Aircraft Pitch: System Modeling

MMAE 443 Systems Analysis and Control

Zain Baig, Caden Gigstad, Mauricio Verduzco Chavira

Table of Contents

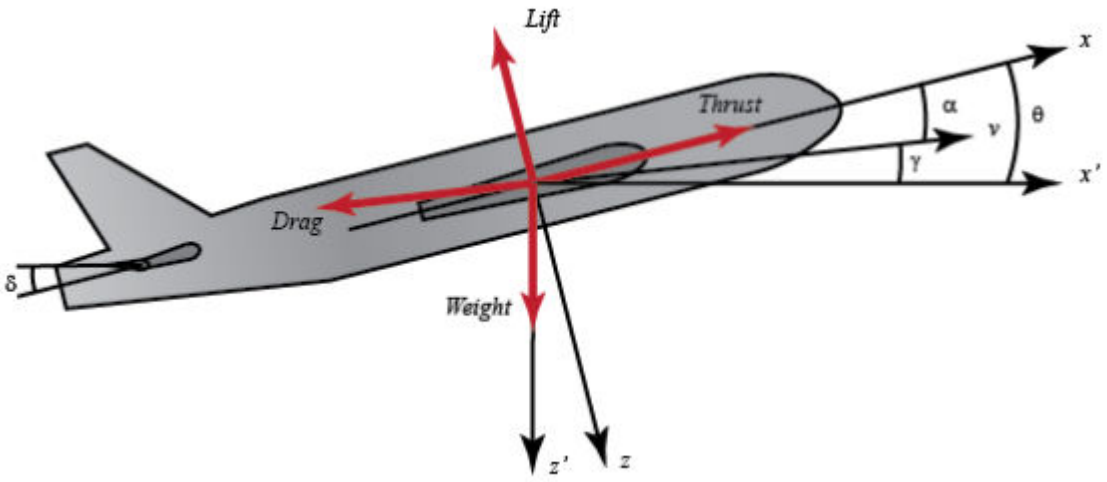
About the Model:

Model Description and Assumptions

In this example we are designing an autopilot controller for one of Boeing's commercial aircraft. Assumptions are:

- Steady cruise in constant altitude and velocity
- Thrust, drag, weight and lift balance each other out.
- Change in pitch angle will not change the speed of the aircraft

The model is linearized and only analyzes the longitudinal perspective.



Mathematical Modeling

Equations of Motion

$$\dot{\alpha} = \mu\Omega\sigma\left[-(C_L+C_D)\alpha + \frac{1}{(\mu-C_L)}q - (C_W\sin\gamma)\theta + C_L\right]$$

$$\dot{q} = \frac{\mu\Omega^2}{2I_{yy}}[[C_M - \eta(C_L+C_D)]\alpha + [C_M + \sigma C_M(1-\mu C_L)]q + (\eta C_W\sin\gamma)\delta]$$

$$\dot{\theta} = \Omega q$$

Plugging in some values

For this system, the output will be the pitch angle θ, and the input the elevator angle δ. The rest of the variables are explained at the bottom of the document. For simplicity, the values provided by University of Michigan Control Tutorials were used for Matlab and Simulink. As numerical values are added, the following is obtained:

$$\dot{\alpha} = -0.313\alpha + 56.7q + 0.232\delta$$

$$\dot{q} = -0.0139\alpha - 0.426q + 0.0203\delta$$

$$\dot{\theta} = 56.7q$$

Laplace Transform

Perform Laplace Transform of these equations to obtain:

$$sA(s) = -0.313A(s) + 56.7Q(s) + 0.232\Delta(s)$$

$$sQ(s) = -0.0139A(s) - 0.426Q(s) + 0.0203\Delta(s)$$

$$s\Theta(s) = 56.7Q(s)$$

Transfer Function

After some more algebra we obtain the transfer function:

$$G(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s}$$

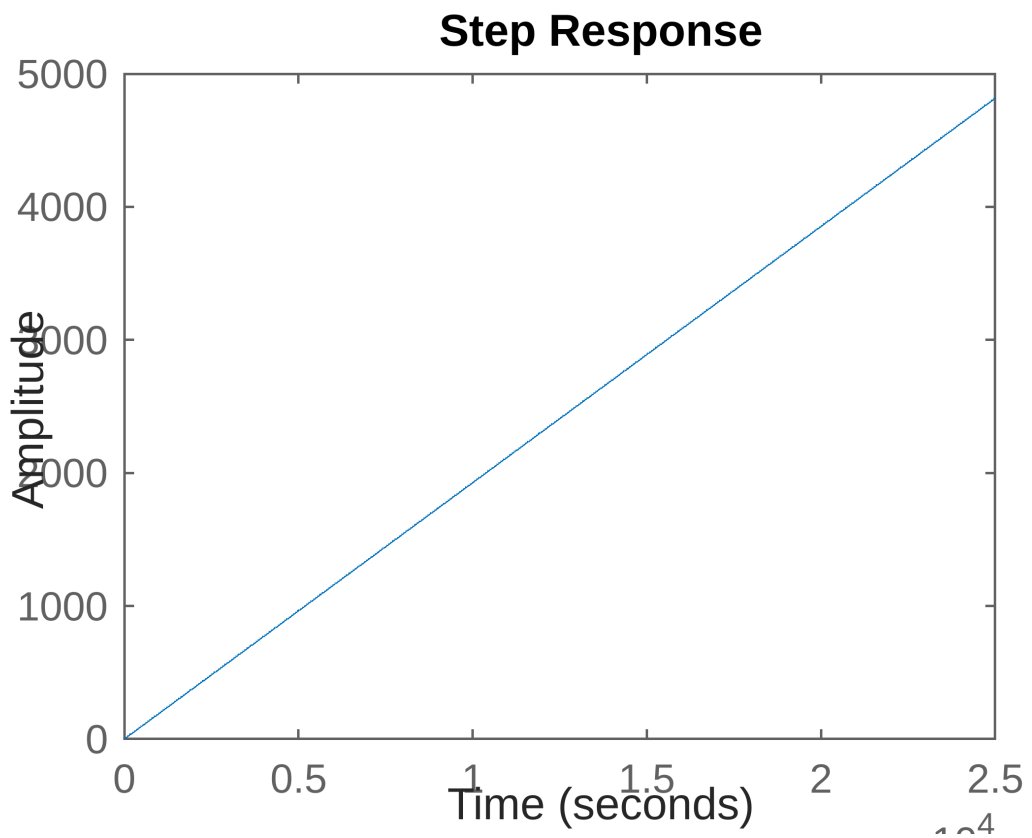
The specific algebraic steps and substitutions can be consulted at the end of the document.

Analysis of step response for plant

Step Response

And we study the step response

```
G =

       1.151 s + 0.1774
  -------------------------
  s^3 + 0.739 s^2 + 0.921 s

Continuous-time transfer function.
Model Properties
```



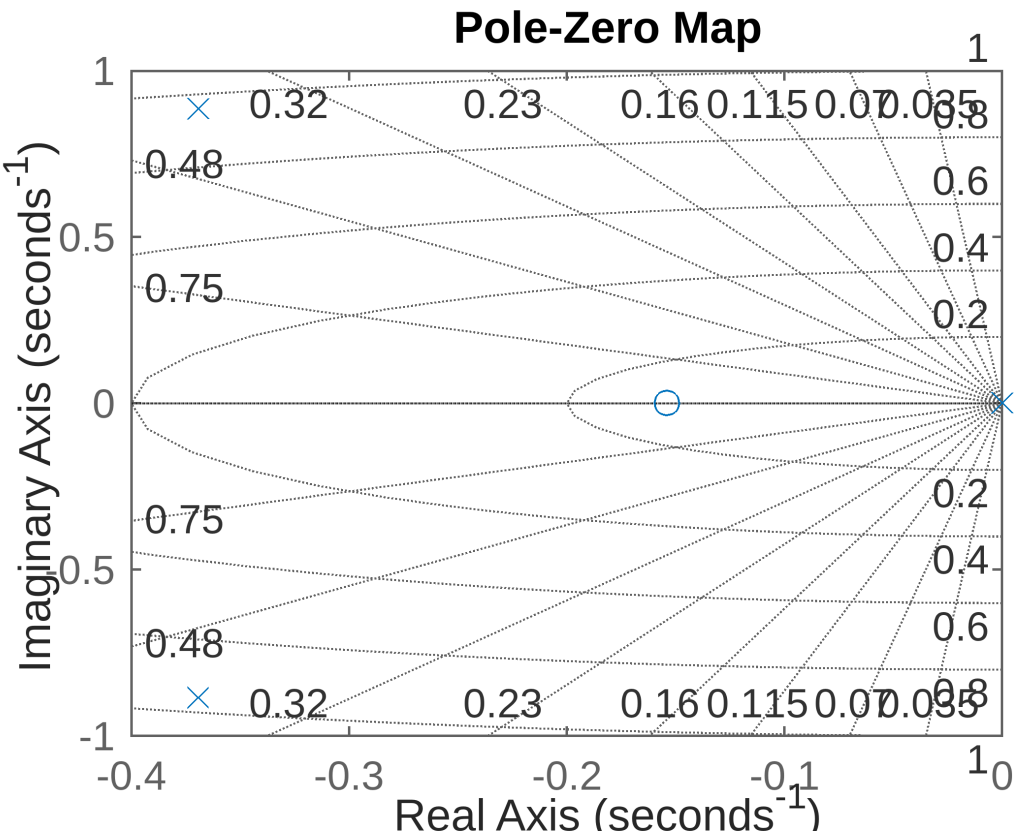
It is evident that the system is not stable. This is because one of the poles of the transfer function will be zero. This means that the best possible result is a marginally stable system.

Stability Analysis

The following shows the location of the poles.

```
poles = 3×1 complex
   0.0000 + 0.0000i
  -0.3695 + 0.8857i
  -0.3695 - 0.8857i
```

And graphically:



Results

The model cannot have static gain nor overshoot because of the instability of the system. Even if the values of the original variables were changed, the shape of the denominator polynomial

$$s^3 + a*s^2 + b*s = 0$$

Will always yield a pole equal to zero as a common *s* can be factored out.

