Control Theory

Laboratory 1

Mikołaj Suchoń and Witold Surdej

Exercise 1

a)

generate tf with no zeros and 10 equally spaced poles

```
Z=[];
K=1;
[~,P]=geteqpols(10);
```

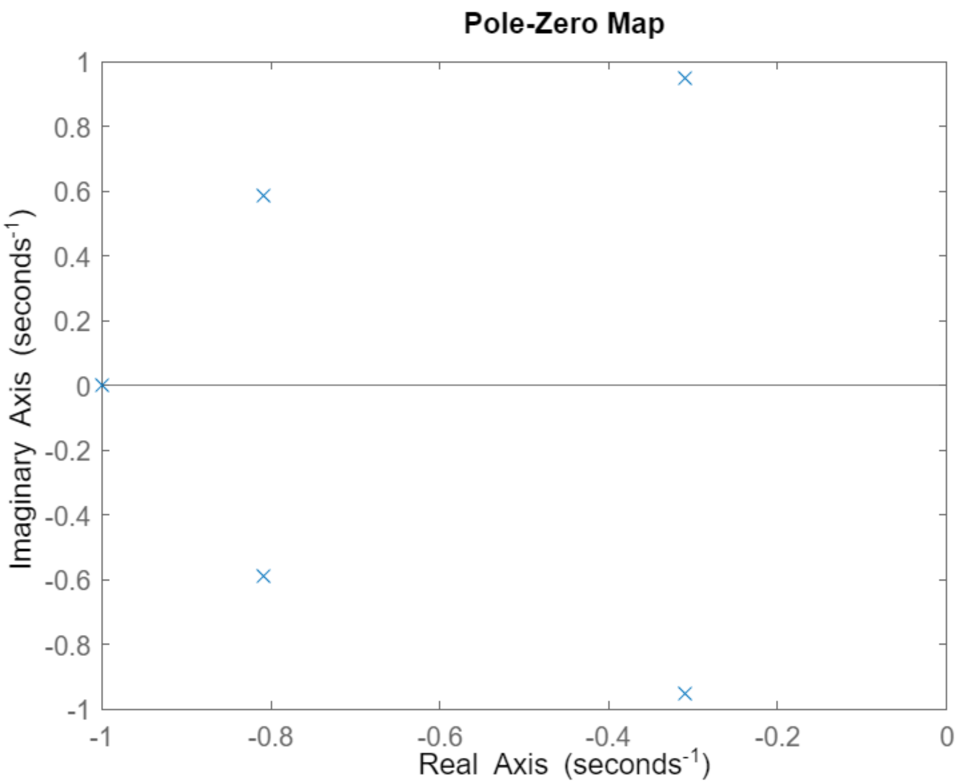
```
Warning: This zpk model has a complex gain or some complex zeros or poles that do not come in conjugate
pairs.
```

generate tf with no zeros and 5 equally spaced poles in LHP

```
P2=P(4:8);
sys = zpk(Z,P2,K);
```

```
Warning: This zpk model has a complex gain or some complex zeros or poles that do not come in conjugate
pairs.
```

```
pzmap(sys)
```



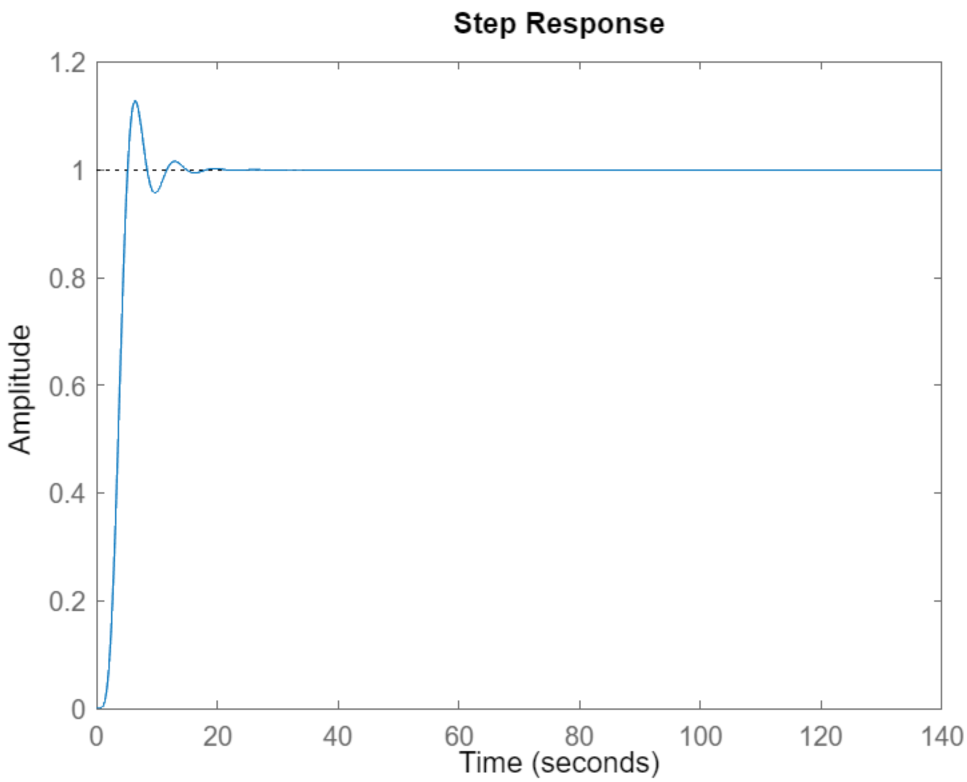
b)

computing step response

```
dat=stepinfo(sys)
```

```
dat = struct with fields:
        RiseTime: 2.5633 + 0.0000i
    TransientTime: 10.8386 + 0.0000i
     SettlingTime: 10.8386 + 0.0000i
      SettlingMin: 0.9090 + 0.0000i
      SettlingMax: 1.1277 + 0.0000i
        Overshoot: 0
       Undershoot: 0
             Peak: 1.1277
         PeakTime: 6.3551
```

```
step(sys)
```



From plot:

t2p,%overshoot,rise time, settling time

```
datp(1).t2p=6.24;
datp(1).ov=(1.13-1)*100;
datp(1).rt=5.07;
datp(1).st=7.7;
```

c)

repeating a) and b) for 6 & 14 poles

```
[~,P2]=geteqpols(6);
```

```
Warning: This zpk model has a complex gain or some complex zeros or poles that do not come in conjugate
pairs.
```

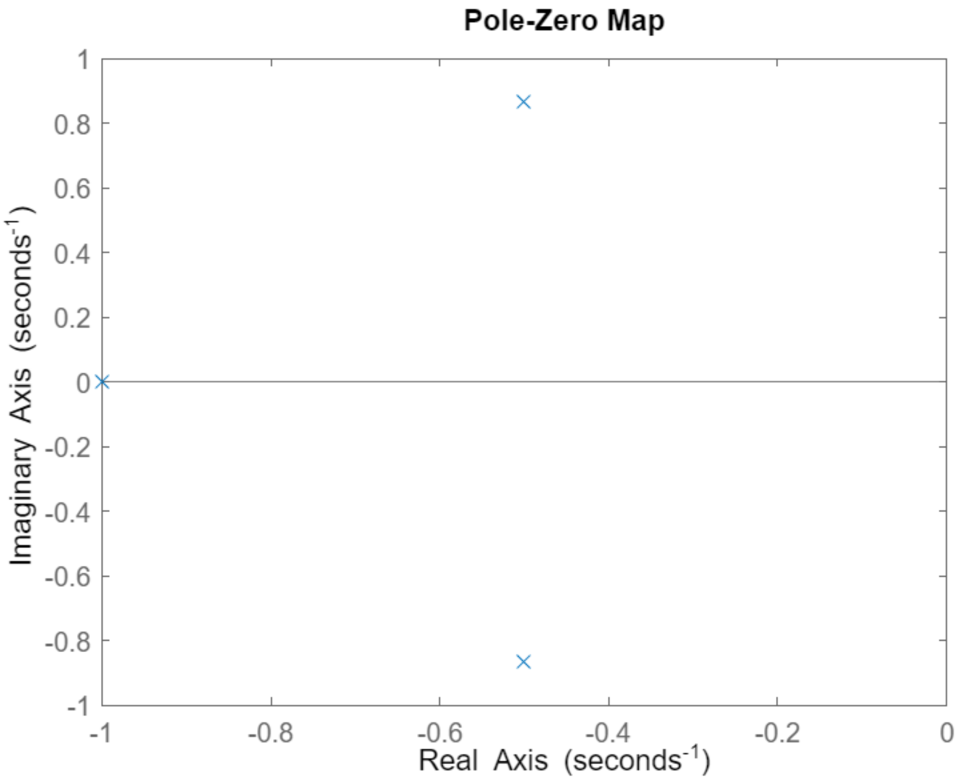
```
[~,P3]=geteqpols(14);
```

```
Warning: This zpk model has a complex gain or some complex zeros or poles that do not come in conjugate
pairs.
```

```
P2=P2(3:5);
sys2 = zpk(Z,P2,K);
```

```
Warning: This zpk model has a complex gain or some complex zeros or poles that do not come in conjugate pairs.
```

