AAE 364: Controls System Analysis

HW9: Controller Design & Root Locus

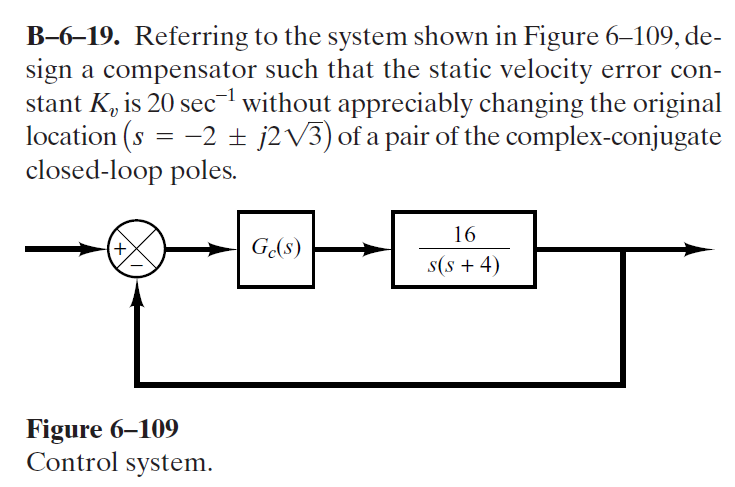
Dr. Sun

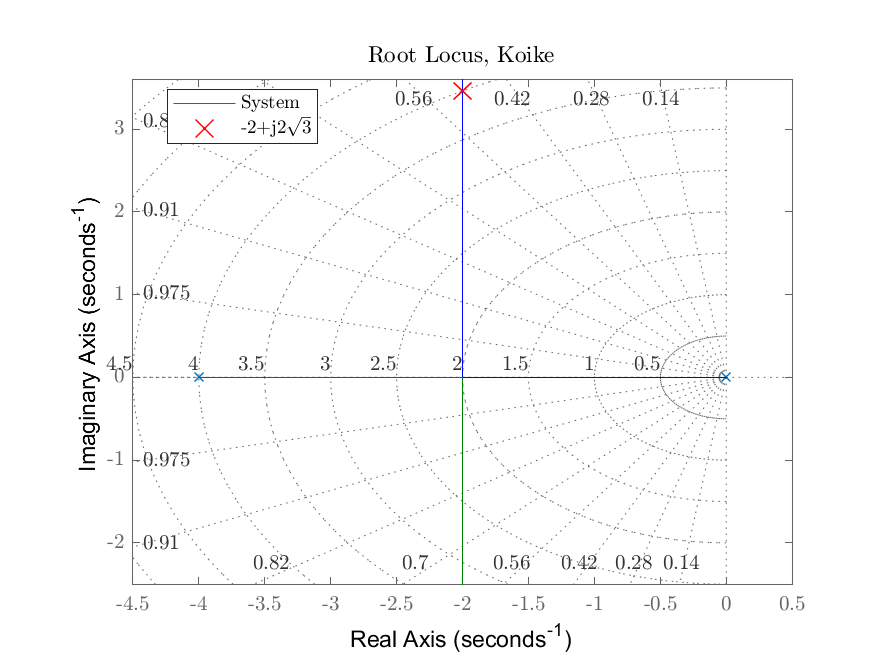
School of Aeronautical and Astronautical

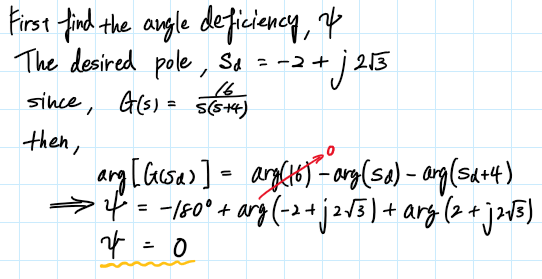
Purdue University

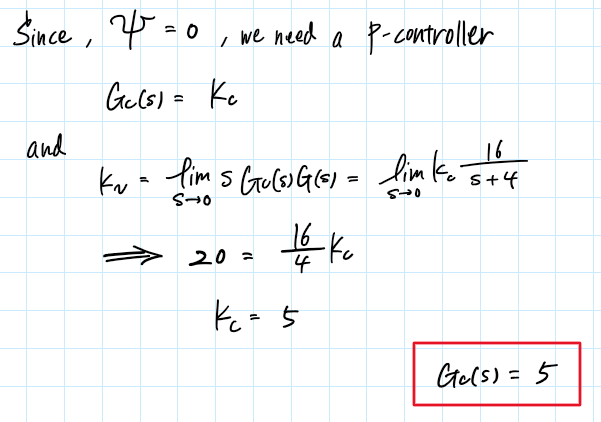
Tomoki Koike

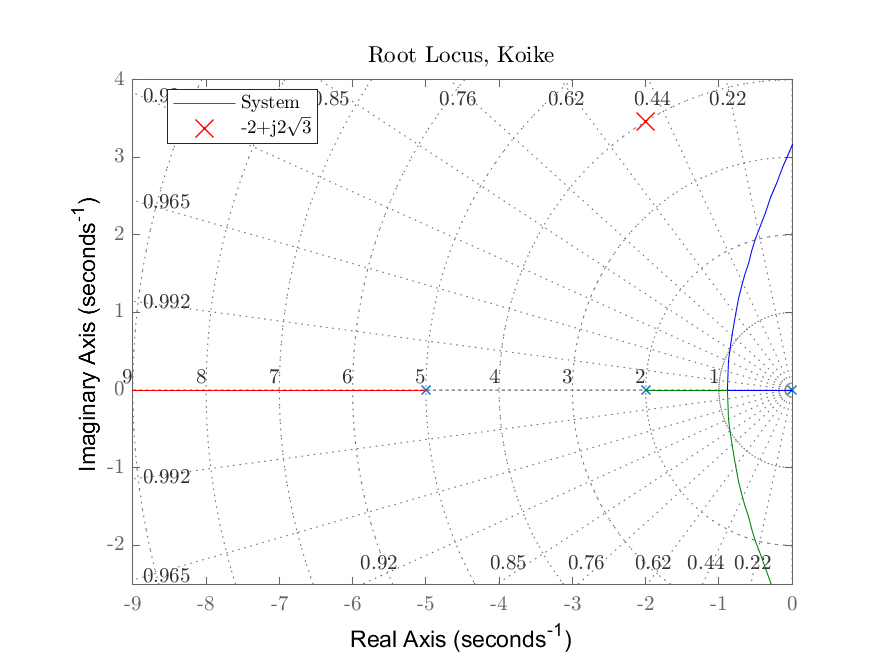
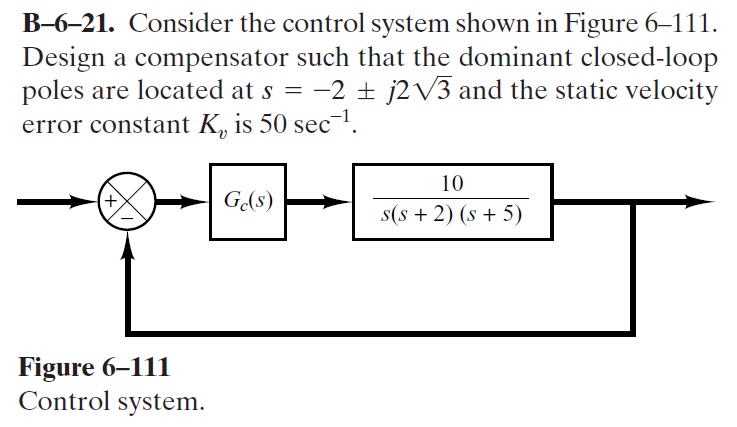
Friday April 3, 2020

