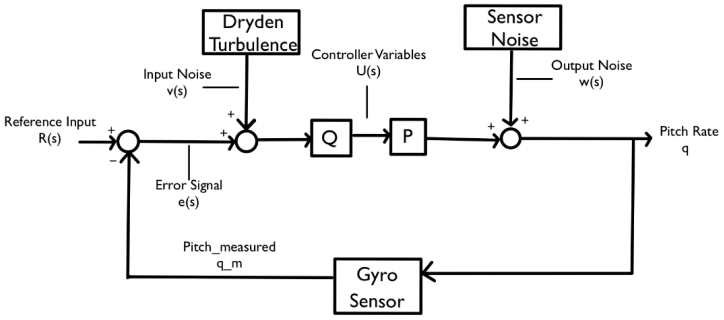


Figure 1: Block Diagram of System

The relevant physical characteristics are displayed in Table 1.

Characteristic	Variable	Value
Wingspan	b	27 ft
Altitude	h	1000 ft
Velocity	V	386 knots
Wind Speed at 20ft	W_{20}	15 knots

Table 1: Physical parameters of aircraft

The state space model for the system is as follows:

$$\begin{bmatrix}\dot{\theta}\\ \dot{q}\end{bmatrix} = \begin{bmatrix}-140.22 & -1.4560\\ -2.0100 & -0.3500\end{bmatrix}\begin{bmatrix}\theta\\ q\end{bmatrix} + \begin{bmatrix}-29.50\\ -5.00\end{bmatrix}\begin{bmatrix}\delta_e\end{bmatrix},\ y = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}\theta\\ q\end{bmatrix} \tag{1}$$

In this problem the states are the pitch rate, q, and pitch angle, θ. The control input is the elevator control, δ_e. The output is the updated pitch rate, q.

1.2 Discrete-Time Plant Definition

Using the $c2d()$ function in Matlab with a sampling time of .01s, the discrete time transfer function is as follows:

$$P(z) = \frac{-.0482(z - .2772)}{(z - .9967)(z - .246)} \tag{2}$$

This is a stable plant because all the poles and zeros are less than one and inside the unit circle. One pole is close to the unit circle boundary, indicated a slow pole in the system.

1.3 Auto Regressive Moving Average Model

The discrete-time plant can be put into a Auto Regressive Moving Average (ARMA) Model or $\phi^T\theta$ form as follows:

$$y(k+1) = \phi^T(k)\theta \tag{3}$$

Using the dynamics with the system for $k = 1, ..., N$, the ϕ and θ matrices are defined as:

$$\phi^T(k) = \begin{bmatrix} y(k-1) & y(k) & u(k-1) & u(k) \end{bmatrix} \tag{4}$$

$$\theta = \begin{bmatrix} -\alpha_0 \\ -\alpha_1 \\ \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} -.2452 \\ 1.2427 \\ .0134 \\ -.0482 \end{bmatrix} \tag{5}$$

The $y(0)$ and $y(1)$ entries are initialized using the simulated pseudo-random output from the first part of the project. Additionally, all the $u(k)$ entries are entered using the psuedo-random input.

2 Discrete Time Output Model Simulation

2.1 Psuedo-Random Input Definition

The parameters to construct the psuedo-random input are as follows:

Parameter	Value
Δt	.01 s
t_s	9.06 s
T	10.28 s
N	1028
$\Delta\omega$	.6112 rad/s
ω_n	314.1593 rad/s
ω_s	628.2185 rad/s
m	18

Table 2: FFT/EFTE Parameters

In the table, δt is the sampling time, t_s is the settling time, T is the time record length, N is the number of samples, $\delta\omega$ is the frequency sampling resolution, ω_n is the nyquist rate, ω_s is the sampling rate, and m is the number of time record lengths taken to ensure the transients from the dynamics settle. This is calculated using the twenty times the settling time, $20t_s$.

2.2 Noise Signals

There are two noise signals simulated in the system at the input and output. The representation for these noise signals is displace in Fig.1. The input noise signal, v is created by using a Dryden Noise Filter [1]. The transfer function for the disturbance for the pitch rate, q, is as follows:

$$H_w(s) = \sigma_w \sqrt{\frac{2L_v}{\pi V}} \frac{(1 + \frac{2\sqrt{3}L_w}{V}s)}{(1 + \frac{2L_w}{V}s)^2} \tag{6}$$

$$H_q(s) = \frac{\frac{s}{V}}{(1 + \frac{4b}{\pi V}s)} \cdot H_w(s) \tag{7}$$

The wingspan, altitude, aircraft velocity, and wind speed are stated in Table 1. Additionally, σ_w is the turbulence intensity and L_v and L_w are length scales, defined as follows:

$$2L_w = h,\ 2L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}} \tag{8}$$

$$\sigma_w = .1W_{20} \tag{9}$$

The output noise signal, w, is simulated by adding a small, random noise vector to the simulated output. Physically, this represents sensor noise in the system.

2.3 Discrete Output Simulation Using Psuedo-Random Input

Next, the ARMA model is simulated with a psuedo-random input and compared to the simulations model from the midterm project. To ensure that the transients settle, the simulation from the $m = 18$ time record length is used. The results in Fig. 2 display the results from the simulation.

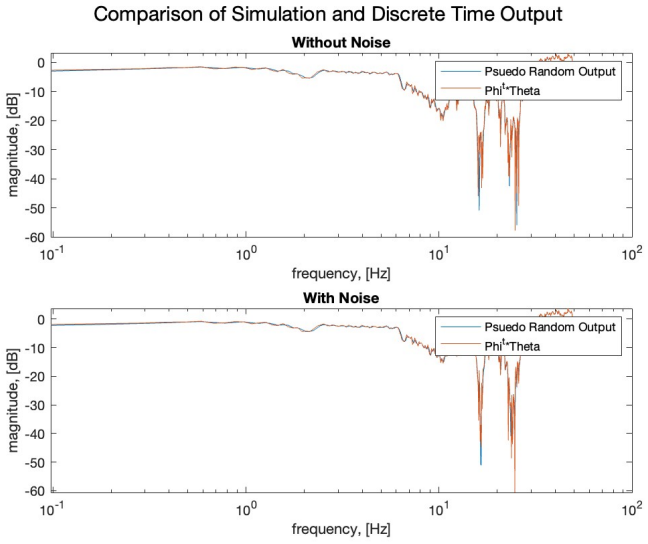


Figure 2: Comparison of Midterm and Discrete-Time Output Simulations

The results are plotted with and without noise to verify the models. In both cases, the ARMA model is validated, and it closely follows the model obtained in the midterm project

3 Predictor Construction

3.1 Predictor Construction

Now, a predictor, $\hat{y}_e(k+1)$ is defined using an estimated state, $\hat{\theta}$. Using the actual output data and output prediction, a prediction/equation error, $e(k+1)$ is defined. The predictor and equation error are:

$$\hat{y}_e(k+1) = \phi^T(k)\hat{\theta} \tag{10}$$

$$e = y(k+1) - \hat{y}_e(k+1) = \phi^T(k)(\theta - \hat{\theta}) = \phi^T(k)\tilde{\theta} \tag{11}$$

Additionally, the parameter $\tilde{\theta}$ represents the difference between the estimate and the actual system parameters.

3.2 Predictor Validation

Initially, some random estimated parameter set, $\hat{\theta}$ is used to validate this predictor. Additionally, to prove the predictor is working correctly, the actual plant parameters, θ, are used. The results of this test are display in Fig. 3.

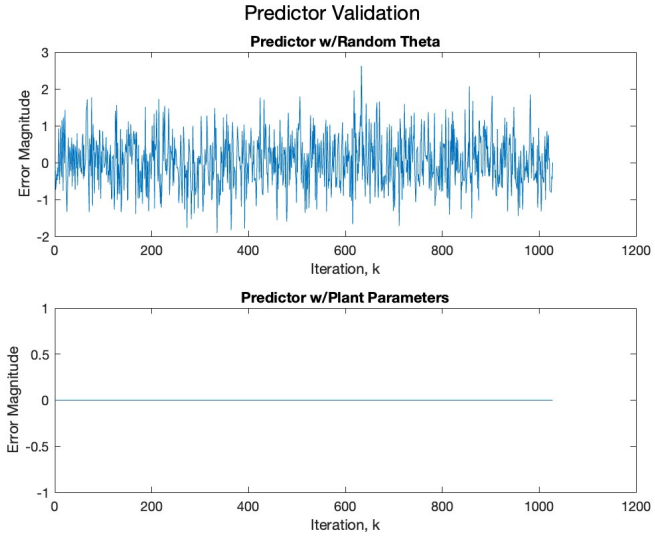


Figure 3: Validation of Predictor using Plant Parameters

The predictor does work correctly, as it has a nonzero error when using the random estimate, and it has zero error when using the actual plant parameters.

4 Gradient Descent System ID

4.1 Gradient Descent Algorithm

An alternative system identification method is the gradient descent method. This method is constructed to create estimates that bring the equation error to zero. The gradient descent algorithm is:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + 2\mu\phi^T(k)e(k+1) \tag{12}$$

In this equation, μ is the gradient step size update. Here, μ must be small enough to guarantee that the Lyapunov bound for monotone parameter error norm reduction is satisfied. To guarantee that μ is small enough, $\phi^T(k)$ must "move around" enough so that there is no orthogonal subspace of $\phi^T(k)\theta$ except at the origin. Therefore, the step size must adhere to the following criteria:

$$\mu \leq \frac{1}{\phi^T\phi} \tag{13}$$

4.2 Gradient Descent w/o Noise

The error norm without noise versus the iteration index is plotted in Fig. 4.

