College of Engineering
School of Aeronautics and Astronautics

AAE 421
Flight Dynamics and Controls

HW 4
Aircraft Classic Control Theory

Author:
Tomoki Koike

Supervisor:
Ran Dai

November 5th, 2020
Purdue University
West Lafayette, Indiana

Problem 1 (20 pts)

The longitudinal dynamics of a plane are described by

$$\begin{bmatrix}\Delta\dot{u}\\ \Delta\dot{w}\\ \Delta\dot{q}\\ \Delta\dot{\theta}\end{bmatrix} = \begin{bmatrix}X_u & X_w & 0 & -g\\ Z_u & Z_w & u_0 & 0\\ M_u & M_w & M_q & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix} + \begin{bmatrix}X_{\delta_e}\\ Z_{\delta_e}\\ M_{\delta_e}\\ 0\end{bmatrix}\Delta\delta_e\ .$$

The phugoid mode has the approximate 2-D dynamics described by:

$$\begin{bmatrix}\Delta\dot{u}\\ \Delta\dot{\theta}\end{bmatrix} = \begin{bmatrix}X_u & -g\\ -Z_u/u_0 & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta\theta\end{bmatrix} + \begin{bmatrix}X_{\delta_e}\\ Z_{\delta_e}/u_0\end{bmatrix}\Delta\delta_e$$

and the short period mode has the approximate 2-D dynamics described by:

$$\begin{bmatrix}\Delta\dot{w}\\ \Delta\dot{\theta}\end{bmatrix} = \begin{bmatrix}Z_w & u_0\\ M_w & M_q\end{bmatrix}\begin{bmatrix}\Delta w\\ \Delta q\end{bmatrix} + \begin{bmatrix}Z_{\delta_e}\\ M_{\delta_e}\end{bmatrix}\Delta\delta_e$$

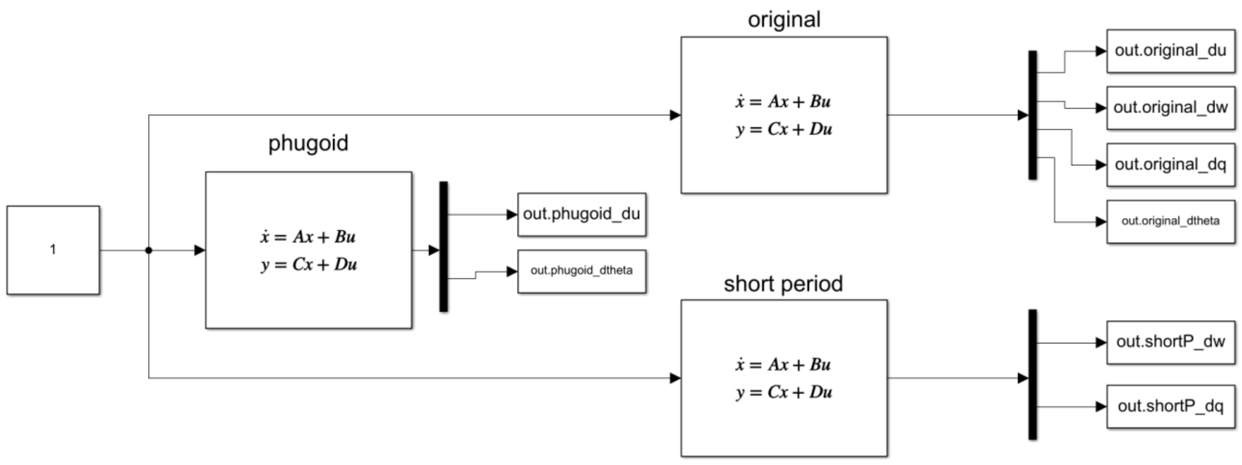
The relative parameters of a fixed-wing aircraft are listed in the following table.

X_u	X_w	Z_u	Z_w
$-0.045s^{-1}$	$-0.036s^{-1}$	$-0.369s^{-1}$	$-2.02s^{-1}$
M_u	M_w	M_q	u_0
0	$-0.05ft^{-1}s^{-1}$	$-2.05ft^{-1}s^{-1}$	$176ft/s$

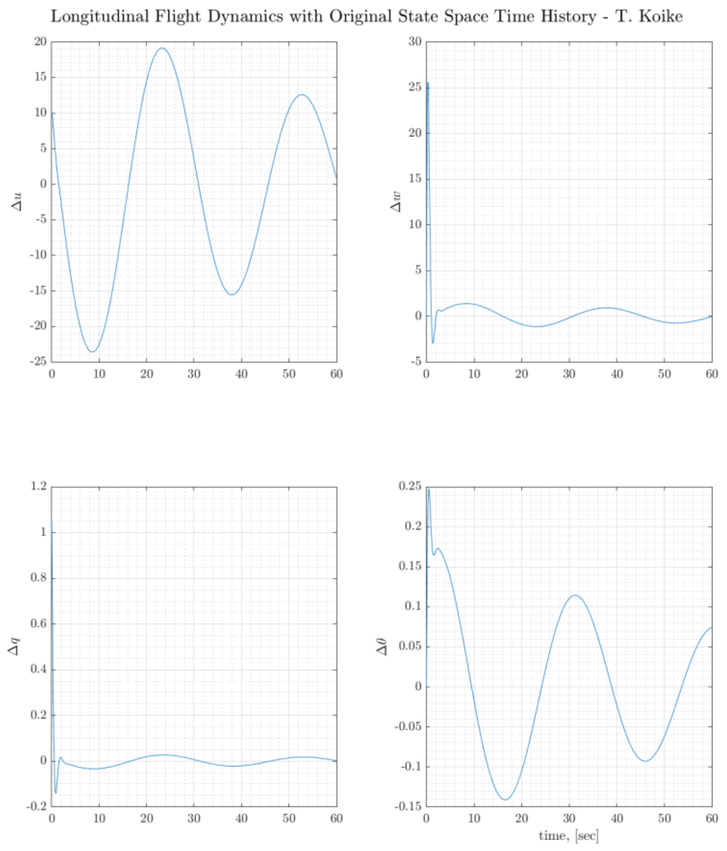
*For this problem assume a homogeneous system.

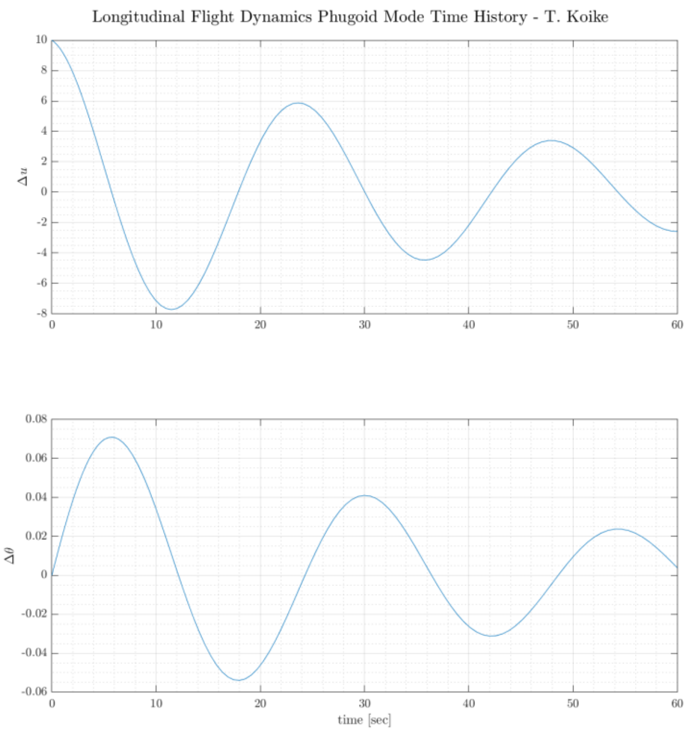
(1) Plot the state response using both the state space and Simulink methods for the original model and approximate model, respectively, assuming initial conditions $\bar{x}_0^{tr} = [10 \quad 0 \quad pi/3 \quad 0]^{tr}$.

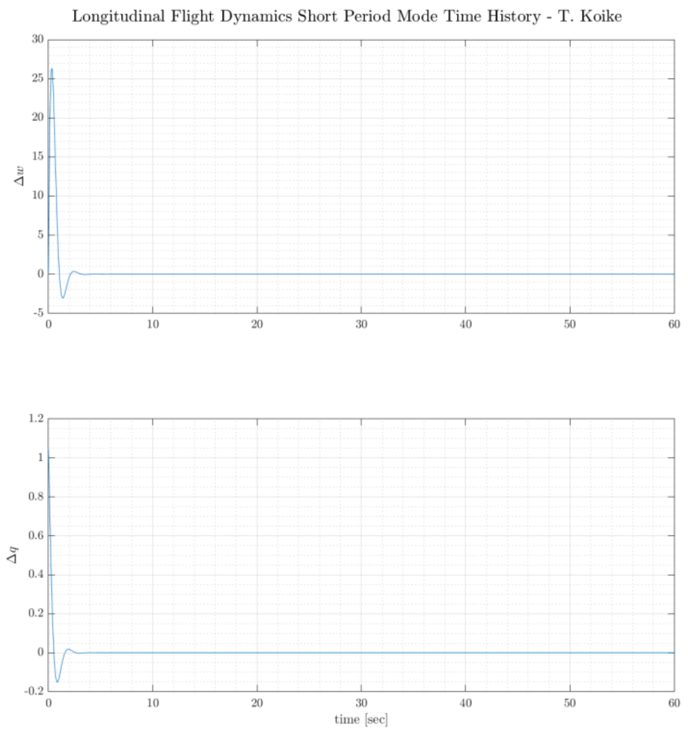
In Simulink we create the following model to simulate the original, phugoid, and short period mode systems,



The plots we obtain are the following




Longitudinal Flight Dynamics Phugoid Mode Time History - T. Koike
Δu
$\Delta\theta$
time [sec]


Longitudinal Flight Dynamics Short Period Mode Time History - T. Koike
Δw
Δq
time [sec]

