Numerical Analysis Final Report

21600372 송윤경

1. Cruciform rocket system

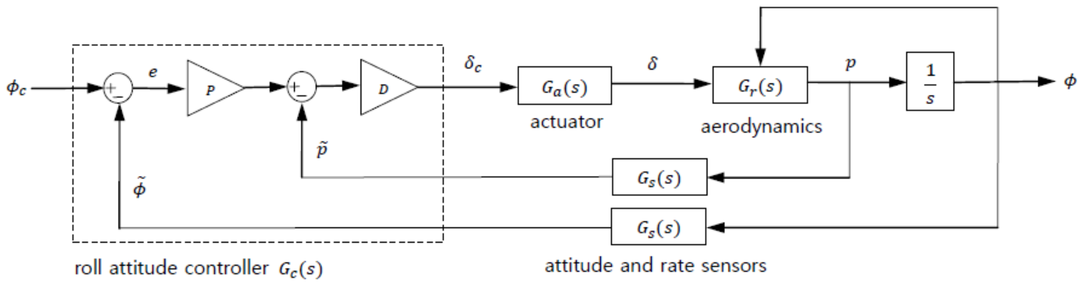

Figure 1. Cruciform Rocket System Block Diagram

ODE solver 를 통해 제어하고자 하는 Cruciform Rocket System 시스템이다.

2. Simulink

   A. Simulink Block diagram

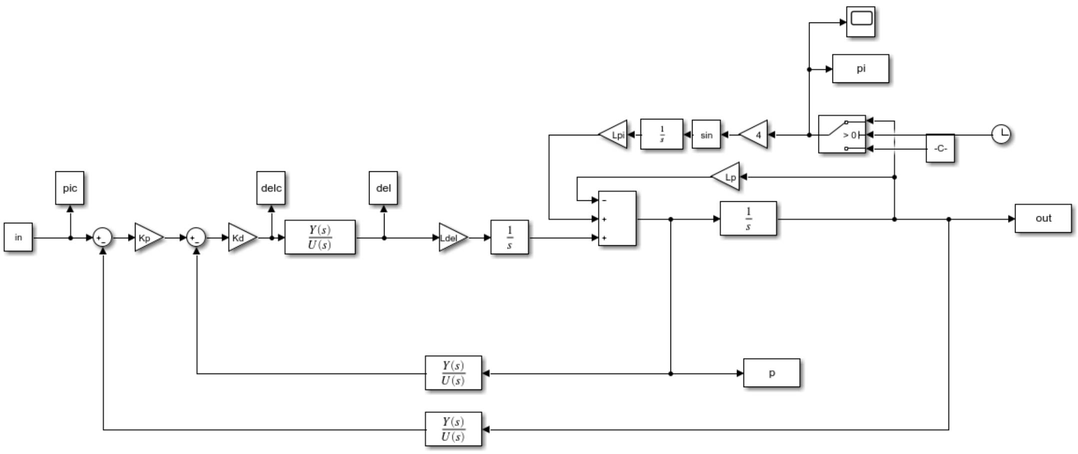

Figure 2. Cruciform Rocket System Block Diagram in Simulink

시뮬레이션을 하기 위해 동적시스템이 반영된 G_r 의 전달함수 부분을 분리하여 시뮬레이션을 진행하였다. 주어진 시스템이 미분한 형태였으므로 $\dot{p}(t)$ 을 적분하고 각각의 입력에 따른 미분적분의 형태를 전달함수로 바꿔 피드백을 진행하였다.