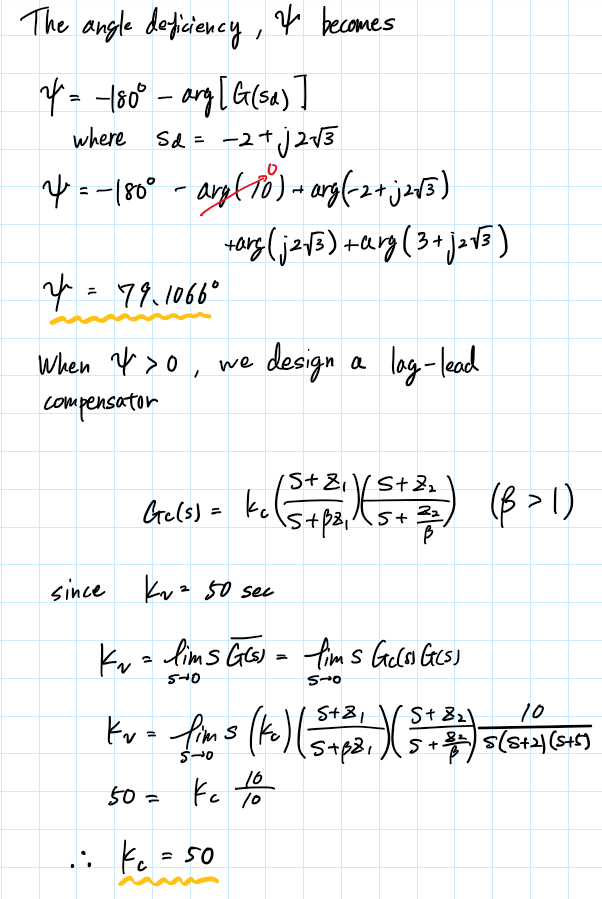


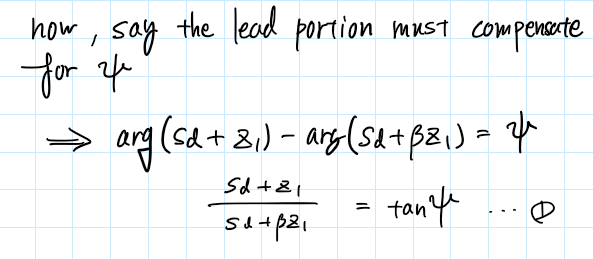


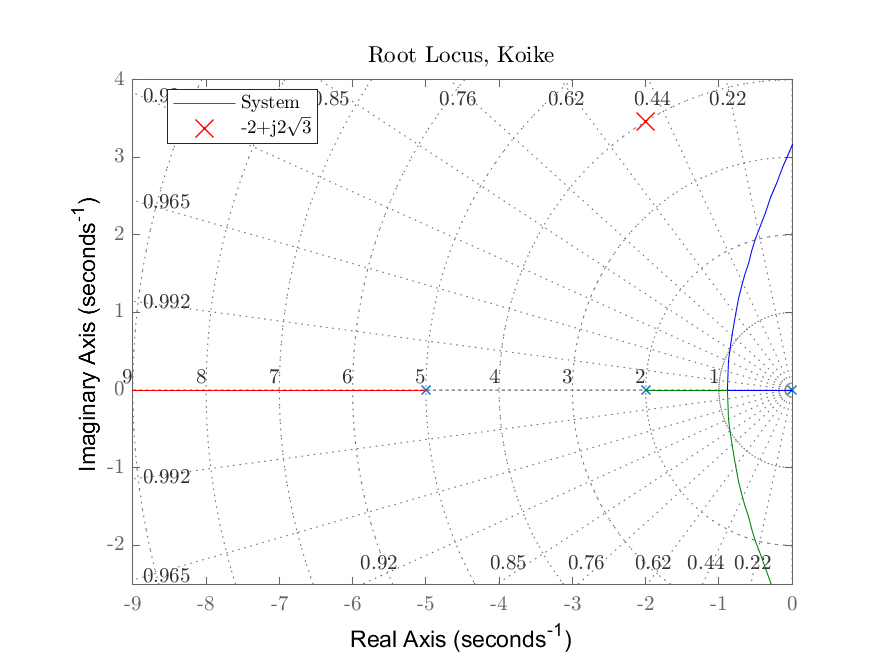


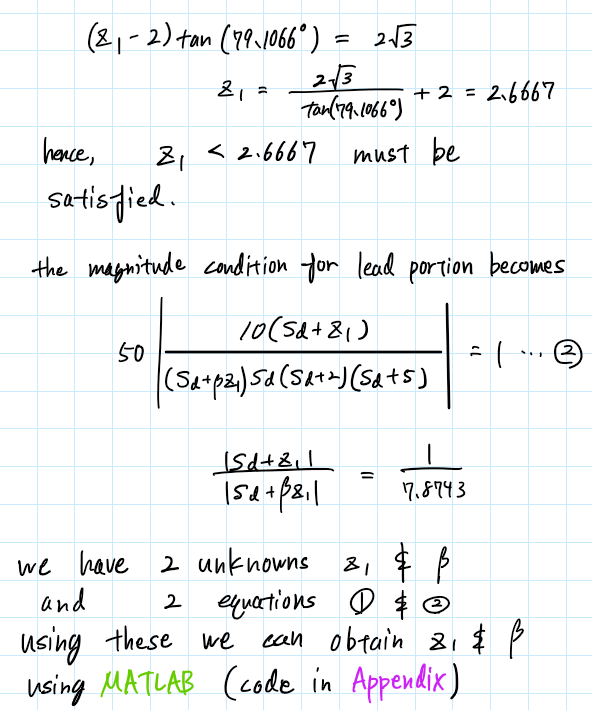


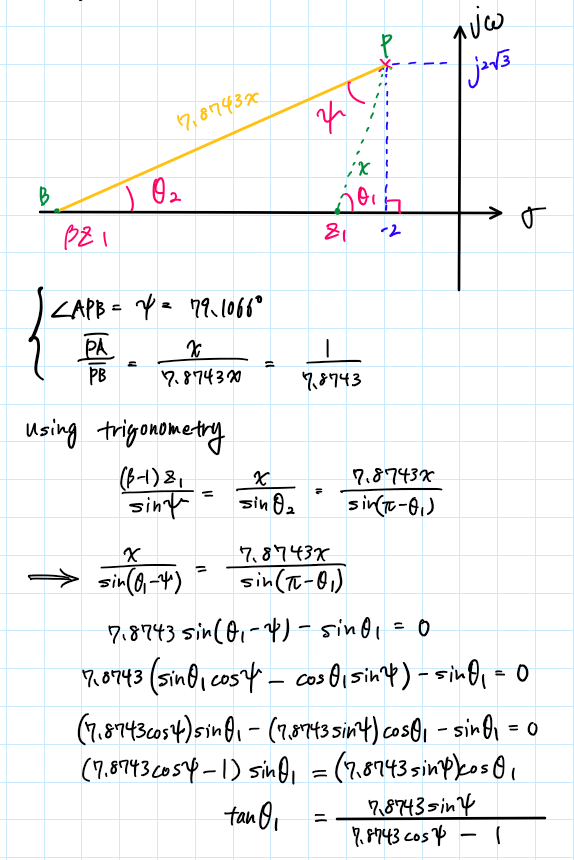


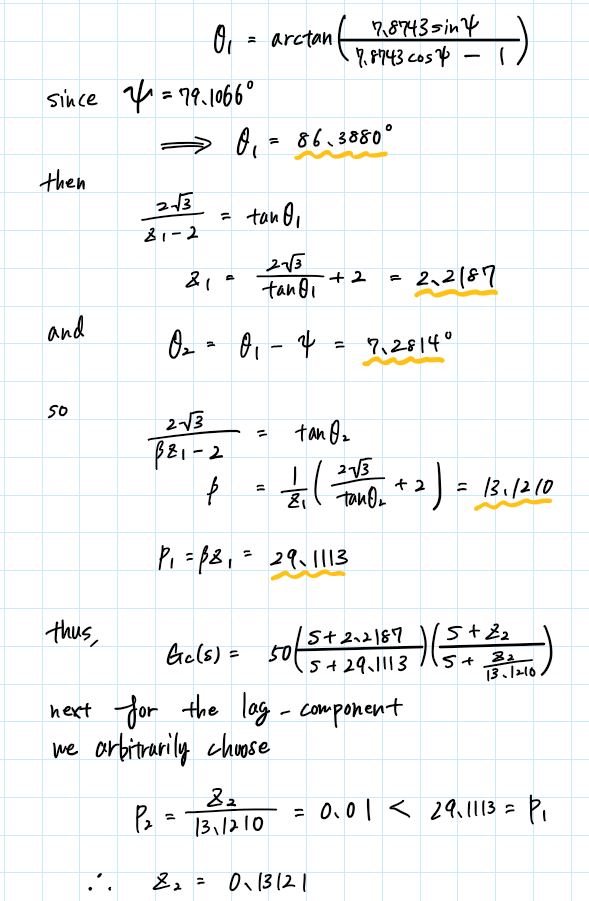


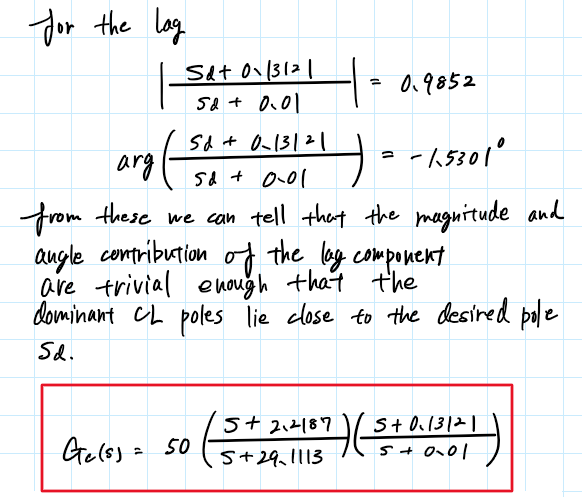


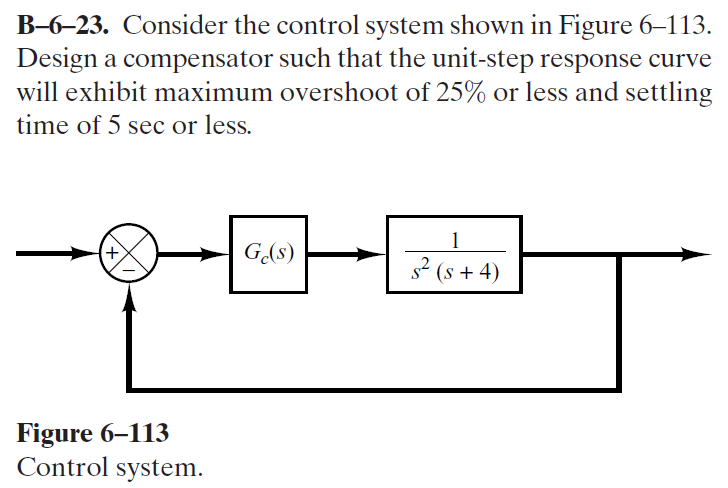


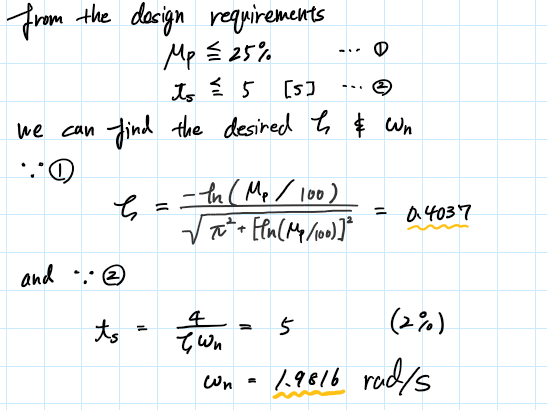