Performance Requirements

For a step reference of 0.2 radians (everything up to 1 unit)

$$t_{ss} \leq 1($$

$$\%OS \leq 10\%$$

$$ss \leq 2\%$$

Feedback

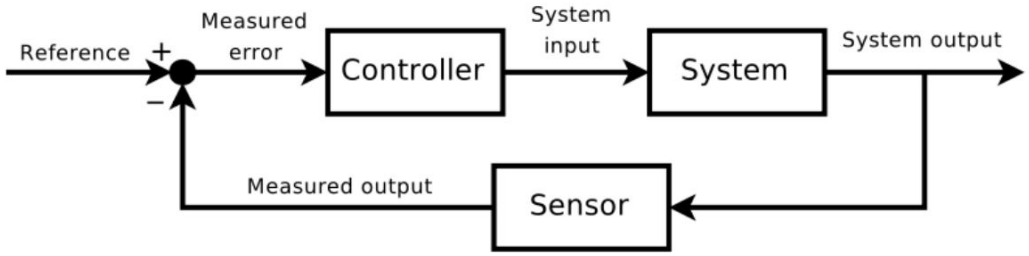
Original System

We had the original system given by:

```
G =

      1.151 s + 0.1774
  ------------------------
  s^3 + 0.739 s^2 + 0.921 s

Continuous-time transfer function.
Model Properties
```

Feedback Equation Modeling

If feedback is added as follows:



We get a new ecuation given by.

$$Y(s) = \frac{G(s)}{1 + G(s)H(s)}$$

Where Y is the output, G is the original plant and H is the sensor.

Sensor Transfer Function

```
Ysym =

```

$$\frac{1.151e+3s + 1.774e+32}{1.774e+32 s^3 + 1.921e+32 s^2 + 1.151e+33 s + 7.39e+32 s^2 + 1.0e+33}$$

Perfect Sensor

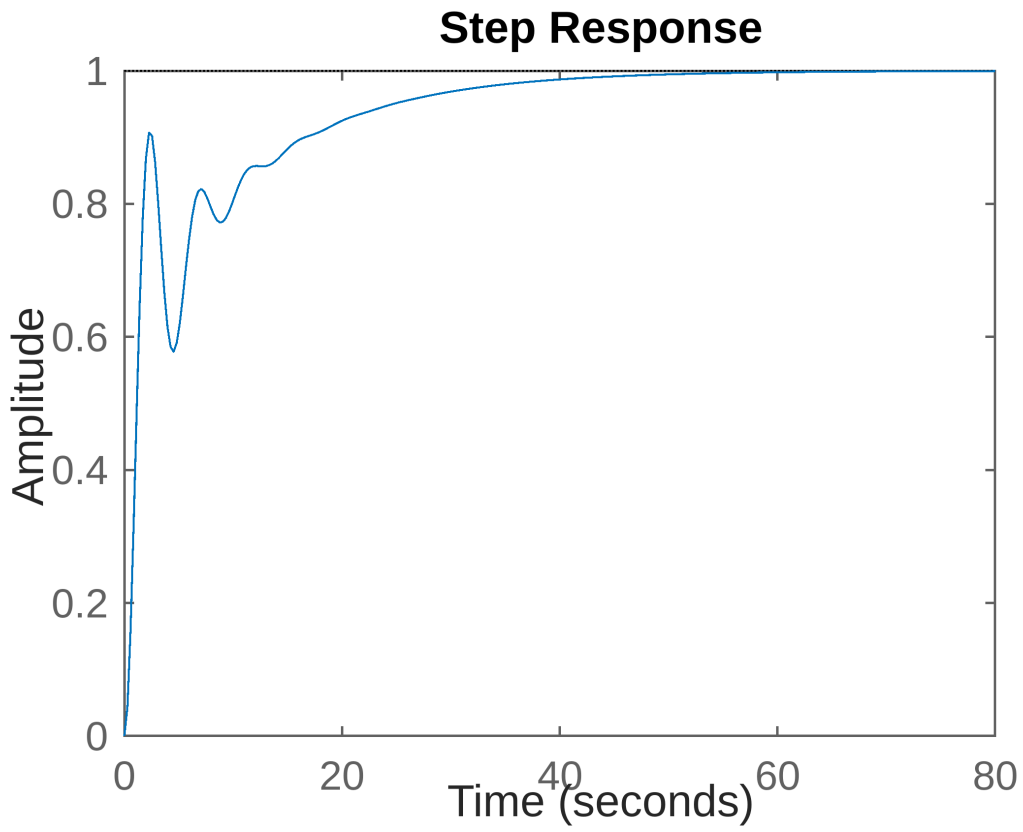
In this case, a perfect sensor is considered, meaning H = 1 and that there is no proportional constant.

Perfect Sensor - Transfer Function

```
Ysym =

```

$$\frac{1.151e+3s + 1.774e+32}{1.0e+33 s^3 + 7.39e+32 s^2 + 2.072e+33 s + 1.774e-}$$

Perfect Sensor - Step Response



Perfect Sensor - Step Information Analysis

A stable system is now obtained. Unfortunately, the system has a lot of oscillation and long settling time.

```
ans = struct with fields:
          RiseTime: 1.7882
     TransientTime: 35.0896
      SettlingTime: 35.0896
       SettlingMin: 0.5777
       SettlingMax: 0.9998
         Overshoot: 0
        Undershoot: 0
              Peak: 0.9998
          PeakTime: 90.2716
```

Non-Perfect Sensor

A non-perfect sensor in most systems is defined as

$$H(s)=\frac{K_h}{\tau*s+1}$$

Where Kh is an adjustable constant, and τ is a time delay constant for the sensor.

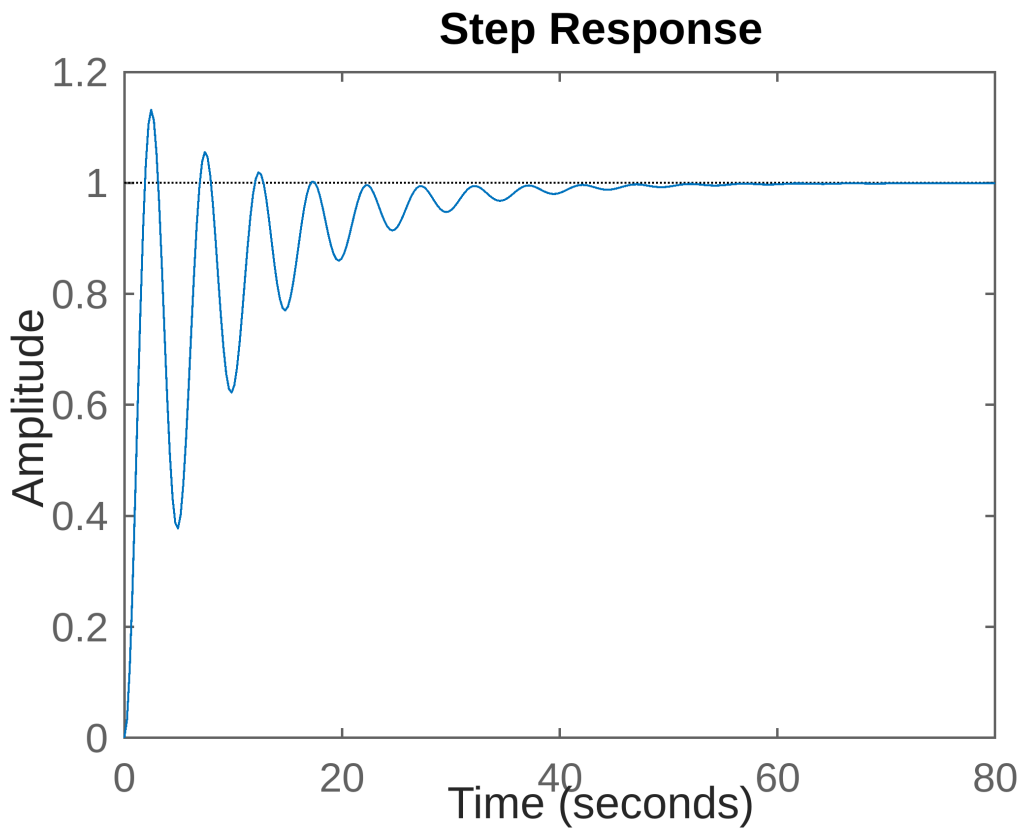
Non-Perfect Sensor Transfer Function

Utilizing the original plant and a delay of 0.5, the following transfer function is obtained:

```
Ysym =
```

$$\frac{1.151e+32 s^2 + 2.4794e+33 s + 3.548e+32}{1.0e+33 + 2.739e+33 s + 2.399e+33 s^2 + 4.144e+33 s^3 + 3.548e+}$$

Non-Perfect Sensor Step Response



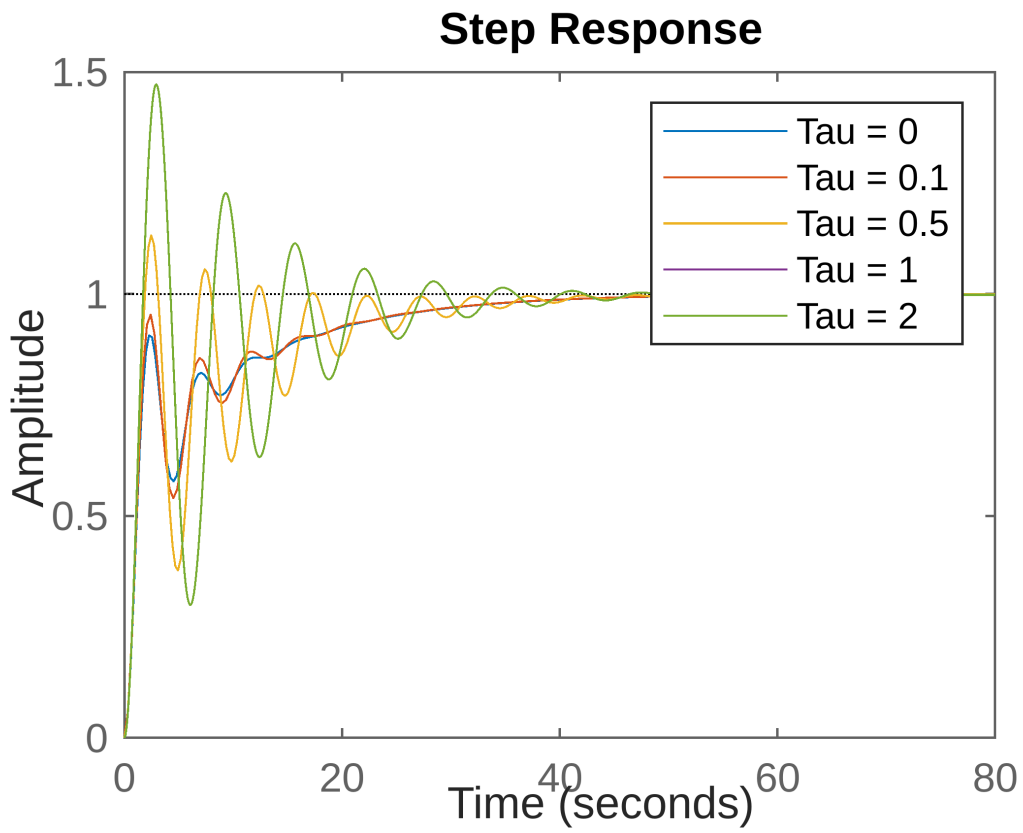
Non-Perfect Sensor Step Information Analysis

```
ans = struct with fields:
          RiseTime: 1.2679
     TransientTime: 35.6863
      SettlingTime: 35.6863
       SettlingMin: 0.3773
       SettlingMax: 1.1320
         Overshoot: 13.1989
        Undershoot: 0
              Peak: 1.1320
          PeakTime: 2.4636
```