B. Simulink result

- $\phi_c = \frac{\pi}{4}$ 인 경우 simulation 결과

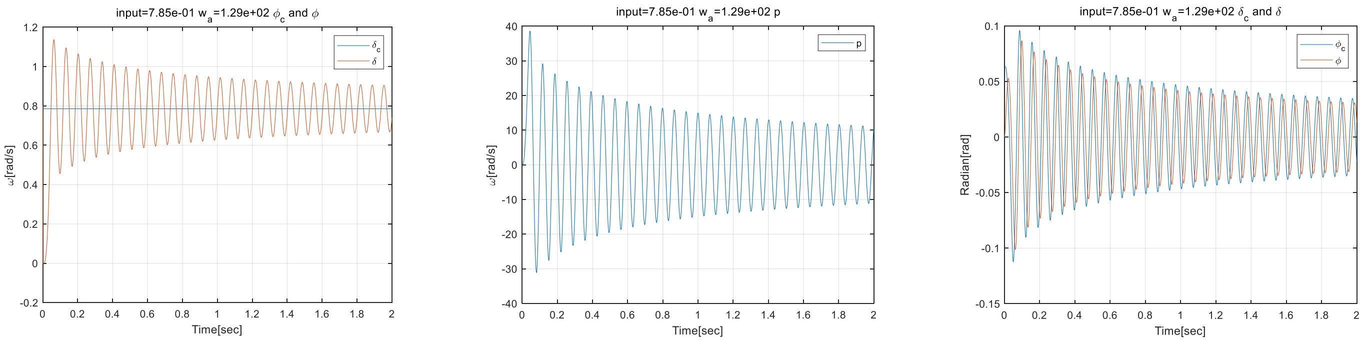

Figure 3. $\omega_a = 20.5[rad/s]$, $\phi_c = \frac{\pi}{4}$

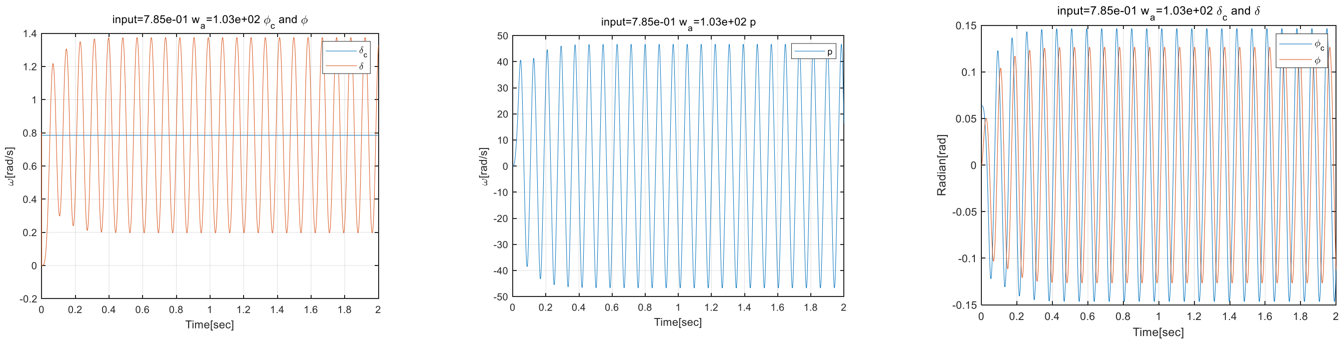

Figure 4. $\omega_a = 16.4[rad/s]$, $\phi_c = \frac{\pi}{4}$

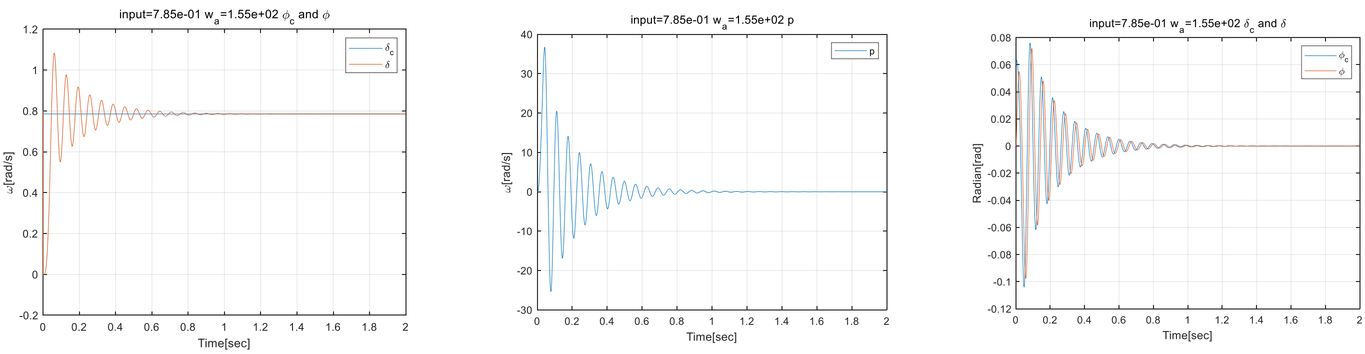

Figure 5. $\omega_a = 24.6[rad/s]$, $\phi_c = \frac{\pi}{4}$

- $\phi_c = \frac{\pi}{8}$ 인 경우 simulation

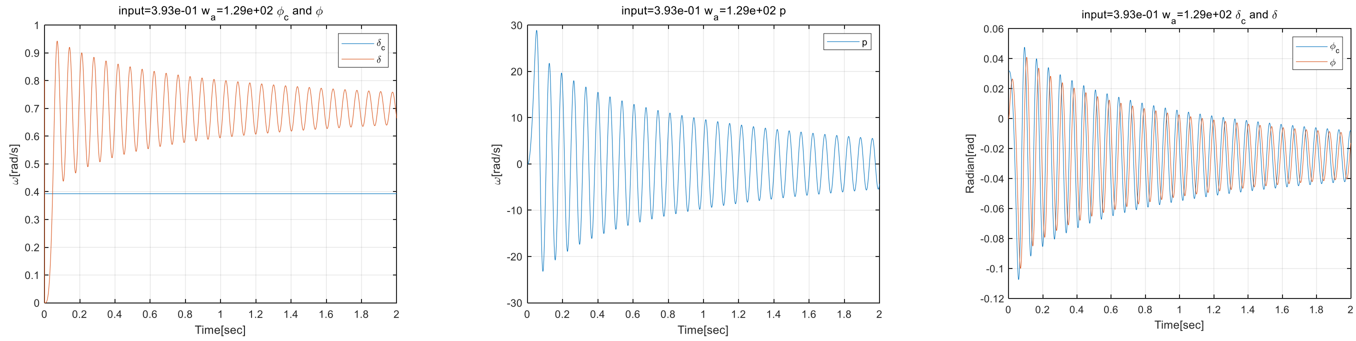

Figure 6. 20.5$[rad/s]$, $\phi_c = \frac{\pi}{8}$

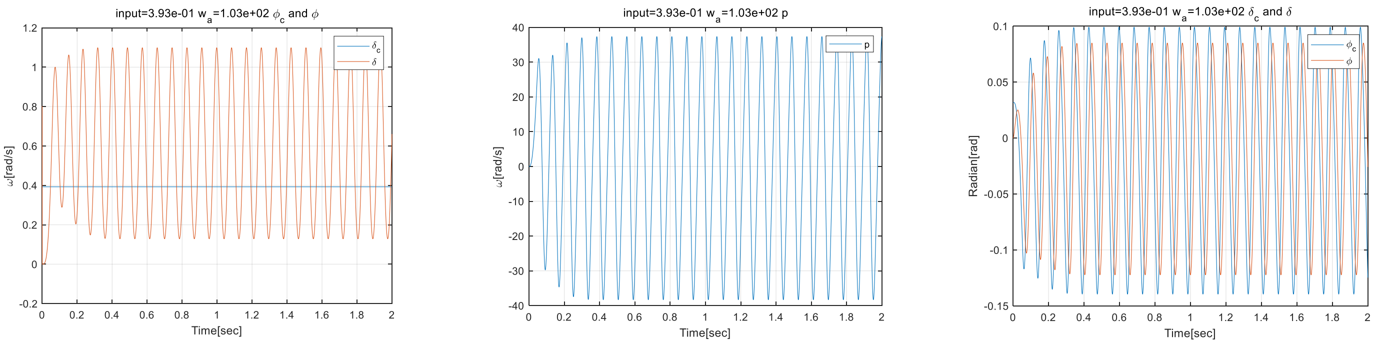

Figure 7. $\omega_a = 16.4[rad/s]$, $\phi_c = \frac{\pi}{8}$

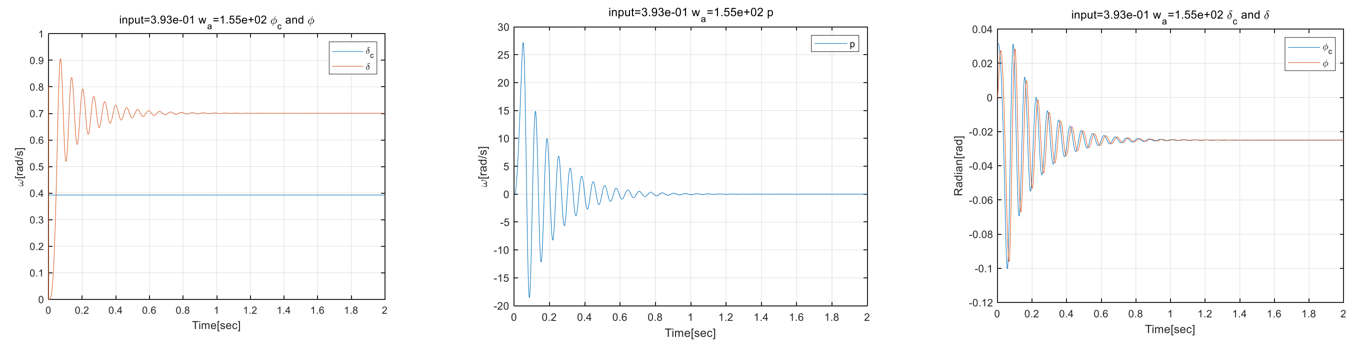

Figure 8. $\omega_a = 24.6[rad/s]$, $\phi_c = \frac{\pi}{8}$

- $\phi_c = 0$ 인 경우 simulation

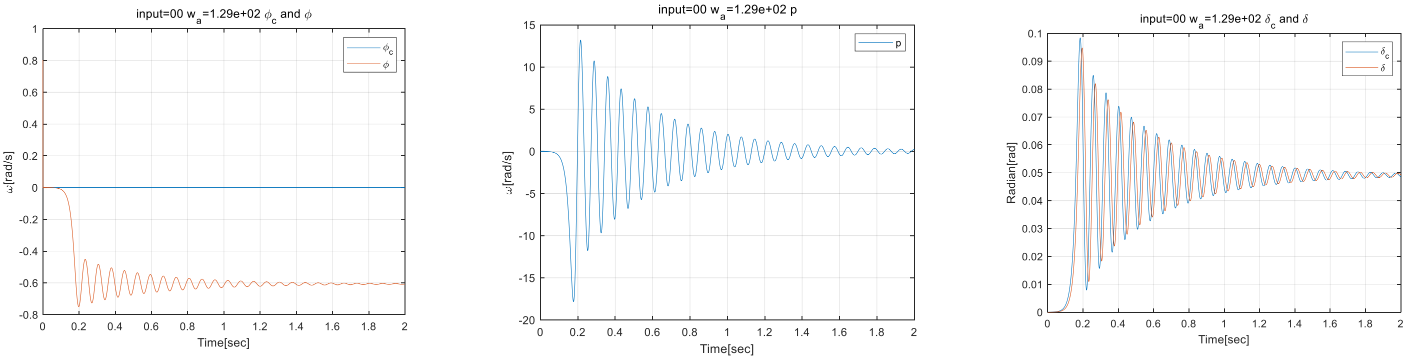

Figure 6. $\omega_a = 20.5[rad/s]$, $\phi_c = 0$

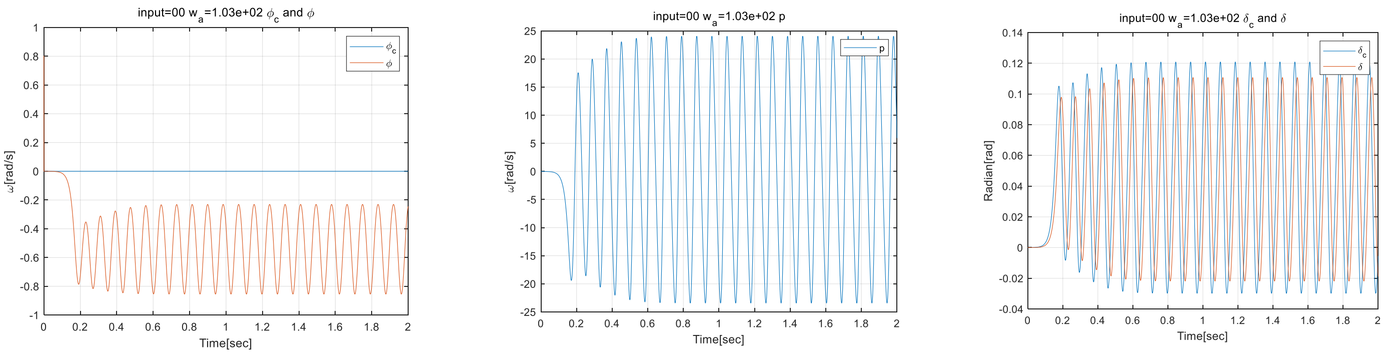

Figure 6. $\omega_a = 16.4[rad/s]$, $\phi_c = 0$

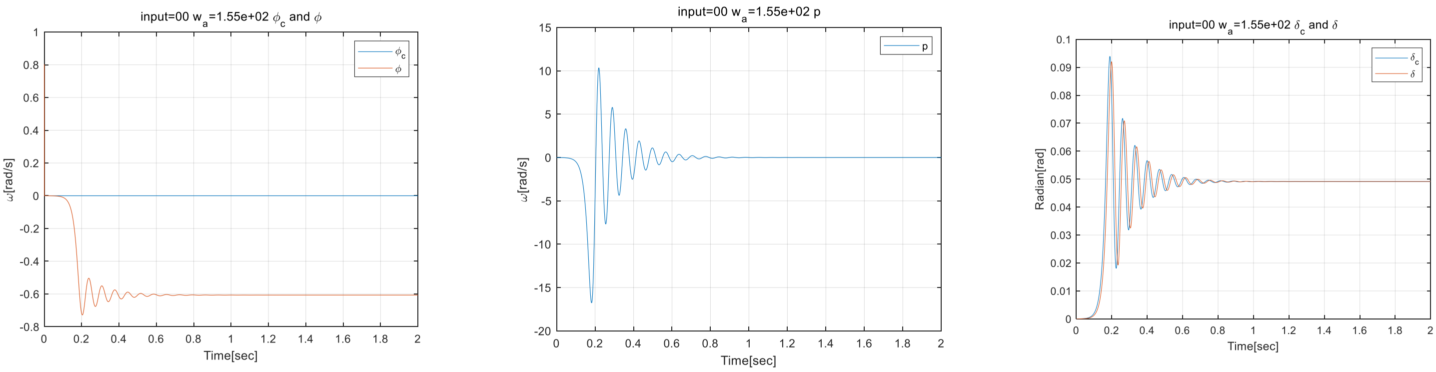

Figure 6. $\omega_a = 24.6[rad/s]$, $\phi_c = 0$

3. C-programing

- Main

```c
/*-----------------------------------------------
            2020-1  Numerucal Analysis
                Final: ODE SOLVER
            21600372  Yoonkyoung Song
-----------------------------------------------*/

#include "myInclude.h"
#include "myUserDefInclude.h"
#include "NAFinal.h"


int sysnum = 8;
int iter = 0;

Matrix sys1A, sys1B;
Matrix sys2A, sys2B;
Matrix dstate, state;
Matrix beforeState, beforedstate;
Matrix k;
Matrix delC;