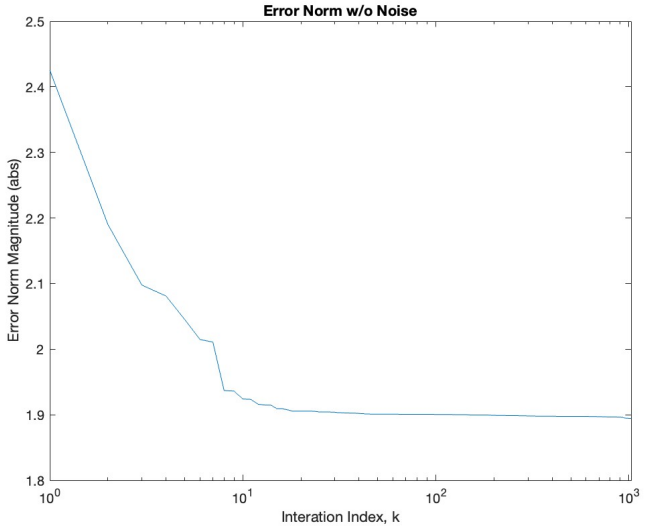


Figure 4: Error Norm of Gradient Descent w/o Noise

In Fig. 4, the error norm, $\tilde{\theta}^T\tilde{\theta}$, in the gradient descent algorithm is reduced quickly at first, but then the algorithm slows down. It converges to a value of around 1.9. This is only 25% of its initial value as compared to the .1% asked for in the project parameters. Further investigation is required because the value does converge, but it does not converge to zero. The error of the system, and the parameter iteration history are plotted in Fig. 5-6.

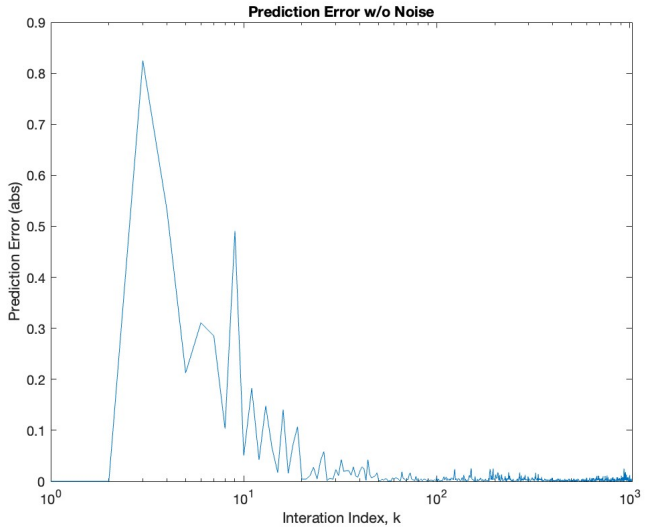


Figure 5: Error of Gradient Descent w/o Noise

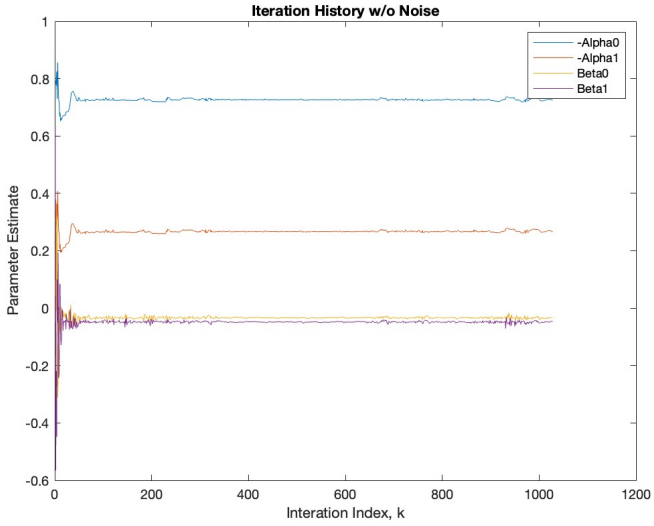


Figure 6: History of Gradient Descent Parameters w/o Noise

In Fig. 5, it can be seen that the error is settling close to zero. This contradiction between the error norm and error typically means that $\phi(k)$ is not "persistently excited". However, when assessing the rank of $\Phi\Phi^T$, it is indeed full rank. This complication could be coming due to the lack of excitation in certain directions of the parameter space. However, later in Section 4.4, the ID is successful. Since the error was zero, the error norm converged, and the overall system ID was successful, I decided to move forward with the estimate from this gradient descent algorithm.

4.3 Gradient Descent w/Noise

In this section of analysis, the simulation of the discrete system with noise is used. The error norm of the gradient descent algorithm with noise is plotted in Fig. 7.

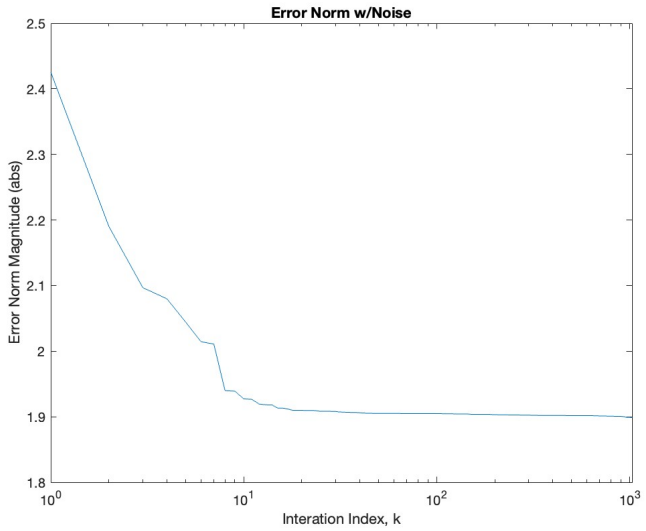


Figure 7: Error Norm of Gradient Descent w/ Noise

Here, we have the same issue as in the case with no noise, that the error norm converges, but it does not go to zero. The error and iteration history with noise are plotted in Fig. 8-9.

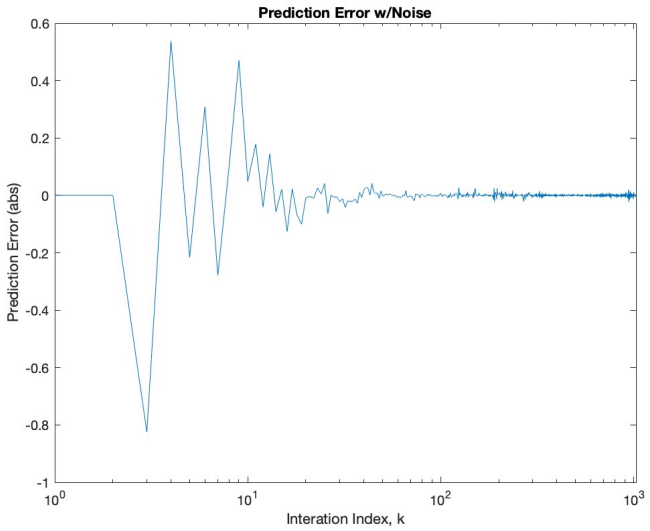


Figure 8: Error of Gradient Descent w/ Noise

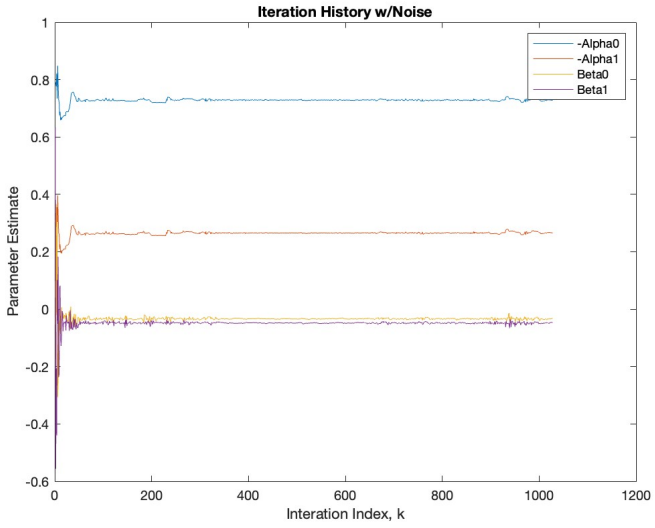


Figure 9: History of Gradient Descent Parameters w/ Noise

Similarly, this system only reduces the error norm value by 25%, but it does converge. However, the error of the system does still go to zero. The $\Phi\Phi^T$ matrix is still full rank as well. However, similar to the case with no noise, the system ID outlined in Section 4.4 still provides satisfactory system ID.

4.4 Comparison of ID Models w/Plant

After validating that the error becomes very small with the gradient descent ID, the transfer functions are developed from the final state estimates with and without noise. The "final" parameter estimates for the gradient descent algorithms without noise, $\hat{\theta}_0$, and with noise, $\hat{\theta}$ are

$$\hat{\theta}_0 = \begin{bmatrix} 2.7873 \\ -1.7991 \\ -.1291 \\ -.0408 \end{bmatrix}, \hat{\theta} = \begin{bmatrix} .7061 \\ .2882 \\ -.0318 \\ -.0465 \end{bmatrix} \tag{14}$$

The gradient descent transfer function without noise is

$$P_{g0}(z) = \frac{-.040832(z + 3.161)}{(z + 2.796)(z - .9969)} \tag{15}$$

The gradient descent transfer function with noise is

$$P_g(z) = \frac{-.046461(z + .6852)}{(z + .7085)(z - .9967)} \tag{16}$$

Figure 10 displays the bode plots of the gradient descent ID TF's with the nominal plant.

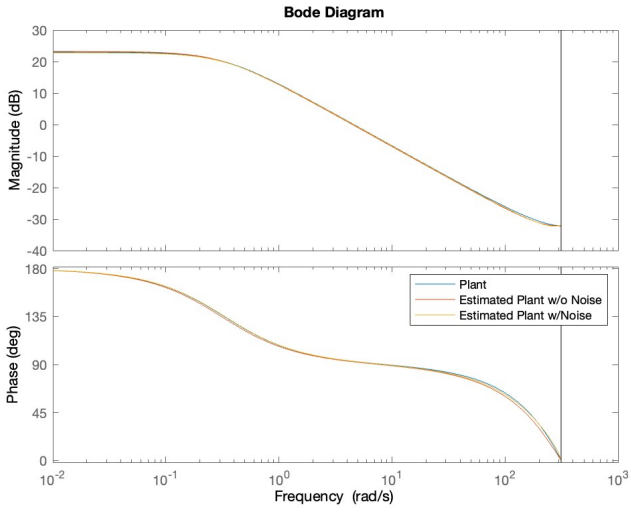


Figure 10: Comparison of Bode Plots of Gradient Descent TF's with Nominal Plant

Overall, the gradient descent transfer function without noise seems to have an unstable transfer function because there is a pole that is outside of the unit circle. However, the transfer function for the gradient descent ID with noise is stable. Overall, Fig. 10 displays that the transfer functions all have very similar behavior. The system proves to be closely identifiable with and without noise using the gradient descent algorithm.