MATLAB Code

```
% (1)

% Define the relative parameters
Xu = -0.045;  % s-1
Xw = -0.036;  % s-1
Zu = -0.369;  % s-1
Zw = -2.020;  % s-1
Mu =      0;  % ft-1s-1
Mw = -0.050;  % ft-1s-1
Mq = -2.050;  % ft-1s-1
u0 = 176.00;  % ft/s
g  = 32.170;  % ft/s2

% Define the original model A matrix
A = [Xu, Xw,  0, -g;
     Zu, Zw, u0,  0;
     Mu, Mw, Mq,  0;
      0,  0,  1,  0];

% Define the phugoid mode approximate 2-D dynamics A matrix
A_pg = [Xu, -g; -Zu/u0, 0];

% Define the short period mode approximate 2-D dynamics A matrix
A_sp = [Zw, u0; Mw, Mq];

% Define other matrices
X_de = 0;
Z_de = 0;
M_de = 0;
theta_de = 0;

B = [X_de, Z_de, M_de, theta_de]';
B_pg = [X_de, Z_de/u0]';
B_sp = [Z_de, M_de]';

C = eye(4);
C_pg = eye(2);
C_sp = eye(2);

D = zeros(4,1);
D_pg = zeros(2,1);
D_sp = zeros(2,1);

% Define the initial conditions
IC = [10; 0; pi/3; 0];
IC_pg = [10; 0];
IC_sp = [0; pi/3];

% Run the simulation
simout = sim('hw4_p1');

t = simout.tout;
du_org = simout.original_du.signals.values;
```

```
dw_org = simout.original_dw.signals.values;
dq_org = simout.original_dq.signals.values;
dtheta_org = simout.original_dtheta.signals.values;

du_phu = simout.phugoid_du.signals.values;
dtheta_phu = simout.phugoid_dtheta.signals.values;

dw_shp = simout.shortP_dw.signals.values;
dq_shp = simout.shortP_dq.signals.values;

% Plotting
% Original
fig1 = figure('Renderer',"painters", 'Position', [10 10 900 1000]);
subplot(2,2,1)
plot(t,du_org)
grid on; grid minor; box on;
ylabel('$\Delta u$')
subplot(2,2,2)
plot(t,dw_org)
grid on; grid minor; box on;
ylabel('$\Delta w$')
subplot(2,2,3)
plot(t,dq_org)
grid on; grid minor; box on;
ylabel('$\Delta q$')
subplot(2,2,4)
plot(t,dtheta_org)
grid on; grid minor; box on;
ylabel('$\Delta \theta$')
xlabel('time, [sec]')
sgtitle('Longitudinal Flight Dynamics with Original State Space Time History - T.
Koike')
saveas(fig1, fullfile(fdir, "p1_1_original.png"))

% Phugoid mode
fig2 = figure('Renderer',"painters", 'Position', [10 10 900 900]);
subplot(2,1,1)
plot(t,du_phu)
ylabel('$\Delta u$')
grid on; grid minor; box on;
subplot(2,1,2)
plot(t,dtheta_phu)
ylabel('$\Delta \theta$')
grid on; grid minor; box on;
sgtitle('Longitudinal Flight Dynamics Phugoid Mode Time History - T. Koike')
xlabel('time [sec]')
saveas(fig2, fullfile(fdir, "p2_1_phugoid.png"))

% Short peroid mode
fig2 = figure('Renderer',"painters", 'Position', [10 10 900 900]);
subplot(2,1,1)
plot(t,dw_shp)
ylabel('$\Delta w$')
grid on; grid minor; box on;
subplot(2,1,2)
```

```
plot(t,dq_shp)
ylabel('$\Delta q$')
grid on; grid minor; box on;
sgtitle('Longitudinal Flight Dynamics Short Period Mode Time History - T. Koike')
xlabel('time [sec]')
saveas(fig2, fullfile(fdir, "p2_1_shortPeriod.png"))
```

(2) Compute the % error of the approximation model to the original model in terms of the damped natural period $T_d = 2\pi/\omega_d$.

We know that

$$\omega_d = \omega\sqrt{1-\zeta^2} \ .$$

And we can obtain the natural frequency and damping ratio from the command "`damp()`" in MATLAB.

```
% Original
sys_org = ss(A, B, C, D)
[omega_n_org, zeta_org, p] = damp(sys_org)
omega_d_org = omega_n_org .* sqrt(1 - zeta_org.^2)
T_d_org = 2*pi / omega_d_org
```

This gives us the results of

```
omega_n_org = 4×1   zeta_org = 4×1   omega_d_org = 4×1   p = 4×1 complex
    0.2141              0.0663           0.2136             -0.0142 + 0.2136i
    0.2141              0.0663           0.2136             -0.0142 - 0.2136i
    3.5985              0.5678           2.9621             -2.0433 + 2.9621i
    3.5985              0.5678           2.9621             -2.0433 - 2.9621i
```

The 4th column, "p" indicates the corresponding poles, and the blue ones are for the phugoid and the red ones are for the short period in the original model.

Thus,

$$T_{4D}^{d}\big|_{phugoid} = \frac{2\pi}{0.2136} = 29.4142 \ s$$

$$T_{4D}^{d}\big|_{shortP} = \frac{2\pi}{2.9621} = 2.1212 \ s$$

Now for the approximate phugoid model

$$\omega_n = \sqrt{\frac{-Z_u g}{u_0}} = 0.2597\ rad/s$$

$$\zeta = \frac{-X_u}{2\omega_n} = 0.0866$$

$$\omega_d = 0.2587$$

```
% Phugoid mode
sys_phu = ss(A_pg, B_pg, C_pg, D_pg);
[omega_n_phu, zeta_phu, p] = damp(sys_phu)
omega_d_phu = omega_n_phu .* sqrt(1 - zeta_phu.^2)
T_d_phu = 2*pi / omega_d_phu(1)
error_phu = (T_d_phu - T_d_org_phu) / T_d_org_phu * 100  % percent error
```

```
omega_n_phu = 2×1
    0.2597
    0.2597
```

```
zeta_phu = 2×1
    0.0866
    0.0866
```

```
omega_d_phu = 2×1
    0.2587
    0.2587
```

```
p = 2×1 complex
  -0.0225 + 0.2587i
  -0.0225 - 0.2587i
```

$$T_{2D}^{d}|_{phugoid} = \frac{2\pi}{0.2587} = 24.2847\ s$$

$$\%error = \frac{24.2847 - 29.4142}{29.4142} \times 100 = -17.4338\,\%\ .$$

Now for the short period mode

$$-\omega_n\zeta = \frac{Z_w + M_q}{2}$$

$$\omega_n\sqrt{1-\zeta^2} = \sqrt{\left(Z_w + M_q\right)^2 - 4\left(M_q Z_w - M_w u_0\right)}$$

$$\therefore \omega_n = 3.5974\ rad/s \quad and \quad \zeta = 0.5657$$

$$\omega_d = 2.9664$$

```
% Short period mode
sys_shp = ss(A_sp, B_sp, C_sp, D_sp);
[omega_n_shp, zeta_shp, p] = damp(sys_shp)
omega_d_shp = omega_n_shp .* sqrt(1 - zeta_shp.^2)
T_d_shp = 2*pi / omega_d_shp(1)
error_shp = (T_d_shp - T_d_org_shp) / T_d_org_shp * 100  % percent error
```

omega_n_shp = 2×1 3.5974 3.5974	zeta_shp = 2×1 0.5657 0.5657	omega_d_shp = 2×1 2.9664 2.9664	p = 2×1 complex -2.0350 + 2.9664i -2.0350 - 2.9664i

$$T_{2D}^{d}|_{shortP} = \frac{2\pi}{2.9664} = 2.1181 \text{ s}$$

$$\%error = \frac{2.1181 - 2.1212}{2.1212} \times 100 = -0.1472\% \ .$$

(3) If you are allowed to change two parameters in the coefficient matrices, what will you choose and how will you change them to decrease the damped natural period?

Using the global optimization toolbox in MATLAB we can optimize the A matrix to minimize the percent error of both phugoid and short period mode. The code is

```
A0 = A;
lb = -200; ub = 200;
J = []; b = [];
Aeq = []; beq = [];
objfunc = @(A) objFunc(A, B, B_pg, B_sp, C, C_pg, C_sp, D, D_pg, D_sp);
[Aopt, fval] = patternsearch(objfunc,A0,J,b,Aeq,beq,lb,ub);

function res = objFunc(A, B, B_pg, B_sp, C, C_pg, C_sp, D, D_pg, D_sp)
    A_pg = [A(1,1), A(1,4); -A(2,1)/A(2,3), 0];
    A_sp = [A(2,2), A(2,3); A(3,2), A(3,3)];

    sys_org = ss(A, B, C, D);
    [omega_n_org, zeta_org, p] = damp(sys_org);
    omega_d_org = omega_n_org .* sqrt(1 - zeta_org.^2);
    T_d_org_phu = 2*pi / omega_d_org(1);
    T_d_org_shp = 2*pi / omega_d_org(3);

    sys_phu = ss(A_pg, B_pg, C_pg, D_pg);
    [omega_n_phu, zeta_phu, p] = damp(sys_phu);
    omega_d_phu = omega_n_phu .* sqrt(1 - zeta_phu.^2);
    T_d_phu = 2*pi / omega_d_phu(1);

    sys_shp = ss(A_sp, B_sp, C_sp, D_sp);
    [omega_n_shp, zeta_shp, p] = damp(sys_shp);
    omega_d_shp = omega_n_shp .* sqrt(1 - zeta_shp.^2);
    T_d_shp = 2*pi / omega_d_shp(1);

    res = abs(T_d_org_phu - T_d_phu) + abs(T_d_org_shp - T_d_shp);
end
```

The results are

```
Aopt = 4×4
    -0.0450   -0.0360   12.0000  -30.5894
    -0.3690   -0.0200  162.2097         0
          0   -0.0500   -2.0500         0
          0         0    1.0000         0
```

```
fval = 0.0055
```

"fval" is the value of

$$fval \coloneqq abs\left(T^d_{4D}|_{phugoid} - T^d_{2D}|_{phugoid}\right) + abs\left(T^d_{4D}|_{shortP} - T^d_{2D}|_{shortP}\right)$$

Thus, the values are

$\boldsymbol{X_u}$	$\boldsymbol{X_w}$	$\boldsymbol{Z_u}$	$\boldsymbol{Z_w}$
$\boldsymbol{-0.045\ s^{-1}}$	$-0.036\ s^{-1}$	$-0.369\ s^{-1}$	$-0.020\ s^{-1}$
$\boldsymbol{M_u}$	$\boldsymbol{M_w}$	$\boldsymbol{M_q}$	$\boldsymbol{u_0}$
0	$-0.05\ ft^{-1}s^{-1}$	$-2.05\ ft^{-1}s^{-1}$	$162.2097\ ft/s$

From this optimization the values of Z_u, u_0 had a significant effect to reduce the percent error of the damped natural period. Hence, we will conduct another optimization, but this time we will not optimize the entire A matrix and will only optimize the values Z_u and u_0.

