```
% Defining the 12 Aircraft States
syms p_n p_e p_d % Inertial positions of the aircraft
syms u v w % Ground Velocity of the drone
syms phi theta psi % Roll angle, Pitch angle, Yaw angle
syms p q r % Body angular rate, roll rate, pitch rate, yaw rate

syms p_dot_n p_dot_e p_dot_d % Inertial positions of the aircraft
syms u_dot v_dot w_dot % Ground Velocity of the drone
syms phi_dot theta_dot psi_dot % Roll angle, Pitch angle, Yaw angle
syms p_dot q_dot r_dot % Body angular rate, roll rate, pitch rate, yaw rate

syms n_1 n_2 n_3 n_4 % Inputs, Propeller speeds

syms J_x J_y J_z J_xz % Moments of inertia about the drone's axis

syms L M N  % Moments about the body frame axes

syms m d g

syms f_x f_y f_z

syms x_dot x

syms C_T C_D

syms I_xx I_yy I_zz

syms u omega_1 omega_2 omega_3 omega_4

syms F_z M_x M_y M_z

syms e_dot e

sympref('FloatingPointOutput',true)
```

```
ans = logical
   1
```

```
% For plotting
dd=0.:.001:2*pi;
xx1=cos(dd)-1;yy1=sin(dd);


% The states vector
X = [p_n;...
          p_e;...
          p_d;...
          psi;...
          theta;...
          phi;...
```

```
          u;...
          v;...
          w;...
          r;...
          q;...
          p;];

X_dot = [p_dot_n;...
          p_dot_e;...
          p_dot_d;...
          psi_dot;...
          theta_dot;...
          phi_dot;...
          u_dot;...
          v_dot;...
          w_dot;...
          r_dot;...
          q_dot;...
          p_dot;];

s = ["x = X"];
displayFormula(["'The 12 states of the drone: '"; s]);
```

The 12 states of the drone:

$$x = \begin{pmatrix} p_n \\ p_e \\ p_d \\ \psi \\ \theta \\ \phi \\ u \\ v \\ w \\ r \\ q \\ p \end{pmatrix}$$

```
s = ["x_dot = X_dot"];
displayFormula(["'The 12 state derivatives of the drone: '"; s]);
```

The 12 state derivatives of the drone:

$$\dot{x} = \begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{r} \\ \dot{q} \\ \dot{p} \end{pmatrix}$$

```
X_dot_ = [w*(sin(phi)*sin(psi) + cos(phi)*cos(psi)*sin(theta))
- v*(cos(phi)*sin(psi) - cos(psi)*sin(phi)*sin(theta)) +
u*cos(psi)*cos(theta);
    v*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta))
- w*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*sin(theta)) +
u*cos(theta)*sin(psi);
    w*cos(phi)*cos(theta) - u*sin(theta) + v*cos(theta)*sin(phi);
    (r*cos(phi))/cos(theta) + (q*sin(phi))/cos(theta);
    q*cos(phi) - r*sin(phi);
    p + (r*cos(phi)*sin(theta))/cos(theta) + (q*sin(phi)*sin(theta))/
cos(theta);
    r*v - q*w + f_x/m + g*theta;
    p*w - r*u + f_y/m - g*phi;
    q*u - p*v + f_z/m;
    N - (J_x*(q*(J_x*p - J_xz*r) - J_y*p*q))/(J_xz^2 - J_x*J_z) -
(J_xz*(q*(J_xz*p - J_z*r) + J_y*q*r))/(J_xz^2 - J_x*J_z);
    L - (J_xz*(q*(J_x*p - J_xz*r) - J_y*p*q))/(J_xz^2 - J_x*J_z) -
(J_z*(q*(J_xz*p - J_z*r) + J_y*q*r))/(J_xz^2 - J_x*J_z);
    M - (p*(J_xz*p - J_z*r) + r*(J_x*p - J_xz*r))/J_y];
s = ["x_dot = X_dot = X_dot_"];
displayFormula(["'The 12 state derivatives of the drone: '"; s]);
```

The 12 state derivatives of the drone:

$$\dot{x} = \begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{r} \\ \dot{q} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} w\ (\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)) - v\ (\cos(\phi)\sin(\psi) - \cos(\psi)\sin(\phi)\sin(\theta)) + u\cos(\\ v\ (\cos(\phi)\cos(\psi) + \sin(\phi)\sin(\psi)\sin(\theta)) - w\ (\cos(\psi)\sin(\phi) - \cos(\phi)\sin(\psi)\sin(\theta)) + u\cos(\\ w\cos(\phi)\cos(\theta) - u\sin(\theta) + v\cos(\theta)\sin(\phi) \\ \frac{r\cos(\phi)}{\cos(\theta)} + \frac{q\sin(\phi)}{\cos(\theta)} \\ q\cos(\phi) - r\sin(\phi) \\ p + \frac{r\cos(\phi)\sin(\theta)}{\cos(\theta)} + \frac{q\sin(\phi)\sin(\theta)}{\cos(\theta)} \\ g\,\theta - q\,w + r\,v + \frac{f_x}{m} \\ p\,w - g\,\phi - r\,u + \frac{f_y}{m} \\ q\,u - p\,v + \frac{f_z}{m} \\ N - \frac{J_x\ (q\ (J_x\,p - J_{xz}\,r) - J_y\,p\,q)}{{J_{xz}}^2 - J_x\,J_z} - \frac{J_{xz}\ (q\ (J_{xz}\,p - J_z\,r) + J_y\,q\,r)}{{J_{xz}}^2 - J_x\,J_z} \\ L - \frac{J_{xz}\ (q\ (J_x\,p - J_{xz}\,r) - J_y\,p\,q)}{{J_{xz}}^2 - J_x\,J_z} - \frac{J_z\ (q\ (J_{xz}\,p - J_z\,r) + J_y\,q\,r)}{{J_{xz}}^2 - J_x\,J_z} \\ M - \frac{p\ (J_{xz}\,p - J_z\,r) + r\ (J_x\,p - J_{xz}\,r)}{J_y} \end{pmatrix}$$

```
Jacobian = simplify(jacobian(X_dot_,X));

s = ["x_dot = Jacobian"];
displayFormula(["'After Taking the Jacobian:'";s]);
```

After Taking the Jacobian:

$$\dot{x} = \begin{pmatrix} 0 & 0 & 0 & w\,(\cos(\psi)\sin(\phi) - \cos(\phi)\sin(\psi)\sin(\theta)) - v\,(\cos(\phi)\cos(\psi) + \sin(\phi)\sin(\psi)\sin(\theta)) - u\cos \\ 0 & 0 & 0 & w\,(\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)) - v\,(\cos(\phi)\sin(\psi) - \cos(\psi)\sin(\phi)\sin(\theta)) + u\cos \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
A = subs(Jacobian,X,[0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0]);
s = ["X_dot = A*X"];
displayFormula(["'After Plugging in x = 0 into the jacobian:'";s]);
```

After Plugging in x = 0 into the jacobian:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{r} \\ \dot{q} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} p_n \\ p_e \\ p_d \\ \psi \\ \theta \\ \phi \\ u \\ v \\ w \\ r \\ q \\ p \end{pmatrix}$$

```
B = [0 0 0 0;
     0 0 0 0;
     0 0 0 0;
     0 0 0 0;
     0 0 0 0;
     0 0 0 0;
     0 0 0 0;
```

```
    0 0 0 0;
    2*C_T/m 2*C_T/m 2*C_T/m 2*C_T/m;
    -2*C_D/I_zz 2*C_D/I_zz -2*C_D/I_zz 2*C_D/I_zz;
    -sqrt(2)*d*C_T/I_yy sqrt(2)*d*C_T/I_yy sqrt(2)*d*C_T/I_yy
-sqrt(2)*d*C_T/I_yy;
    -sqrt(2)*d*C_T/I_xx -sqrt(2)*d*C_T/I_xx sqrt(2)*d*C_T/I_xx
sqrt(2)*d*C_T/I_xx];
u_ = [omega_1; omega_2; omega_3; omega_4];


s = ["X_dot = A*X + B*u_"];
displayFormula(["'The entire linearized system:'";s]);
```

The entire linearized system:

$$\begin{pmatrix}\dot{p}_n\\ \dot{p}_e\\ \dot{p}_d\\ \dot{\psi}\\ \dot{\theta}\\ \dot{\phi}\\ \dot{u}\\ \dot{v}\\ \dot{w}\\ \dot{r}\\ \dot{q}\\ \dot{p}\end{pmatrix} = \begin{pmatrix}0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&g&0&0&0&0&0&0&0\\ 0&0&0&0&0&-g&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\end{pmatrix}\begin{pmatrix}p_n\\ p_e\\ p_d\\ \psi\\ \theta\\ \phi\\ u\\ v\\ w\\ r\\ q\\ p\end{pmatrix} + \begin{pmatrix}0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ \frac{2\,C_T}{m}&\frac{2\,C_T}{m}&\frac{2\,C_T}{m}&\frac{2\,C_T}{m}\\ -\frac{2\,C_D}{I_{zz}}&\frac{2\,C_D}{I_{zz}}&-\frac{2\,C_D}{I_{zz}}&\frac{2\,C_D}{I_{zz}}\\ -\frac{\sqrt{2}\,C_T\,d}{I_{yy}}&\frac{\sqrt{2}\,C_T\,d}{I_{yy}}&\frac{\sqrt{2}\,C_T\,d}{I_{yy}}&-\frac{\sqrt{2}\,C_T\,d}{I_{yy}}\\ -\frac{\sqrt{2}\,C_T\,d}{I_{xx}}&-\frac{\sqrt{2}\,C_T\,d}{I_{xx}}&\frac{\sqrt{2}\,C_T\,d}{I_{xx}}&\frac{\sqrt{2}\,C_T\,d}{I_{xx}}\end{pmatrix}$$

```
C_T_ = 3.1582e-10; % N/rpm^2
C_D_ = 7.9379e-12; % Nm/rpm^2
m_ = 0.27; % kg
d_ = 39.73e-3; % m
r_ = 23.1348e-3; %m
I_xx_ = 1.395e-5; % kg*m^2
I_yy_ = 1.436e-5; % kg*m^2
I_zz_ = 2.173e-5; % kg*m^2
K_T_ = 0.2025;
K_D_ = 0.11;
w_e = 1;
g_ = 9.81;
u__ = 1/(2*w_e)*[1/4]
```

```
u__ = 0.1250
```

```
variables = [C_T; C_D; I_xx; I_yy; I_zz; d; m];
```

```
variables_ = [3.1582e-10; 7.9379e-12; 1.395e-5; 1.436e-5; 2.173e-5;
39.73e-3; 0.27];
B_subs = subs(B,variables,variables_)
```

B_subs =

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2.3394\text{e-}09 & 2.3394\text{e-}09 & 2.3394\text{e-}09 & 2.3394\text{e-}09 \\ -7.3059\text{e-}07 & 7.3059\text{e-}07 & -7.3059\text{e-}07 & 7.3059\text{e-}07 \\ -1.2357\text{e-}06 & 1.2357\text{e-}06 & 1.2357\text{e-}06 & -1.2357\text{e-}06 \\ -1.2720\text{e-}06 & -1.2720\text{e-}06 & 1.2720\text{e-}06 & 1.2720\text{e-}06 \end{pmatrix}$$

```
var = double(B_subs)
```

```
var = 12×4
10^-5 ×
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
    0.0002    0.0002    0.0002    0.0002
   -0.0731    0.0731   -0.0731    0.0731
    ⋮
```

```
Ap = [0 0 0 1 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 1;
      0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 0 0;]
```

```
Ap = 6×6
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

```
Bp = [0    0    0    0;
0    0    0    0;
0    0    0    0;
-7.3059e-07    7.3059e-07    -7.3059e-07    7.3059e-07;
```

```
-1.2357e-06    1.2357e-06    1.2357e-06    -1.2357e-06;
-1.2720e-06    -1.2720e-06    1.2720e-06    1.2720e-06;]
```

```
Bp = 6×4
10^-5 ×
         0         0         0         0
         0         0         0         0
         0         0         0         0
   -0.0731    0.0731   -0.0731    0.0731
   -0.1236    0.1236    0.1236   -0.1236
   -0.1272   -0.1272    0.1272    0.1272
```

Vertical Subsystem

This subsystem describes the dynamics of the upward movements of the crazyflie drone. This subsytem is primariliy affected by the thrust inputted into the drone.

```
Ap = [0 1;
    0 0];
Bp = [0;
    1/m];

s = ["[z_dot;w_dot] = Ap*[z;w] + Bp*F_z"];
displayFormula(["'The upward Thrust dynamics:'";s]);
```

The upward Thrust dynamics:

$$\begin{pmatrix}\dot{z}\\ \dot{w}\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}z\\ w\end{pmatrix} + \begin{pmatrix}0\\ \frac{1}{m}\end{pmatrix}F_z$$

```
Bp = double(subs(Bp,m,m_));
s = ["[z_dot;w_dot] = Ap*[z;w] + Bp*F_z"];
displayFormula(["'The upward Thrust dynamics:'";s]);
```

The upward Thrust dynamics:

$$\begin{pmatrix}\dot{z}\\ \dot{w}\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}z\\ w\end{pmatrix} + \begin{pmatrix}0\\ \frac{100}{27}\end{pmatrix}F_z$$

```
Co = ctrb(Ap,Bp);
unco = length(Ap) - rank(Co);

Cp = [1 0
      0 1];

Dp = zeros(2,1);

% Wiggle System, Adding an error intgral state
Aw = [zeros(1,1) Cp(1,:);
```

```
     zeros(2,1) Ap];
Bw = [Dp(1,:);
      Bp];
s = ["[e_dot;z_dot;w_dot] = Aw*[e;z;w] + Bw*F_z"];
displayFormula(["'The upward Thrust dynamics with an en error
integrator:'";s]);
```

The upward Thrust dynamics with an en error integrator:

$$\begin{pmatrix} \dot{e} \\ \dot{z} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ z \\ w \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{100}{27} \end{pmatrix} F_z$$

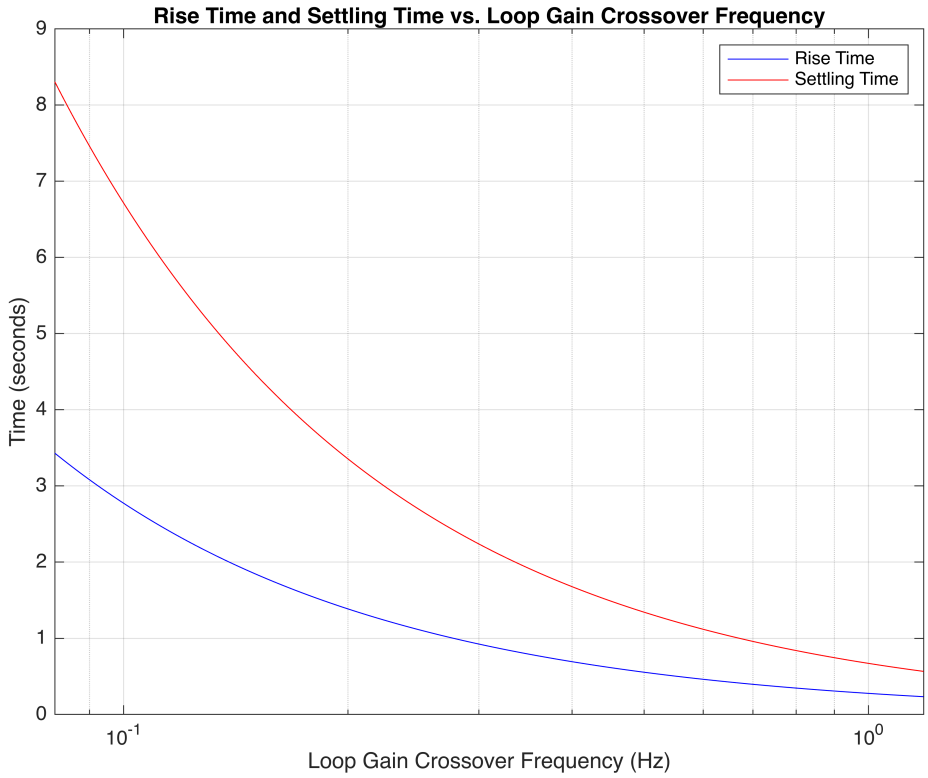
```
% Setup range of penalties for the LQR
Q=0.*Aw; %sets up a matrix Q that is the size of Aw and is all 0s
R=1;
num_step = 200;
%qq is a vector which each element scales the LQR Q matrix
qq=logspace(-2, 5,num_step); %these are the varying values of q11
step_info = zeros(size(qq,2),6);
poles = zeros(size(qq,2),3);
margins = zeros(size(qq,2),4);
mins = zeros(size(qq,2),2);
gains = zeros(size(qq,2),3);
for ii = 1:num_step

    % The only penalty is the 1,1 element on the error state
    % Desing the gains
    Q(1,1)=qq(ii);
    [Kx_lqr,~,~]=lqr(Aw,Bw,Q,R);

    % populate the controller matrices
    Ac = 0.;
    Bc1 = [1. 0. ];
    Bc2 = -1;
    Cc = -Kx_lqr(1);
    Dc1 = [-Kx_lqr(2:3)];
    Dc2 = 0;

    Zi = inv(eye(1) - Dc1*Dp);
    Acl = [     Ap + Bp*Zi*Dc1*Cp                Bp*Zi*Cc;
             Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
    Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
    Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
    Dcl = Dp*Zi*Dc2;
    sys_cl = ss(Acl,Bcl,Ccl,Dcl);

    % at plant input
```

```
    Alu = [Ap     zeros(2,1);
           Bc1*Cp     Ac];
    Blu = [  Bp;
           zeros(1,1)];
    Clu = -[Dc1*Cp Cc];
    Dlu = 0;
    sys_u = ss(Alu,Blu,Clu,Dlu);

    % Loop input break Info
    lu_margins = allmargin(sys_u);
    gm_lu = 20*log10(lu_margins.GainMargin);
    pm_lu = lu_margins.PhaseMargin;
    lgcf_lu = lu_margins.PMFrequency;
    pc_lu = lu_margins.GMFrequency;

    % SV Info
    [a,b] = ss2tf(Alu,Blu,Clu,Dlu);
    c = b + a;
    sys_IL = tf(c,b);
    alpha_0 = min(sigma(sys_IL));

    sys_ILi = tf(c,a);
    beta_0 = (min(sigma(sys_ILi)));

    info = stepinfo(sys_cl,'RiseTimeThreshold', [0
0.63],'SettlingTimeThreshold', 0.05);
    step_info(ii,:) = [info(1).RiseTime info(1).SettlingTime
info(2).RiseTime, info(2).SettlingTime info(1).Overshoot
info(1).Undershoot];
    poles(ii,:) = eig(Acl);
    margins(ii,:) = [gm_lu pm_lu lgcf_lu pc_lu];
    mins(ii,:) = [alpha_0 beta_0];
    gains(ii,:) = Kx_lqr;

end

rise_time = step_info(:,1); % Rise time for the first output
settling_time = step_info(:,2); % Settling time for the first output
lgcf_lu = margins(:,4)./(2*pi); % lgcf in Hz
overshoot = step_info(:,5);
undershoot = step_info(:,6);
```

Rise time (blue) and settling time (red) versus loop gain crossover frequency.