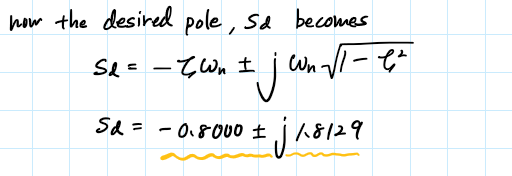


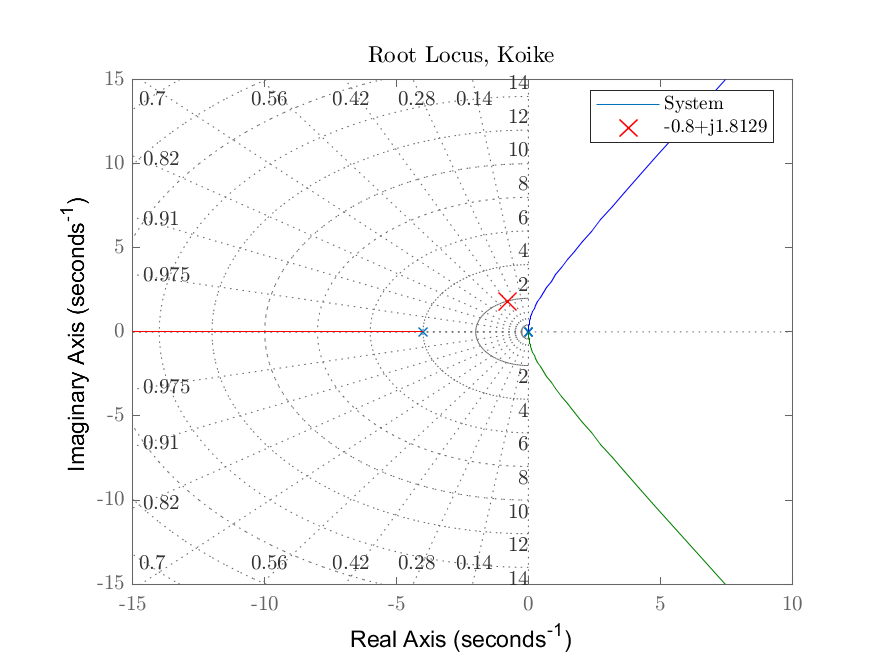


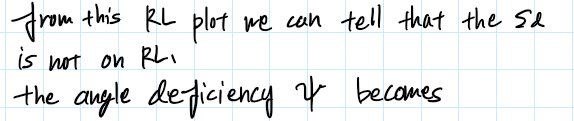


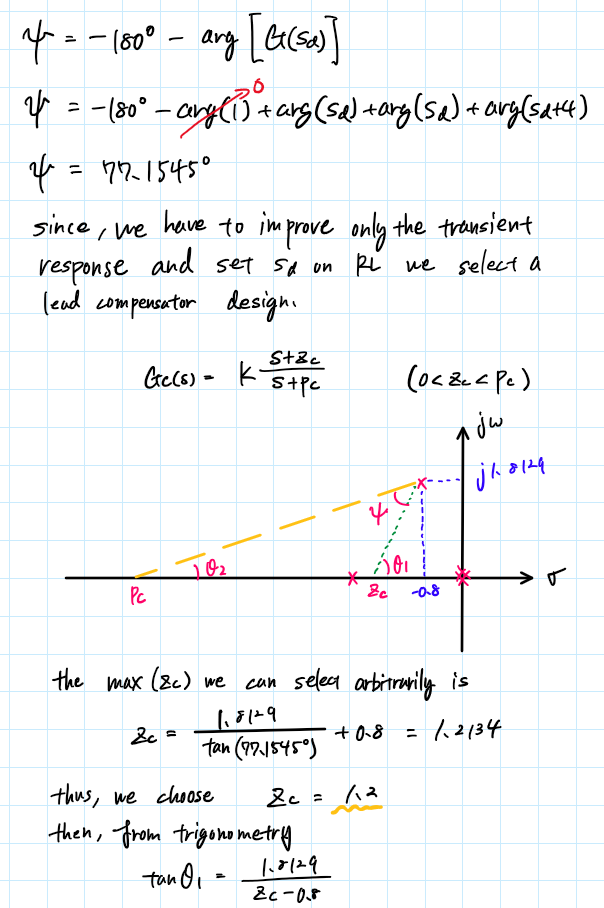


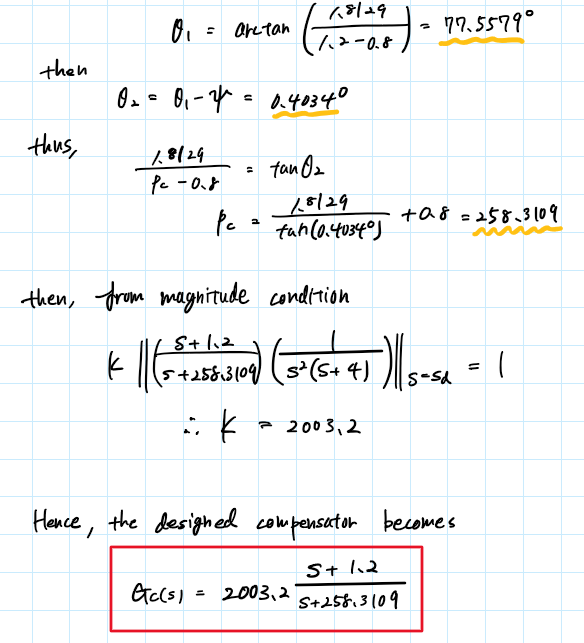


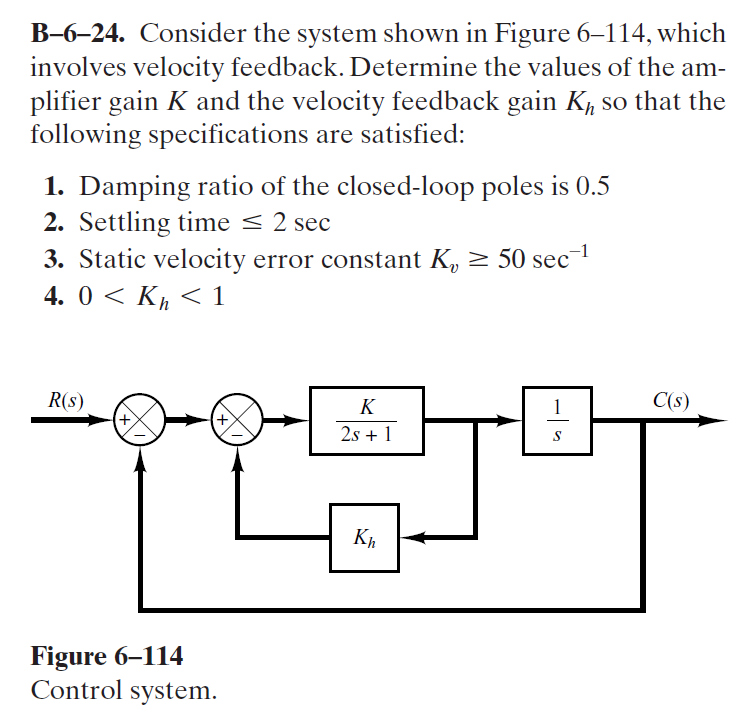


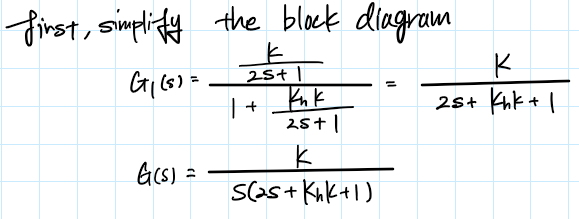


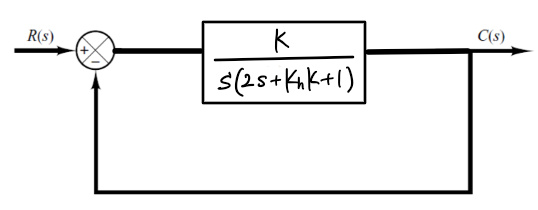


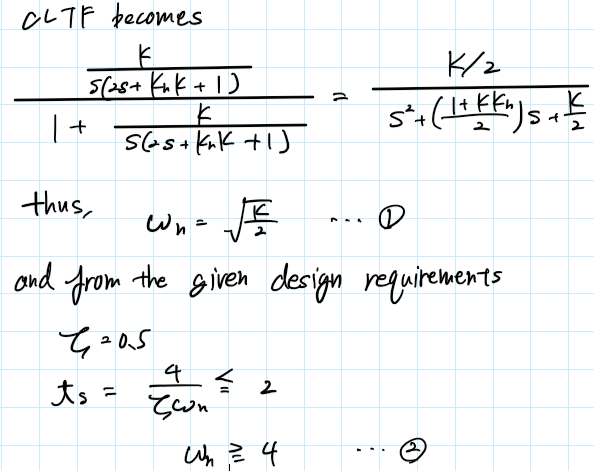