Results

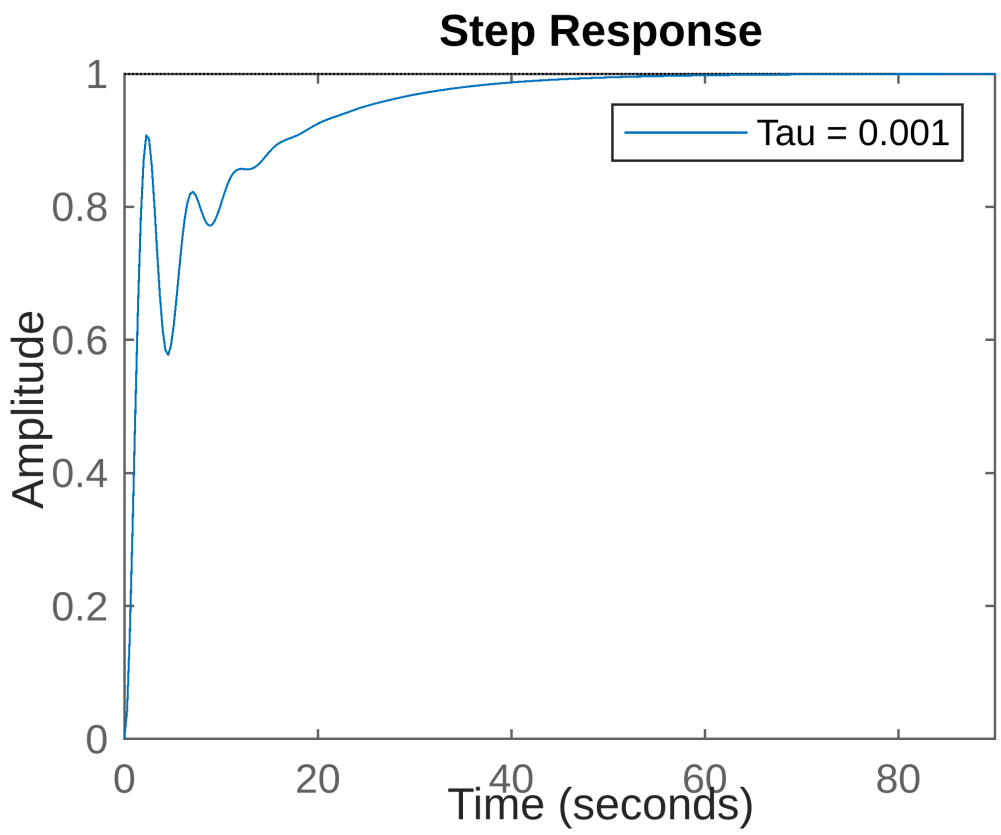
As can be seen in the step response, the system response is not satisfactory. There are many oscillations and a long settling time. This means further control must be added to gain a better system response.

The impact of different Tau configuration values in the performance



Observing various time constant values, the best possible result is given by = 0. However, there is no real sensor that gives a response with zero delay. Doing some research, it was learned that most gyro sensors take about 1000 samples per second, meaning that the time delay of a gyro sensor is about 0.001.

Final Configuration for Feedback



Results

```
ans = struct with fields:
          RiseTime: 1.7867
     TransientTime: 35.0877
      SettlingTime: 35.0877
       SettlingMin: 0.5774
       SettlingMax: 0.9999
         Overshoot: 0
        Undershoot: 0
              Peak: 0.9999
          PeakTime: 96.1018
```

The system does not meet the desired requirements, but has improved from the initial system that wasn't even stable. This is the best result considering only feedback. A controller will be added to further improve the result.

PID Controller Design

The Original System

```
G =
```

$$\frac{1.15s+0.1774}{s^3+0.739s^2+0.92s}$$

The system with the chosen feedback configuration

```
H =
```

$$\frac{1}{0.003s+1.0}$$

The new transfer function is obtained:

```
Ysym =
```

$$\frac{2.0(1.0e+3s+1.0e+6)(5.755e+3s+8.87e+3)}{1.0e+6s^4+1.001e+7s^3+7.399e+6s^2+2.072e+6s+1.774e+}$$

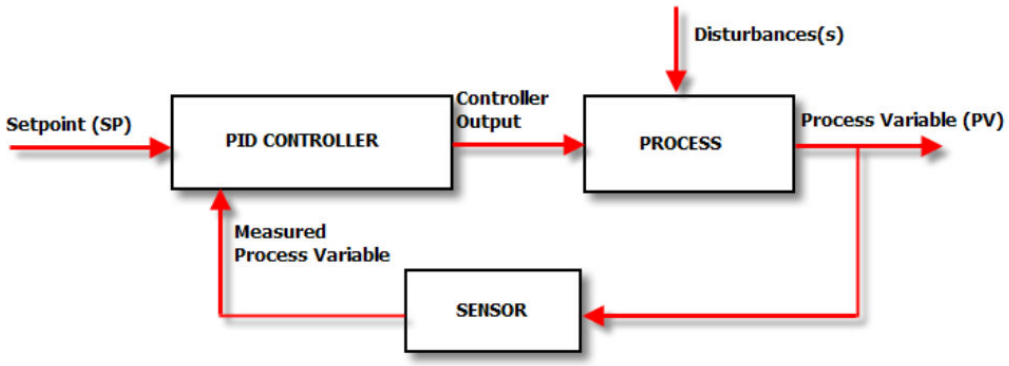
The PID controller

The PID controller has a proportional, integral and derivative term as follows:

$$C(s)=K_p+\frac{K_i}{s}+K_d*s$$

The PID controller equation modeling

When the controller is added as follows:



The new equation is given by:

$$Y(s)=\frac{C(s)G(s)}{1+C(s)G(s)H(s)}$$

Where Y is the output, G is the original plant, H is the sensor and C is the controller.

The PID controller Transfer Function

Modeling the controller into MatLab:

```
C =
```

$$Kp+Kd*s+\frac{K}{s}$$

The new transfer function is:

Ysym =

$$\frac{2.0(1.0e+3+1.0e+3(5.755e+3+3.87e+3)s^2K_ds^2+K_ps+K_i)}{1.774e+6K_6+1.151e+6K_7+1.774e+6K_p+1.774e+6K_ds^2+1.151e+6K_7s^3+1.151e+6K_ps^2+9.21e+5s^6+7.399e+5s^6+1.001e+6s^6}$$

Adding Pole Placement

This objective of pole placement is to know which poles would give the ideal result and perform reverse engineering to implement it into the system. If the desired poles are known, the desired characteristic polynomial can be written. the current polynomial and desired polynomial can be compared to find the values for the variables K, a, and b that will yield the desired result.

As shown, the final equation will have 5 poles.

If the following poles are desired:

```
desiredPoles = 1×5
  -250  -225  -200  -175  -150
```

That would mean that the desired polynomial would be:

DesiredPolynomial = $s^5+1006s^4+396875s^3+7812500s^2+7626562500s+29531250($

The current polynomial is:

aux2 = $177.4K_i+1151K_is+177.4K_p+177.4K_ds^2+1151K_ds^3+1151K_ps^2+921.6s^3+739.9s^4+1001s^5+1.0s^5$

Matching coefficients:

$739.9+1151K_d=3968$

$921.6+1151K_p+177.4K_d=78125($

$1151K_i+177.4K_p=7626562$

Solving for Kd, Kp, and Ki:

Kd = 344

Kp = 67820

Ki = 6.616e

So our controller final setup:

C =

$$344.2s+\frac{6.616e+6}{s}+67820($$

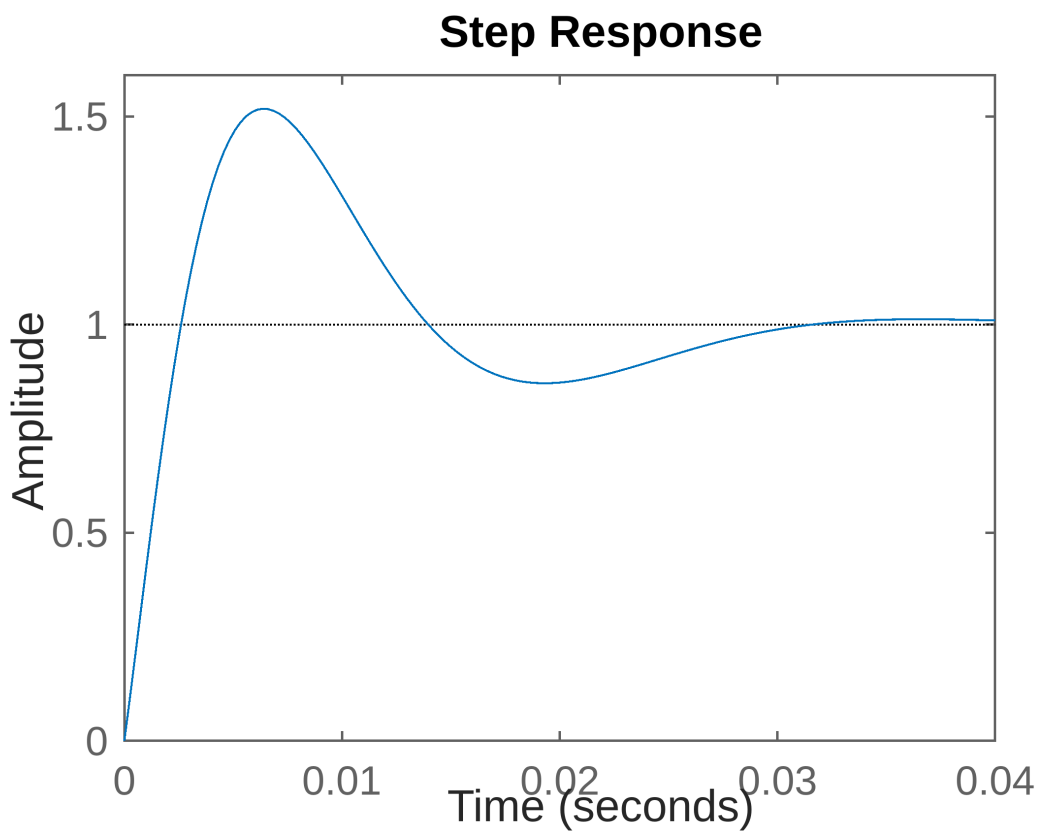
The new transfer function is:

```
Y =

        3.961e33 s^4 + 4.743e36 s^3 + 8.575e38 s^2 + 7.628e40 s + 1.174e40
  ------------------------------------------------------------------------
  1e31 s^5 + 1.001e34 s^4 + 3.969e36 s^3 + 7.813e38 s^2 + 7.627e40 s + 1.174e40

Continuous-time transfer function.
Model Properties
```

The roots of this system are:

```
ans = 5×1 complex
10^2 ×
  -3.6225 + 1.1176i
  -3.6225 - 1.1176i
  -1.3804 + 1.8420i
  -1.3804 - 1.8420i
  -0.0015 + 0.0000i
```

Step response:




```
ans = struct with fields:
         RiseTime: 0.0020
    TransientTime: 0.0291
     SettlingTime: 0.0291
      SettlingMin: 0.8593
```

```
SettlingMax: 1.5188
  Overshoot: 51.8756
 Undershoot: 0
       Peak: 1.5188
   PeakTime: 0.0064
```

Note: Why were these specific roots chosen?