5 Least Squares Solution

5.1 Batch Least Squares

Next, using the same source, measurement noise, and closed loop plant simulation from Section 4, the batch least squares (BLS) solution is computed. The algorithm is compiled by squaring the prediction error and finding the minimum solution. The BLS estimate is as follows:

$$\hat{\theta} = (\Phi\Phi^T)^{-1}\Phi Y \tag{17}$$

Here, Y is the matrix of the simulated output with noise, and Φ is the same information vector from Sec. 1.3, but now it contains the u and y signals with noise included. The optimal estimate from the BLS solution is

$$\hat{\theta} = \begin{bmatrix} -.2452 \\ 1.22427 \\ .0134 \\ -.0482 \end{bmatrix} \tag{18}$$

5.2 Parameter Set Estimation

When measurement noise is introduced to the system, the estimate can never converge to the true estimate because the noise cannot be predicted. However, the estimate can get within a region close to the estimate. This region is characterized by parameter set estimation where a family of

plant parameters is included to account for the noise in the system. If there is some estimate

$$\hat{\theta} = \theta^* + \Delta\theta \tag{19}$$

where θ^* is the BLS solution, and $\Delta\theta$ defines the region that accounts for the noise in the family of plant parameters near the optimum estimate. The region $\Delta\theta$ is calculated by

$$||\Delta\theta||_i \leq \left(\frac{\delta^2(E^TE)^*}{\lambda_i(\bar{\Phi}\bar{\Phi}^T)}\right)^{\frac{1}{2}} \tag{20}$$

Here, δ is the uncertainty parameter, $\bar{\Phi}$ is the information vector with noise, λ_i is the eigenvalue in the i'th eigen direction. The variable $(E^TE)^*$ is the prediction error squared, or further,

$$(E^TE)^* = (\bar{Y} - \bar{\Phi}^T\theta^*)^T(\bar{Y} - \bar{\Phi}^T\theta^*) \tag{21}$$

First, since the exact size of E^TE is known from the simulation, the algorithm is implemented without overestimation. The estimate region size for each eigen direction is defined in the following parameter set:

$$\Delta\theta = 10^{-8} * \begin{bmatrix} .1602 \\ .002 \\ .002 \\ .002 \end{bmatrix} \tag{22}$$

This matrix represents the maximum distance in each eigen direction that the parameter set can extend, but still accurately describe the plant. Two locations are tested to compute a family of feasible discrete time models. One at the boundary, and another half way to the boundary at $\frac{1}{2}\Delta\theta$. The estimates are as follows:

$$\hat{\theta}_{boundary} = \begin{bmatrix} -.2452 \\ 1.2427 \\ .0134 \\ -.0482 \end{bmatrix}, \; \hat{\theta}_{.5} = \begin{bmatrix} -.2452 \\ 1.2427 \\ .0134 \\ -.0482 \end{bmatrix}, \tag{23}$$

They look identical because the $\Delta\theta$ value is very small. The resulting transfer functions are the because of how small the $\Delta\theta$ value, and they are represented by

$$P_{BLS0}(z) = \frac{-.0482(z - .2772)}{(z - .9967)(z - .246)} \tag{24}$$

The bode plot of this identification compared to the analytic plant is plotted in Fig. 11.

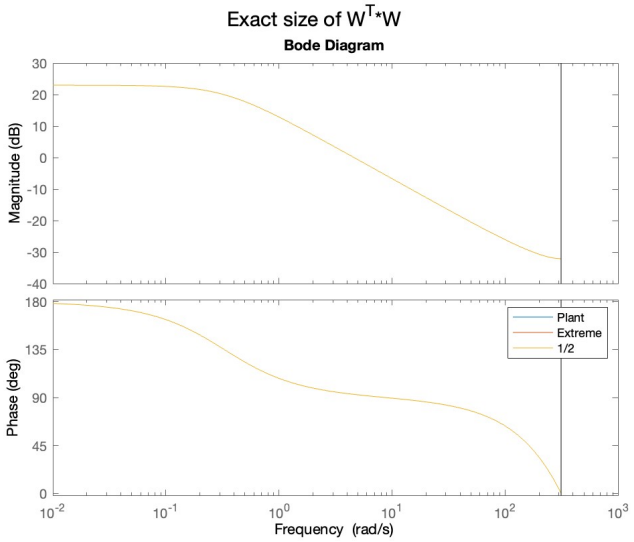


Figure 11: Comparison of Bode Plots of Batch Least Squares Parameter Set Estimation TF's with Nominal Plant

Here, it is displayed that the fit of the BLS ID method is extremely successful, and it looks like it follows the analytic plant exactly.

Next, $E^T E$ is overestimated by 10%. Now setting the δ^2 value to be .1, the estimate region size for each eigen direction is

$$\Delta\theta = 10^{-9} * \begin{bmatrix} .5067 \\ .006 \\ .006 \\ .005 \end{bmatrix} \tag{25}$$

Taking the same locations along the major axes as in the exact case the estimates at the region boundary and half way to the region boundary are

$$\hat{\theta}_{boundary} = \begin{bmatrix} -.2452 \\ 1.2427 \\ .0134 \\ -.0482 \end{bmatrix}, \hat{\theta}_{.5} = \begin{bmatrix} -.2452 \\ 1.2427 \\ .0134 \\ -.0482 \end{bmatrix}, \tag{26}$$

Again, since the $\Delta\theta$ value is very low, the estimates look identical to the exact case above. Additionally, the transfer function will be the same as in Eq. 24. The bode plot of this ID transfer function compared to the nominal plant is displayed in Fig. 12.

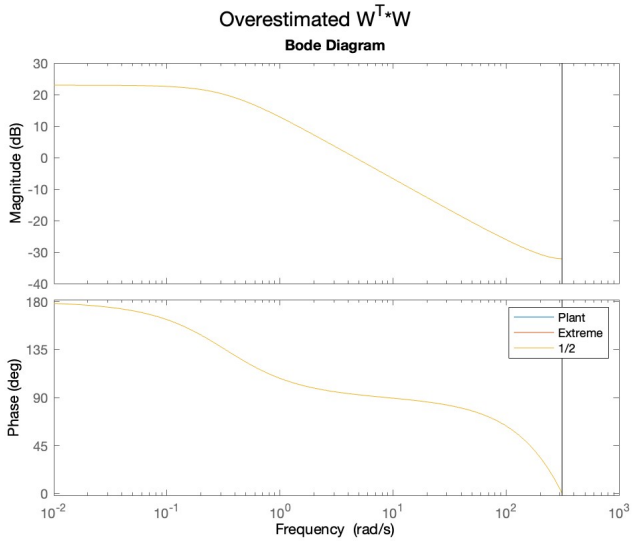


Figure 12: Comparison of Bode Plots of Overestimated Batch Least Squares Parameter Set Estimation TF's with Nominal Plant

Overall, the noise in this system is very small, which led to a very small region in the parameter set estimation. However, the BLS solution and the parameter set estimation approach efficiently identified the dynamics of the nominal plant.

6 Modeling Accuracy of ID Methods Compared to EFTE

The results from the EFTE approach are displayed in Fig. 13.

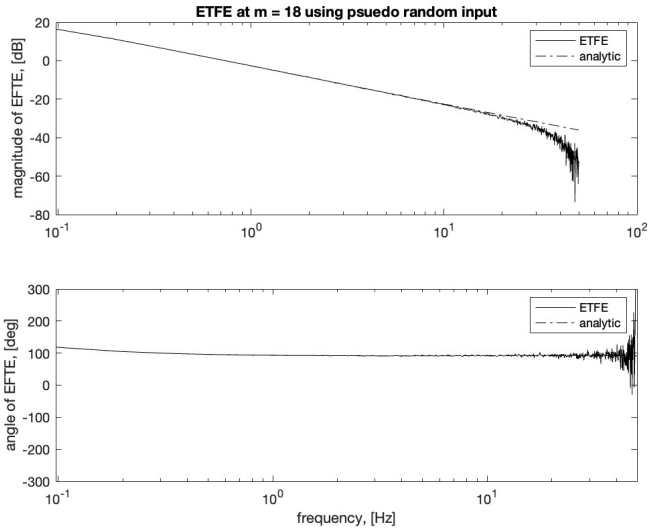


Figure 13: Comparison of Bode Plots of EFTE with Analytic Plant

In the EFTE approach, it can be seen that while it follows the plant very well in the low frequency

region, this fall apart in the high frequency domain. The amplification of noise at high frequencies causes a lack of efficiency in identification. Comparatively, both of the system identification methods outlined in Sections 4-5, much more accurately represent the plant despite the presence of noise.

7 Conclusions

There are trade offs between completing system identification in the frequency domain (EFTE methods) versus the time domain (gradient descent/least squares). In time domain ID, the estimate of the plant, $\hat{P}$ is iteratively determined by bringing the system error to closer to zero over each iteration. It involves relatively simple algorithms with constant updates of the parameters in the system. Alternatively, in frequency domain ID, $\hat{P}$ is constructed by matching the transfer function, instead of actually being built as a prediction. In this approach, a whole batch of data is necessary before DFT analysis can be applied to it, and these calculations can be expensive due to the necessity of the DFT calculations for prediction. However, this algorithm can be useful to find identification information at frequencies of interest. Additionally, the amplitude of the pseudo-random input can be easily tuned according to these frequencies of interest.

Overall, as it applies to my system, the batch identification method was the most successful in identifying the plant of my system. However, in further studies, it would be useful to implement larger noise signals to see how efficient this approach still is. Although, since all of the other methods implemented in this project use the same noise signals, it proves to be extremely efficient in comparison. It creates the same exact transfer function as the nominal plant, even with noise sources in the system.

The gradient descent produces the next most accurate identification technique. Here, a challenge was excitation issues within the pseudo-random signal. While it produced an efficient identification with zero error, the error norm of the system was not driven to zero. Further adjustment of the excitation amplitude to more efficiently excite the system could help drive this error norm towards zero.

