AAE 364: Control Systems Analysis

HW8: Controller Design and Root Locus Anaylsis

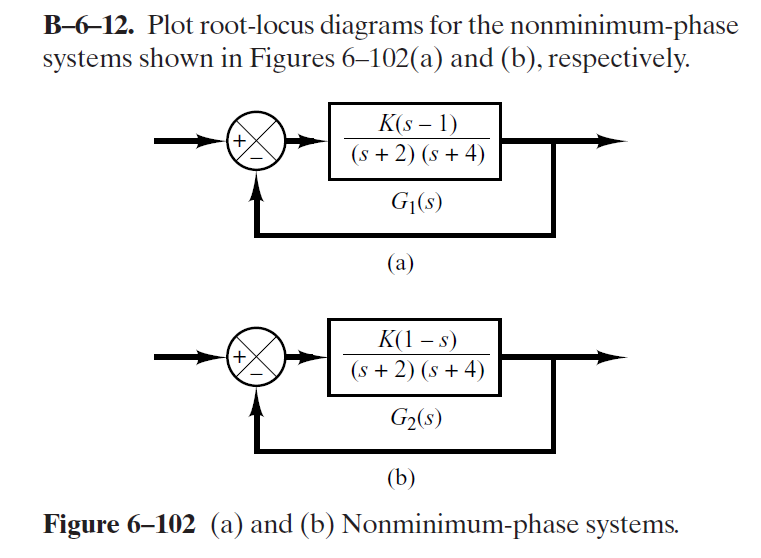
Dr. Sun

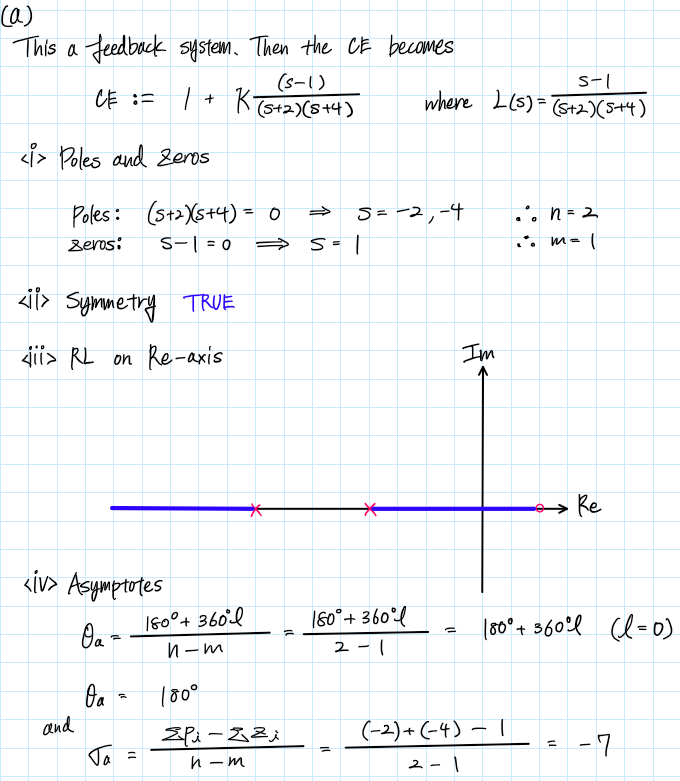
School of Aeronautical & Astronautical Engineering