The MATLAB Code is

```
function res = newObjFunc(input, data, B, B_pg, B_sp, C, C_pg, C_sp, D, D_pg, D_sp,
g)
    Zw = input(1); u0 = input(2);
    Xu = data(1); Xw = data(2); Zu = data(3); Mu = data(4);
    Mw = data(5); Mq = data(6);

    A = [Xu, Xw,  0, -g;
         Zu, Zw, u0,  0;
         Mu, Mw, Mq,  0;
          0,  0,  1,  0];
    A_pg = [A(1,1), A(1,4); -A(2,1)/A(2,3), 0];
    A_sp = [A(2,2), A(2,3); A(3,2), A(3,3)];

    sys_org = ss(A, B, C, D);
    [omega_n_org, zeta_org, p] = damp(sys_org);
    omega_d_org = omega_n_org .* sqrt(1 - zeta_org.^2);
    T_d_org_phu = 2*pi / omega_d_org(1);
    T_d_org_shp = 2*pi / omega_d_org(3);

    sys_phu = ss(A_pg, B_pg, C_pg, D_pg);
    [omega_n_phu, zeta_phu, p] = damp(sys_phu);
    omega_d_phu = omega_n_phu .* sqrt(1 - zeta_phu.^2);
    T_d_phu = 2*pi / omega_d_phu(1);

    sys_shp = ss(A_sp, B_sp, C_sp, D_sp);
    [omega_n_shp, zeta_shp, p] = damp(sys_shp);
    omega_d_shp = omega_n_shp .* sqrt(1 - zeta_shp.^2);
    T_d_shp = 2*pi / omega_d_shp(1);

    res = abs(T_d_org_phu - T_d_phu) + abs(T_d_org_shp - T_d_shp);
end
```

```
otherInputs = [Xu, Xw, Zu, Mu, Mw, Mq];
input0 = [Zw, u0];
lb = -200; ub = 200;
J = []; b = [];
Aeq = []; beq = [];
objfunc = @(input) newObjFunc(input, otherInputs, B, B_pg, B_sp, C, C_pg, C_sp, D,
D_pg, D_sp, g);
[Aopt, fval] = patternsearch(objfunc,input0,J,b,Aeq,beq,lb,ub);
```

The results for this optimization are

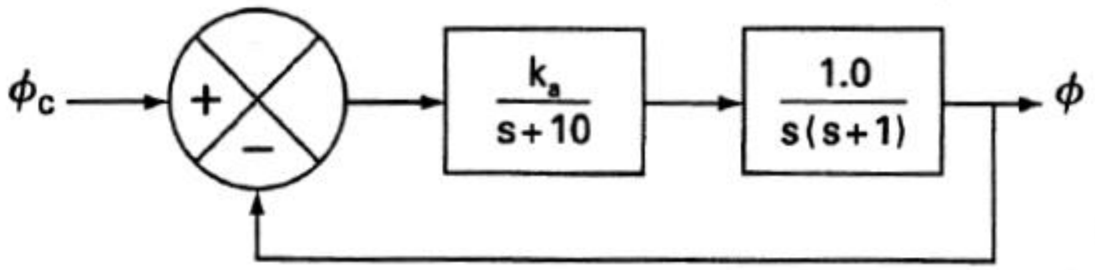
Zw_opt = -0.0825	u0_opt = 2.0695e+03
fval = 5.6407e-05	

The “fval” indicates the same value as the previous optimization, and we can see that the magnitude of Z_w is reduced and the magnitude of u_0 is increased to minimize the error.

Hence, it is a best practice to reduce the magnitude of Z_u, and increase the magnitude of u_0 to reduce the error of the damped natural period.

Problem 2

A roll control system is shown below.



(1) Sketch the root locus diagram for this system

This is a negative feedback system. The CE (characteristic equation) becomes

$$CE \coloneqq 1 + k_a\left(\frac{1}{s+10}\right)\left(\frac{1.0}{s(s+1)}\right) = 1 + k_a\frac{1.0}{s(s+1)(s+10)}$$

where

$$L(s) = \frac{1.0}{s(s+1)(s+10)} \ .$$

(i) Poles of $L(s)$:

$$p = 0, -1, -10 \quad \therefore\ n = 3 \ .$$

(ii) Zeros of $L(s)$:

$$z = none \quad \therefore m = 0 \ .$$

There are 3 branches, and all 3 go to ∞.

(iii) Asymptotes:

$$\theta_a = \frac{180° + 360° l}{n-m} = \frac{180° + 360° l}{3} = 60° + 120° l \quad where\ l = 0, 1, 2$$

$$\therefore \theta_a = 60°, 180°, 300° \ .$$

And

$$\sigma_a = \frac{\sum p_i - \sum z_i}{n-m} = \frac{0-1-10}{3} = \frac{-11}{3} = -3.6667 \ .$$

(iv) Break-in/away points:

$$\frac{d}{ds}\left(-\frac{1}{L(s)}\right) = -\frac{d}{ds}\big(s(s+1)(s+10)\big) = 0$$

$$\Rightarrow \ \frac{d}{ds}(s^3 + 11s^2 + 10s) = 0$$

$$\Rightarrow \ 3s^2 + 22s + 10 = 0$$

$$\therefore s \ = \ -6.8465, -0.4869\ .$$

Since,

$$-1 \ < \ -0.4869 \ < 0$$

the break away point is only

$$s \ = \ -0.4869 \ \ .$$

(v) Angle of departure:

$$s = 0 \Rightarrow \theta_d = 180° - arg\big(L(0)\big) = 180° - arg\left(\frac{1}{0}\right) = 180°.$$

$$s = -1 \Rightarrow \theta_d = 180° - \arg\big(L(-1)\big) = \theta_d = 180° - arg\left(\frac{1}{-0}\right) = 0°.$$

$$s = -10 \Rightarrow \theta_d = 180° - \arg\big(L(-10)\big) = \theta_d = 180° - arg\left(\frac{1}{0}\right) = 180°.$$

(vi) Intersection of Root Locus with the imaginary axis:

$$1 + \hat{k}L(j\widehat{\omega}) = 0 \quad \Rightarrow \quad 1 + \hat{k}\frac{1}{j\widehat{\omega}(j\widehat{\omega}+1)(j\widehat{\omega}+10)} = 0$$

This gives us

$$Real: \qquad \hat{k} = 11\widehat{\omega}^2$$

$$Imag: \quad 0 = \widehat{\omega}^3 - 10\widehat{\omega}$$

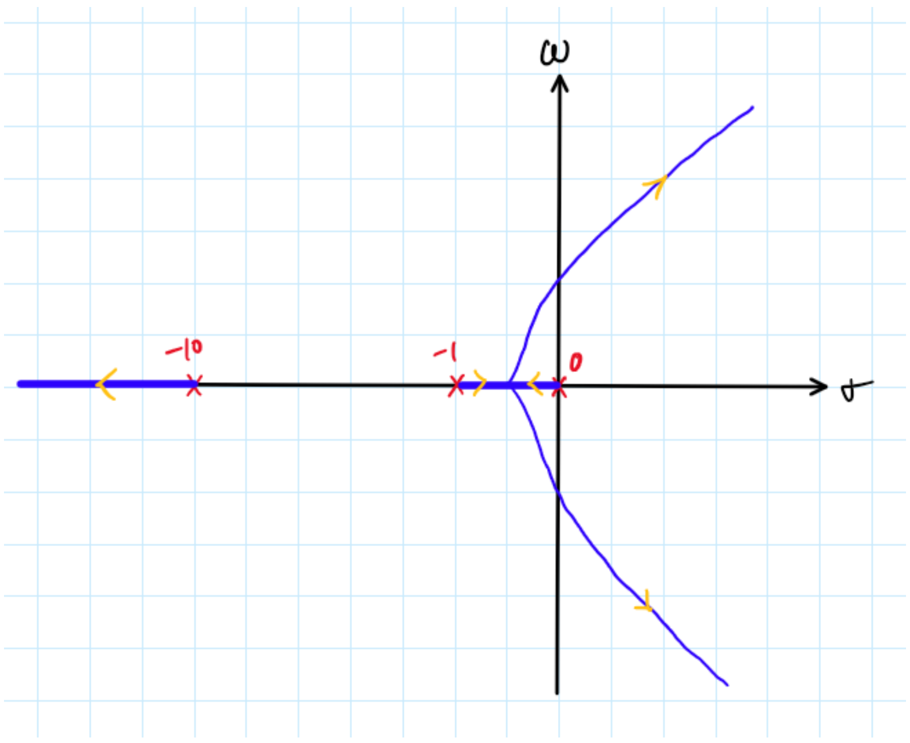
Thus,

$$\hat{k} = \begin{pmatrix} 0 \\ 110 \\ 110 \end{pmatrix}, \qquad \widehat{\omega} = \begin{pmatrix} 0 \\ -3.1623 \\ 3.1623 \end{pmatrix}.$$

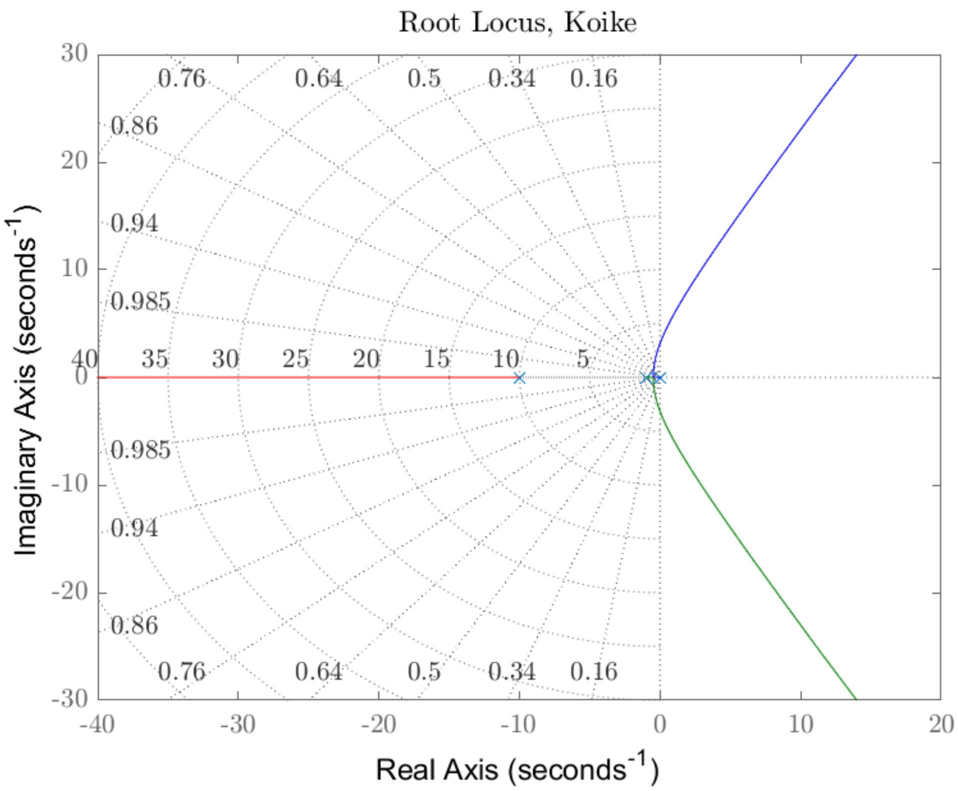
So the intersection of the Root Locus with the imaginary axis is

$$s \ = \ \pm j3.1623 \ \ .$$

With all this information we can draw the Root Locus



The MATLAB RL plot is



(2) Determine the value of the gain, K_a, so that control system has a damping ratio, $\zeta = 0.707$.