double t[DATANUM] = { 0.0 };
double phic[DATANUM] = {  INITpic };
double delc[DATANUM] = { 0.0 };
double del[DATANUM] = { 0.0 };
double p[DATANUM] = { 0.0 };
double phi[DATANUM] = { 0.0 };

void main()
{
    initalizeSystme(&sys1A, &sys1B, &sys2A, &sys2B);
    initalizeState(&dstate, &state, &beforeState, &beforedstate, &delC, &k, sysnum);

    do {

        input(phic,iter);


        system4(&sys1A, &sys1B, &dstate, &state, &beforeState, &beforedstate);
        system1(&dstate, &state, &beforeState, phic, &delC, iter);
        system2(&sys2A, &sys2B, &dstate, &state, &delC);
        systme3(&dstate, &state, &beforeState);


        storageData(&state, &beforeState, &dstate, &beforedstate, &delC, delc, del, p, phi, iter);

        integration(&dstate, &state, &k, RK4, &iter);
        checkStop(t, iter);

    } while (t[iter]<tf);

    WriteTxtFile("in0_wa24.txt", DATANUM, t, phic, delc, del, p, phi);

}
```

Figure 7. main 문

Iteration 이 증가함에 따라 Runge-Kutta integration 이 진행된다. 초기값일 때 피드백 시스템부터 시작하여 각 시스템에서 dstate 를 계산하고 그 계산결과를 integration 에서 한 번에 Runge-Kutta 4 order 를 사용하여 수치적으로 적분한다.

<table>
<tr><td rowspan="8">state</td><td>[0]</td><td>δ</td><td rowspan="8">dstate</td><td>[0]</td><td>$\dot{\delta}$</td></tr>
<tr><td>[1]</td><td>$\dot{\delta}$</td><td>[1]</td><td>$\ddot{\delta}$</td></tr>
<tr><td>[2]</td><td>ϕ</td><td>[2]</td><td>$\dot{\phi}$</td></tr>
<tr><td>[3]</td><td>$\dot{\phi}$</td><td>[3]</td><td>$\ddot{\phi}$</td></tr>
