

به نام نور



پروژه نهایی طراحی کنترلر

استاد درس: دکتر الستی

دانشجویان: ریحانه نیکوبیان 99106747

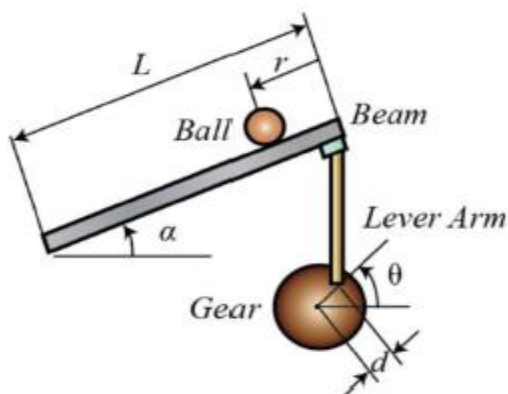
محمد جواد شمس الدین سعید 99106266

سال تحصیلی: 1402

شرح پروژه:

تعریف مسئله

سیستم **Ball and Beam**، از معروف ترین و ساده ترین سیستم های کنترل است. این سیستم شامل یک تیر یلند است که قابلیت حرکت توپ داخل آن را دارد. هدف کنترلی در این سیستم، کنترل مکان توپ دقیقاً در وسط تیر است. به این منظور یک سنسور التراسونیک برای تشخیص مکان و سرعت توپ در هر لحظه و یک سروو موتور در وسط یا اطراف تیر برای تولید حرکت دورانی در تیر و کنترل مکان توپ تعبیه شده است.



فایل شبیه سازی شده ی این سیستم با عنوان mod1BB۱۵ موجود می باشد که می توان از طریق آن، رابطه ی میان زاویه ی موتور (θ) و موقعیت توپ بر روی تیر (r) را استخراج کرد. می خواهیم زاویه ی θ را با یک موتور و گیربکس کاهنده یا ضریب ۵ کنترل کنیم. تابع تبدیل موتور بصورت زیر می باشد.

$$\frac{\theta}{V} = \frac{0.0274}{0.003228 s^2 + 0.003508 s}$$

رابطه ی ولتاژ یا مکان مطلوب بصورت زیر است.

$$R = 2 V$$

خواسته ها

- فراجش کمتر از ۲۰ درصد
- زمان نشست کمتر از ۸ ثانیه

(1)

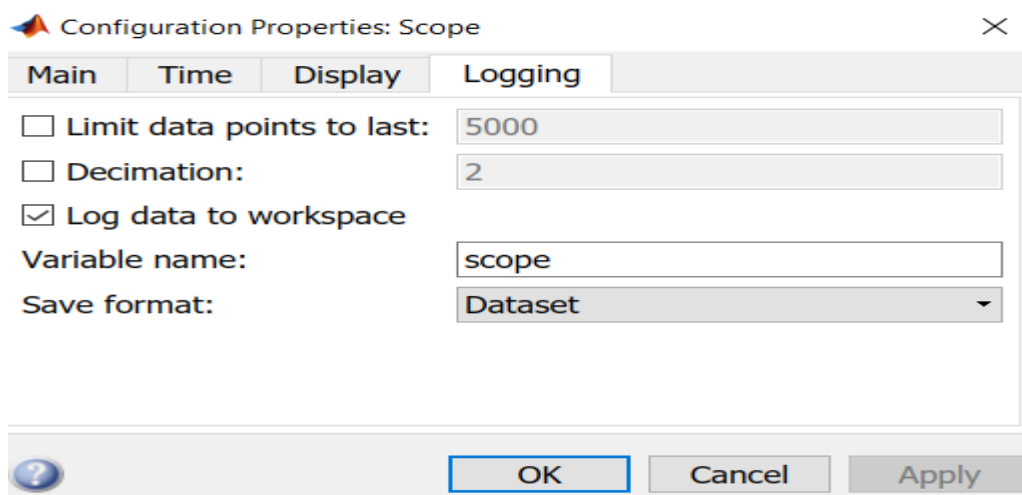
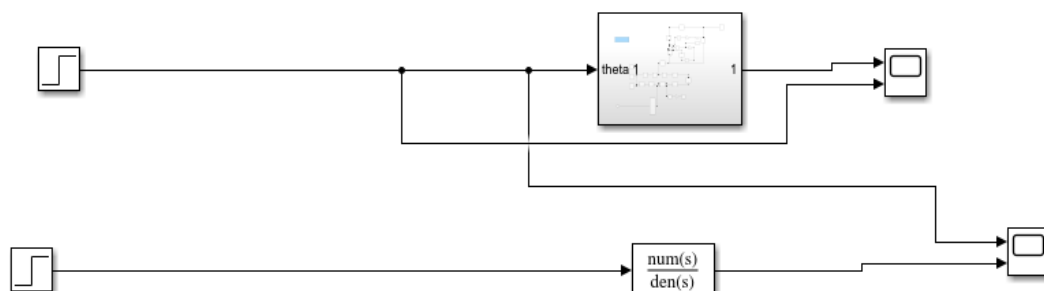
۱. با استفاده از مدل شبیه‌سازی شده و بهره‌گیری از روش‌های شناسایی سیستم (*System Identification*)، تابع تبدیل

$$\frac{R}{\theta}$$

را بدست آورید.

در این پروژه شبیه‌سازی مدل ball and beam به ما داده شده و سیستم طراحی شده است طوری که با اعمال ورودی زاویه، موقعیت مطلوب حاصل می‌شود. حال از ما خواسته شده تابع تبدیل این سیستم را به دست بیاوریم که از سیستم identification استفاده می‌کنیم.

سیستم شبیه‌سازی شده داخل باکس قرار دارد و ورودی پله (زاویه) را به این سیستم شبیه‌سازی شده اعمال می‌کنیم. خروجی موقعیت را به همراه زاویه در scope بالایی نمایش می‌دهیم و از این طریق، ورودی، خروجی، و زمان را به workspace می‌بریم.



کد لازم برای ذخیره ورودی و خروجی سیستم و تشخیص سیستم به شرح زیر است:

Collect Data

```
dt = 0.01; %sample time
input = scope{2}.Values
```

timeseries

Common Properties:

Name: ''
Time: [3001x1 double]
TimeInfo: [1x1 tsdata.timemetadata]
Data: [3001x1 double]
DataInfo: [1x1 tsdata.datametadata]

[More properties, Methods](#)

```
output=scope{1}.Values
```

timeseries

Common Properties:

Name: ''
Time: [3001x1 double]
TimeInfo: [1x1 tsdata.timemetadata]
Data: [3001x1 double]
DataInfo: [1x1 tsdata.datametadata]

[More properties, Methods](#)

```
t=input.Time; %time

% Build the step input
u = input.Data;

% Simulate a step response
yreal =output.Data;

plot(t, [u, yreal])
grid on
legend(['u'; 'y']);
```

Identify a transfer function model

```
% Fit data to a transfer function with unknown delay term
data = iddata(yreal, u, dt);
Gest = tfest(data, 2, 0, NaN)
```



Warning: For transient data (step or impulse experiment), make sure that the change in input signal does not happen too early relative to the order of the desired model. You can achieve this by prepending sufficient number of zeros (equilibrium values) to the input and output signals. For example, a step input must be represented as `[zeros(nx,1); ones(N,1)]` rather than `ones(N,1)`, such that `nx > model order`.

Gest =

From input "u1" to output "y1":
0.3494
 $\exp(-0.02s) * \frac{0.3494}{s^2 + 0.2381s + 4.578e-13}$

Continuous-time identified transfer function.

Parameterization:

Number of poles: 2 Number of zeros: 0

Number of free coefficients: 3

Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using TFEST on time domain data "data".

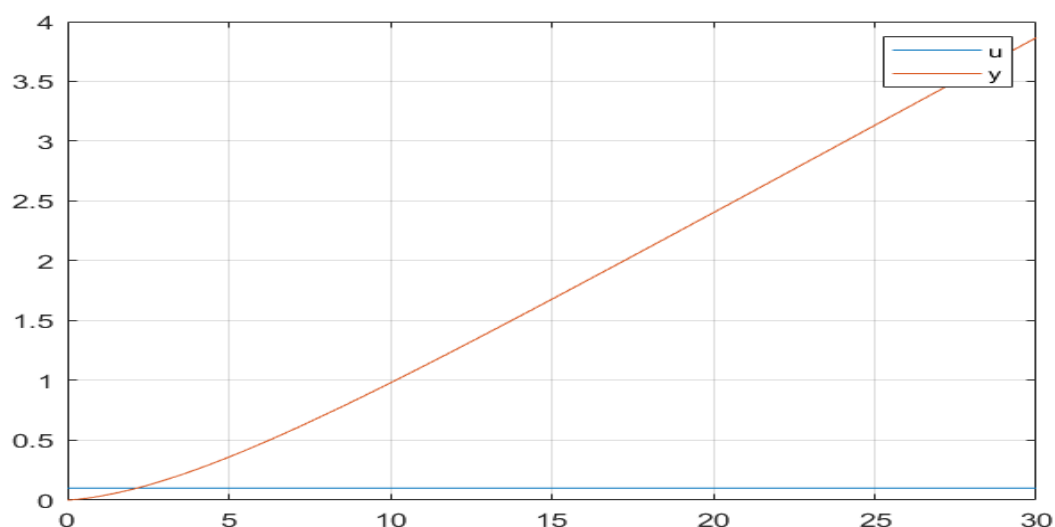
Fit to estimation data: 100%

FPE: 2.18e-13, MSE: 2.173e-13

Validate the model

```
opt = compareOptions('InitialCondition', 'z');
compare(data, Gest, opt);
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
grid on
```