Examining the shape of the polynomial that will be managed with Kp, Ki & Kd:

$s^5+as^4+bK_1s^3+cK_2s^2+dK_3s+eK_4$

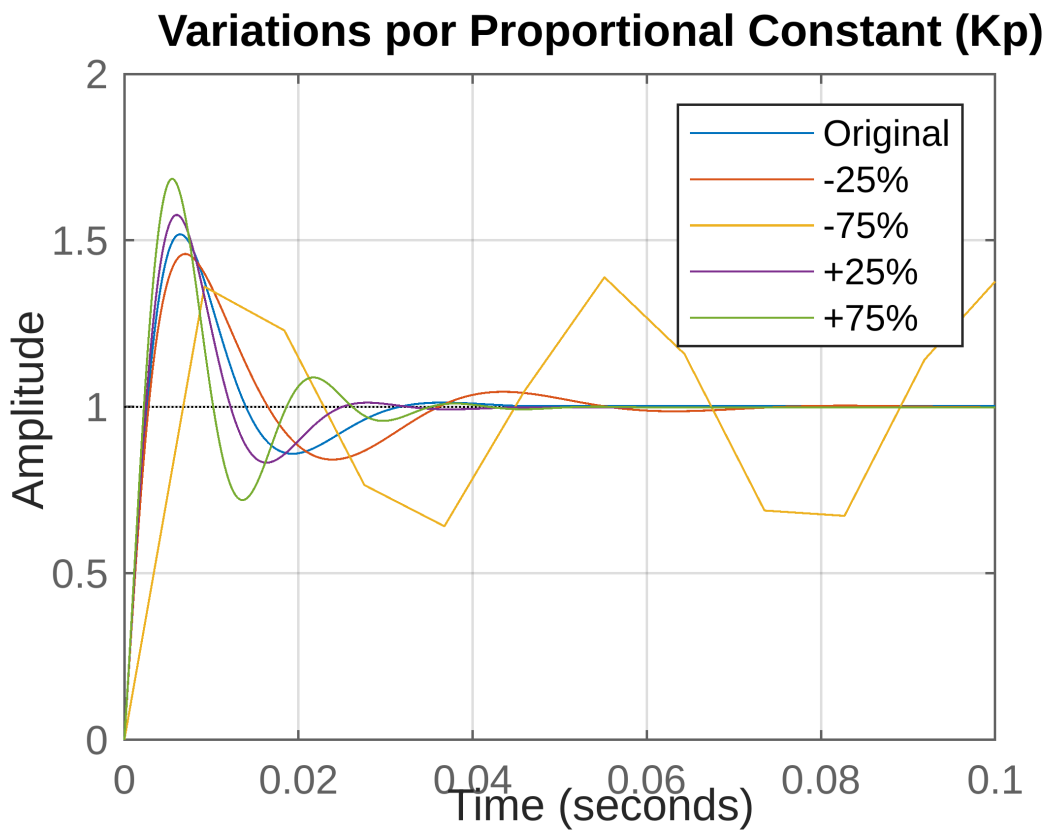
With a, b, c, d and e as some real constants obtained by the shape and values of our plant, sensor and controller

And K_i as some different mix of coefficients of the proportional, integral and derivative which we will obtain to match our desired shape because of the pole placem

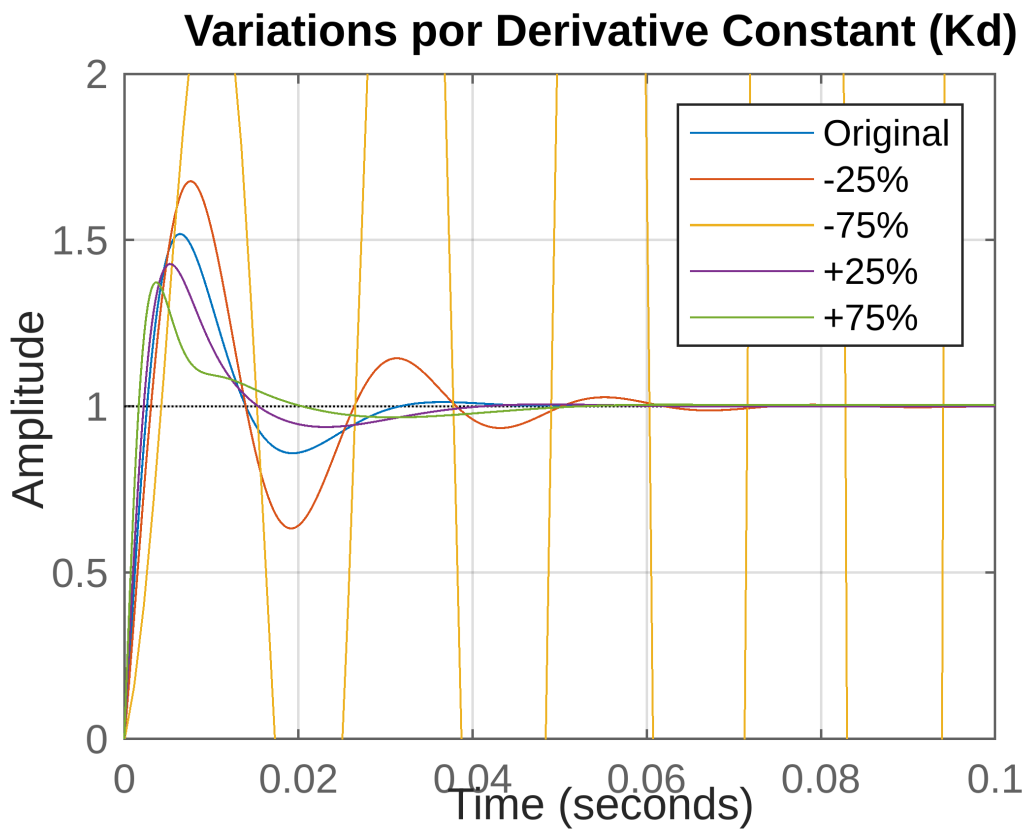
The problem is that the value of "a" cannot be changed by modifying the controller's coefficients. The only solution is to match the *s*4 coefficient of the desired polynomial with the current polynomial. The natural coefficient of the *s*4 term is about 1000. The roots were picked so the polynomial of those roots would match this, no change K for the other *s* terms is needed. There 4 equations and 3 variables, so one of them will not be satisfied. Therefore, the desired polynomial will not be obtained, and complex results are likely. This would mean that steady state and settling time will be satisfied, but not overshoot.