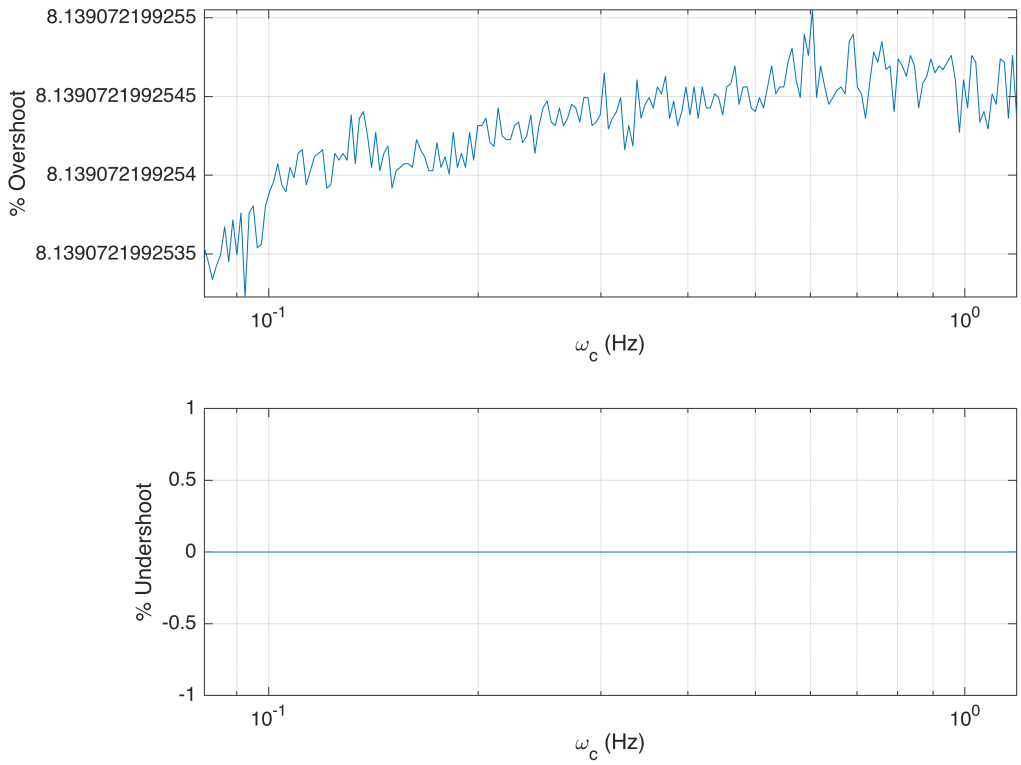
```
% Plotting
figure;
plot(lgcf_lu, rise_time, 'b'); % Plot Rise Time in blue
hold on; % Holds the current plot to allow multiple plots in the same figure
plot(lgcf_lu, settling_time, 'r'); % Plot Settling Time in red
xlabel('Loop Gain Crossover Frequency (Hz)'); % X-axis label
```

```
ylabel('Time (seconds)'); % Y-axis label
title('Rise Time and Settling Time vs. Loop Gain Crossover Frequency'); %
Title of the plot
legend('Rise Time', 'Settling Time'); % Legend to differentiate between the
two plots
set(gca, 'XScale', 'log');
grid on; % Adds a grid to the plot for better readability
hold off;
```



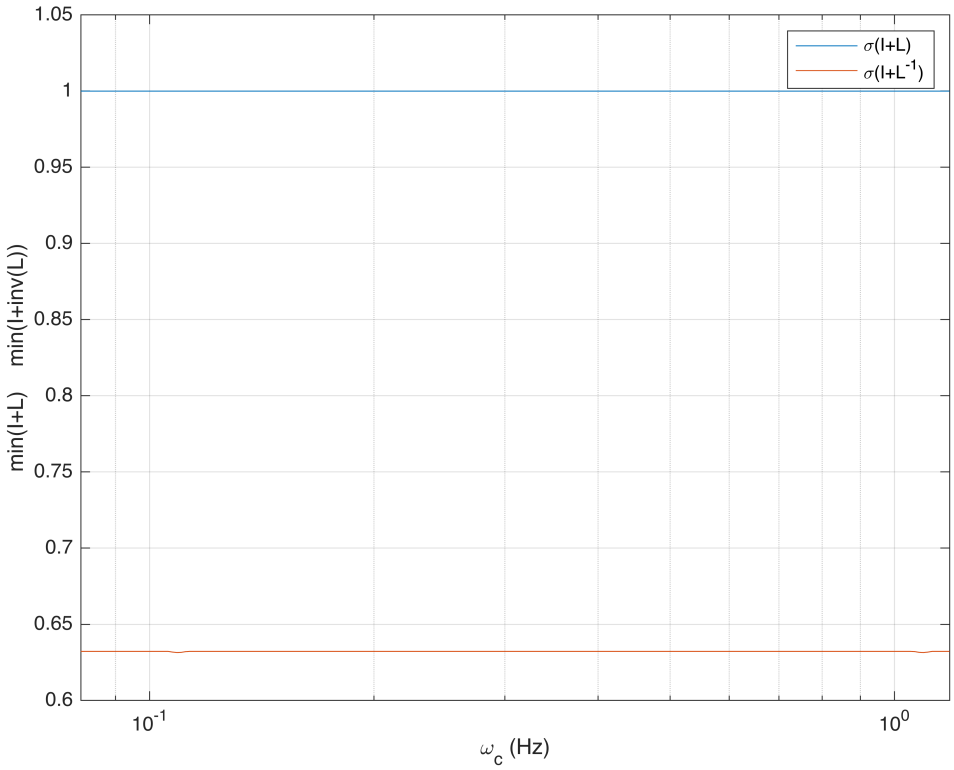
Percent overshoot, undershoot versus loop gain crossover frequency ω_c

```
figure;
subplot(2,1,1),
plot(lgcf_lu, overshoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Overshoot');
xlabel('\omega_{c} (Hz)');
subplot(2,1,2),
plot(lgcf_lu, undershoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Undershoot');
xlabel('\omega_{c} (Hz)');
```



Singular value frequency domain metrics versus loop gain crossover frequency.

```
figure;
plot(lgcf_lu,mins(:,1))
hold on;
plot(lgcf_lu,mins(:,2))
ylabel('min(I+L)    min(I+inv(L))');
legend('\sigma(I+L)','\sigma(I+L^{-1})'); % Using TeX interpreter with an
underline
set(gca, 'XScale', 'log'), grid on;
xlabel('\omega_{c} (Hz)'); % X-axis label
hold off;
```



States of the system responding to a unit acceleration step command.

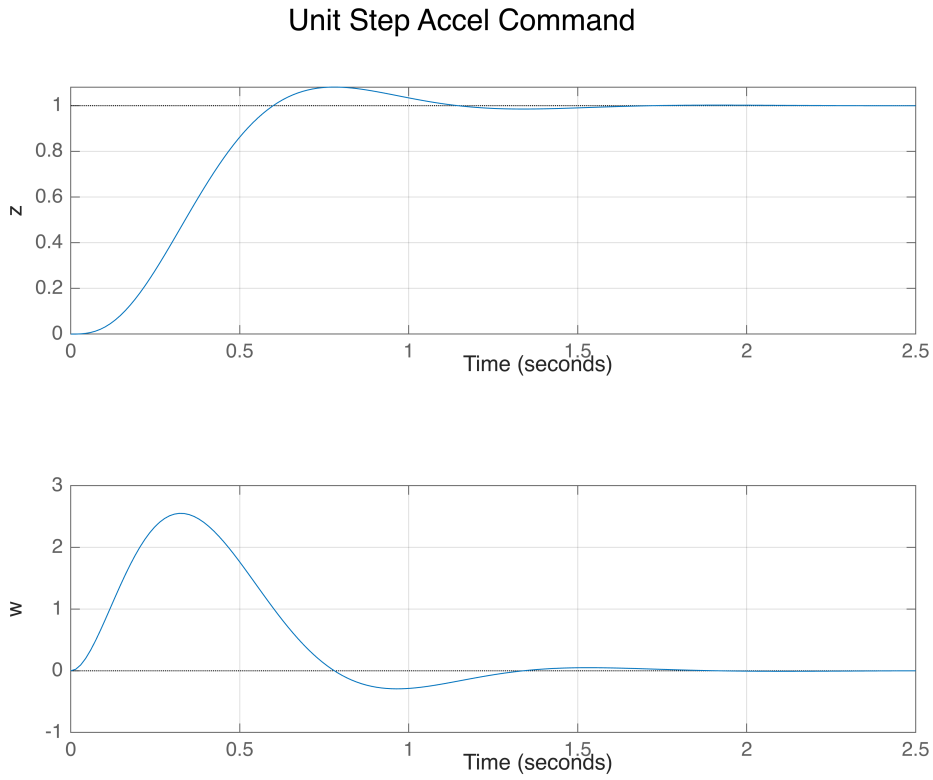
```
Kx_lqr = gains(162,:)
```

```
Kx_lqr = 1×3
   67.8669   21.5075    3.4079
```

```
% populate the controller matrices
```

```
% populate the controller matrices
Ac = 0.;
Bc1 = [1. 0. ];
Bc2 = -1;
Cc = -Kx_lqr(1);
Dc1 = [-Kx_lqr(2:3)];
Dc2 = 0;

Zi = inv(eye(1) - Dc1*Dp);
Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
            Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
Dcl = Dp*Zi*Dc2;
sys_cl = ss(Acl,Bcl,Ccl,Dcl);

figure;
```

```
sgtitle('Unit Step Accel Command')
subplot(2,1,1);
step(sys_cl(1,1));
xlim([0 2.5]), grid on;
ylabel('z'), title('')

subplot(2,1,2);
step(sys_cl(2,1));
xlim([0 2.5]), grid on;
ylabel('w'), title('')
```



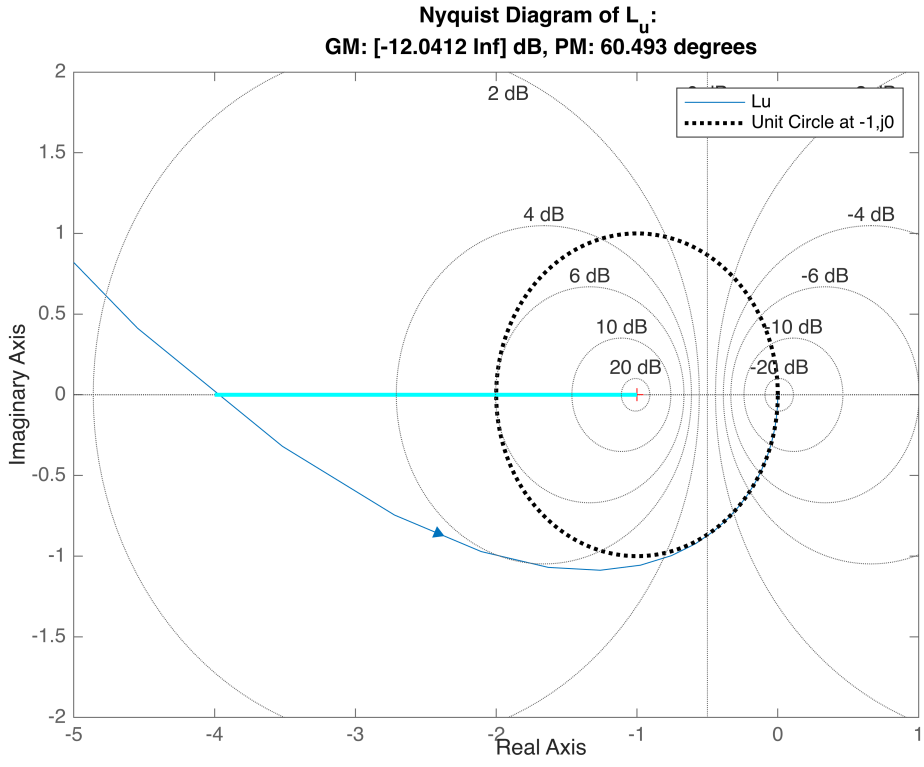
<u>Final Design Analysis</u> ,Break the loop at the plant input and evaluate the design from in the frequency domain.

```
% at plant input
Alu = [Ap     zeros(2,1);
          Bc1*Cp     Ac];
Blu = [  Bp;
          zeros(1,1)];
Clu = -[Dc1*Cp Cc];
Dlu = 0;
sys_u = ss(Alu,Blu,Clu,Dlu);

lu_margins = allmargin(sys_u);
gm_lu = 20*log10(lu_margins.GainMargin);
pm_lu = lu_margins.PhaseMargin;
```

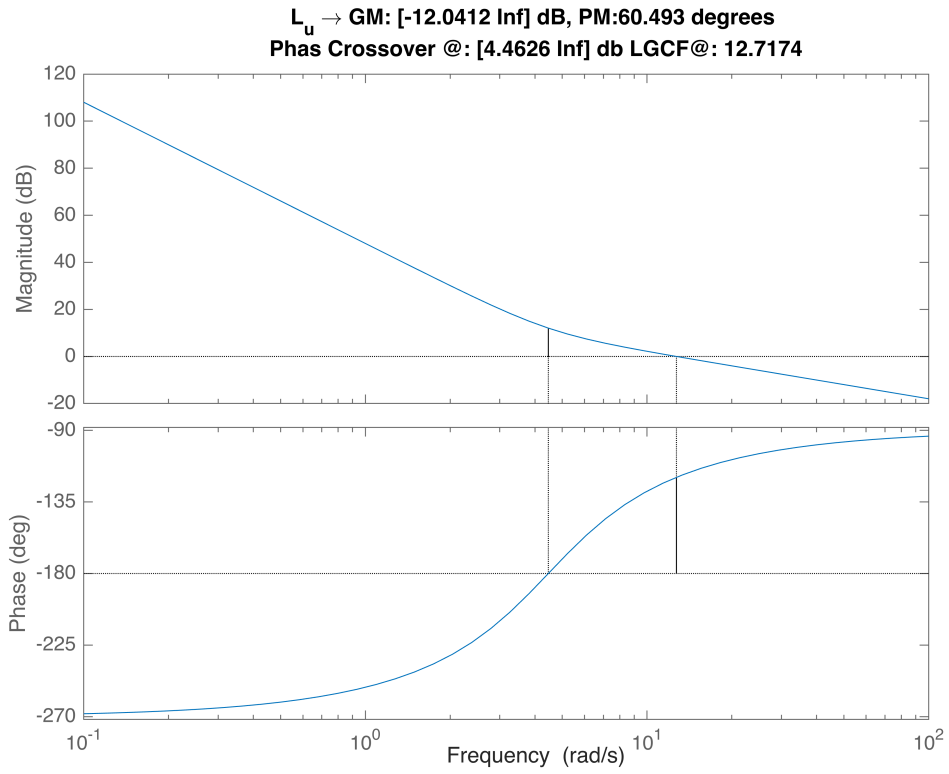
```
lgcf_lu = lu_margins.PMFrequency;
pc_lu = lu_margins.GMFrequency;
```

i) Plot a Nyquist plot and Bode plot for u L . Identify the LGCF, Phase crossover frequency.

```
figure;
n = nyquistplot(sys_u);
hold on;
plot(xx1,yy1,'k:',[-1 -1/lu_margins.GainMargin(1)],[0.
0.],'c','LineWidth',2);grid
xlim([-5,1]), ylim([-2,2])
setoptions(n,'ShowfullContour','off'), grid on
title(['Nyquist Diagram of L_u: ',...
newline 'GM: [', num2str(gm_lu(1)),' Inf] dB, ' 'PM: ' num2str(pm_lu),'
degrees'])
legend('Lu', 'Unit Circle at -1,j0')
hold off;
```

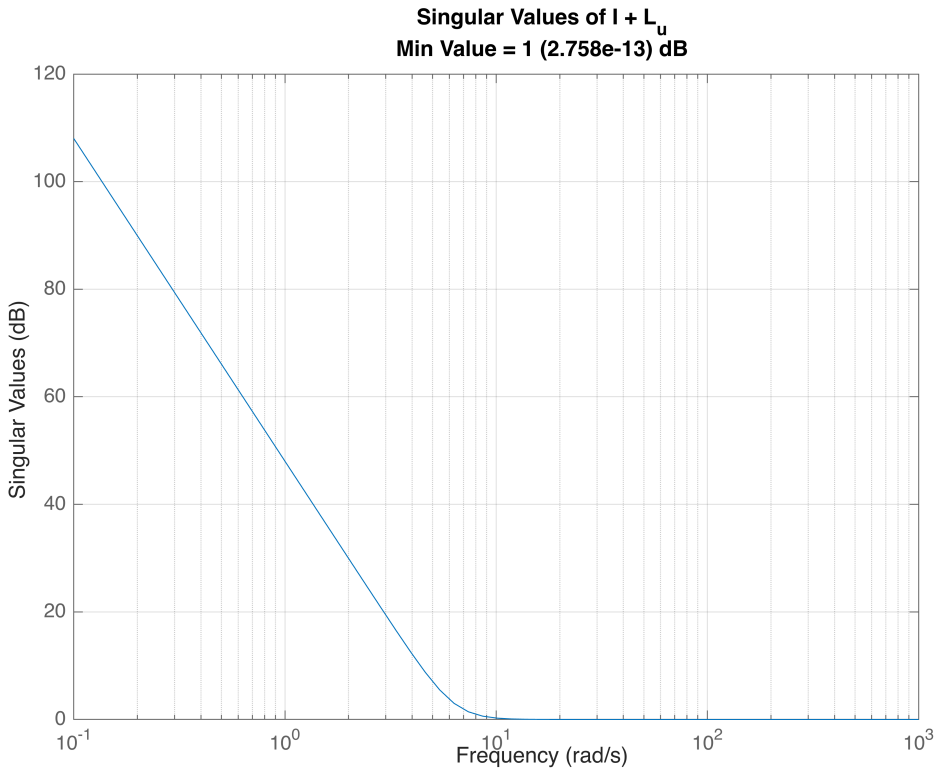



```
figure;
margin(sys_u);
title(['L_u \rightarrow GM: [', num2str(gm_lu(1)), ' Inf] dB, ',
'PM:',num2str(pm_lu(1)), ' degrees ',...
newline 'Phas Crossover @: [', num2str(pc_lu(1)),' Inf] db ','LGCF@:
',num2str(lgcf_lu(1))])
```



ii) Plot $\underline{\sigma}(I + L_u)$ and $\underline{\sigma}\left(I + L_u^{-1}\right)$.

```
[a,b] = ss2tf(Alu,Blu,Clu,Dlu);
c = b + a;
sys_IL = tf(c,b);
alpha_0 = min(sigma(sys_IL));

sigma(sys_IL);
grid on, title(['Singular Values of I + L_u',...
                newline 'Min Value = ',num2str(alpha_0), ' (',
num2str(20*log10(alpha_0)),') dB'])
```



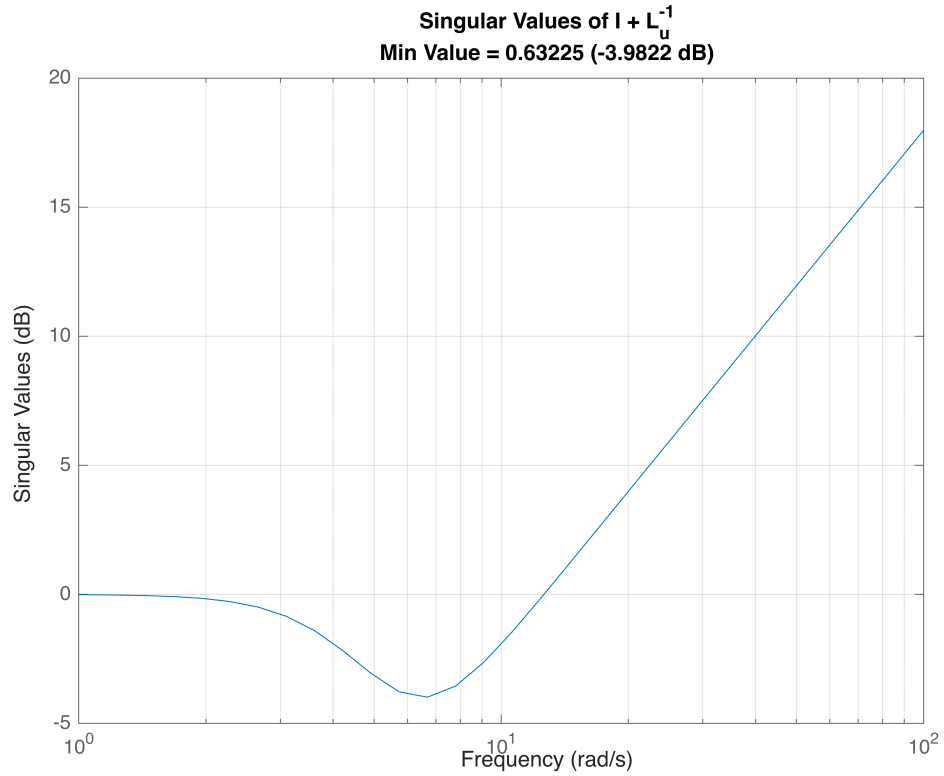

```
text = ['Minimum SV of (I +L_u): ',num2str(20*log10(alpha_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +L_u): 2.758e-13 dB
```

```
pm = 2*asin(alpha_0/2)*180/pi;
GM_l = 20*log10(1/(1 + alpha_0));
GM_u = 20*log10(1/(1 - alpha_0));
Gm_IL = [GM_l GM_u];
pm_IL = [-pm pm];
text = ['Gain Margin from (I+L_u):  [',num2str(Gm_IL(1)),',
',num2str(Gm_IL(2)),'] dB',...
        newline 'Phase Margin from (I+L_u): [',num2str(pm_IL(1)),',
',num2str(pm_IL(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+L_u):  [-6.0206, 269.9645+27.28753i] dB
Phase Margin from (I+L_u): [-60, 60] degrees
```

```
sys_ILi = tf(c,a);
beta_0 = (min(sigma(sys_ILi)));

sigma(sys_ILi)
grid on, title(['Singular Values of I + L_u^{-1}',...
                newline 'Min Value = ',num2str(beta_0), ' (',
num2str(20*log10(beta_0)),' dB)'])
```




```
text = ['Minimum SV of (I +inv(L_u)): ',num2str(20*log10(beta_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +inv(L_u)): -3.9822 dB
```

```
GM_u = 20*log10(1 + beta_0);
GM_l = 20*log10(1 - beta_0);
pm = 2*asin(beta_0/2)*180/pi;
Gm_ILi = [GM_l GM_u];
pm_ILi = [-pm pm];
text = ['Gain Margin from (I+inv(L_u)):  [',num2str(Gm_ILi(1)),',
',num2str(Gm_ILi(2)),'] dB',...
        newline 'Phase Margin from inv((I+L_u)): [',num2str(pm_ILi(1)),',
',num2str(pm_ILi(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+inv(L_u)):  [-8.6889, 4.2557] dB
Phase Margin from inv((I+L_u)): [-36.8574, 36.8574] degrees
```

iii) Compute classical stability margins and singular value stability margins at the plant input.