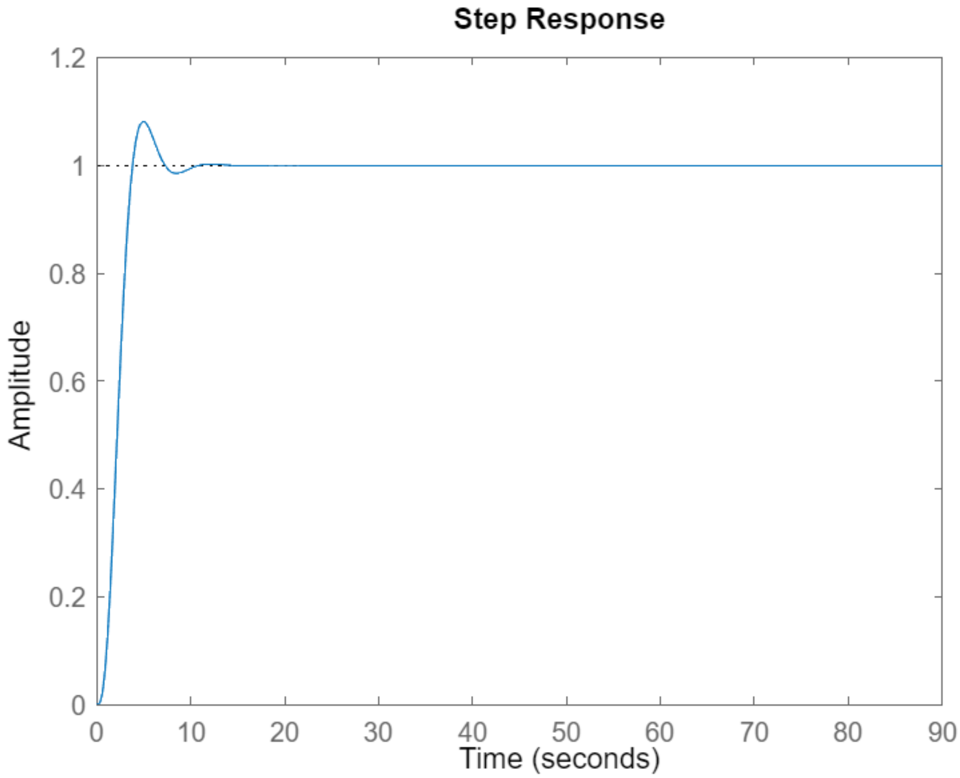
```
pzmap(sys2)
```




```
dat2=stepinfo(sys2)
```

```
dat2 = struct with fields:
         RiseTime: 2.2911 - 0.0000i
    TransientTime: 6.6376 + 0.0000i
     SettlingTime: 6.6376 + 0.0000i
      SettlingMin: 0.9050 + 0.0000i
      SettlingMax: 1.0814 + 0.0000i
        Overshoot: 0
       Undershoot: 0
             Peak: 1.0814
         PeakTime: 4.8815
```

```
step(sys2)
```



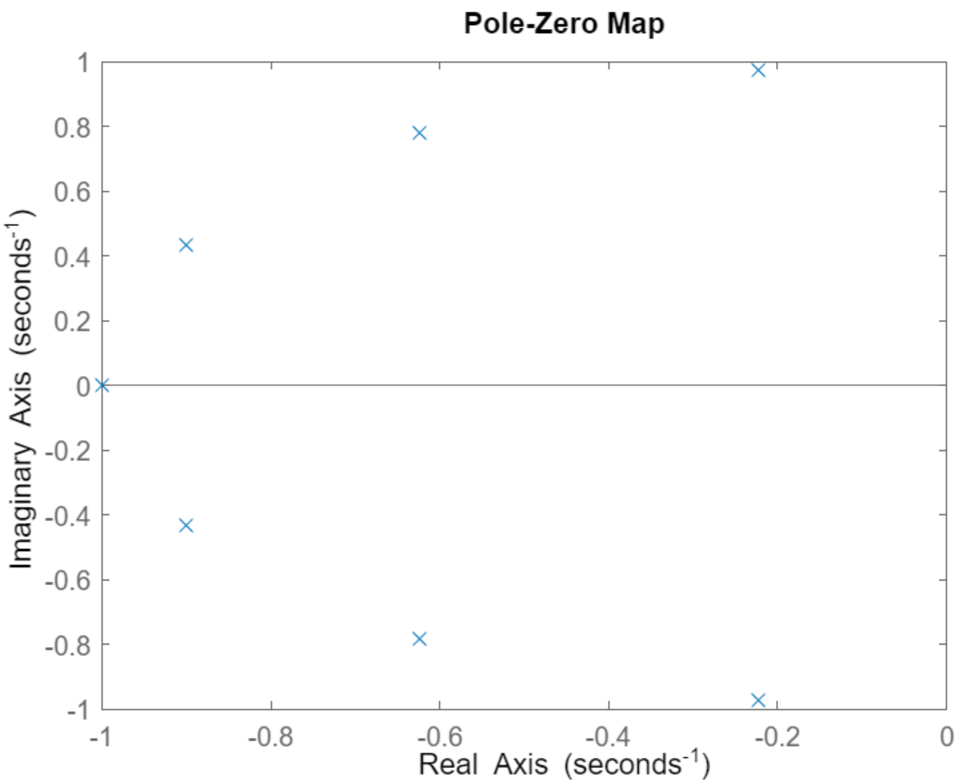
From plot:

t2p,%overshoot,rise time, settling time

```
datp(2).t2p=4.91;
datp(2).ov=(1.08-1)*100;
datp(2).rt=3.78;
datp(2).st=6;

P3=P3(5:11);
sys3 = zpk(Z,P3,K);
```

```
Warning: This zpk model has a complex gain or some complex zeros or poles that do not come in conjugate
pairs.
```

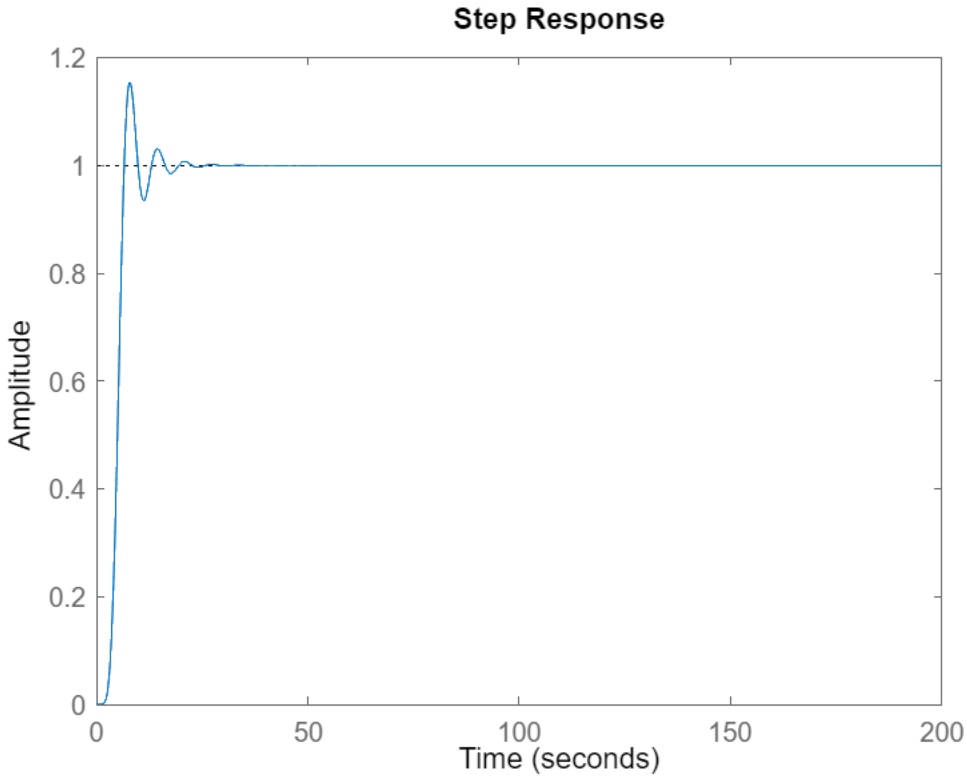
```
pzmap(sys3)
```




```
dat3=stepinfo(sys3)
```

```
dat3 = struct with fields:
         RiseTime: 2.7880 - 0.0000i
    TransientTime: 15.2063 + 0.0000i
     SettlingTime: 15.2063 + 0.0000i
      SettlingMin: 0.9046 + 0.0000i
      SettlingMax: 1.1541 + 0.0000i
        Overshoot: 0
       Undershoot: 0
             Peak: 1.1541
         PeakTime: 7.7554
```

```
step(sys3)
```



