# Control Theory homework 3

Kamil Hayrullin
BS18-02

## 1 Introduction

My Innopolis mail is k.hayrullin@innopolis.ru
My variant is "e"
Link to my GitHub repository is here

## 2 Implement given tasks using python

**(A)** Design PD-controller that tracks time varying reference states i.e.
[x*(t), x'*(t)] as closely as possible.

Lets consider second order linear ODE

$$x'' + \mu x' + kx = u, \mu = 14, k = 85 \tag{1}$$

To create Proportional-derivative controller we should have P control:

$$u = k_p(x^* - x) \tag{2}$$

And add derivative term to it

$$u = k_p(x^* - x) + k_d(x'^* - x') \tag{3}$$

Where control error equals

$$e = x^* - x \tag{4}$$

Finally, PD controller:

$$u = k_p e + k_d e' \tag{5}$$
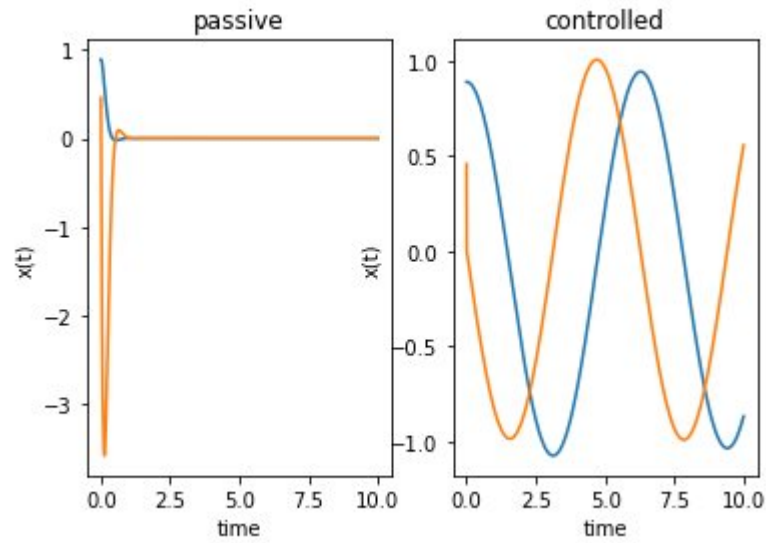
Link to python code in Google Colaboratory is here

Figure 1: for COS function result is following, kp = 1000, kd = 10000



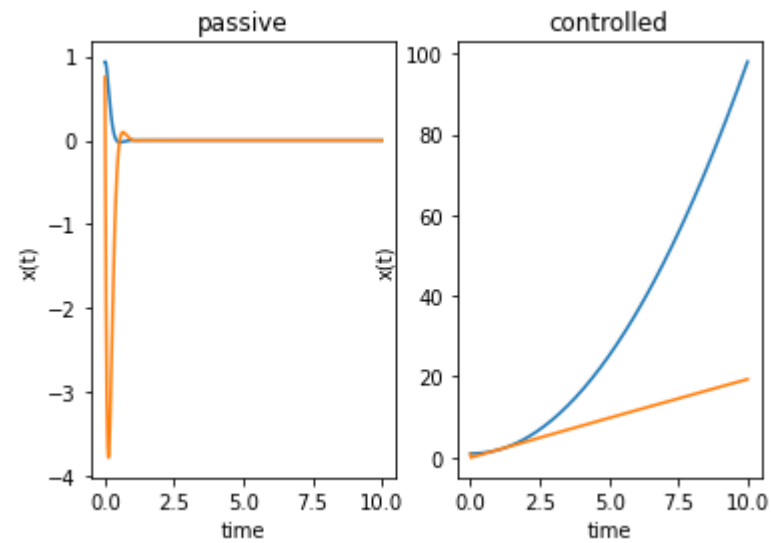Figure 2: for t*t function result is following, kp = 1000, kd = 10000

**(B)** Tune controller gains kp and kd . Find gains that provide no oscillations and no overshoot. Prove it with step input.