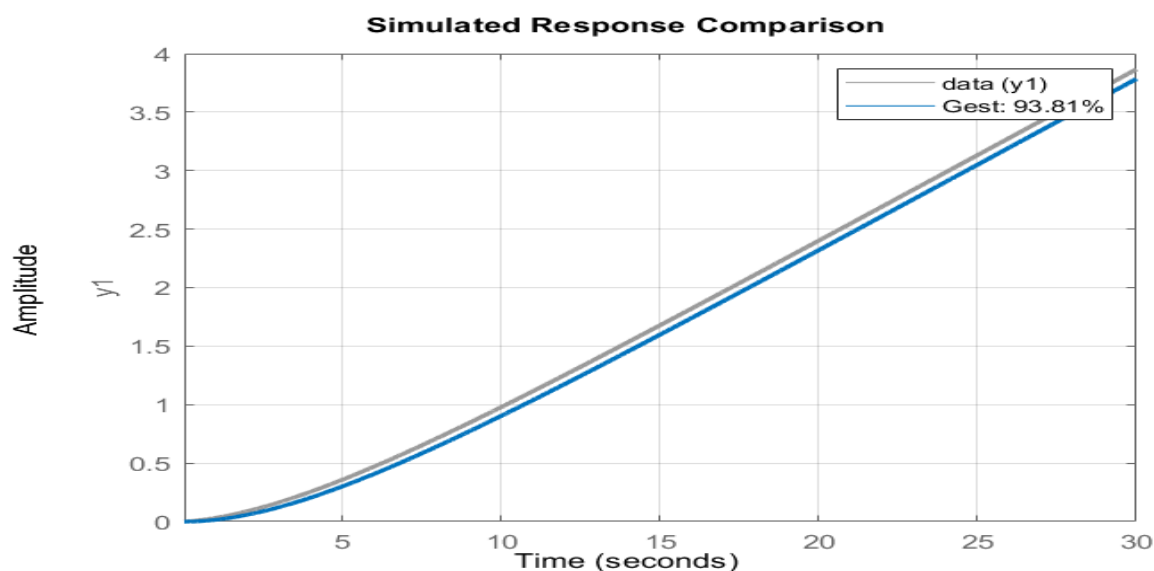
بخش اول کد داده های ورودی و خروجی و تایم را ذخیره می کند و ورودی را بر حسب زمان رسم می کند (u ورودی و y خروجی است)



بخش بعدی تابع تبدیل را با توجه به دیتاها تقریب می زند و خروجی تابع تبدیل تشخیص داده شده و خروجی که داشتیم را رسم می کند و میزان دقت تشخیص را مشخص می کند.

در کد tfest دیتا (خروجی، ورودی، فاصله زمانی)، صفر و قطب تابع تبدیل و وجود یا عدم وجود دیلی مشخص می کنیم (Nan یعنی خودکار اگر وجود داشت مقدار دیلی را تعیین کند)

ما برای این سیستم تعداد صفر و قطب را به ترتیب 2،0 قرار می دهیم و stop time=30 قرار می دهیم و sample time را 0.01s قرار می دهیم. نتایج به شرح زیر است:



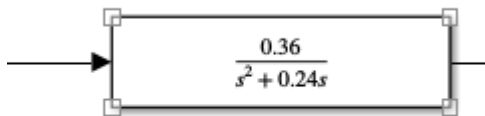
می بینیم که مدل تشخیص داده شده 93 درصد به سیستم اصلی نزدیک است. تابع تبدیل تشخیص داده شده :

Gest =

From input "u1" to output "y1":

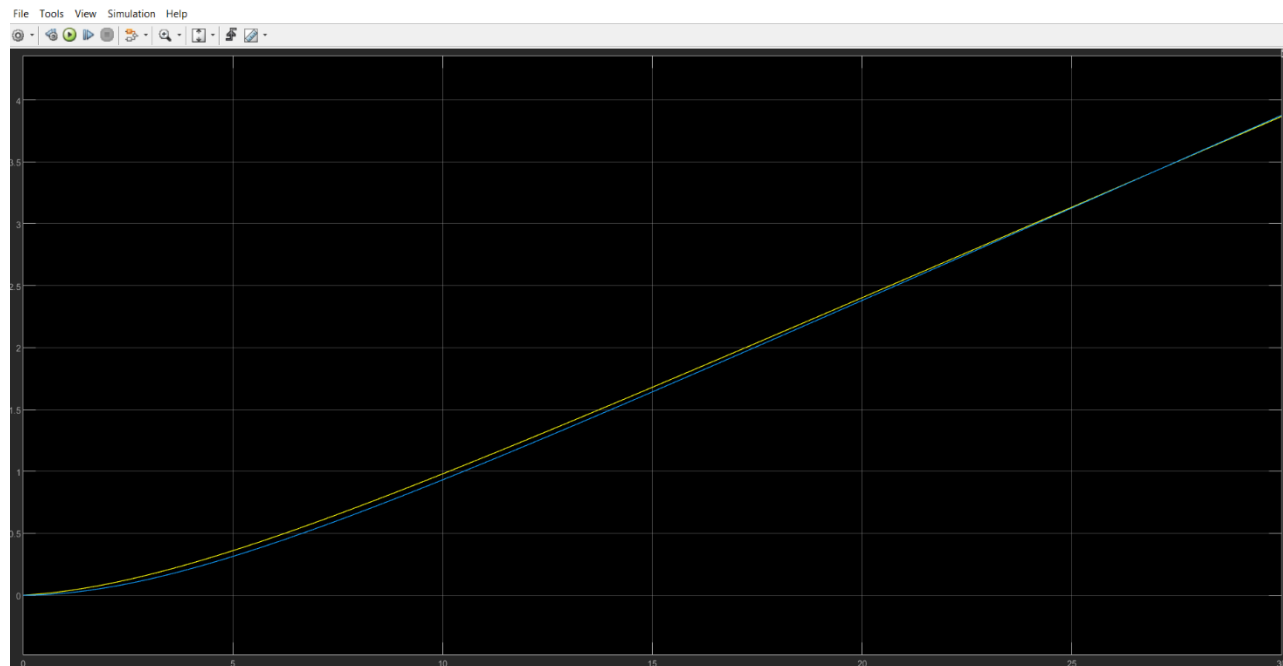
$$\exp(-0.02*s) * \frac{0.3494}{s^2 + 0.2381 s + 4.578e-13}$$

که دلی بسیار پایین است و حذف می کنیم. ضریب s0 نیز بسیار کوچک و تقریباً صفر است. به علاوه برای نزدیک تر شدن مدل به سیستم مقدار خیلی کمی ضرایب را عوض می کنیم و به تابع تبدیل زیر می رسیم:



مقایسه خروجی سیستم اصلی و سیستم تقریبی ما

اسکوپ 1

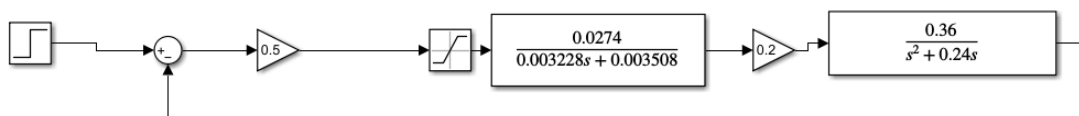


فایل های مورد استفاده: system_Identification ; model1

(2)

۲. با استفاده از جعبه ابزار *SISO* کنترلی از خانواده‌ی PID طراحی کنید بگونه‌ای که شرایط فوق حاصل گردد.

ابتدا مدل مدار بسته سیستم بدون کنترلر را رسم می کنیم:



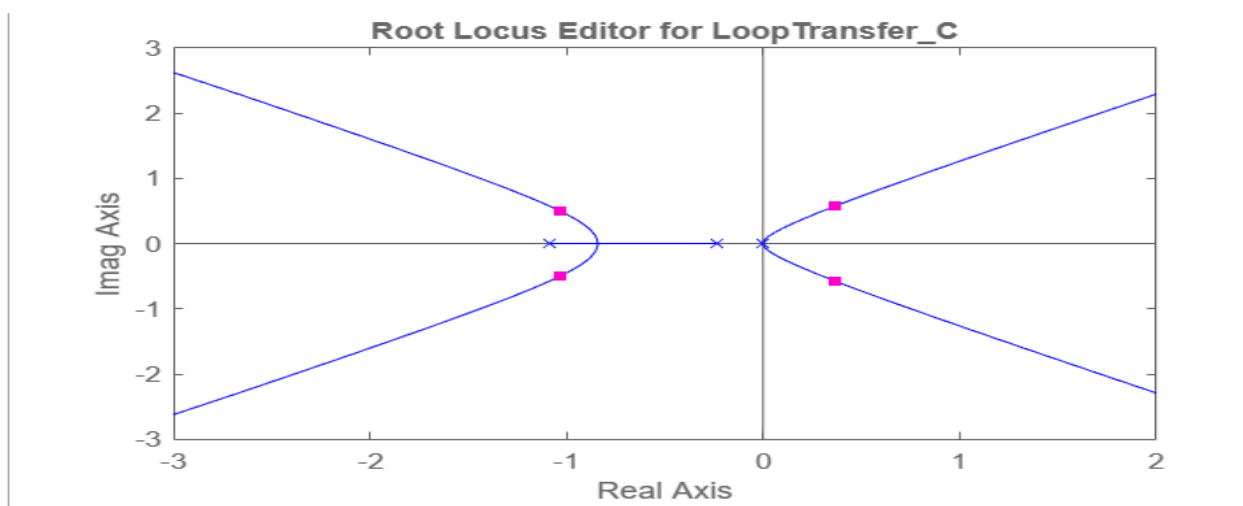
پلنت ما می شود:

G =

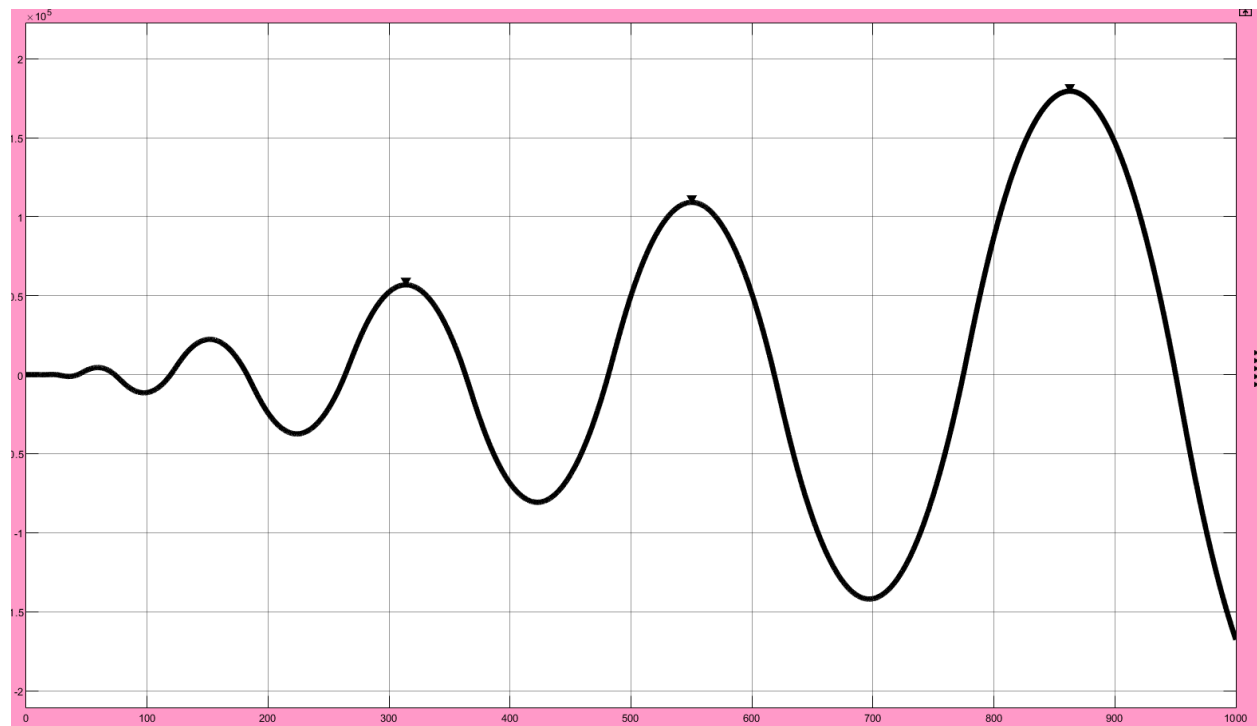
0.001973

0.003228 s^4 + 0.004283 s^3 + 0.0008419 s^2

که مکان هندسی قطب های مدار بسته به شکل زیر است:



پاسخ پله سیستم

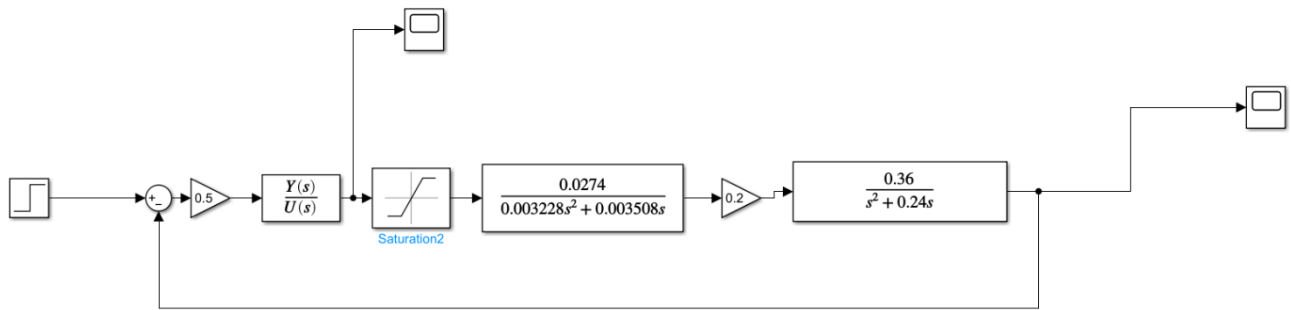


با توجه به مکان هندسی ، با PID نمی شود به پایداری رسید یا محدودیت در ts و اورشوت داریم. بهتر است ابتدا با کنترلری شبیه به کنترلر IMC مدل پایدار ، سیستم را پایدار کنیم.

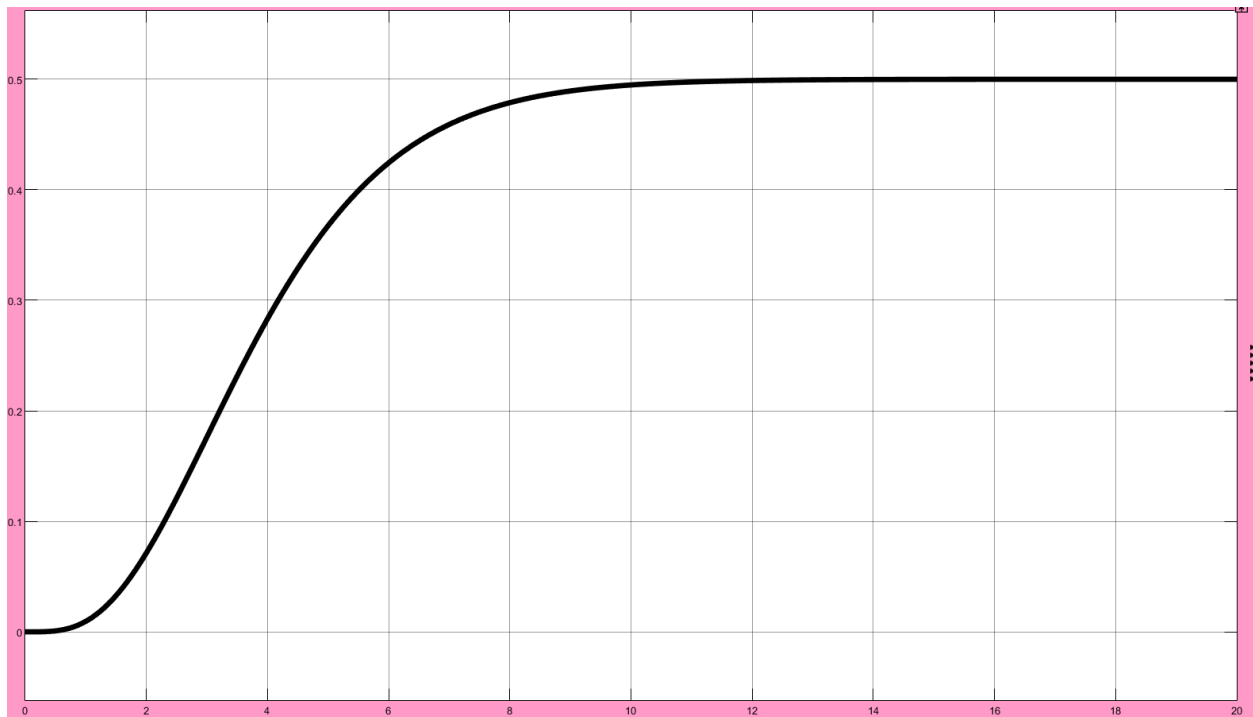
```
s=tf('s');
theta_v=0.0274/(0.003228 *s^2 + 0.003508 *s);
R_theta=0.36/(s^2+0.24*s);
gain=0.2;
G=theta_v*R_theta*gain*0.5;

Q=1/G/(s+1)^4;
K=Q/(1-Q*G);
close_loop=1/(s+1)^4;
```


سیستم با کنترلر K طراحی شده همراه saturation



پاسخ پله سیستم



حال همه plant همراه کنترلی که طراحی کردیم را یک سیستم مدار بسته در نظر می گیریم و تابع تبدیل را در کد حساب می کنیم (close_loop) و این تابع می شود سیستم مدار باز ما و برای آن در سیستم کنترلر PID همراه با فیلتر طراحی می کنیم.

 PID Tuning

Compensator

$C = 0.31726 \times \frac{(1 + 0.42s)(1 + 2.8s)}{s(1 + 0.025s)}$

▼ Select Loop to Tune

LoopTransfer_C ▼

Add new loop ...