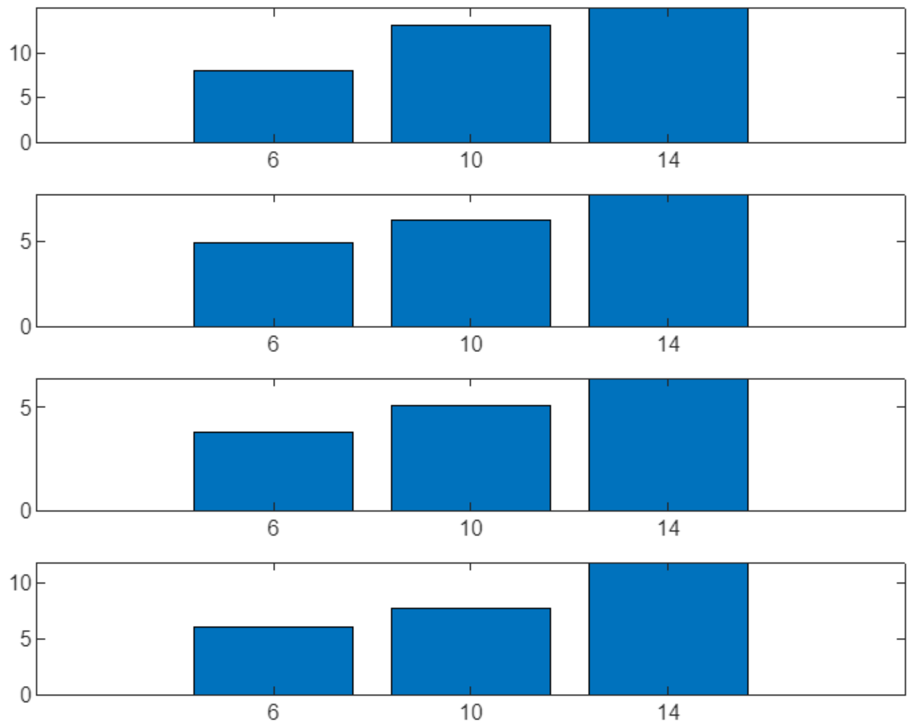
From plot:

t2p,%overshoot,rise time, settling time

```
datp(3).t2p=7.76;
datp(3).ov=(1.15-1)*100;
datp(3).rt=6.42;
datp(3).st=11.8;
```

Data visualisation

```
for i=1:3
vis(1,i)=datp(i).ov;
vis(2,i)=datp(i).t2p;
vis(3,i)=datp(i).rt;
vis(4,i)=datp(i).st;
end

for i=1:4
subplot(4,1,i)
bar([10,6,14],vis(i,:))
end
```



From subplots we can conclude that the higher number of poles the higher the overshoot, rise time, time to peak and settlnig time.

```
function [sys,P]=geteqpols(n)

theta=zeros(1,n);
rho=zeros(1,n);

theta(1)=0;
rho(1)=1;

for i = 1:n-1
theta(i+1)=(2*i*pi/n);
rho(i+1)=1;
end
K = 1;
[x,y]=pol2cart(theta,rho);
for i=1:length(x)
P(i)=x(i)+y(i)*1i;
end
Z=[];
sys = zpk(Z,P,K);

end
```

Exercise 2:

In this task we will consider a stiff system with poles close to the imaginary axis, and second far from it (all poles in the LHP - Left Half-Plane). Based on the transfer funtion and the nyquist plot of the system we will try to determine its stability and regions of interest, as well as stability of the closed-loop system.

```
clear all
clc
```

1) Defining the transfer function

our system is defined by the following transfer function:

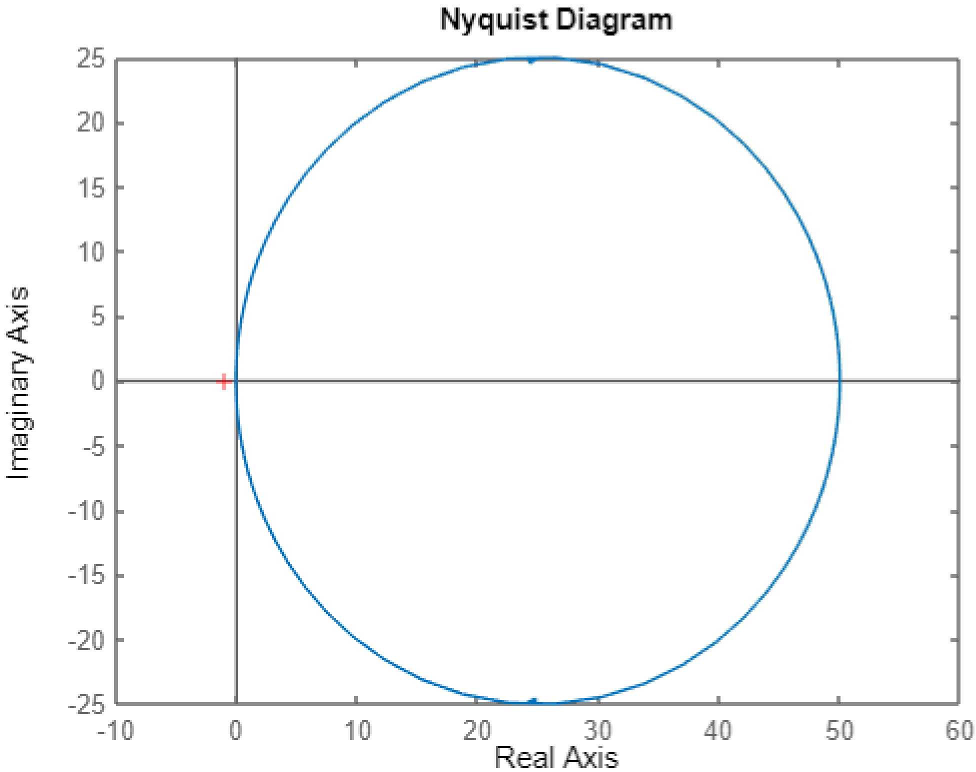
$$G(s) = \frac{1}{(s+20)(s+0.001)(s+1)}$$

2) Checking stability of the open-loop system

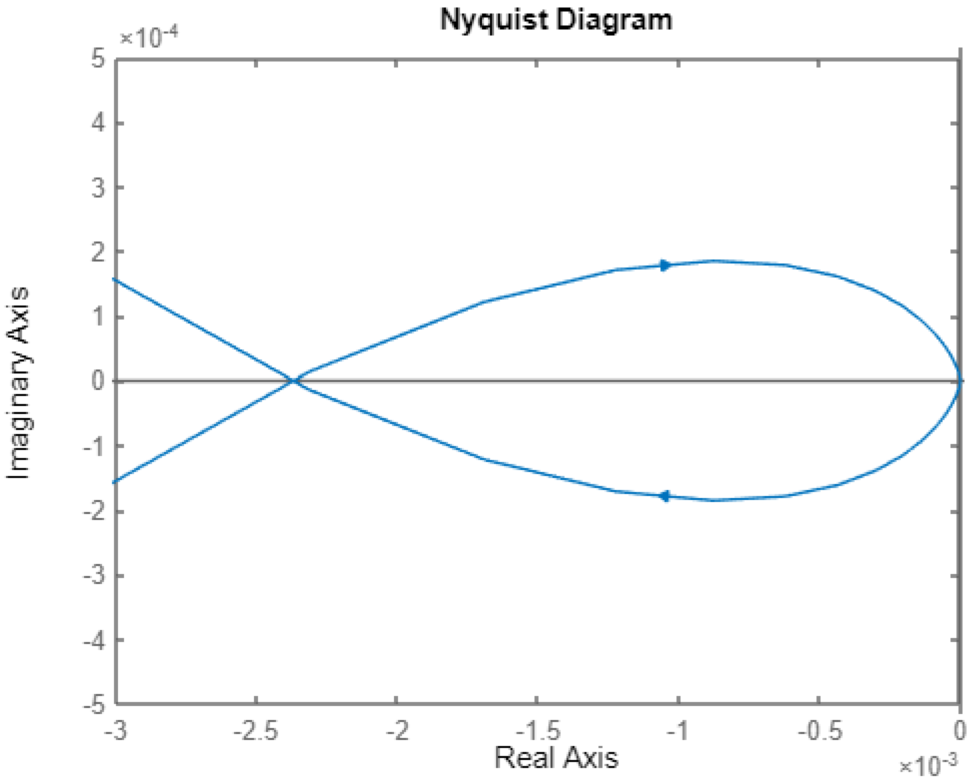
From the lectures we know that our system G(s)=N(s)/D(s) is stable iif all of the roots of the characteristic polynomial need to lie in the left-half plane (LHP) ie.