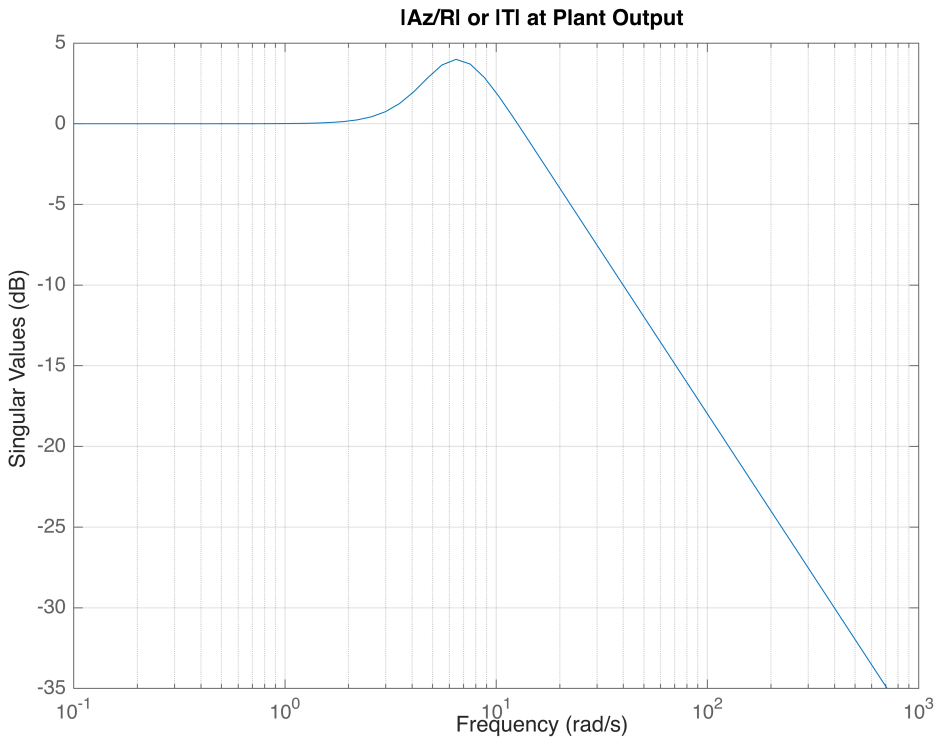
```
Gm_sv = [min(Gm_ILi(1),Gm_IL(1)),max(Gm_ILi(2),Gm_IL(2))];
Pm_sv = [min(pm_ILi(1),pm_IL(1)),max(pm_ILi(2),pm_IL(2))];

text = ['Classical Gain Margin:  [',num2str(gm_lu(1)),', Inf] dB',...
        newline 'Classical Phase Margin: ',num2str(pm_lu),' degrees',...
```

```
        newline 'Singular Values Gain Margin:  [',num2str(Gm_sv(1)),',
',num2str(Gm_sv(2)),'] dB',...
        newline 'Singular Values Phase Margin: [',num2str(Pm_sv(1)),',
',num2str(Pm_sv(2)),'] degrees'];
disp(text);
```

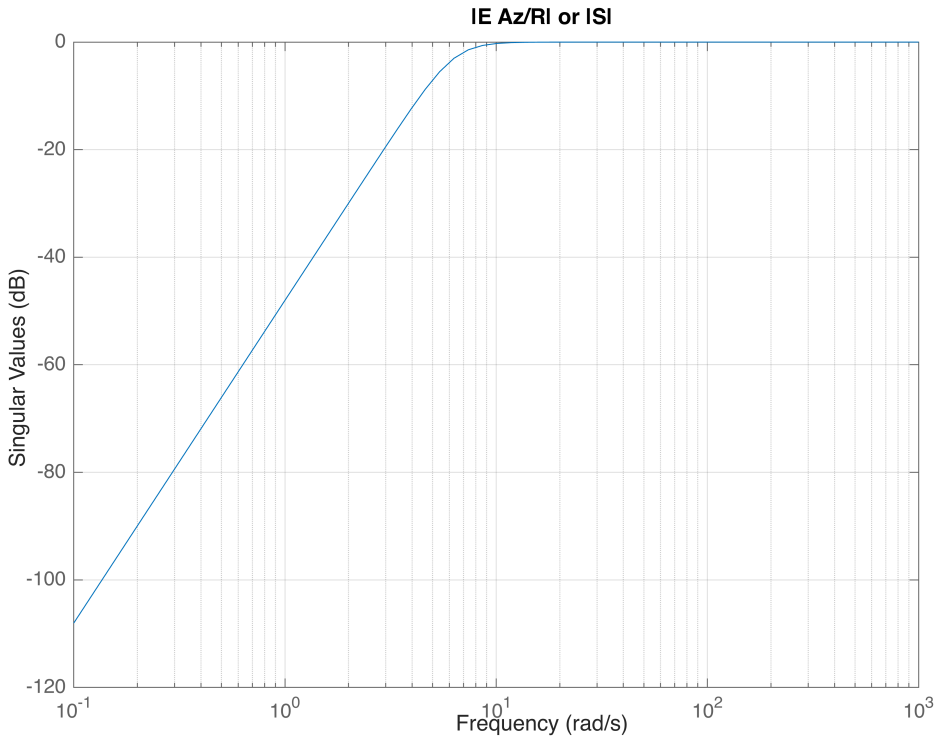
```
Classical Gain Margin:  [-12.0412, Inf] dB
Classical Phase Margin: 60.493 degrees
Singular Values Gain Margin:  [-8.6889, 269.9645+27.28753i] dB
Singular Values Phase Margin: [-60, 60] degrees
```

iv) Plot the sensitivity S and comp sensitivity T for the commanded variable.

```
T_y = sys_u(1,1)/(1+sys_u(1,1));
sigma(T_y)
grid on, title('|Az/R| or |T| at Plant Output')
xlim([.1 1000])
```




```
S_y = 1/(1+sys_u(1,1));
sigma(S_y), grid on, title('|E Az/R| or |S|')
xlim([.1 1000])
```



List out all matrices used in the final design and analysis, closed loop eigenvalues and eigenvectors.

```
[eigenvectors,eigenvalues] = eig(Acl)
```

```
eigenvectors = 3×3 complex
   0.0782 + 0.1355i   0.0782 - 0.1355i  -0.1565 + 0.0000i
  -0.9874 + 0.0000i  -0.9874 + 0.0000i   0.9874 + 0.0000i
   0.0124 - 0.0215i   0.0124 + 0.0215i   0.0248 + 0.0000i
eigenvalues = 3×3 complex
  -3.1555 + 5.4655i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -3.1555 - 5.4655i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -6.3110 + 0.0000i
```

Directional Subsystem: Yaw angle and Yaw rate

The yaw angle and its velocity dictates the dynamics of the crazyflie drone in the xy plane. The yaw angle is related to the turning of the drone.

```
Ap = [0 1;
    0 0];
Bp = [0;
    1/I_zz];
Cp = eye(2);
Dp = zeros(2,1);

s = ["[psi_dot;r_dot] = Ap*[psi;r] + Bp*M_z"];
```

```
displayFormula(["'The Turning/Yaw dynamics:'";s]);
```

The Turning/Yaw dynamics:

$$\begin{pmatrix}\dot{\psi}\\ \dot{r}\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}\psi\\ r\end{pmatrix} + \begin{pmatrix}0\\ \frac{1}{I_{zz}}\end{pmatrix} M_z$$

```
Bp = double(subs(Bp,I_zz,I_zz_));
s = ["[psi_dot;r_dot] = Ap*[psi;r] + Bp*M_z"];
displayFormula(["'The Turning/Yaw dynamics:'";s]);
```

The Turning/Yaw dynamics:

$$\begin{pmatrix}\dot{\psi}\\ \dot{r}\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}\psi\\ r\end{pmatrix} + \begin{pmatrix}0\\ \frac{3162424147998159}{68719476736}\end{pmatrix} M_z$$

```
Co = ctrb(Ap,Bp);
unco = length(Ap) - rank(Co);

% Wiggle System, Adding an error intgral state
Aw = [zeros(1,1) Cp(1,:);
     zeros(2,1) Ap];
Bw = [Dp(1,:);
      Bp];
s = ["[e_dot;psi_dot;r_dot] = Aw*[e;psi;r] + Bw*M_z"];
displayFormula(["'The Turning/Yaw dynamics with an en error
integrator:'";s]);
```

The Turning/Yaw dynamics with an en error integrator:

$$\begin{pmatrix}\dot{e}\\ \dot{\psi}\\ \dot{r}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix}e\\ \psi\\ r\end{pmatrix} + \begin{pmatrix}0\\ 0\\ \frac{3162424147998159}{68719476736}\end{pmatrix} M_z$$

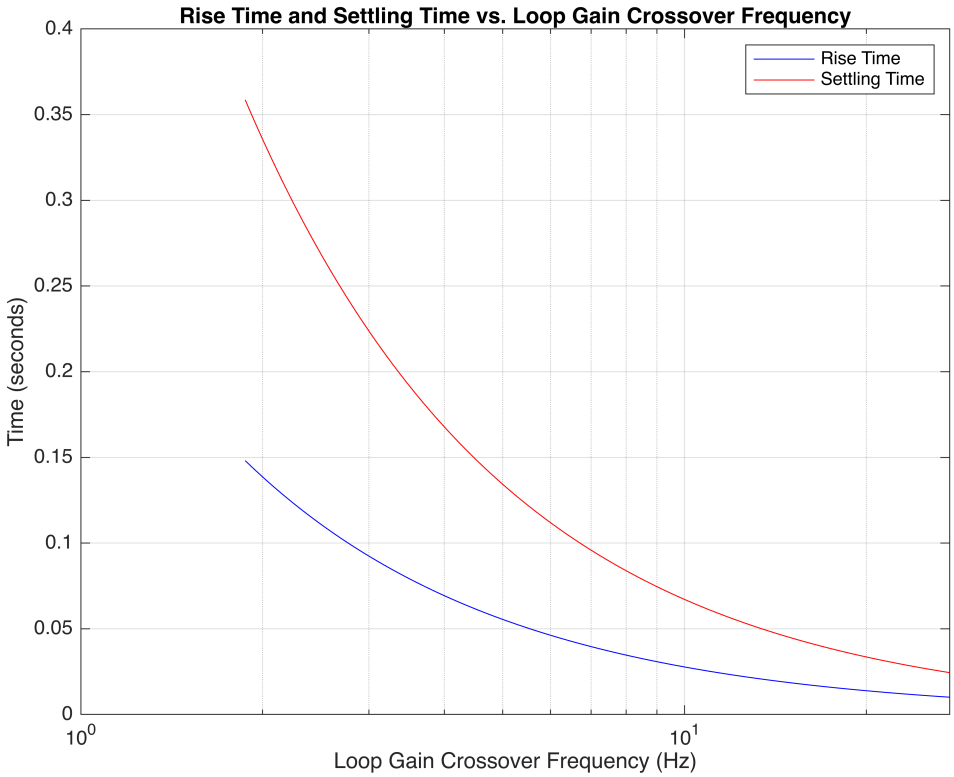
```
% Setup range of penalties for the LQR
Q=0.*Aw; %sets up a matrix Q that is the size of Aw and is all 0s
R=1;
num_step = 200;
%qq is a vector which each element scales the LQR Q matrix
qq=logspace(-2, 5,num_step); %these are the varying values of q11
step_info = zeros(size(qq,2),6);
poles = zeros(size(qq,2),3);
margins = zeros(size(qq,2),4);
mins = zeros(size(qq,2),2);
gains = zeros(size(qq,2),3);
for ii = 1:num_step
```

```
    % The only penalty is the 1,1 element on the error state
    % Desing the gains
    Q(1,1)=qq(ii);
    [Kx_lqr,~,~]=lqr(Aw,Bw,Q,R);

    % populate the controller matrices
    Ac = 0.;
    Bc1 = [1. 0. ];
    Bc2 = -1;
    Cc = -Kx_lqr(1);
    Dc1 = [-Kx_lqr(2:3)];
    Dc2 = 0;

    Zi = inv(eye(1) - Dc1*Dp);
    Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
             Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
    Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
    Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
    Dcl = Dp*Zi*Dc2;
    sys_cl = ss(Acl,Bcl,Ccl,Dcl);

    % at plant input
    Alu = [Ap     zeros(2,1);
           Bc1*Cp     Ac];
    Blu = [  Bp;
           zeros(1,1)];
    Clu = -[Dc1*Cp Cc];
    Dlu = 0;
    sys_u = ss(Alu,Blu,Clu,Dlu);

    % Loop input break Info
    lu_margins = allmargin(sys_u);
    gm_lu = 20*log10(lu_margins.GainMargin);
    pm_lu = lu_margins.PhaseMargin;
    lgcf_lu = lu_margins.PMFrequency;
    pc_lu = lu_margins.GMFrequency;

    % SV Info
    [a,b] = ss2tf(Alu,Blu,Clu,Dlu);
    c = b + a;
    sys_IL = tf(c,b);
    alpha_0 = min(sigma(sys_IL));

    sys_ILi = tf(c,a);
    beta_0 = (min(sigma(sys_ILi)));

    info = stepinfo(sys_cl,'RiseTimeThreshold', [0
0.63],'SettlingTimeThreshold', 0.05);
```

```
    step_info(ii,:) = [info(1).RiseTime info(1).SettlingTime
info(2).RiseTime, info(2).SettlingTime info(1).Overshoot
info(1).Undershoot];
    poles(ii,:) = eig(Acl);
    margins(ii,:) = [gm_lu pm_lu lgcf_lu pc_lu];
    mins(ii,:) = [alpha_0 beta_0];
    gains(ii,:) = Kx_lqr;

end

rise_time = step_info(:,1); % Rise time for the first output
settling_time = step_info(:,2); % Settling time for the first output
lgcf_lu = margins(:,4)./(2*pi); % lgcf in Hz
overshoot = step_info(:,5);
undershoot = step_info(:,6);
```

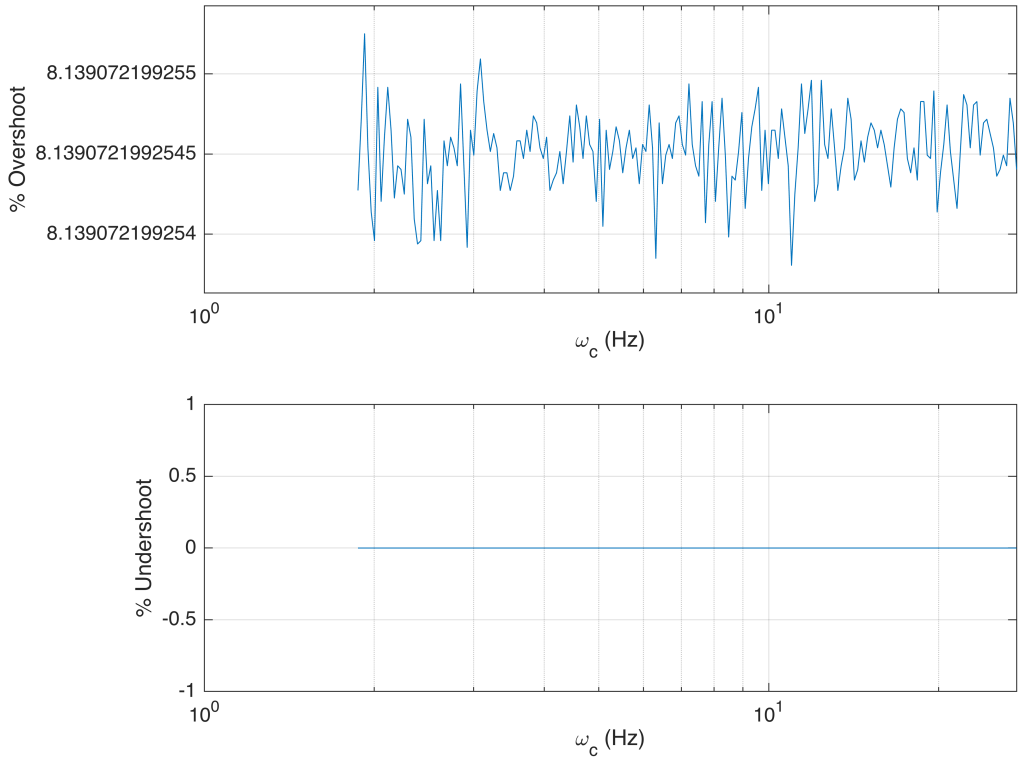
Rise time (blue) and settling time (red) versus loop gain crossover frequency.

```
% Plotting
figure;
plot(lgcf_lu, rise_time, 'b'); % Plot Rise Time in blue
hold on; % Holds the current plot to allow multiple plots in the same figure
plot(lgcf_lu, settling_time, 'r'); % Plot Settling Time in red
xlabel('Loop Gain Crossover Frequency (Hz)'); % X-axis label
ylabel('Time (seconds)'); % Y-axis label
title('Rise Time and Settling Time vs. Loop Gain Crossover Frequency'); %
Title of the plot
legend('Rise Time', 'Settling Time'); % Legend to differentiate between the
two plots
set(gca, 'XScale', 'log');
grid on; % Adds a grid to the plot for better readability
hold off;
```



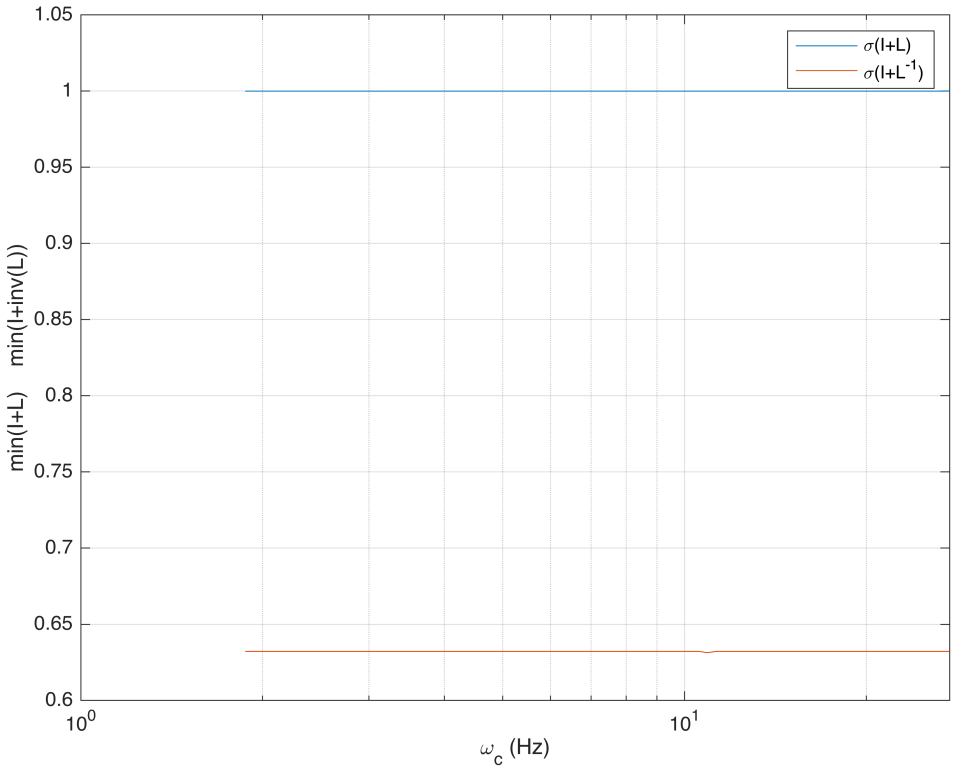
Percent overshoot, undershoot versus loop gain crossover frequency ω_c

```
figure;
subplot(2,1,1),
plot(lgcf_lu, overshoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Overshoot');
xlabel('\omega_{c} (Hz)');
subplot(2,1,2),
plot(lgcf_lu, undershoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Undershoot');
xlabel('\omega_{c} (Hz)');
```



Singular value frequency domain metrics versus loop gain crossover frequency.

```
figure;
plot(lgcf_lu,mins(:,1))
hold on;
plot(lgcf_lu,mins(:,2))
ylabel('min(I+L)    min(I+inv(L))');
legend('\sigma(I+L)','\sigma(I+L^{-1})'); % Using TeX interpreter with an
underline
set(gca, 'XScale', 'log'), grid on;
xlabel('\omega_{c} (Hz)'); % X-axis label
hold off;
```



States of the system responding to a unit acceleration step command.