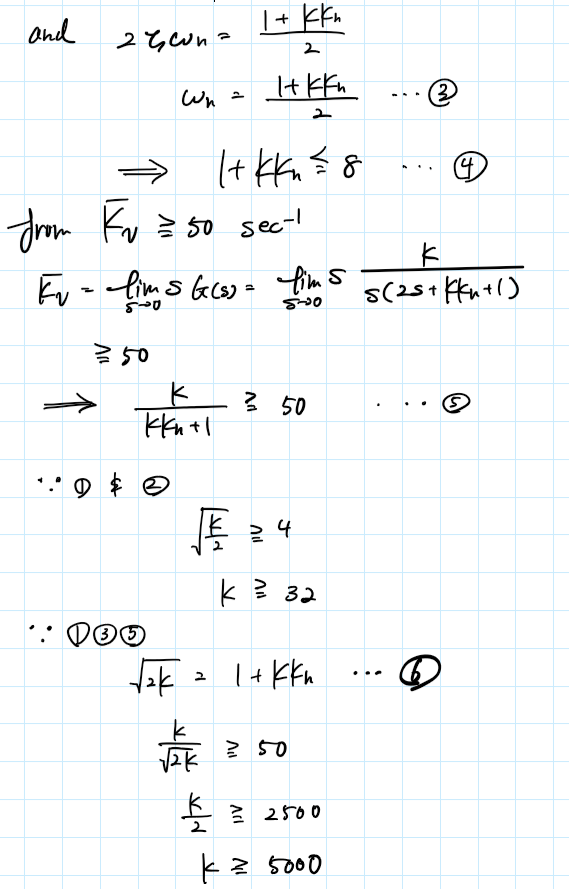


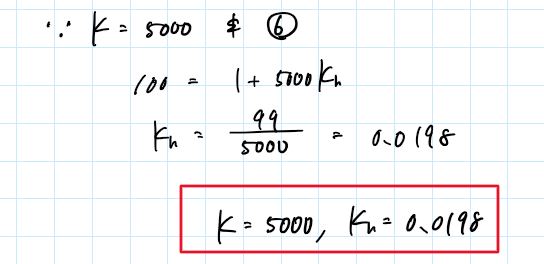


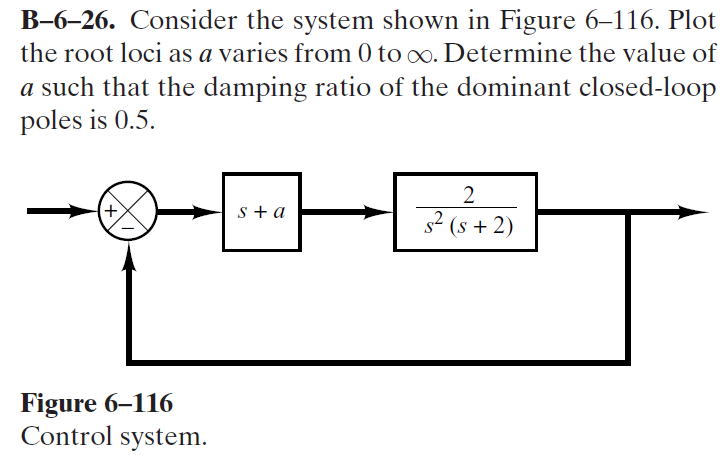


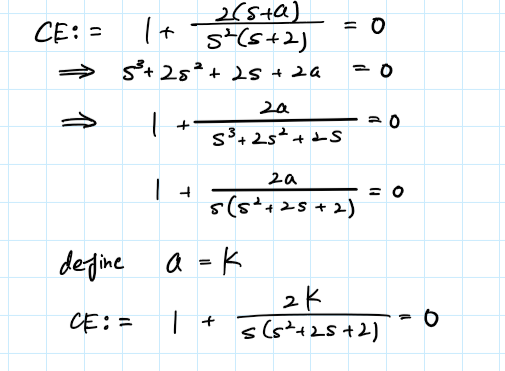


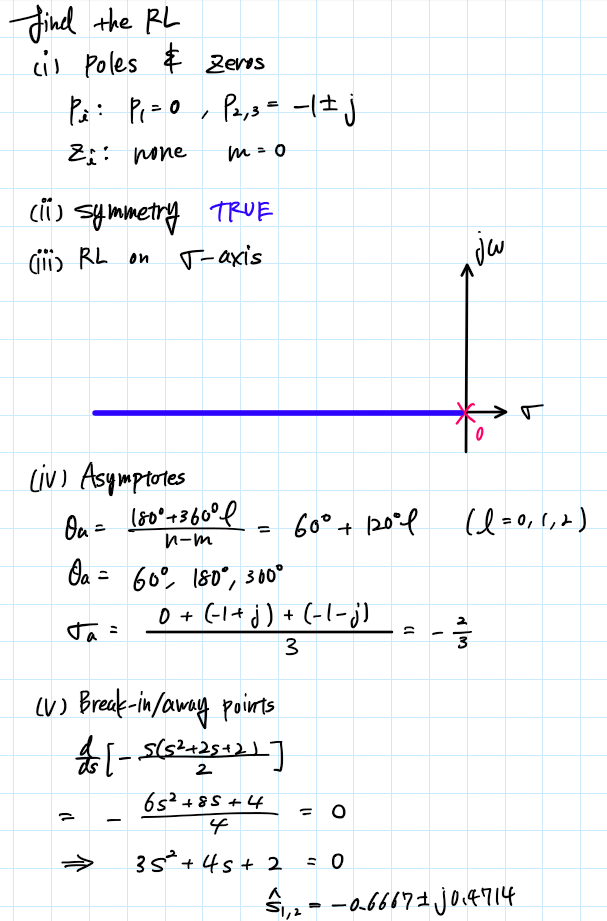


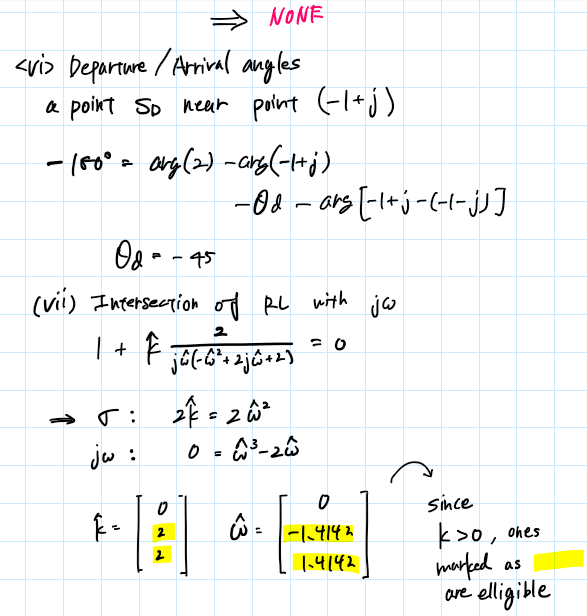


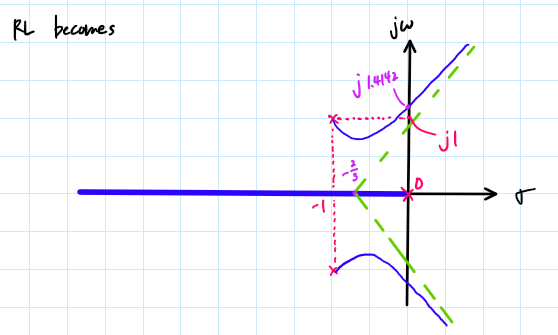


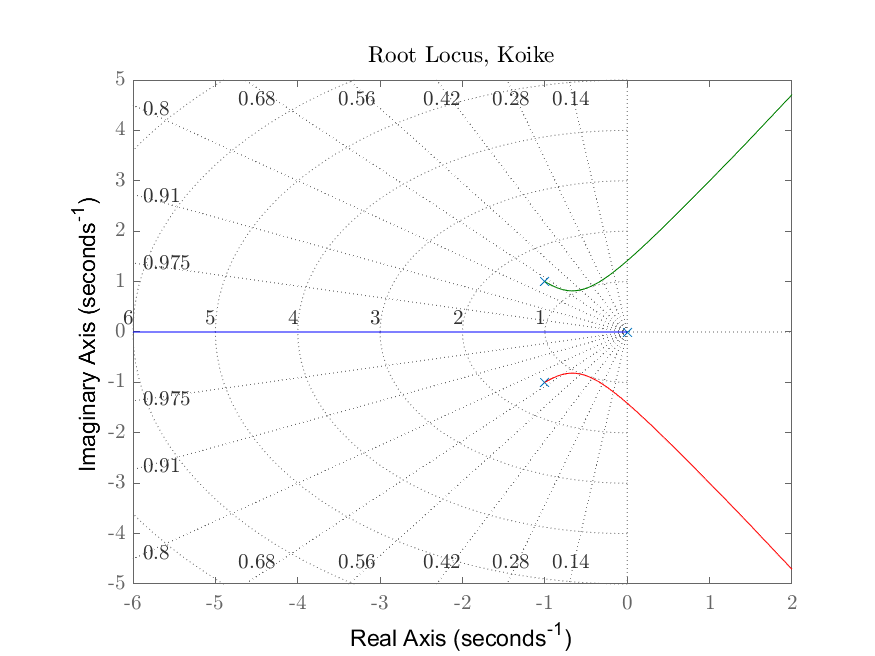


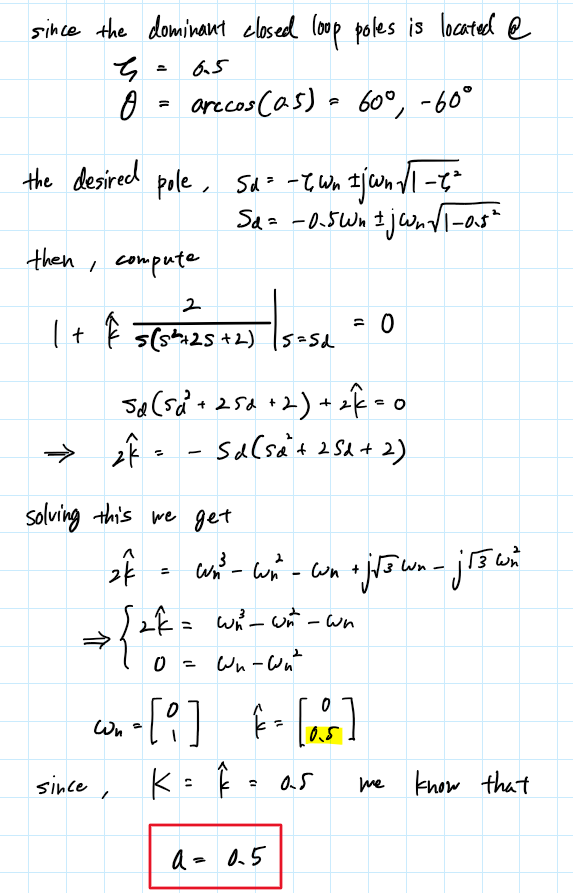