all the zeros of the denominator have to be negative. Our poles are at s={-20, -0.001, -1} therefore the system is stable.

```
N=1;                        %numerator
D=[1 21.001 20.021 0.02];   %denumerator
sys=tf(N,D);                %transfer function

figure()
nyquist(sys)                %plotting nyquist plot
```




```
figure()
nyquist(sys)                              %plotting nyquist plot
axis([-0.003 0 -0.0005 0.0005])           %zooming in on the region of interest
```



3) Checking close-loop system stability

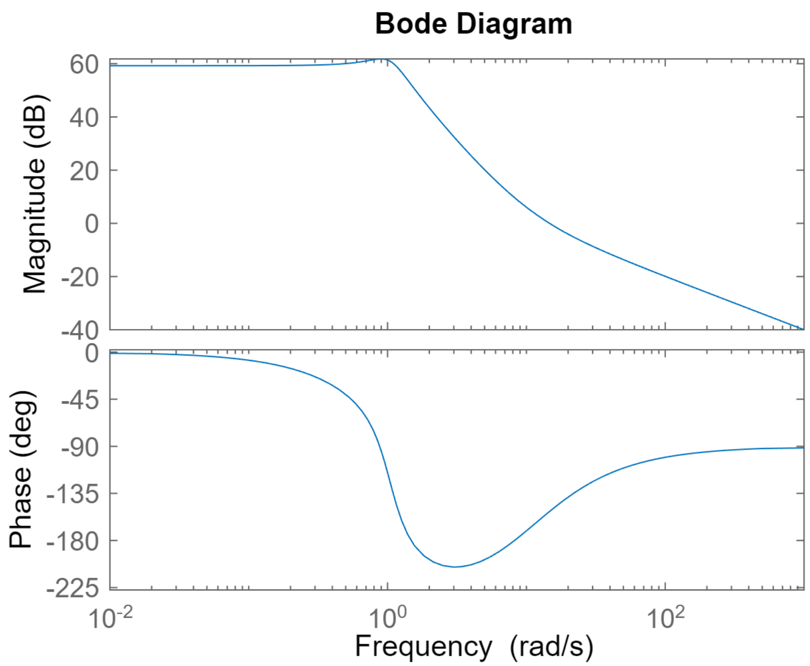
We knew this region would be interesting, as stiff systems have some poles close to the origin of the coordinate frame, and we have to verify that there is no more than 1 pole on the imaginary axis, otherwise it is unstable.

Given the fact that the open-loop system is stable,and the Nyquist plot of the open-loop system presented above does not encircle the point (-1, j0) in the complex plane we can conclude that the closed-loop system is stable.

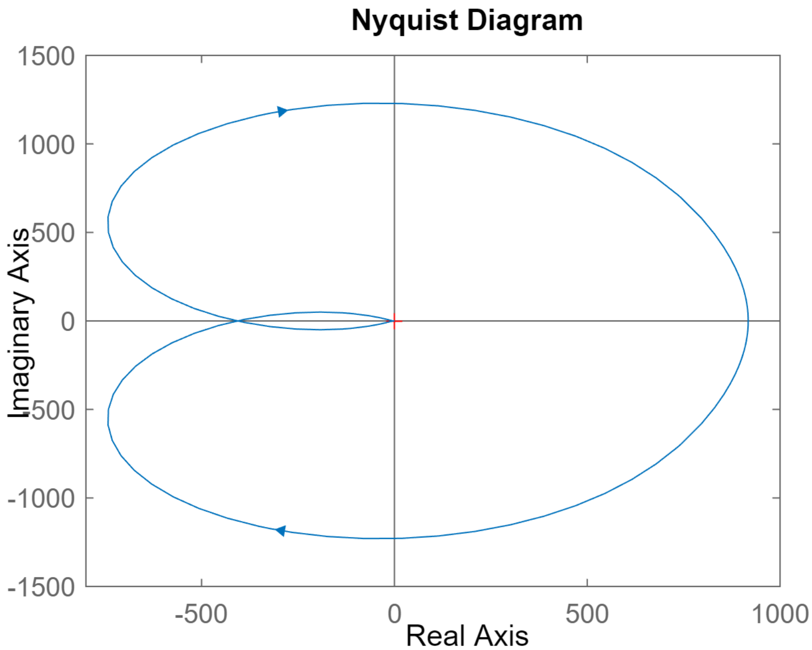
```
K=10;
Z=[-10 -10];
P=[-1 -0.3-1i -0.3+1i];
sys=zpk(Z,P,K)
```

```
sys =
 
        10 (s+10)^2
  -------------------------
  (s+1) (s^2 + 0.6s + 1.09)
 
Continuous-time zero/pole/gain model.
```

```
bode(sys)
```

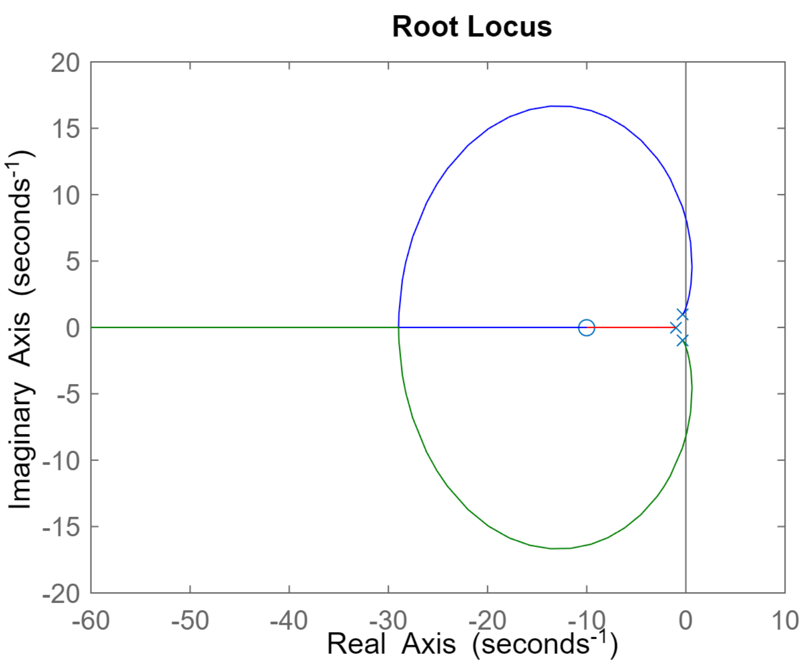



```
nyquist(sys)
```



Nyquist Diagram
1500
1000
500
0
-500
-1000
-1500
-500
0
500
1000
Real Axis
Imaginary Axis


```
rlocus(sys)
```



Root Locus
20
15
10
5
0
-5
-10
-15
-20
-60
-50
-40
-30
-20
-10
0
10
Real Axis (seconds^{-1})
Imaginary Axis (seconds^{-1})