<tr><td>[4]</td><td>$\tilde{p}$</td><td>[4]</td><td>$\tilde{p}'$</td></tr>
<tr><td>[5]</td><td>$\tilde{p}'$</td><td>[5]</td><td>$\tilde{p}''$</td></tr>
<tr><td>[6]</td><td>$\tilde{\phi}$</td><td>[6]</td><td>$\tilde{\phi}$</td></tr>
<tr><td>[7]</td><td>$\tilde{\phi}'$</td><td>[7]</td><td>$\tilde{\phi}''$</td></tr>
</table>

Table 1. 변수 정의

- Source file

```c
#include "NAFinal.h"

void initalizeState(Matrix* dstate, Matrix* state, Matrix* beforeState, Matrix* beforedstate, Matrix* dleC, Matrix* k, int sysnum) {

    *dstate = createZerosMathrix(4, sysnum);
    *state = createZerosMathrix(4, sysnum);
    *beforeState = createZerosMathrix(4, sysnum);
    *beforedstate = createZerosMathrix(4, sysnum);
    *k = createZerosMathrix(4, sysnum);

    *dleC = createZerosMathrix(4, 1);

    state->at[0][2] = INITpic + 1.0 * PI / 180;
}

void initalizeSystme(Matrix* sys1A, Matrix* sys1B, Matrix* sys2A, Matrix* sys2B) {

    *sys1A = createZerosMathrix(2, 2);
    *sys1B = createZerosMathrix(2, 1);

    *sys2A = createZerosMathrix(2, 2);
    *sys2B = createZerosMathrix(2, 1);

    sys1A->at[0][0] = 0;
    sys1A->at[0][1] = 1;
    sys1A->at[1][0] = pow(Ws, 2) * (-1);
    sys1A->at[1][1] = Zs * Ws * (-2);

    sys1B->at[0][0] = 0;
    sys1B->at[1][0] = pow(Ws, 2);

    sys2A->at[0][0] = 0;
    sys2A->at[0][1] = 1;
    sys2A->at[1][0] = pow(Wa, 2) * (-1);
    sys2A->at[1][1] = Za * Wa * (-2);

    sys2B->at[0][0] = 0;
    sys2B->at[1][0] = pow(Wa, 2);
}

void input(double pic[], int iter) {

    pic[iter] = INITpic;

}
```

