Daniil Manakovskiy
BS18-02

Homework 3

Linear Control Theory
March 28, 2020

Email: d.manakovskiy@innopolis.university
Variant: C

Question 2

Source Code
$\mu = 13, k = 2$

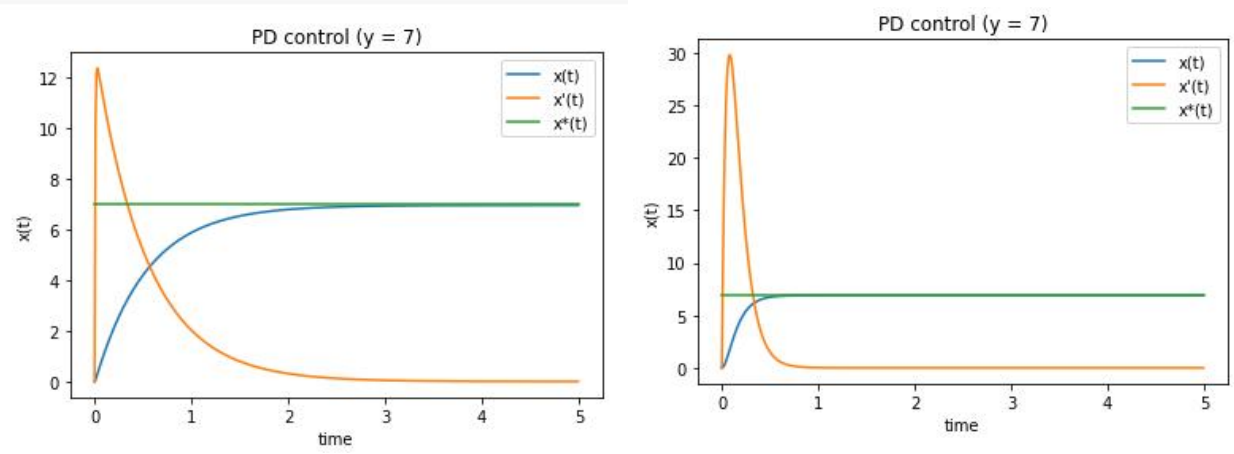
Sub-tasks A and B

Below there are comparison between outputs using non-tuned and tuned controllers.
PD coefficients:

- non-tuned : $K_p = 300, K_d = 150$
- tuned: $K_p = 143, K_d = 12$

Figures presented below show the comparison between tuned and not tuned PD controllers on different trajectories:

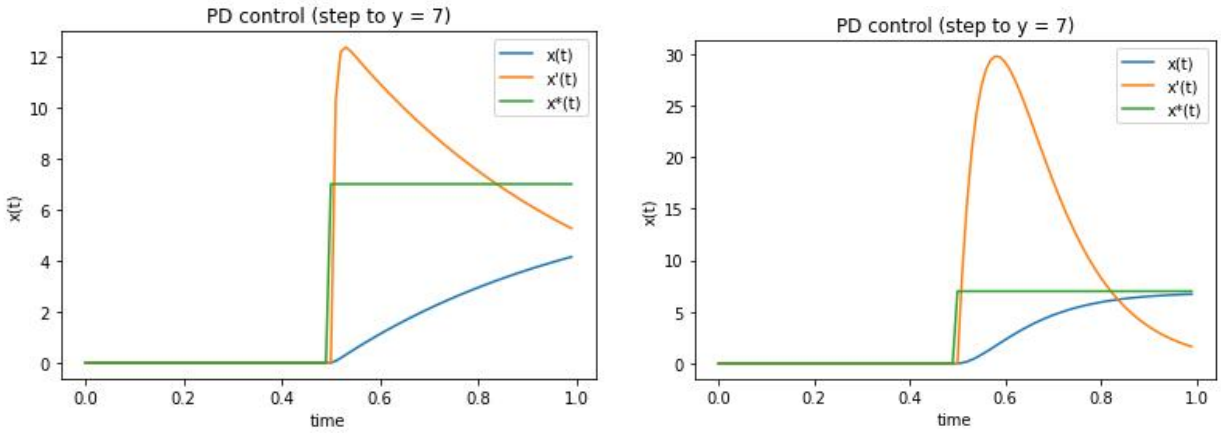
- constant - fig. 1
- step - fig. 2
- sine - fig. 3



(a) [PD] Not-tuned

(b) [PD] Tuned

Figure 1: Constant trajectory $x^*(t) = 7$



(a) [PD] Not-tuned

(b) [PD] Tuned

Figure 2: Step trajectory

Result: tuned controller converges faster.