My choice of gains is:

$$k_p = 1000, k_d = 10000 \qquad (6)$$
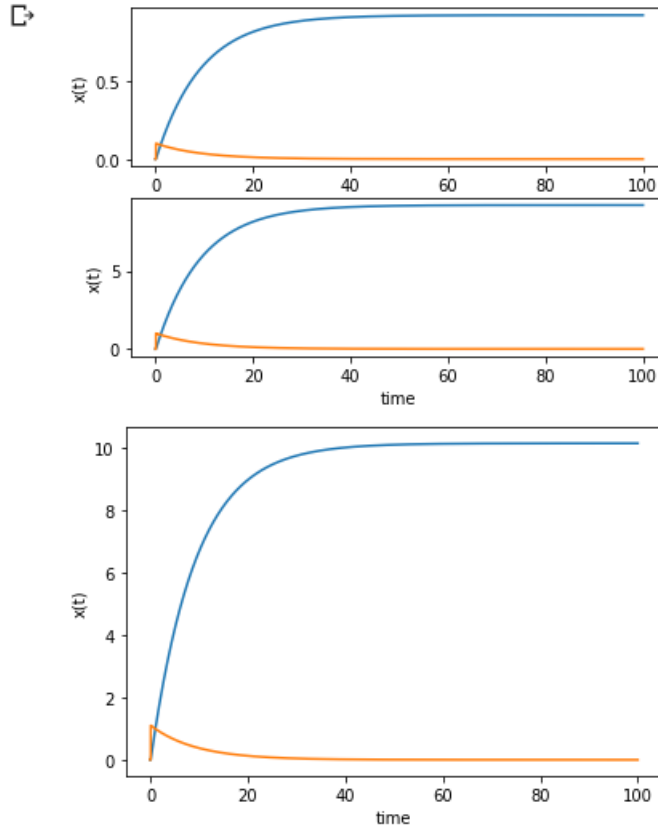
Link to python code in Google Colaboratory is here



Figure 3: As we can see, no oscillations and no overshoot

**(C)** Prove that controlled oscillator dynamics is stable for your choice of kp, kd

Lets consider second order linear ODE

$$x'' + 14x' + 85x = 1000 * (x^* - x) + 10000 * (x'^* - x') \qquad (7)$$

$$x'' = -14x' - 85x + 1000 * x^* - 1000 * x + 10000 * x'^* - 10000 * x' \qquad (8)$$

$$\begin{pmatrix} x'' \\ x' \end{pmatrix} = \begin{pmatrix} -14 - 10000 & -85 - 1000 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x' \\ x \end{pmatrix} + \begin{pmatrix} 10000 & 1000 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x^{*\prime} \\ x^* \end{pmatrix} \qquad (9)$$

3

Now let's find eigenvalues of matrix A

$$det(A - \lambda I) = (-10014 - \lambda)(-\lambda) + 1085 = \lambda^2 + 10014\lambda + 1085 = 0 \quad (10)$$

$$\lambda_1, \lambda_2 = -0.10834, -10013.89 \quad (11)$$

Both eigenvalues less than zero, system is **stable**

**(D)** Think of how you would implement PD control for a linear system:

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 10 & 3 \\ 5 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (12)$$

,

To implement PD controller I would add control law

$$u = k_p e + k_d e' \quad (13)$$

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 10 & 3 \\ 5 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} (k_p e + k_d e') \quad (14)$$

I would translate this state space form into Linear Time Varying system and apply 1st task's code to this LTV.

**(E)** Implement a PI/PID controller for the system: x + 14x' +85x + 9.8 = u. Test your controller on different trajectories, at least two.
Link to python code in Google Colaboratory is here



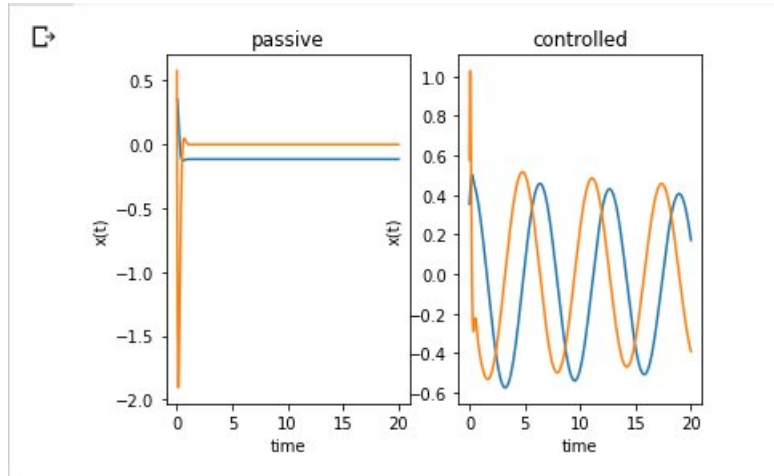Figure 4: for COS function result is following, kp = 2, kd = 100