Figure 8. initial value setting

시스템이 돌아가면서 사용되는 변수들을 모두 초기화 한다.

```c
void system1(Matrix* dstate, Matrix* state, Matrix* beforeState, double input[], Matrix* delC, int iter) {

    for (int i = 0; i < 4; i++) {
        delC->at[i][0] = Kd * (Kp * (input[iter] - state->at[i][6]) - state->at[i][4]);
    }

}

void system2(Matrix* A, Matrix* B, Matrix* dstate, Matrix* state, Matrix* delC) {

    for (int i = 0; i < 4; i++) {

        dstate->at[i][0] = state->at[i][1];
        dstate->at[i][1] = A->at[1][0] * state->at[i][0] + A->at[1][1] * state->at[i][1] + B->at[1][0] * delC->at[i][0];
    }

}

void systme3(Matrix* dstate, Matrix* state, Matrix* beforeState) {

    for (int i = 0; i < 4; i++) {

        dstate->at[i][2] = state->at[i][3];
        dstate->at[i][3] = Lpi * sin(4 * beforeState->at[i][2]) - Lp * state->at[i][3] + Ldel * state->at[i][0];


    }

}

void system4(Matrix* A, Matrix* B, Matrix* dstate, Matrix* state, Matrix* beforeState, Matrix* beforedstate) {


    for (int i = 0; i < 4; i++) {

        dstate->at[i][4] = state->at[i][5];
        dstate->at[i][5] = A->at[1][0] * state->at[i][4] + A->at[1][1] * state->at[i][5] + B->at[1][0] * state->at[i][3];

        dstate->at[i][6] = state->at[i][7];
        dstate->at[i][7] = A->at[1][0] * state->at[i][6] + A->at[1][1] * state->at[i][7] + B->at[1][0] * state->at[i][2];

    }

}
```

Figure 9. update dstate using state

2 차 시스템을 벡터를 사용하여 1 차 시스템으로 근사한 후 시스템의 상태변수를 사용하여 시스템을 정의하였다.

```
void integration(Matrix* dstate, Matrix* state, Matrix* k, int method, int* iter) {

    switch(method){

    case 1:
        Euler(dstate, state);

    case 2:

        RungeKutta4(dstate, state, k);

    default :
        break;

    }

    *iter += 1;

}

void Euler(Matrix* dstate, Matrix* state) {

    for (int i = 0; i < (*dstate).cols; i++) {

        state->at[0][i] = state->at[0][i] + ts * dstate->at[0][i];
    }

}

void RungeKutta4(Matrix* dstate, Matrix* state, Matrix* k) {

    for (int j = 0; j < (*dstate).cols/2; j++) {

        for (int i = 0; i < 4; i++) {
            k->at[i][2 * j] = ts * dstate->at[i][2 * j + 1];
        }

        dstate->at[0][2 * j] = dstate->at[0][2 * j] + (k->at[0][2 * j] + 2 * k->at[1][2 * j] + 2 * k->at[2][2 * j] + k->at[3][2 * j]) / 6;
        dstate->at[1][2 * j] = dstate->at[0][2 * j] + 0.5 * k->at[0][2 * j];
        dstate->at[2][2 * j] = dstate->at[0][2 * j] + 0.5 * k->at[1][2 * j];
        dstate->at[3][2 * j] = dstate->at[0][2 * j] + k->at[2][2 * j];

        state->at[0][2 * j + 1] = dstate->at[0][2 * j];
        state->at[1][2 * j + 1] = dstate->at[1][2 * j];
        state->at[2][2 * j + 1] = dstate->at[2][2 * j];
        state->at[3][2 * j + 1] = dstate->at[3][2 * j];

    }

    for (int j = 0; j < (*dstate).cols / 2; j++) {

        for (int i = 0; i < 4; i++) {
            k->at[i][2 * j] = ts * dstate->at[i][2 * j];
        }

        state->at[0][2 * j] = state->at[0][2 * j] + (k->at[0][2 * j] + 2 * k->at[1][2 * j] + 2 * k->at[2][2 * j] + k->at[3][2 * j]) / 6;
        state->at[1][2 * j] = state->at[0][2 * j] + 0.5 * k->at[0][2 * j];
        state->at[2][2 * j] = state->at[0][2 * j] + 0.5 * k->at[1][2 * j];
        state->at[3][2 * j] = state->at[0][2 * j] + k->at[2][2 * j];

    }




}
```

Figure 9. update dstate using state

Runge-Kutta 4th order 를 사용하여 ODE 를 푼다. State 가 미분한 상태까지 두 개씩 돌아가므로 각각의 state 를 update 하기 위하여 두 번 미분한 datae 의 홀수 index 를 사용하여 datae 의 짝수 index 와 state 의 홀수 index 를 업데이트 해준다. 그 후에 업데이트된 datate 의 짝수 index 를 사용하여 satae 의 짝수 index 를 업데이트 해주는 방식으로 진행된다.

```c
void checkStop(double t[], int iter) {

    t[iter] = t[iter-1] + ts;

}

void storageData(Matrix* state, Matrix* beforeState, Matrix* dstate, Matrix* beforedstate, Matrix* delC, double delc[], double del[], double p[], double phi[], int iter) {

    del[iter] = state->at[0][0];
    delc[iter] = delC->at[0][0];
    p[iter] = state -> at[0][3];
    phi[iter] = state->at[0][2];

    *beforeState = *state;
    *beforedstate = *dstate;

}



void WriteTxtFile(char Name[], double N_DATA, double data1[], double data2[], double data3[], double data4[], double data5[], double data6[]) {

    FILE* pF1;

    //char way[1000] = "C:\\Users\\xde12\\OneDrive\\바탕 화면\\";


    pF1 = fopen(Name, "w");

    for (int count = 0; count < N_DATA; count++)
    {
        fprintf(pF1, "%lf %lf %lf %lf %lf %lf \n", data1[count], data2[count], data3[count], data4[count], data5[count], data6[count]);
    }

    fclose(pF1);

    printf("Storage data file");
}
```