Lastly, the EFTE approach was the least accurate, but still exhibited success in predicting the system. In the high frequency, this method fell off from tracking the plant behavior. However, it still produced a stable transfer function that is effective in the low frequency range. Overall, the time-domain methods proved to be more efficient, and overall easier to implement.

8 Appendix

Table of Contents


```
clc, clear all, close all, format compact
beep off
```

Center frequency and sampling intervals

```
wc = 8.54 ;
delt = .01 ;                    % Time sampling period [sec]
T = 10.28 ;                     % Time record length [sec]
N = T/delt ;                    % Number of samples in record
ts = 9.06 ;                     % Settling Time
sim_len_min = 20*ts ;           % 181.2s
m = 18 ;                        % total number of N-periods/Tperiods simulated
t = [0:m*N-1]*delt ;            % Simulation length
delw = 2*pi/T ;                 % Frequency sampling resolution [rad/sec]
wn = pi/delt ;                  % Nyquist rate [rad/sec]
ws = 2*pi/delt ;                % Sampling rate [rad/sec]  %.1 to 50
w = [0:N/2-1]*delw ;            % FFT frequency vector up to wn
w_m = [0:N*m-1]*delw ;
w_f = [0:N-1]*delw ;
f = w/(2*pi) ;                  % FFT freq in Hz
rand('state',0)
```

Plant

```
Za = -140.22 ;
Zq = -1.456 ;
Ma = -2.01 ;
Mq = -.35 ;
Zde = -25.9 ;
Mde = -5 ;

A = [Za Zq; Ma Mq] ;
Aa = [Za Zq; Ma Mq] ;
B = [Zde; Mde] ;
```

```
C = [0 1] ;
D = 0 ;
G = ss(A,B,C,D) ;

[num,den] = ss2tf(A,B,C,D) ;
OL_tf = tf(num,den) ;
[mag,phase] = bode(OL_tf,w) ;
Mag = reshape(mag,1,length(w)) ;
Phase = reshape(phase,1,length(w)) ;
plant = zeros(1,length(w_m)) ;
for i = 1:length(w_m)
    plant(:,i) = evalfr(OL_tf,j*w_m(i)) ;
end

% control
w0 = 8 ;
a = 1/100 ;
M = 1.5 ;
s = tf('s') ;
W1 = (s/M+w0)/(s+w0*a) ;
W2 = (s+.1)/(s+100) ;
W3 = [];

[K,CL,Gam,INFO] = mixsyn(G,W1,W2,W3) ;
[numK,denK] = ss2tf(K.A,K.B,K.C,K.D) ;
K_tf = tf(numK,denK) ;

[Magk,Phasek] = bode(K,w) ;
magk = reshape(Magk,1,length(w)) ;
phasek = reshape(Phasek,1,length(w)) ;
```

Noise plants

```
%Dryeden Turbulence
b = 27*3 ; % wingspan in m
h = 3000 ; % altitude in m
V = 385/1.944 ; % speed knots to m/s
Lw = h/2 ;
Lv = h/2/(.177+.000823*h)^(1.2) ;
W20 = 15/1.944 ;      % wind speed of 15 knots in m at height of 60m
sig_w = .1*W20 ;
s = tf('s') ;


Hw = sig_w*sqrt(2*Lv/(pi*V))*(1+2*sqrt(3)*Lw/V*s)/(1+2*Lw/V*s)^2 ;
Hq = s/V/(1+4*b/(pi*V)*s)*Hw ;
v_in = impulse(Hq,t)+.01*rand(size(t))' ;      % Simulate Dryden function with
 psuedo random input & noise
```

Psuedo Random Input

```
%psuedo random input
```

```
upRMS = 1 ;
up0 = zeros(size(t))+sqrt(2/N)*upRMS ;

for k = 1:N/2-1
    A = 2/sqrt(N) ;             % Vary amplitude
    phi = (2*pi)*rand ;      % Random phase uniformly dist. in 0:2pi
    up0 = up0+A*cos(2*pi*k/T*t+phi) ;
end

up0_1 = up0(:,1:N) ;
up0_3 = up0(:,2*N+1:3*N) ;
up0_m = up0((m-1)*N+1:m*N) ;

% Psuedo random input
up = up0+v_in' ;                % Add input noise from Dryden

up_1 = up(:,1:N) ;
up_3 = up(:,2*N+1:3*N) ;
up_m = up((m-1)*N+1:m*N) ;
```

Psuedo Random Ouput

Simulate plant output

```
wd = .01*rand(size(t)) ;
yp = lsim(num,den,up,t)' + wd ;      % Output with noise
yp_1 = yp(:,1:N) ;
yp_3 = yp(:,2*N+1:3*N) ;
yp_m = yp((m-1)*N+1:m*N) ;

yp0 = lsim(num,den,up0,t)' ;          % Output w/o noise
yp0_1 = yp0(:,1:N) ;
yp0_3 = yp0(:,2*N+1:3*N) ;
yp0_m = yp0((m-1)*N+1:m*N) ;
```

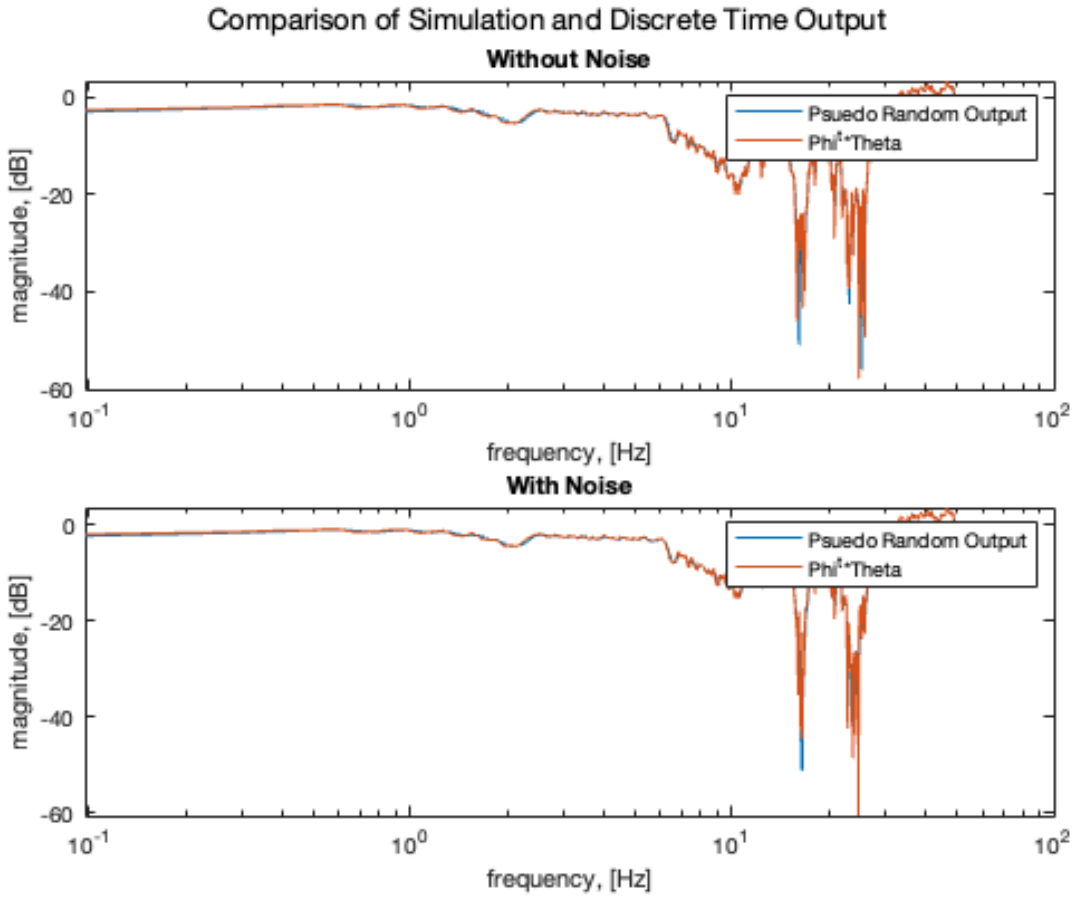
Discrete Time Model

```
tf_ol = c2d(OL_tf,delt,'zoh') ;

theta = [-tf_ol.Denominator{1,1}(3); -tf_ol.Denominator{1,1}(2);
 tf_ol.Numerator{1,1}(3); tf_ol.Numerator{1,1}(2)] ;
alpha = [-tf_ol.Denominator{1,1}(3); -tf_ol.Denominator{1,1}(2)] ;
beta = [tf_ol.Numerator{1,1}(3); tf_ol.Numerator{1,1}(2)] ;

% No noise
y0n = zeros(1,m*N) ;
mu0 = zeros(m*N,1) ;
phi0_t = zeros(m*N,4) ;
for k = 1:m*N-2
    y0n(:,1) = yp0(:,1) ;
    y0n(:,2) = yp0(:,2) ;
    phi0_t(k+1,:) = [y0n(:,k) y0n(:,k+1) up0(:,k) up0(:,k+1)] ;
```

```
    y0n(:,k+2) = phi0_t(k+1,:)*theta ;
end
y0 = phi0_t*theta ;            % Shift y to y(k+1)


% Phi'Theta output sectioning
y0_1 = y0(1:N,:) ;
y0_3 = y0(2*N+1:3*N,:) ;
y0_m = y0((m-1)*N+1:m*N,:) ;

% With noise
yn = zeros(1,N) ;
phi_t = zeros(m*N,4) ;
for k = 1:m*N-2
    yn(:,1) = yp(:,1) ;
    yn(:,2) = yp(:,2) ;
    phi_t(k+1,:) = [yn(:,k) yn(:,k+1) up(:,k) up(:,k+1)] ;
    yn(:,k+2) = phi_t(k+1,:)*theta ;
end
y = phi_t*theta ;            % Shift y to y(k+1)

% Phi'Theta output sectioning
y_1 = y(1:N,:) ;
y_3 = y(2*N+1:3*N,:) ;
y_m = y((m-1)*N+1:m*N,:) ;
```