With this damping ratio we know that

$$s = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} = -0.707\omega_n \pm 0.707j\omega_n \; .$$

So, we solve this to get the gain and natural frequency corresponding to the damping ratio that we want

$$1 + KL(-0.707\omega_n + 0.707\omega_n) = 0$$

$$\Rightarrow 707\,\omega_n + {\omega_n}^3\,(-26.8060 - 65.3098\,j)$$
$$= {\omega_n}^2\,(227.5339 - 841.9298\,j) + 541.2947\,\omega_n\,j + 100\,k$$

The real part is

$$707\,\omega_n - 26.8060\,{\omega_n}^3 = 227.5339\,{\omega_n}^2 + 100\,k$$

The imaginary part is

$$-65.3098\,{\omega_n}^3 = 541.2947\,\omega_n - 841.9298\,{\omega_n}^2$$

Solving this with MATLAB

```
syms sigma omega k omega_n
assume([sigma, omega, k, omega_n], {'real', 'positive'});
s = sigma + omega*1i;
eqn = k == -(s^3 + 11*s^2 + 10*s);
zeta = 0.707;
eqn = subs(eqn, sigma, -zeta*omega_n);
eqn = simplify(subs(eqn, omega, omega_n*sqrt(1-zeta^2)))
eqn_LHS = lhs(eqn);
eqn_LHS_real = real(eqn_LHS);
eqn_LHS_imag = imag(eqn_LHS);
eqn_RHS = rhs(eqn);
eqn_RHS_real = real(eqn_RHS);
eqn_RHS_imag = imag(eqn_RHS);
eqn_real = eqn_LHS_real == eqn_RHS_real;
eqn_imag = eqn_LHS_imag == eqn_RHS_imag;
res1 = solve(eqn_imag, omega_n);
eqn_real = subs(eqn_real, omega_n, res1(1));
res2 = solve(eqn_real, k)
```

we get

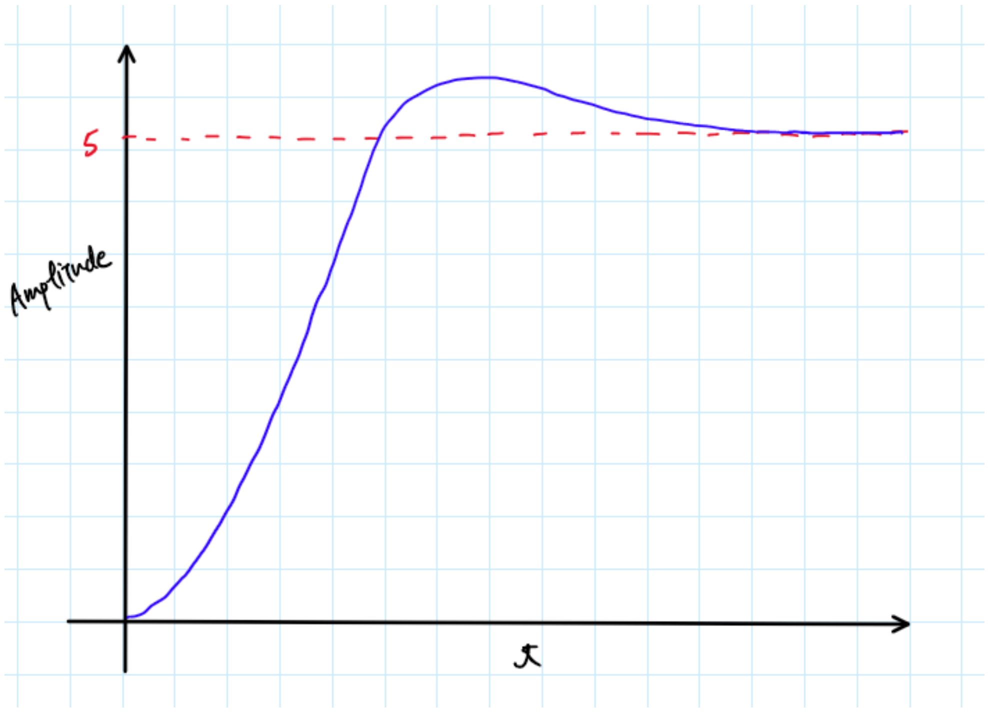
$$\omega_n = 0.6719, \qquad K = 4.5374$$

(3) What is the steady-state error for a step and ramp input with the designed system?

$$e_{ss} = \frac{1}{1+\bar{K}_p} = \frac{1}{1+\lim_{s\to 0} G(s)} = \frac{1}{\infty} = 0 \ .$$

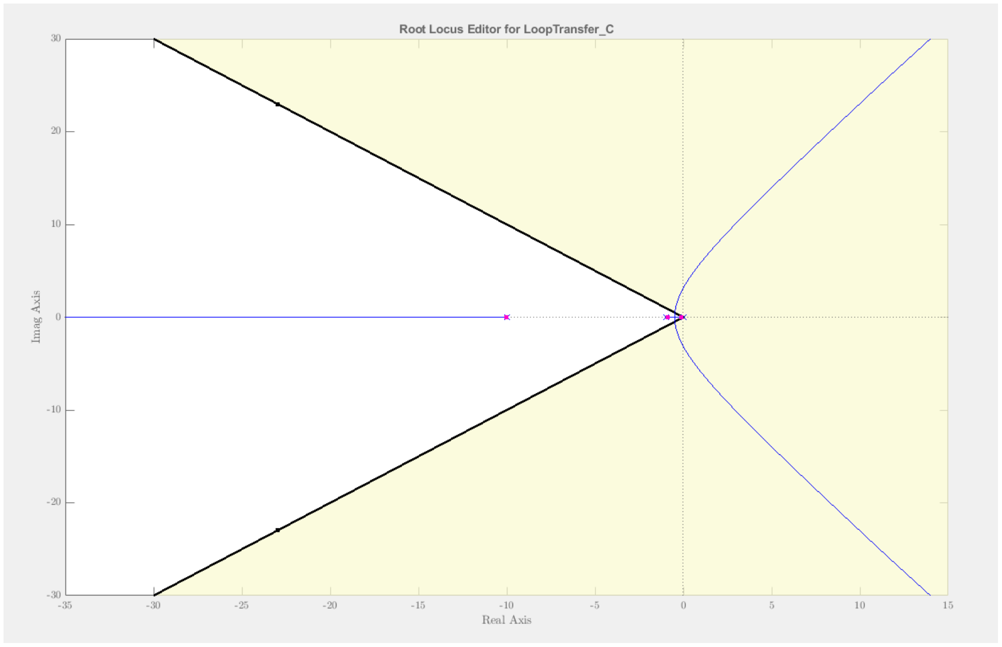
This agrees with the plot above.

(4) Sketch the response of the control system to a 5° step change in bank angle command.

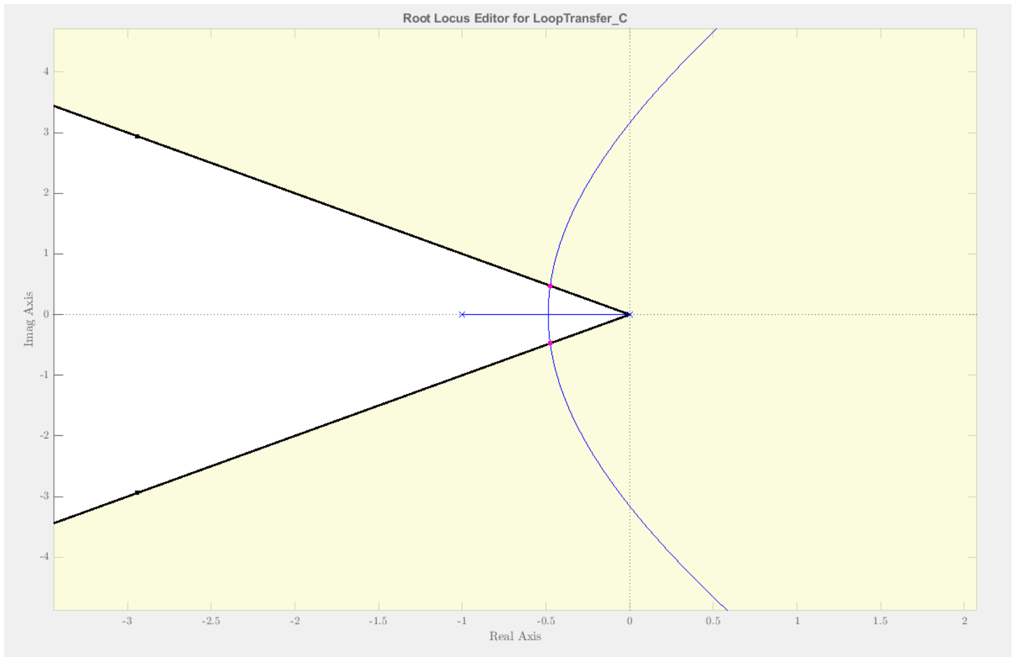


(5) Repeat this problem using control design software in MATLAB

Using the Control System Designer with the open loop transfer function, we can find the gain corresponding to the required damping ratio.



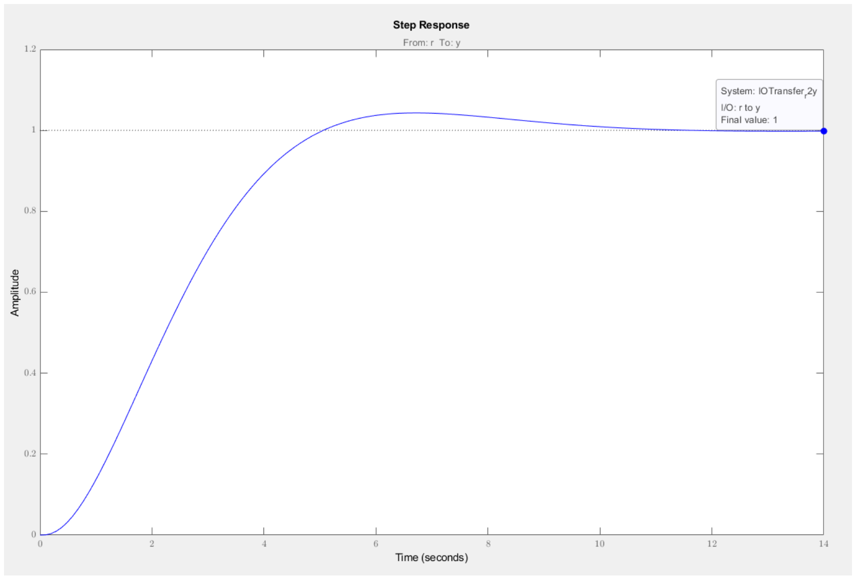
We move the poles to get the gains we want