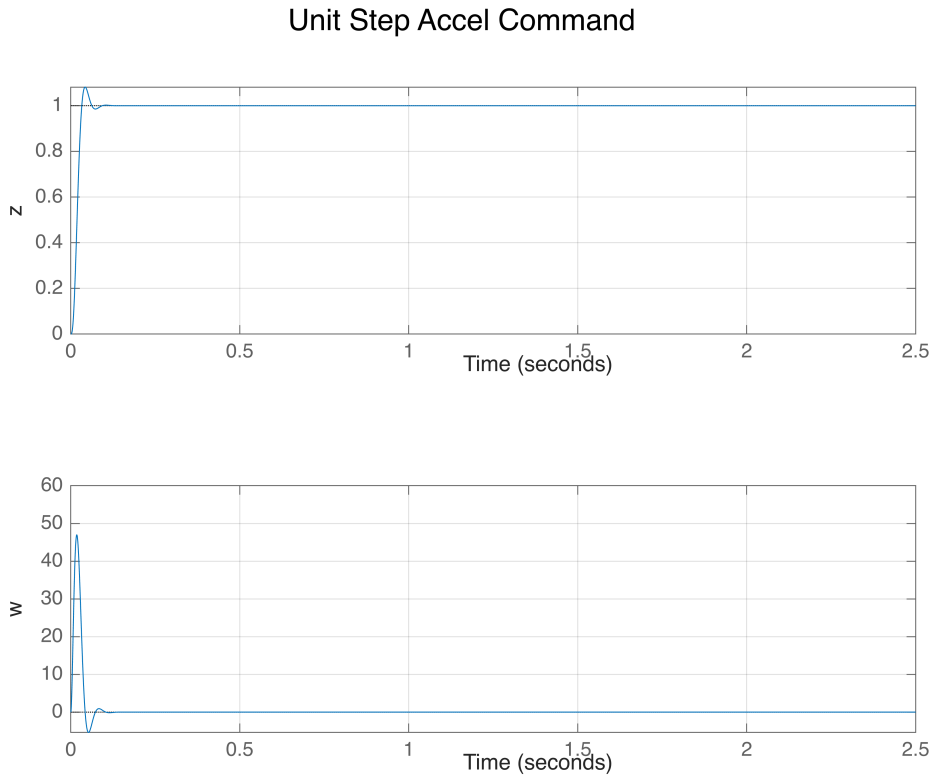
```
Kx_lqr = gains(145,:)
```

```
Kx_lqr = 1×3
   34.0929    0.5868    0.0050
```

```
% populate the controller matrices
```

```
% populate the controller matrices
Ac = 0.;
Bc1 = [1. 0. ];
Bc2 = -1;
Cc = -Kx_lqr(1);
Dc1 = [-Kx_lqr(2:3)];
Dc2 = 0;

Zi = inv(eye(1) - Dc1*Dp);
Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
            Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
Dcl = Dp*Zi*Dc2;
sys_cl = ss(Acl,Bcl,Ccl,Dcl);

figure;
```

```
sgtitle('Unit Step Accel Command')
subplot(2,1,1);
step(sys_cl(1,1));
xlim([0 2.5]), grid on;
ylabel('z'), title('')

subplot(2,1,2);
step(sys_cl(2,1));
xlim([0 2.5]), grid on;
ylabel('w'), title('')
```



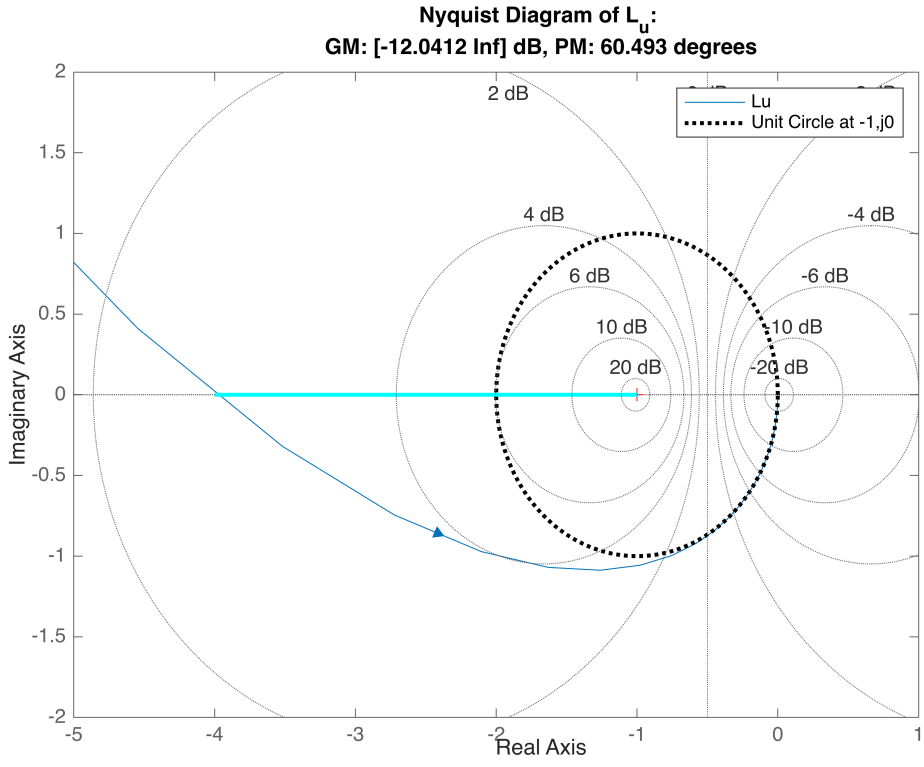
<u>Final Design Analysis</u> ,Break the loop at the plant input and evaluate the design from in the frequency domain.

```
% at plant input
Alu = [Ap     zeros(2,1);
          Bc1*Cp     Ac];
Blu = [  Bp;
          zeros(1,1)];
Clu = -[Dc1*Cp Cc];
Dlu = 0;
sys_u = ss(Alu,Blu,Clu,Dlu);

lu_margins = allmargin(sys_u);
gm_lu = 20*log10(lu_margins.GainMargin);
pm_lu = lu_margins.PhaseMargin;
```

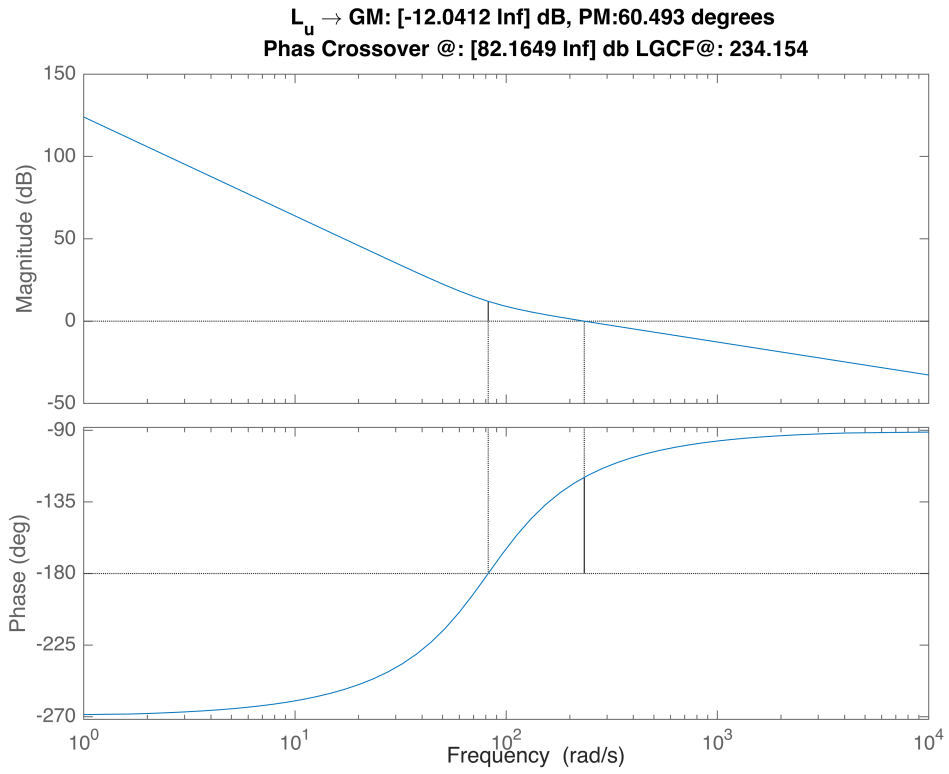
```
lgcf_lu = lu_margins.PMFrequency;
pc_lu = lu_margins.GMFrequency;
```

i) Plot a Nyquist plot and Bode plot for u L . Identify the LGCF, Phase crossover frequency.

```
figure;
n = nyquistplot(sys_u);
hold on;
plot(xx1,yy1,'k:',[-1 -1/lu_margins.GainMargin(1)],[0.
0.],'c','LineWidth',2);grid
xlim([-5,1]), ylim([-2,2])
setoptions(n,'ShowfullContour','off'), grid on
title(['Nyquist Diagram of L_u: ',...
newline 'GM: [', num2str(gm_lu(1)),' Inf] dB, ' 'PM: ' num2str(pm_lu),'
degrees'])
legend('Lu', 'Unit Circle at -1,j0')
hold off;
```

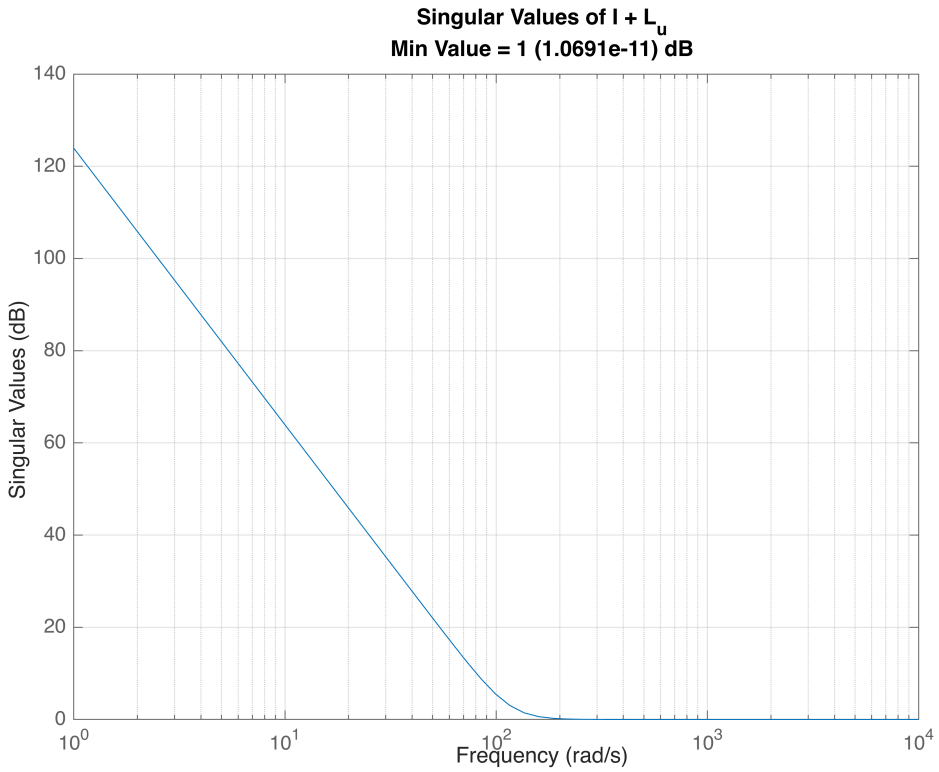



```
figure;
margin(sys_u);
title(['L_u \rightarrow GM: [', num2str(gm_lu(1)), ' Inf] dB, ',
'PM:',num2str(pm_lu(1)), ' degrees ',...
newline 'Phas Crossover @: [', num2str(pc_lu(1)),' Inf] db ','LGCF@:
',num2str(lgcf_lu(1))])
```



ii) Plot $\underline{\sigma}(I + L_u)$ and $\underline{\sigma}\left(I + L_u^{-1}\right)$.

```
[a,b] = ss2tf(Alu,Blu,Clu,Dlu);
c = b + a;
sys_IL = tf(c,b);
alpha_0 = min(sigma(sys_IL));

sigma(sys_IL);
grid on, title(['Singular Values of I + L_u',...
                newline 'Min Value = ',num2str(alpha_0), ' (',
num2str(20*log10(alpha_0)),') dB'])
```



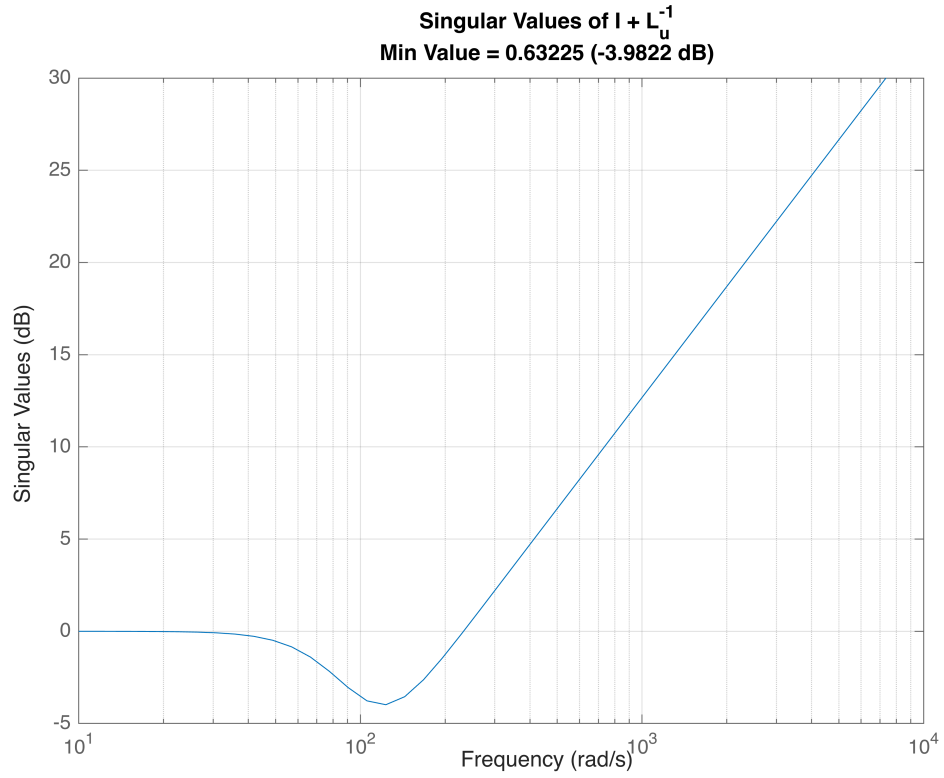

```
text = ['Minimum SV of (I +L_u): ',num2str(20*log10(alpha_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +L_u): 1.0691e-11 dB
```

```
pm = 2*asin(alpha_0/2)*180/pi;
GM_l = 20*log10(1/(1 + alpha_0));
GM_u = 20*log10(1/(1 - alpha_0));
Gm_IL = [GM_l GM_u];
pm_IL = [-pm pm];
text = ['Gain Margin from (I+L_u):  [',num2str(Gm_IL(1)),',
',num2str(Gm_IL(2)),'] dB',...
        newline 'Phase Margin from (I+L_u): [',num2str(pm_IL(1)),',
',num2str(pm_IL(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+L_u):  [-6.0206, 238.1963+27.28753i] dB
Phase Margin from (I+L_u): [-60, 60] degrees
```

```
sys_ILi = tf(c,a);
beta_0 = (min(sigma(sys_ILi)));

sigma(sys_ILi)
grid on, title(['Singular Values of I + L_u^{-1}',...
                newline 'Min Value = ',num2str(beta_0), ' (',
num2str(20*log10(beta_0)),' dB)'])
```




```
text = ['Minimum SV of (I +inv(L_u)): ',num2str(20*log10(beta_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +inv(L_u)): -3.9822 dB
```

```
GM_u = 20*log10(1 + beta_0);
GM_l = 20*log10(1 - beta_0);
pm = 2*asin(beta_0/2)*180/pi;
Gm_ILi = [GM_l GM_u];
pm_ILi = [-pm pm];
text = ['Gain Margin from (I+inv(L_u)):  [',num2str(Gm_ILi(1)),',
',num2str(Gm_ILi(2)),'] dB',...
        newline 'Phase Margin from inv((I+L_u)): [',num2str(pm_ILi(1)),',
',num2str(pm_ILi(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+inv(L_u)):  [-8.6889, 4.2557] dB
Phase Margin from inv((I+L_u)): [-36.8574, 36.8574] degrees
```

iii) Compute classical stability margins and singular value stability margins at the plant input.