Specifications

Tuning method: Robust response time ▼

Controller Type: ☐ P ☐ I ☐ PI ☐ PD ☒ PID

☒ Design with first order derivative filter

Design mode: Time ▼

Slower | Response Time (seconds) | Faster

Aggressive | Transient Behavior | Robust

5.603

0.6

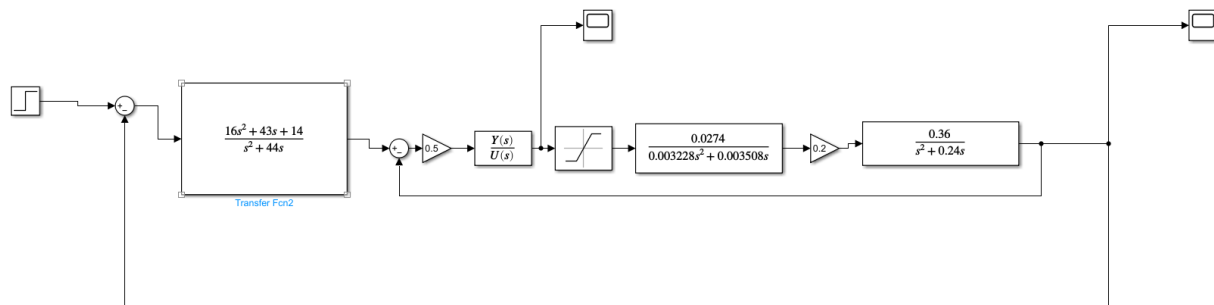
Reset Parameters

Update Compensator Help

کنترلر طراحی شده در سیمو:

```
Tunable Block
Name: C
Sample Time: 0
Value:
      15.322 (s+2.366) (s+0.357)
      -----
              s (s+40.8)
```

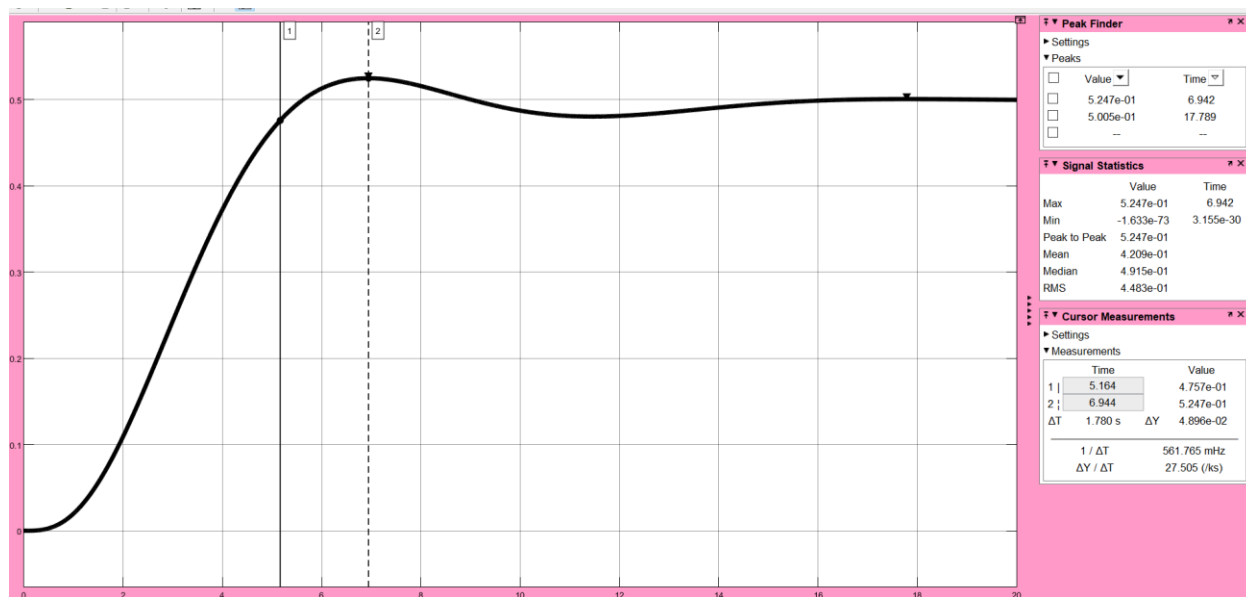
این کنترلر را در سیمولینک به مدار بسته اضافه می کنیم و کمی ضرایب را تیون می کنیم تا پاسخ بهتر شود. سیستم مدار بسته ما با سیستم مدل تقریبی به شکل زیر است:



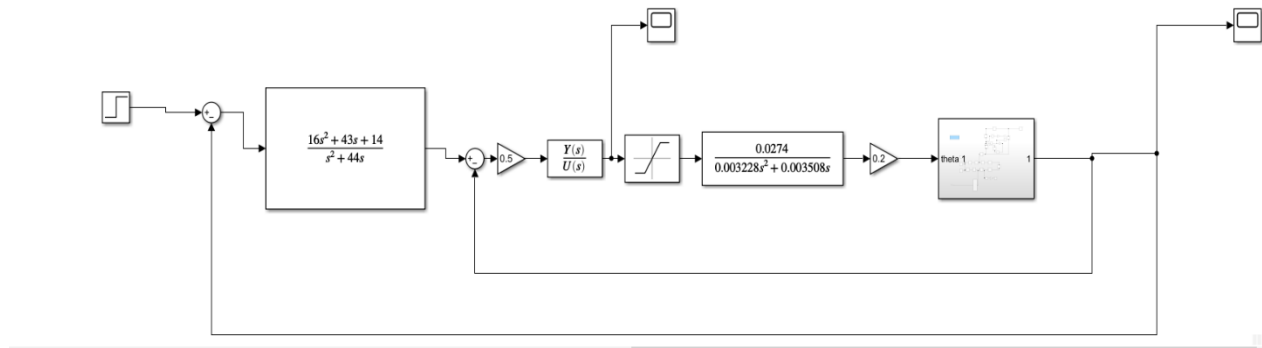
پسرخ پله:

$t_s = 5.14s$

$M_p = 5\%$



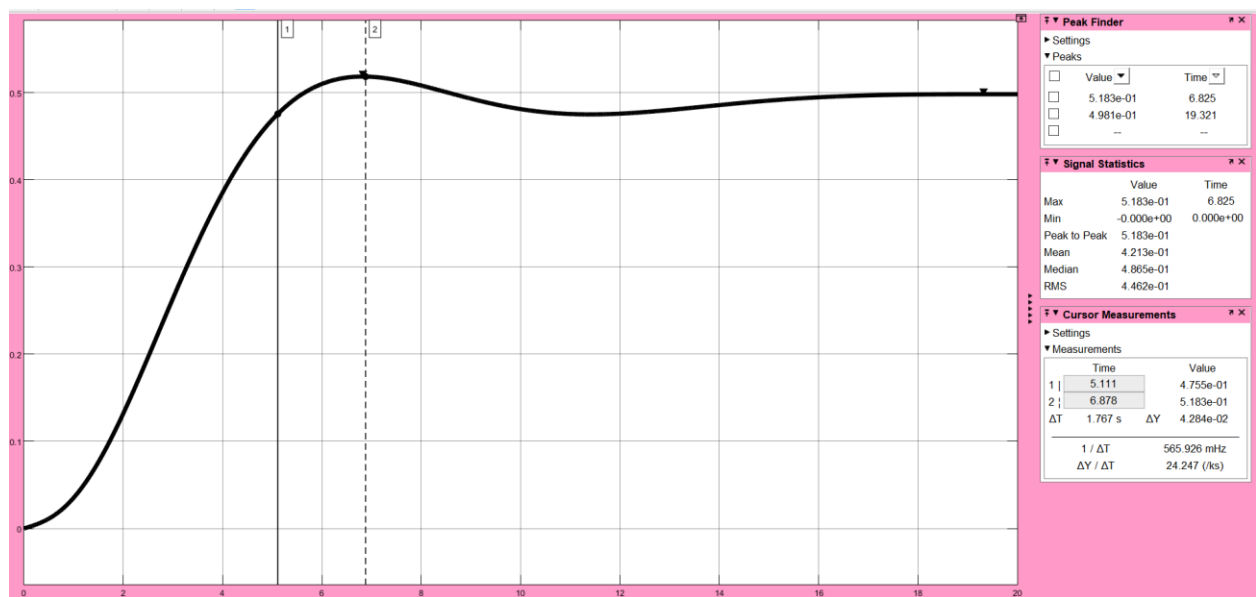
مدار بسته همراه با کنترلر های طراحی شده و مدل واقعی سیستم :