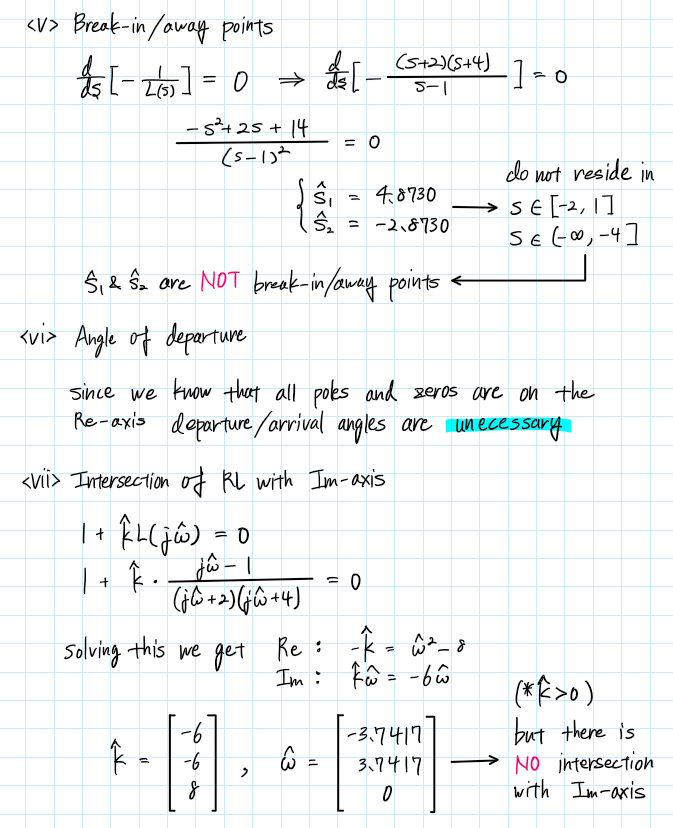
Purdue University

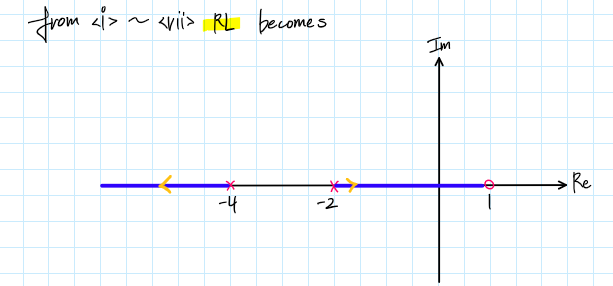
Tomoki Koike

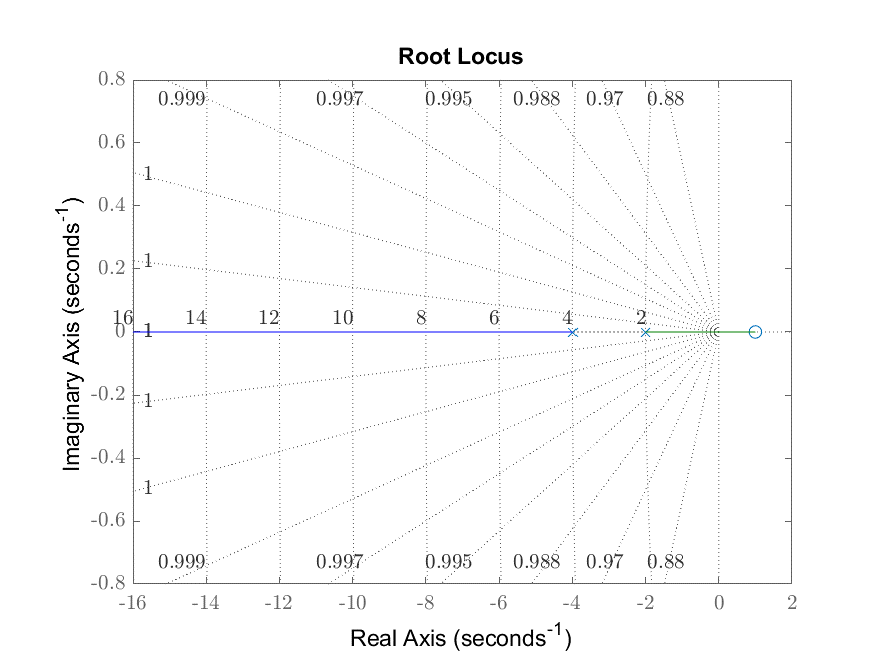
Friday March 27th 2020

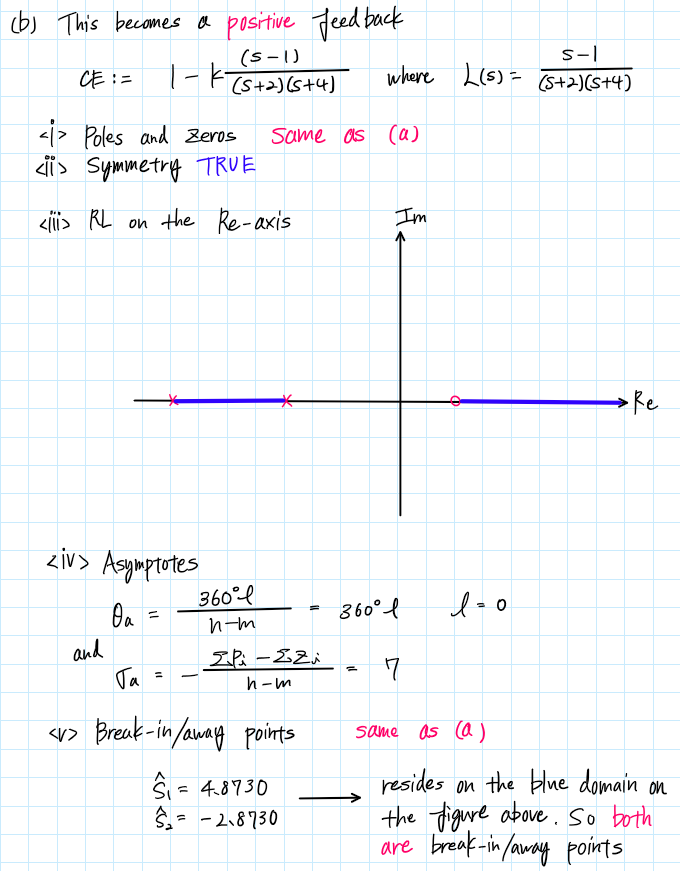


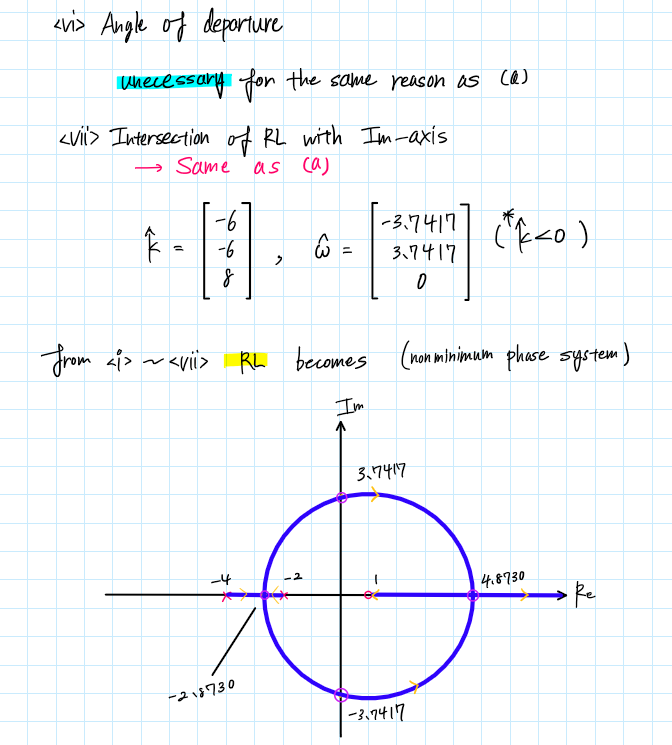


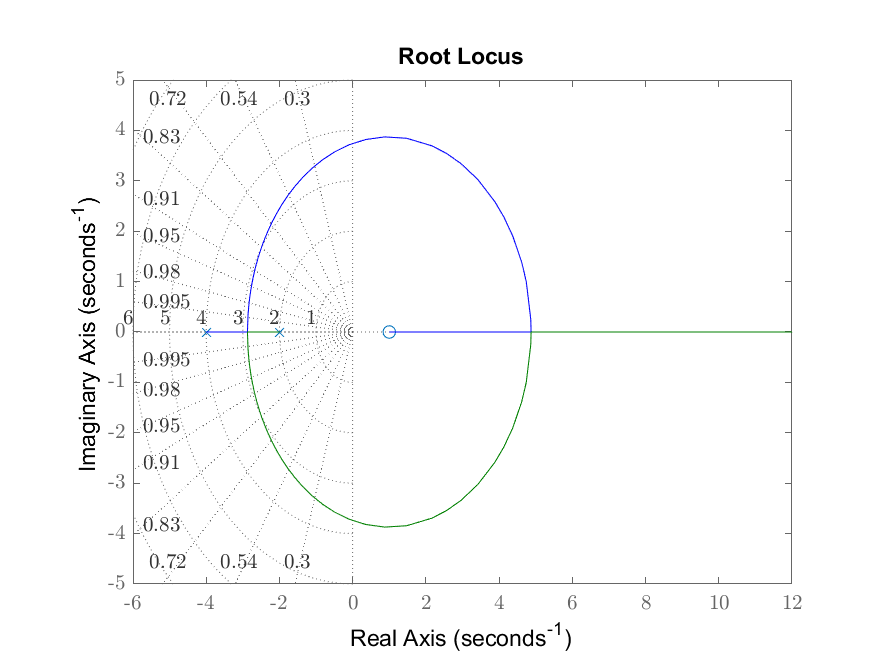


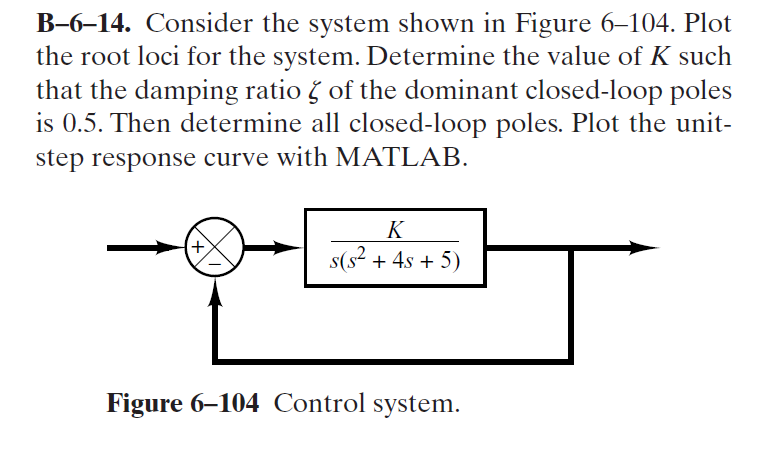


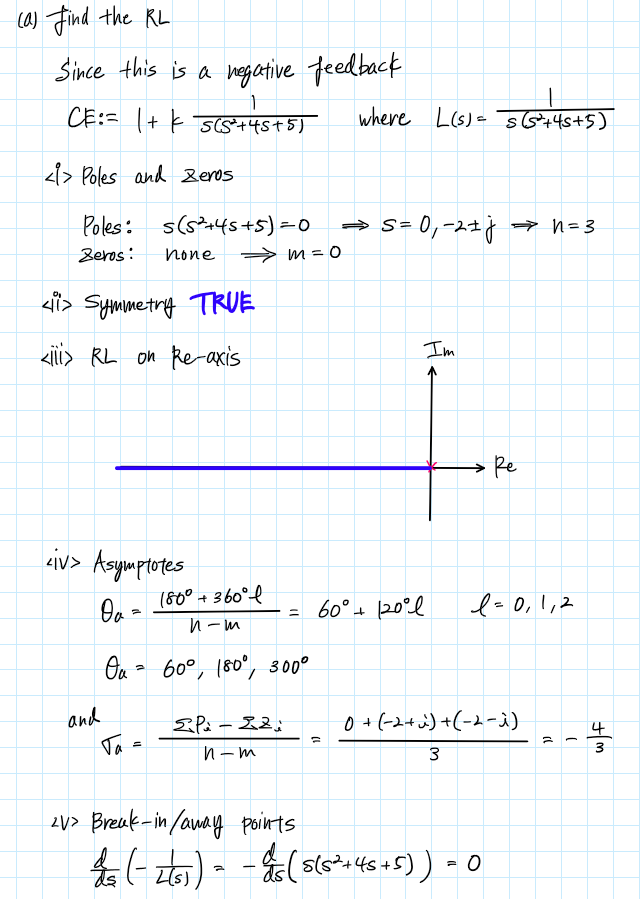


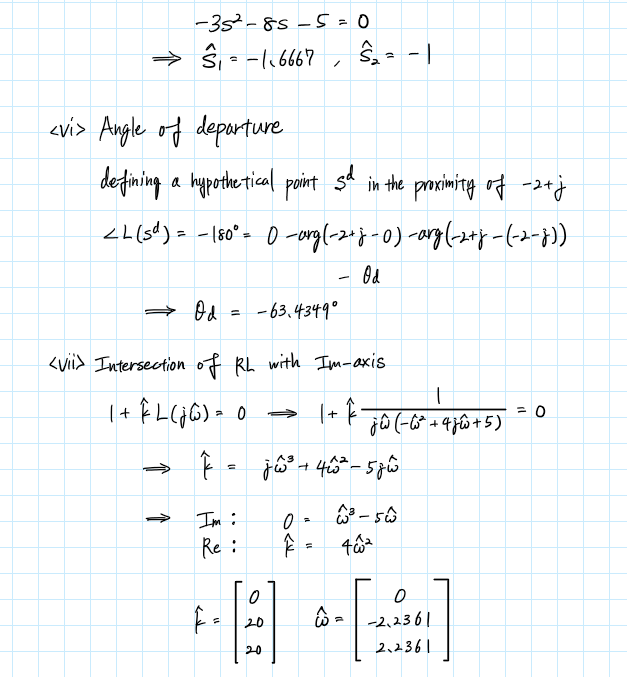


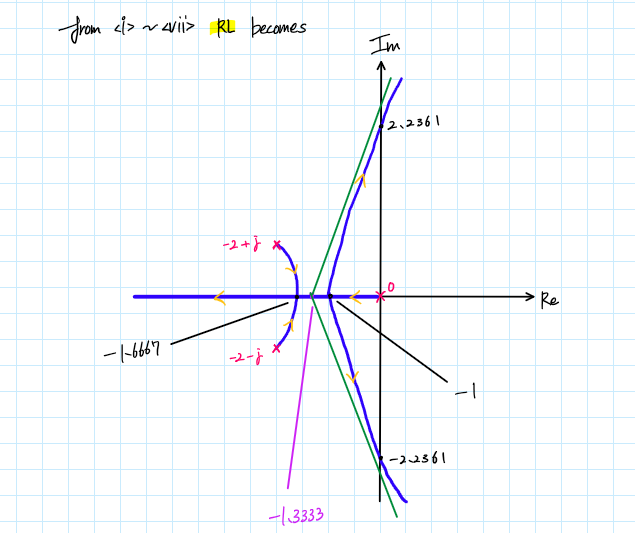


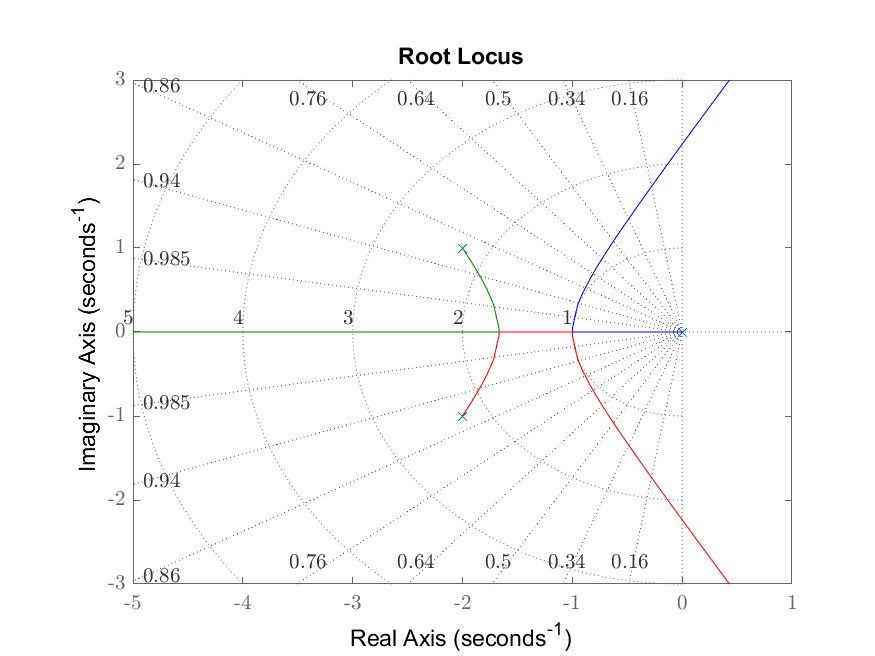


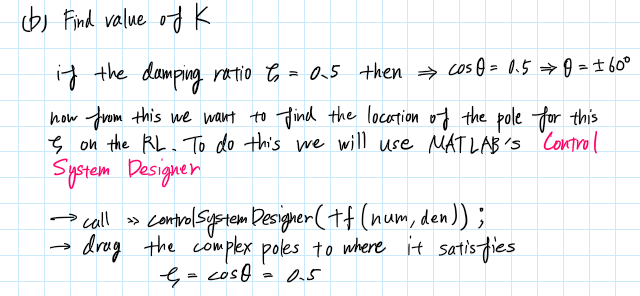


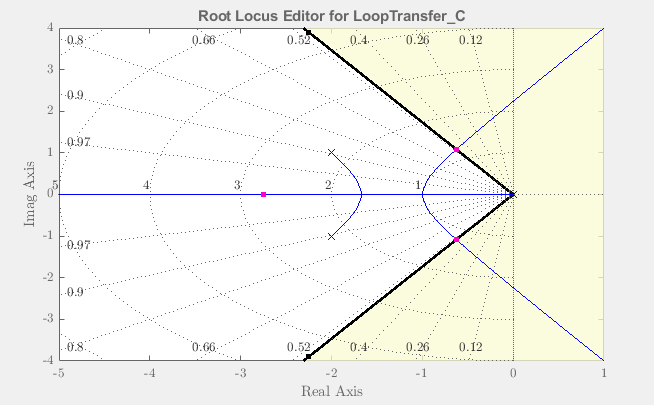


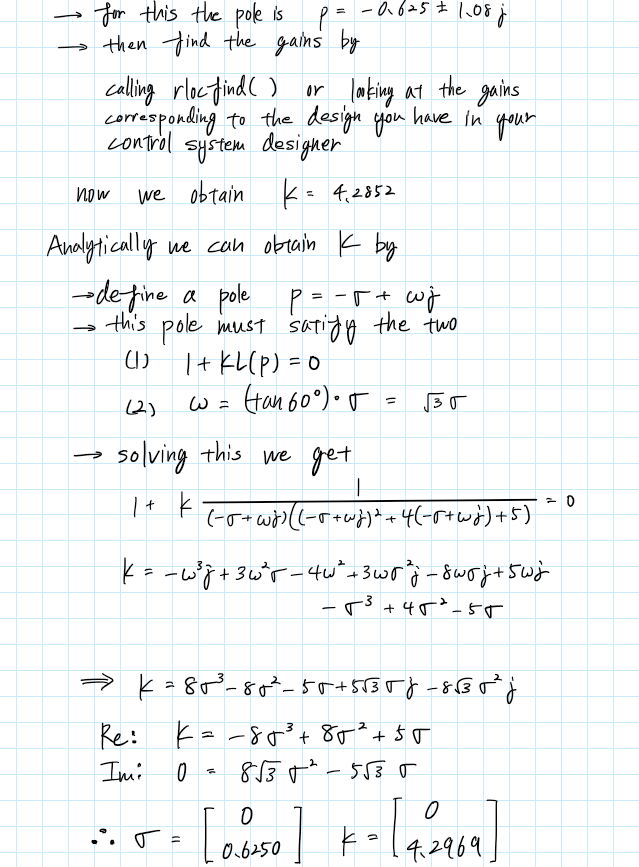


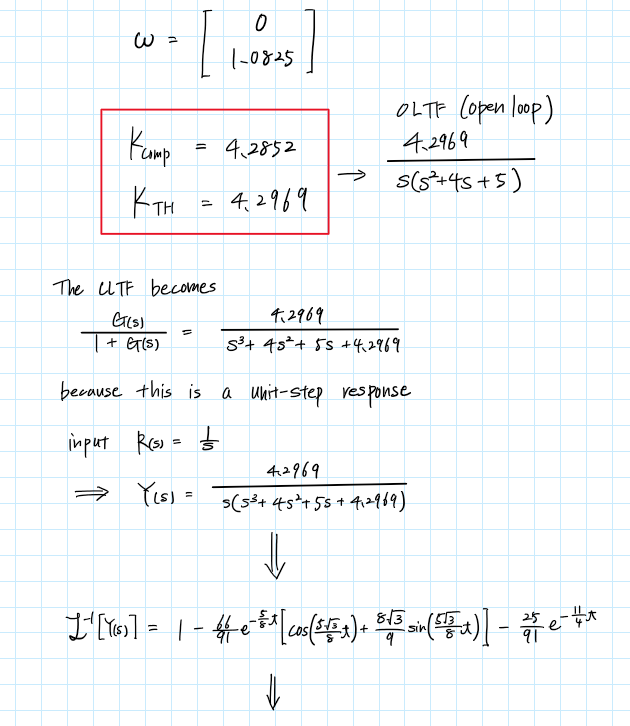


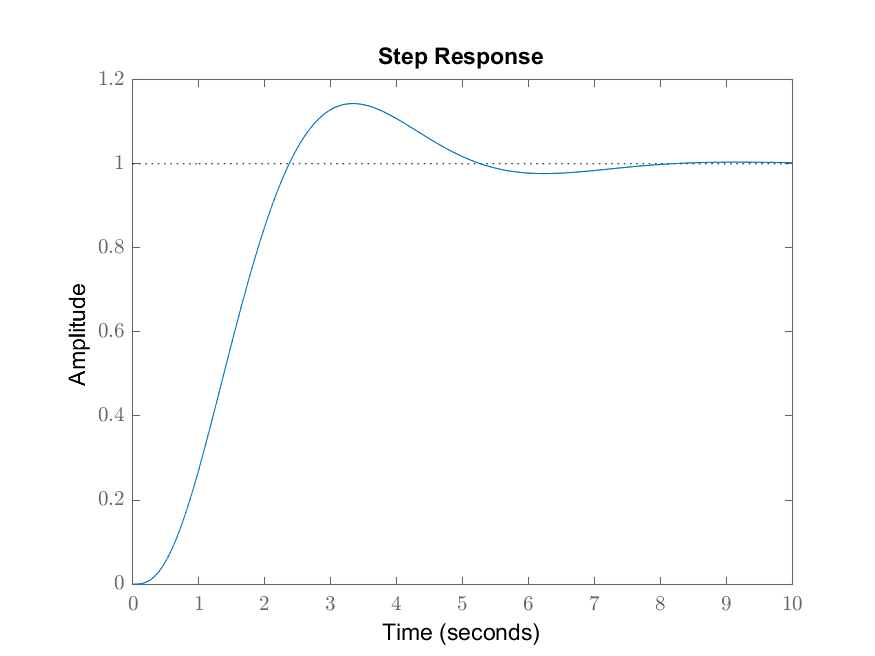


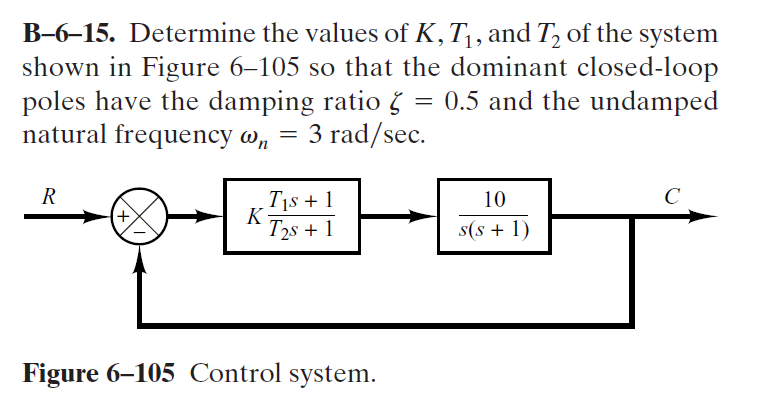


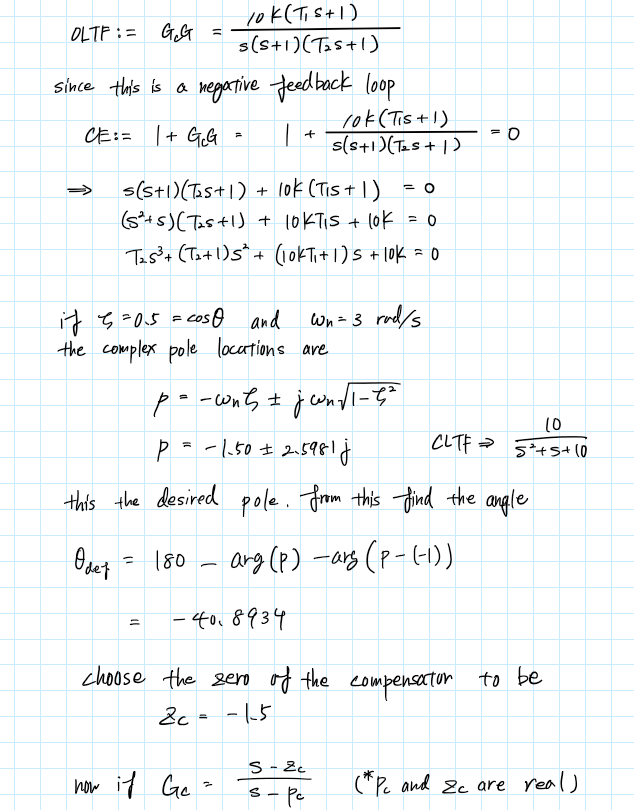


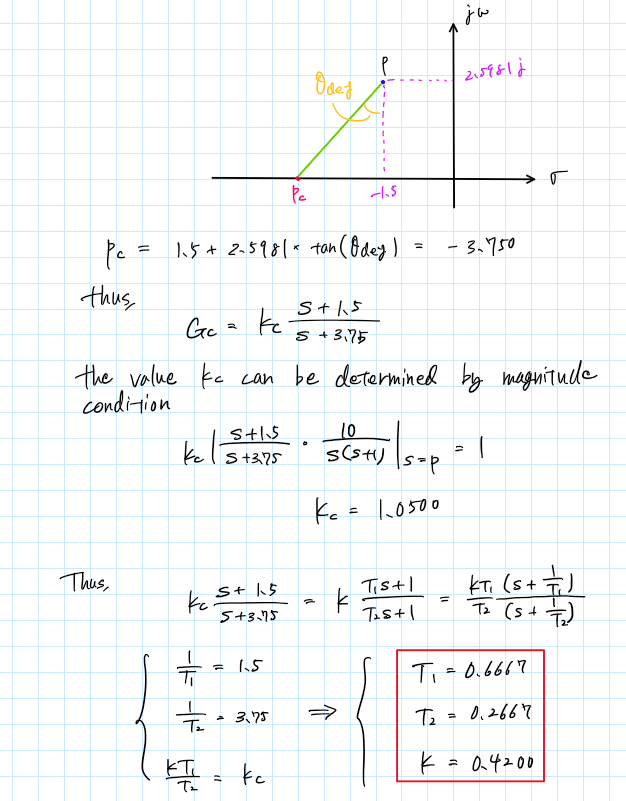


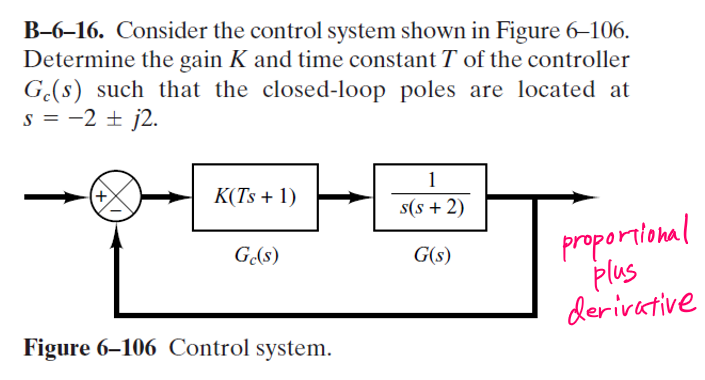


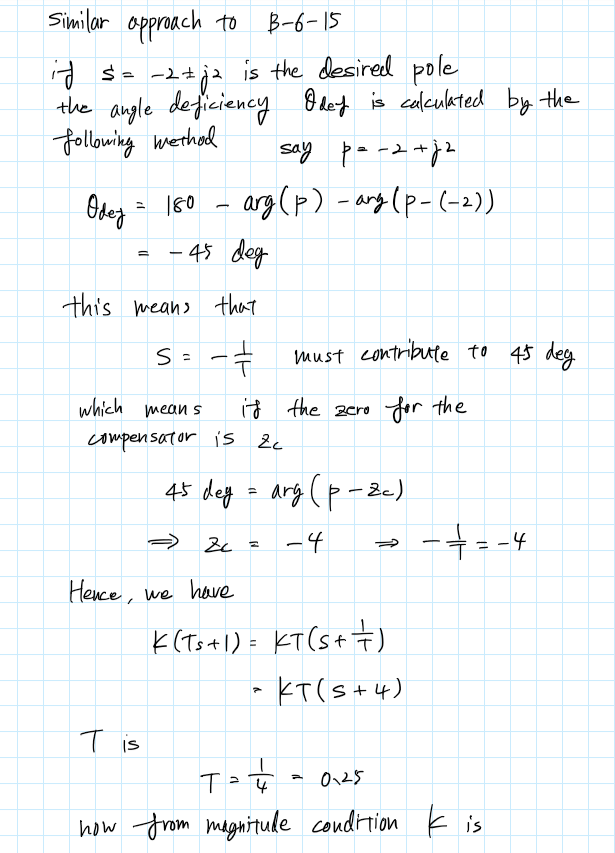


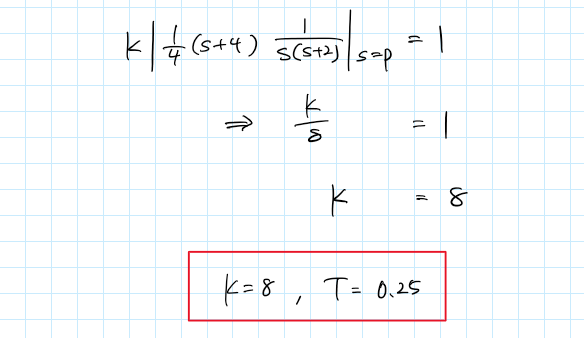


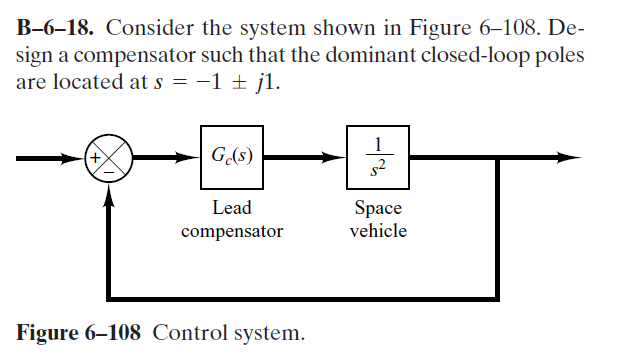


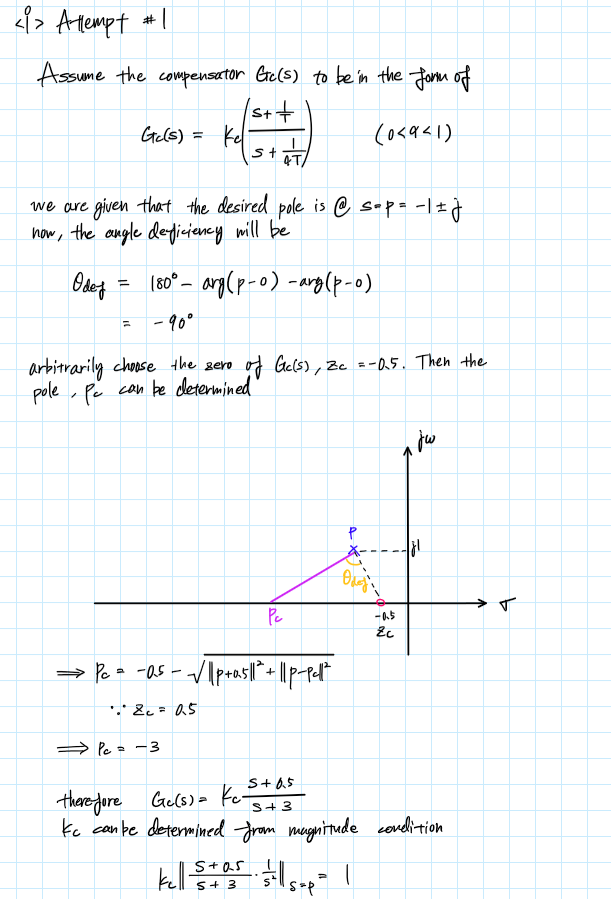


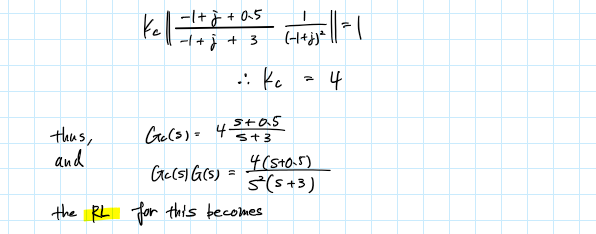


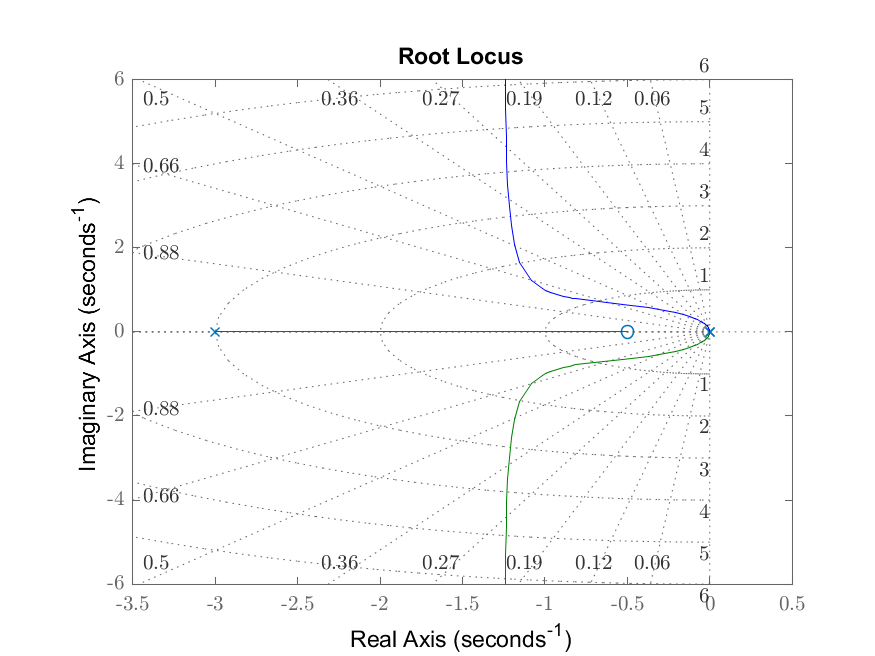