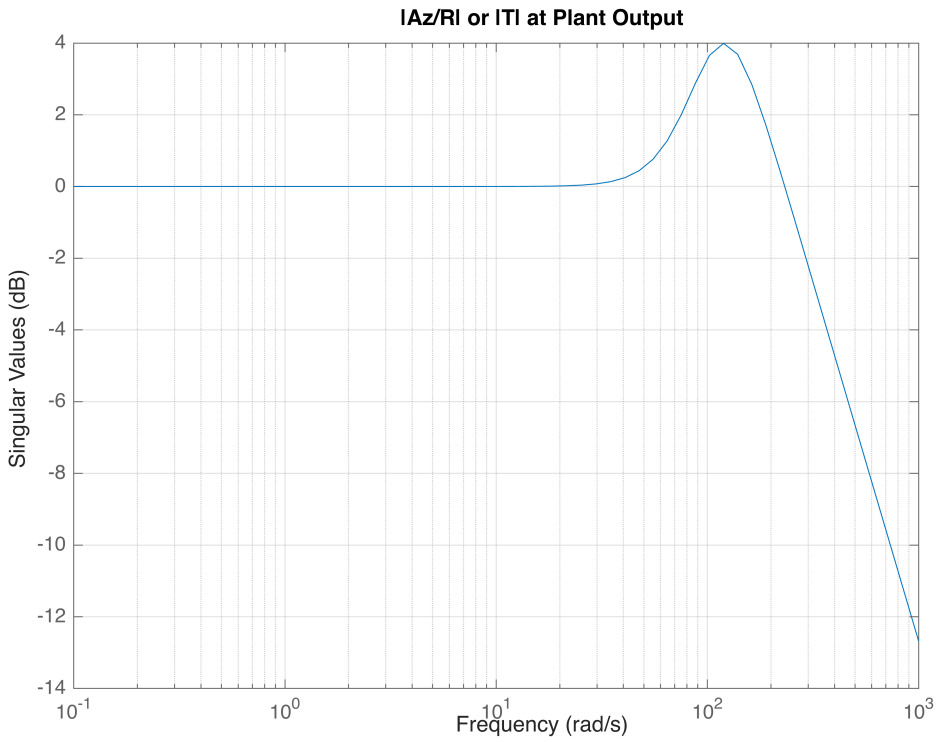
```
Gm_sv = [min(Gm_ILi(1),Gm_IL(1)),max(Gm_ILi(2),Gm_IL(2))];
Pm_sv = [min(pm_ILi(1),pm_IL(1)),max(pm_ILi(2),pm_IL(2))];

text = ['Classical Gain Margin:  [',num2str(gm_lu(1)),', Inf] dB',...
        newline 'Classical Phase Margin: ',num2str(pm_lu),' degrees',...
```

```
        newline 'Singular Values Gain Margin:  [',num2str(Gm_sv(1)),',
',num2str(Gm_sv(2)),'] dB',...
        newline 'Singular Values Phase Margin: [',num2str(Pm_sv(1)),',
',num2str(Pm_sv(2)),'] degrees'];
disp(text);
```

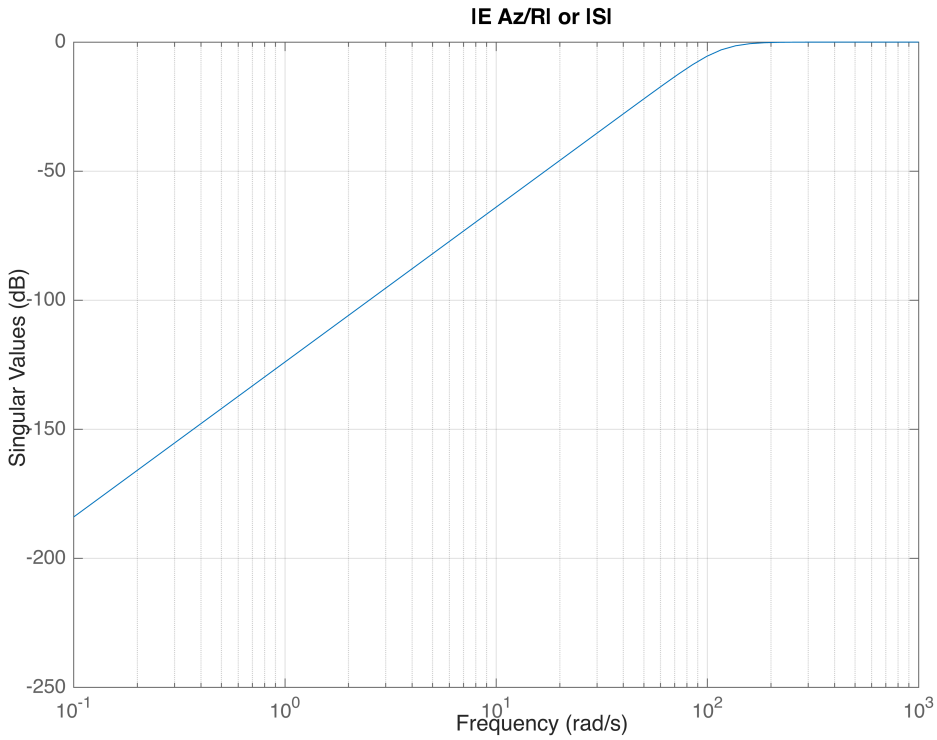
```
Classical Gain Margin:  [-12.0412, Inf] dB
Classical Phase Margin: 60.493 degrees
Singular Values Gain Margin:  [-8.6889, 238.1963+27.28753i] dB
Singular Values Phase Margin: [-60, 60] degrees
```

iv) Plot the sensitivity S and comp sensitivity T for the commanded variable.

```
T_y = sys_u(1,1)/(1+sys_u(1,1));
sigma(T_y)
grid on, title('|Az/R| or |T| at Plant Output')
xlim([.1 1000])
```




```
S_y = 1/(1+sys_u(1,1));
sigma(S_y), grid on, title('|E Az/R| or |S|')
xlim([.1 1000])
```



List out all matrices used in the final design and analysis, closed loop eigenvalues and eigenvectors.

```
[eigenvectors,eigenvalues] = eig(Acl)
```

```
eigenvectors = 3×3 complex
  -0.0043 - 0.0075i  -0.0043 + 0.0075i  -0.0086 + 0.0000i
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
  -0.0000 + 0.0001i  -0.0000 - 0.0001i   0.0001 + 0.0000i
eigenvalues = 3×3 complex
10^2 ×
  -0.5810 + 1.0063i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.5810 - 1.0063i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -1.1620 + 0.0000i
```

Lateral Subsystem

ral dynamic governs the pitch movement of the quadcopter, as well

n the inertial frame:

$$\begin{bmatrix} \Delta\dot{p} \\ \Delta\dot{\phi} \\ \Delta\dot{v} \\ \Delta\dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -g & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta\phi \\ \Delta v \\ \Delta y \end{bmatrix} + \begin{bmatrix} 1/I_{xx} \\ 0 \\ 0 \\ 0 \end{bmatrix} \Delta M_x$$

```
% A = [0 0 0 0;
%      1 0 0 0;
%      0 -g 0 0;
%      0 0 1 0];

Ap = [0 1;
    0 0];
Bp = [0;
    1/I_xx];
Cp = [1 0
       0 1];
Dp = zeros(2,1);

s = ["[phi_dot;p_dot] = Ap*[phi;p] + Bp*M_x"];
displayFormula(["'The lateral dynamics:'";s]);
```

The lateral dynamics:

$$\begin{pmatrix} \dot{\phi} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \phi \\ p \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{I_{xx}} \end{pmatrix} M_x$$

```
Bp = double(subs(Bp,I_xx,I_xx_));
s = ["[phi_dot;p_dot] = Ap*[phi;p] + Bp*M_x"];
displayFormula(["'The lateral dynamics:'";s]);
```

The lateral dynamics:

$$\begin{pmatrix} \dot{\phi} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \phi \\ p \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{2463063682293907}{34359738368} \end{pmatrix} M_x$$

```
Co = ctrb(Ap,Bp);
```

```
unco = length(Ap) - rank(Co);

% Wiggle System, Adding an error intgral state
Aw = [zeros(1,1) Cp(1,:);
     zeros(2,1) Ap];
Bw = [Dp(1,:);
      Bp];
s = ["[e_dot;phi_dot;p_dot] = Aw*[e;phi;p] + Bw*M_x"];
displayFormula(["'The lateral dynamics with an en error integrator:'";s]);
```

The lateral dynamics with an en error integrator:

$$\begin{pmatrix} \dot{e} \\ \dot{\phi} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ \phi \\ p \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{2463063682293907}{34359738368} \end{pmatrix} M_x$$

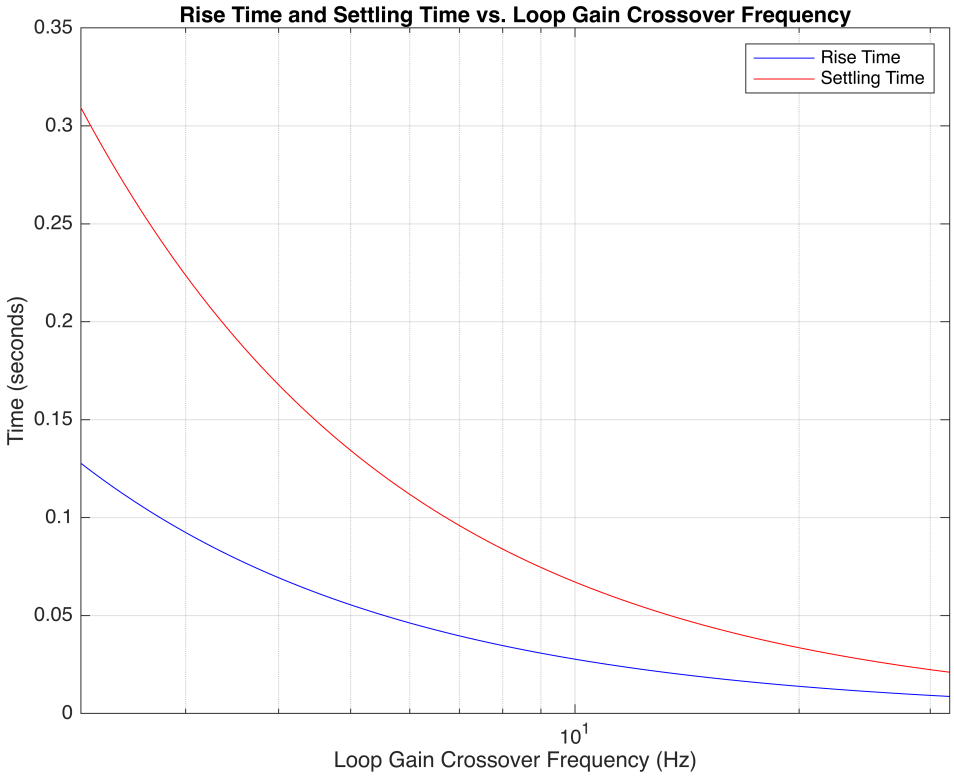
```
% Setup range of penalties for the LQR
Q=0.*Aw; %sets up a matrix Q that is the size of Aw and is all 0s
Q(2,2) = 0;
Q(3,3) = 0;
R=1;
num_step = 200;
%qq is a vector which each element scales the LQR Q matrix
qq=logspace(-2, 5,num_step); %these are the varying values of q11
step_info = zeros(size(qq,2),6);
poles = zeros(size(qq,2),3);
margins = zeros(size(qq,2),4);
mins = zeros(size(qq,2),2);
gains = zeros(size(qq,2),3);
for ii = 1:num_step

    % The only penalty is the 1,1 element on the error state
    % Desing the gains
    Q(1,1)=qq(ii);
    [Kx_lqr,~,~]=lqr(Aw,Bw,Q,R);

    % populate the controller matrices
    Ac = 0.;
    Bc1 = [1. 0. ];
    Bc2 = -1;
    Cc = -Kx_lqr(1);
    Dc1 = [-Kx_lqr(2:3)];
    Dc2 = 0;

    Zi = inv(eye(1) - Dc1*Dp);
    Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
             Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
    Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
```

```
    Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
    Dcl = Dp*Zi*Dc2;
    sys_cl = ss(Acl,Bcl,Ccl,Dcl);

    % at plant input
    Alu = [Ap     zeros(2,1);
           Bc1*Cp     Ac];
    Blu = [  Bp;
           zeros(1,1)];
    Clu = -[Dc1*Cp Cc];
    Dlu = 0;
    sys_u = ss(Alu,Blu,Clu,Dlu);

    % Loop input break Info
    lu_margins = allmargin(sys_u);
    gm_lu = 20*log10(lu_margins.GainMargin);
    pm_lu = lu_margins.PhaseMargin;
    lgcf_lu = lu_margins.PMFrequency;
    pc_lu = lu_margins.GMFrequency;

    % SV Info
    [a,b] = ss2tf(Alu,Blu,Clu,Dlu);
    c = b + a;
    sys_IL = tf(c,b);
    alpha_0 = min(sigma(sys_IL));

    sys_ILi = tf(c,a);
    beta_0 = (min(sigma(sys_ILi)));

    info = stepinfo(sys_cl,'RiseTimeThreshold', [0
0.63],'SettlingTimeThreshold', 0.05);
    step_info(ii,:) = [info(1).RiseTime info(1).SettlingTime
info(2).RiseTime, info(2).SettlingTime info(1).Overshoot
info(1).Undershoot];
    poles(ii,:) = eig(Acl);
    margins(ii,:) = [gm_lu pm_lu lgcf_lu pc_lu];
    mins(ii,:) = [alpha_0 beta_0];
    gains(ii,:) = Kx_lqr;

end

rise_time = step_info(:,1); % Rise time for the first output
settling_time = step_info(:,2); % Settling time for the first output
lgcf_lu = margins(:,4)./(2*pi); % lgcf in Hz
overshoot = step_info(:,5);
undershoot = step_info(:,6);
```

Rise time (blue) and settling time (red) versus loop gain crossover frequency.

```
% Plotting
```

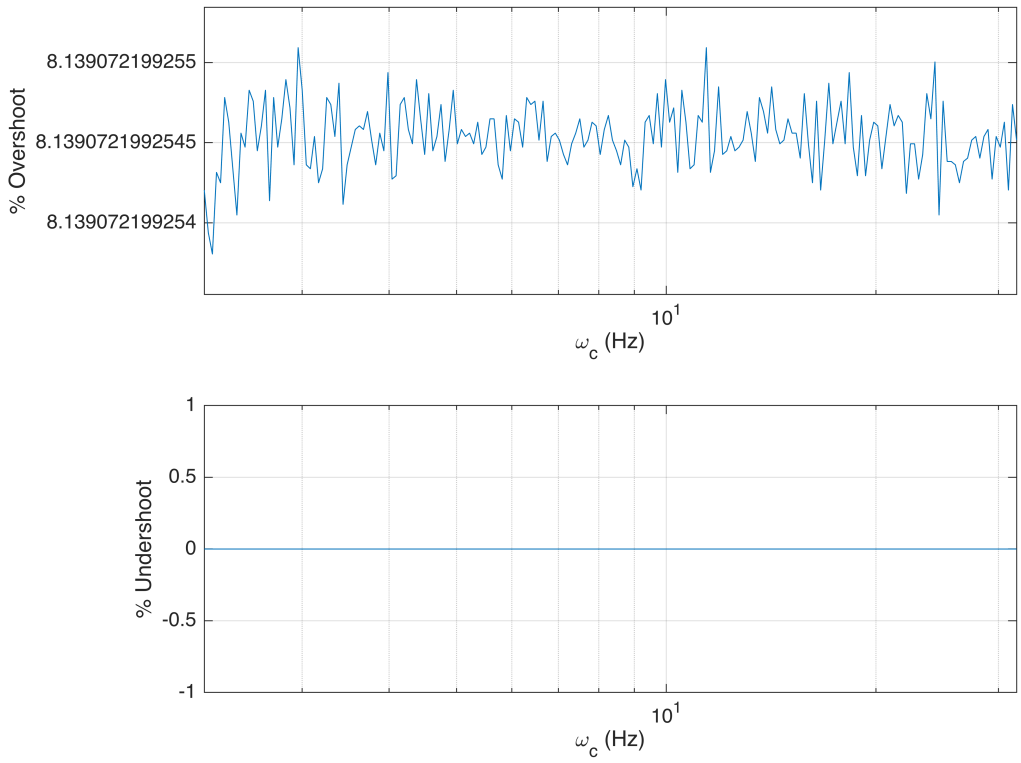
```
figure;
plot(lgcf_lu, rise_time, 'b'); % Plot Rise Time in blue
hold on; % Holds the current plot to allow multiple plots in the same figure
plot(lgcf_lu, settling_time, 'r'); % Plot Settling Time in red
xlabel('Loop Gain Crossover Frequency (Hz)'); % X-axis label
ylabel('Time (seconds)'); % Y-axis label
title('Rise Time and Settling Time vs. Loop Gain Crossover Frequency'); %
Title of the plot
legend('Rise Time', 'Settling Time'); % Legend to differentiate between the
two plots
set(gca, 'XScale', 'log');
grid on; % Adds a grid to the plot for better readability
hold off;
```



Percent overshoot, undershoot versus loop gain crossover frequency ω_c

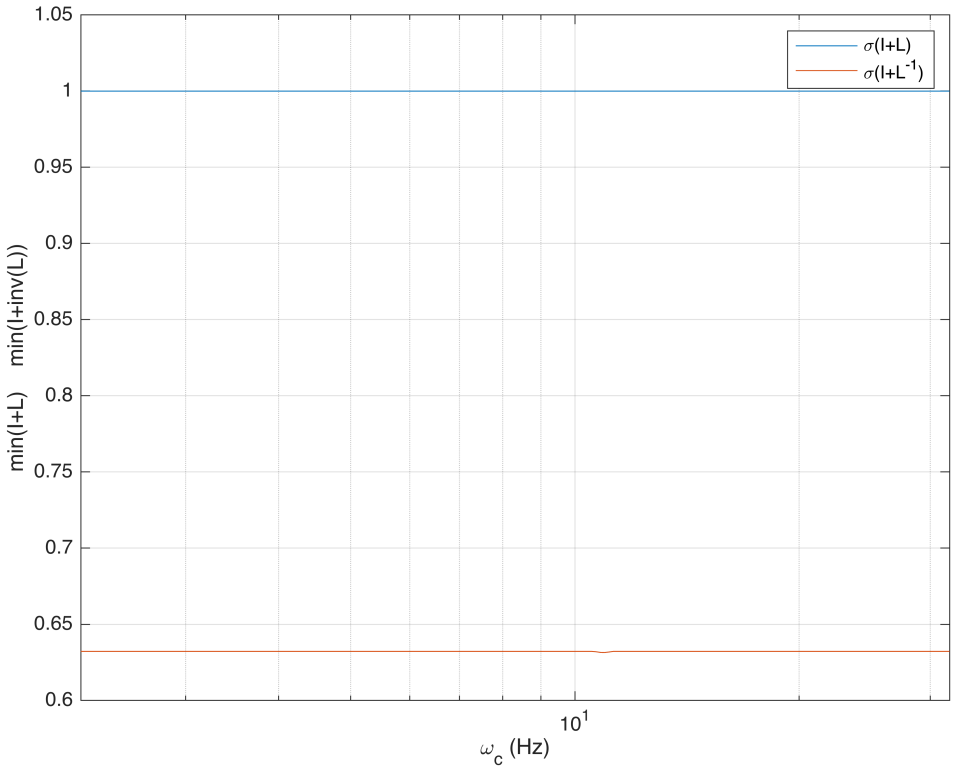
```
figure;
subplot(2,1,1),
plot(lgcf_lu, overshoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Overshoot');
xlabel('\omega_{c} (Hz)');
subplot(2,1,2),
plot(lgcf_lu, undershoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Undershoot');
```

```
xlabel('\omega_{c} (Hz)');
```



Singular value frequency domain metrics versus loop gain crossover frequency.

```
figure;
plot(lgcf_lu,mins(:,1))
hold on;
plot(lgcf_lu,mins(:,2))
ylabel('min(I+L)    min(I+inv(L))');
legend('\sigma(I+L)','\sigma(I+L^{-1})'); % Using TeX interpreter with an
underline
set(gca, 'XScale', 'log'), grid on;
xlabel('\omega_{c} (Hz)'); % X-axis label
hold off;
```



States of the system responding to a unit acceleration step command.