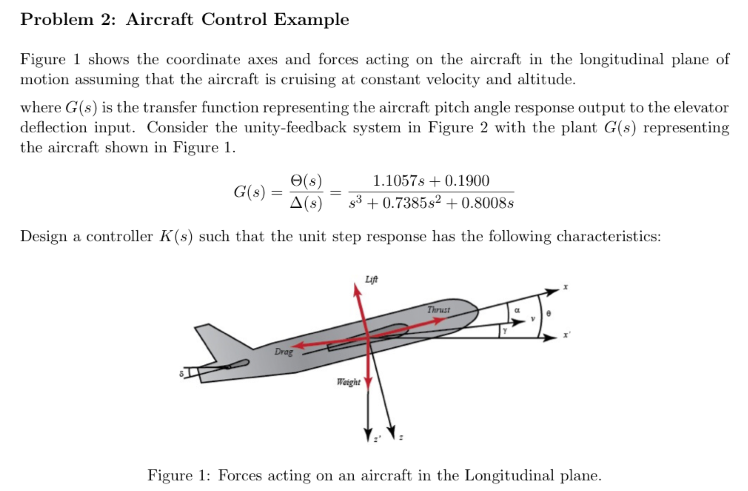


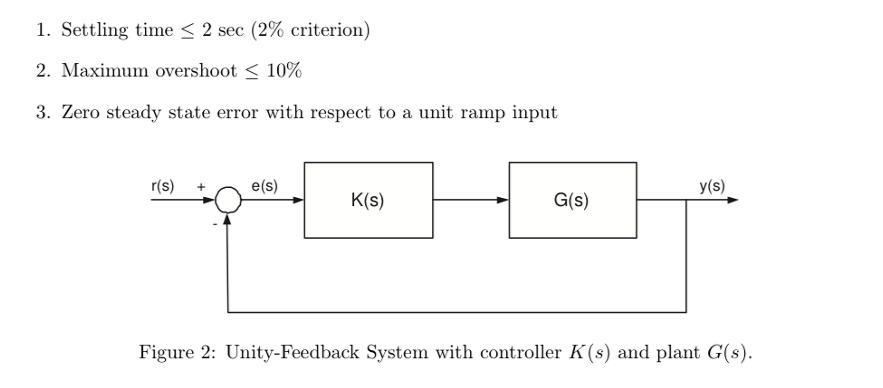


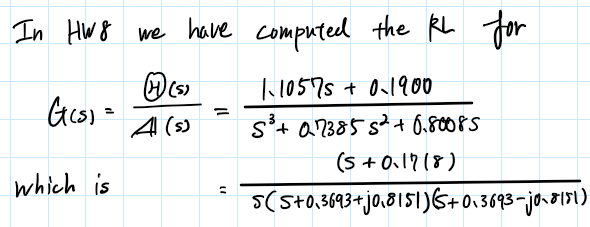


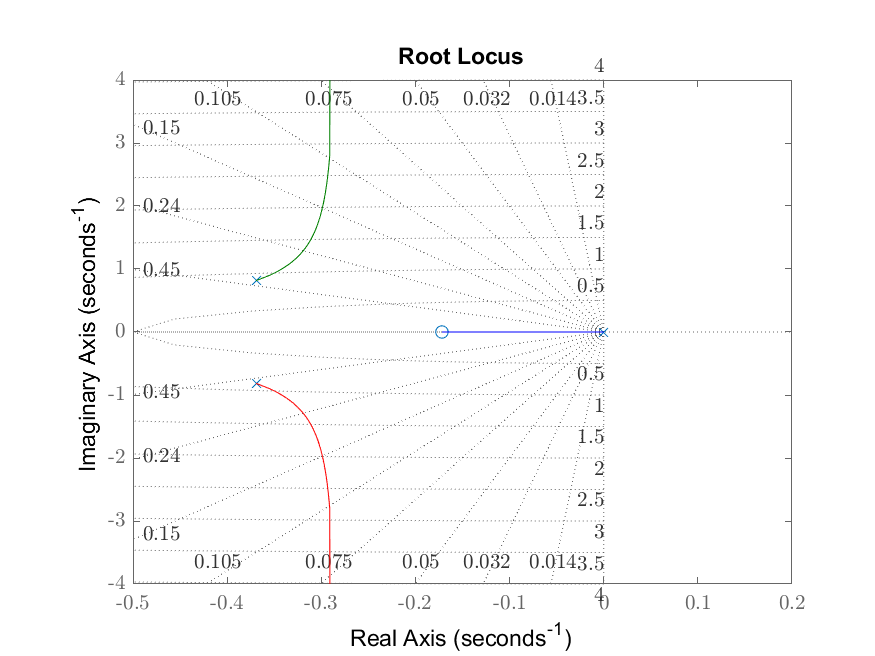


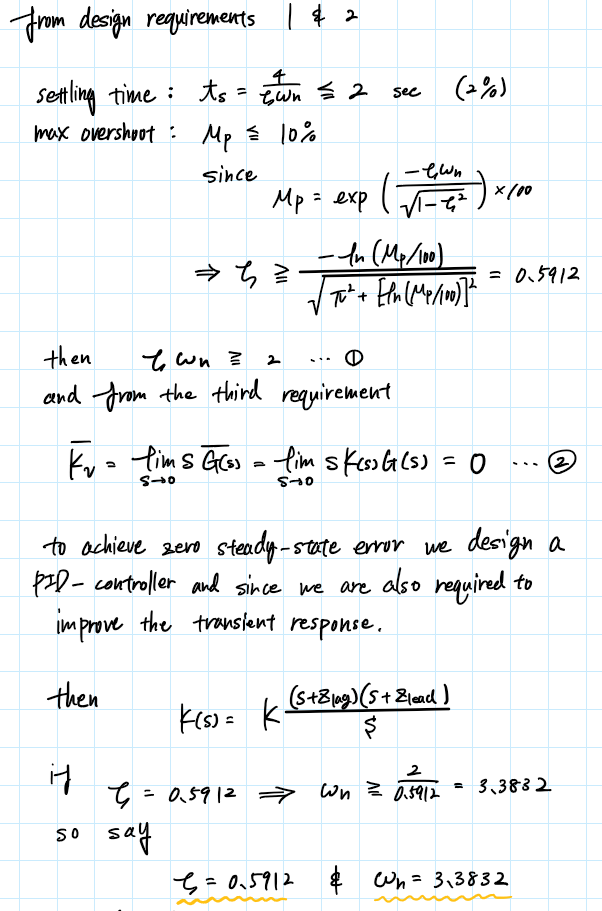


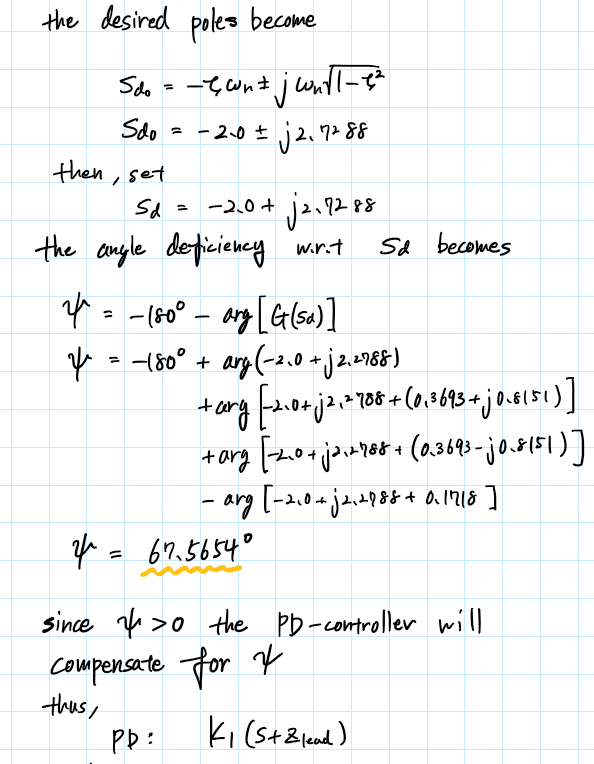


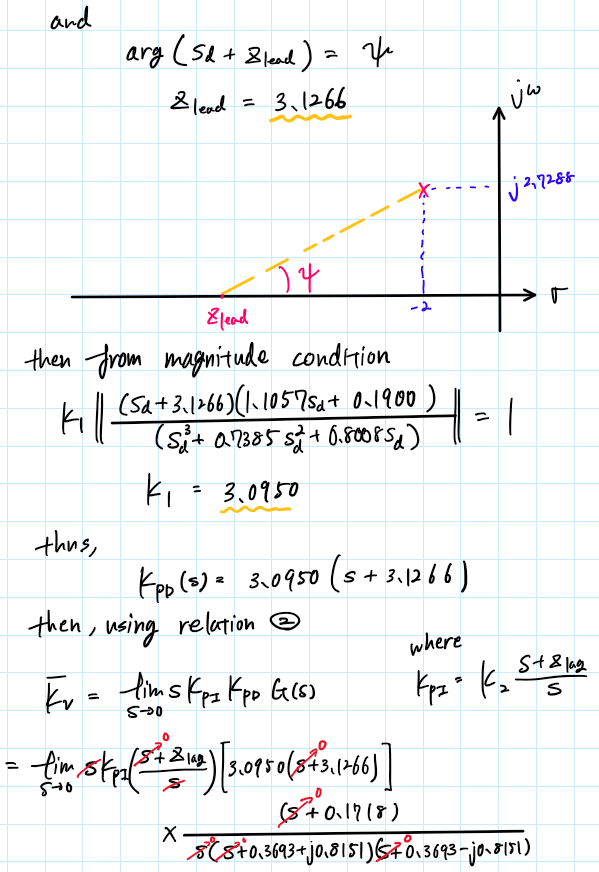


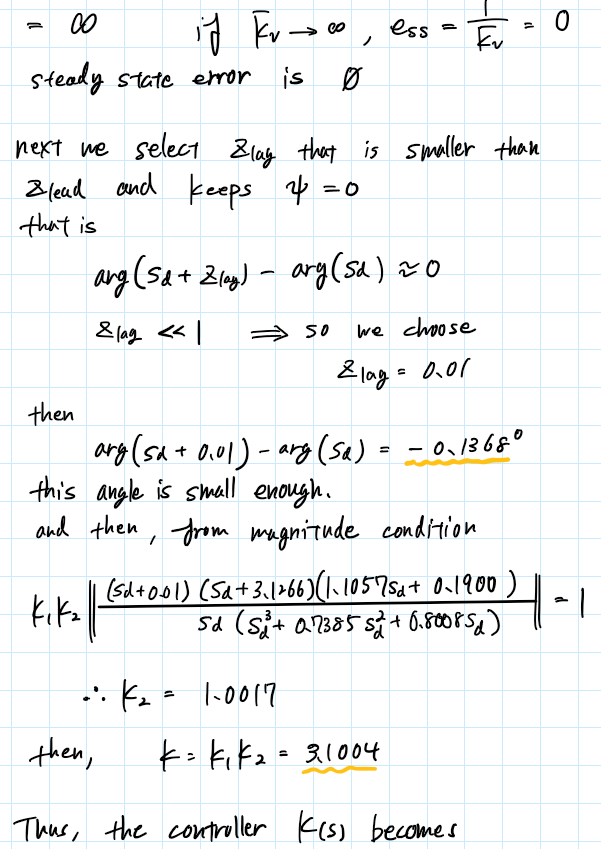