Evaluating different values as K's

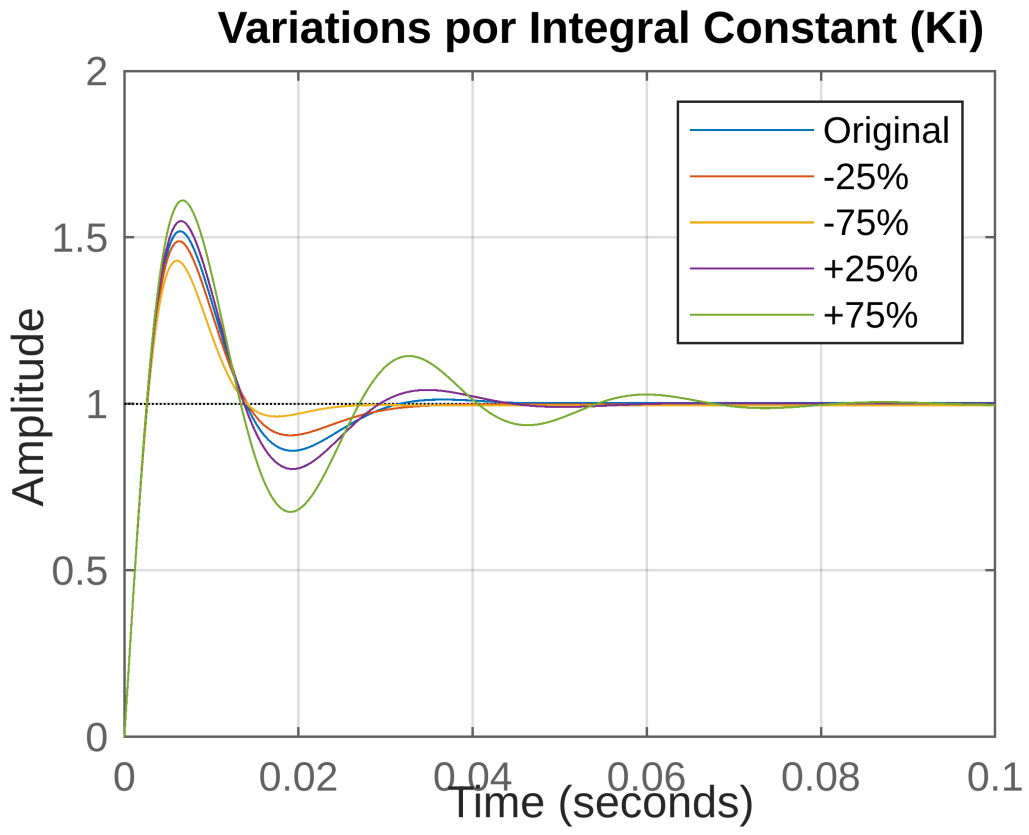
Sensibility for the Proportional Kp



Sensibility for the Derivative Kd



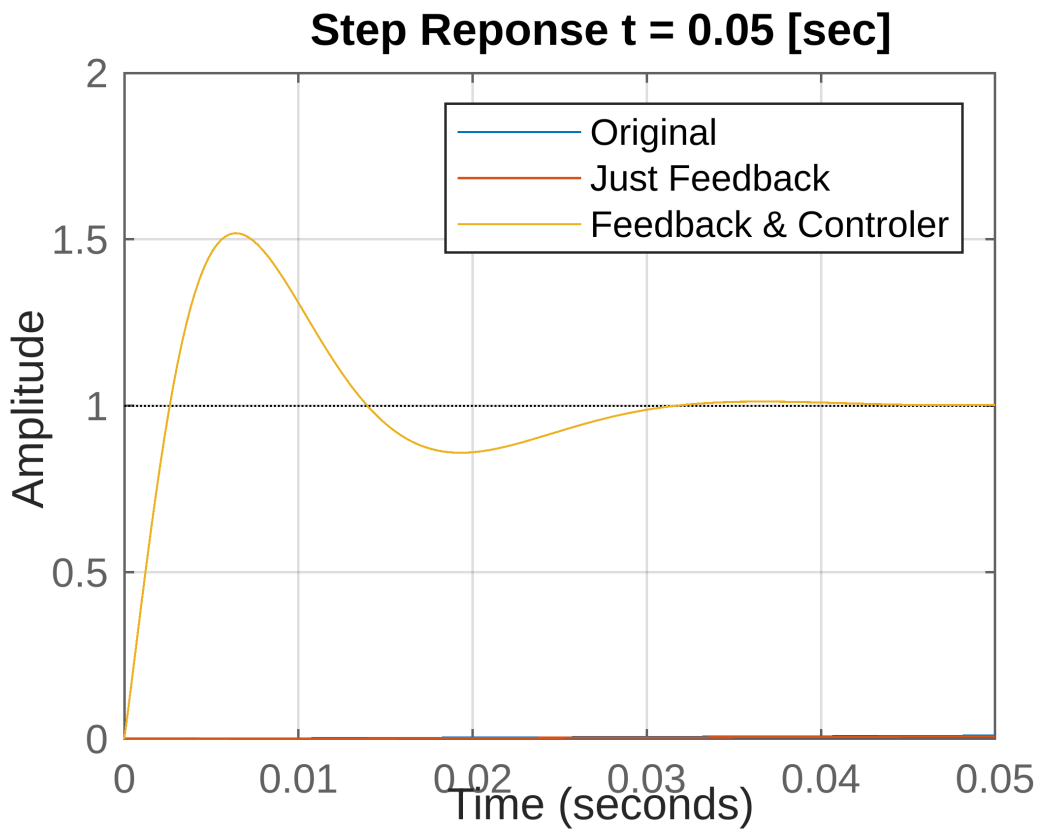
Sensibility for the Integral Ki

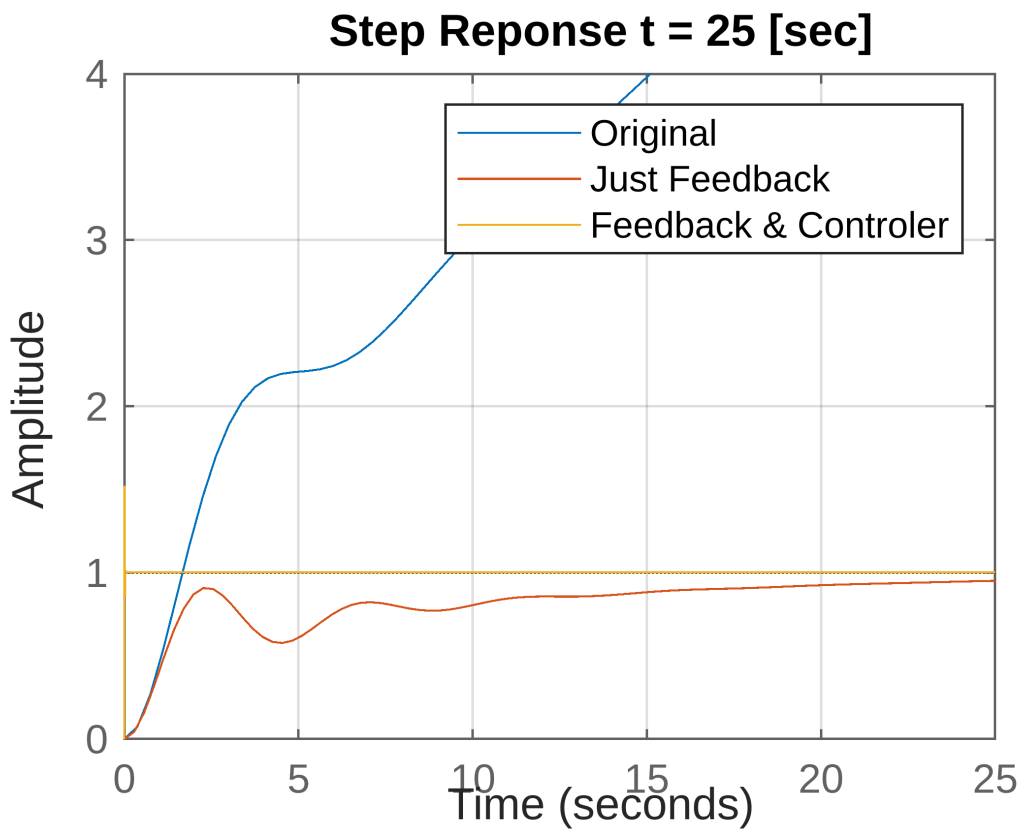


This analysis helps us realize that the configuration obtained from pole placement is the best we will possible get.

The results so far

Step response for different systems:





The yellow line has a very small settling time. This is a lot of improvement when compared to the red and blue curves.

```
ans = struct with fields:
          RiseTime: 0.0020
     TransientTime: 0.0291
      SettlingTime: 0.0291
       SettlingMin: 0.8594
       SettlingMax: 1.5187
         Overshoot: 51.8702
        Undershoot: 0
              Peak: 1.5187
          PeakTime: 0.0064
```

All requirements are met from the system except for the overshoot requirement. The very small settling time and no steady state error somewhat compensate for the extra overshoot. However, extra measures will be taken to test if overshoot can be mitigated.

Root Locus Plots

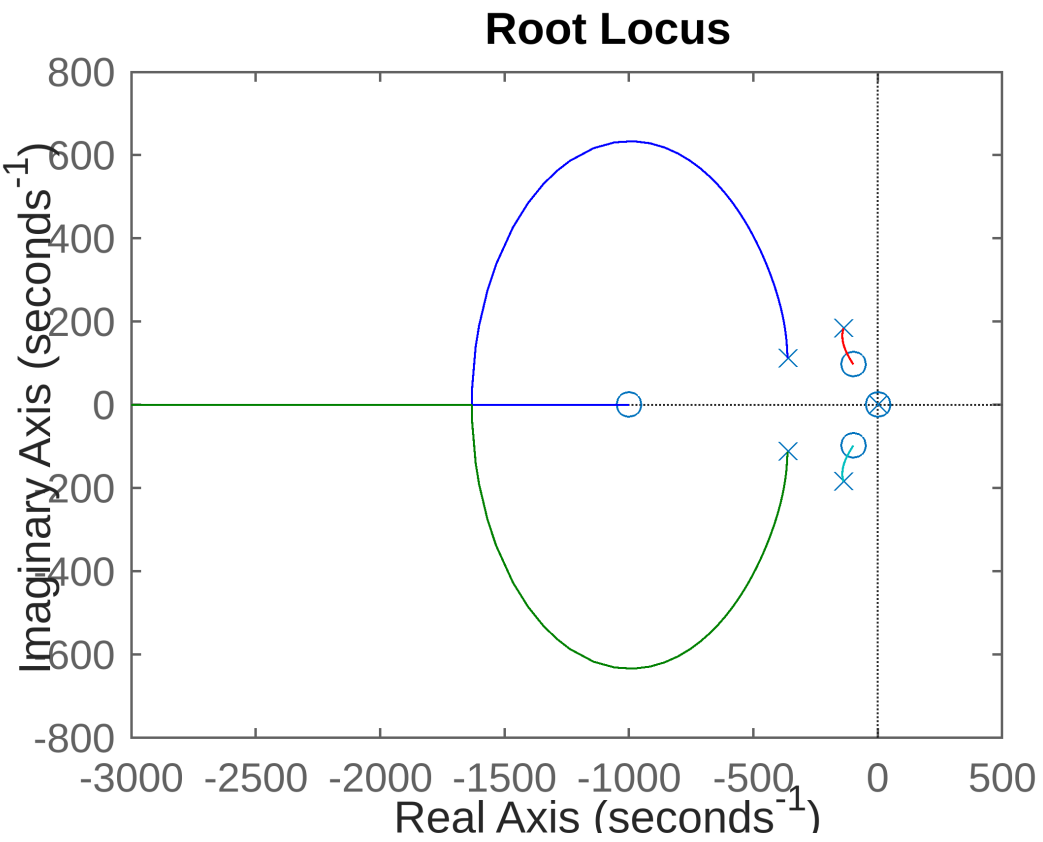
The system is not fully operational because the overshoot requirement has not been met.

Analyzing the roots of the system, complex numbers are present.

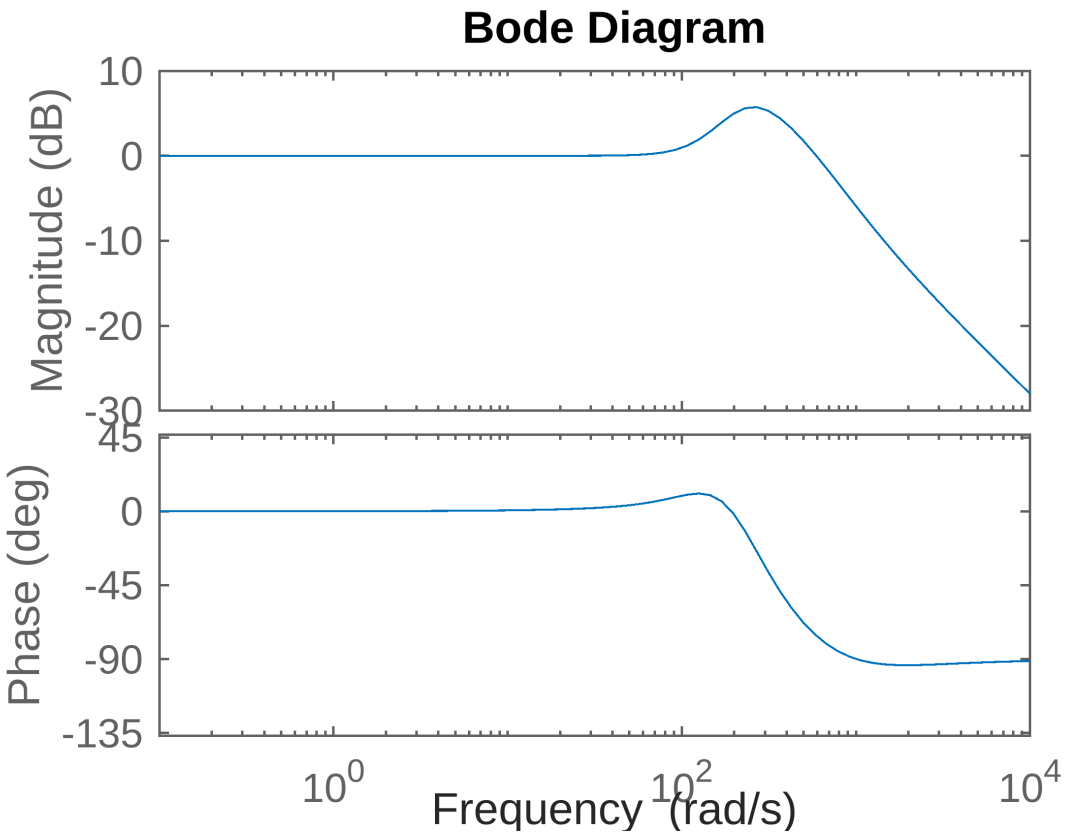
```
ans = 5×1 complex
10^2 ×
  -3.6227 + 1.1208i
```

```
-3.6227 - 1.1208i
-1.3802 + 1.8412i
-1.3802 - 1.8412i
-0.0015 + 0.0000i
```