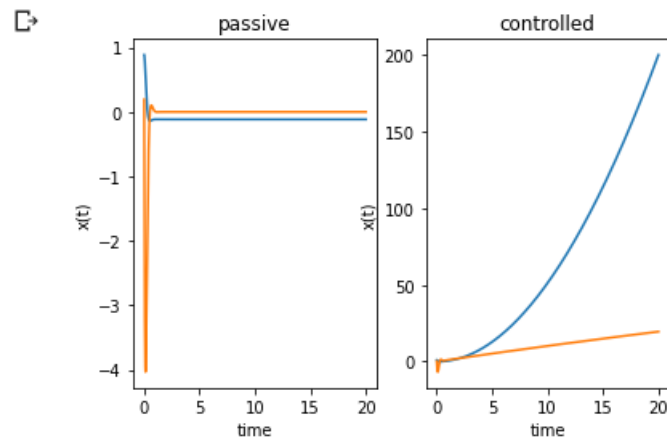Figure 5: for t*t function result is following, kp = 2, kd = 100

# 3 Design a PID controller. Use step input function and try to improve rise time, overshoot and steady-state error, comparing with no controller system
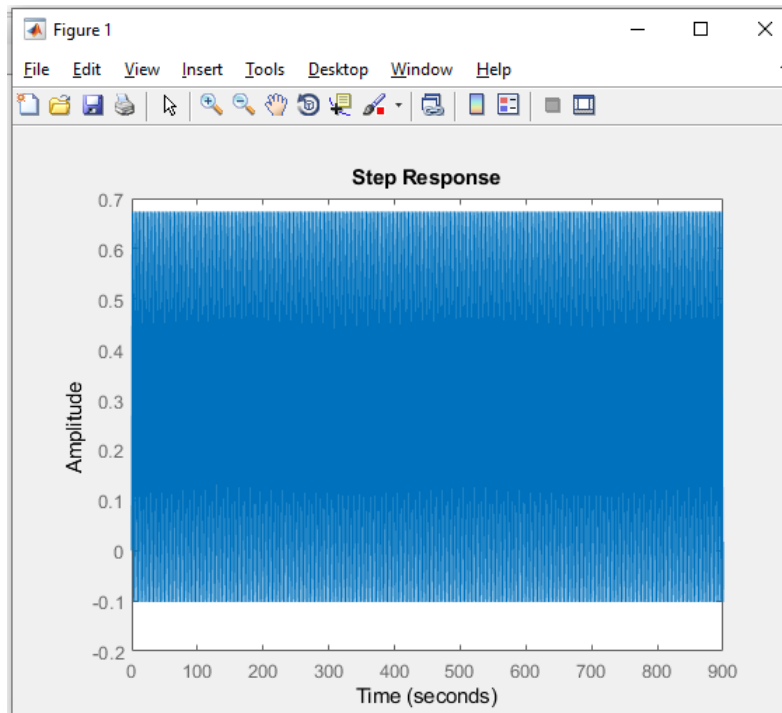
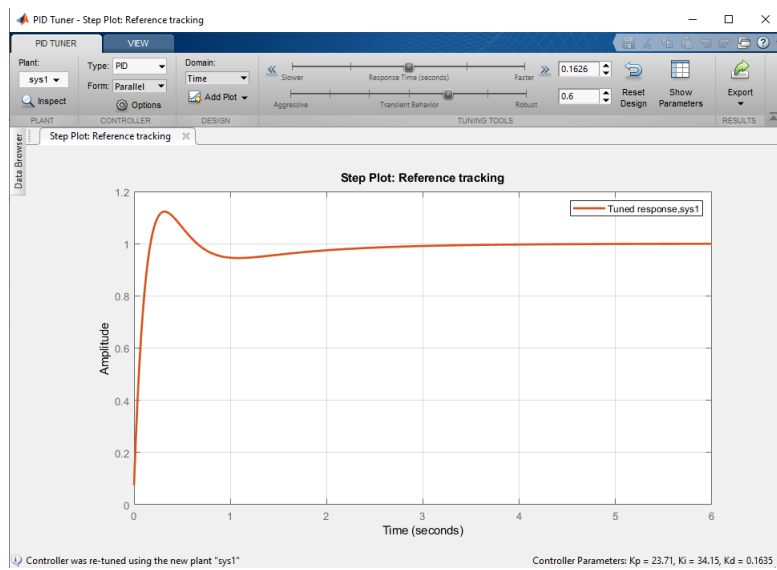$$W(s) = \frac{s+2}{2s^2 + 7} \tag{15}$$