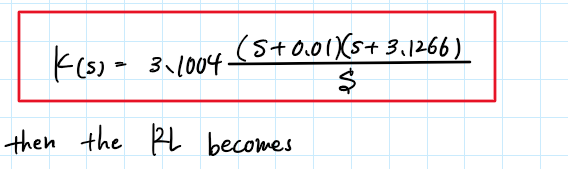


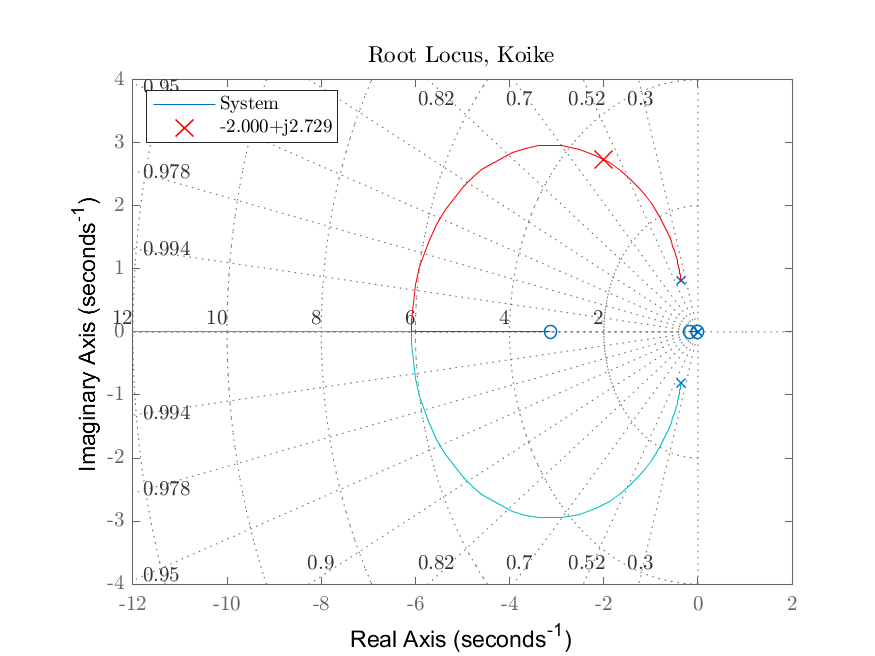












Appendix

## **AAE364 HW9 MATLAB CODE**

clear all; close all; clc;

fdir = 'C:\Users\Tomo\Desktop\studies\2020-Spring\AAE364\matlab\matlab\_output\hw9';

set(groot, 'defaulttextinterpreter',"latex");

set(groot, 'defaultAxesTickLabelInterpreter',"latex");

set(groot, 'defaultLegendInterpreter',"latex");