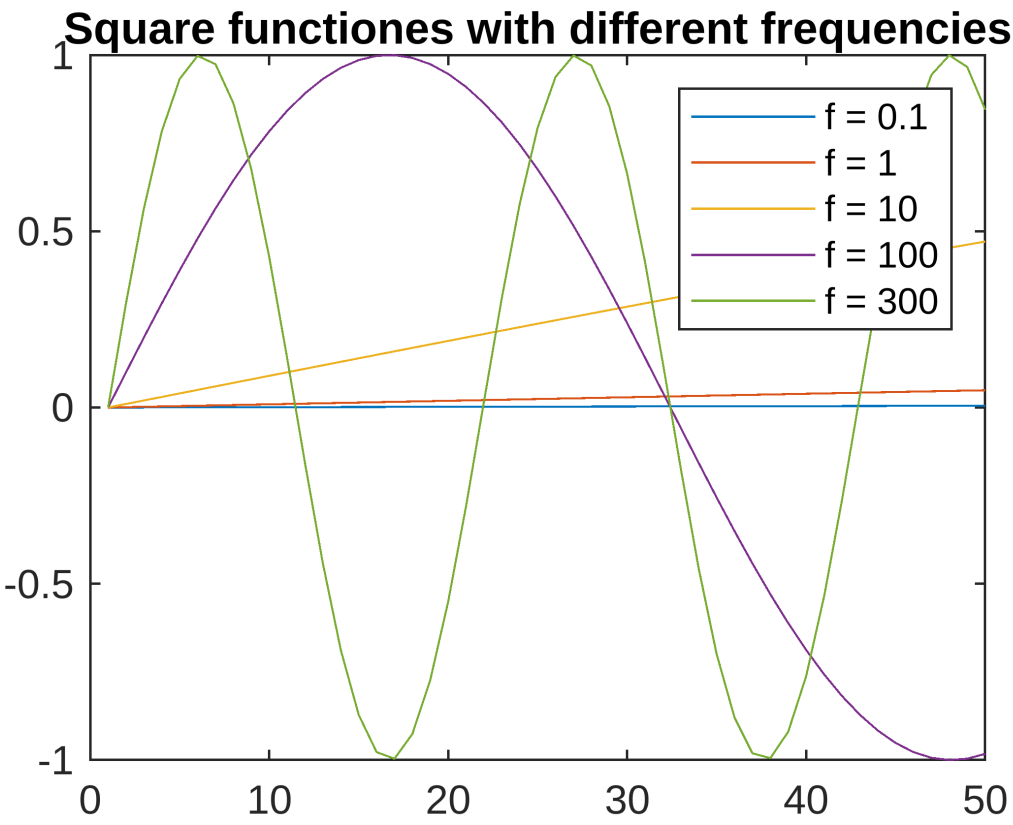
Plotting the root locus it is evident that changing values of K's cannot have them exist exclusively in the real axis. Therefore, this is the best possible result.