Thus, we get the gain of

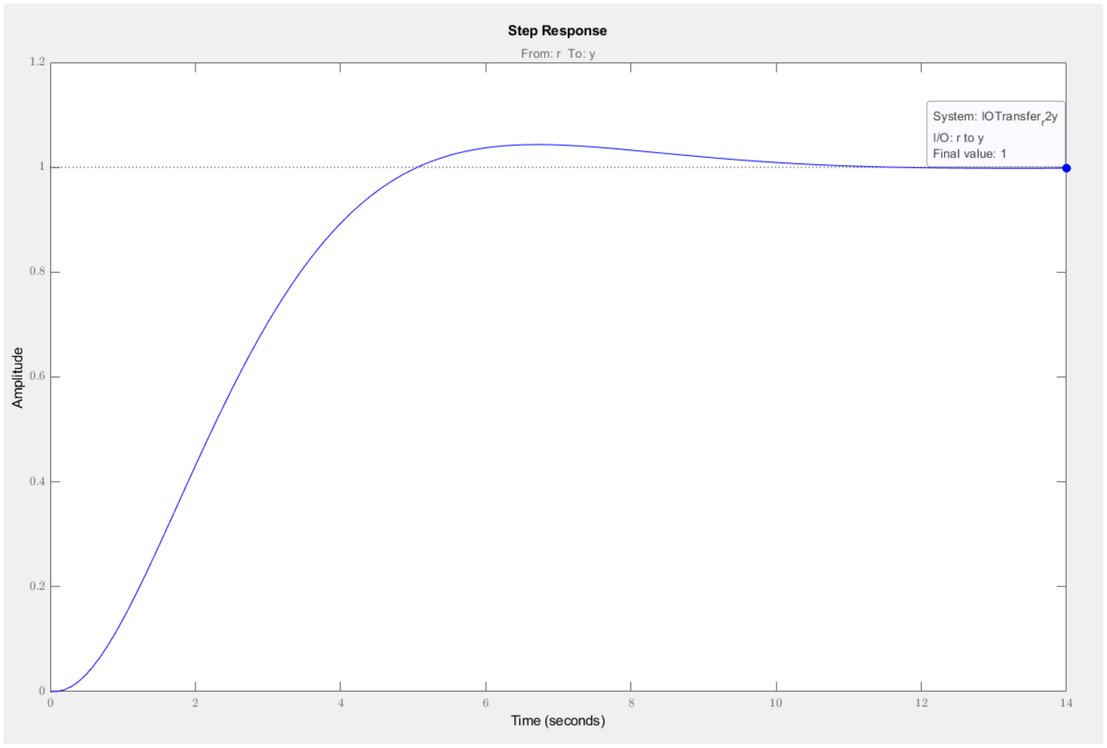
$$K = 4.5408 \ .$$

The step response plot in the Control System Designer is



$$e_{ss} = 0$$

For a step input of 5°,



MATLAB Code

Execution Code

```
% (1)

num = 1;
den = conv([1,0],[1, 1]);
den = conv(den, [1, 10]);
[poles,zrs,angs,sigma,bi_pt,T_P,T_Z,k,w,fig1] = rootLocus_stepBystep_negFeedback(num,
den);
saveas(fig1, fullfile(fdir, "p2_RL.png"));
```

```
% (5)

sys = tf(num, den);
controlSystemDesigner(sys);
```

Function “rootLocus_stepBystep_negFeedback()”

```
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 “RL_asymptote()”

```
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 “break_in_away_pt()”

```
function rts = break_in_away_pt(num,den)
    [q, d] = polyder(-den,num);
    rts = roots(q);
    rts = rts(rts==real(rts));
end
```

Function “departure_arrival_angle_calc()”

```
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 "intersection_IM_axis()"

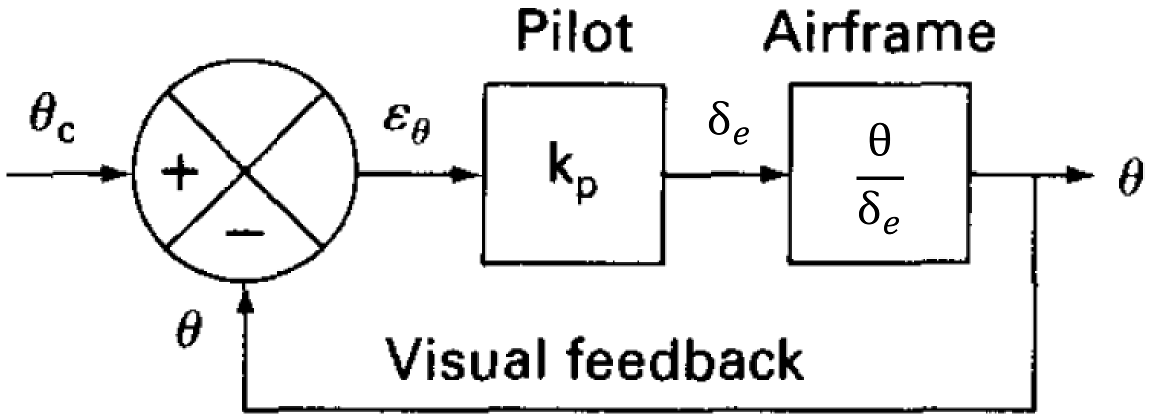
```
function [K, W] = intersection_IM_axis(num, den)
    syms k w
    n = length(den);
    m = length(num);
    f1 = 0; f2 = 0; p1 = 0; p2 = 0;

    % 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
```

Problem 3 (20 pts)

The Wright Flyer was statically and dynamically unstable. However, because the Wright brothers incorporated sufficient control authority into their design they were able to fly their airplane successfully. Although the airplane was difficult to fly, the combination of the pilot and airplane could be a stable system. The closed-loop pilot is represented as a pure gain, K_p, and the pitch attitude to canard deflection is given as follows:

$$\frac{\theta}{\delta_e} = \frac{11.0(s+0.5)(s+3.0)}{(s^2+0.72s+1.44)(s^2+5.9s-11.9)}$$

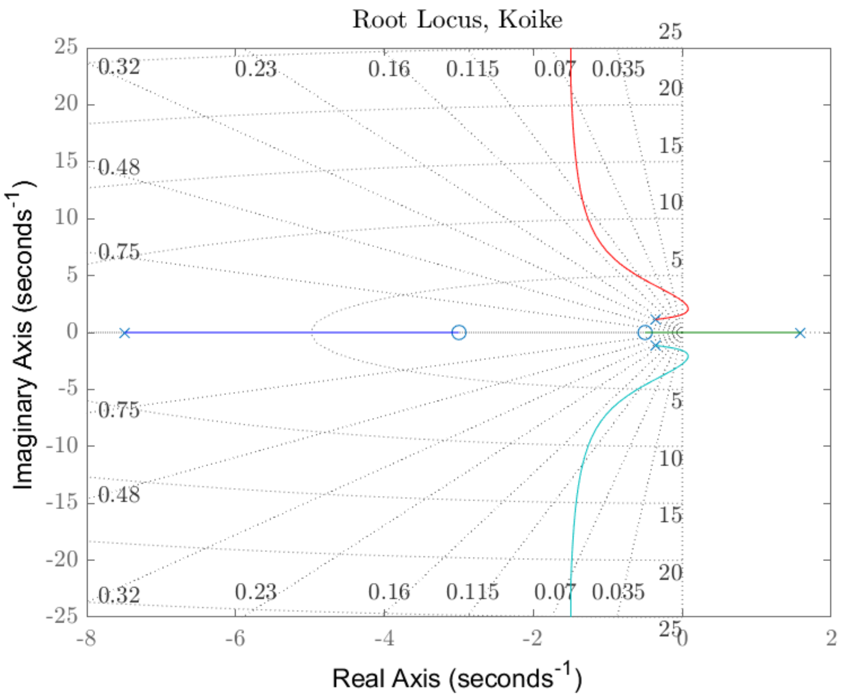
Determine the root locus plot of the closed loop system show below. For what range of pilot gain is the system stable?



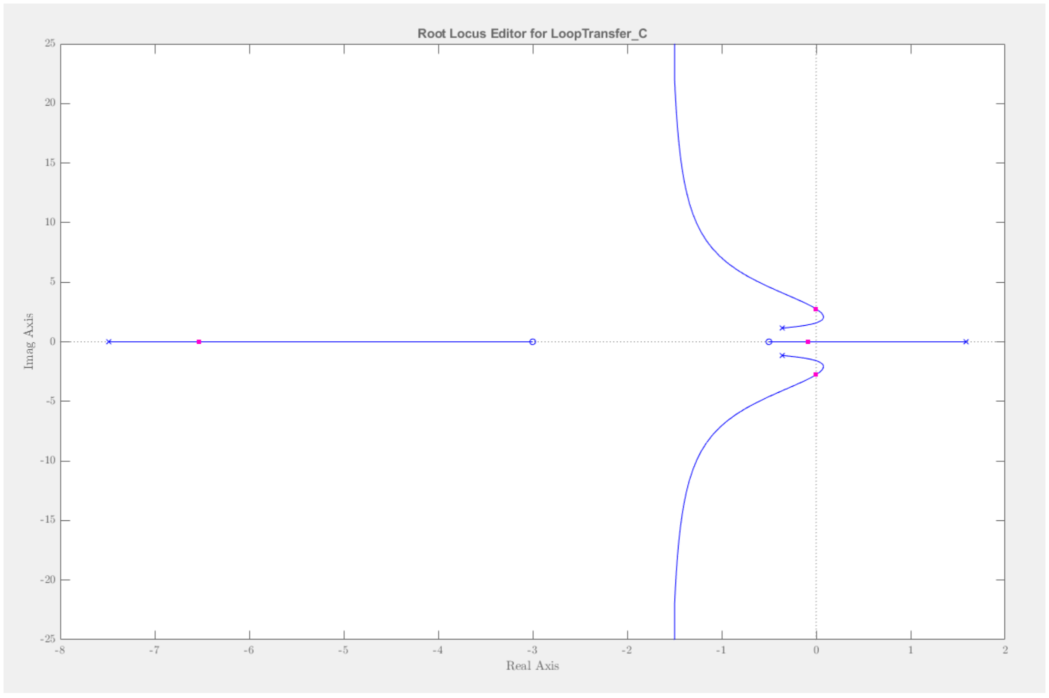
Using MATLAB, we are able to plot the Root Locus. The code is the following.

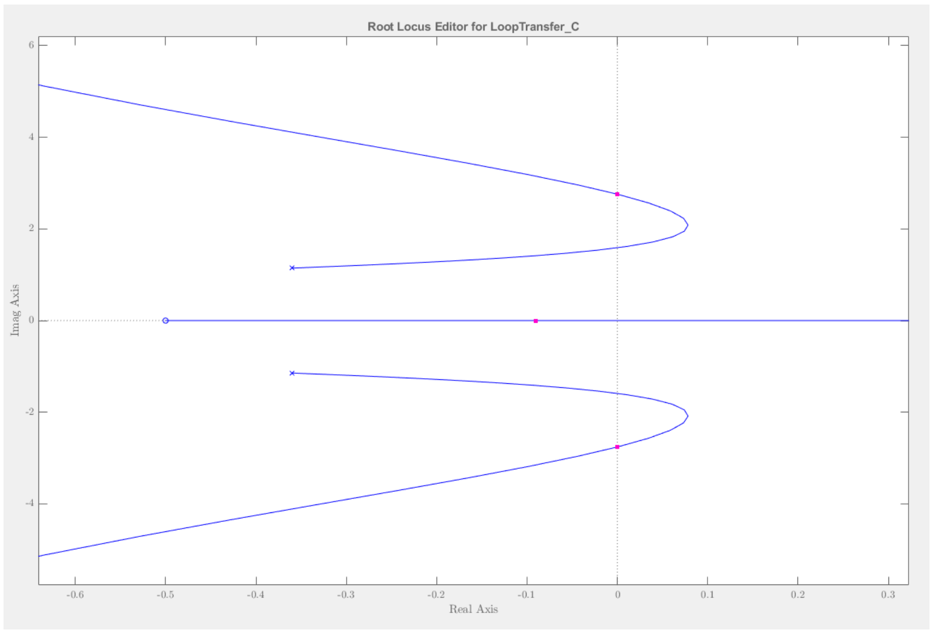
```
num = 11.0*conv([1, 0.5], [1, 3.0]);
den = conv([1, 0.72, 1.44], [1, 5.9, -11.9]);
L = tf(num, den);
% Plot
fig1 = figure(1);
    rlocus(L)
    title('Root Locus, Koike','interpreter','Latex')
    sgrid
saveas(fig1, fullfile(fdir, 'p3_RL.png'));
```

The Root Locus is the following

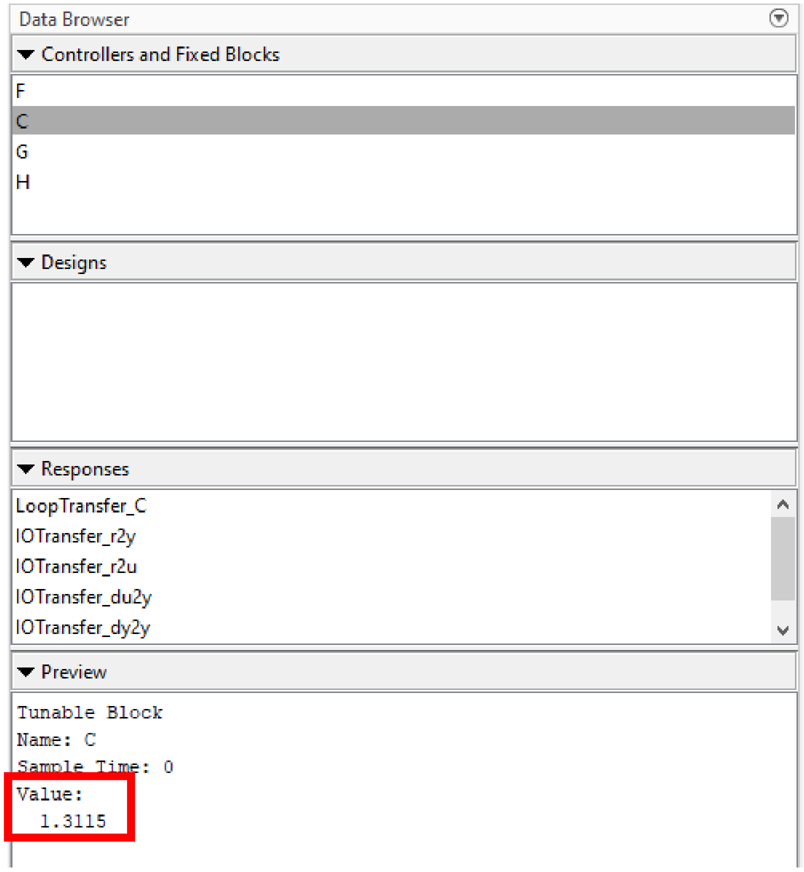


Then using the Control System Designer, we can find the range of gain k_p which makes the close loop system stable.





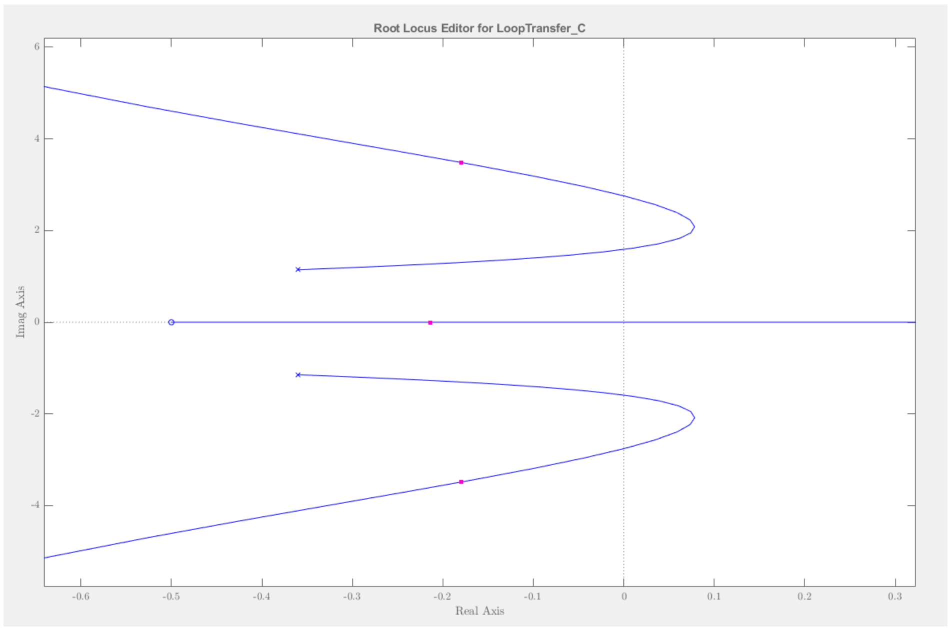
The plot above shows the poles of the closed loop system on the RL (pink) and at the specific threshold where the system is stable. At this threshold the gain value is