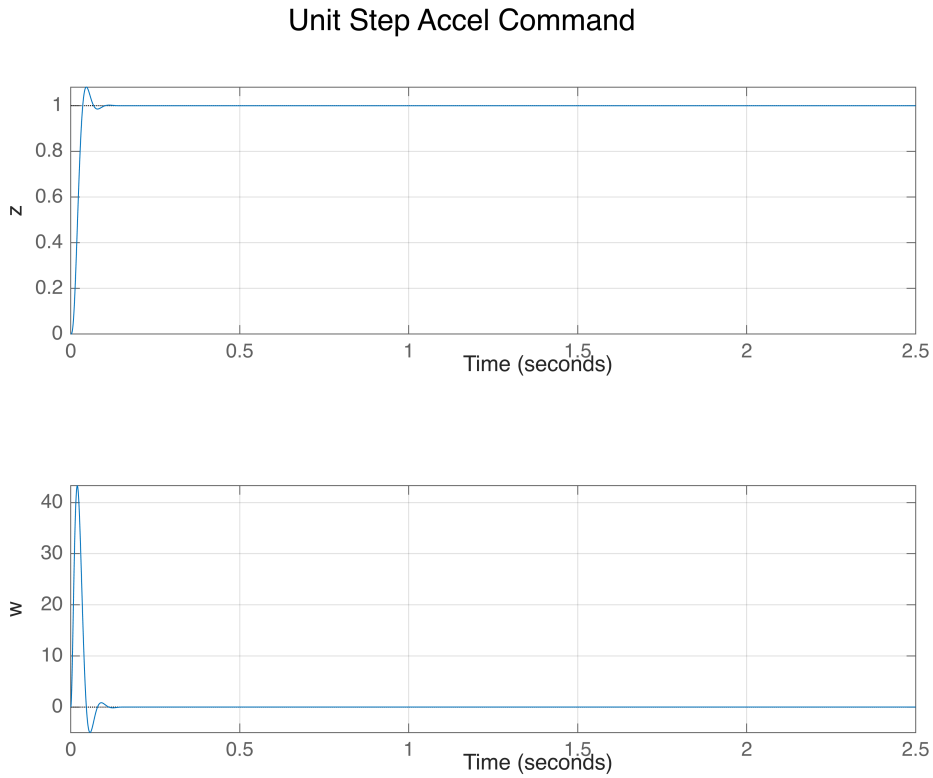
```
Kx_lqr = gains(128,:)
```

```
Kx_lqr = 1×3
   17.1265    0.3199    0.0030
```

```
% populate the controller matrices
```

```
% populate the controller matrices
Ac = 0.;
Bc1 = [1. 0. ];
Bc2 = -1;
Cc = -Kx_lqr(1);
Dc1 = [-Kx_lqr(2:3)];
Dc2 = 0;

Zi = inv(eye(1) - Dc1*Dp);
Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
             Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
Dcl = Dp*Zi*Dc2;
sys_cl = ss(Acl,Bcl,Ccl,Dcl);

figure;
```

```
sgtitle('Unit Step Accel Command')
subplot(2,1,1);
step(sys_cl(1,1));
xlim([0 2.5]), grid on;
ylabel('z'), title('')

subplot(2,1,2);
step(sys_cl(2,1));
xlim([0 2.5]), grid on;
ylabel('w'), title('')
```



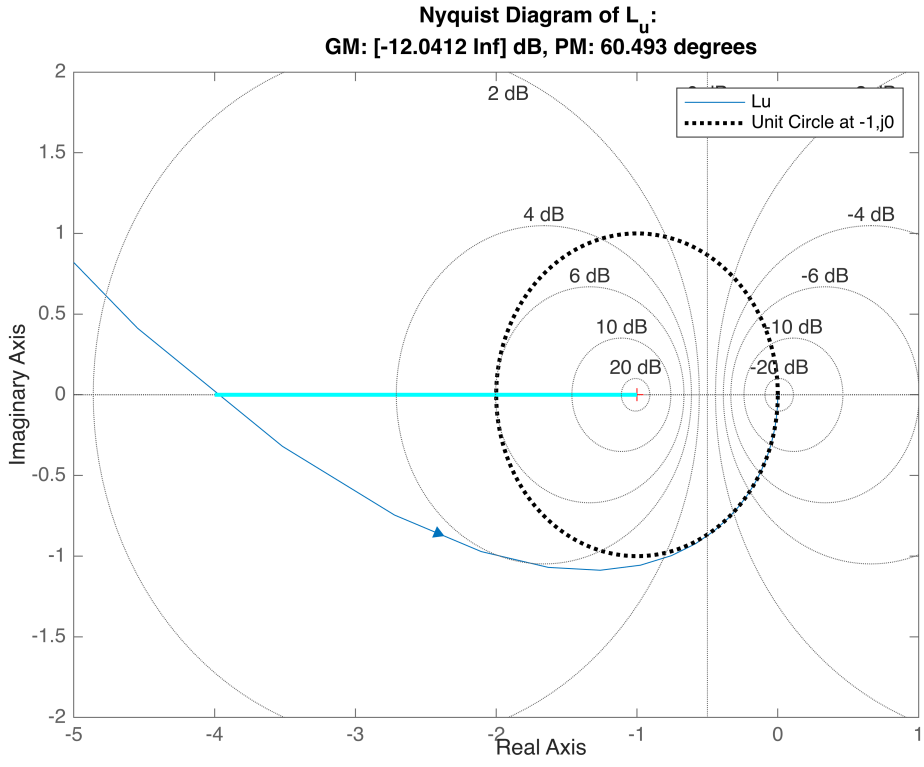
<u>Final Design Analysis</u> ,Break the loop at the plant input and evaluate the design from in the frequency domain.

```
% at plant input
Alu = [Ap     zeros(2,1);
          Bc1*Cp     Ac];
Blu = [  Bp;
          zeros(1,1)];
Clu = -[Dc1*Cp Cc];
Dlu = 0;
sys_u = ss(Alu,Blu,Clu,Dlu);

lu_margins = allmargin(sys_u);
gm_lu = 20*log10(lu_margins.GainMargin);
pm_lu = lu_margins.PhaseMargin;
```

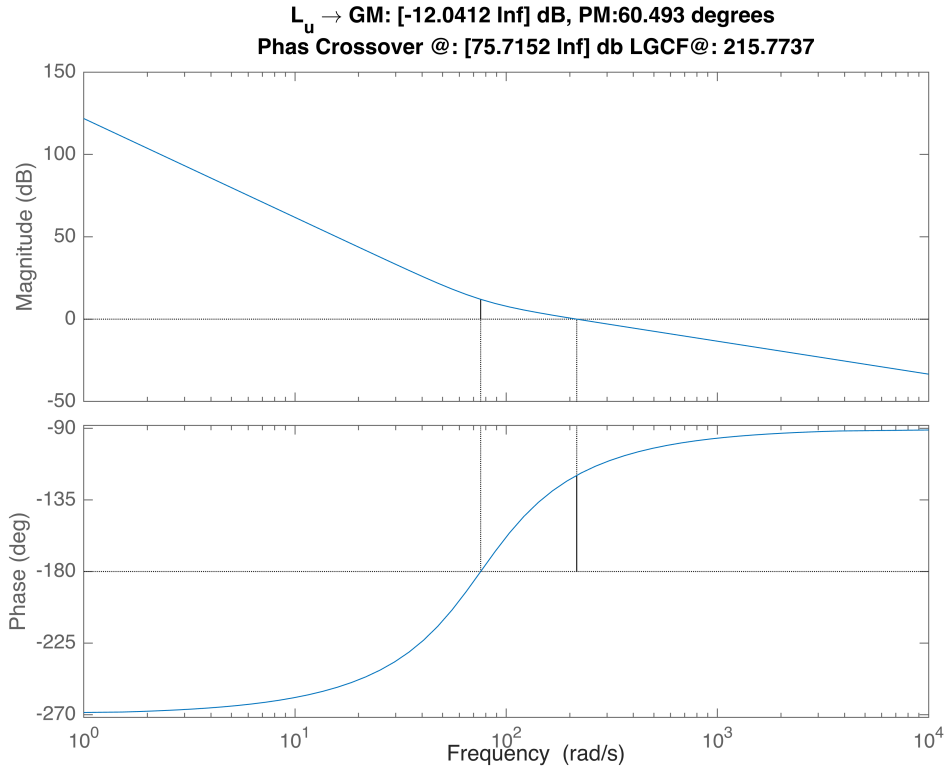
```
lgcf_lu = lu_margins.PMFrequency;
pc_lu = lu_margins.GMFrequency;
```

i) Plot a Nyquist plot and Bode plot for u L . Identify the LGCF, Phase crossover frequency.

```
figure;
n = nyquistplot(sys_u);
hold on;
plot(xx1,yy1,'k:',[-1 -1/lu_margins.GainMargin(1)],[0.
0.],'c','LineWidth',2);grid
xlim([-5,1]), ylim([-2,2])
setoptions(n,'ShowfullContour','off'), grid on
title(['Nyquist Diagram of L_u: ',...
newline 'GM: [', num2str(gm_lu(1)),' Inf] dB, ' 'PM: ' num2str(pm_lu),'
degrees'])
legend('Lu', 'Unit Circle at -1,j0')
hold off;
```

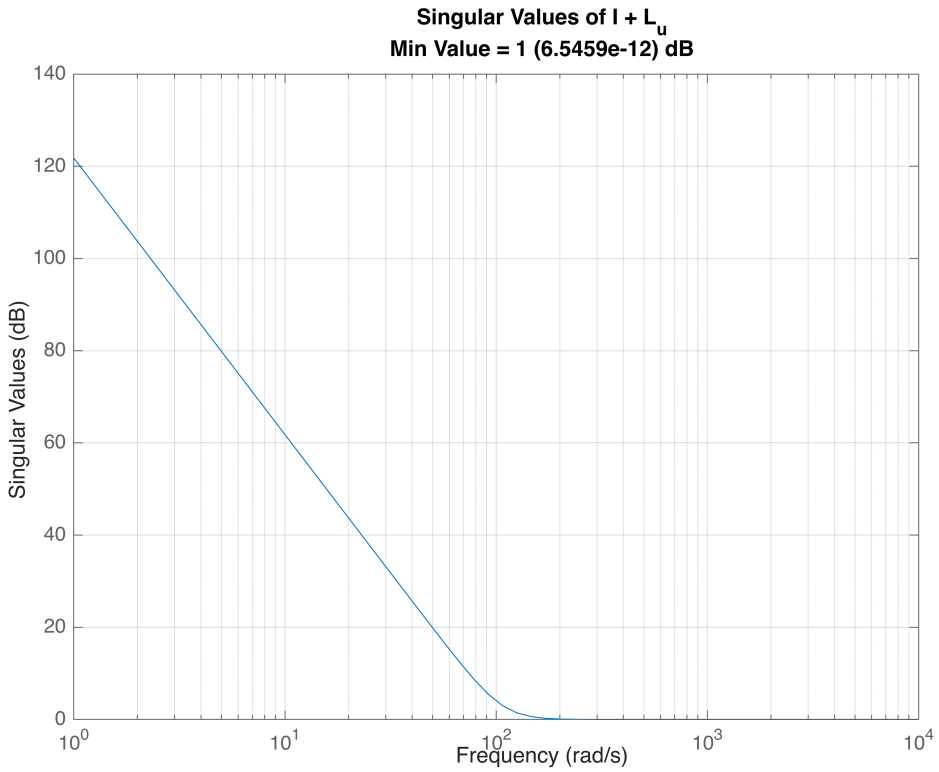



```
figure;
margin(sys_u);
title(['L_u \rightarrow GM: [', num2str(gm_lu(1)), ' Inf] dB, ',
'PM:',num2str(pm_lu(1)), ' degrees ',...
newline 'Phas Crossover @: [', num2str(pc_lu(1)),' Inf] db ','LGCF@:
',num2str(lgcf_lu(1))])
```



ii) Plot $\underline{\sigma}(I + L_u)$ and $\underline{\sigma}\left(I + L_u^{-1}\right)$.

```
[a,b] = ss2tf(Alu,Blu,Clu,Dlu);
c = b + a;
sys_IL = tf(c,b);
alpha_0 = min(sigma(sys_IL));

sigma(sys_IL);
grid on, title(['Singular Values of I + L_u',...
                newline 'Min Value = ',num2str(alpha_0), ' (',
num2str(20*log10(alpha_0)),') dB'])
```



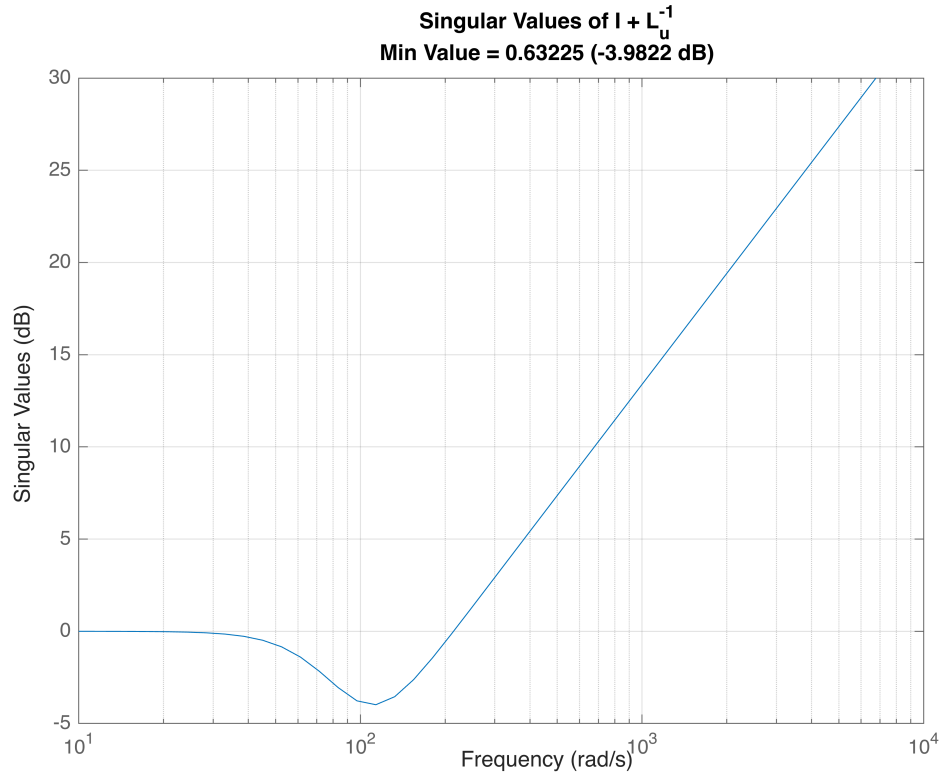

```
text = ['Minimum SV of (I +L_u): ',num2str(20*log10(alpha_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +L_u): 6.5459e-12 dB
```

```
pm = 2*asin(alpha_0/2)*180/pi;
GM_l = 20*log10(1/(1 + alpha_0));
GM_u = 20*log10(1/(1 - alpha_0));
Gm_IL = [GM_l GM_u];
pm_IL = [-pm pm];
text = ['Gain Margin from (I+L_u):  [',num2str(Gm_IL(1)),',
',num2str(Gm_IL(2)),'] dB',...
        newline 'Phase Margin from (I+L_u): [',num2str(pm_IL(1)),',
',num2str(pm_IL(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+L_u):  [-6.0206, 242.457+27.28753i] dB
Phase Margin from (I+L_u): [-60, 60] degrees
```

```
sys_ILi = tf(c,a);
beta_0 = (min(sigma(sys_ILi)));

sigma(sys_ILi)
grid on, title(['Singular Values of I + L_u^{-1}',...
                newline 'Min Value = ',num2str(beta_0), ' (',
num2str(20*log10(beta_0)),' dB)'])
```




```
text = ['Minimum SV of (I +inv(L_u)): ',num2str(20*log10(beta_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +inv(L_u)): -3.9822 dB
```

```
GM_u = 20*log10(1 + beta_0);
GM_l = 20*log10(1 - beta_0);
pm = 2*asin(beta_0/2)*180/pi;
Gm_ILi = [GM_l GM_u];
pm_ILi = [-pm pm];
text = ['Gain Margin from (I+inv(L_u)):  [',num2str(Gm_ILi(1)),',
',num2str(Gm_ILi(2)),'] dB',...
        newline 'Phase Margin from inv((I+L_u)): [',num2str(pm_ILi(1)),',
',num2str(pm_ILi(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+inv(L_u)):  [-8.6889, 4.2557] dB
Phase Margin from inv((I+L_u)): [-36.8574, 36.8574] degrees
```

iii) Compute classical stability margins and singular value stability margins at the plant input.

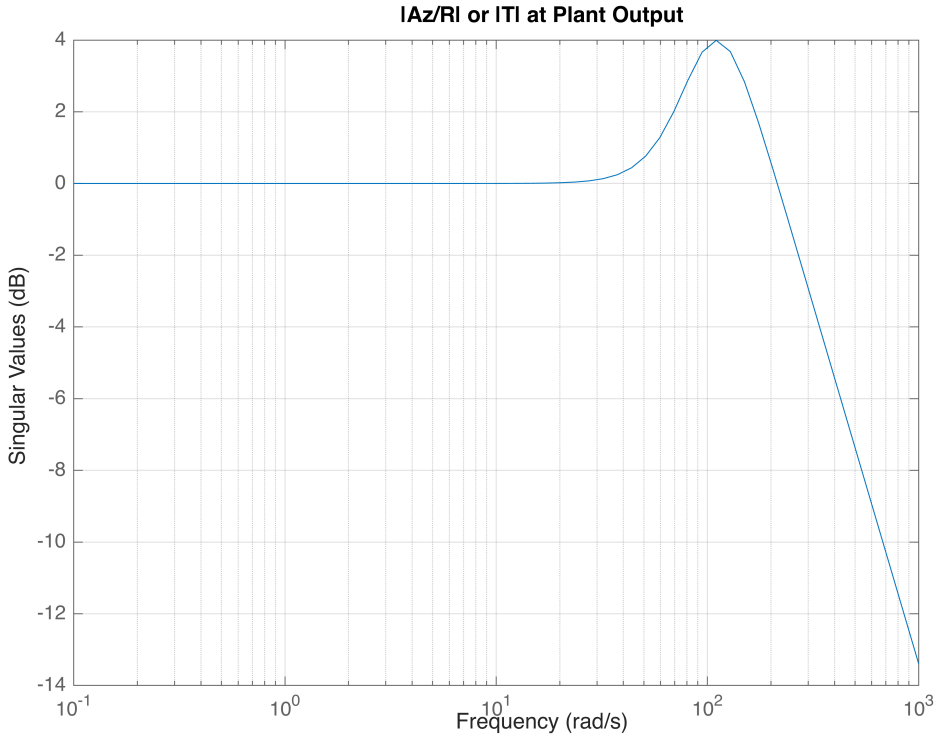
```
Gm_sv = [min(Gm_ILi(1),Gm_IL(1)),max(Gm_ILi(2),Gm_IL(2))];
Pm_sv = [min(pm_ILi(1),pm_IL(1)),max(pm_ILi(2),pm_IL(2))];

text = ['Classical Gain Margin:  [',num2str(gm_lu(1)),', Inf] dB',...
        newline 'Classical Phase Margin: ',num2str(pm_lu),' degrees',...
```

```
        newline 'Singular Values Gain Margin:  [',num2str(Gm_sv(1)),',
',num2str(Gm_sv(2)),'] dB',...
        newline 'Singular Values Phase Margin: [',num2str(Pm_sv(1)),',
',num2str(Pm_sv(2)),'] degrees'];
disp(text);
```

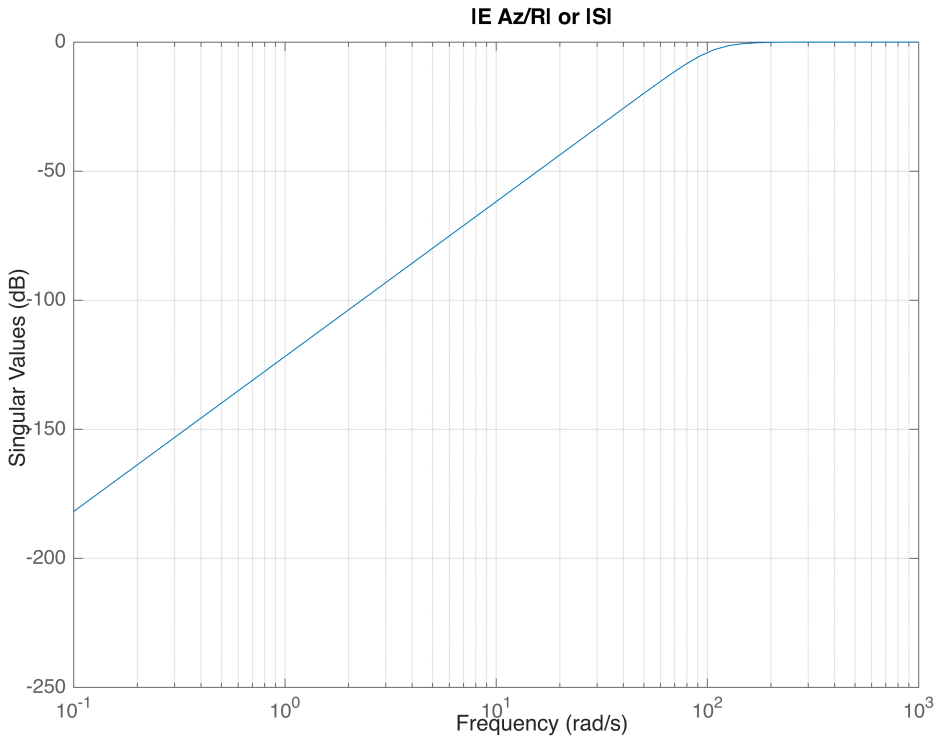
```
Classical Gain Margin:  [-12.0412, Inf] dB
Classical Phase Margin: 60.493 degrees
Singular Values Gain Margin:  [-8.6889, 242.457+27.28753i] dB
Singular Values Phase Margin: [-60, 60] degrees
```

iv) Plot the sensitivity S and comp sensitivity T for the commanded variable.

```
T_y = sys_u(1,1)/(1+sys_u(1,1));
sigma(T_y)
grid on, title('|Az/R| or |T| at Plant Output')
xlim([.1 1000])
```




```
S_y = 1/(1+sys_u(1,1));
sigma(S_y), grid on, title('|E Az/R| or |S|')
xlim([.1 1000])
```



List out all matrices used in the final design and analysis, closed loop eigenvalues and eigenvectors.

```
[eigenvectors,eigenvalues] = eig(Acl)
```

```
eigenvectors = 3×3 complex
  -0.0047 - 0.0081i  -0.0047 + 0.0081i  -0.0093 + 0.0000i
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
  -0.0000 + 0.0001i  -0.0000 - 0.0001i   0.0001 + 0.0000i
eigenvalues = 3×3 complex
10^2 ×
  -0.5354 + 0.9273i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.5354 - 0.9273i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -1.0708 + 0.0000i
```

Getting the lateral gains:

```
Ap = [0 1 0 0;
    0 0 0 0;
    -g 0 0 0;
    0 0 1 0];

Bp = [0;
    1/I_xx;
    0;
    0;];
```

```
Cp = eye(4);
Dp = zeros(4,1);

s = ["[phi_dot;p_dot;v;y] = Ap*[phi;p;v;y] + Bp*M_x"];
displayFormula(["'The lateral dynamics:'";s]);
```