Let's wtire such code in matlab

```
numerator = [1, 2];
denominator = [2,0,7];
sys1 = tf(numerator,denominator)
step(sys1)
```
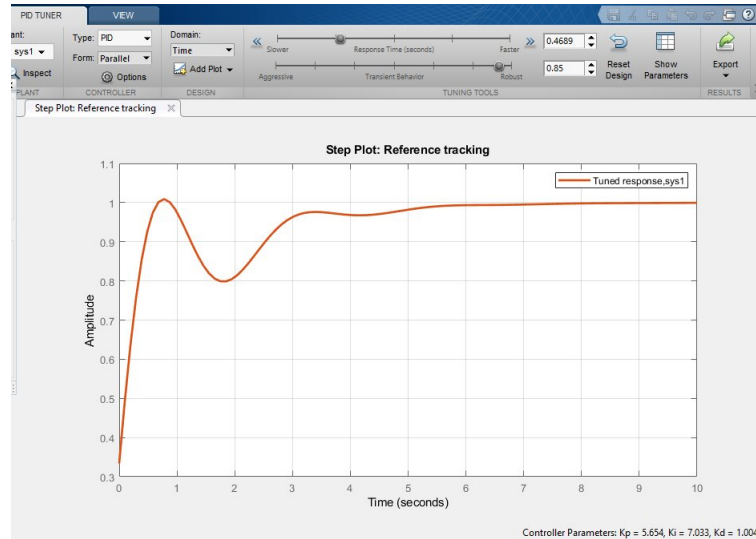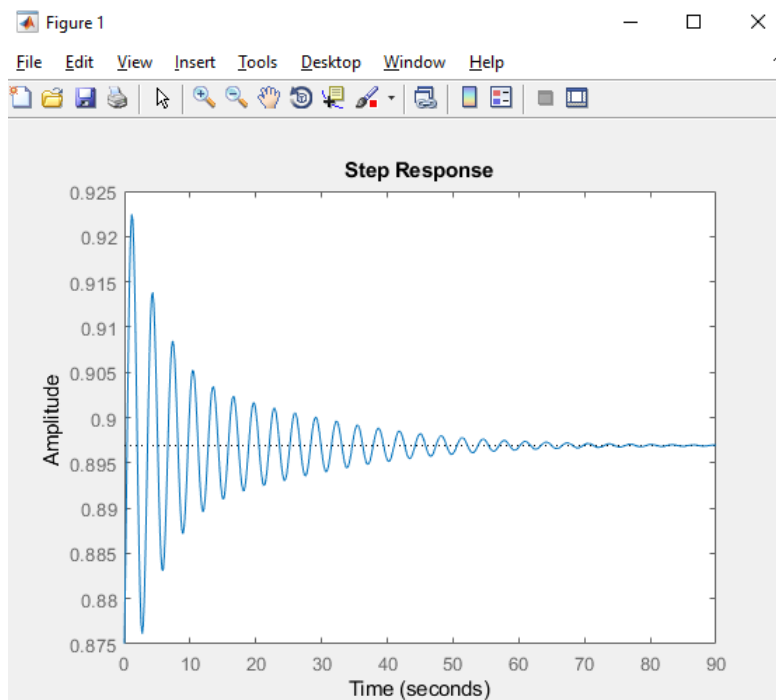
non controlled step function



In PID Tuner it looks like that

Now we tune it a bit



and get this step function



Was:

| Controller Parameters | |
| --- | --- |
| | Tuned |
| Kp | 4.2362 |
| Ki | 5.9182 |
| Kd | 0.59223 |
| Tf | n/a |
| | |
| | |

| Performance and Robustness | |
| --- | --- |
| | Tuned |
| Rise time | 0.486 seconds |
| Settling time | 5.15 seconds |
| Overshoot | 2.71 % |
| Peak | 1.03 |
| Gain margin | Inf dB @ NaN rad/s |
| Phase margin | 64.6 deg @ 3.46 rad/s |
| Closed-loop stability | Stable |