Sub-task C

To prove that controlled dynamics is stable, lets consider the corresponding transfer function (ref.):

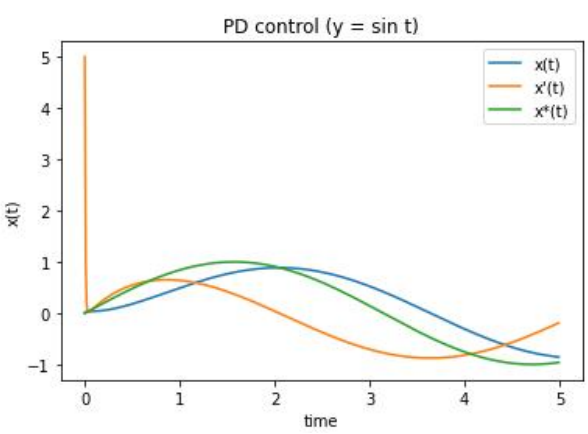
$$\frac{K_d s + K_p}{s^2 + (\mu + K_d)s + (k + K_p)} = \frac{12s + 143}{s^2 + 25s + 145} = \frac{12(s + 11.9167)}{(s + 9.1459)(s + 15.8541)}$$

Since all poles are negative the dynamics is stable.

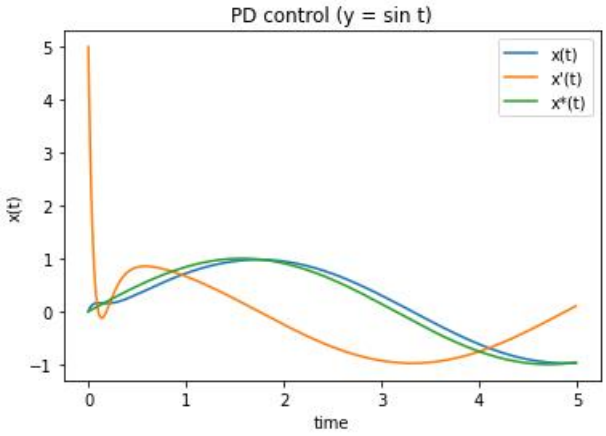
Sub-task D

In order to implement PD control for the described linear system, we should keep proportional error of the last step, so that we can calculate derivative term as follows:

$$ERROR_d = \frac{\Delta ERROR_p}{\Delta t}$$



(a) [PD] Not-tuned



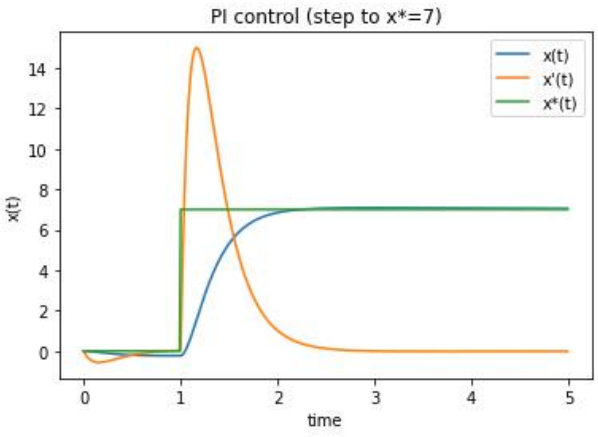
(b) [PD] Tuned

Figure 3: Trajectory $x^*(t) = sin(t)$

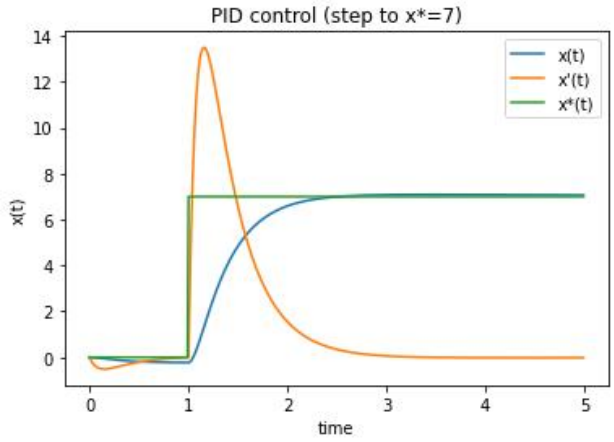
Sub-task E

Figures 4 and 5 compare PI and PID controllers on step and sine trajectories correspondingly. Coefficients for controllers:

- PI: $K_p = 36, K_i = 10$
- PID: $K_p = 36, K_d = 2, K_i = 9$

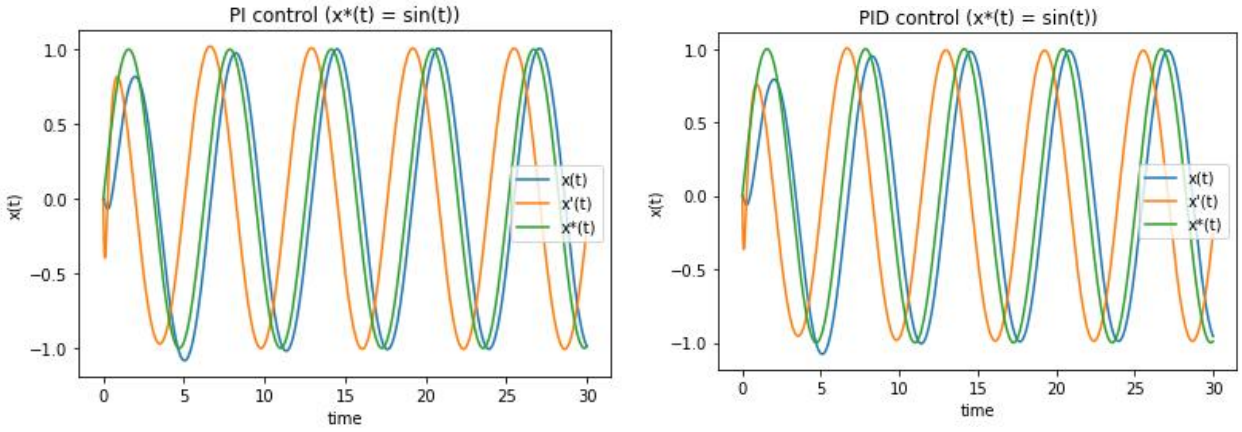


(a) PI controller



(b) PID controller

Figure 4: Step trajectory



(a) PI controller

(b) PID controller

Figure 5: Sine trajectory

Question 3

The transfer function in my variant:

$$W = \frac{s+1}{s^3 + 2s^2 + 9s + 20}$$

My actions were as follows:

1. Create a new model from `Feedback Controller` template. The schema is depicted on figure 6.

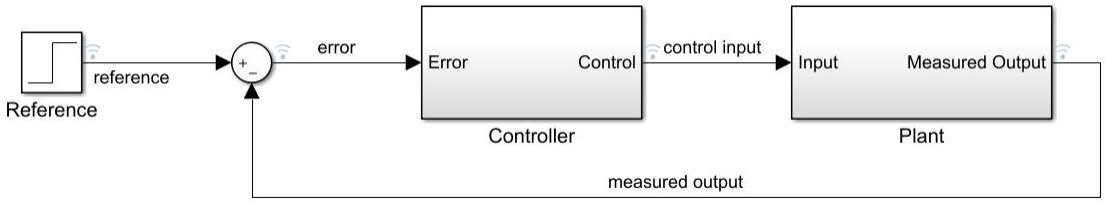


Figure 6: Feedback controller

2. Replace a plant for a given TF. The plant is shown on figure 7.

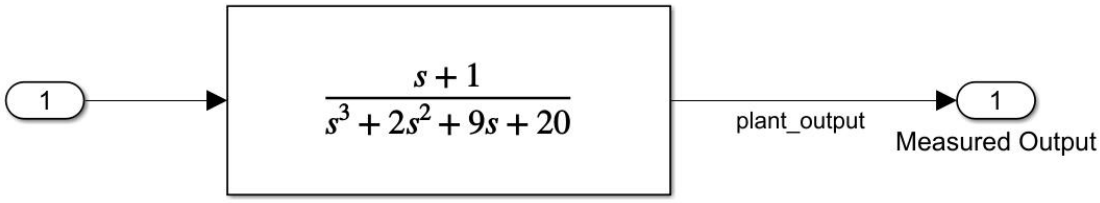


Figure 7: The plant

3. Having inspected in-build `PID controller` block and understood that it provides noise filtering thus require additional coefficients, I decided to build the controller from scratch to keep simple classic model.