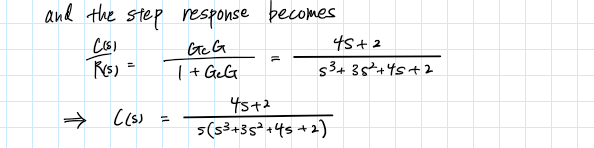


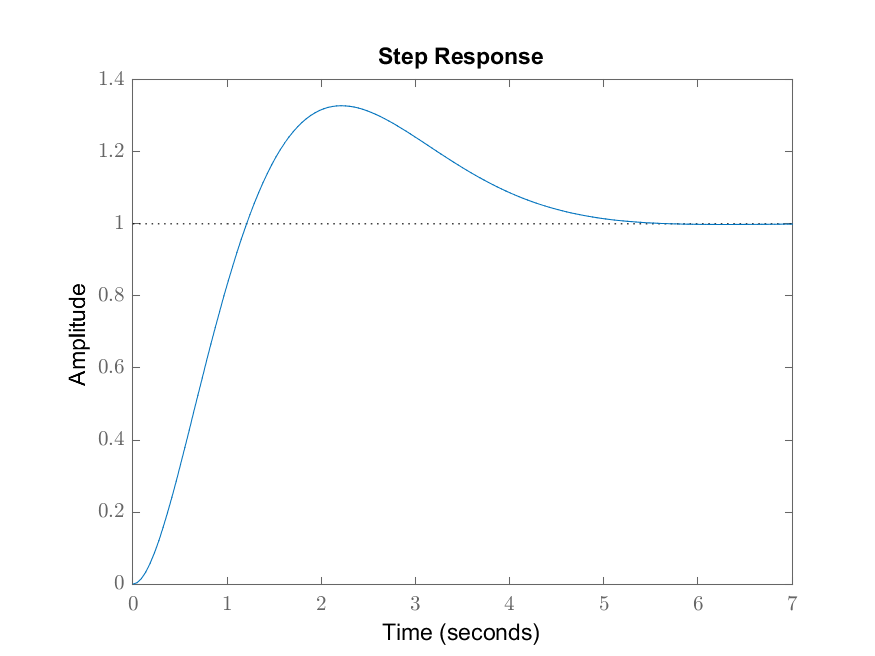


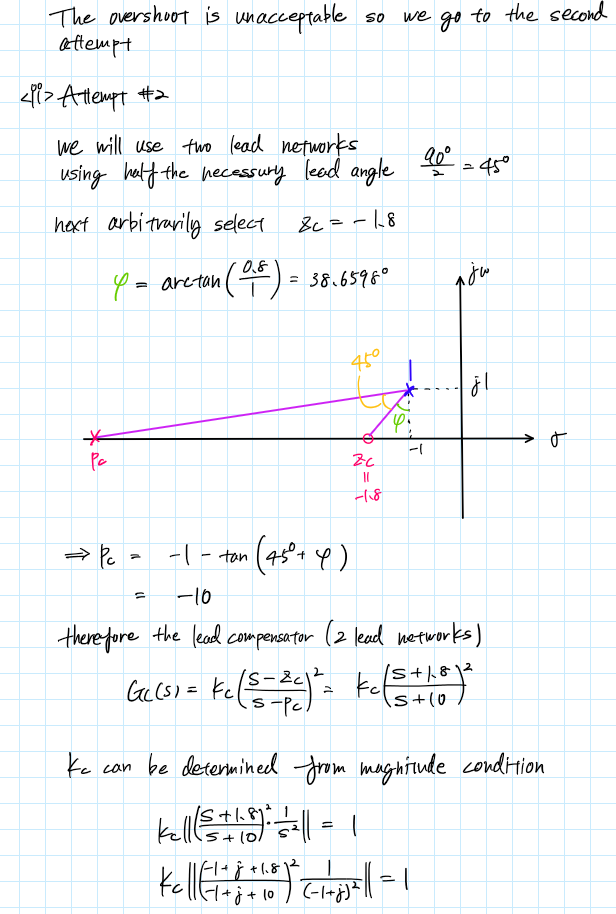


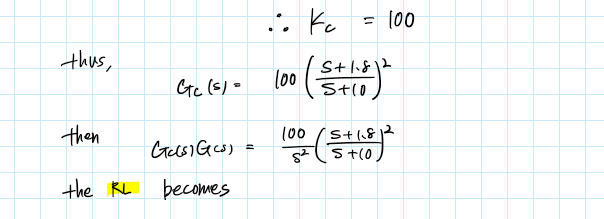


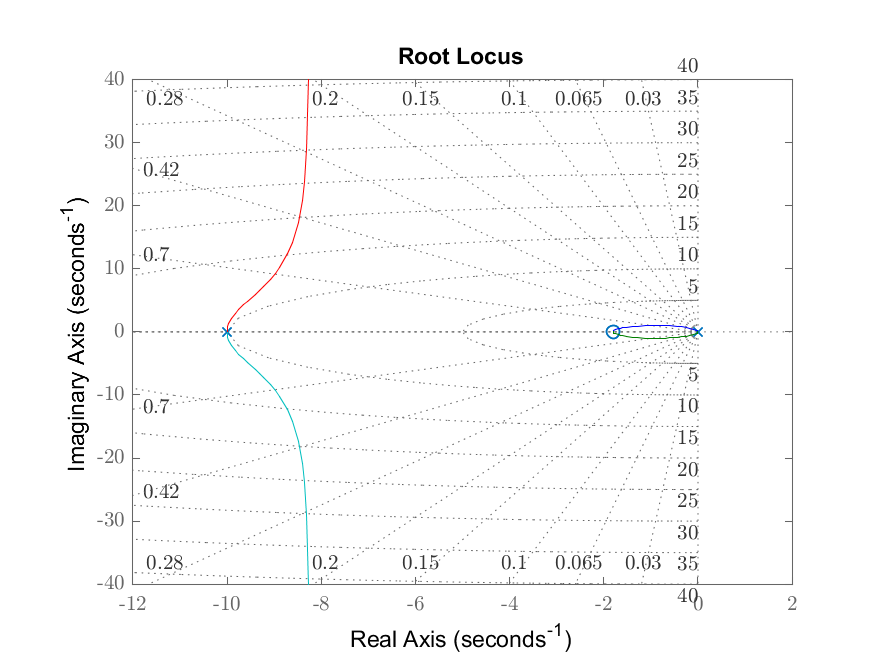


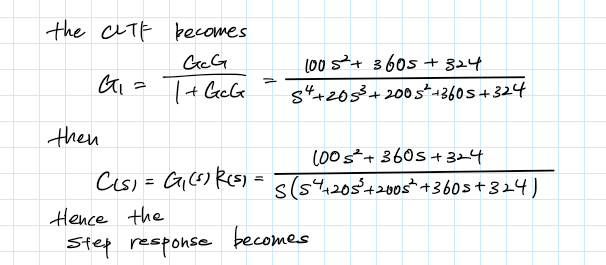


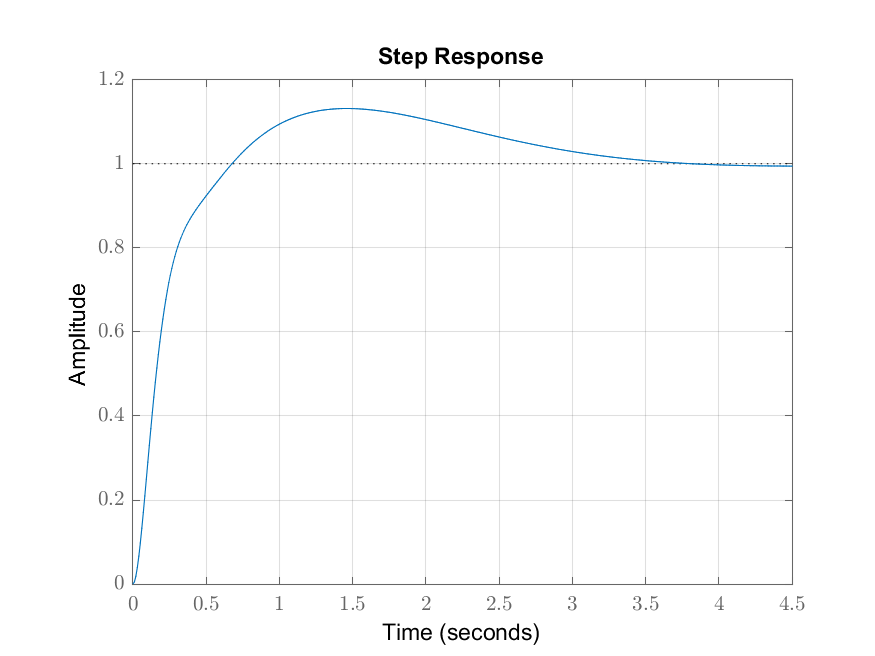


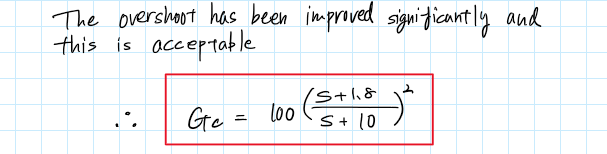


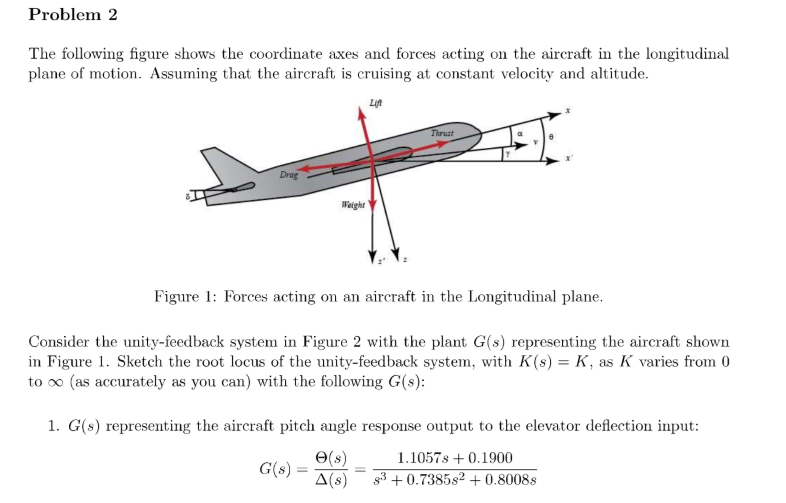


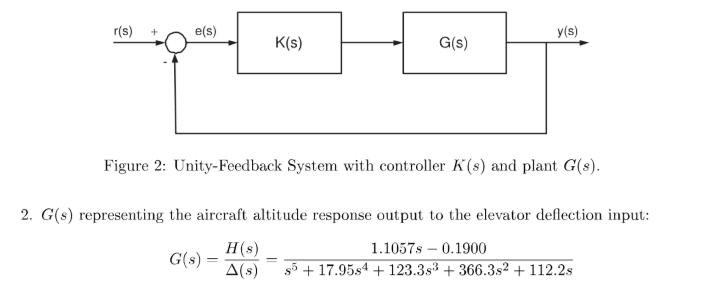


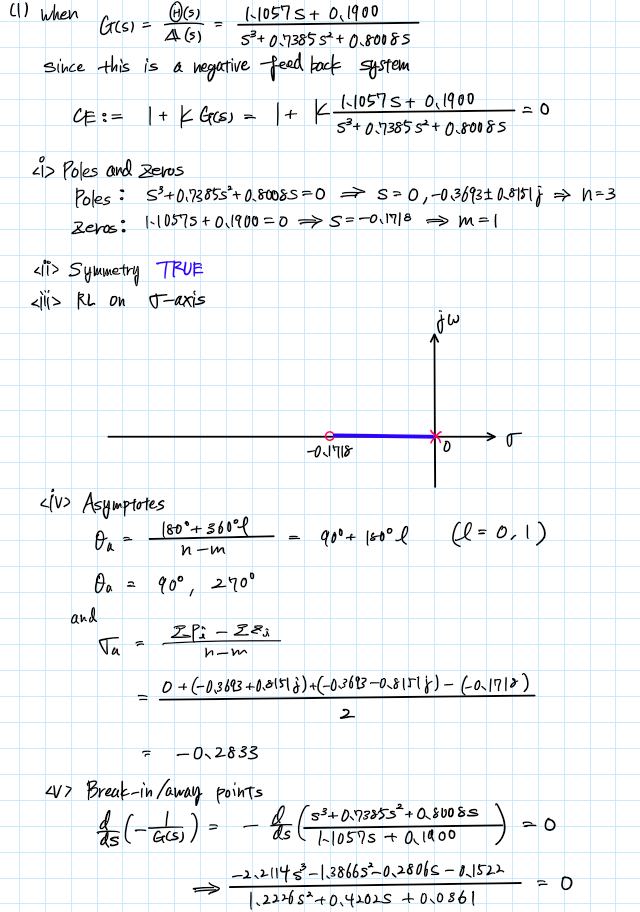


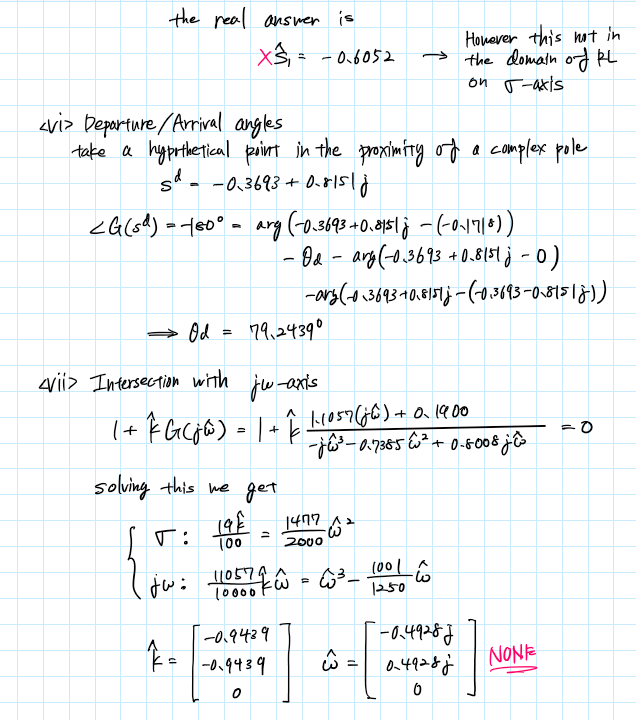


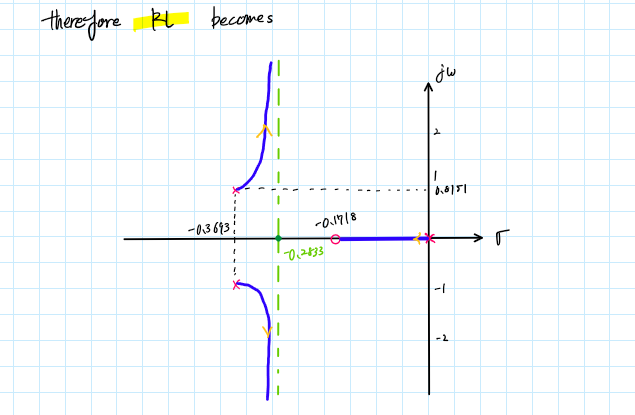


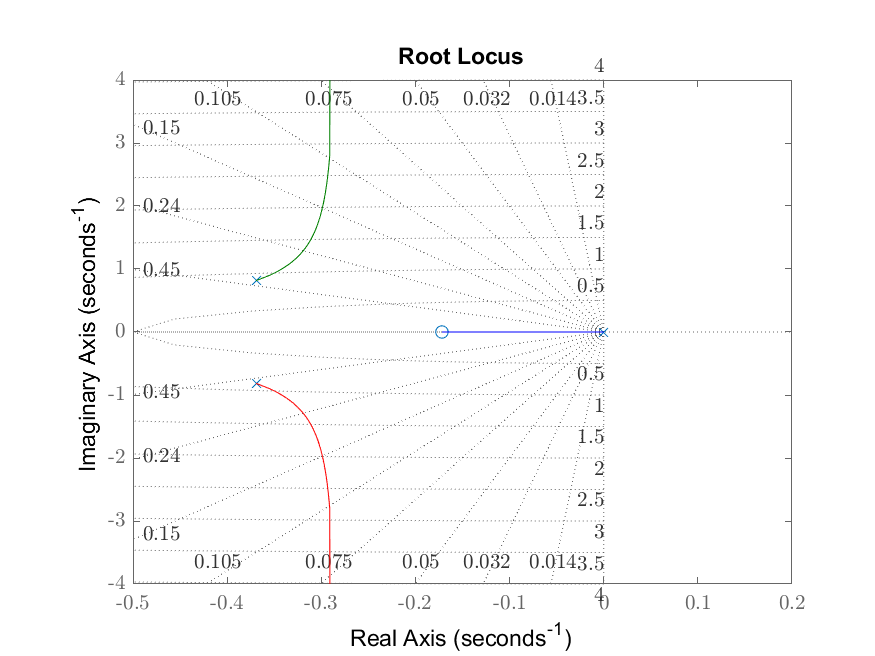


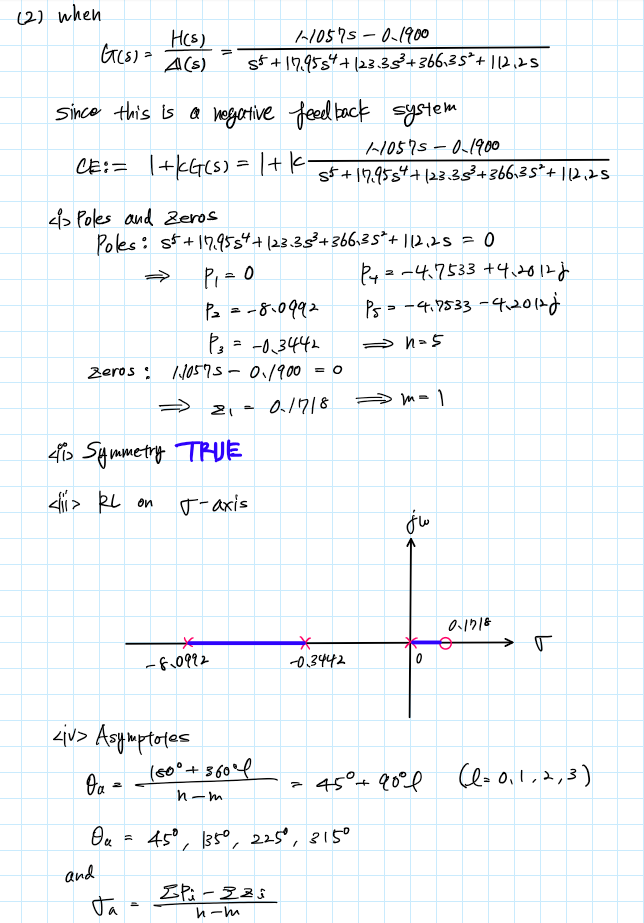


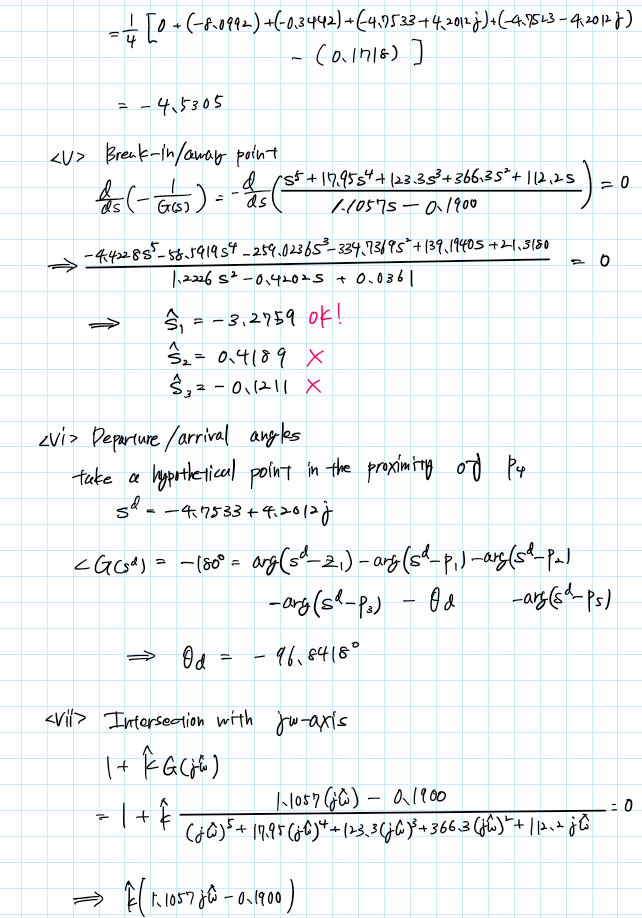


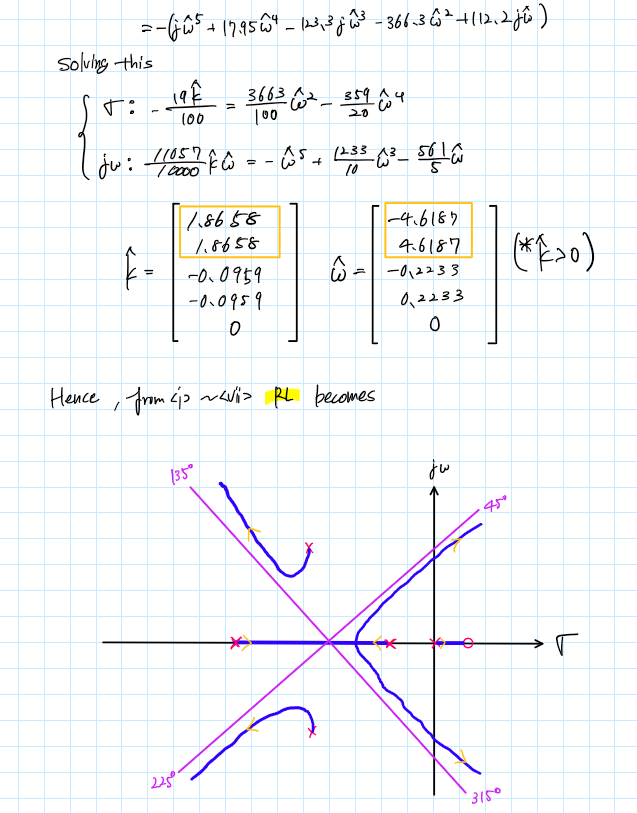


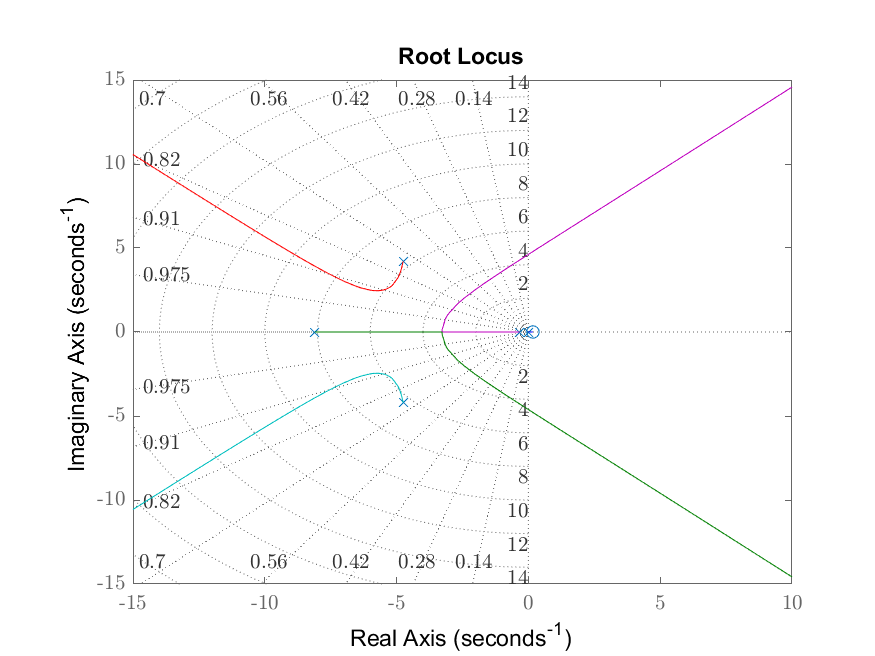












Appendix

**AAE364 HW8 MATLAB CODE**

clear all; close all; clc;

fdir = 'C:\Users\Tomo\Desktop\studies\2020-Spring\AAE364\matlab\matlab\_output\hw8';

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

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

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

**B-6-12**

% (a)

num = [1 -1];