The lateral dynamics:

$$\begin{pmatrix}\dot{\phi}\\ \dot{p}\\ v\\ y\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ -g & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\phi\\ p\\ v\\ y\end{pmatrix} + \begin{pmatrix}0\\ \frac{1}{I_{xx}}\\ 0\\ 0\end{pmatrix} M_x$$

```
Bp = double(subs(Bp,I_xx,I_xx_));
Ap = double(subs(Ap,g,g_));
s = ["[phi_dot;p_dot;v;y] = Ap*[phi;p;v;y] + Bp*M_x"];
displayFormula(["'The lateral dynamics:'";s]);
```

The lateral dynamics:

$$\begin{pmatrix}\dot{\phi}\\ \dot{p}\\ v\\ y\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ -\frac{981}{100} & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\phi\\ p\\ v\\ y\end{pmatrix} + \begin{pmatrix}0\\ \frac{2463063682293907}{34359738368}\\ 0\\ 0\end{pmatrix} M_x$$

```
Q = [800 0 0 0;
     0 800 0 0;
     0 0 800 0;
     0 0 0 1];
R=1;

k = lqr(Ap,Bp,Q,R);
v_kp = k(1,3)
```

```
v_kp = -28.7470
```

```
y_kp = k(1,4)
```

```
y_kp = -1.0000
```

Longitudinal Subsystem:

This subsystem controls the pitch and pitch rate of the drone about the x axis of the body frame.

```
% Ap = [0 0 0 0;
%     1 0 0 0;
%     0 g_ 0 0;
```

```
%     0 0 1 0];
% Bp = [1/I_yy_;
%     0;
%     0;
%     0;];
% Cp = eye(4);
%
% Dp = 0;

Ap = [0 1;
    0 0];
Bp = [0;
    1/I_yy];
Cp = [1 0
       0 1];
Dp = zeros(2,1);


s = ["[theta_dot;q_dot] = Ap*[theta;q] + Bp*M_x"];
displayFormula(["'The Longitudinal Pitch dynamics:'";s]);
```

The Longitudinal Pitch dynamics:

$$\begin{pmatrix}\dot{\theta}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}\theta\\ q\end{pmatrix} + \begin{pmatrix}0\\ \frac{1}{I_{yy}}\end{pmatrix} M_x$$

```
Bp = double(subs(Bp,I_yy,I_yy_));
s = ["[theta_dot;q_dot] = Ap*[theta;q] + Bp*M_x"];
displayFormula(["'The Longitudinal Pitch dynamics:'";s]);
```

The Longitudinal Pitch dynamics:

$$\begin{pmatrix}\dot{\theta}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}\theta\\ q\end{pmatrix} + \begin{pmatrix}0\\ \frac{4785478881337047}{68719476736}\end{pmatrix} M_x$$

```
Co = ctrb(Ap,Bp);
unco = length(Ap) - rank(Co);

% Wiggle System, Adding an error intgral state
Aw = [zeros(1,1) Cp(1,:);
     zeros(2,1) Ap];
Bw = [Dp(1,:);
      Bp];
s = ["[e_dot;theta_dot;q_dot] = Aw*[e;theta;q] + Bw*M_x"];
displayFormula(["'The Longitudinal Pitch dynamics with an en error
integrator:'";s]);
```

The Longitudinal Pitch dynamics with an en error integrator:

$$\begin{pmatrix} \dot{e} \\ \dot{\theta} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ \theta \\ q \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{4785478881337047}{68719476736} \end{pmatrix} M_x$$

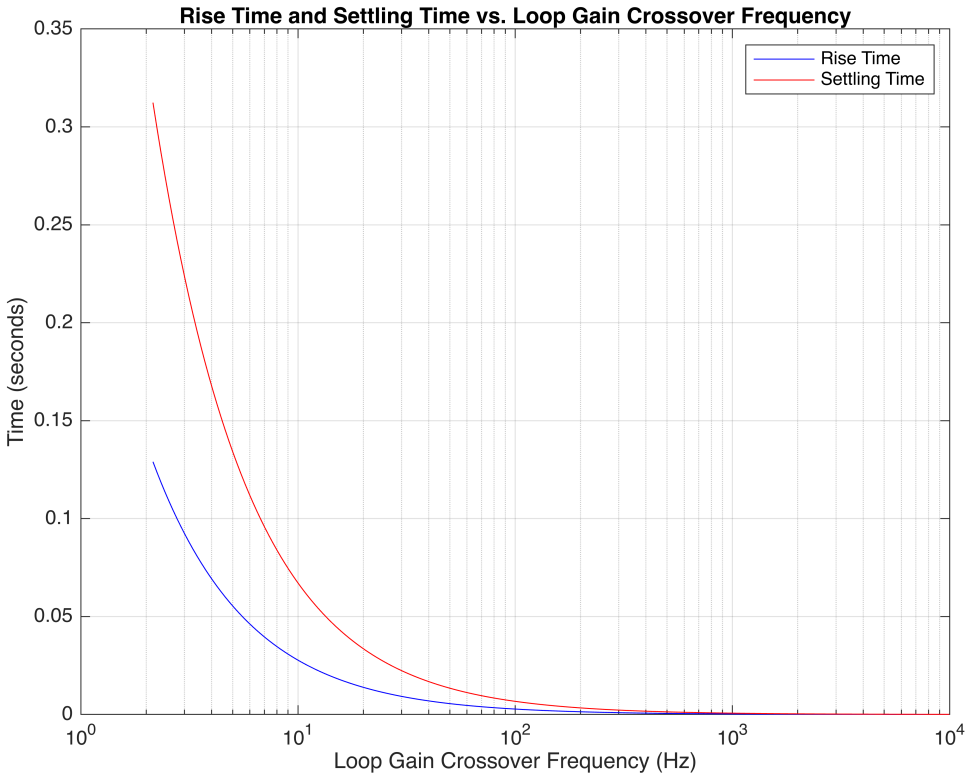
```
% Setup range of penalties for the LQR
Q=0.*Aw; %sets up a matrix Q that is the size of Aw and is all 0s
Q(2,2) = 0;
Q(3,3) = 0;
R=1;
num_step = 200;
%qq is a vector which each element scales the LQR Q matrix
qq=logspace(-2, 20,num_step); %these are the varying values of q11
step_info = zeros(size(qq,2),6);
poles = zeros(size(qq,2),3);
margins = zeros(size(qq,2),4);
mins = zeros(size(qq,2),2);
gains = zeros(size(qq,2),3);
for ii = 1:num_step

    % The only penalty is the 1,1 element on the error state
    % Desing the gains
    Q(1,1)=qq(ii);
    [Kx_lqr,~,~]=lqr(Aw,Bw,Q,R);

    % populate the controller matrices
    Ac = 0.;
    Bc1 = [1. 0. ];
    Bc2 = -1;
    Cc = -Kx_lqr(1);
    Dc1 = [-Kx_lqr(2:3)];
    Dc2 = 0;

    Zi = inv(eye(1) - Dc1*Dp);
    Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
             Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
    Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
    Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
    Dcl = Dp*Zi*Dc2;
    sys_cl = ss(Acl,Bcl,Ccl,Dcl);

    % at plant input
    Alu = [Ap     zeros(2,1);
           Bc1*Cp     Ac];
    Blu = [  Bp;
           zeros(1,1)];
    Clu = -[Dc1*Cp Cc];
```

```
    Dlu = 0;
    sys_u = ss(Alu,Blu,Clu,Dlu);

    % Loop input break Info
    lu_margins = allmargin(sys_u);
    gm_lu = 20*log10(lu_margins.GainMargin);
    pm_lu = lu_margins.PhaseMargin;
    lgcf_lu = lu_margins.PMFrequency;
    pc_lu = lu_margins.GMFrequency;

    % SV Info
    [a,b] = ss2tf(Alu,Blu,Clu,Dlu);
    c = b + a;
    sys_IL = tf(c,b);
    alpha_0 = min(sigma(sys_IL));

    sys_ILi = tf(c,a);
    beta_0 = (min(sigma(sys_ILi)));

    info = stepinfo(sys_cl,'RiseTimeThreshold', [0
0.63],'SettlingTimeThreshold', 0.05);
    step_info(ii,:) = [info(1).RiseTime info(1).SettlingTime
info(2).RiseTime, info(2).SettlingTime info(1).Overshoot
info(1).Undershoot];
    poles(ii,:) = eig(Acl);
    margins(ii,:) = [gm_lu pm_lu lgcf_lu pc_lu];
    mins(ii,:) = [alpha_0 beta_0];
    gains(ii,:) = Kx_lqr;

end

rise_time = step_info(:,1); % Rise time for the first output
settling_time = step_info(:,2); % Settling time for the first output
lgcf_lu = margins(:,4)./(2*pi); % lgcf in Hz
overshoot = step_info(:,5);
undershoot = step_info(:,6);
```

Rise time (blue) and settling time (red) versus loop gain crossover frequency.

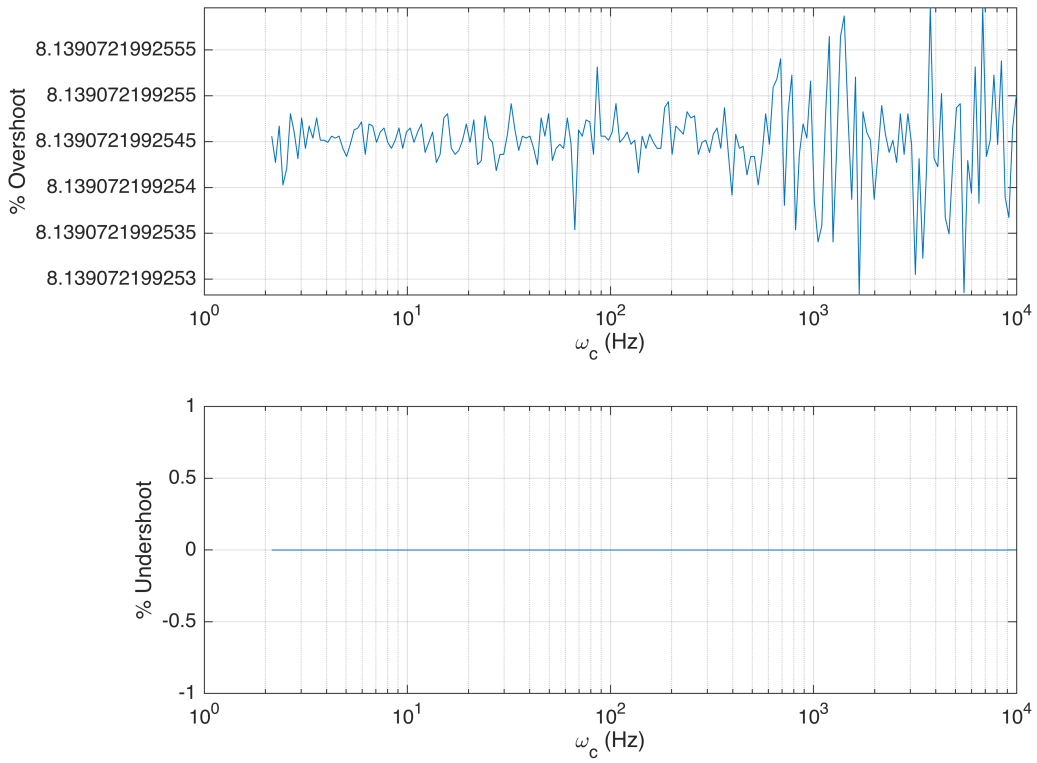
```
% Plotting
figure;
plot(lgcf_lu, rise_time, 'b'); % Plot Rise Time in blue
hold on; % Holds the current plot to allow multiple plots in the same figure
plot(lgcf_lu, settling_time, 'r'); % Plot Settling Time in red
xlabel('Loop Gain Crossover Frequency (Hz)'); % X-axis label
ylabel('Time (seconds)'); % Y-axis label
title('Rise Time and Settling Time vs. Loop Gain Crossover Frequency'); %
Title of the plot
legend('Rise Time', 'Settling Time'); % Legend to differentiate between the
two plots
```

```
set(gca, 'XScale', 'log');
grid on; % Adds a grid to the plot for better readability
hold off;
```



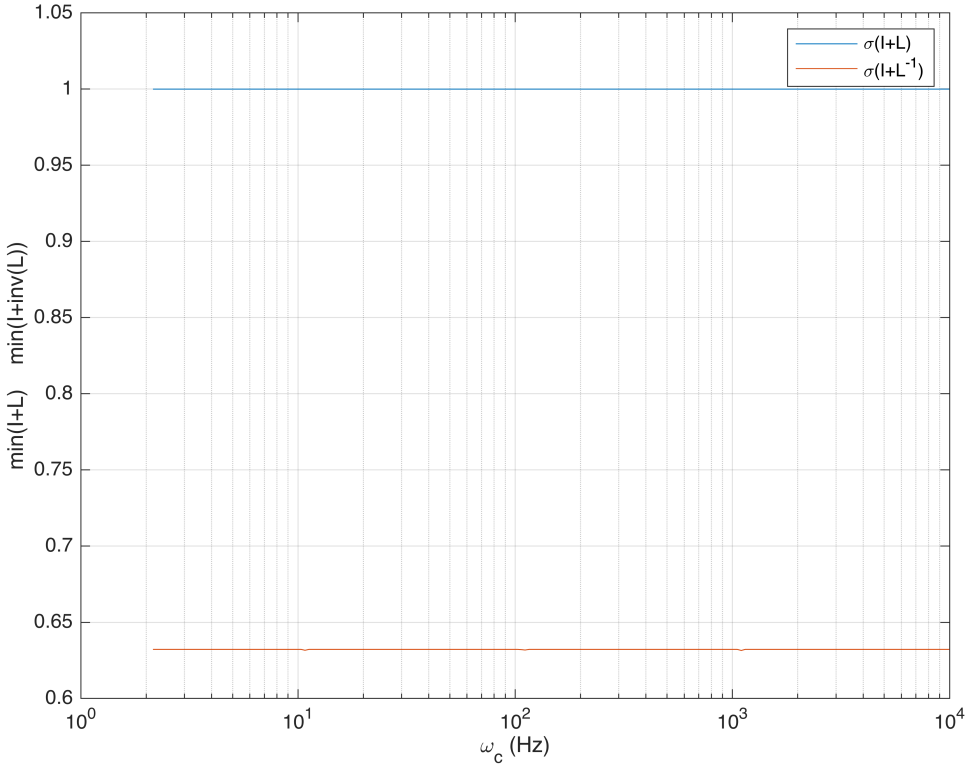
Percent overshoot, undershoot versus loop gain crossover frequency ω_c

```
figure;
subplot(2,1,1),
plot(lgcf_lu, overshoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Overshoot');
xlabel('\omega_{c} (Hz)');
subplot(2,1,2),
plot(lgcf_lu, undershoot);
set(gca, 'XScale', 'log'), grid on;
ylabel('% Undershoot');
xlabel('\omega_{c} (Hz)');
```



Singular value frequency domain metrics versus loop gain crossover frequency.

```
figure;
plot(lgcf_lu,mins(:,1))
hold on;
plot(lgcf_lu,mins(:,2))
ylabel('min(I+L)    min(I+inv(L))');
legend('\sigma(I+L)','\sigma(I+L^{-1})'); % Using TeX interpreter with an
underline
set(gca, 'XScale', 'log'), grid on;
xlabel('\omega_{c} (Hz)'); % X-axis label
hold off;
```



States of the system responding to a unit acceleration step command.

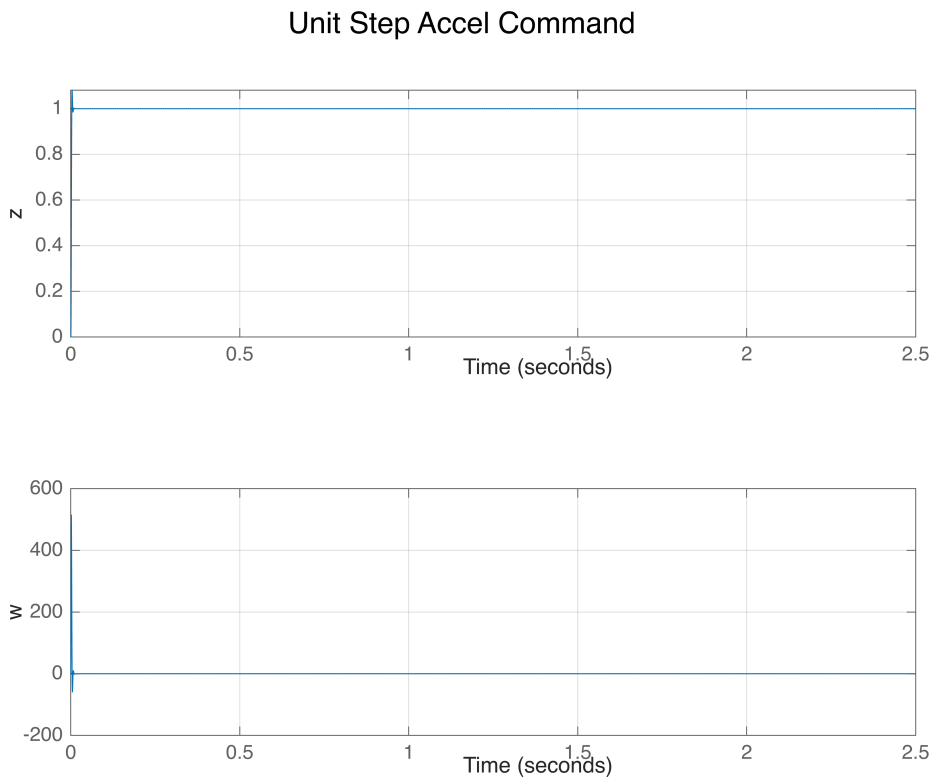
```
Kx_lqr = gains(100,:)
```

```
Kx_lqr = 1×3
10^4 ×
    2.9673    0.0047    0.0000
```

```
% populate the controller matrices
```

```
% populate the controller matrices
Ac = 0.;
Bc1 = [1. 0. ];
Bc2 = -1;
Cc = -Kx_lqr(1);
Dc1 = [-Kx_lqr(2:3)];
Dc2 = 0;

Zi = inv(eye(1) - Dc1*Dp);
Acl = [     Ap + Bp*Zi*Dc1*Cp                 Bp*Zi*Cc;
             Bc1*(eye(2) + Dp*Zi*Dc1)*Cp    Ac + Bc1*Dp*Zi*Cc];
Bcl = [    Bp*Zi*Dc2;
           Bc2 + Bc1*Dp*Zi*Dc2];
Ccl = [(eye(2) + Dp*Zi*Dc1)*Cp Dp*Zi*Cc];
Dcl = Dp*Zi*Dc2;
sys_cl = ss(Acl,Bcl,Ccl,Dcl);
```

```
figure;
sgtitle('Unit Step Accel Command')
subplot(2,1,1);
step(sys_cl(1,1));
xlim([0 2.5]), grid on;
ylabel('z'), title('')

subplot(2,1,2);
step(sys_cl(2,1));
xlim([0 2.5]), grid on;
ylabel('w'), title('')
```



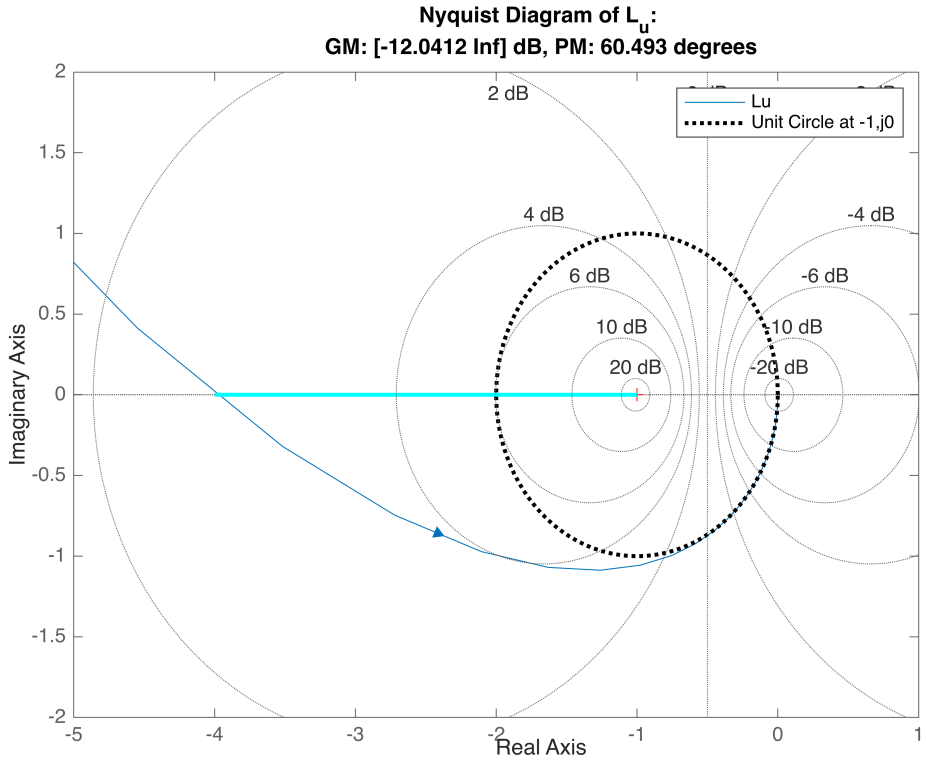
<u>Final Design Analysis</u> ,Break the loop at the plant input and evaluate the design from in the frequency domain.

```
% at plant input
Alu = [Ap     zeros(2,1);
          Bc1*Cp     Ac];
Blu = [  Bp;
          zeros(1,1)];
Clu = -[Dc1*Cp Cc];
Dlu = 0;
sys_u = ss(Alu,Blu,Clu,Dlu);

lu_margins = allmargin(sys_u);
gm_lu = 20*log10(lu_margins.GainMargin);
```