Figure 10. data storage function

- Header file

```c
#ifndef HEADER_USER_DEFINED_NAFINAL
#define HEADER_USER_DEFINED_NAFINAL

#include "myInclude.h"
#include "myMatrix.h"

#define  PI (double) 3.14159265358979323846

#define INITpic (double) 0.0  // PI/8 PI/4 0

#define Za (double) 0.7
#define Wa (double) 2*PI*24.6 //16.4 24.6 20.5

#define Zs (double) 0.7
#define Ws (double) 2*PI*100

#define Kp (double) 27.0
#define Kd (double) 0.003

#define Lp (double) 3.2
#define Lpi (double) 1200
#define Ldel (double) 16000

#define ts (double) 0.0005
#define tf (double) 2.0

#define DATANUM (int) 4000

enum methodODE { EULER = 1, RK4};
//enum systemVari { DEL = 0, dDEL, PHI, dPHI, hPHI, dhPHI, hP, dhP};

void initalizeState(Matrix* dstate, Matrix* state, Matrix* beforeState, Matrix* beforedstate, Matrix* dleC, Matrix* k, int sysnum);
void initalizeSystme(Matrix* sys1A, Matrix* sys1B, Matrix* sys2A, Matrix* sys2B);

void input(double pic[], int iter);
```

```c
void system1(Matrix* dstate, Matrix* state, Matrix* beforeState, double input[], Matrix* delC, int iter);
void system2(Matrix* A, Matrix* B, Matrix* dstate, Matrix* state, Matrix* delC);
void systme3(Matrix* dstate, Matrix* state, Matrix* beforeState);
void system4(Matrix* A, Matrix* B, Matrix* dstate, Matrix* state, Matrix* beforeState, Matrix* beforedstate);

void storageData(Matrix* state, Matrix* beforeState, Matrix* dstate, Matrix* beforedstate, Matrix* delC, double delc[], double del[], double p[], double phi[], int iter);

void integration(Matrix* dstate, Matrix* state, Matrix* k, int method, int* iter);

void Euler(Matrix* dstate, Matrix* state);
void RungeKutta4(Matrix* dstate, Matrix* state, Matrix* k);


void checkStop(double t[], int iter);
void WriteTxtFile(char Name[], double N_DATA, double data1[], double data2[], double data3[], double data4[], double data5[], double data6[]);


#endif
#pragma once
```