### B-6-19

% Define OLTF G(s)

num = 16; % numerator

den = conv([1 0],[1 4]); % denominator

% Desired pole

sd = -2 + 2i\*sqrt(3);

% Compute deficiency angle [deg]

psi = calc\_ang\_deficiency(num,den,sd);

% Plotting the RL (negative feedback)

fig1 = figure("Renderer","painters");

rlocusplot(tf(num,den)); sgrid;

title('Root Locus, Koike',"Interpreter","latex")

hold on

plot(real(sd),imag(sd),'xr','MarkerSize',12)

hold off

ylim([-2.5 3.6])

legend('System','-2+j2$\sqrt{3}$',"location","best")

saveas(fig1,fullfile(fdir,'B-6-19\_RL.png'));

### B-6-21

% Define OLTF G(s)

num = [0 10]; % numerator

den = conv([1 0],[1 2]); % denominator

den = conv(den,[1 5]);

% Desired pole

Kv = 50; % [s-1]

sd = -2 + 2i\*sqrt(3);

% Compute deficiency angle [deg]

psi = calc\_ang\_deficiency(num,den,sd)

% Plotting the RL (negative feedback)

fig2 = figure("Renderer","painters");

rlocusplot(tf(num,den)); sgrid;

title('Root Locus, Koike',"Interpreter","latex")

hold on

plot(real(sd),imag(sd),'xr','MarkerSize',12)

hold off

ylim([-2.5 4.0])

legend('System','-2+j2$\sqrt{3}$',"location","best")

saveas(fig2,fullfile(fdir,'B-6-21\_RL.png'));

% Finding z1, p1 (beta\*z1), and Kc

Kc = Kv;

z1\_lim = imag(sd)/tand(psi) - real(sd); % condition for z1

theta1 = atan(7.8743\*sind(psi)/(7.8743\*cosd(psi) - 1));

theta1\_deg = rad2deg(theta1);

z1 = imag(sd)/tan(theta1) + 2;

theta2\_deg = theta1\_deg - psi;

beta = (imag(sd)/tand(theta2\_deg) + 2)/z1;

% lag component

p2 = 0.01;

mag\_lag = abs((sd + p2\*beta)/(sd + p2));

ang\_lag = rad2deg(angle(sd + p2\*beta) - angle(sd + p2));

### B-6-23

% Design requirements

Mp = 25; % [%]

ts = 5; % settling time

% Design parameters

zeta = calc\_zetaFromMOS\_or\_MOSFromzeta(Mp,"zeta");

wn = 4/5/zeta;

% Desired pole (positive imag)

sd = -zeta\*wn + wn\*1i\*sqrt(1 - zeta^2);

% OLTF

num = [0 1];

den = conv([1 0],[1 0]);

den = conv(den,[1 4]);

% Plotting the RL (negative feedback)

fig3 = figure("Renderer","painters");

rlocusplot(tf(num,den)); sgrid;

title('Root Locus, Koike',"Interpreter","latex")

hold on

plot(real(sd),imag(sd),'xr','MarkerSize',12)

hold off

sd\_txt = sprintf("%0.3f+j%0.3f",real(sd),imag(sd));

legend('System',sd\_txt,"location","best")

saveas(fig3,fullfile(fdir,'B-6-23\_RL.png'));

% Angle deficiency

psi = calc\_ang\_deficiency(num,den,sd)

% Selecting zc, pc, and K

zc\_lim = imag(sd)/tand(psi) - real(sd) % condition for zc

zc = 1.2;

theta1\_deg = atand(imag(sd)/(zc + real(sd)));

theta2\_deg = theta1\_deg - psi;

pc = imag(sd)/tand(theta2\_deg) - real(sd);

% Calculate K

syms K

Gc = (sd + zc)/(sd + pc);

G = 1/sd^2/(sd + 4);

eqn = K\*abs(Gc\*G) == 1

K = double(solve(eqn,K));

### B-6-24

% Design parameters

zeta = 0.5;

ts = 2;

wn = 4/ts/zeta

% Desired pole (positive imag)

sd = -zeta\*wn + wn\*1i\*sqrt(1 - zeta^2)

Kh = 99/5000;

### B-6-26

% RL

num = [0 2];

den = [1 2 2 0];

[poles,zrs,angs,sigma,bi\_pt,T\_P,T\_Z,k,w,fig1] = rootLocus\_stepBystep\_negFeedback(num,den)

saveas(fig1,fullfile(fdir,"B-6-26\_RL.png"))

% Find gain, K when zeta = 0.5

zeta = 0.5;

syms K wn s

assume(K,'real');

assume(wn,'real');

p = -zeta\*wn + wn\*1j\*sqrt(1 - zeta^2);

RHS = (s\*(s^2 + 2\*s + 2))

RHS = subs(RHS,s,p)

RHS = expand(RHS)

eqn1 = 2\*K == -real(RHS)

eqn2 = 0 == -imag(RHS)

res = solve([eqn1 eqn2],[wn K]);

wn = double(res.wn)

Kh = double(res.K)

Kh = nonzeros(Kh)

### P2

% Define system

num = [1.1057 0.1900];

den = [1 0.7385 0.8008 0];

[poles,zrs,angs,sigma,bi\_pt,T\_P,T\_Z,k,w,fig1] = rootLocus\_stepBystep\_negFeedback(num,den);

% Design parameters

zeta = calc\_zetaFromMOS\_or\_MOSFromzeta(10,"zeta");

wn = 2/zeta;

% Desired pole (positive imag)

sd = -zeta\*wn + wn\*1i\*sqrt(1 - zeta^2);

% Angle deficiency

psi = calc\_ang\_deficiency(num,den,sd)

% Find z\_lead

syms z\_lead

assume(z\_lead,{'real','positive'})

eqn = angle(sd + z\_lead) == deg2rad(psi);

z\_lead = double(solve(eqn,z\_lead))

% Find K1

syms K1

num2 = conv(num,[1 z\_lead]);

G = create\_TF\_expression(num2,den); % Expression with "syms" of "s"

G\_sd = subs(G,s,sd);

eqn = K1\*abs(G\_sd) == 1;

K1 = double(solve(eqn,K1))

% Find z\_lag

z\_lag = 0.01;

d\_ang = rad2deg(angle(sd + z\_lag) - angle(sd))

% Find K2

syms K2

num3 = conv(num2,[1 z\_lag]);

den2 = conv(den,[1 0]);

G = create\_TF\_expression(num3,den2);

G\_sd = subs(G,s,sd);

eqn = K1\*K2\*abs(G\_sd) == 1;

K2 = double(solve(eqn,K2))

% Find K

K = K1\*K2

% Plot RL

fig6 = figure("Renderer","painters")

rlocus(tf(K\*num3,den2))

sgrid

title("Root Locus, Koike",'Interpreter',"latex")

hold on

plot(real(sd),imag(sd),'xr','MarkerSize',12)

hold off

sd\_txt = sprintf("%0.3f+j%0.3f",real(sd),imag(sd));

legend('System',sd\_txt,"location","best")

saveas(fig6,fullfile(fdir,'P2\_RL\_new.png'));

function [poles,zrs,angs,sigma,bi\_pt,T\_P,T\_Z,k,w,fig1] = rootLocus\_stepBystep\_negFeedback(num,den)

%{

NAME: ROOTLOCUS\_STEPBYSTEP\_NEGFEEDBACK

AUTHOR: TOMOKI KOIKE

INPUTS: (1) num: THE NUMERATOR OF THE TRANSFER FUNCTION

(2) den: THE DENOMINATOR OF THE TRANSFER FUNCTION

OUTPUTS: (1) poles: POLES OF THE TRANSFER FUNCTION

(2) zrs: ZEROS OF THE TRANSFER FUNCTION

(3) angs: ANGLES OF THE ASYMPTOTES

(4) sigma: INTERSECTION OF THE ASYMPTOTES

(5) bi\_pt: BREAK-IN/AWAY POINT

(6) T\_P: TABLE WITH EACH POLE AND THEIR

DEPARTURE OR ARRIVAL ANGLES

(6) T\_Z: TABLE WITH EACH ZERO AND THEIR

DEPARTURE OR ARRIVAL ANGLES

(7) k: VALUE K\_HAT FOR INTERSECTION WITH IM AXIS

(8) w: INTERSECTION POINT WITH THE IM AXIS

(9) fig1: THE FIGURE WITH THE ROOT LOCUS PLOT

DESCRIPTION: CONDUCTS THE 7 STEP PROCEDURE OF THE ROOT LOCUS

ANALYSIS AND DISPLAYS THE RESULTS AS WELL AS THE PLOT FOR A NEGATIVE

FEEDBACK LOOP

%}

% STEP1 - POLES & ZEROS

poles = roots(den);

zrs = roots(num);

% STEP2 - SYMMETRY (\*TAKEN FOR GRANTED)

% STEP3 - ROOT LOCUS ON REAL AXIS (\*OMMITTED)

% STEP4 - ASYMPTOTES

[angs,sigma] = RL\_asymptote(zrs,poles);

% STEP5 - BREAK-IN/AWAY POINTS

bi\_pt = break\_in\_away\_pt(num,den);

% STEP6 - ANGLE OF DEPARTURE

[T\_P, T\_Z] = departure\_arrival\_angle\_calc(zrs, poles);

% STEP7 - INTERSECTION WITH IMAGINARY AXIS

[k,w] = intersection\_IM\_axis(num,den);

% DEFINE THE TRANSFER FUNCTION

L = tf(num, den);

% PLOTTING THE ROOT LOCUS

fig1 = figure(1);

rlocus(L)

title('Root Locus, Koike','interpreter','Latex')

sgrid

end

function psi = calc\_ang\_deficiency(num,den,s\_d)

%{

Function: calc\_ang\_deficiency

Author: Tomoki Koike

Description: Computes the deficiency angle for a certain system from

its given open-loop transfer function and desired

pole.