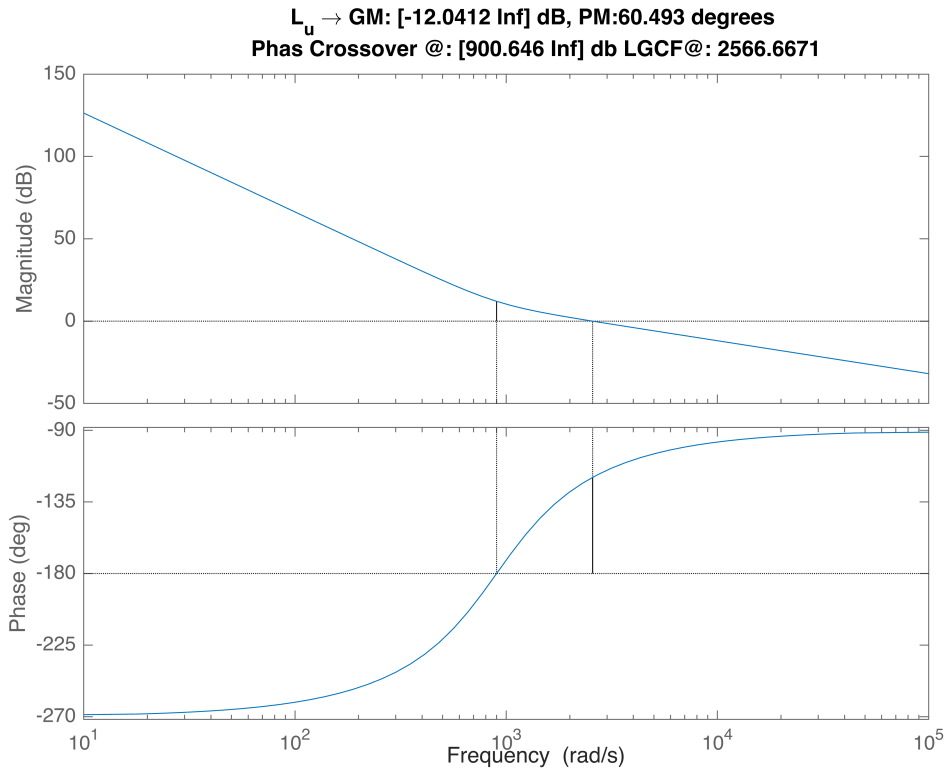
```
pm_lu = lu_margins.PhaseMargin;
lgcf_lu = lu_margins.PMFrequency;
pc_lu = lu_margins.GMFrequency;
```

i) Plot a Nyquist plot and Bode plot for u L . Identify the LGCF, Phase crossover frequency.

```
figure;
n = nyquistplot(sys_u);
hold on;
plot(xx1,yy1,'k:',[-1 -1/lu_margins.GainMargin(1)],[0.
0.],'c','LineWidth',2);grid
xlim([-5,1]), ylim([-2,2])
setoptions(n,'ShowfullContour','off'), grid on
title(['Nyquist Diagram of L_u: ',...
newline 'GM: [', num2str(gm_lu(1)),' Inf] dB, ' 'PM: ' num2str(pm_lu),'
degrees'])
legend('Lu', 'Unit Circle at -1,j0')
hold off;
```

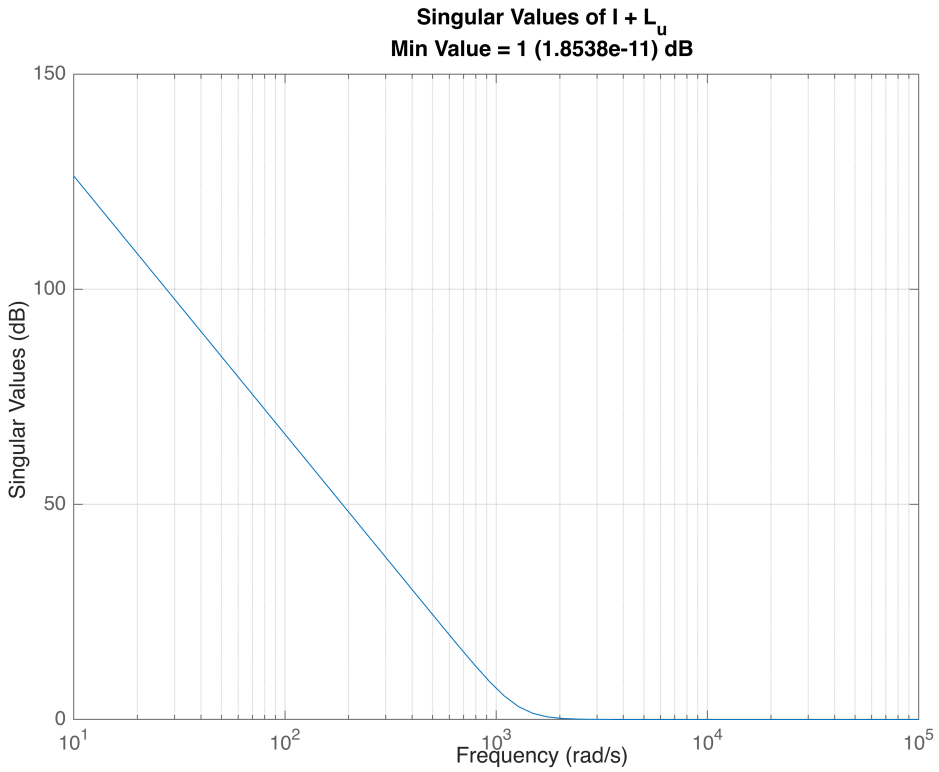



```
figure;
margin(sys_u);
title(['L_u \rightarrow GM: [', num2str(gm_lu(1)), ' Inf] dB, ',
'PM:',num2str(pm_lu(1)), ' degrees ',...
newline 'Phas Crossover @: [', num2str(pc_lu(1)),' Inf] db ','LGCF@:
',num2str(lgcf_lu(1))])
```



ii) Plot $\underline{\sigma}(I + L_u)$ and $\underline{\sigma}\left(I + L_u^{-1}\right)$.

```
[a,b] = ss2tf(Alu,Blu,Clu,Dlu);
c = b + a;
sys_IL = tf(c,b);
alpha_0 = min(sigma(sys_IL));

sigma(sys_IL);
grid on, title(['Singular Values of I + L_u',...
                newline 'Min Value = ',num2str(alpha_0), ' (',
num2str(20*log10(alpha_0)),') dB'])
```



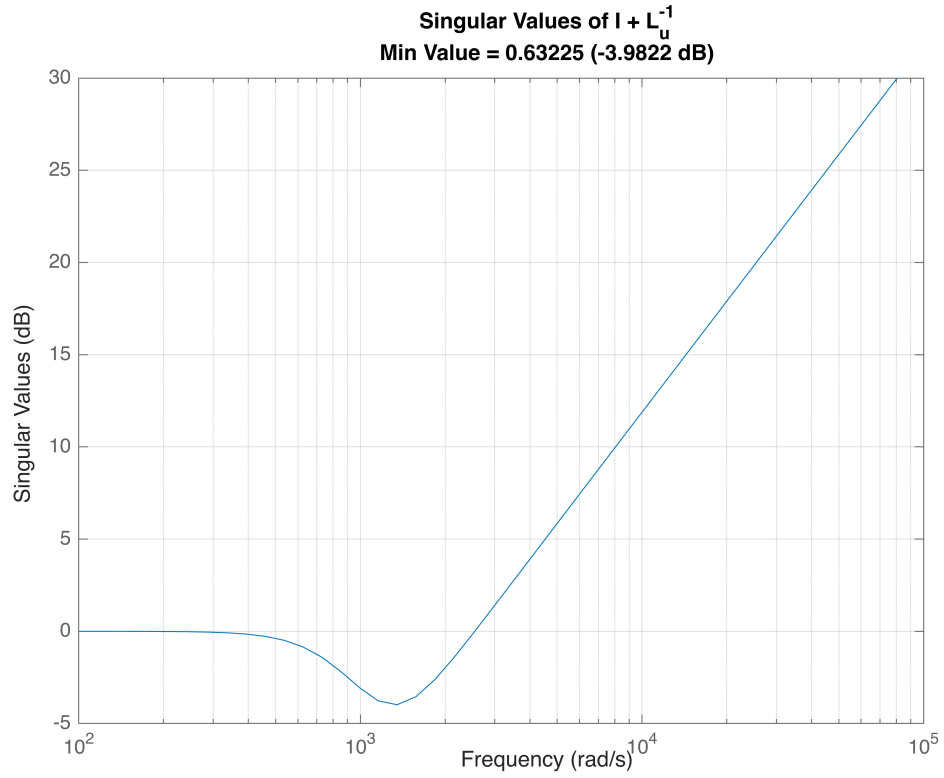

```
text = ['Minimum SV of (I +L_u): ',num2str(20*log10(alpha_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +L_u): 1.8538e-11 dB
```

```
pm = 2*asin(alpha_0/2)*180/pi;
GM_l = 20*log10(1/(1 + alpha_0));
GM_u = 20*log10(1/(1 - alpha_0));
Gm_IL = [GM_l GM_u];
pm_IL = [-pm pm];
text = ['Gain Margin from (I+L_u):  [',num2str(Gm_IL(1)),',
',num2str(Gm_IL(2)),'] dB',...
        newline 'Phase Margin from (I+L_u): [',num2str(pm_IL(1)),',
',num2str(pm_IL(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+L_u):  [-6.0206, 233.4149+27.28753i] dB
Phase Margin from (I+L_u): [-60, 60] degrees
```

```
sys_ILi = tf(c,a);
beta_0 = (min(sigma(sys_ILi)));

sigma(sys_ILi)
grid on, title(['Singular Values of I + L_u^{-1}',...
                newline 'Min Value = ',num2str(beta_0), ' (',
num2str(20*log10(beta_0)),' dB)'])
```




```
text = ['Minimum SV of (I +inv(L_u)): ',num2str(20*log10(beta_0)), ' dB'];
disp(text)
```

```
Minimum SV of (I +inv(L_u)): -3.9822 dB
```

```
GM_u = 20*log10(1 + beta_0);
GM_l = 20*log10(1 - beta_0);
pm = 2*asin(beta_0/2)*180/pi;
Gm_ILi = [GM_l GM_u];
pm_ILi = [-pm pm];
text = ['Gain Margin from (I+inv(L_u)):  [',num2str(Gm_ILi(1)),',
',num2str(Gm_ILi(2)),'] dB',...
        newline 'Phase Margin from inv((I+L_u)): [',num2str(pm_ILi(1)),',
',num2str(pm_ILi(2)),'] degrees'];
disp(text);
```

```
Gain Margin from (I+inv(L_u)):  [-8.6889, 4.2557] dB
Phase Margin from inv((I+L_u)): [-36.8574, 36.8574] degrees
```

iii) Compute classical stability margins and singular value stability margins at the plant input.

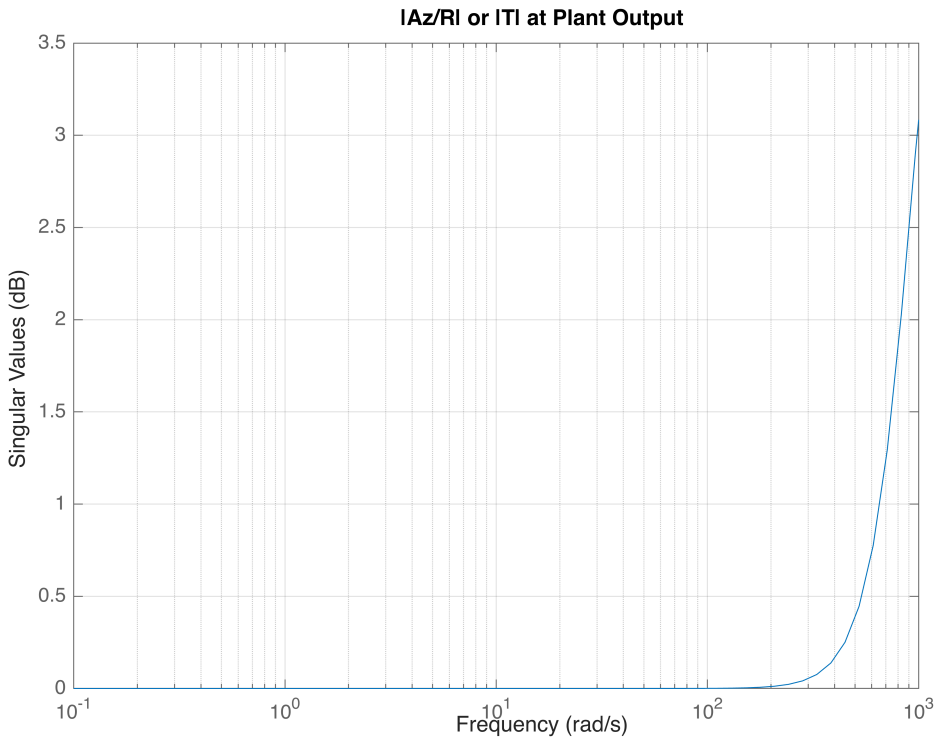
```
Gm_sv = [min(Gm_ILi(1),Gm_IL(1)),max(Gm_ILi(2),Gm_IL(2))];
Pm_sv = [min(pm_ILi(1),pm_IL(1)),max(pm_ILi(2),pm_IL(2))];

text = ['Classical Gain Margin:  [',num2str(gm_lu(1)),', Inf] dB',...
        newline 'Classical Phase Margin: ',num2str(pm_lu),' degrees',...
```

```
        newline 'Singular Values Gain Margin:  [',num2str(Gm_sv(1)),',
',num2str(Gm_sv(2)),'] dB',...
        newline 'Singular Values Phase Margin: [',num2str(Pm_sv(1)),',
',num2str(Pm_sv(2)),'] degrees'];
disp(text);
```

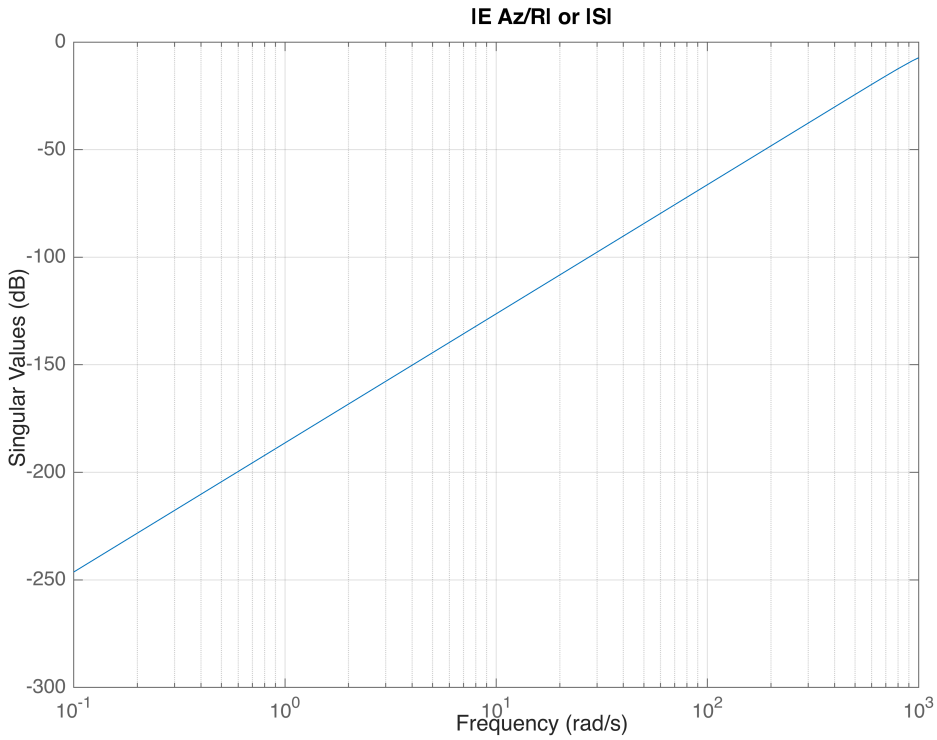
```
Classical Gain Margin:  [-12.0412, Inf] dB
Classical Phase Margin: 60.493 degrees
Singular Values Gain Margin:  [-8.6889, 233.4149+27.28753i] dB
Singular Values Phase Margin: [-60, 60] degrees
```

iv) Plot the sensitivity S and comp sensitivity T for the commanded variable.

```
T_y = sys_u(1,1)/(1+sys_u(1,1));
sigma(T_y)
grid on, title('|Az/R| or |T| at Plant Output')
xlim([.1 1000])
```




```
S_y = 1/(1+sys_u(1,1));
sigma(S_y), grid on, title('|E Az/R| or |S|')
xlim([.1 1000])
```



List out all matrices used in the final design and analysis, closed loop eigenvalues and eigenvectors.

```
[eigenvectors,eigenvalues] = eig(Acl)
```

```
eigenvectors = 3×3 complex
  -0.0004 - 0.0007i  -0.0004 + 0.0007i  -0.0008 + 0.0000i
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
  -0.0000 + 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i
eigenvalues = 3×3 complex
10^3 ×
  -0.6369 + 1.1031i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.6369 - 1.1031i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -1.2737 + 0.0000i
```

Getting the lateral gains by choosing the Q matrix to have the same gain factor as the factor we used to transform the lqr gains we found to work with PWM pid controller we must use on the crazyflie drone.

```
Ap = [0 1 0 0;
    0 0 0 0;
    g_ 0 0 0;
    0 0 1 0];
Bp = [0;
    1/I_yy;
    0;
    0;];
```

```
Cp = eye(4);

Dp = zeros(4,1);

s = ["[theta_dot;q_dot;u_dot;x_dot] = Ap*[theta;q;u;x] + Bp*M_y"];
displayFormula(["'The Longitudinal Pitch dynamics:'";s]);
```

The Longitudinal Pitch dynamics:

$$\begin{pmatrix}\dot{\theta}\\ \dot{q}\\ \dot{u}\\ \dot{x}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ \frac{981}{100} & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\theta\\ q\\ u\\ x\end{pmatrix} + \begin{pmatrix}0\\ \frac{1}{I_{yy}}\\ 0\\ 0\end{pmatrix} M_y$$

```
Bp = double(subs(Bp,I_yy,I_yy_));
s = ["[theta_dot;q_dot;u_dot;x_dot] = Ap*[theta;q;u;x] + Bp*M_y"];
displayFormula(["'The Longitudinal Pitch dynamics:'";s]);
```

The Longitudinal Pitch dynamics:

$$\begin{pmatrix}\dot{\theta}\\ \dot{q}\\ \dot{u}\\ \dot{x}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ \frac{981}{100} & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\theta\\ q\\ u\\ x\end{pmatrix} + \begin{pmatrix}0\\ \frac{4785478881337047}{68719476736}\\ 0\\ 0\end{pmatrix} M_y$$

```
Q = [800 0 0 0;
     0 800 0 0;
     0 0 800 0;
     0 0 0 1];
R=1;

k = lqr(Ap,Bp,Q,R);
u_kp = k(1,3)
```

```
u_kp = 28.7470
```

```
x_kp = k(1,4)
```

```
x_kp = 1.0000
```