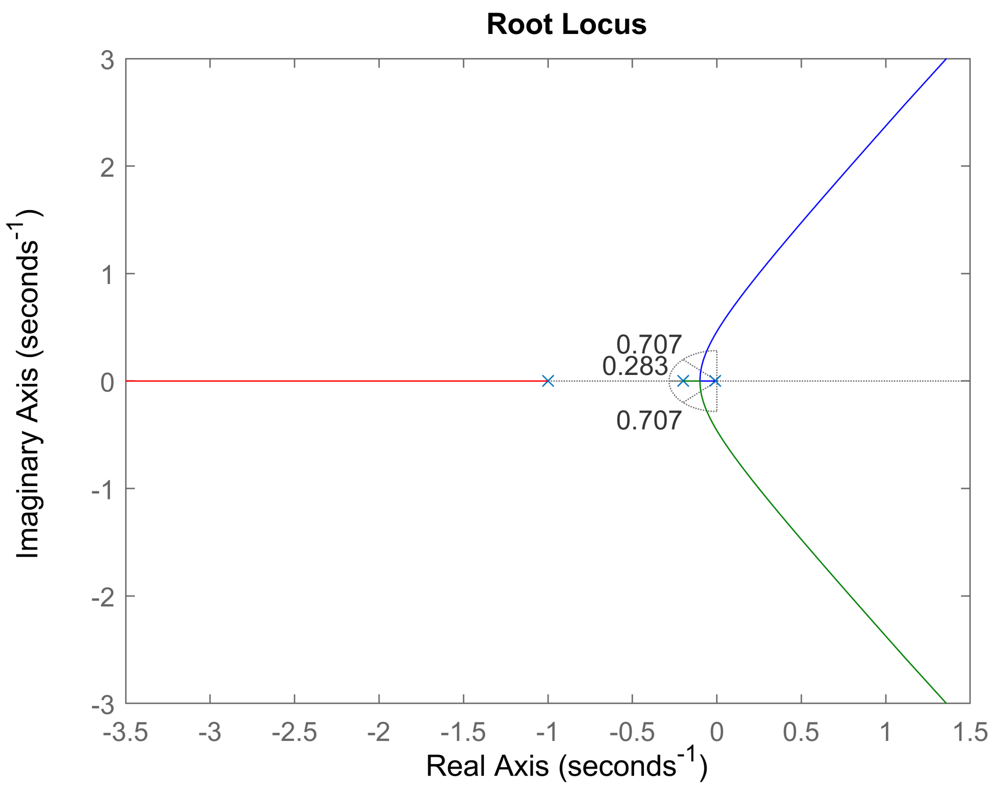
Exercise 4:

In htis task we investigate two methods of design of a feedback control system. Firstly we will utilize Matlab's Control System Designer GUI, then we will apply Ziegler-Nichols method for the same system, and finally we will compare the results, as well as ease of use of each method.

```
clear all
clc
```

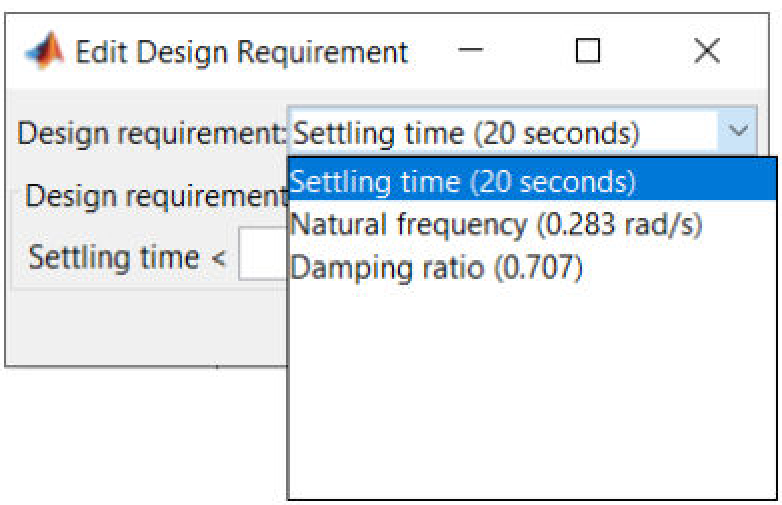
1) Defining the system and time domain specifications:

```
%defining the system of interest

K=8;                %gain
Z=[];               %zeros
P=[-0.01,-0.2,-1]; %poles
sys=zpk(Z,P,K);    %transfer funtion

%defining system specifications:

zeta=0.707; %damping ratio of dominant close-loop poles
wn=0.2829;  %natural frequency of dominant close-loop poles

%plotting the rlocus plot of the system, and generating
% a grid of constant damping factors and natural frequencies

rlocus(sys)
sgrid(zeta,wn)
```



2) The Root-Locus Design GUI (sisotool) tool of Matlab Control System Toolbox:

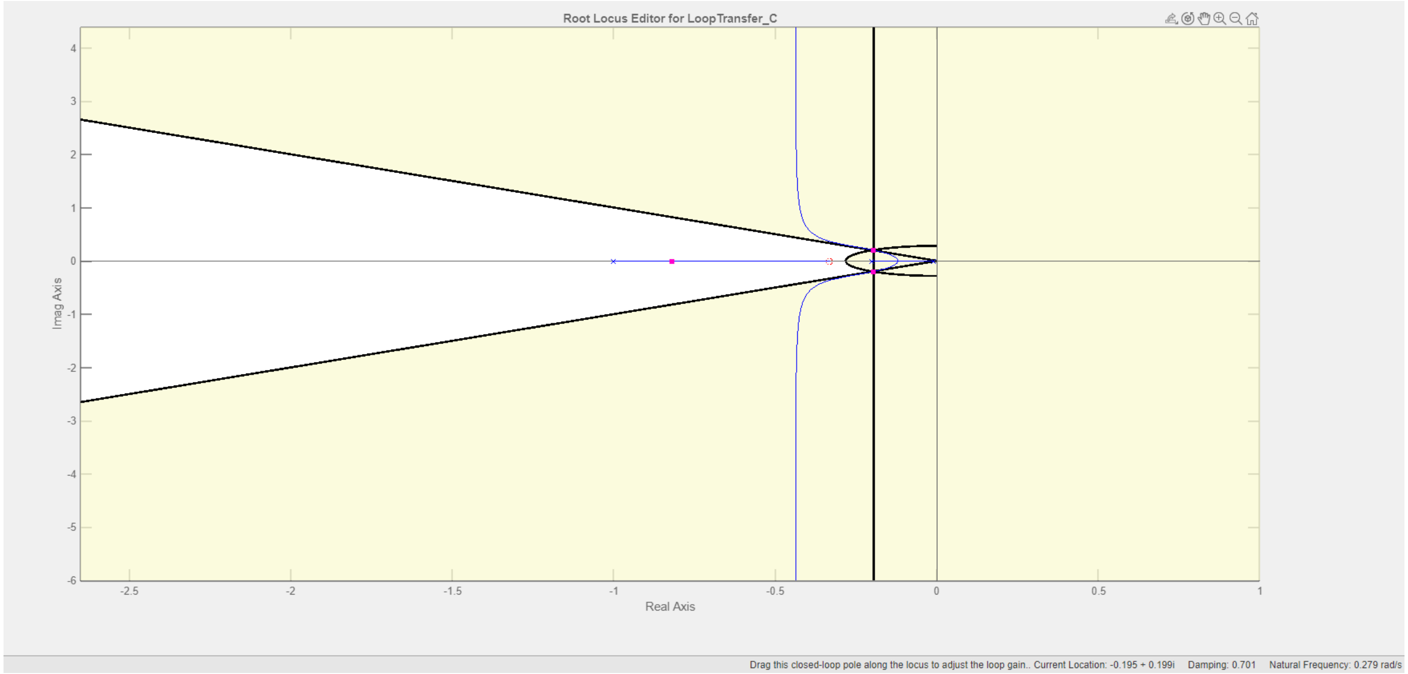
- Firsty we set up system specifications:



rys.1 system specifications

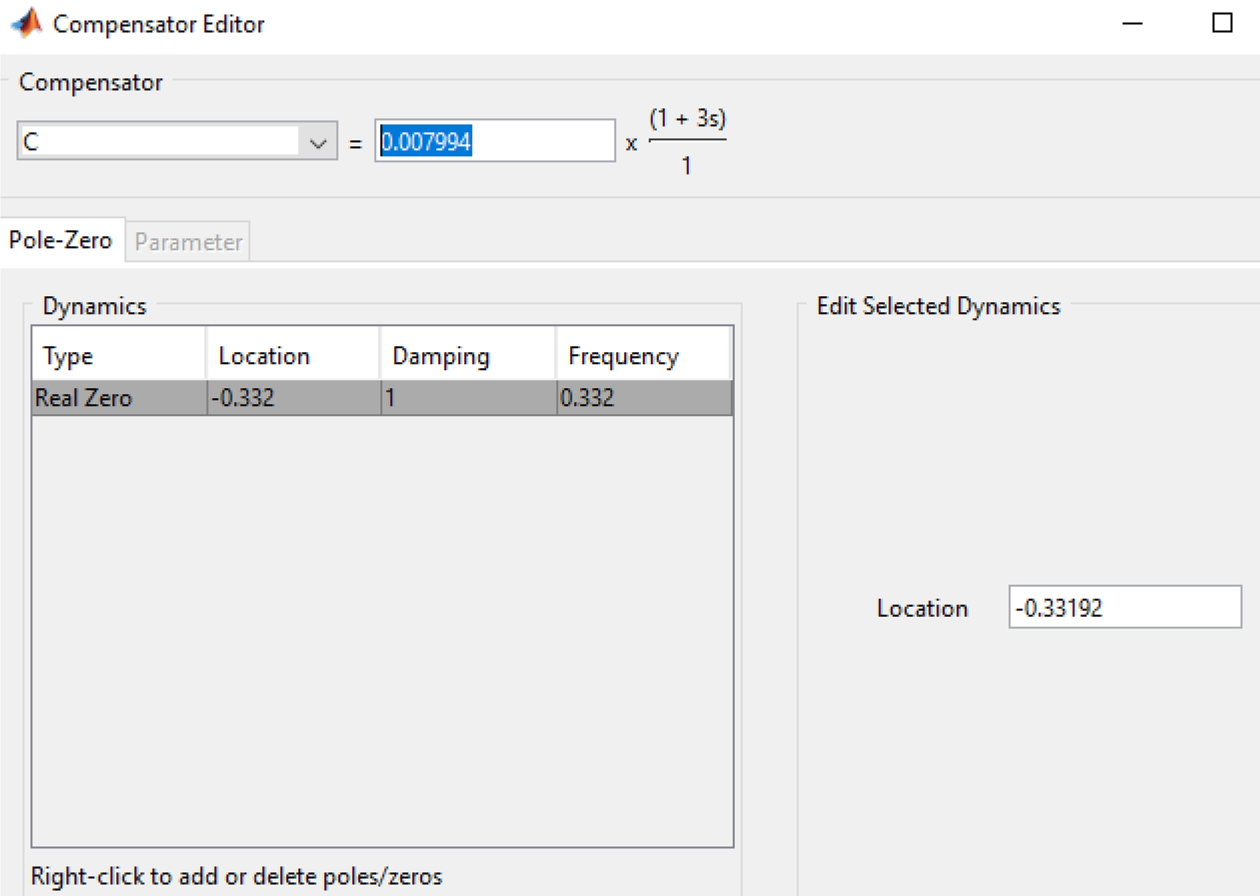
- Secondly we add a zero to the system to the right of the pole with the smallest magnitude. Adding zero allows us to shift the dominant poles to the left half-plane.

• Finally we adjust the position of the poles such that they fit within the system specifictations. As a result we obtain the following locus plot:



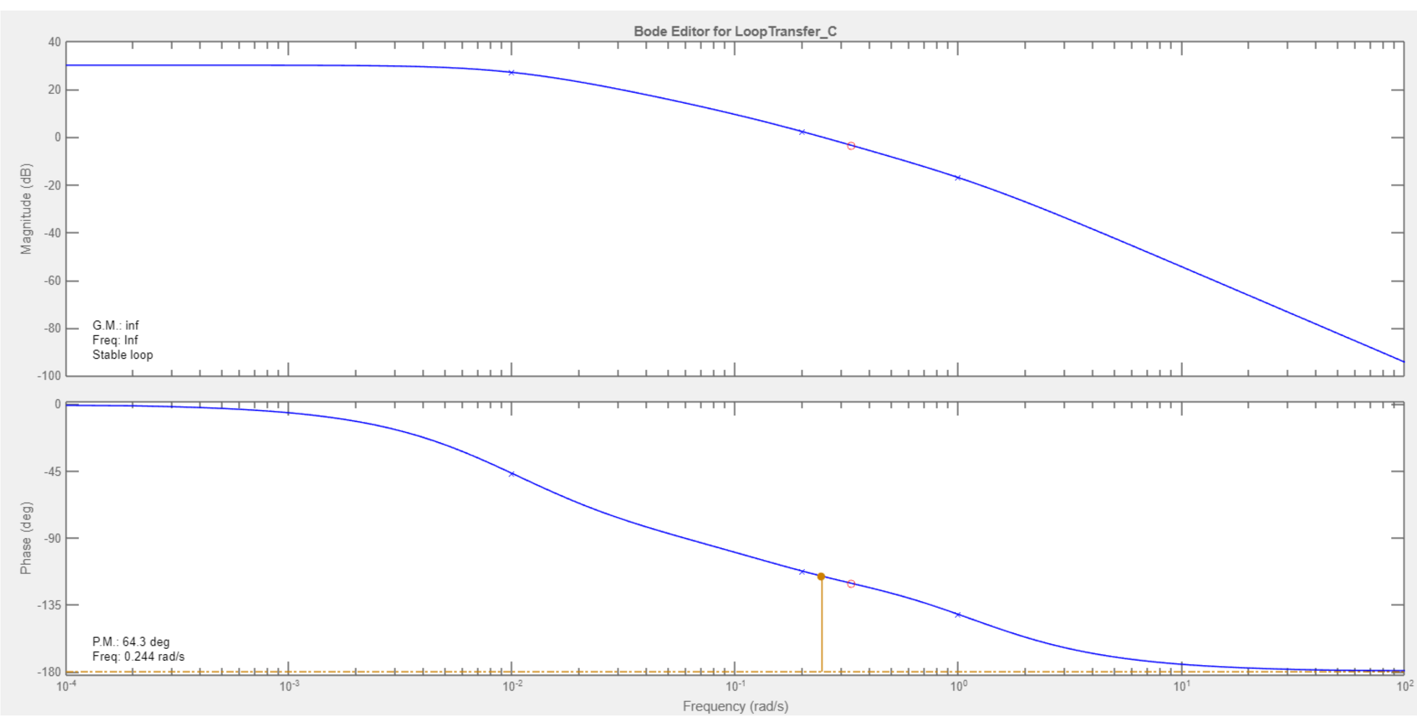
rys.2 Root locus plot with system specification limitations

Based on the compensator data:

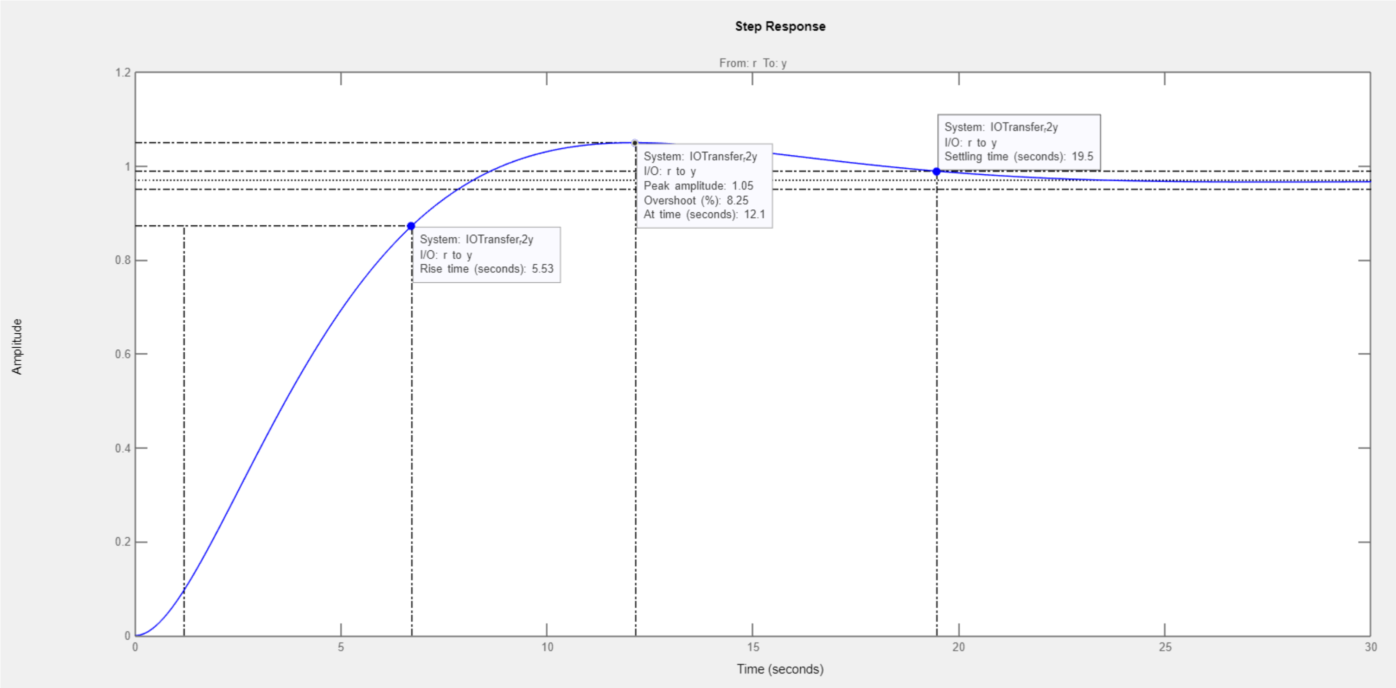


rys.3 obtained compensator values

the PD controller is approximately 0.008 + 0.024s.