4. Tune the controller using Matlab PIDtuner:

   (a) Open the tuner:

```
>> transfer_func = tf([1 1], [1 2 9 20])

transfer_func =

          s + 1
  ----------------------
  s^3 + 2 s^2 + 9 s + 20

Continuous-time transfer function.

>> pidTuner(transfer_func, 'PID')
```

   (b) In tuner play with response time and transient behavior to minimize overshoot. Figure 8 show the tuner's interface after tuning. Figure 9 shows tuned parameters. As we can see, there is still small overshoot of 0.0362%, but combining with small rise and settling time, the result is good. Final version of the controllers is at figure 10.

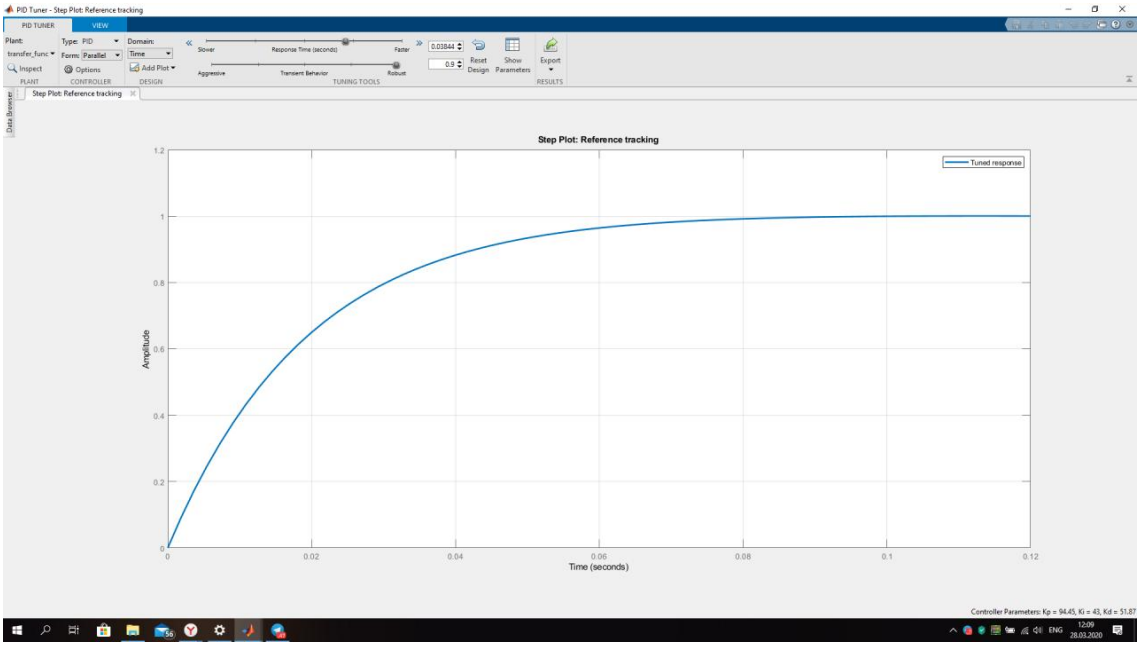


Figure 8: Tuner

Controller Parameters

	Tuned
Kp	94.4497
Ki	42.9999
Kd	51.865
Tf	n/a

Performance and Robustness

	Tuned
Rise time	0.0409 seconds
Settling time	0.0687 seconds
Overshoot	0.0362 %
Peak	1
Gain margin	-50 dB @ 3.07 rad/s
Phase margin	89.1 deg @ 52 rad/s
Closed-loop stability	Stable