Now:

| Controller Parameters | |
| --- | --- |
| | Tuned |
| Kp | 5.6536 |
| Ki | 7.0333 |
| Kd | 1.004 |
| Tf | n/a |
| | |
| | |

| Performance and Robustness | |
| --- | --- |
| | Tuned |
| Rise time | 0.458 seconds |
| Settling time | 5.23 seconds |
| Overshoot | 0.894 % |
| Peak | 1.01 |
| Gain margin | Inf dB @ NaN rad/s |
| Phase margin | 89.9 deg @ 4.27 rad/s |
| Closed-loop stability | Stable |

Results became better.

# 4 Design a lag or lead compensator (if applicable), play with zero and pole to find optimal values (of overshoot, peak time, transient process time, stationary error, etc.) for transient process.

$$W(s) = \frac{s+4}{s^2 + 3s + 15} \tag{16}$$
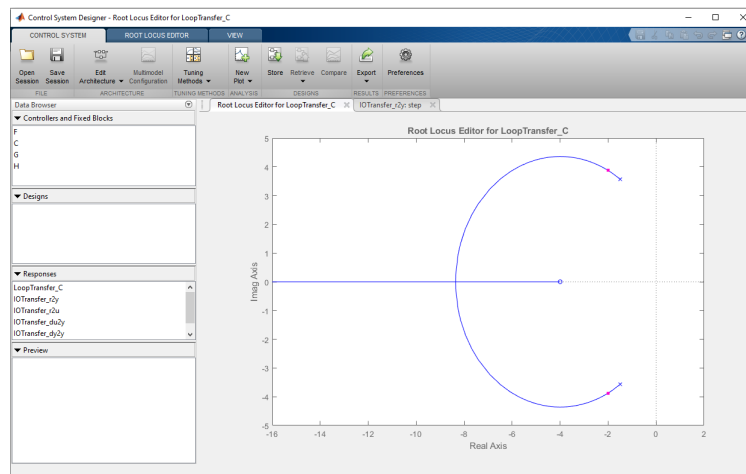
Lag or lead compensator looks like that:

$$C(s) = K_c \frac{(s - z_0)}{(s - p_0)}$$
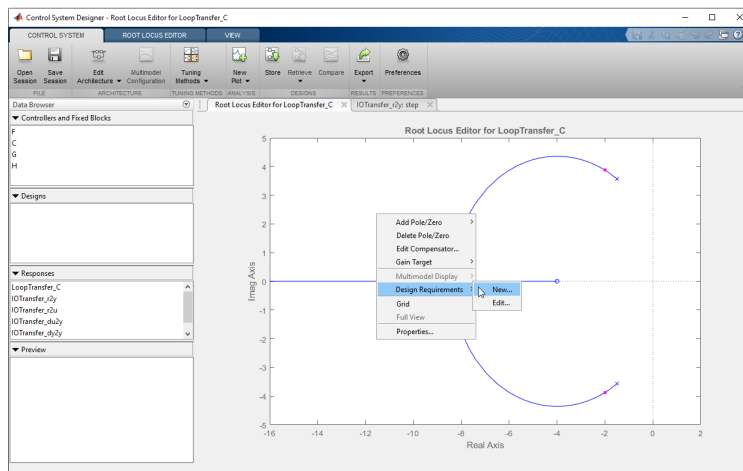
Figure 6: first-order lead compensator C(s)

Let's run such program in matlab:

```
numerator = [1, 4];
denominator = [1,3,15];
sys = tf(numerator,denominator)
rltool(sys)
```

We can see, that Root Locus of our transfer function in Control System Designer.