```
Tunable Block
Name: C
Sample Time: 0
Value:
  1.3115
```

$$k_p = 1.3115$$

If we increase the k_p value to 1.9929 we get something like

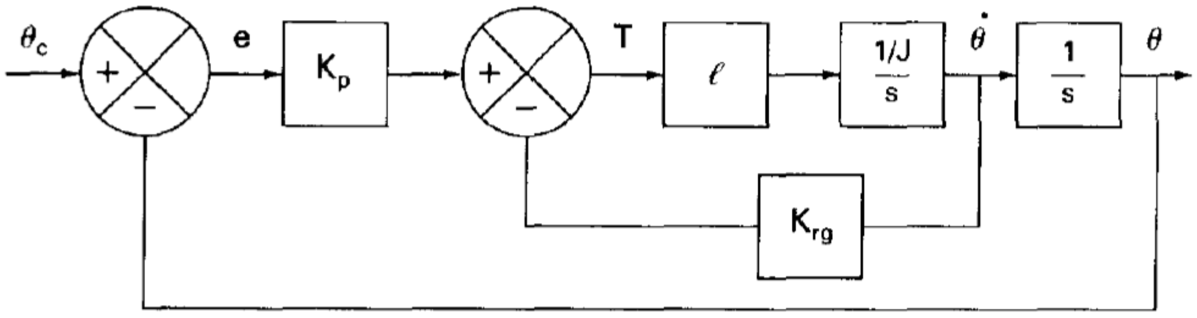


Thus, the range of the gain is

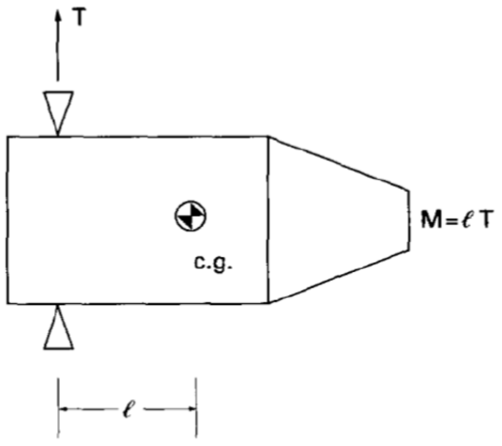
$$1.3115 \leq k_p \leq \infty \ .$$

Problem 4 (20pts)

The block diagram for a pitch attitude control system of a spacecraft is shown in Figure (a). Control of the spacecraft is achieved through thrusters located on the side of the spacecraft as illustrated in Figure (b). $\ell/J = 100$.



(a)



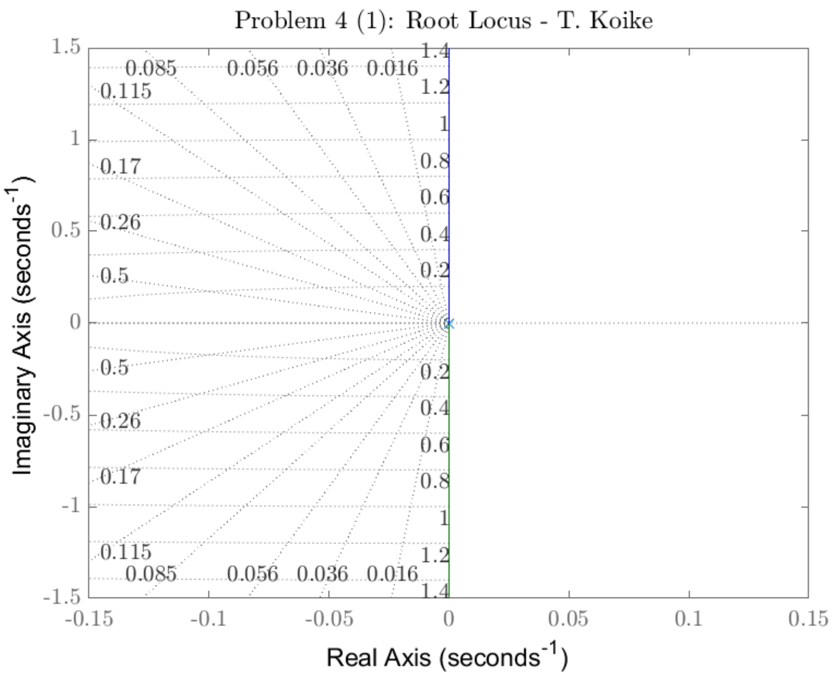
(b)

(1) Determine the root locus plot for the control system if the rate loop is disconnected (K_{rg}=0). Comment on the potential performance of this system for controlling the pitch attitude.

Then, the open loop transfer function becomes

$$G(s) = \frac{100}{s^2}$$

Thus, the RL (Root Locus) becomes



This RL shows that the potential performance of this system is unstable and will not give us acceptable responses.

MATLAB Code

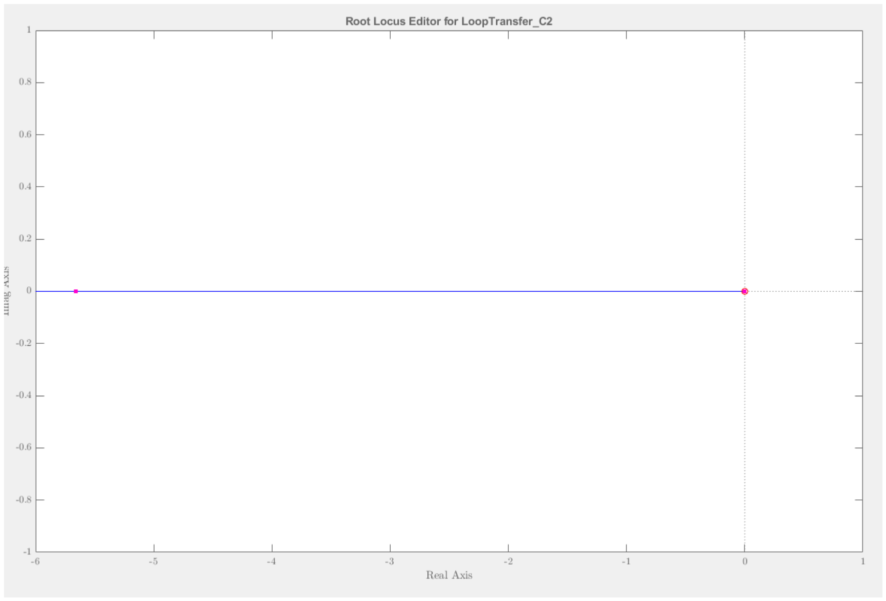
```
G = tf([0, 100], [J, 0, 0]);
% Plot
fig1 = figure(1);
    rlocus(G)
    title('Problem 4 (1): Root Locus - T. Koike ','interpreter','Latex')
    sgrid
saveas(fig1, fullfile(fdir, 'p4-1_RL.png'));
```

(2) Determine the rate gain K_{rg}, and the outer loop gyro gain K_p, so that the system has a damping ratio = 0.707 and settling time $t_s \leq 1.5\ s$.

Open the Control System Designer

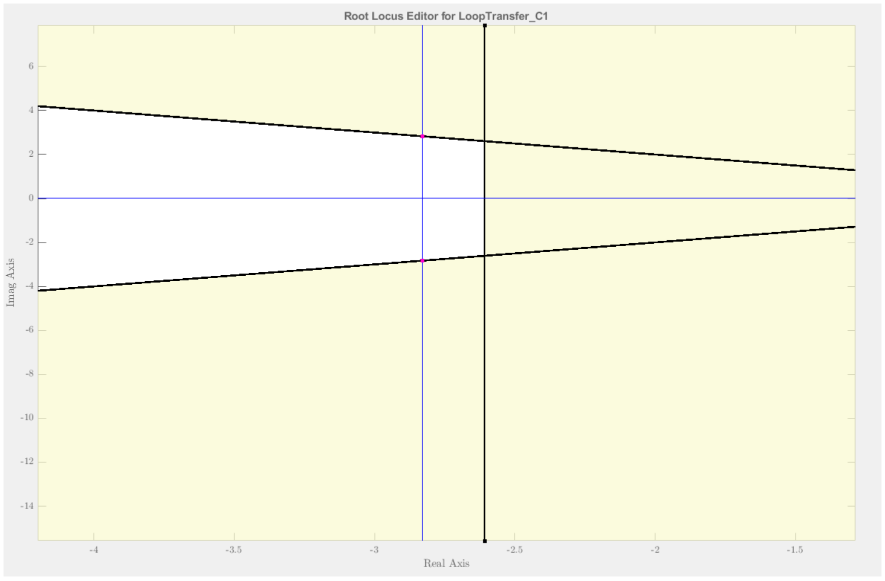
```
G = tf([0, 1], [J, 0, 0]);
C2 = tf([Kp, 0], [0, 1]);
controlSystemDesigner(G)
```

For the inner loop



We tune the gains so that the poles satisfy our requirements

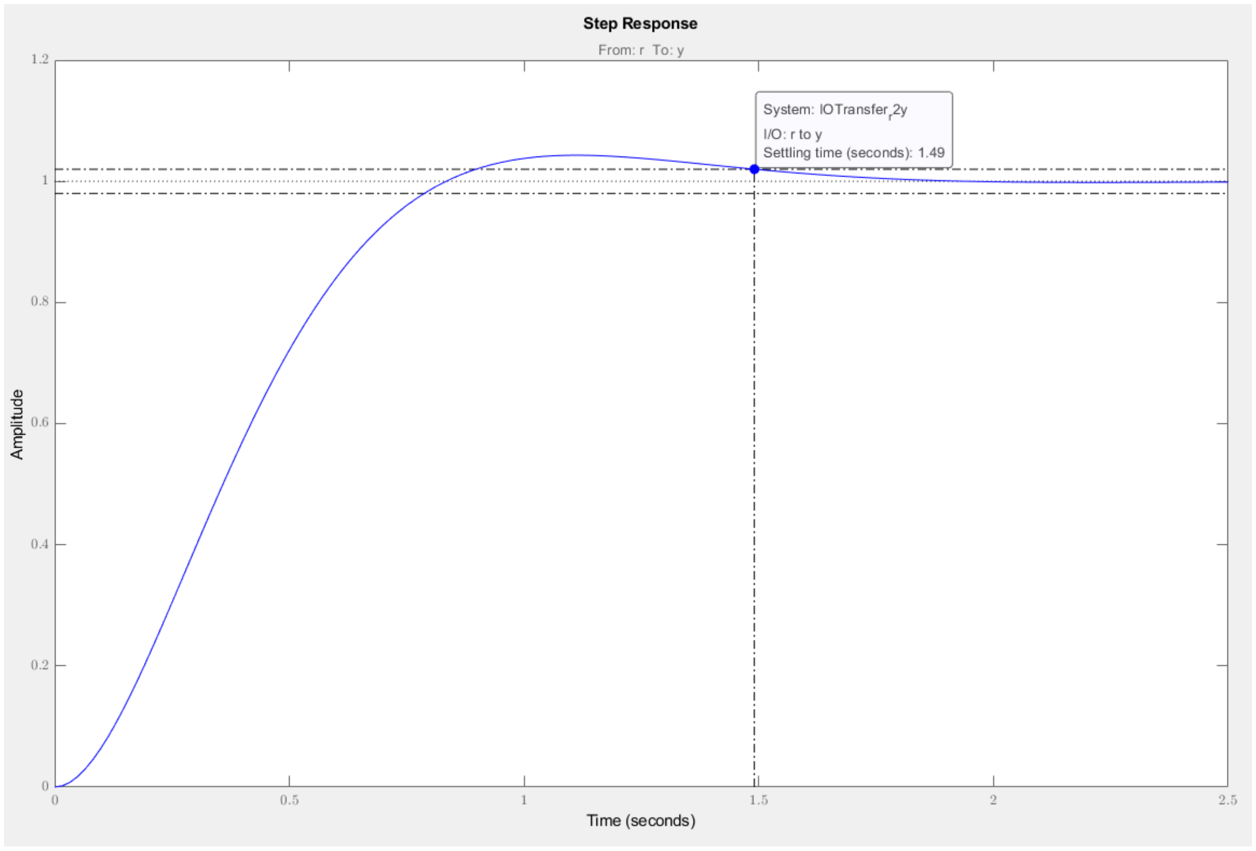
$$K_{rg} = 0.056595$$



Then for our outer loop

$$K_p = 0.15998 \ .$$

(3) Verify your design by providing the plot of system unit step response.



Problem 5 (20pts)

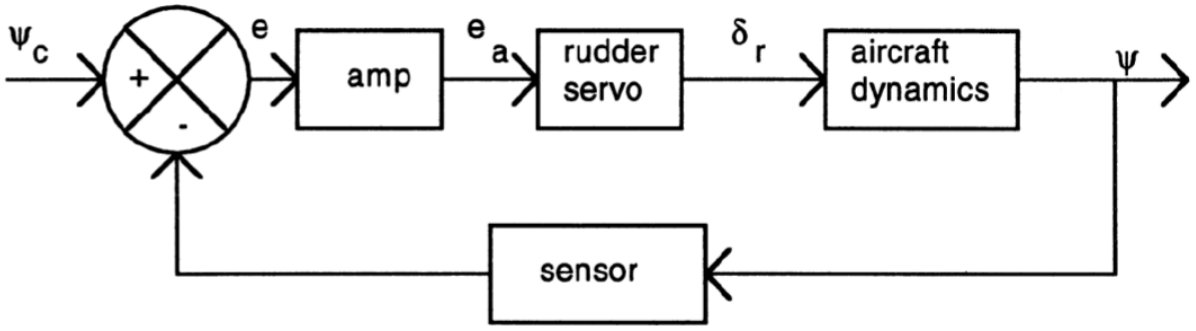