Comparison of models

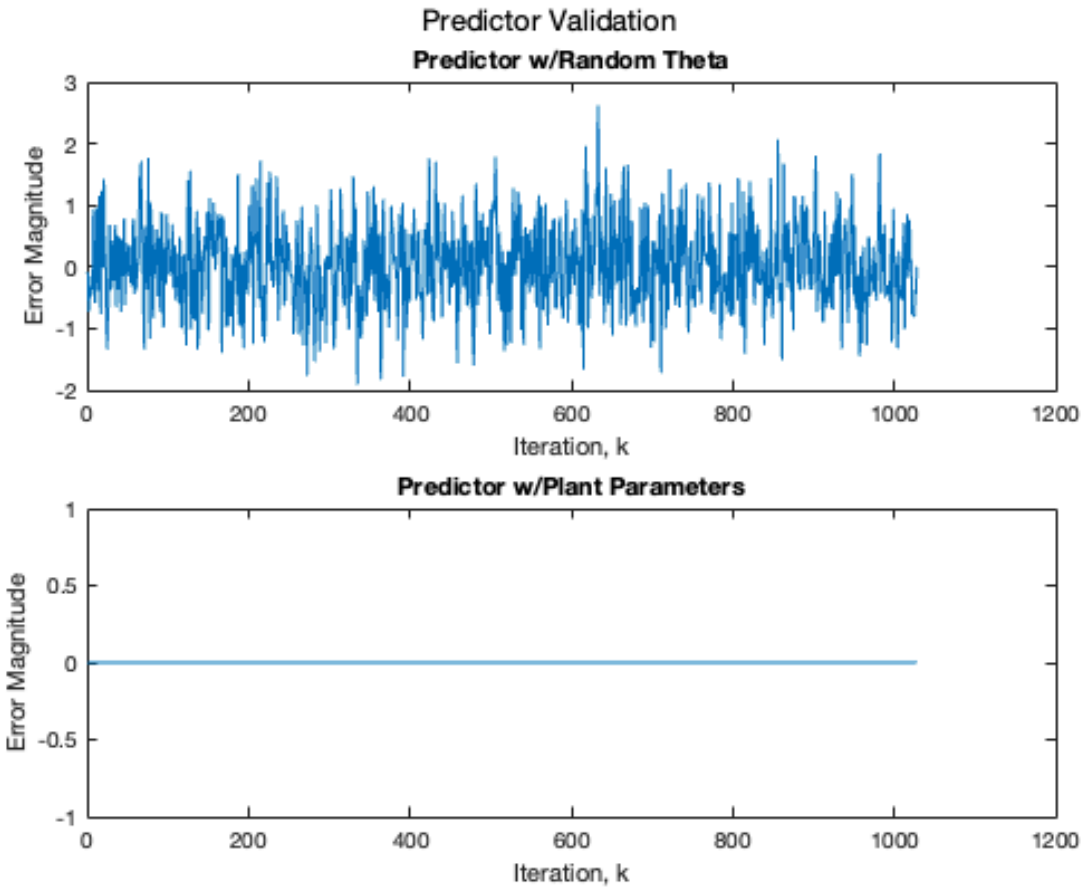
```
figure(1)
sgtitle('Comparison of Simulation and Discrete Time Output')
subplot(2,1,1)
semilogx(f,20*log10(abs(yp0_m(1:N/2))))
hold on
semilogx(f,20*log10(abs(y0_m(1:N/2))))
legend('Psuedo Random Output','Phi^t*Theta')
title('Without Noise')
xlabel('frequency, [Hz]')
ylabel('magnitude, [dB]')

subplot(2,1,2)
semilogx(f,20*log10(abs(yp_m(1:N/2))))
hold on
semilogx(f,20*log10(abs(y_m(1:N/2))))
legend('Psuedo Random Output','Phi^t*Theta')
title('With Noise')
xlabel('frequency, [Hz]')
ylabel('magnitude, [dB]')
```



Predictor

```
theta_est = 1.5*rand(4,1) ;                % assign random parameter vector
yeh = phi0_t*theta_est ;                   % prediction
e_eh = phi0_t*(theta-theta_est) ;          % Equation Error


theta_h = theta ;
y_eh = phi0_t*theta_h ;
theta_t = theta-theta_h ;
e_e = phi0_t*theta_t ;           % PROVE w/Plant Param

figure(2)
sgtitle('Predictor Validation')
subplot(2,1,1)
plot(1:N,e_eh((m-1)*N+1:m*N,:))
title('Predictor w/Random Theta')
ylabel('Error Magnitude')
xlabel('Iteration, k')
subplot(2,1,2)
plot(1:N,e_e((m-1)*N+1:m*N,:))
title('Predictor w/Plant Parameters')
```

```
ylabel('Error Magnitude')
xlabel('Iteration, k')
```