den = conv([1 2],[1 4]);

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

saveas(fig1, fullfile(fdir,'RL\_B-6-12\_a.png'))

% (b)

num = -[1 -1];

den = conv([1 2],[1 4]);

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

saveas(fig1, fullfile(fdir,'RL\_B-6-12\_b.png'))

**B-6-14**

num = [0 1];

den = conv([1 0],[1 4 5]);

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

saveas(fig1, fullfile(fdir,'RL\_B-6-14.png'))

% Step Response

% Find gain, K when zeta = 0.5

% Analytical

syms K x w s

assume(K,'real');

assume(x,'real');

assume(w,'real');

p = -x + w\*1j

RHS = (s\*(s^2 + 4\*s + 5))

RHS = subs(RHS,s,p)

RHS = expand(RHS)

RHS = subs(RHS,w,tand(60)\*x)

eqn1 = K == -real(RHS)

eqn2 = 0 == -imag(RHS)

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

sigma = double(res.x)

w = tand(60)\*sigma

K\_th = double(res.K)

K\_th = nonzeros(K\_th)

% Verify (computational)

G = tf(num,den);

% controlSystemDesigner(G) % from this we find that the pole we need is

% the following variable opz to satisfy zeta = 0.5

opz = [-0.625+1.08i, -0.625-1.08i];

[K\_comp,pls] = rlocfind(G,opz)

% Plot step response