A wind-tunnel model is constrained so that it can rotate only about the z-axis; that is, pure yawing motion. The equation of motion for a constrained yawing motion was shown as follows:

$$\Delta\ddot{\psi} - N_r\Delta\dot{\psi} + N_\beta\Delta\psi = N_{\delta_r}\Delta\delta_r$$

where $N_\beta = 2.0s^{-2}, N_r = -0.5s^{-1}$ and $N_{\delta_r} = -10s^{-2}$. Design a heading control system so that the model has the following closed-loop performance characteristics: $\zeta = 0.6, t_s \leq 2.5$. Assume that the rudder servo transfer function can be represented as

$$\frac{\Delta\delta_r}{e} = \frac{k}{s+10}\ .$$

(Hint: A possible concept for a heading control system is shown below)



For the aircraft dynamics we will define a state space system

$$\Delta\ddot{\psi} = N_r\Delta\dot{\psi} - N_\beta\Delta\psi + N_{\delta_r}\Delta\delta_r$$

$$A = \begin{pmatrix} 0 & 1 \\ -N_\beta & N_r \end{pmatrix}, \qquad B = \begin{pmatrix} 0 \\ N_{\delta_r} \end{pmatrix}, \qquad C = (1 \quad 0), \qquad D = 0$$

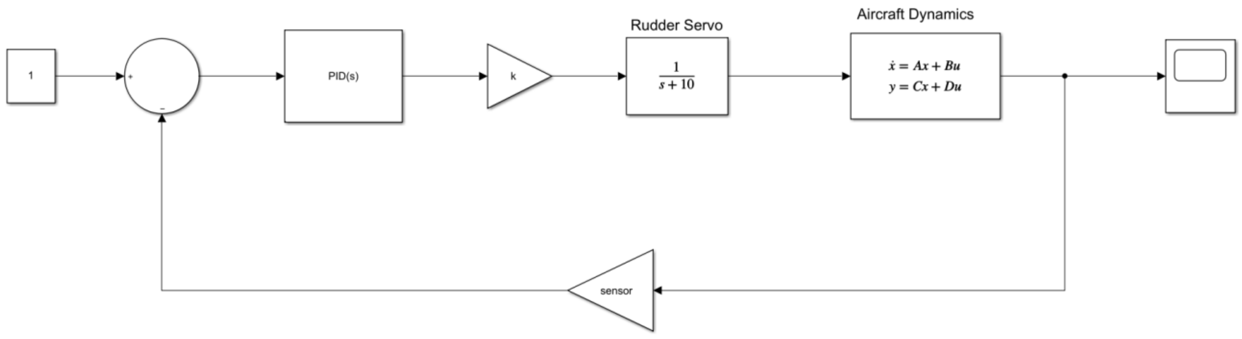
Plug in the given values and we have

$$A = \begin{pmatrix} 0 & 1 \\ -2.0 & -0.5 \end{pmatrix}, \qquad B = \begin{pmatrix} 0 \\ -10 \end{pmatrix}, \qquad C = (1 \quad 0), \qquad D = 0$$

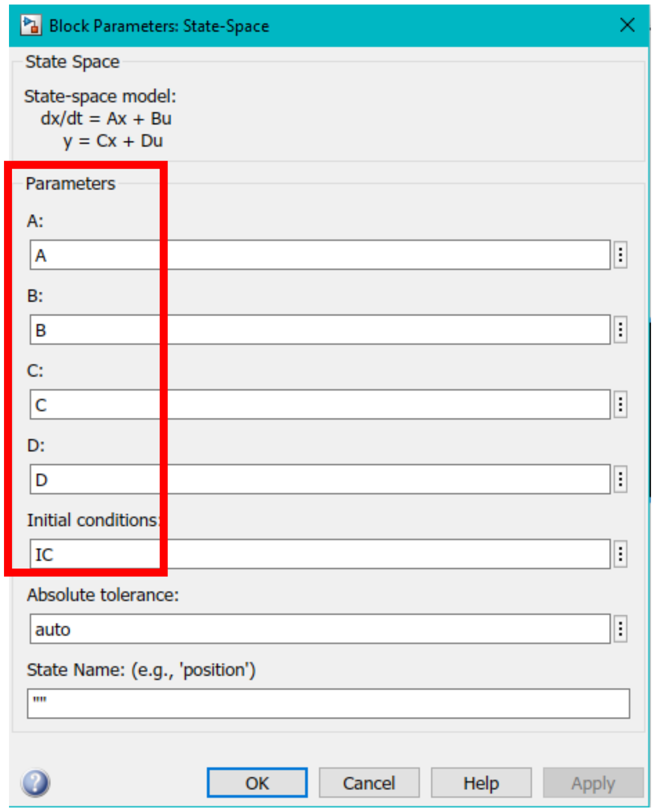
Let the sensor be represented by a constant gain block, and the amp is also a constant gain. The requirements imply that we improve both the transient response and the steady state response, and therefore, we will use a PID controller to optimize the control.

The first step is to construct the system in Simulink.

The Simulink model is the following.



Set the State Space Blocks as follows


Block Parameters: State-Space

State Space

State-space model:
dx/dt = Ax + Bu
y = Cx + Du

Parameters

A: A

B: B

C: C

D: D

Initial conditions: IC

Absolute tolerance: auto

State Name: (e.g., 'position') ''

With the parameters set as

