Control Theory Homework 3

Selina Varouqa

April 1, 2020

1 Variant

Name: Selina Varouqa
Email: s.varouqa@innopolis.university
The variant for this assignment is A

2 Python Calculations

2.1 PD-controller Design

$\ddot{x} + \mu\dot{x} + kx = u$ is the time dependent second order linear ordinary differential equation, for my variant it is $\ddot{x}+3\dot{x}+45x = u$. Using PD Control: $u = k_p\dot{e}+k_d e$ and having the error e = x* - x we substitute and get $u = k_p(x^* - x)+k_d(\dot{x}^* - \dot{x})$
This is how my system looks like without control:

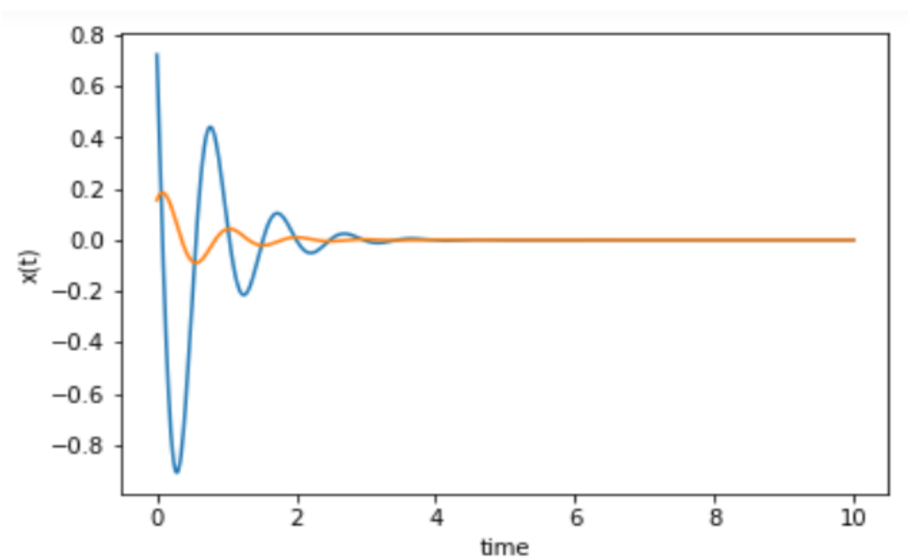

Figure 1: system without control

The first control applied to the system was as following: $desiredx = 2sin(2t)$ and $desiredxdot = 4cos(2t)$ with tuning $k_p = 1000$ and $k_d = 100$ we get the control as the graph shows, the orange is the $x(t)$ and the blue is $x'(t)$:

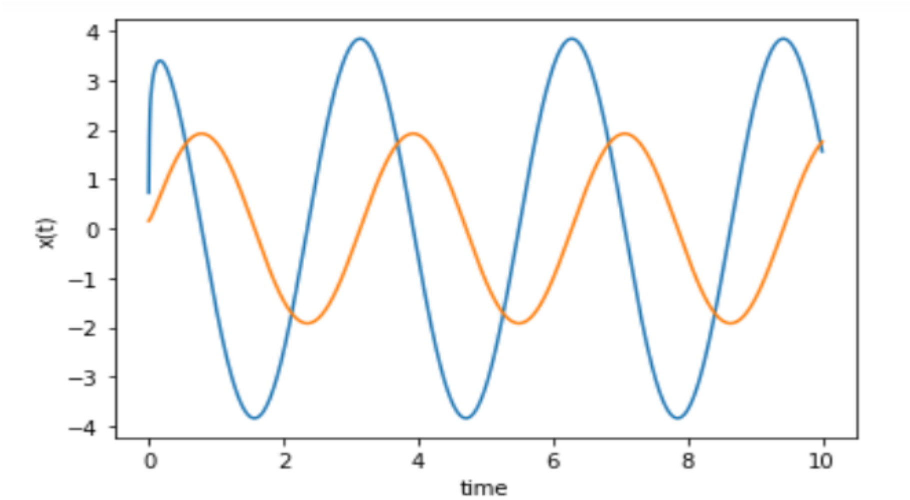

Figure 2: system with control with x desired as a sin function

The second control applied to the system was as following: $desiredx = 5 * t$ and $desiredxdot = 5$ along with tuning with $k_p = 1000$ and $k_d = 200$ we get the following control as the graph shows, orange is the $x(t)$ and blue is $x'(t)$:

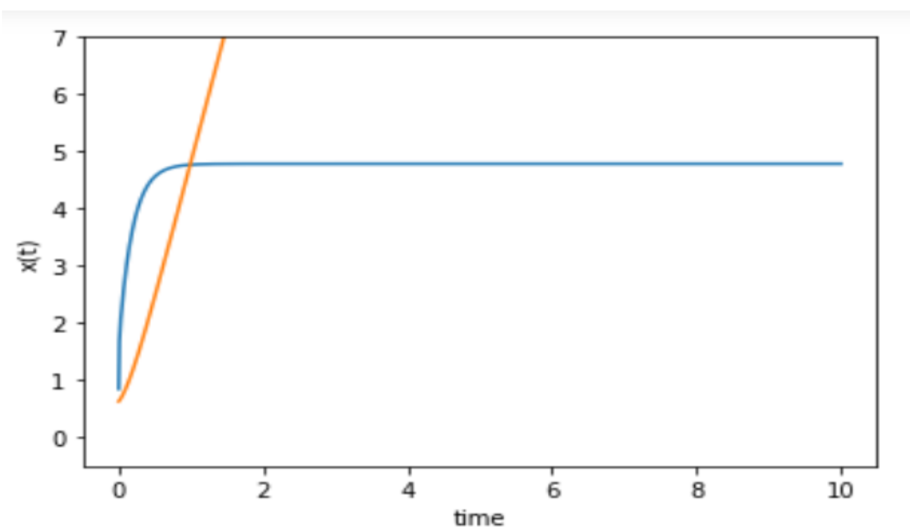

Figure 3: system with control with x desired as a linear function

2.2 Step function

I have tuned my kp and kd to be 50000 for each of them, it is a big number but from the observation of how the graph changes, with increasing kp it makes the x desired approach the step function but the x dot desired overshoots, that was solved by increasing kd as well. It was observed that the less difference I had between kp and kd the more that kd approached my desired function, it ended up being a very high number so I can get them both to approach the desired function, below is the resulting graph (green is step function, orange is the $x(t)$ and the blue is $x'(t)$):

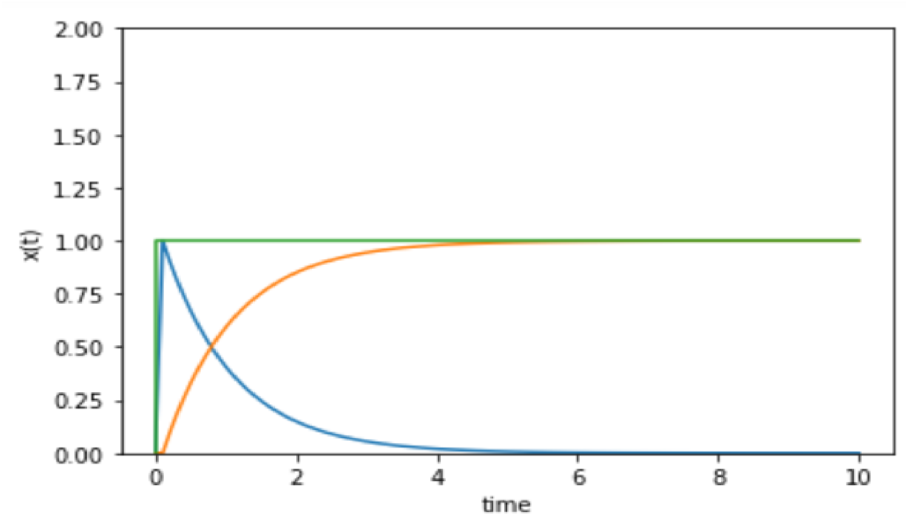

Figure 4: system with control approaching step input

2.3 Proving Stability

Proving stability for $\ddot{x} + \mu\dot{x} + kx = u$ for my choice of kp and kd which are 50000 and 50000. $\mu = 3, k = 45$ for my variant.