>>Inputs

num: the numerator of the open-loop transfer function

den: the denominator of the open-loop transfer function

s\_d: the desired pole

Outputs<<

psi: the angle deficiency

%}

% Get the length of each numerator and denominator

num\_len = length(num);

den\_len = length(den);

% Preset an array with the order of magnitudes (i.e. s^3, s^2, s^1, s^0)

% corresponding to the numerator and denominator

O\_num = (num\_len-1):-1:0;

O\_den = (den\_len-1):-1:0;

% Define a system equation of s to compute deficiency angle

syms s

N = factor(dot(num,s.^O\_num),'FactorMode','complex');

D = factor(dot(den,s.^O\_den),'FactorMode','complex');

N\_angs = angle(subs(N,s,s\_d));

D\_angs = angle(subs(D,s,s\_d));

psi = -pi - sum(N\_angs) + sum(D\_angs);

psi = double(rad2deg(psi));

end

function output = calc\_zetaFromMOS\_or\_MOSFromzeta(MOS\_or\_zeta, type)

%{

inputs: 1) MOS\_or\_zeta: maximum overshoot or zeta (damping ratio) input

the one of the two will be chosen depending on the second

input "type"

2) type: string "MOS" or "zeta" indicates what output the

user requires

outputs: 1) output: returns either the MOS or zeta

%}

if type == "MOS"

zeta = MOS\_or\_zeta;

output = exp(-zeta\*pi/sqrt(1-zeta^2))\*100;

elseif type == "zeta"

MOS = MOS\_or\_zeta;

output = -log(MOS/100)/sqrt(pi^2 + (log(MOS/100))^2);

end

end

function [angs, sigma] = RL\_asymptote(zrs, poles)

n = length(poles);

m = length(zrs);

angs = zeros([1,n-m]);

for i = 0:(n-m)-1

angs(i+1) = (180 + 360\*i)/(n - m);

end

sigma = (sum(poles) - sum(zrs))/(n - m);

end

function [table\_P, table\_Z] = departure\_arrival\_angle\_calc(zrs, poles)

%{

NAME: DEPARTURE\_ARRIVAL\_ANGLE\_CALC

AUTHOR: TOMOKI KOIKE

INPUTS: (1) zrs: THE ZEROS OF THE TRANSFER FUNCTION

(2) poles: THE POLES OF THE TRANSFER FUNCTION

OUTPUTS: (1) TABLE\_P : TABLE OF ALL THE POLES' DEPARTURE ANGLES

(2) TABLE\_Z : TABLE OF ALL THE ZOLES' DEPARTURE ANGLES

DESCRIPTION: CALCULATES ALL THE DEPARTURE ANGLES AND ARRIVALS ANGLES

FOR THE PROVIDED ZEROS AND POLES OF A TRANSFER FUNCTION FOR NEGATIVE

FEEDBACK LOOP

%}

% PREALLOCATE ARRAY THETA TO STORE ALL ANGLES FOR THE POLES

theta\_P = zeros([1,(length(poles))]);

% ANGLE FOR A FICTICIOUS POINT CLOSE TO EACH POLE

for n = 1:length(poles)

obj = poles(n);

% ANGLES FROM THE ZERO POINT TO A THE CURRENT POINT

if not(isempty(zrs))

for i = 1:length(zrs)

theta\_P(n) = theta\_P(n) + angle(obj - zrs(i));

end

end

% ANGLE FROM ANOTHER POLE TO THE CURRENT POLE

for i = 1:length(poles)

theta\_P(n) = theta\_P(n) - angle(obj - poles(i));

end

% THE ANGLE BECOMES

theta\_P(n) = theta\_P(n) + deg2rad(180); % [rad]

end

% CREATING TABLE

rad\_P = reshape(theta\_P,[length(theta\_P),1]);

deg\_P = rad2deg(rad\_P);

table\_P = table(reshape(poles,[length(poles),1]),rad\_P,deg\_P);

table\_P.Properties.VariableNames = {'POLES','RADIUS','DEGREES'};

if not(isempty(zrs))

% PREALLOCATE ARRAY THETA TO STORE ALL ANGLES FOR THE POLES

theta\_Z = zeros([1,(length(zrs))]);

% ANGLE FOR A FICTICIOUS POINT CLOSE TO EACH ZERO

for n = 1:length(zeros)

obj = zrs(n);

% ANGLES FROM THE ZERO POINT TO A THE CURRENT POINT

if not(isempty(zrs))

for i = 1:length(zrs)

theta\_Z(n) = theta\_Z(n) + angle(obj - zrs(i));

end

end

% ANGLE FROM A POLE TO THE CURRENT ZERO POINT

for i = 1:length(poles)

theta\_Z(n) = theta\_Z(n) - angle(obj - poles(i));

end

% THE ANGLE BECOMES

theta\_Z(n) = -deg2rad(180) - theta\_Z(n); % [rad]

end

% CREATING TABLE

rad\_Z = reshape(theta\_Z,[length(theta\_Z),1]);

deg\_Z = rad2deg(rad\_Z);

table\_Z = table(reshape(zrs,[length(zrs),1]),rad\_Z,deg\_Z);

table\_Z.Properties.VariableNames = {'ZEROS','ANGLES','DEGREES'};

else

table\_Z = [];

end

end

function rts = break\_in\_away\_pt(num,den)

[q, d] = polyder(-den,num)

rts = roots(q)

rts = rts(rts==real(rts));

end

function [K, W] = intersection\_IM\_axis(num, den)

syms k w

n = length(den);

m = length(num);

f1 = 0; f2 = 0; p1 = 0; p2 = 0;

% RHS (denominator)

% when the largest order of s is even

if rem(n,2) == 1

% powers to the even numbers (real)

for i = 1:2:n

if rem(n-i,4) == 0

f1 = f1 + den(i)\*w^(n-i);

else

f1 = f1 + den(i)\*w^(n-i)\*(-1);

end

end

% powers to the odd numbers (imaginary)

for i = 2:2:n-1

if rem(n-i,4) == 1

f2 = f2 + den(i)\*w^(n-i);

else

f2 = f2 + den(i)\*w^(n-i)\*(-1);

end

end

% when the largest order of s is odd

elseif rem(n,2) == 0

% powers to the even numbers (real)

for i = 2:2:n

if rem(n-i,4) == 0

f1 = f1 + den(i)\*w^(n-i);

else

f1 = f1 + den(i)\*w^(n-i)\*(-1);

end

end

% powers to the odd numbers (imaginary)

for i = 1:2:n-1

if rem(n-i,4) == 1

f2 = f2 + den(i)\*w^(n-i);

else

f2 = f2 + den(i)\*w^(n-i)\*(-1);

end

end

end

% LHS

% when the largest order of s is even

if rem(m,2) == 1

% powers to the even numbers (real)

for i = 1:2:m

if rem(m-i,4) == 0

p1 = p1 + num(i)\*w^(m-i);

else

p1 = p1 + num(i)\*w^(m-i)\*(-1);

end

end

% powers to the odd numbers (imaginary)

for i = 2:2:m-1

if rem(m-i,4) == 1

p2 = p2 + num(i)\*w^(m-i);

else

p2 = p2 + num(i)\*w^(m-i)\*(-1);

end

end

% when the largest order of s is odd

elseif rem(m,2) == 0

% powers to the even numbers (real)

for i = 2:2:m

if rem(m-i,4) == 0

p1 = p1 + num(i)\*w^(m-i);

else

p1 = p1 + num(i)\*w^(m-i)\*(-1);

end

end

% powers to the odd numbers (imaginary)

for i = 1:2:m-1

if rem(m-i,4) == 1

p2 = p2 + num(i)\*w^(m-i);

else

p2 = p2 + num(i)\*w^(m-i)\*(-1);

end

end

end

% Solving the system equations

Re = k\*p1 == -f1

Im = k\*p2 == -f2

a = vpasolve([Re Im], [k w]);

K = double(a.k);

W = double(a.w);

end