[numCL, denCL] = cloop((K\_th)\*num, den)

L\_inv\_expr = return\_inverseLaplace\_expression(numCL,conv(denCL,[1 0]))

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

step(numCL,denCL);

grid on; grid minor; box on;

saveas(fig2,fullfile(fdir,"STEP\_RES\_B-6-14.png"));

**B-6-15**

wn = 3; % natural frequency [rad/s]

zeta = 0.5; % damping ratio

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

% deficiency angle

theta\_def = pi + angle(10/p/(p + 1))

theta\_deg\_deg = rad2deg(theta\_def)

% find the pole and zero for compensator

pc = real(p) - tan(2\*pi - theta\_def)\*imag(p)

zc = real(p)

% find Kc

syms Kc

eqn = Kc\*abs((p+1.5)/(p+3.75) \* 10/(p\*(p+1))) == 1;

K\_c = double(solve(eqn,Kc))

% T1 T2 and K

T1 = 1/-zc

T2 = 1/-pc

K = K\_c\*T2/T1

**B-6-16**

% angle deficiency

p = -2+2j;

theta\_def = deg2rad(180) - angle(p) - angle(p + 2)

theta\_def\_deg = rad2deg(theta\_def)

% find zero

syms zc

assume(zc,'real')

eqn = -theta\_def == angle(p - zc)

Z\_c = double(solve(eqn,zc))

syms K

eqn = K\*0.25\*abs((p+4)/p/(p+2)) == 1

K\_p = double(solve(eqn,K))

**B-6-18**

% First attempt

% angle deficiency

p = -1+1j;