Figure 9: Parameters after the tuning

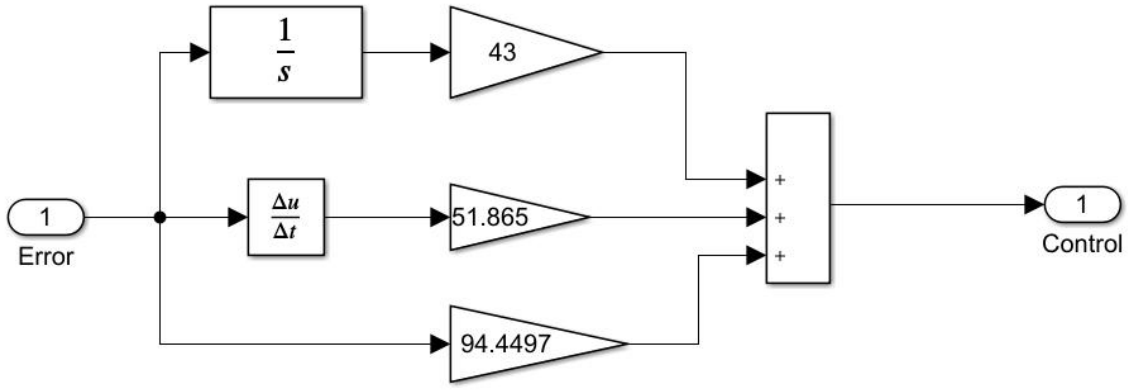


Figure 10: The controller

5. Build a schema to compare outputs with and without a controller. The result is at figure 11.

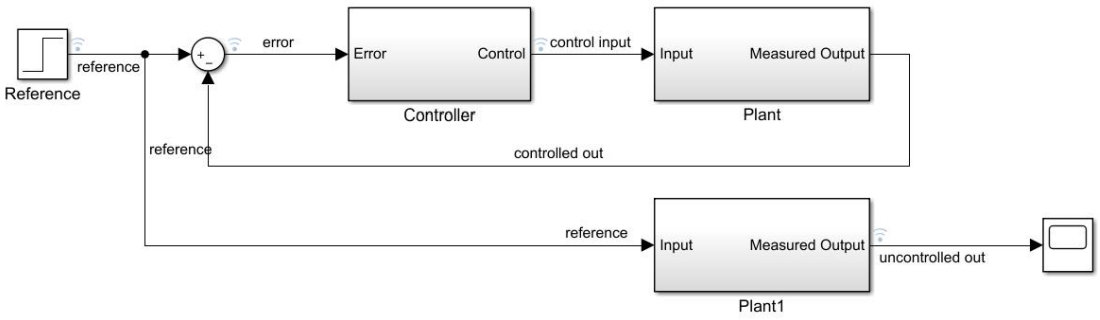


Figure 11: The schema for comparison responses

6. Mark signals to compare to be logged.

7. Run simulation

8. Inspect signals using data analyser. The comparison is shown at figure 12. So we can see, that tuned PID controller really does its job well.