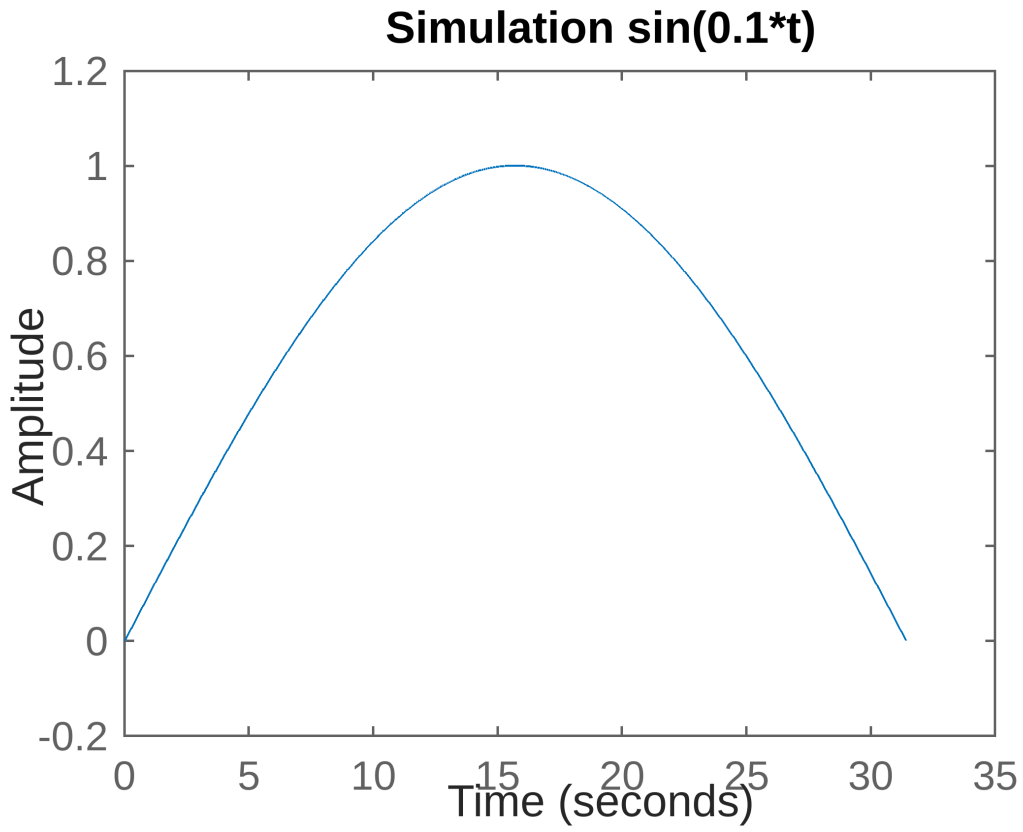
Analysis of Frecuency Reponse

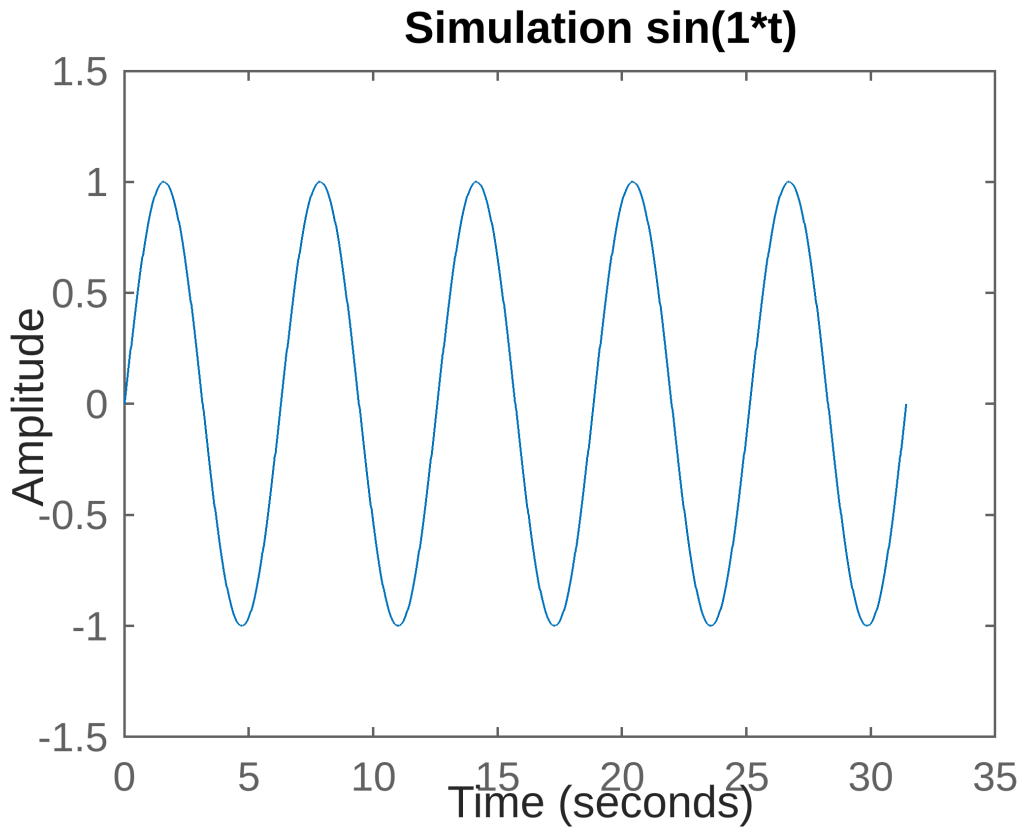


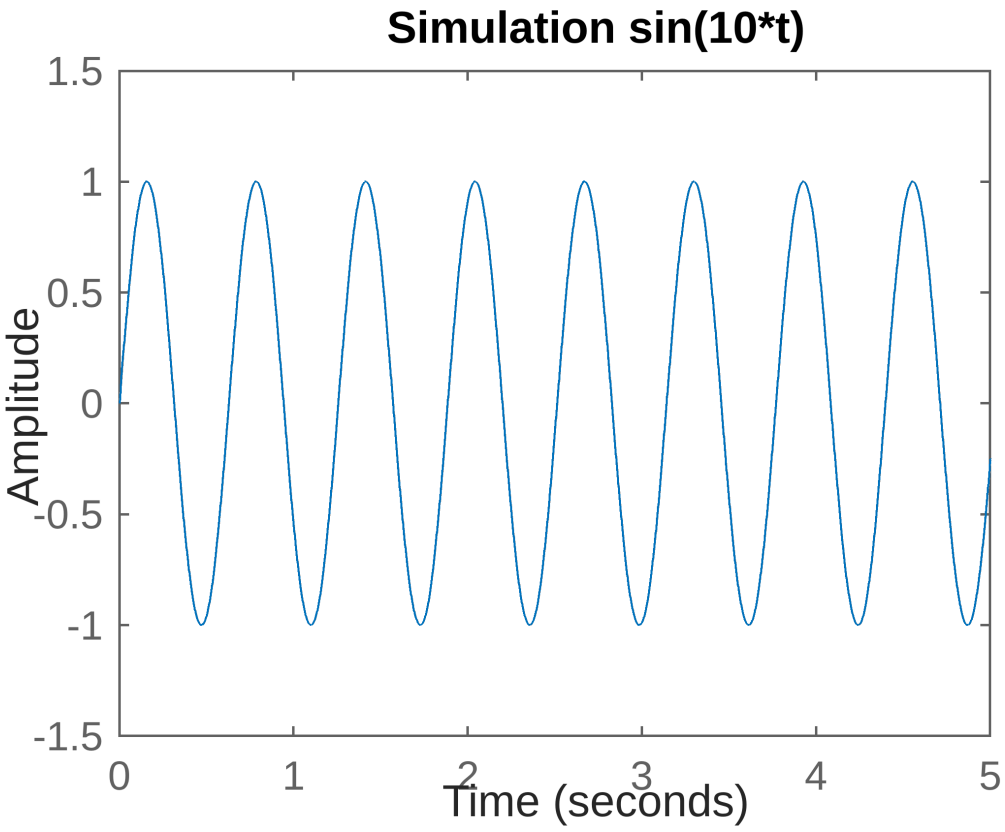
This analysis studies how the system would respond to different frequencies. The Bode diagram shows that the system should be fully operational in frequencies from 0.1 to 100 rads/sec. Frequencies greater than this will result in an inconsistent system. Also the phase will start shifting.

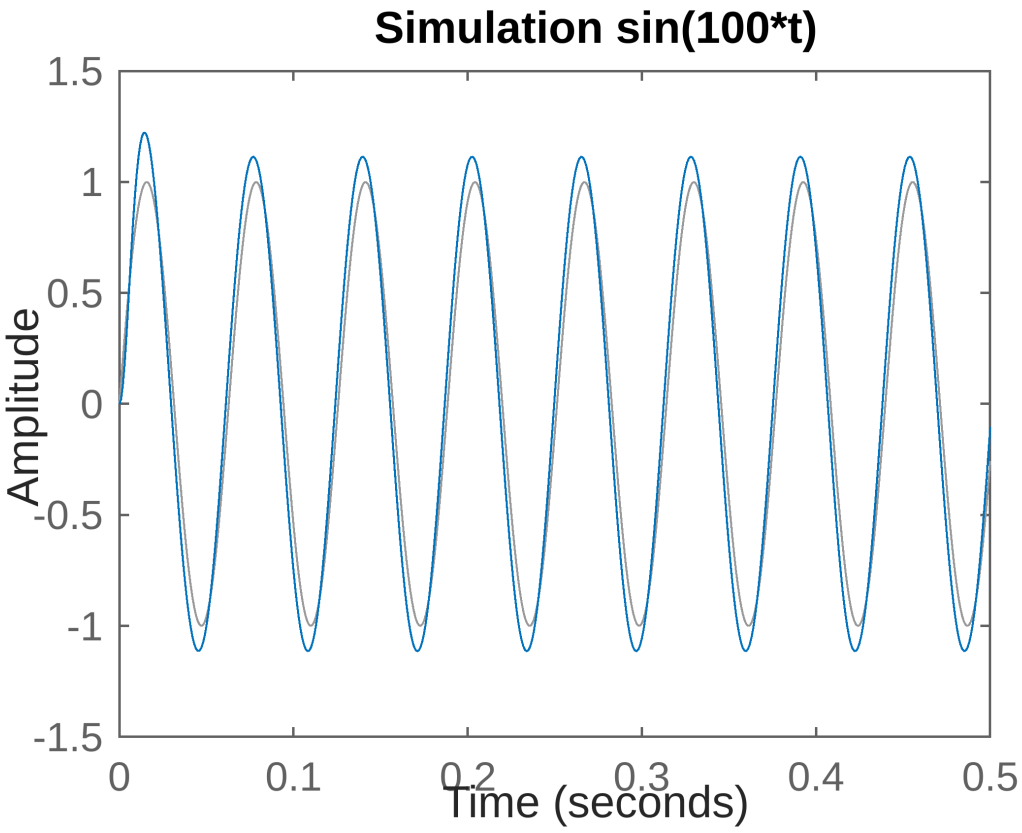


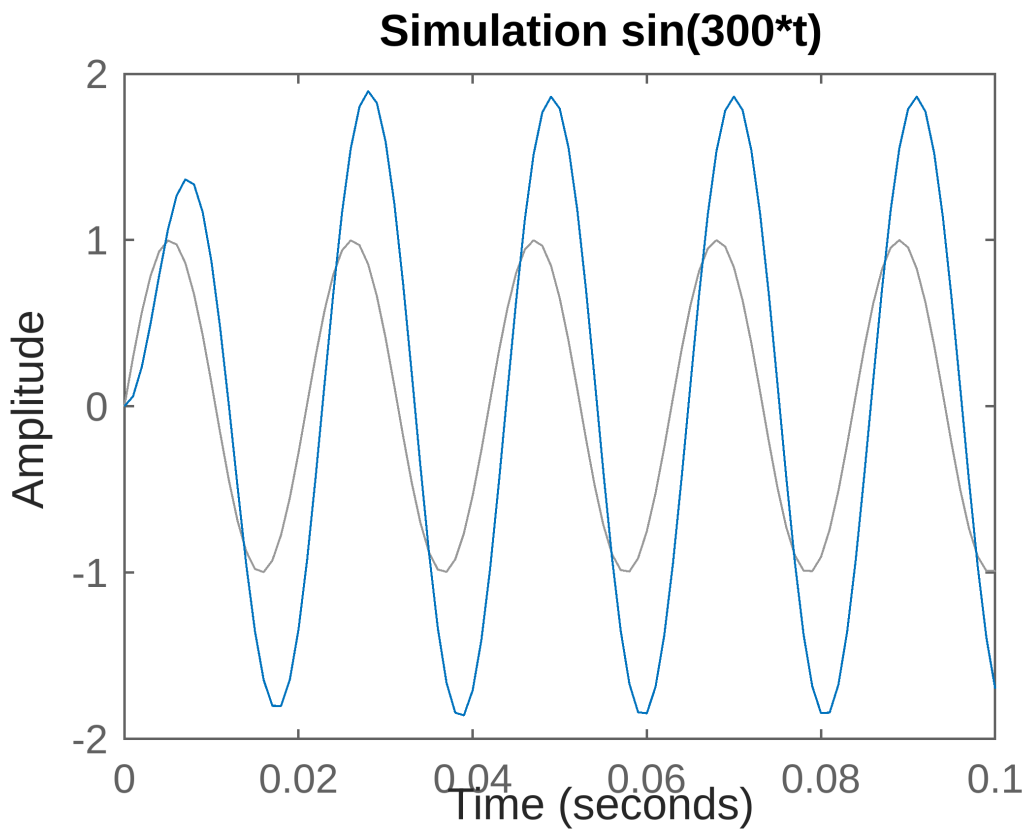
Considering multiple inputs, the green line will not work on our system due to the high frequency:


Simulation sin(0.1*t)
Amplitude
Time (seconds)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25
30
35


Simulation sin(1*t)
Amplitude
Time (seconds)
1.5
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25
30
35


Simulation sin(10*t)
Amplitude
Time (seconds)
1.5
1
0.5
0
-0.5
-1
-1.5
0
1
2
3
4
5


Simulation sin(100*t)
Amplitude
Time (seconds)
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.1
0.2
0.3
0.4
0.5



As can be seen, all linear simulations didn't distinguish between the input "grey" and the output "blue" until the last two plots. The frequency of 100 rad/sec is starting to get a bit of phase shifting and some unclear magnitude. But when at 300 rad/sec the magnitude and phasing goes terribly bad.

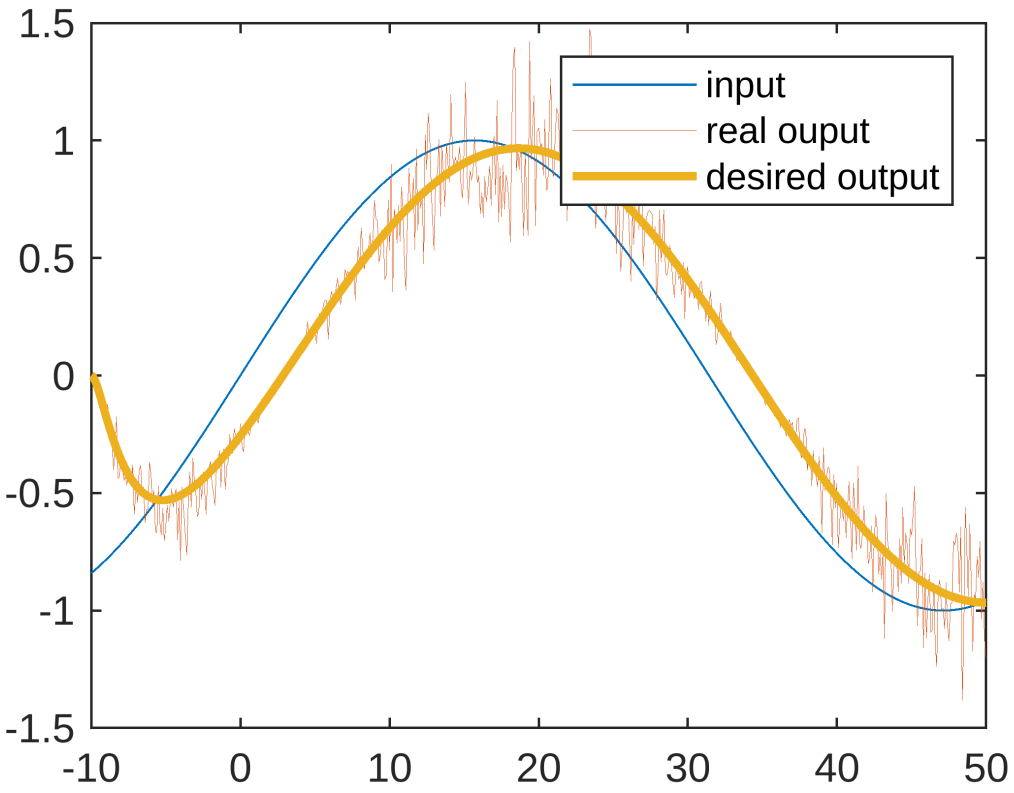
This confirms the behavior observed on the Bode plot, the operation range for our system goes from frequencies 0.1 to 100 rads/sec.