rys.4 Bode plot of the system



rys.5 Close-loop step response of the system

```
%loading finished system from Control Designer

load('ControlSystemDesignerSession.mat')
%controlSystemDesigner('ControlSystemDesignerSession.mat')
```

3) Ziegler-Nichols method of tuning a PID controller:

It is performed by setting the *I* (integral) and *D* (derivative) gains to zero. The P (proportional) gain (Kp) is then increased until it reaches the ultimate gain (Ku), at which the output of the control loop has stable and consistent oscillations. Ku and the oscillation period Tu are then used to set the P, I, and D gains depending on the type of controller used and behaviour desired:

Ziegler–Nichols method[1]

Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.8\overline{3}T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.10K_uT_u$
classic PID[2]	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$
Pessen Integral Rule[2]	$0.7K_u$	$0.4T_u$	$0.15T_u$	$1.75K_u/T_u$	$0.105K_uT_u$
some overshoot[2]	$0.3\overline{3}K_u$	$0.50T_u$	$0.3\overline{3}T_u$	$0.6\overline{6}K_u/T_u$	$0.1\overline{1}K_uT_u$
no overshoot[2]	$0.20K_u$	$0.50T_u$	$0.3\overline{3}T_u$	$0.40K_u/T_u$	$0.06\overline{6}K_uT_u$

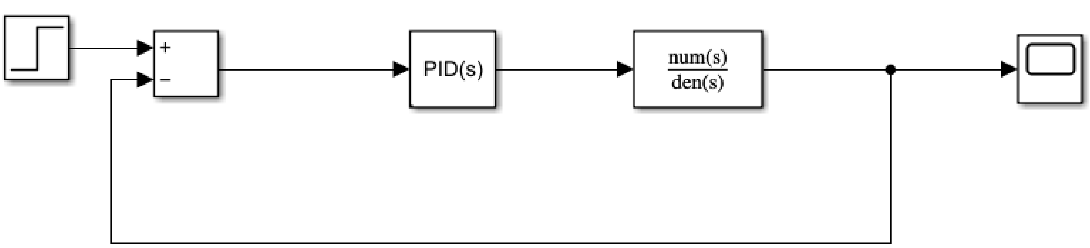
The ultimate gain (K_u) is defined as 1/M, where M = the amplitude ratio, $K_i = K_p/T_i$ and $K_d = K_pT_d$.

rys.6 Table for PID tunning using Ziegler-Nichols method.

- The ultimate gain Ku can be read from the rlocus plot present in the part 1) of the task. It lays on the crossing of the locus line and the imaginary axis. The value for our system is:

```
%From locusplot in 1)
Ku=0.0319;
```

- The fastest way to find Tu is by creating a simulink model of our system, then using the cursor measurement function within the scope we can read the period Tu directly from the obtained plot.



rys.7 simulink model of our system

rys.8 scope of the system for the ultimate gain Ku with period of oscillation Tu marked on top of it.

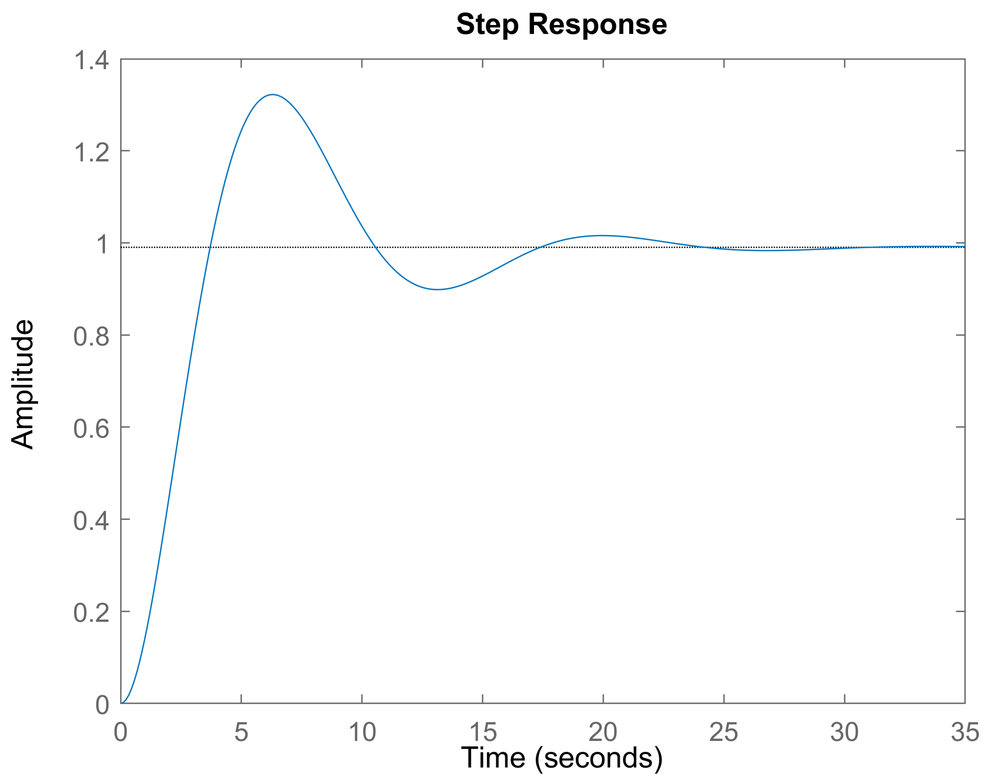
```
%From simulink
% =open('Ex4sim.slx')
Tu=13.653;
```

- Having found Ku and Tu we can now find the remaining values neededend to tune our PD.

```
Kp=0.8*Ku;
Td=0.125*Tu;
Kd=0.1*Ku*Tu;


W=pid(Kp,0,Kd);
H=1;
```

- Finally we can create the system with feedback tuned by the obtained PD, and plot its step function

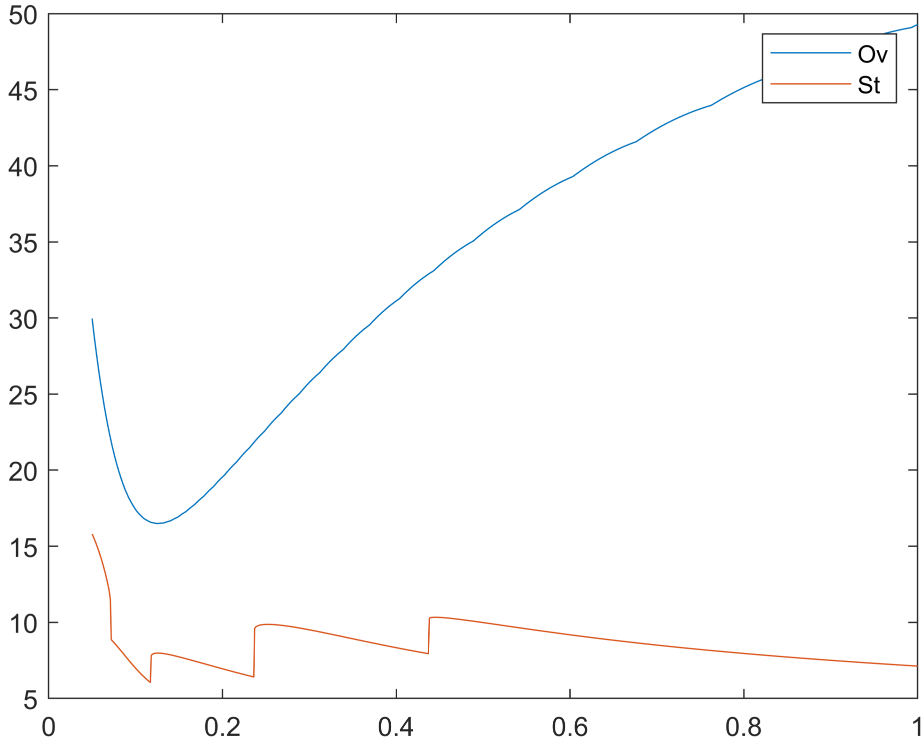
```
Y=feedback(sys*W,H);
step(Y)
```




```
stepinfo(Y)
```

```
ans = struct with fields:
         RiseTime: 2.5313
    TransientTime: 21.4618
     SettlingTime: 21.4618
      SettlingMin: 0.8985
      SettlingMax: 1.3220
        Overshoot: 33.4962
       Undershoot: 0
             Peak: 1.3220
         PeakTime: 6.2913
```

```
Kd=0.05:0.001:1;

for i=1:length(Kd)
W=pid(Kp,0,Kd(i));
H=1;
Y=feedback(sys*W,H);
G=stepinfo(Y);
Ov(i)=G.Overshoot;
St(i)=G.SettlingTime;
end
figure()
plot(Kd,Ov,Kd,St)
legend('Ov','St')
```