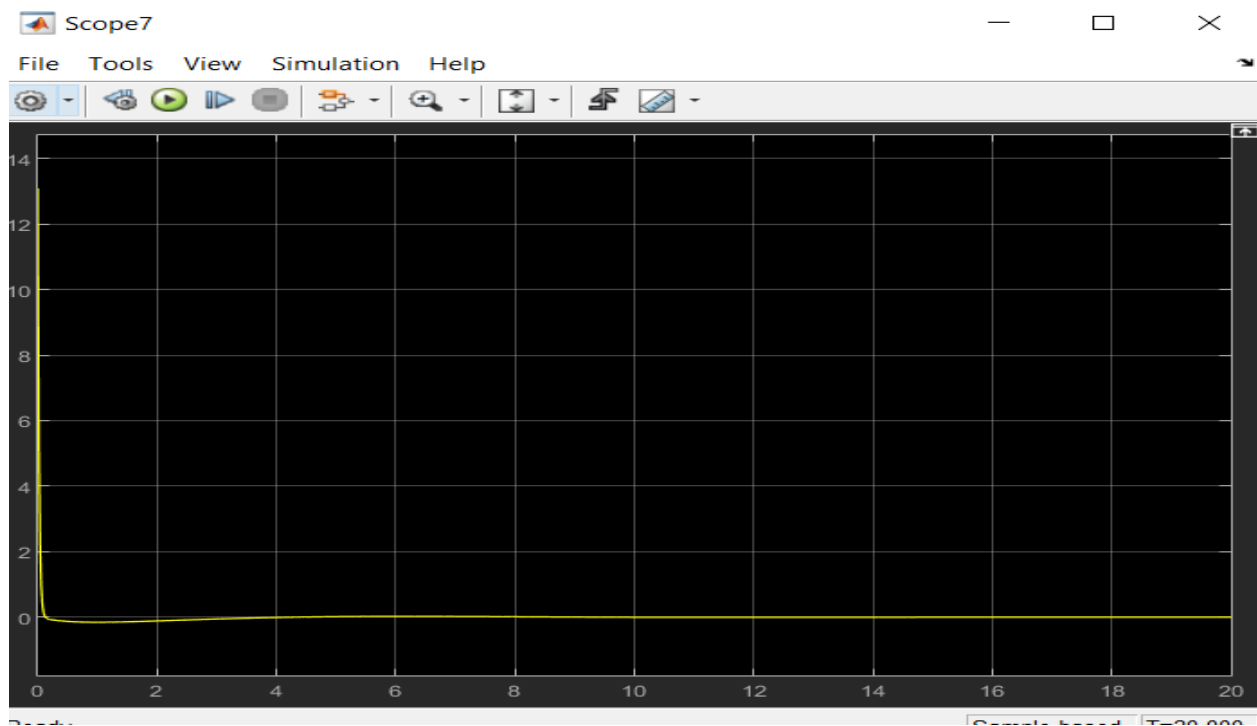
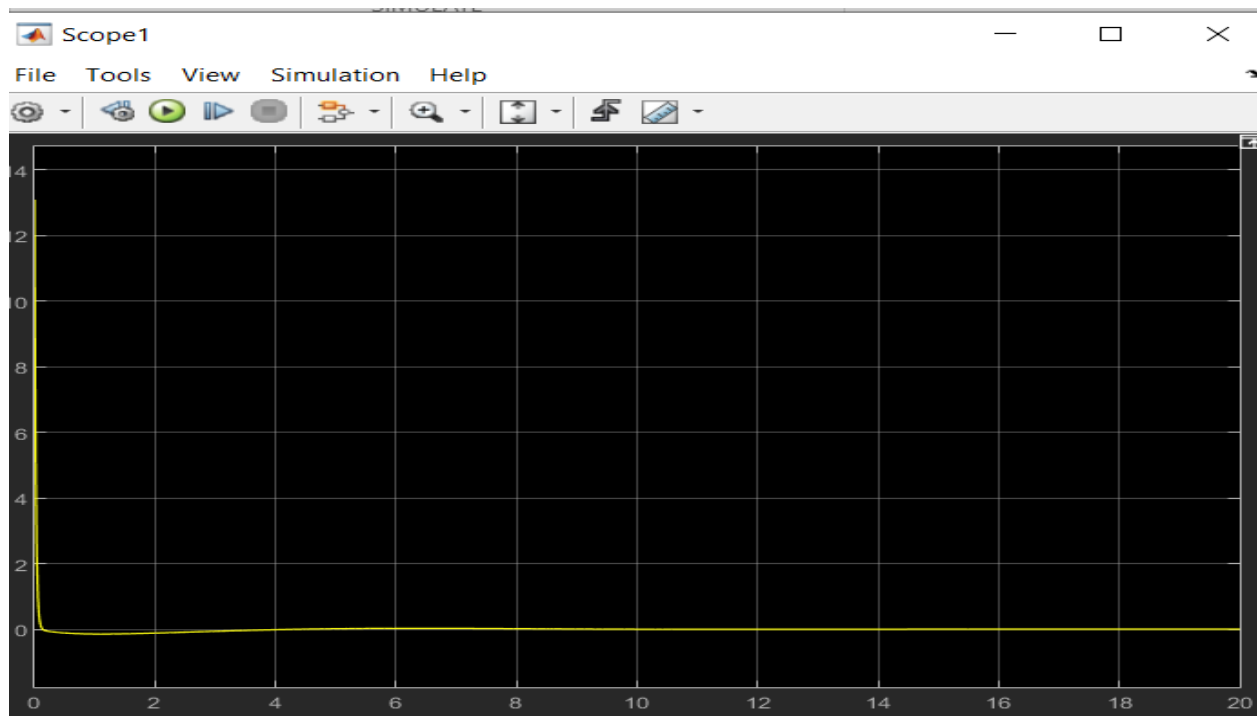
پاسخ پله سیستم:

$T_s=5.11s$

$M_p=3.6\%$



سیگنال کنترلی سیستم اول و دوم به شکل زیر است:



که همانطور که مشخص ست چون از 20 ولت کمتر است اشباع رخ نمی دهد و نیازی به آنتی وینداپ نداریم. به شرایط مطلوب مسئله یعنی ستلینگ تایم زیر 8 ثانیه و اورشوت کمتر از 20 درصد رسیدیم.

فایل های مورد استفاده:

Model2

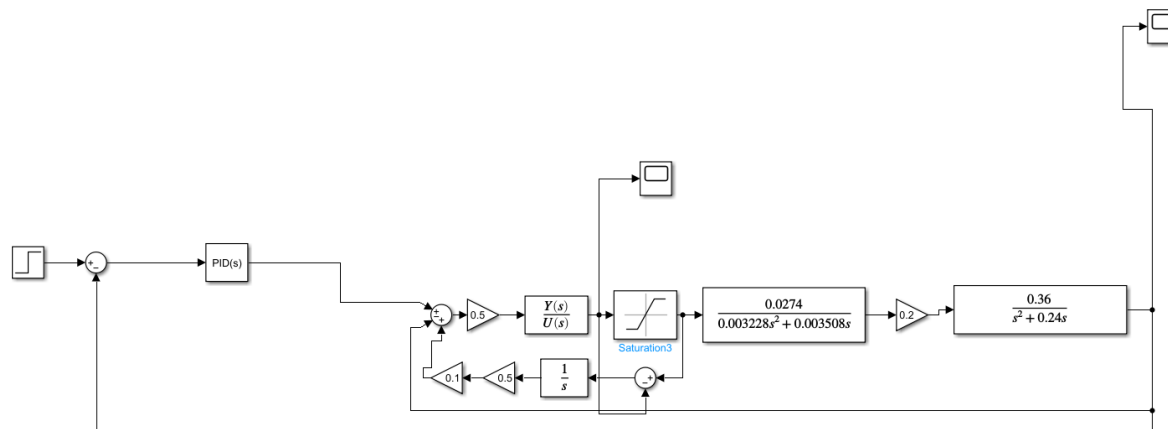
q2 q2.mat q1_a

(4

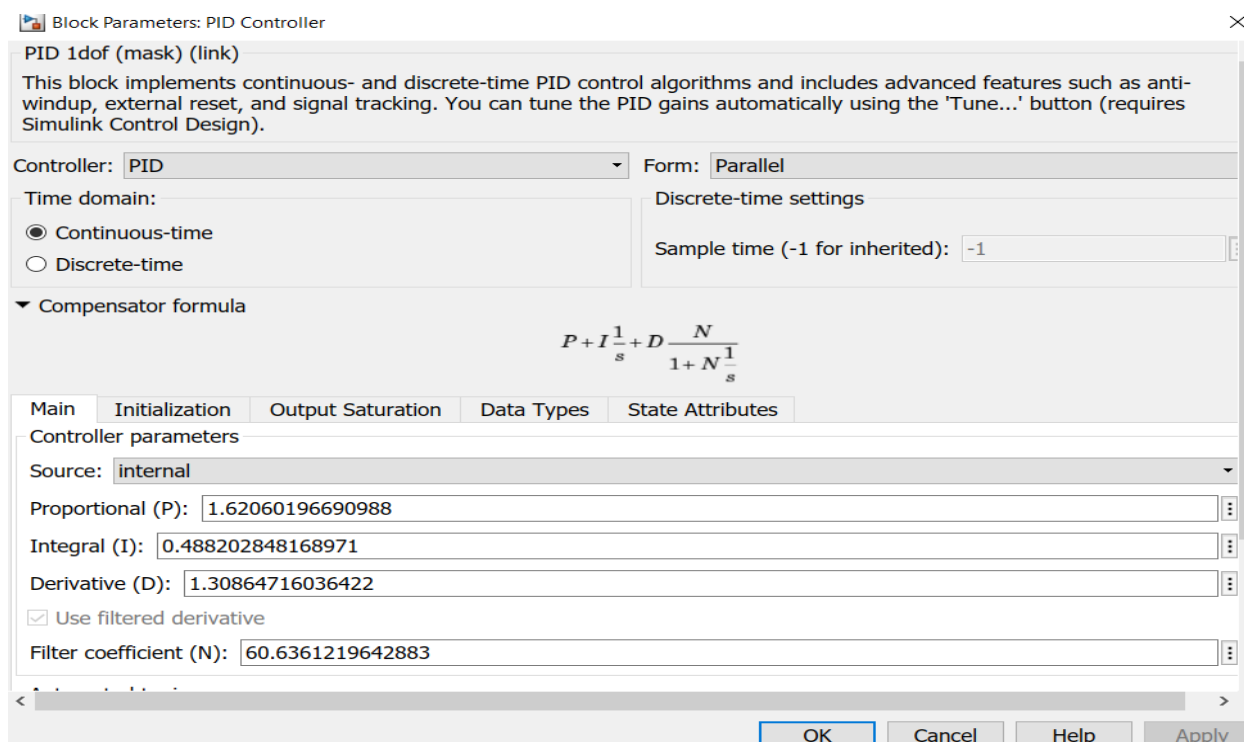
۴. به کمک ابزار PID-Tuner متلب، کنترلی از خانواده‌ی PID طراحی کنید بگونه‌ای که شرایط فوق حاصل گردد.