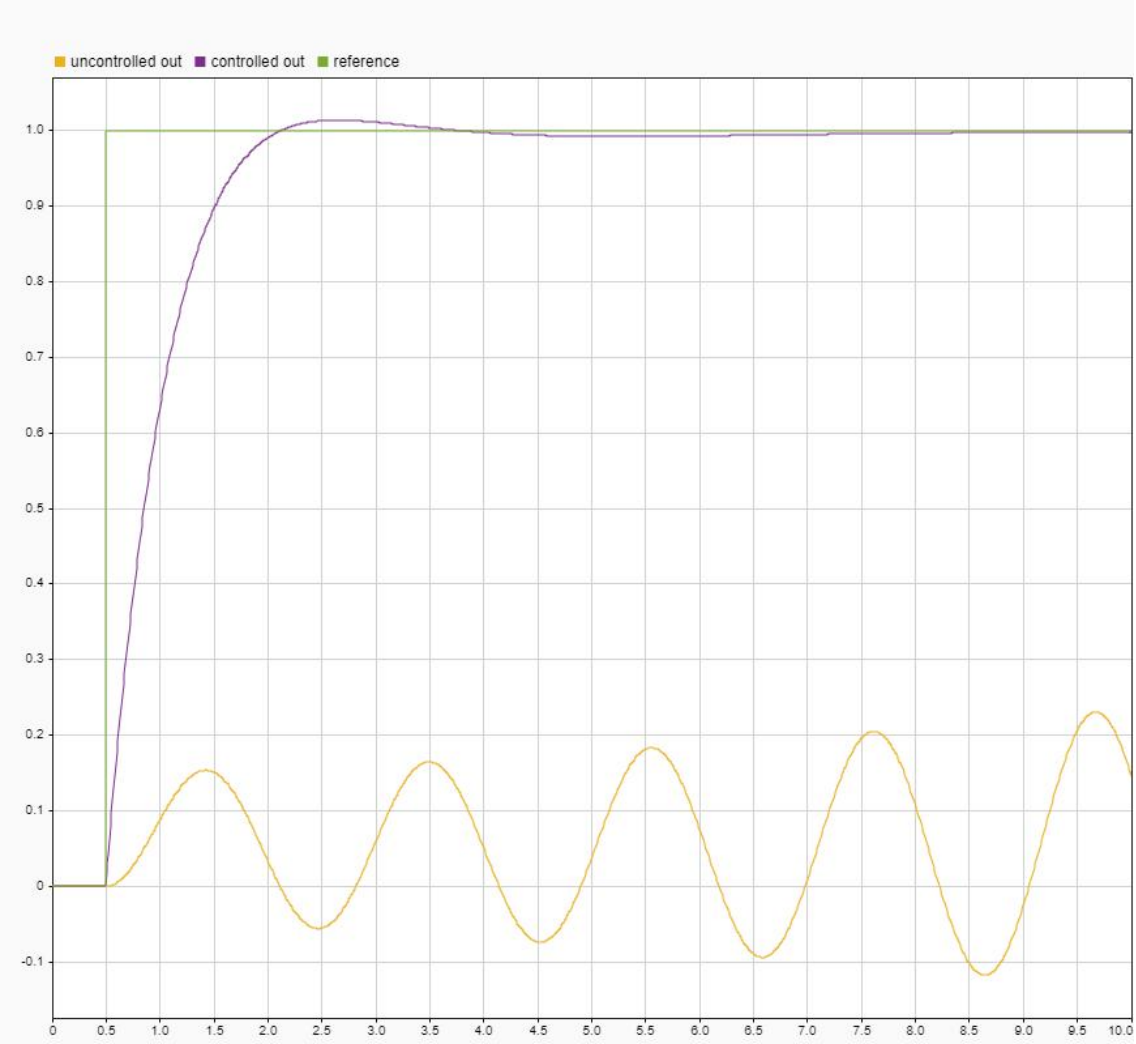


Figure 12: The signals' output

Question 4

The transfer function in my variant:

$$W = \frac{s+2}{s^2+4s+11}$$

Figure 13 shows Bode plot, Root Locus and Step Response of the initial system. As we can see, the system is stable, the only problem is that step response is to be tuned. An overshoot now is about 30%, increasing gain will lead to greater overshoot, therefore it is not a good idea.