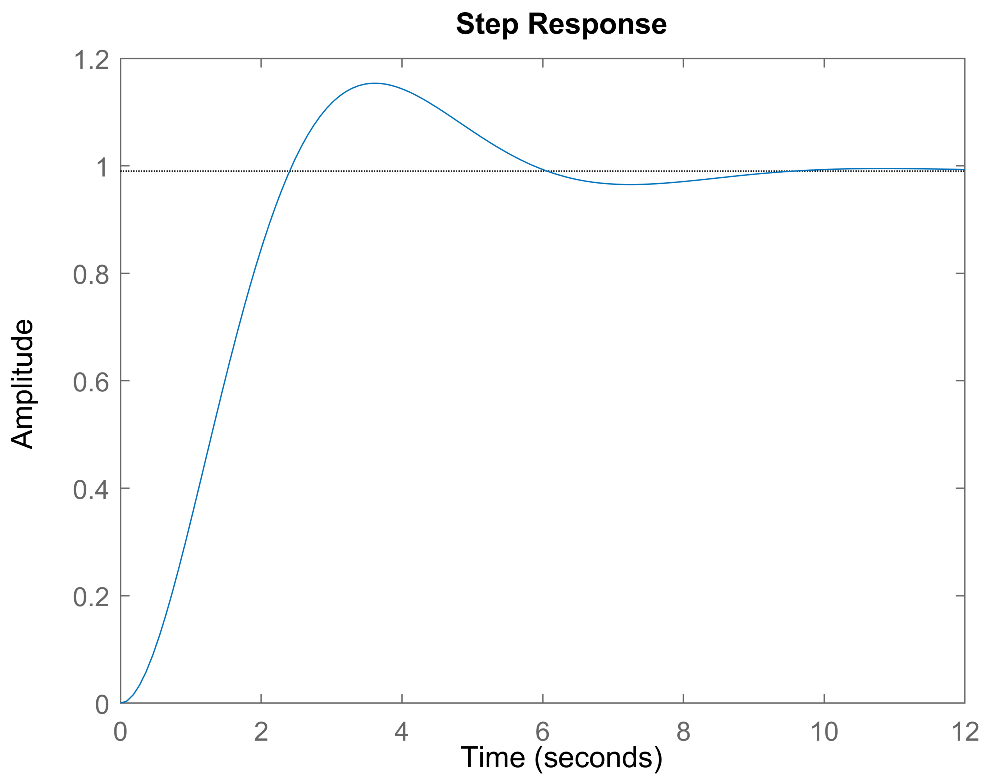

```
[Ovopt,ind]=min(Ov)
```

```
Ovopt = 16.4931
ind = 76
```

```
Stopt=St(ind)
```

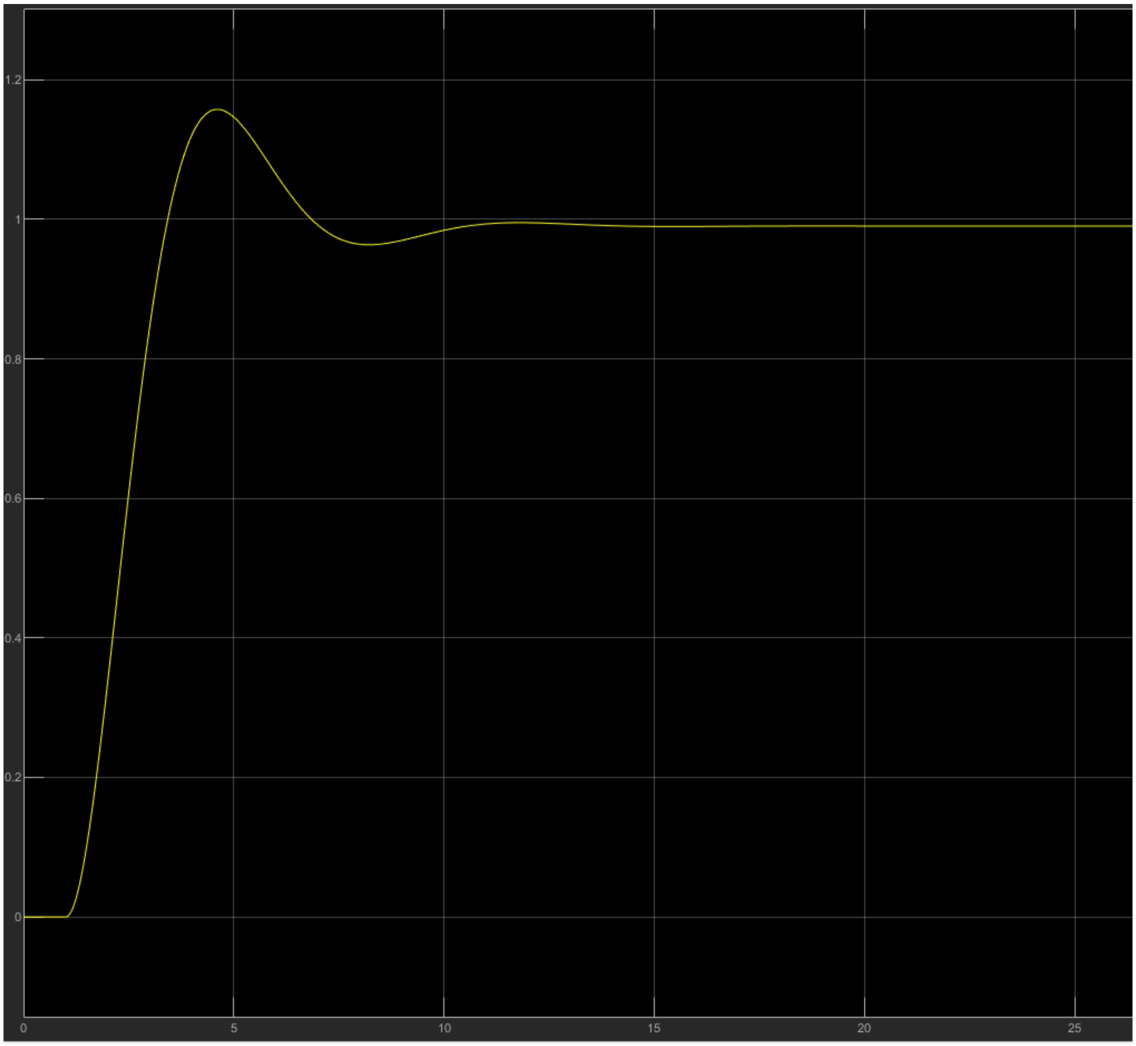
```
Stopt = 7.9807
```

```
Kd=Kd(ind);

W=pid(Kp,0,Kd);
H=1;
Y=feedback(sys*W,H);
step(Y)
```




```
stepinfo(Y)
```

```
ans = struct with fields:
         RiseTime: 1.6298
    TransientTime: 7.9807
     SettlingTime: 7.9807
      SettlingMin: 0.9229
      SettlingMax: 1.1536
        Overshoot: 16.4931
       Undershoot: 0
             Peak: 1.1536
         PeakTime: 3.5739
```



rys.9 step response after fine tuning using ziegler-nichols method from simulink

4) Results:

Method	Peak Amplitude	Overshoot	Rise Time	Settling Time
Root-Locus	1.050	8.25%	5.529	19.500
Ziegler-Nichols	1.579	57.95%	2.081	28.839

rys.10 table containing key characteristics of each designed system.

Based on the results presented above we can conclude that the Root-Locus method is better than Ziegler-Nichols. We have obtained more accurate values with it, while having to perform less work. In general Root-Locus method will always have an advantage, because it does not require any hand calculations which due to rounding errors increase the overall error of the obtained results. Root-Locus method was more intuitive, and quicker (since we only had to adjust pre-made plots). Due to those considerations we would always prefer to use Root-Locus.