Figure 11. define variation and function

4. Result

- $\phi_c = \frac{\pi}{4}$ 인 경우 결과

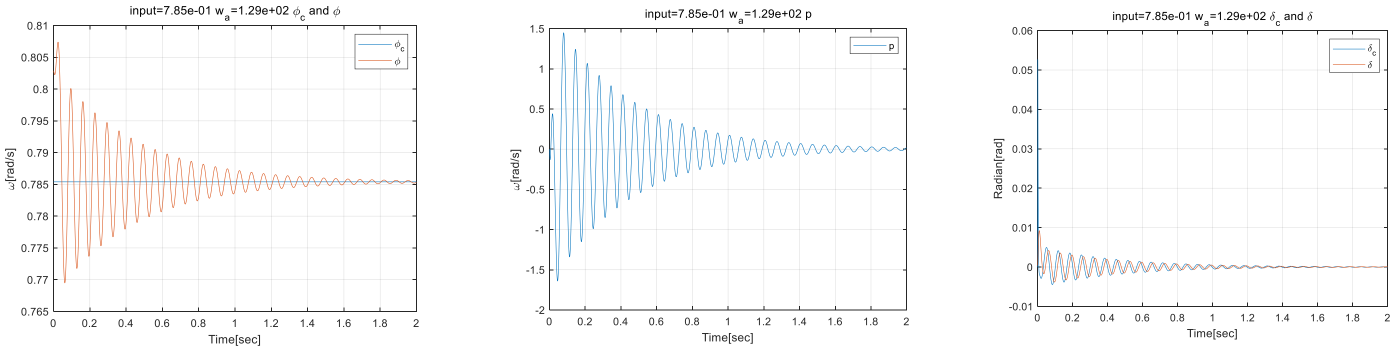

Figure 3. $\omega_a = 20.5[rad/s]$, $\phi_c = \frac{\pi}{4}$

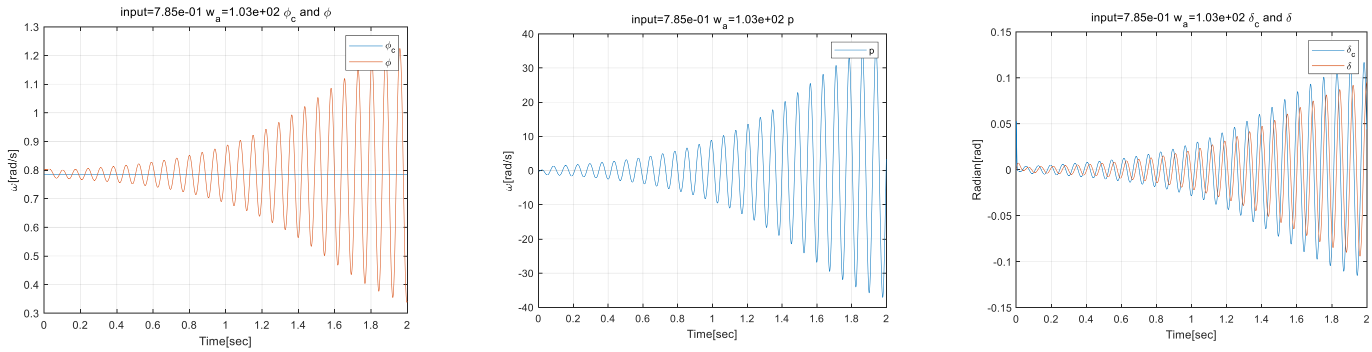

Figure 4. $\omega_a = 16.4[rad/s]$, $\phi_c = \frac{\pi}{4}$

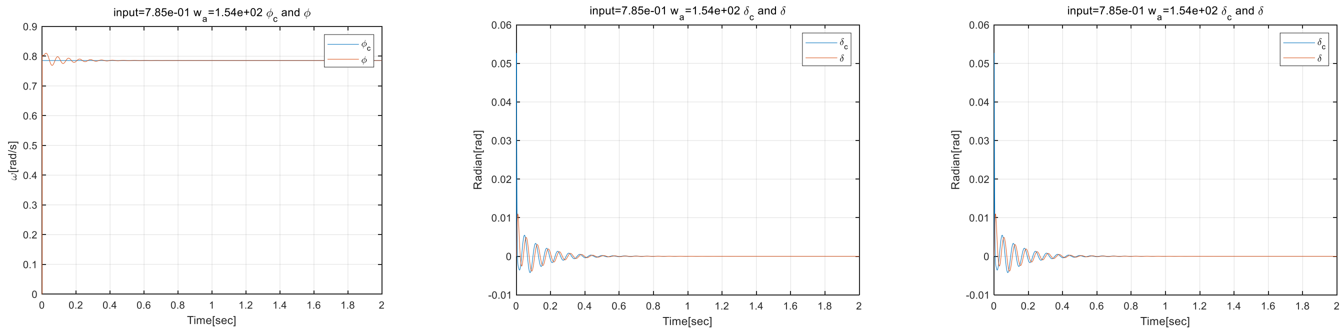

Figure 5. $\omega_a = 24.6[rad/s]$, $\phi_c = \frac{\pi}{4}$

시뮬레이션 결과와 비교했을 때 다른 경우들에 비해 경향성이 다르게 나타난다. 입력이 $\frac{\pi}{4}$인 경우에 시뮬레이션에 비해 진동하는 크기가 매우 작고 setting time 이 시뮬레이션 보다 빠르게 나타난다. $\omega_a = 16.4[rad/s]$일 때는 다른 시스템에 비해 band width 가 작아 수렴하지 못하고 발산함을 보인다. 하지만 다른 두 경우, $\omega_a = 20.5[rad/s]$ 와 $\omega_a = 24.6[rad/s]$ 일 때는 출력이 입력과 같은 $\frac{\pi}{4} = 0.78539$로 수렴함을 보인다. ω_a가 클수록 시스템의 margin 이 커져 시스템의 진동이 작아지고 steady state 상태에 빠르게 도달한다.

- $\phi_c = \frac{\pi}{8}$ 인 경우 결과

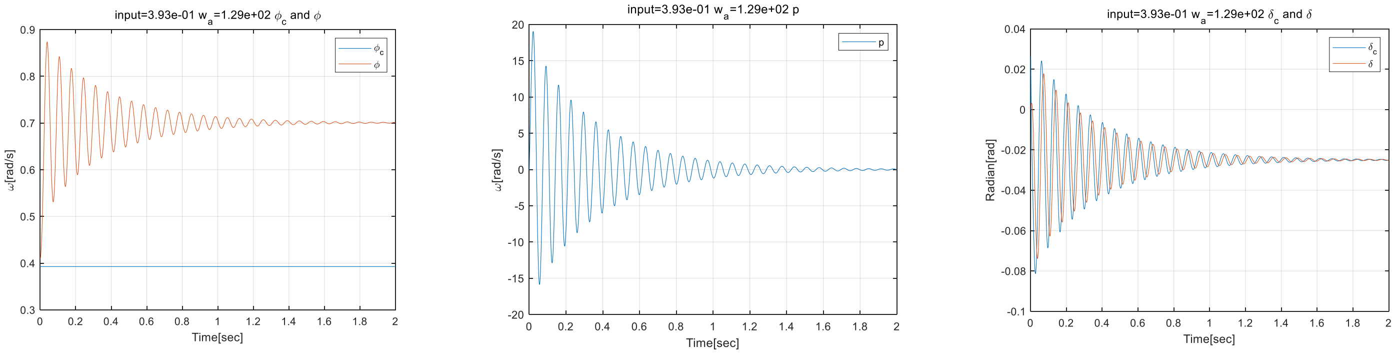

Figure 6. $\omega_a = 20.5[rad/s]$, $\phi_c = \frac{\pi}{8}$

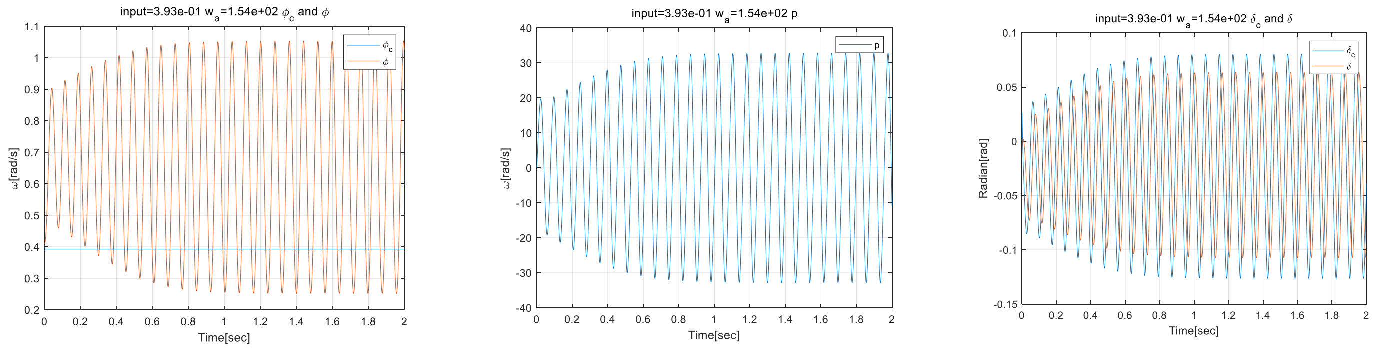

Figure 7. $\omega_a = 16.4[rad/s]$, $\phi_c = \frac{\pi}{8}$

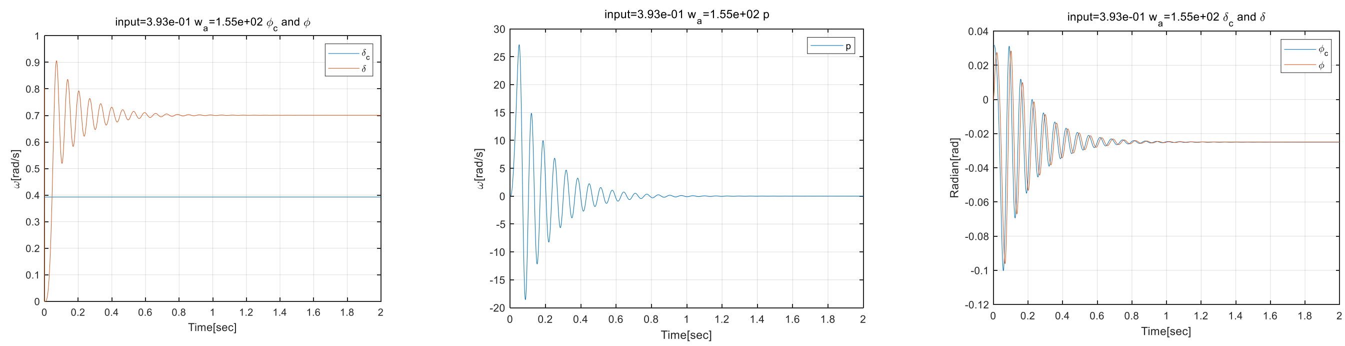

Figure 8. $\omega_a = 24.6[rad/s]$, $\phi_c = \frac{\pi}{8}$

$\phi_c = \frac{\pi}{8}$ 인 경우 시뮬레이션과 비슷한 결과를 보이며, 진동의 크기 및 수렴과 발산의 형태는 동일하나 steady state 상태에 도달하는 시간에서 차이를 보인다. 그러나 출력이 입력인 $\frac{\pi}{8}$=0.3927 에 근사하지 않고 0.7006 에 근사한다. 이 경우 또한 ω_a가 커짐에 따른 분석은 앞과 동일하며, $\omega_a = 16.4[rad/s]$ 인 경우에는 발산하지 않고 수렴하는 정도가 일정하여 일정한 크기의 진동이 발생하는 출력파형이 나타난다.