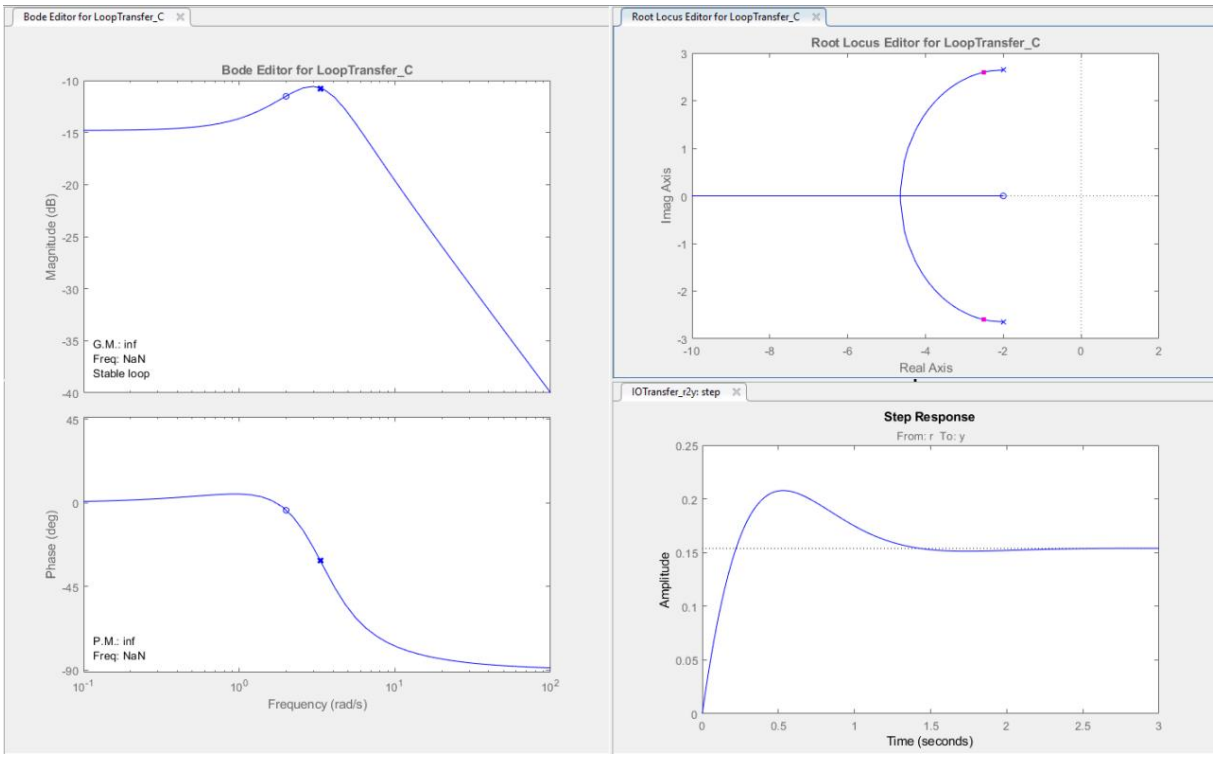


Figure 13: Initial system metrics

To make the situation better, let's introduce lead compensator via compensator editor:

$$5000\frac{s+1}{s+10}$$

So that the system metrics become much better (fig 14):