از پایدارساز طراحی شده در قسمت قبل استفاده می کنیم تا سیستم همچنان پایدار باشد و برای سیستم پایدار کنترلر طراحی کنیم.

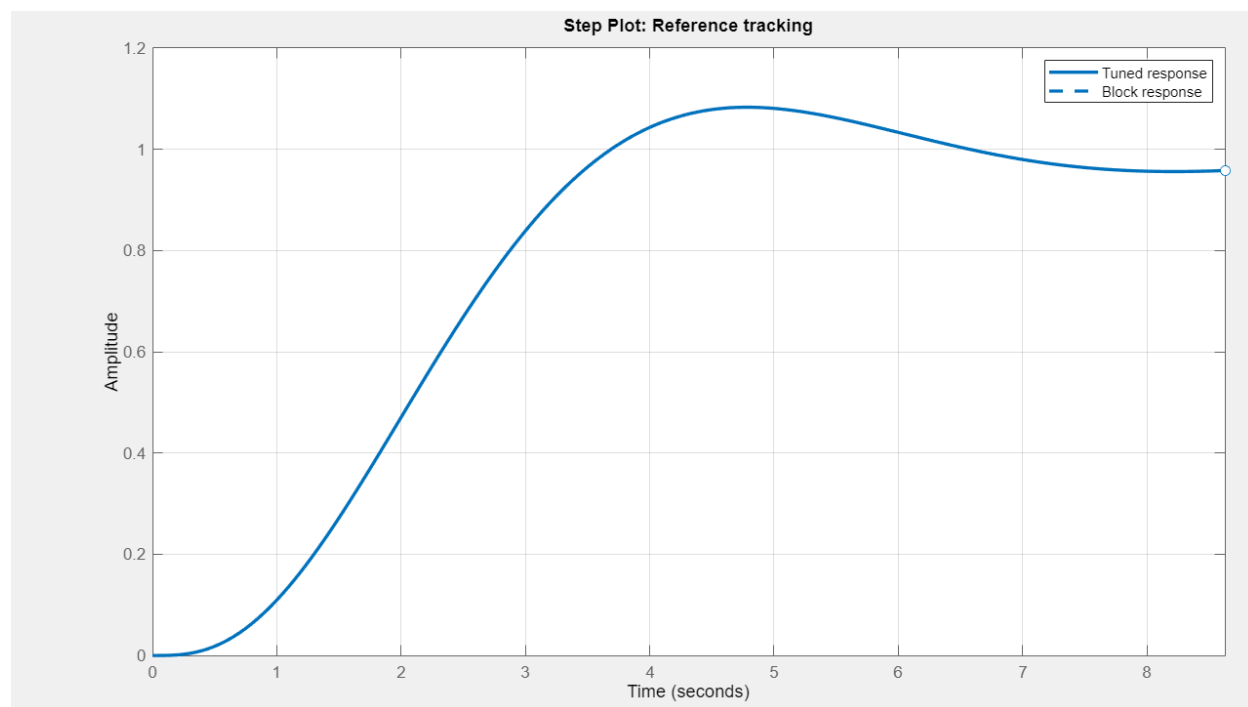
سیستم مدار بسته همراه انتی وینداپ PID Tuner به شکل زیر است:



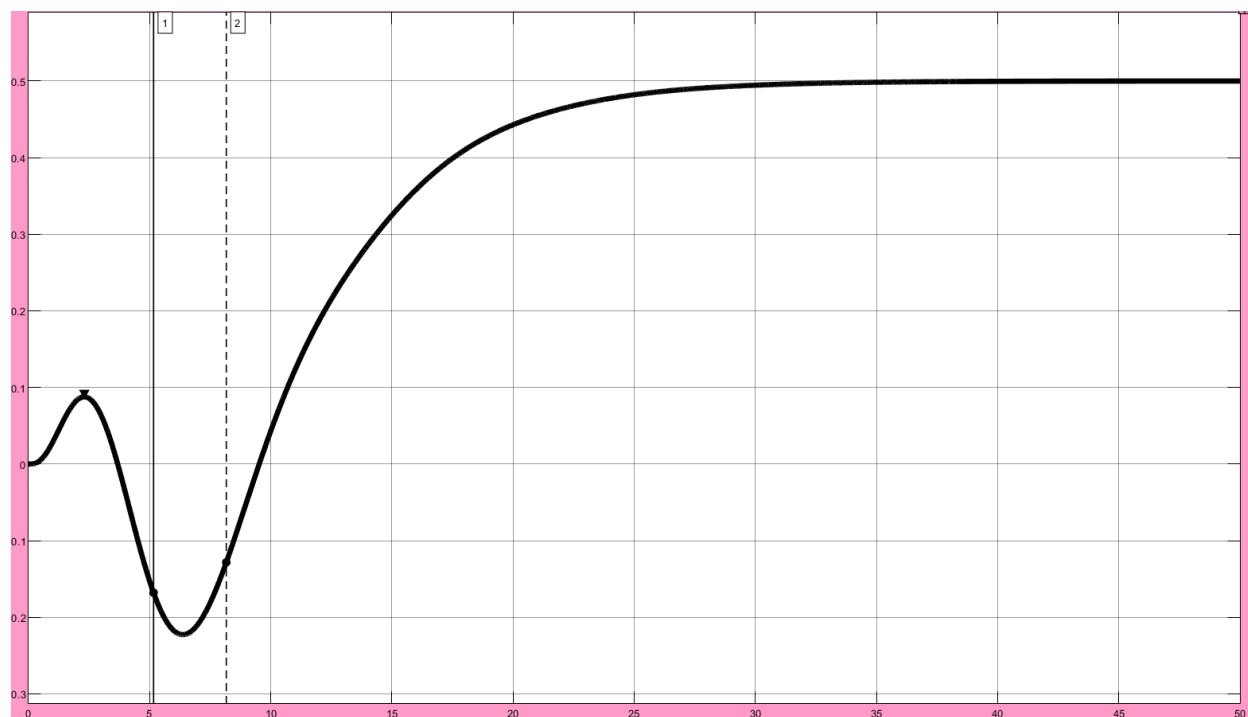
ضرایب بعد از تیون شدن توسط PID Tuner



شکل پاسخ بعد از تیون شدن :

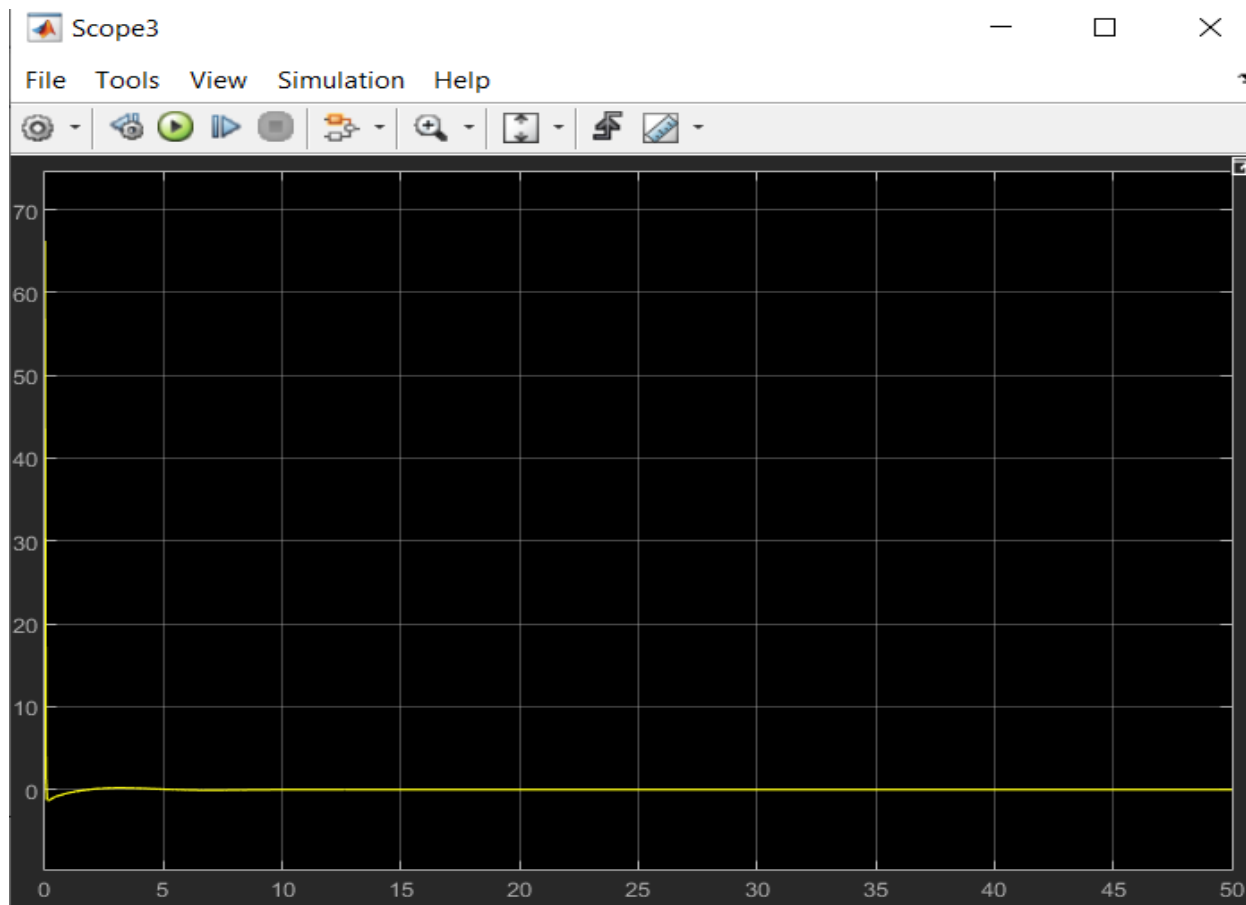


شکل واقعی پاسخ کل سیستم مدار بسته همراه این کنترلر تیون شده:

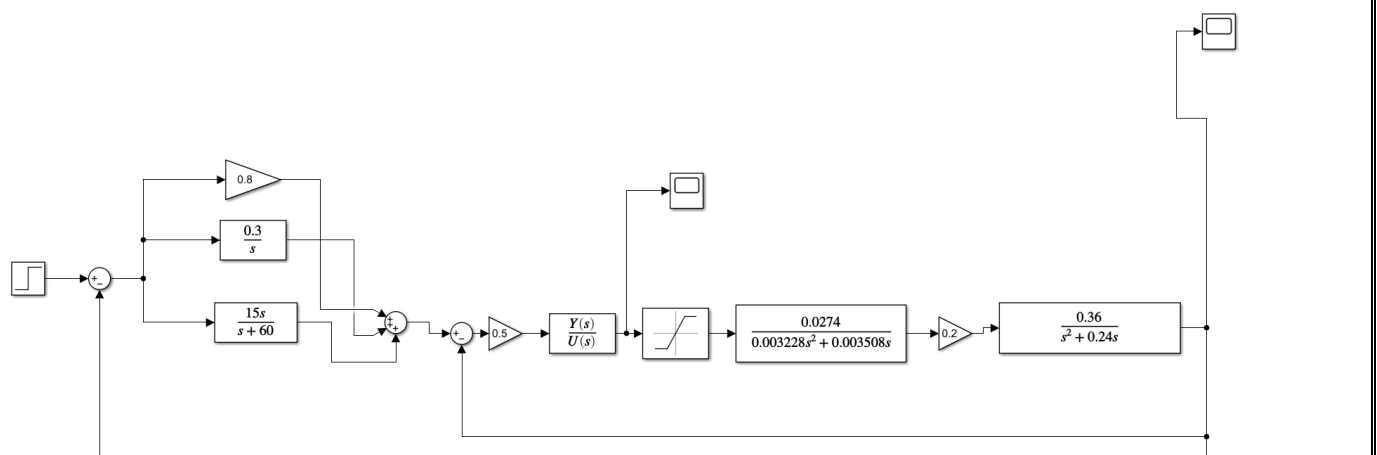


همانطور که مشخص است به دلیل وجود saturation پی ای دی طراحی شده برای سیستم ما پاسخ متفاوتی دارد و به خواسته های مطلوب نمی رسیم و باید ضرایب را کمی تغییر دهیم و fine tune کنیم.

سیگنال کنترلی:



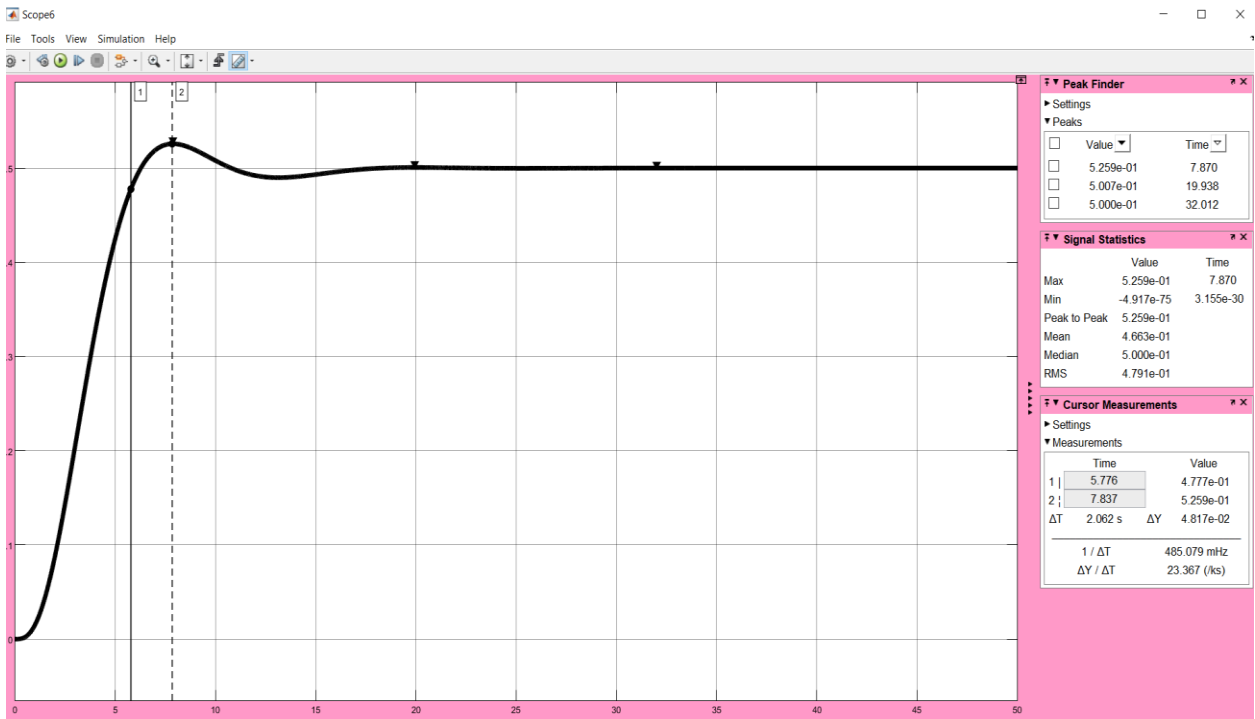
سیستم مدار بسته با مدل تقریبی و ضرایب تیون شده:



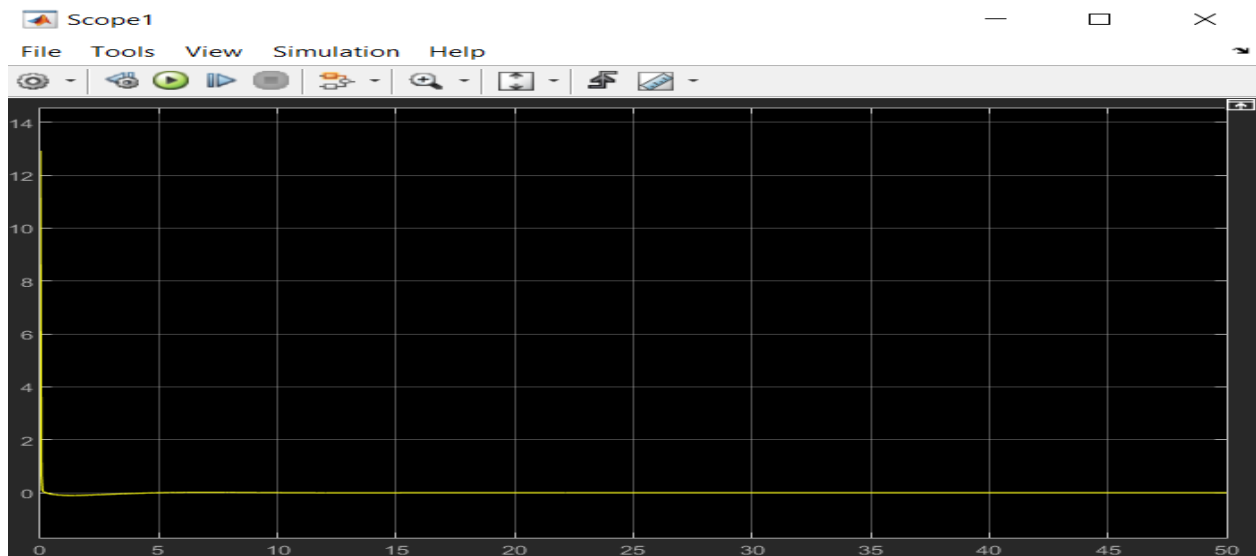
شکل پاسخ پله سیستم

$T_s=5.77s$

$M_p=5.8\%$

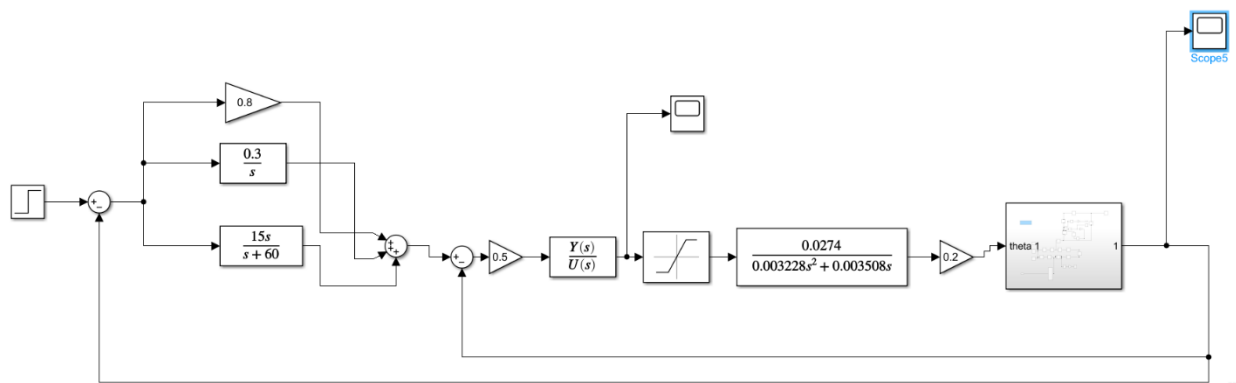


سیگنال کنترلی :



که چون به مقدار اشباع نمی رسد پدیده وینداپ رخ نمی دهد و نیازی به ضدکوک انتگرال نداریم.