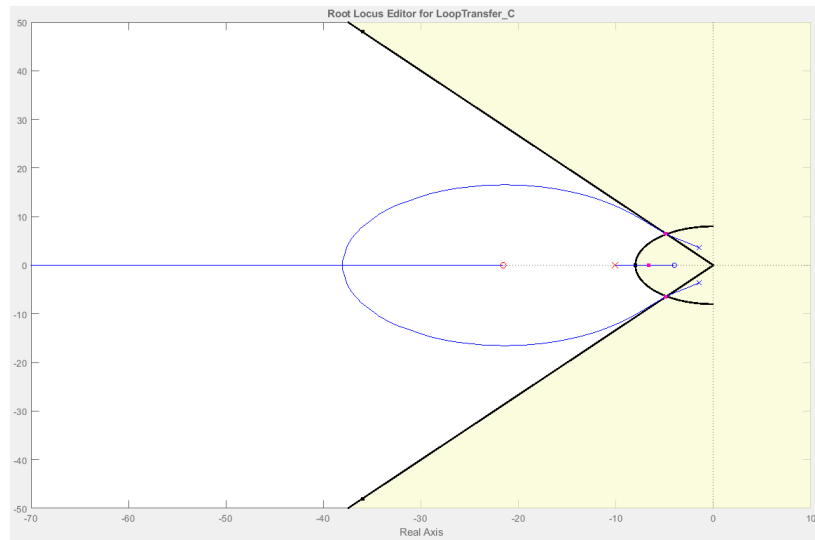Let's add some design requirements
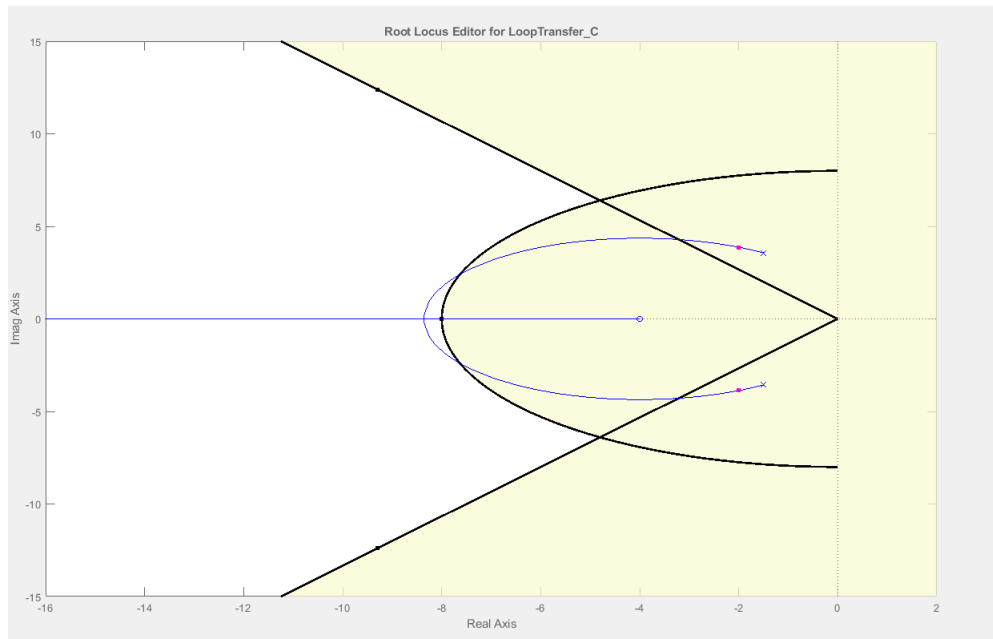