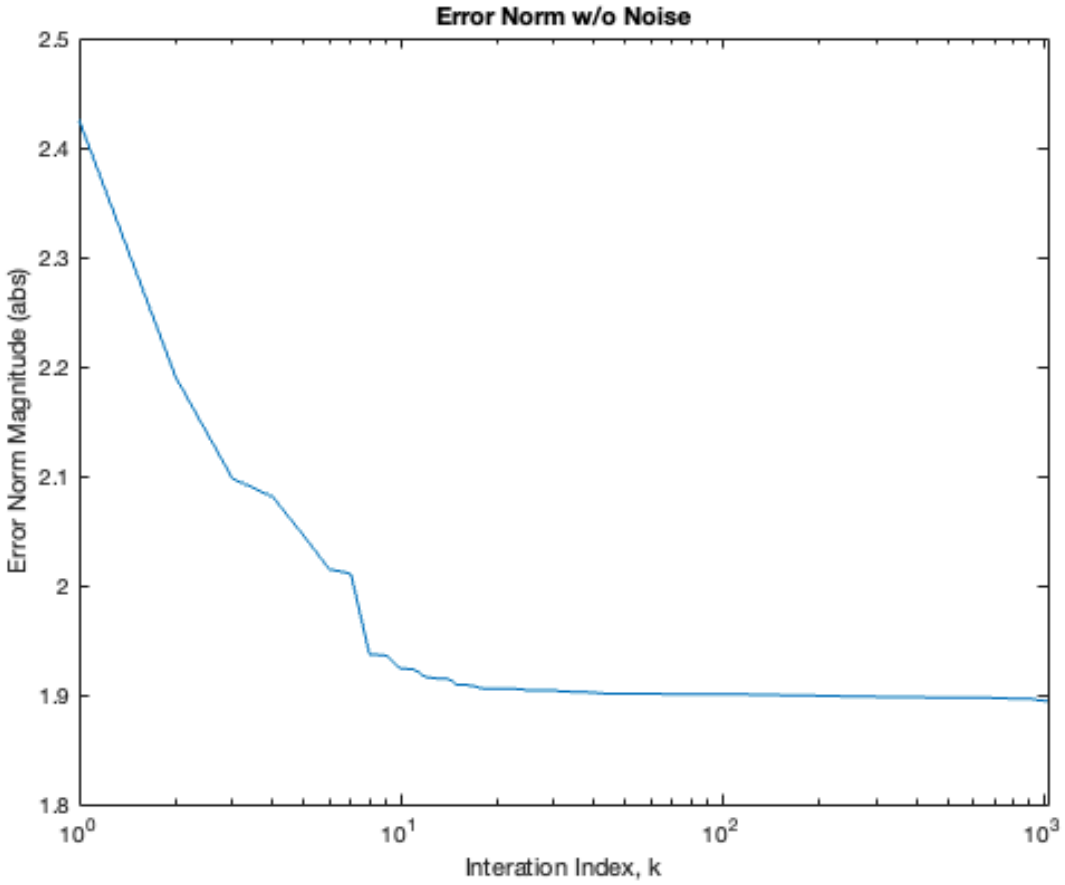


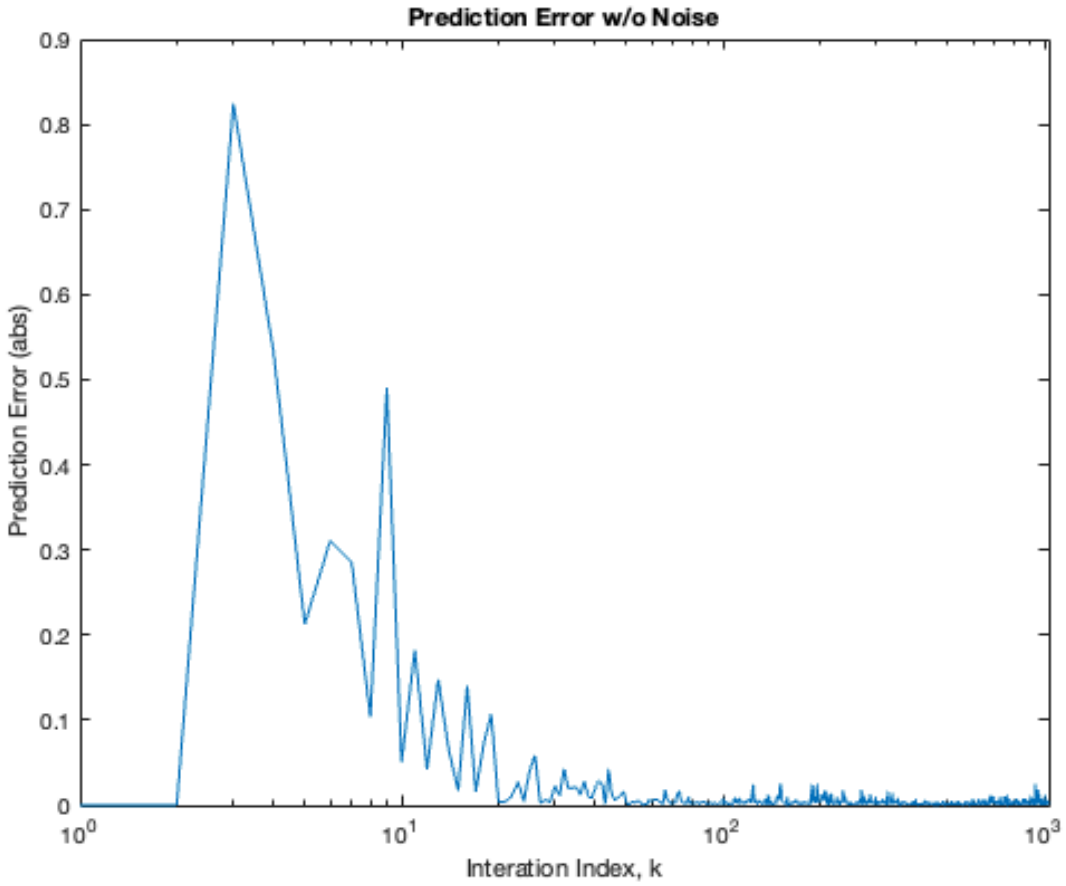
Gradient Descent

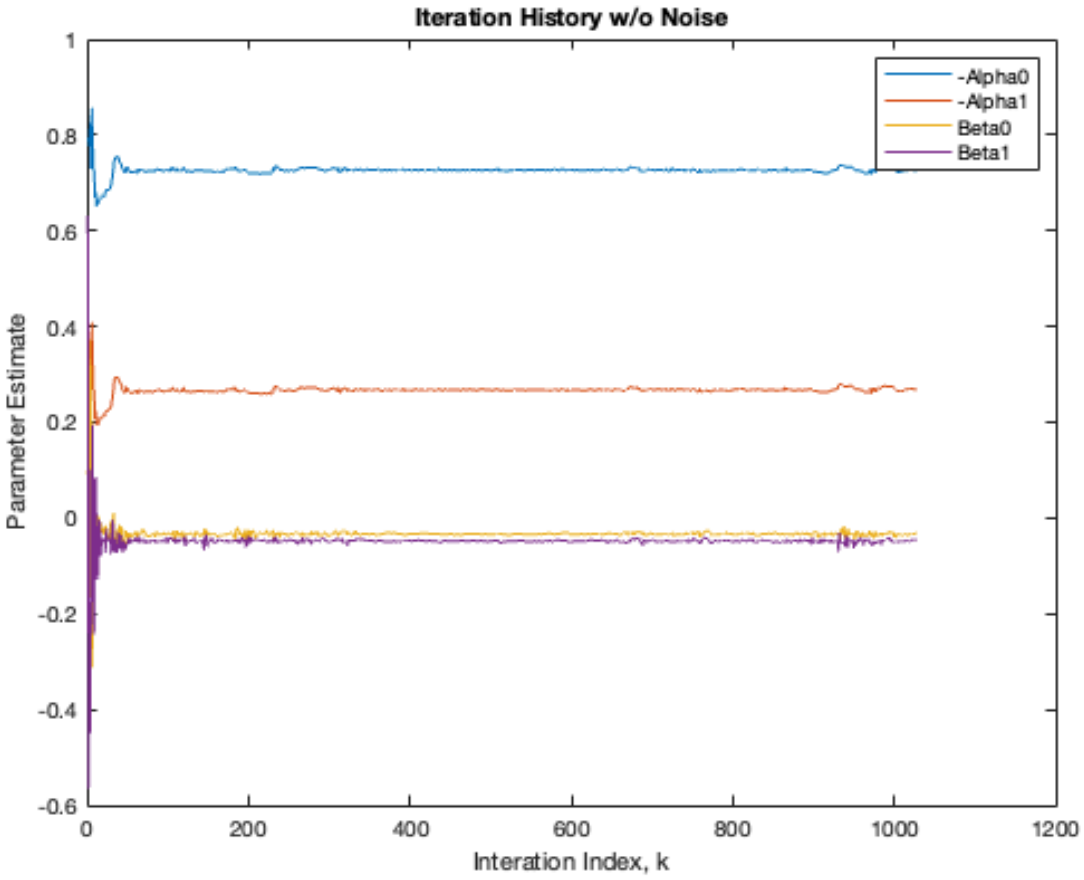
```
% Without Noise

for k = 1:m*N-2
    theta_hat0(:,1) = theta_est ;
    mu0(k,:) = abs(1/(phi0_t(k+1,:)*phi0_t(k+1,:)')-.1) ;
    e_g0(k+2,:) = phi0_t(k+1,:)*(theta-theta_hat0(:,k)) ;
    theta_tilde0(:,k) = (theta-theta_hat0(:,k)) ;
    norm_eg0(k,:) = theta_tilde0(:,k)'*theta_tilde0(:,k) ;
    theta_hat0(:,k+1) = theta_hat0(:,k)+2*mu0(k,:)*phi0_t(k+1,:)'*e_g0(k
+2,:) ;
end

figure(3)
semilogx(1:N-2,abs(norm_eg0(1:N-2)'))
title('Error Norm w/o Noise')
xlabel('Interation Index, k')
ylabel('Error Norm Magnitude (abs)')
```

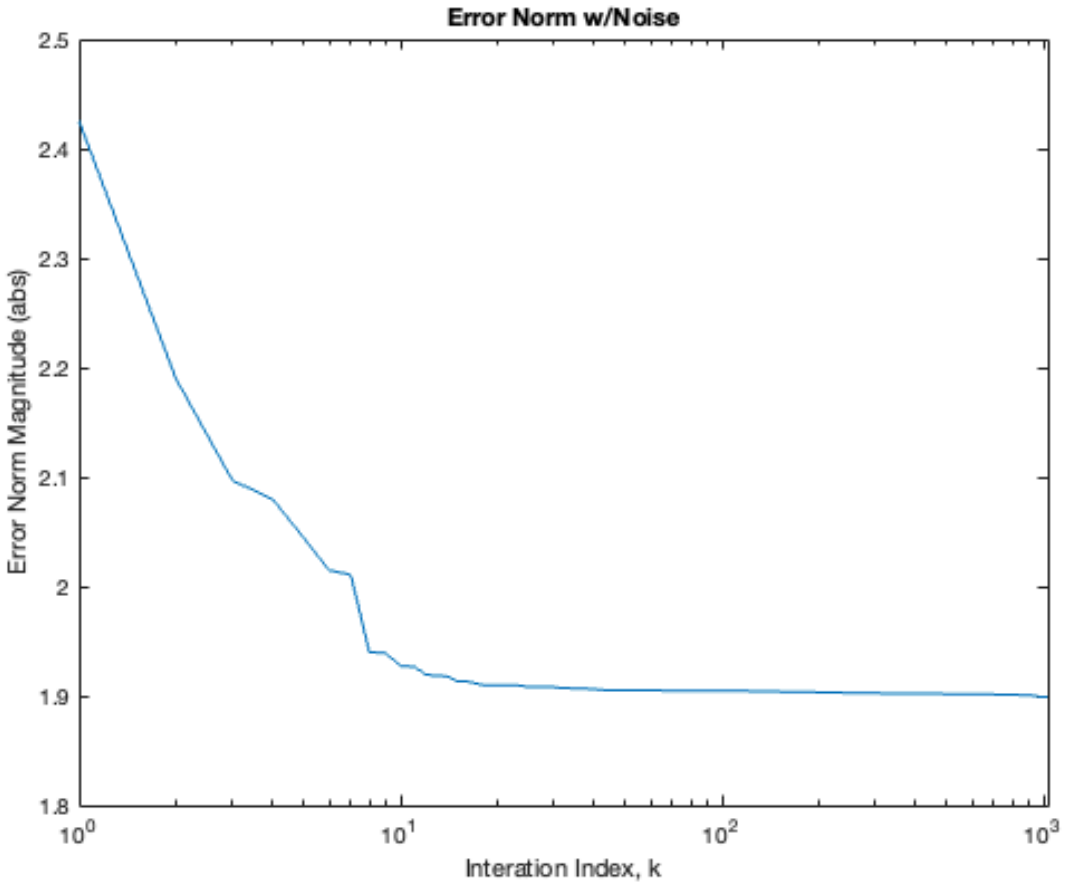
```
figure(4)
semilogx(1:N,abs(e_g0(1:N,:)'))
title('Prediction Error w/o Noise')
xlabel('Interation Index, k')
ylabel('Prediction Error (abs)')

figure(5)
plot(1:N,theta_hat0(:,1:N))
title('Iteration History w/o Noise')
xlabel('Interation Index, k')
ylabel('Parameter Estimate')
legend('-Alpha0','-Alpha1','Beta0','Beta1')

% With Noise
theta_hat = zeros(4,m*N) ;
yg_h = zeros(1,m*N) ;
e_g = zeros(m*N,1) ;
norm_eg = zeros(m*N,1) ;
flag = zeros(m*N,1) ;


for k = 1:m*N-2
    theta_hat(:,1) = theta_est ;
    mu(k,:) = abs(1/(phi_t(k+1,:)*phi_t(k+1,:)')-.1) ;
    e_g(k+2,:) = phi_t(k+1,:)*(theta-theta_hat(:,k)) ;
    theta_tilde(:,k) = (theta-theta_hat(:,k)) ;
    norm_eg(k,:) = theta_tilde(:,k)'*theta_tilde0(:,k) ;
    theta_hat(:,k+1) = theta_hat(:,k)+2*mu(k,:)*phi_t(k+1,:)'*e_g(k+2,:) ;
end

k = 1:N ;
figure(6)
semilogx(k,norm_eg(1:N)')
title('Error Norm w/Noise')
xlabel('Interation Index, k')
ylabel('Error Norm Magnitude (abs)')

figure(7)
semilogx(k,e_g(1:N,1)')
title('Prediction Error w/Noise')
xlabel('Interation Index, k')
ylabel('Prediction Error (abs)')

figure(8)
plot(k,theta_hat(:,1:N))
title('Iteration History w/Noise')
xlabel('Interation Index, k')
ylabel('Parameter Estimate')
legend('-Alpha0','-Alpha1','Beta0','Beta1')

% Transfer Function Comparison
num0 = [theta_hat0(4,m*N-1) theta_hat0(3,m*N-1)] ;
den0 = [1 -theta_hat0(2,m*N-1) -theta_hat0(1,m*N-1)] ;
tf_est0 = tf(num0,den0,delt) ;
```