- $\phi_c = 0$ 인 경우 결과

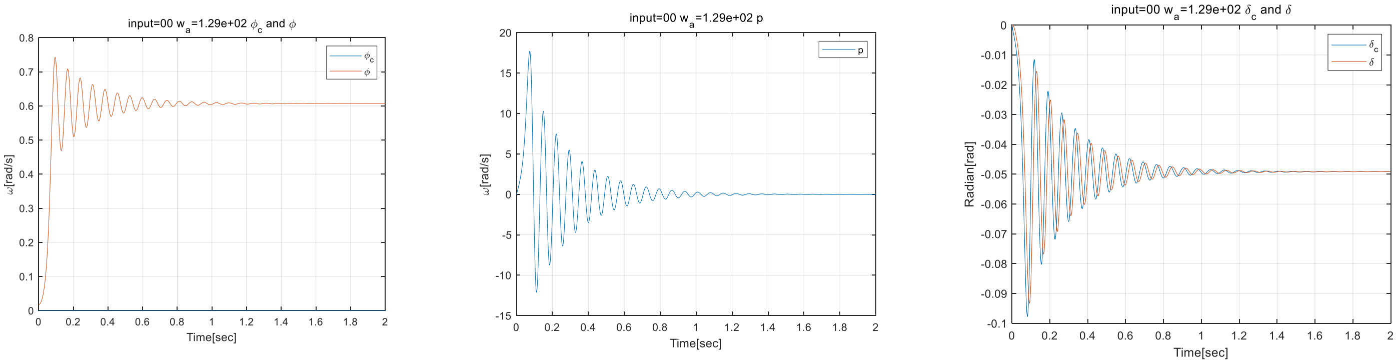

Figure 6. $\omega_a = 20.5[rad/s]$, $\phi_c = 0$

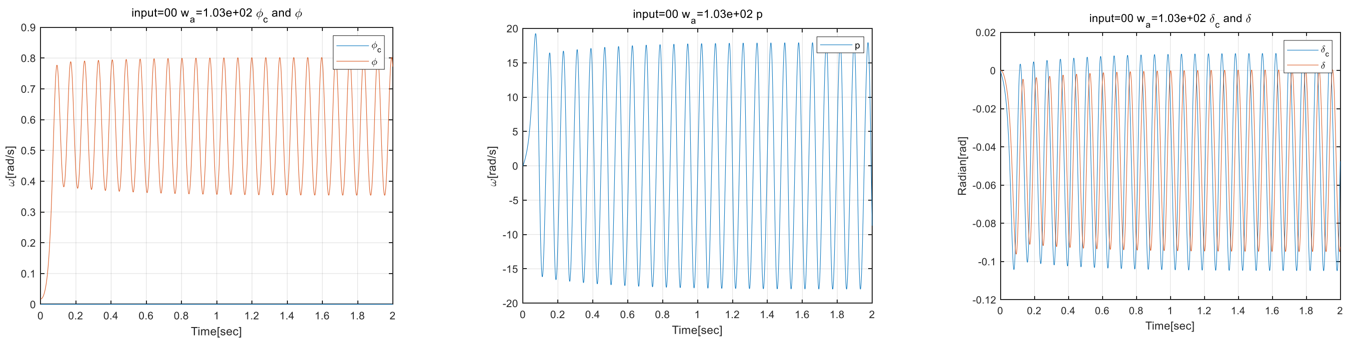

Figure 6. $\omega_a = 16.4[rad/s]$, $\phi_c = 0$

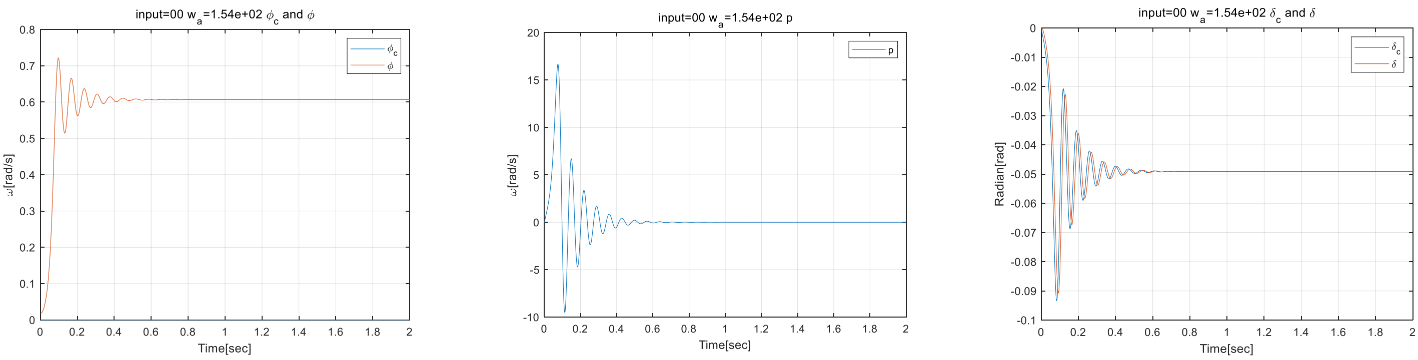

Figure 6. $\omega_a = 24.6[rad/s]$, $\phi_c = 0$

$\phi_c = 0$인 경우는 진동의 크기와 steady state 상태에 도달하는 시간은 비슷하나 출력의 부호가 반대됨을 보인다. 이 경우에도 위의 분석과 마찬가지로 $\omega_a = 16.4[rad/s]$ 일 때는 일정한 크기로 진동을 지속하나 다른 두 경우에는 0.6067로 수렴한다.