- 1% steady-state error
- $1.5 \cdot 10^{-3}$ sec rise and transient time

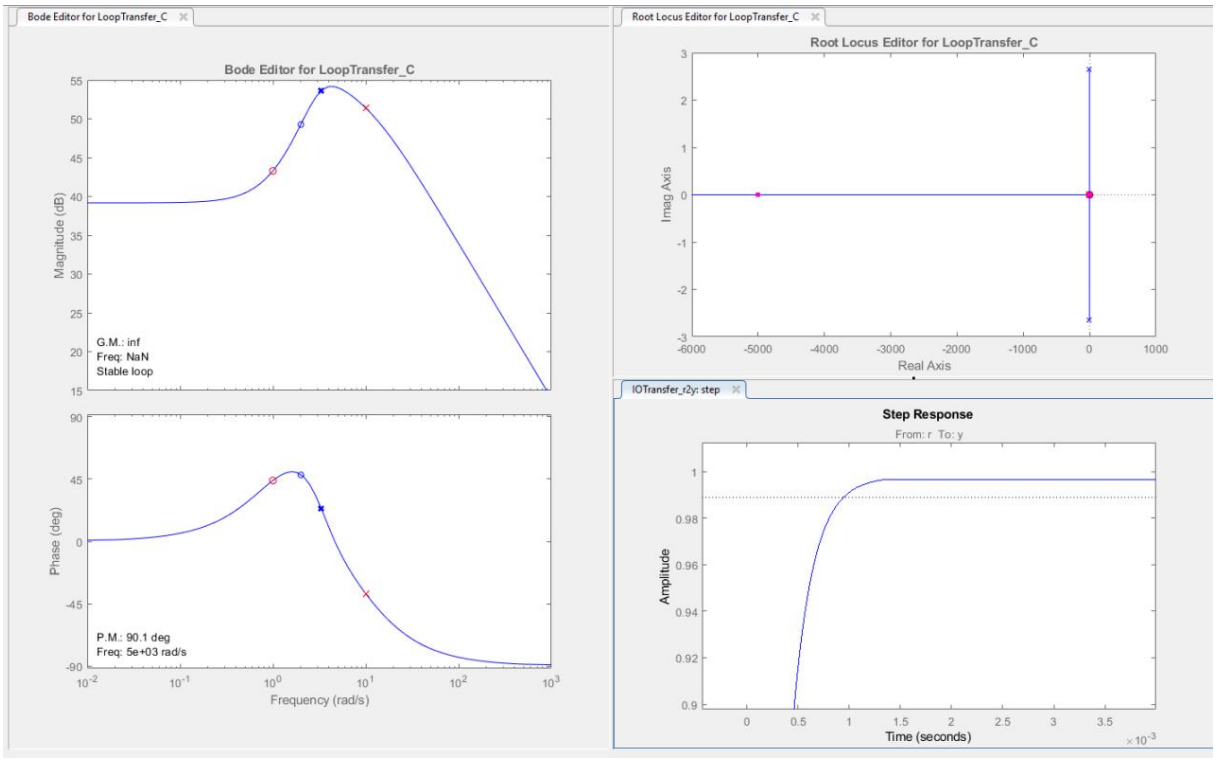


Figure 14: Tuned system metrics

Increasing the coefficient to 10^5 the system can have (fig 15):

- 0.03% steady-state error
- 10^{-4} sec rise and transient time

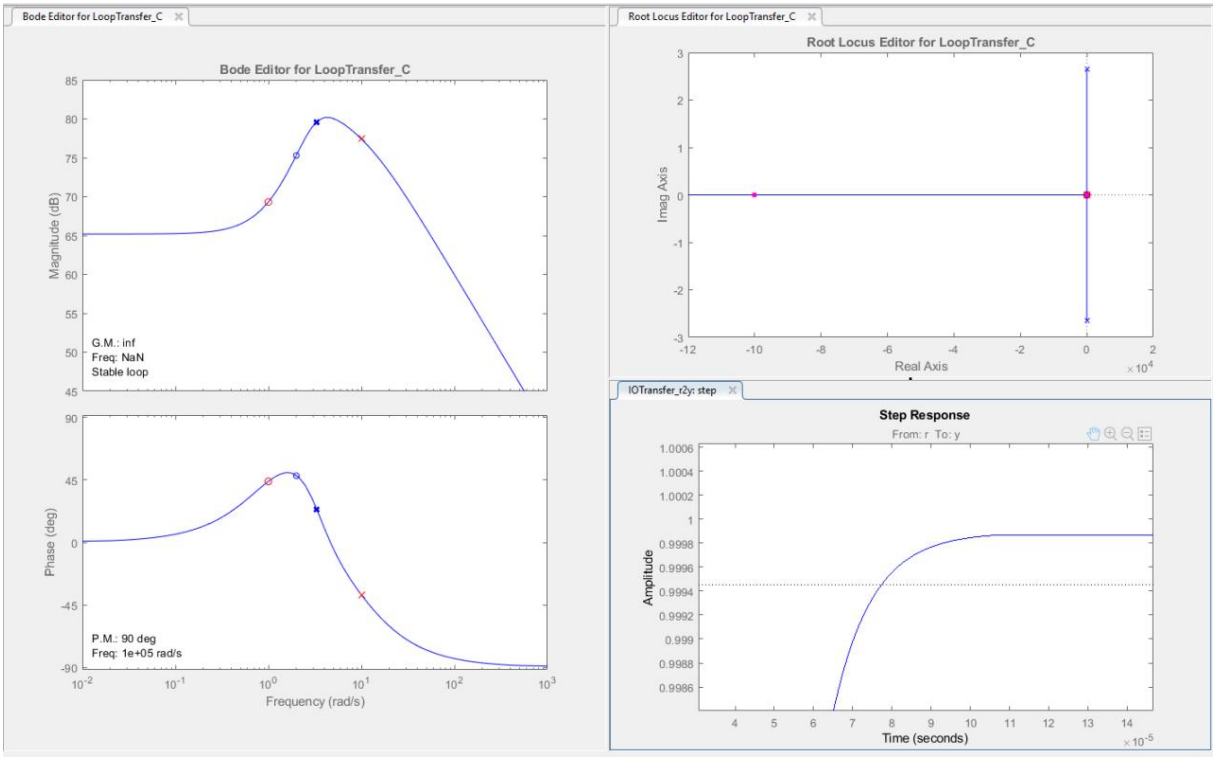


Figure 15: Over-tuned system metrics

But usually nobody needs such precision, we 'over-tuned' the system.