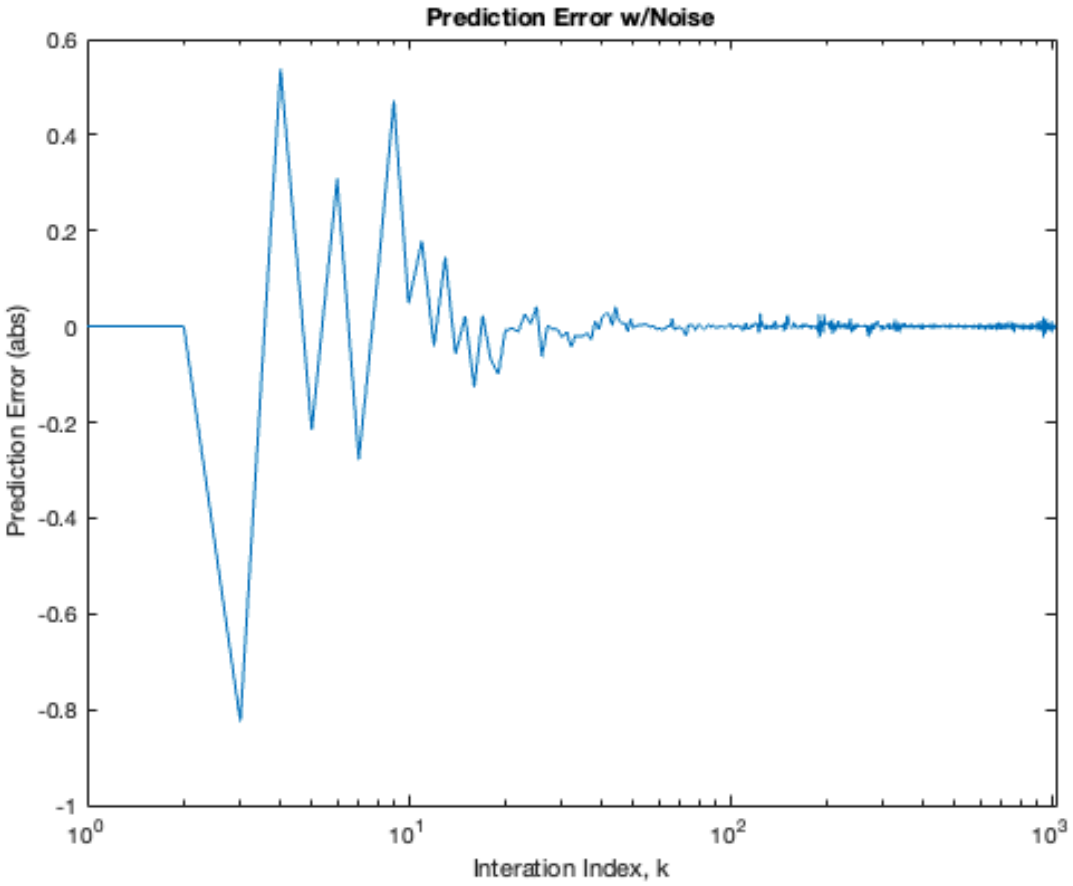
```
num1 = [theta_hat(4,m*N-1) theta_hat(3,m*N-1)] ;
den1 = [1 -theta_hat(2,m*N-1) -theta_hat(1,m*N-1)] ;
tf_est = tf(num1,den1,delt) ;

figure(9)
bode(tf_ol)
hold on
bode(tf_est0)
bode(tf_est)
legend('Plant','Estimated Plant w/o Noise','Estimated Plant w/Noise')
```

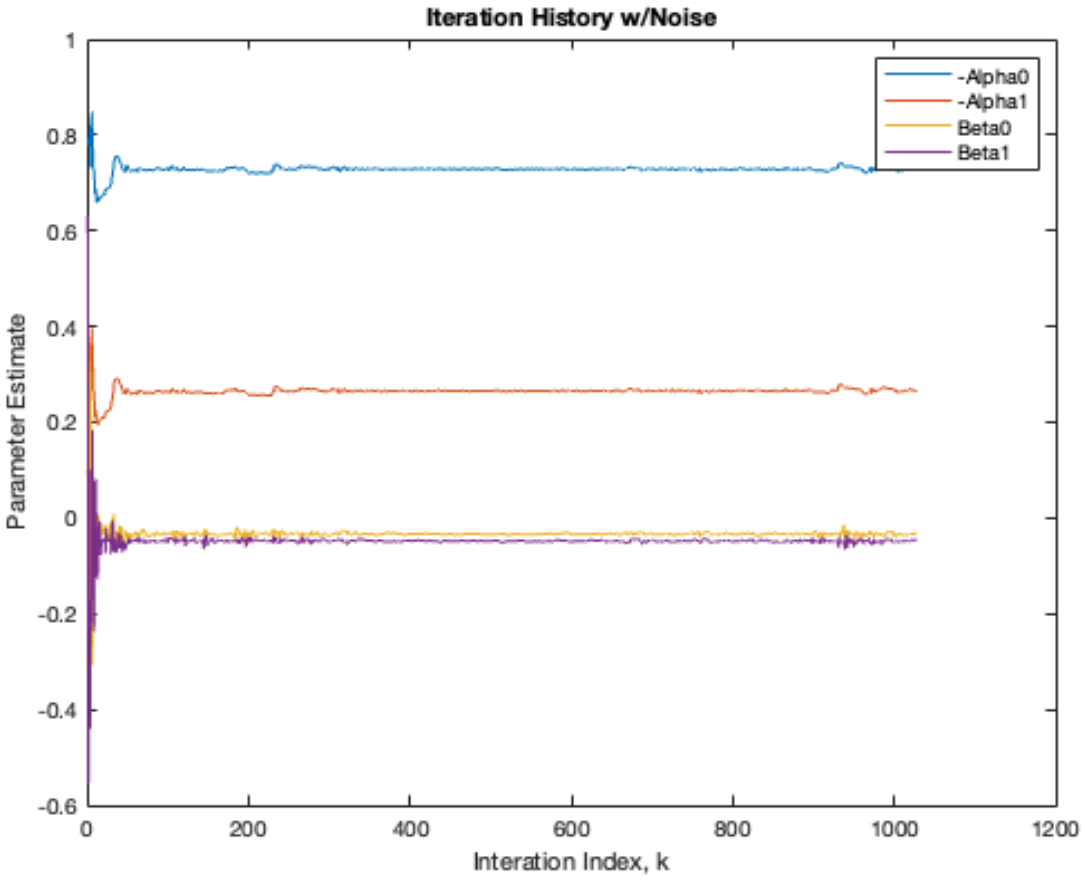


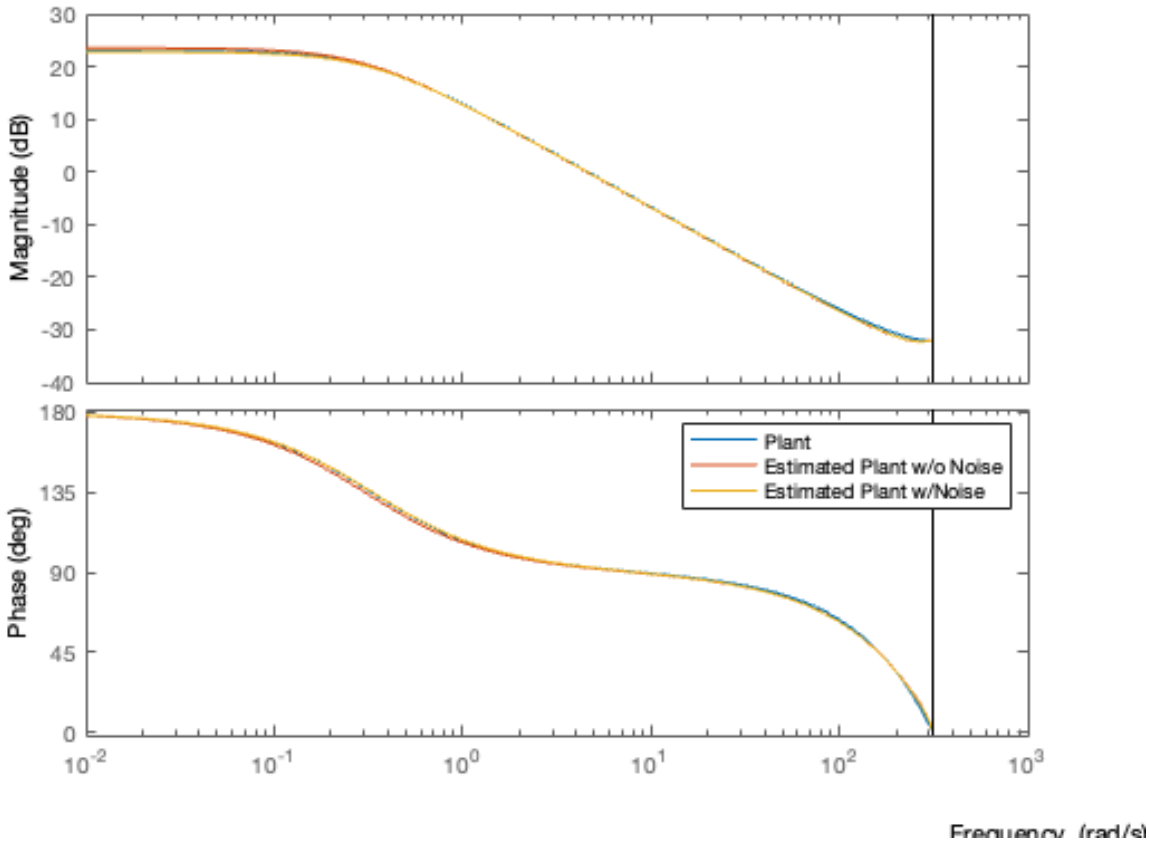

Prediction Error w/o Noise
Prediction Error (abs)
Interation Index, k
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10^0
10^1
10^2
10^3


Iteration History w/o Noise
-Alpha0
-Alpha1
Beta0
Beta1
Parameter Estimate
Interation Index, k
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
200
400
600
800
1000
1200


Error Norm w/Noise
Error Norm Magnitude (abs)
2.5
2.4
2.3
2.2
2.1
2
1.9
1.8
10^0
10^1
10^2
10^3
Interation Index, k


Prediction Error w/Noise
Prediction Error (abs)
Interation Index, k
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
10^0
10^1
10^2
10^3