We have:
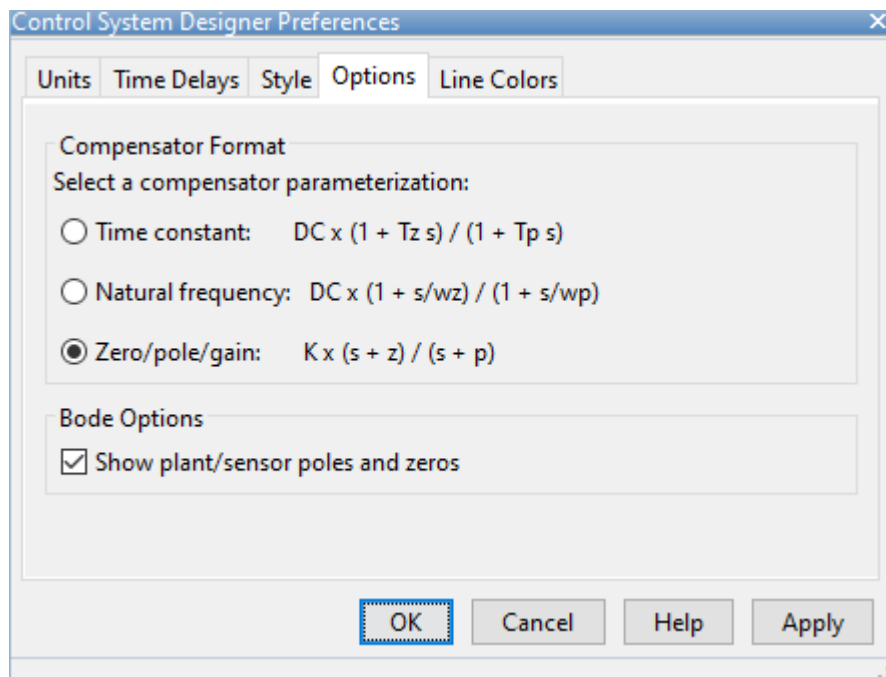
Our goal is to place red squares in intersection of damping ratio and natural frequency graph.
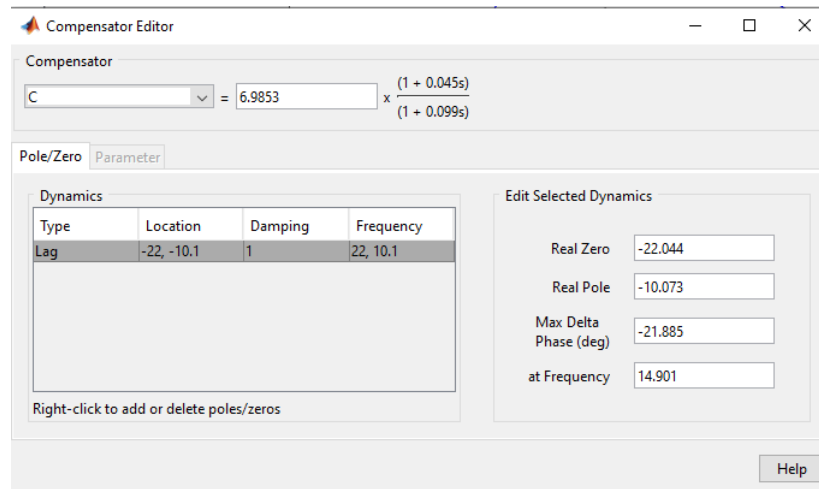To do this let's add Lead
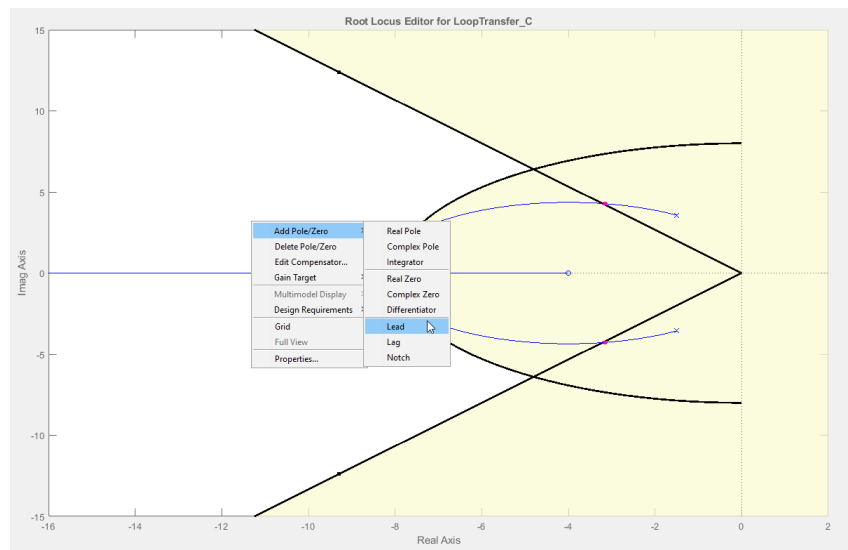


Now we move new zero point

11

In preferences let's change compensator format to Zero/Pole/Gain



Now in C controller we have our solution.

Here overshoot is 29.7

Peak amplitude is 0.844 at time 0.339

transient process time 0.906

After some manipulations with poles and zeros and setting Real zero = -3.04,

Real Pole = -1.07 overshoot is 7.36

Peak amplitude is 0.69 at time 0.52

transient process time 1.61