theta\_def = pi - angle(p - 0) - angle(p - 0)

theta\_def\_deg = rad2deg(theta\_def)

% find pc

syms pc

eqn = pc == -0.5 - sqrt((abs(p + 0.5))^2 + (abs(p - pc))^2);

P\_c = double(solve(eqn,pc))

% find Kc

syms Kc

eqn = Kc\*abs((p + 0.5)/(p + 3)/p^2) == 1;

Kc = double(solve(eqn,Kc))

% Plot RL

num = Kc\*[1 0.5];

den = conv([1 0 0],[1 3]);

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

rlocus(tf(num,den))

sgrid

saveas(fig1, fullfile(fdir,'RL\_B-6-18.png'))

% Step response

[numCL, denCL] = cloop(num,den)

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

step(numCL,denCL);

grid on; grid minor; box on;

saveas(fig2,fullfile(fdir,"STEP\_RES\_B-6-18.png"));

% Second attempt

zc = -1.8;

phi = atan(0.8)

phi\_deg = rad2deg(phi)

pc = -1 - tan(-theta\_def/2 + phi)

% find Kc

syms Kc

eqn = Kc\*abs(((p - zc)/(p - pc))^2/p^2) == 1;

Kc = double(solve(eqn,Kc))

% Plot RL

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

num = Kc\*conv([1 -zc],[1 -zc])

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

den = conv(den,[1 -pc])

rlocus(tf(num,den));

sgrid

saveas(fig1, fullfile(fdir,"RL\_B-6-18\_2.png"));

% Step response