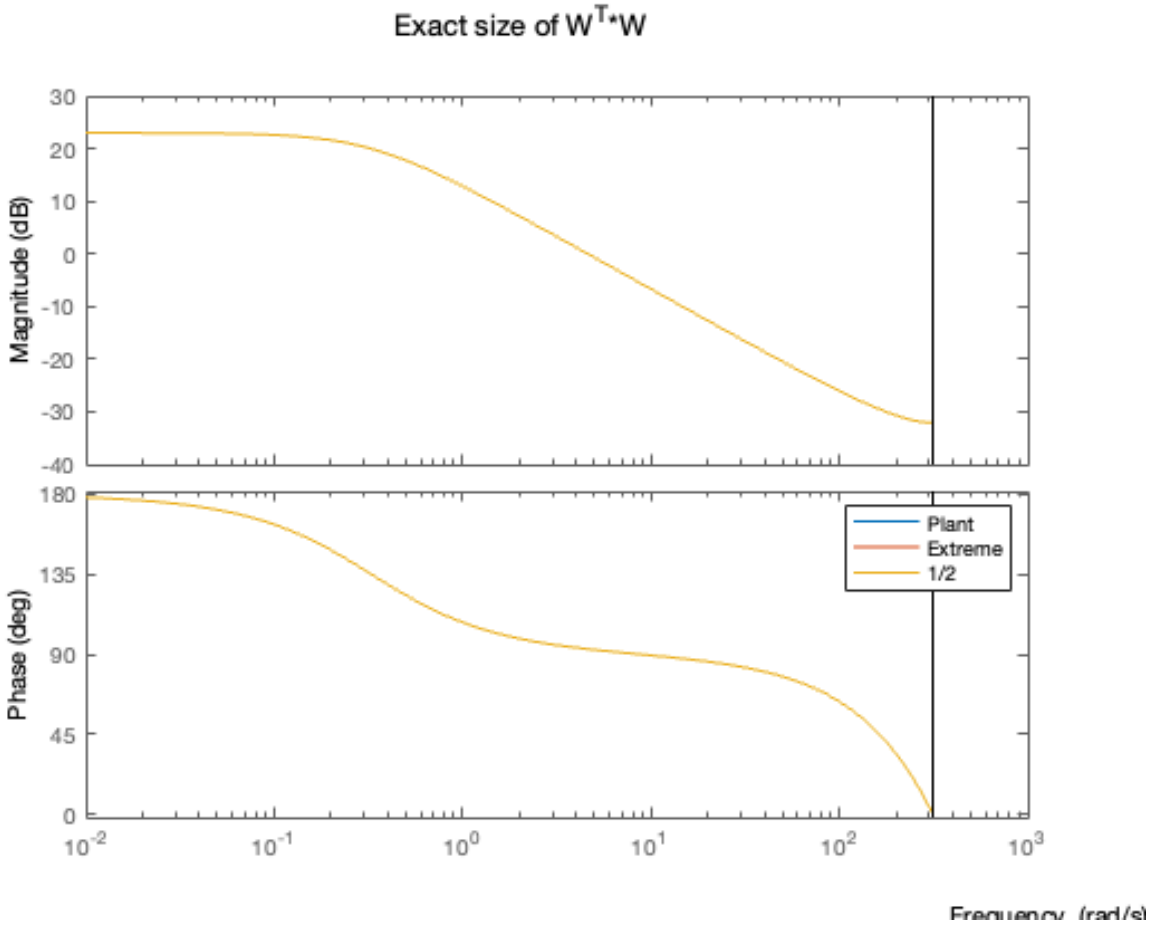

Iteration History w/Noise
-Alpha0
-Alpha1
Beta0
Beta1
Parameter Estimate
Interation Index, k
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
200
400
600
800
1000
1200

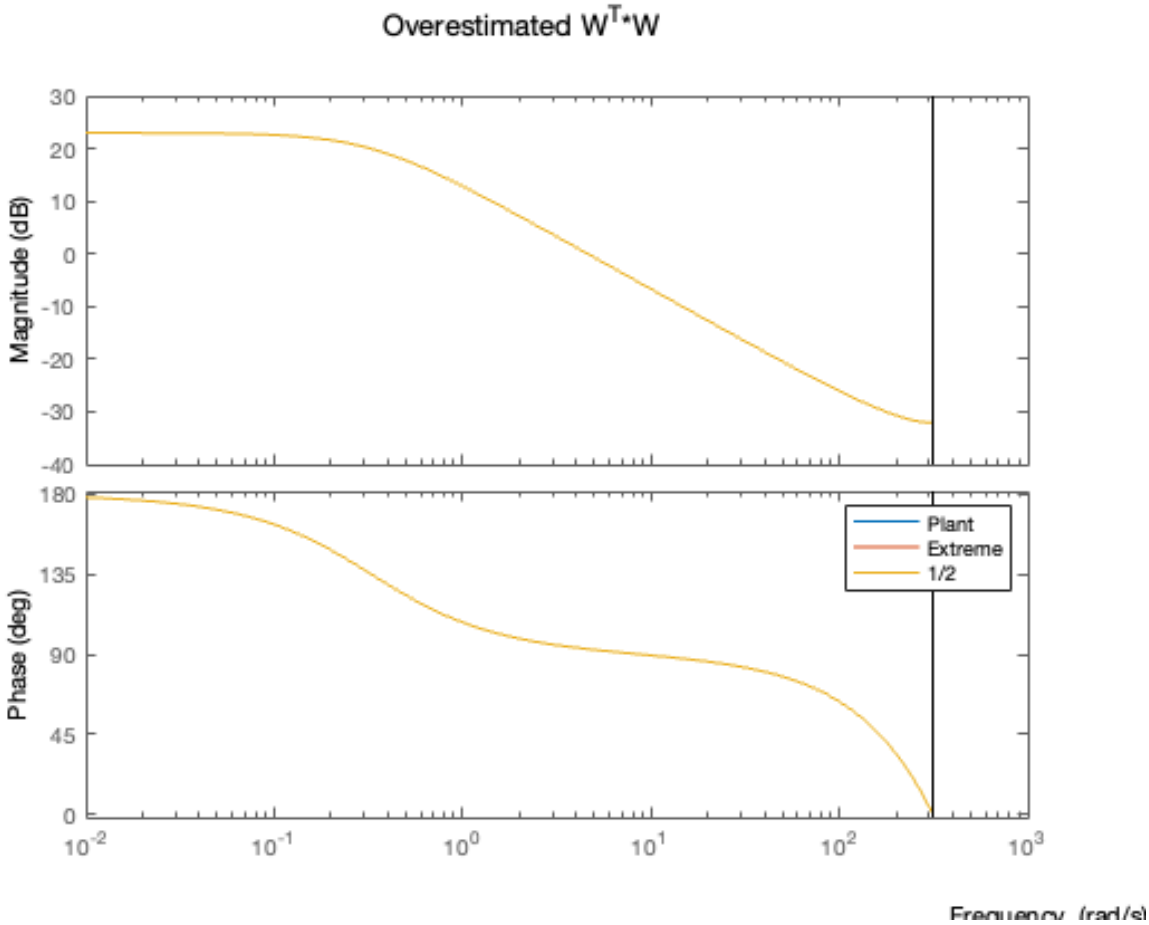


Batch

```
% Batch Least Squares
theta_b = (phi_t'*phi_t)^-1*phi_t'*y ;
rnk = rank(phi_t'*phi_t) ;      % full rank ---> persistent excitation!

% W^T*W size known
ETE1 = (y-phi_t*theta_b)'*(y-phi_t*theta_b) ;
R = (phi_t'*phi_t) ;
eige = eig(R) ;

delta = 1 ;
d_thetaa = (abs(delta*ETE1/eige(1,1)))^(1/2) ;
d_thetab = (abs(delta*ETE1/eige(2,1)))^(1/2) ;
d_thetac = (abs(delta*ETE1/eige(3,1)))^(1/2) ;
d_thetad = (abs(delta*ETE1/eige(4,1)))^(1/2) ;
d_theta_e = [d_thetaa; d_thetab; d_thetac; d_thetad] ;
theta_eset1 = d_theta_e+theta_b ;
theta_eset2 = .5*d_theta_e+theta_b ;

num_eset1 = [theta_eset1(4,1) theta_eset1(3,1)] ;
den_eset1 = [1 -theta_eset1(2,1) -theta_eset1(1,1)] ;
```

```
tf_eset1 = tf(num_eset1,den_eset1,delt) ;

num_eset2 = [theta_eset2(4,1) theta_eset2(3,1)] ;
den_eset2 = [1 -theta_eset2(2,1) -theta_eset2(1,1)] ;
tf_eset2 = tf(num_eset2,den_eset2,delt) ;

figure(10)
sgtitle('Exact size of W^T*W')
bode(tf_ol)
hold on
bode(tf_eset1)
bode(tf_eset2)
legend('Plant','Extreme','1/2')


% W^T*W + 10%
delta = .1 ;                    % actually delta^2
d_theta1 = (abs(delta*ETE1/eige(1,1)))^(1/2) ;
d_theta2 = (abs(delta*ETE1/eige(2,1)))^(1/2) ;
d_theta3 = (abs(delta*ETE1/eige(3,1)))^(1/2) ;
d_theta4 = (abs(delta*ETE1/eige(4,1)))^(1/2) ;
d_theta = [d_theta1; d_theta2; d_theta3; d_theta4] ;
theta_set1 = d_theta+theta_b ;
theta_set2 = .5*d_theta+theta_b ;

num_set1 = [theta_set1(4,1) theta_set1(3,1)] ;
den_set1 = [1 -theta_set1(2,1) -theta_set1(1,1)] ;
tf_set1 = tf(num_set1,den_set1,delt) ;

num_set2 = [theta_set2(4,1) theta_set2(3,1)] ;
den_set2 = [1 -theta_set2(2,1) -theta_set2(1,1)] ;
tf_set2 = tf(num_set2,den_set2,delt) ;

figure(11)
sgtitle('Overestimated W^T*W')
bode(tf_ol)
hold on
bode(tf_set1)
bode(tf_set2)
legend('Plant','Extreme','1/2')
```


Exact size of W^T*W
Magnitude (dB)
30
20
10
0
-10
-20
-30
-40
Phase (deg)
180
135
90
45
0
10^-2
10^-1
10^0
10^1
10^2
10^3
Plant
Extreme
1/2



Published with MATLAB® R2023a

References

[1] T. M. I. Hakim and O. Arifianto, "Implementation of dryden continuous turbulence model into simulink for lsa-02 flight test simulation," *Journal of Physics: Conf. Ser.*, no. 1005, pp. 3–6, 2017.