Machine-Learning Controler

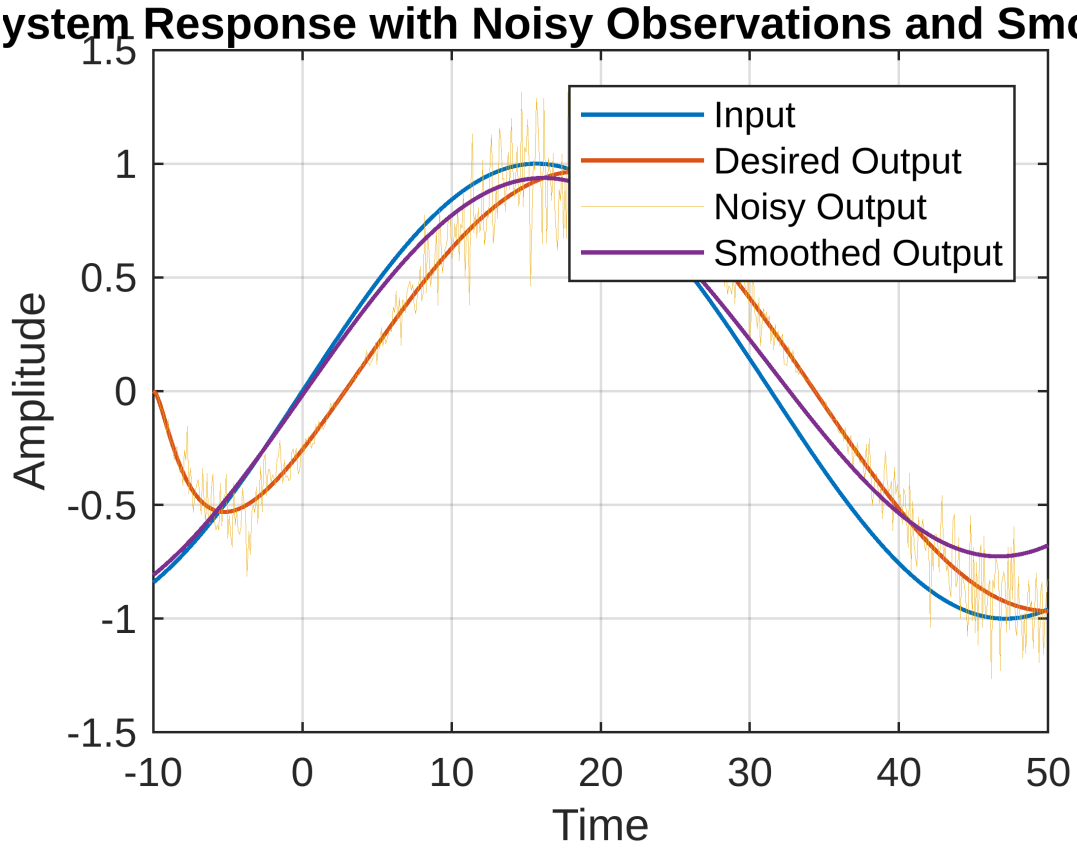
Assume a system with perfect feedback. It has a simple input but an unknown disturbance is added to the response. To solve this, a machine learning approach can be taken. For this scenario, a discrete approach can be taken, assuming the input is a mere unit step function. The output should also be a step response but somehow is not.

For this example, a simple upwards waveform simulates the increase of elevation at the lift of a flight, peak altitude, and then descent for landing.

```
G =

         s + 1
  ---------------------
  s^3 + 4 s^2 + 4 s + 1

Continuous-time transfer function.
Model Properties
```



As can be seen, noise is present in the real output, but the end desired result output is the step response. Performing linear regression over the output allows for the approximation of the result.



To conclude, with the machine learning approach, linear regression managed to smooth out the output, dealing with the uncertainty of a noisy channel.

Appendices

First: Variable summary

α = Angle of attac

q = Pitch rate

θ = Pitch ang

δ = Elevator deflection ang

$\mu = \frac{\rho S c}{4m}$

$\rho = 1.125 \frac{kg}{m^3}$ = Density of ai

$S = 124 m^2$ = Platform area of the win

$c = 3.4 m$ = Average cord leng

$m = 79{,}000 - 41{,}145 kg$ = Mass of the aircra

$\Omega = \frac{2U}{c}$

$U = 450$ knots = Equilibrium flight speed

$C_T = 0.3 - 0.5$ = Coefficient of thru

$C_D = 0.02 - 0.04$ = Coefficient of dra

$C_L = 0.5 - 1.2$ = Coefficient of lift

$C_W = 1$ = Coefficient of weig

$C_M = 0.01$ = Coefficient of pitch mome

γ = Flight path ang

$\sigma = \frac{1}{1+\mu C_L}$ = consta

i_{yy} = Normalized moment of inert

$\eta = \mu\sigma C_M$ = consta

Second: Algebra

From equations of motion to Laplace Transform

We begin with our equations of motion

```
eqn1(t) =
```

$$\frac{d}{dt}\alpha(t) = \frac{29\theta(t)}{125} - \frac{313\alpha(t)}{1000} + \frac{567\gamma(t)}{10}$$

```
eqn2(t) =
```

$$\frac{d}{dt}q(t) = \frac{203\theta(t)}{10000} - \frac{139\alpha(t)}{10000} - \frac{213\delta(t)}{500}$$

```
eqn3(t) =
```

$$\frac{d}{dt}\theta(t) = \frac{567\gamma(t)}{10}$$

Then we do Laplace Transform and set initials conditions to zero

Because MATLAB does not support solving for laplace-type variables, we are going to substitute our own variables.

```
EQN1 =
```

$$As = \frac{29D}{125} - \frac{313A}{1000} + \frac{567\Gamma}{10}$$

```
EQN2 =
```

$$Qs = \frac{203D}{10000} - \frac{139A}{10000} - \frac{213\Delta}{500}$$

```
EQN3 =
```

$$Ts = \frac{567}{10}$$

From Laplace Transform to Transfer Function

Now we do some algebraic manipulation to get to the Transfer Function with output us Theta and Input Delta.

$$G(s) = \frac{\Theta(s)}{\Delta(s)}$$

We solve Eq. 2 for A and name it Eq.4

```
EQN4 =
```

$$\frac{2031}{139} - \frac{4260}{139} - \frac{10000}{139}$$

We substitute Eq. 4 as A in Eq.1 and name it Eq. 5

```
EQN5 =
```

$$-s\left(\frac{4260}{139} - \frac{2031}{139} + \frac{10000}{139}\right) = \frac{230367}{3475} - \frac{31291}{139000} + \frac{3130}{139}$$

We solve Eq. 5 for Q and name it Eq. 6

```
EQN6 =
```

$$\frac{31291 + 203000}{1000000 + 7390000 + 92146}$$

We substitute Eq. 6 as Q in Eq. 6 and name it Eq. 7

```
EQN7 =
```

$$Ts = \frac{567(31291 + 203000)}{10(1000000 + 7390000 + 92146)}$$

We solve Eq. 7 for T and name it Eq.8

```
EQN8 =
```

$$\frac{567(31291 + 203000)}{10s(1000000 + 7390000 + 92146)}$$

We divide Eq. 8 over D to get the transfer function and to simplify the answer.

```
G =
```

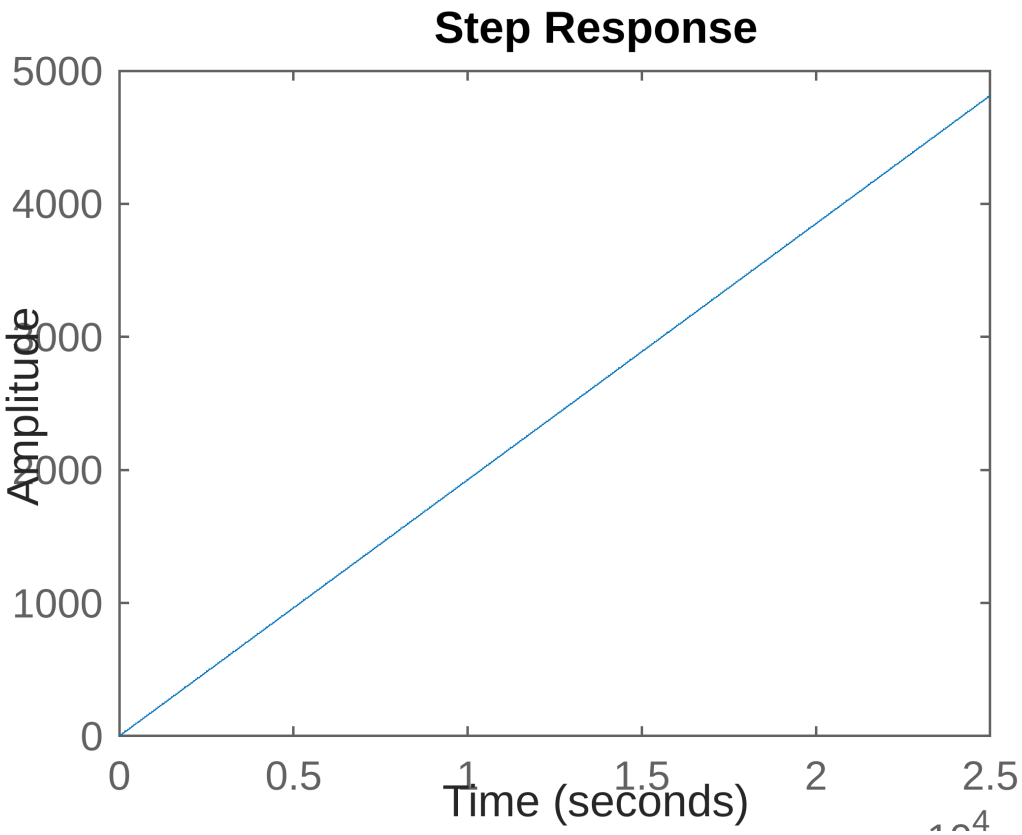
$$\frac{115101000 + 17741997}{400(2500000 + 1847500 + 230367)}$$

Now we obtain our transfer function:

```
G =
```

$$\frac{0.0025(1.151e8 + 1.7742e7)}{s(2500000 + 1847500 + 230367)}$$

```
G =

         2.878e05 s + 4.435e04
  ---------------------------------
  250000 s^3 + 184750 s^2 + 230367 s

Continuous-time transfer function.
Model Properties
```


Step Response
5000
4000
3000
2000
1000
0
Amplitude
0
0.5
1
1.5
2
2.5
Time (seconds)