[numCL, denCL] = cloop(num,den)

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

step(numCL,denCL);

grid on; grid minor; box on;

saveas(fig2,fullfile(fdir,"STEP\_RES\_B-6-18\_2.png"));

**P2**

% (1)

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)

saveas(fig1, fullfile(fdir,'RL\_P2\_1.png'))

% (2)

num = [1.1057 -0.1900];

den = [1 17.95 123.3 366.3 112.2 0];

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

saveas(fig1, fullfile(fdir,'RL\_P2\_2.png'))

function [poles,zrs,angs,sigma,bi\_pt,T\_P,T\_Z,k,w,fig1] = rootLocus\_stepBystep\_posFeedback(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 POSITIVE

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\_posFeedback(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\_posFeedback(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)

sgrid

end

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)

sgrid

end

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

n = length(poles);

m = length(zrs);

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

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

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

end

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

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 [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

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

%{

NAME: ROOTLOCUS\_STEPBYSTEP

AUTHOR: TOMOKI KOIKE

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

(2) poles: THE POLES OF THE TRANSFER FUNCTION

OUTPUTS: (1) theta: THE DEPARTURE ANGLES AND ARRIVAL ANGLES FOR THE

TRANSFER FUNCTION

DESCRIPTION: CALCULATES ALL THE DEPARTURE ANGLES AND ARRIVALS ANGLES

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

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(0); % [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(0) - 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 [table\_P, table\_Z] = departure\_arrival\_angle\_calc(zrs, poles)

%{

NAME: ROOTLOCUS\_STEPBYSTEP

AUTHOR: TOMOKI KOIKE

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

(2) poles: THE POLES OF THE TRANSFER FUNCTION

OUTPUTS: (1) theta: THE DEPARTURE ANGLES AND ARRIVAL ANGLES FOR THE

TRANSFER FUNCTION

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