```
% Aircraft dynamics
A = [0, 1 ; -2.0, -0.5];
B = [0 ; -10];
C = [1, 0];
D = 0;
```

```
% Initial conditions of the state space
IC = [0 ; 0];
```

And we initially set the PID controller, k, and sensor gains to

```
k = 5;
sensor = 1.2;


Kp = 25;
Ki = 8;
Kd = 2;
```

Now we use the Control System Tuner to tune these gains.

From the requirement specifications,

$$\%OS = exp\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right) \times 100 \ = \ 4.3255\ \% \quad \because \zeta = \ 0.707$$

Calculated using this MATLAB function

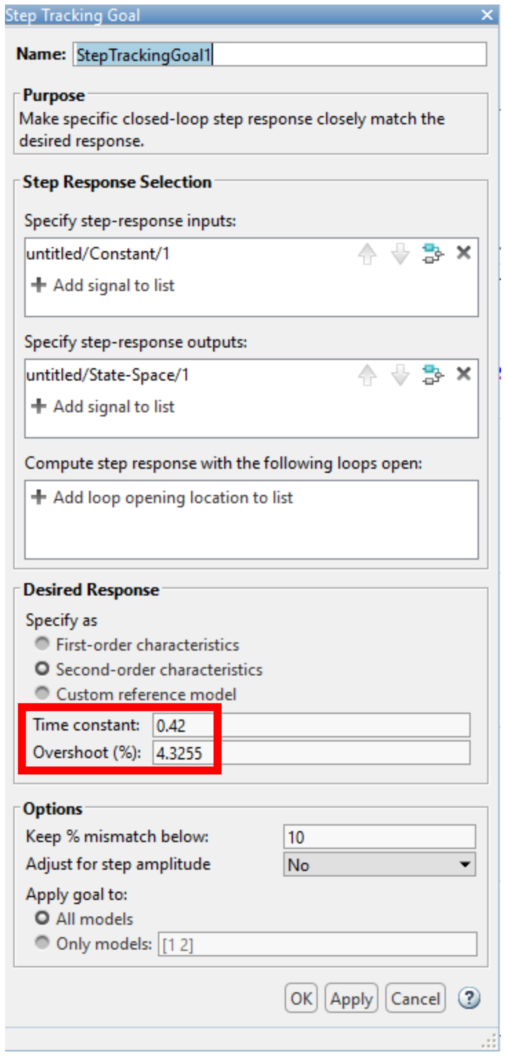
```
function output = calc_zetaFromMOS_or_MOSFromzeta(MOS_or_zeta, type)
    %{
      inputs:  1) MOS_or_zeta: maximum overshoot or zeta (damping ratio) input
                  the one of the two will be chosen depending on the second
                  input "type"
               2) type: string "MOS" 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
```

Also, the time constant, τ will have to satisfy

$$\text{time constant} \ \coloneqq \ \tau = \frac{t_s}{4.6} \leq 2.5 \ \ s$$

$$\Rightarrow \tau \leq 0.5435 \ \ s \ .$$

Thus, in the tuning goals we select “Step Tracking Goals” and set the specifications as



Name: StepTrackingGoal1

Purpose
Make specific closed-loop step response closely match the desired response.

Step Response Selection
Specify step-response inputs:
untitled/Constant/1

Specify step-response outputs:
untitled/State-Space/1

Compute step response with the following loops open:

Desired Response
Specify as
First-order characteristics [✓ Second-order characteristics] Custom reference model

Time constant:	0.42
Overshoot (%):	4.3255

Options

Keep % mismatch below:	10
Adjust for step amplitude	No

Apply goal to:
[✓ All models] Only models: [1 2]

Then we are able to obtain the optimized gains for the PID controller, rudder servo, and the sensor.

PID:

$$K_p + K_i \frac{1}{s} + K_d \frac{s}{T_f s + 1}$$

$$where \quad K_p = -0.112, \qquad K_i = -1.42, \qquad K_d = -0.675, \qquad T_f = 0.166$$

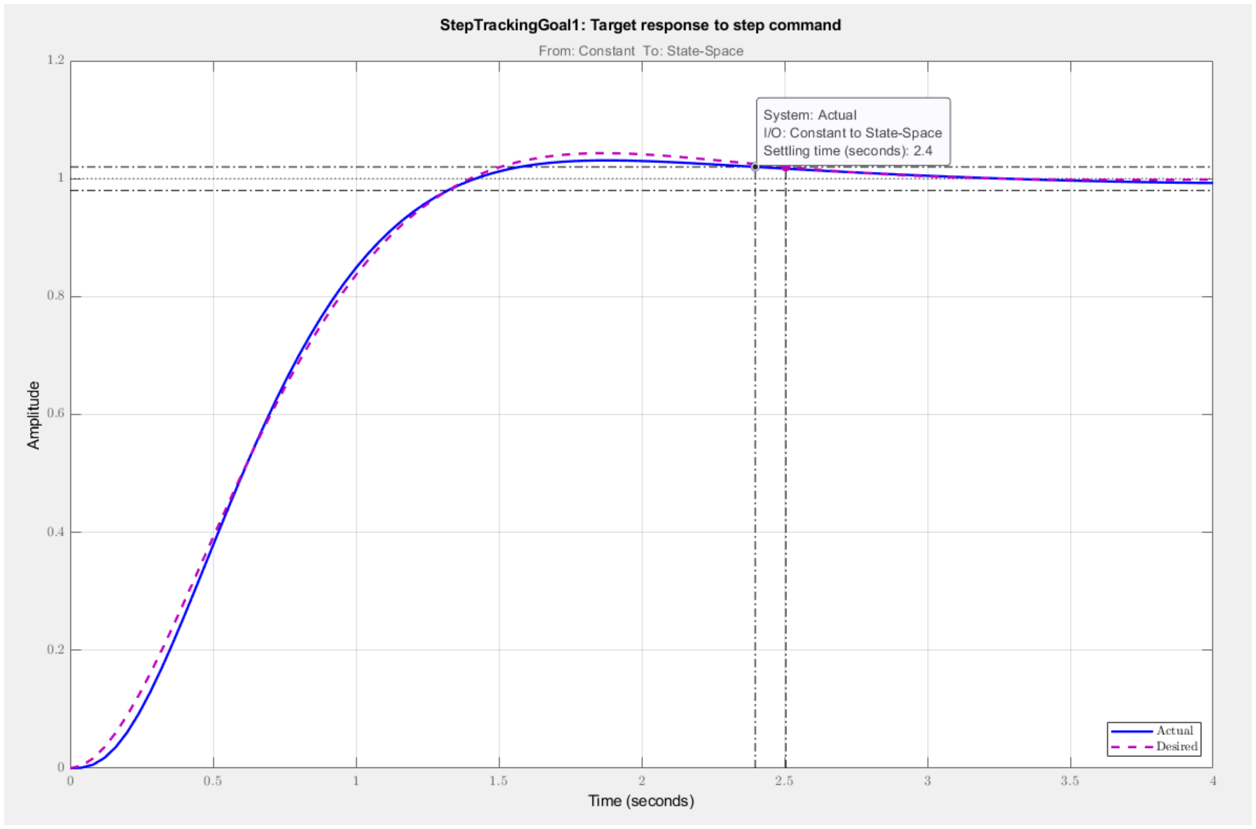
Rudder servo, k:

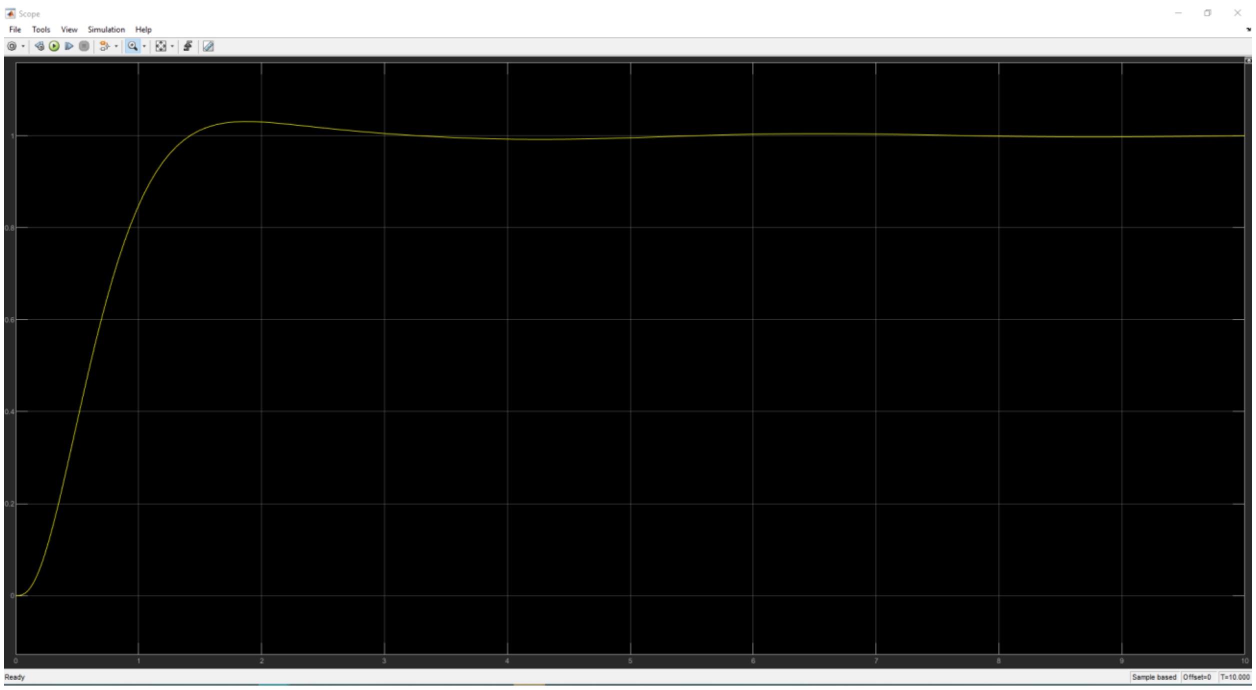
$$k = 2.295 \;.$$

Sensor gain:

$$K_{sensor} = 1.0 \;.$$

The step response of this PID controlled system becomes


StepTrackingGoal1: Target response to step command
From: Constant To: State-Space
System: Actual
I/O: Constant to State-Space
Settling time (seconds): 2.4
Amplitude
Time (seconds)
Actual
Desired


Scope
File Tools View Simulation Help
Ready
Sample based
Offset=0
T=10.000