$\ddot{x} + \mu\dot{x} + kx = u$

$x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$ and $\dot{x} = \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \end{bmatrix}$

$x_0 = x$, $x_1 = \dot{x}$

$\dot{x_0} = \dot{x}$, $_1 = \dot{x}$

so, we have: $\ddot{x} = -\mu x_2 - kx_1 + u = \dot{x_1}$ state space representation would be: $\begin{bmatrix} \dot{x_0} \\ \dot{x_1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & -\mu \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ u let's say that matrix A is $\begin{bmatrix} 0 & 1 \\ -k & -\mu \end{bmatrix}$ and matrix B is $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Assume there is a $x^* = x^*(t)$ that $\dot{x^*} = Ax^*$ then our control error e would be $Ae - Bu$ so u = Ke when K = $\begin{bmatrix} Kd & Kp \end{bmatrix}$

C = A - KB = $\begin{bmatrix} 0 & 1 \\ -k & -\mu \end{bmatrix}$ - $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $\begin{bmatrix} Kd & Kp \end{bmatrix}$

which is $\begin{bmatrix} 0 & 1 \\ -k - k_d & -\mu - k_p \end{bmatrix}$ so we understand that for the system to be stable we have to have the eigenvalues to be negative. which means we can use that $\lambda_1 + \lambda_2 = -3 - k_p < 0$ and $\lambda_1 \cdot \lambda_2 = 45 + k_d > 0$

so k_p and k_d stabilizes the system such that $k_d > -45, k_p > -3$ and my choice of k_p and k_d matches that.

2.4 PD control for a MIMO

this is a multiple input multiple output system. We basically will have a state matrix and an input matrix. this is how the system looks before control:

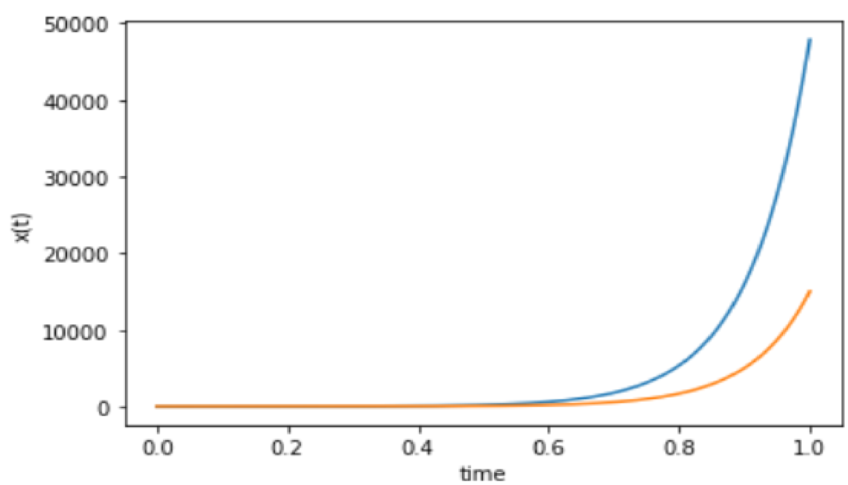

Figure 5: system before control

this is how the system looks after control:

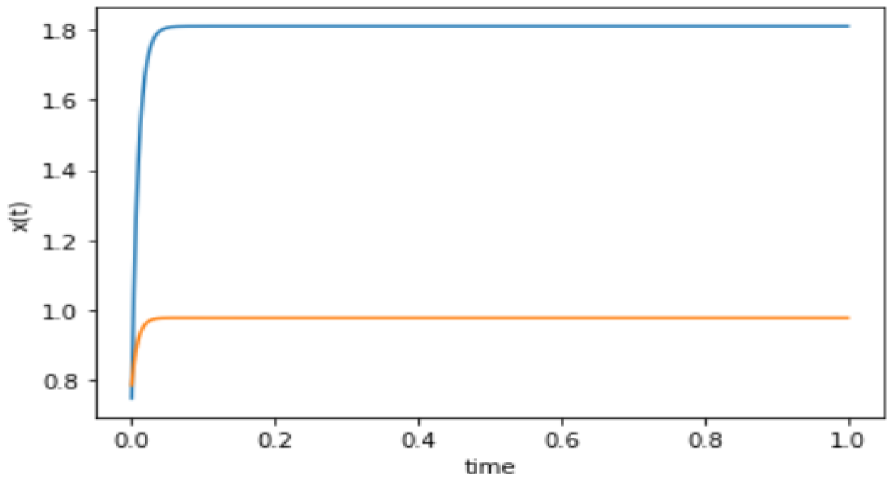

Figure 6: system after control

2.5 PI/PID controller

I implemented a PID control for the system for $\ddot{x} + 3\dot{x} + 45x + 9.8 = u$

$u = k_p e(t) + k_i \int_0^t e(t')dt' + k_d \frac{de(t)}{dt}$

so my x desired $= 5t$, x dot desired $= 5$, x int desired $= (5/2) * t^2$

the values of for my choice of kp and kd and ki are 500 and 30 and 10 for stabilizing the system.

this is how the system looks before PID control (green is integral component, orange is the $x(t)$ and the blue is derivative component):

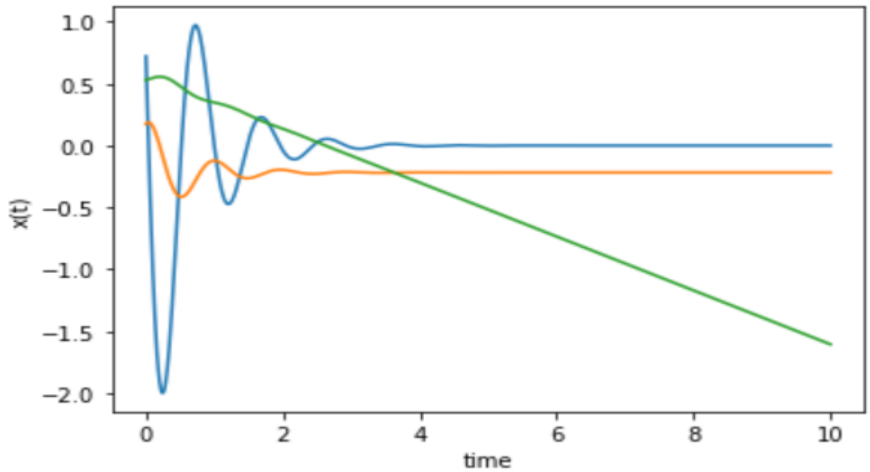

Figure 7: system before PID control

and this is how the system looks after PID control (green is integral component, orange is the $x(t)$ and the blue is derivative component):

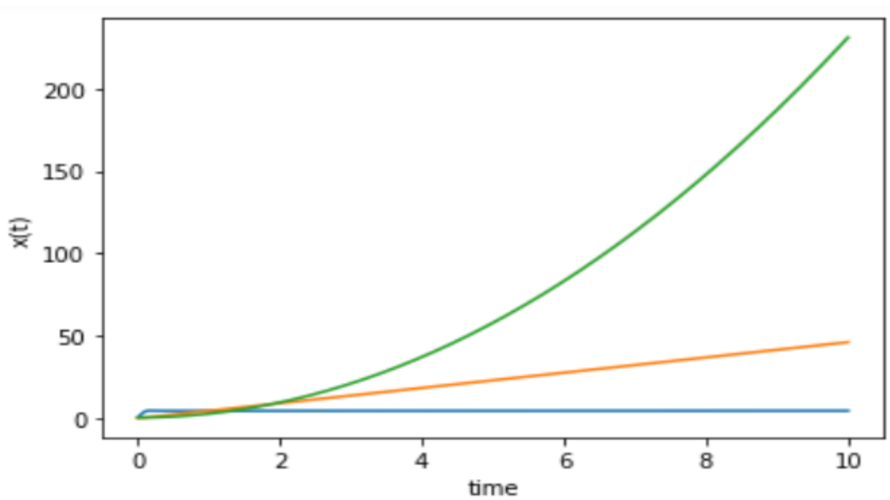

Figure 8: system after PID control

3 PID Controller Scheme

Using simulink, this is the scheme of the system before the control:

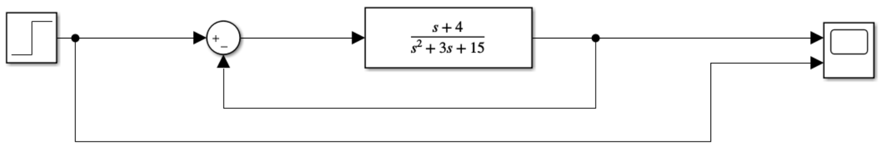

Figure 9: system scheme before PID control

and this is the graph that is in the scope before the control:

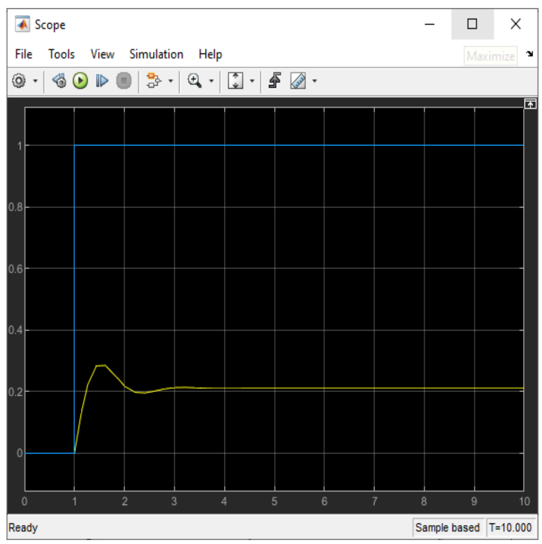

Figure 10: system graph before PID control

and below is the scheme of the system after adding PID controller:

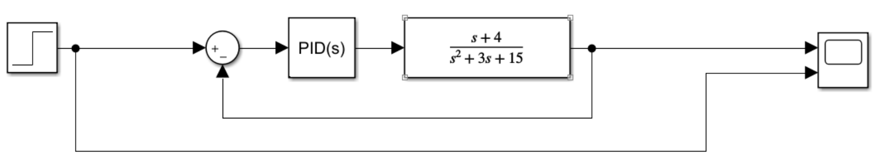

Figure 11: system scheme after PID control

First, I have tuned my Kp, Ki, and Kd for 100, 50, and 1 (respectively) and I got the result of the following graph (before using auto-tune):

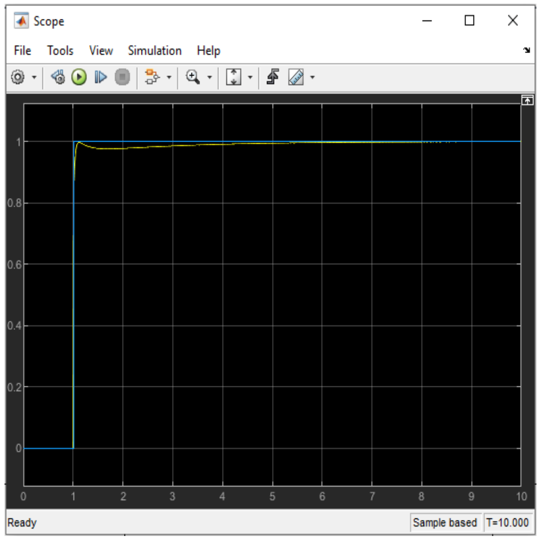

Figure 12: system graph after PID control

After using opening to auto-tune, I actually looked at the following table which compared the one I have and the one proposed:

Controller Parameters

	Tuned	Block
P	8.2985	100
I	28.7197	50
D	0	1
N	100	100

Performance and Robustness

	Tuned	Block
Rise time	0.155 seconds	0.0247 seconds
Settling time	1.33 seconds	NaN seconds
Overshoot	5.37 %	0 %
Peak	1.05	0.997
Gain margin	Inf dB @ NaN rad/s	Inf dB @ NaN rad/s
Phase margin	69 deg @ 10.3 rad/s	103 deg @ 182 rad/s
Closed-loop stability	Stable	Stable

Figure 13: table

4 Compensator Scheme

This is how the scheme of the system looks before the compensator:

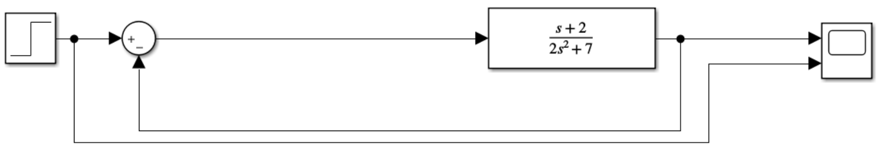

Figure 14: before compensator

and this is how the graph of the system before the compensator:

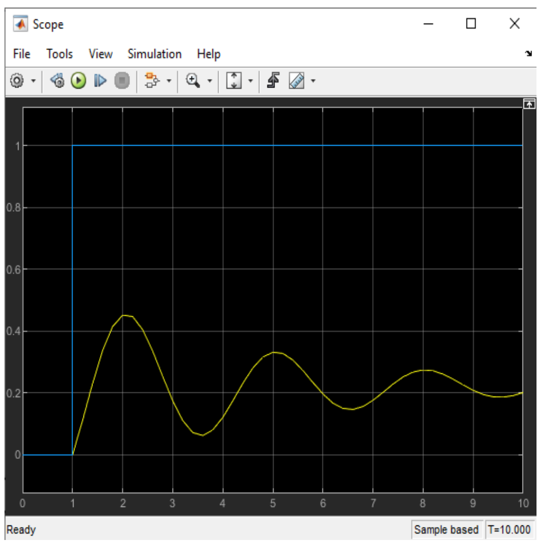

Figure 15: before compensator

after adding the compensator and but before changing coefficients and gain, it looks like this:

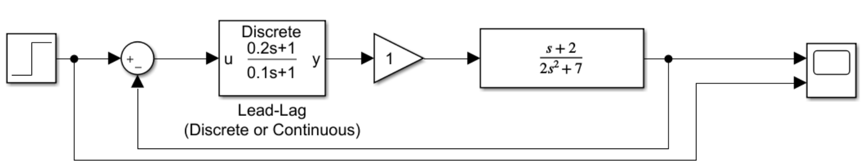

Figure 16: compensator with system scheme

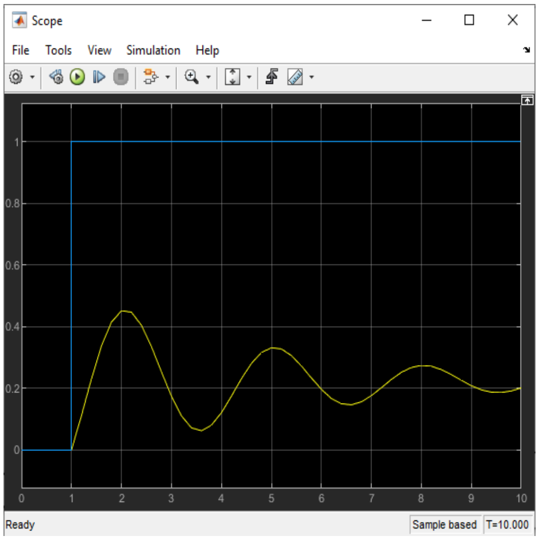

Figure 17: compensator w/o manual tuning

but, then I tuned the coefficients and the gain manually until I got this graph:

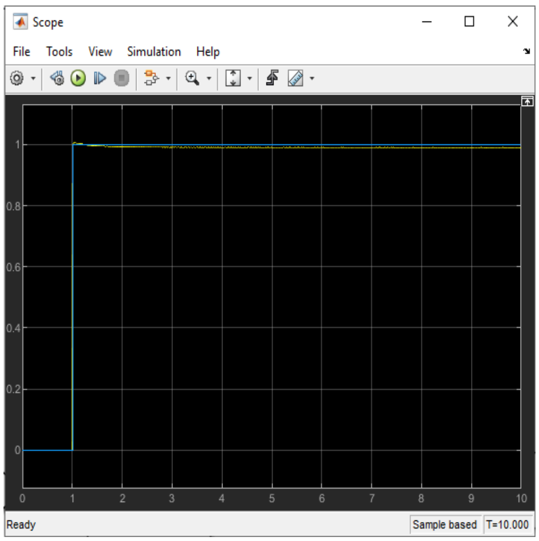

Figure 18: compensator with manual tuning

and the scheme looked like this:

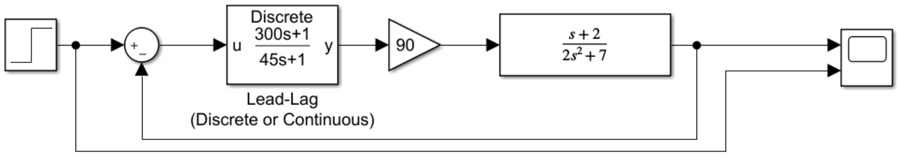

Figure 19: compensator with manual tuning

and last but not least, when I used the control system designer with this code

```
controlSystemDesigner(tf([1,2],[2,0,7]))
```

and it showed that my system (the one I tuned manually) was pretty stable according to the following graphs:

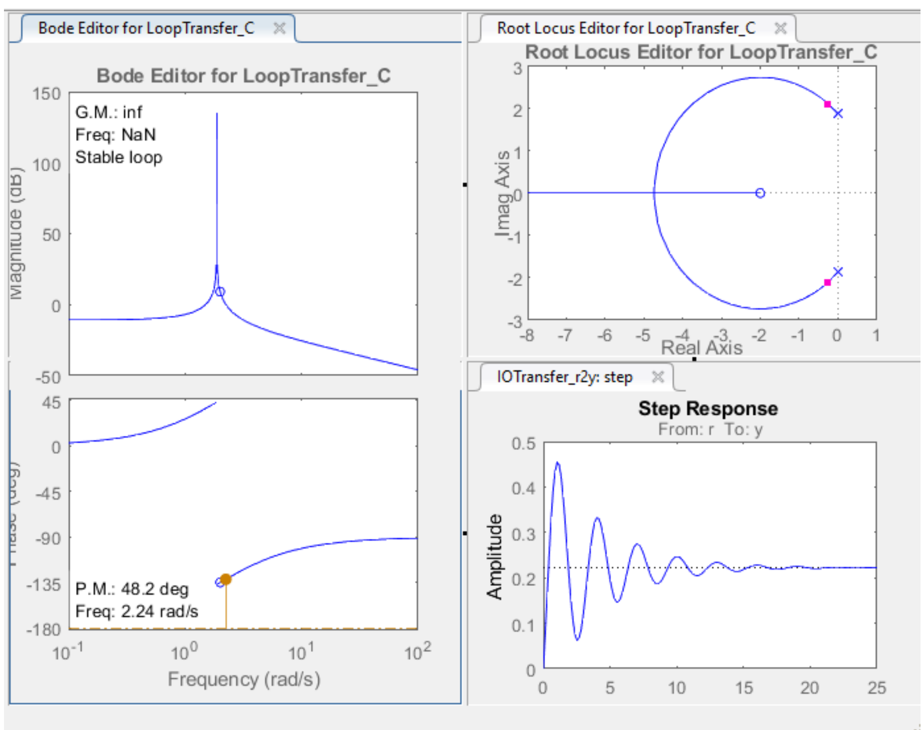

Figure 20: compensator with manual tuning