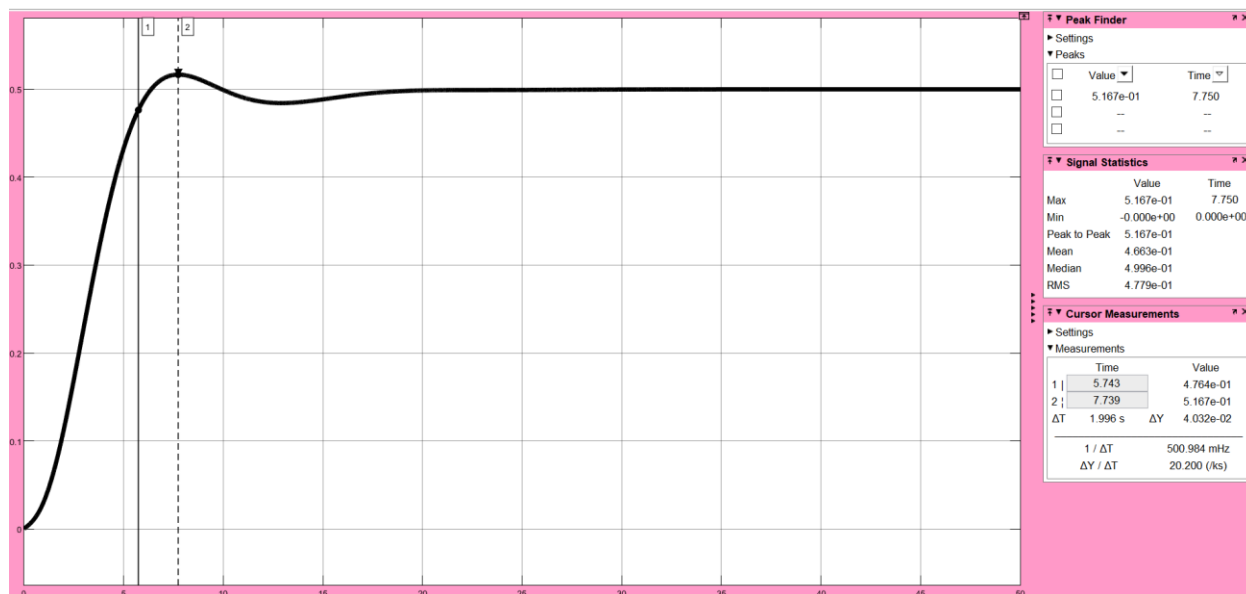
سیستم مدار بسته با مدل واقعی و ضرایب تیون شده:



شکل پاسخ پله سیستم واقعی

$$T_s=5.7s$$

$$M_p=3\%$$



سیگنال کنترلی:



بنابراین به شرایط مطلوب مسئله رسیدیم.

فایل این سوال q2 , model3

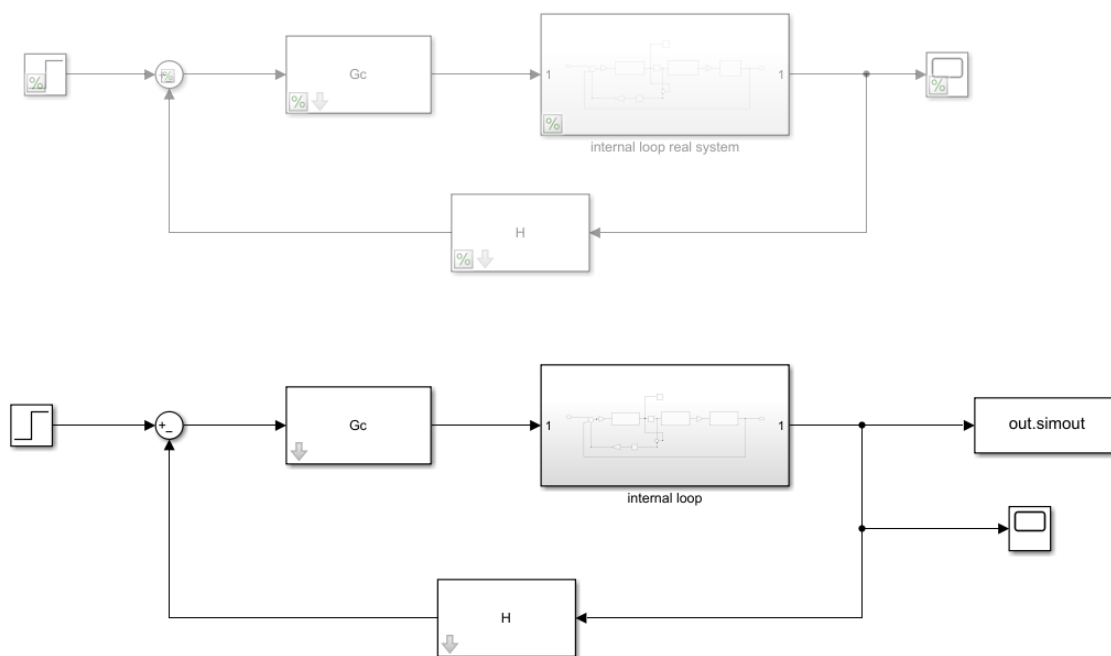
۵. با بکارگیری روش‌های تدریس شده (زیگلر نیکولز، آستروم هاگلند و ...) کنترلر PID مناسب را طراحی کنید.

از بین روش‌های گفته شده در درس و نوع کنترلر هایی که می‌توانند طراحی کنند باید انتخاب کنیم. اول از همه از آنجا که مدل مدار بسته حلقه داخلی به صورت زیر است :

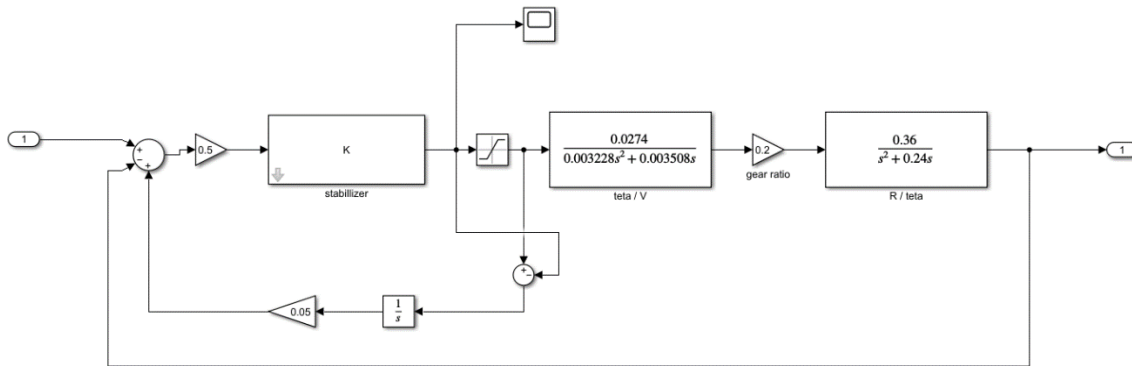
$$G(s) = \frac{1}{(s + 1)^4}$$

و انتگرالگیر ندارد پس برای تبعیت از فرمان و صفر شدن خطای ماندگار باید در کنترلر انتگرالگیر وجود داشته باشد پس طراحی کنترلر P را انجام نمی‌دهیم.

به این منظور مدل سیمولینک زیر را ایجاد می‌کنیم که مدار بسته حلقه داخلی را در یک subsystem قرار داده ایم. و توابع تبدیل G_c و H را در بلوک‌های LTI system قرار می‌دهیم و خروجی را به ورک اسپیس می‌فرستیم. یک مدل نیز با سیستم واقعی تشکیل می‌دهیم که در ابتدا کامنت می‌کنیم تا اجرا نشود. (مدل به اسم q5)



و سیستم داخل subsystem به صورت زیر است :



حال کد زیر را اجرا می کنیم :

```
% controller design project Q5

clear
clc
close all

run('code.m');

s = tf('s');
Gest = 1/((s+1)^4);

% FOPDT model estimation
[K0,L0,T0] = get_fod(Gest,0);
G_freq = exp(-L0*s)*K0/(T0*s+1);

[K1,L1,T1] = get_fod(Gest,1);
G_step = exp(-L1*s)*K1/(T1*s+1);

G_opt = opt_app(Gest,0,1,1);

figure(1)
step(Gest,G_freq,G_step,G_opt);
legend show

% G_step is chosen
[Kc,pp,wc,wp] = margin(Gest);
Tc = 2*pi/wc;
N = 10;

% ZN for PI PID PI-D for tf response
figure(2); hold on; title('ZN tf')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=ziegler_nic(2,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)
```

```

[Gc,H,Kp,Ti,Td]=ziegler_nic(3,[Kl,Ll,Tl,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=ziegler_nic(4,[Kl,Ll,Tl,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI','PID','PI-D')

% ZN for PI PID PI-D for freq response
figure(3); hold on; title('ZN freq')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=ziegler_nic(2,[Kc,Tc,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=ziegler_nic(3,[Kc,Tc,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=ziegler_nic(4,[Kc,Tc,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI','PID','PI-D')

% refined ZN
figure(4); hold on; title('refined ZN')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td,beta]=rziegler_nic([Kl,Ll,Tl,N,Kc,Tc]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

% CHR PI PID PI-D for set point and no overshoots
figure(5); hold on; title('CHR sp os=0')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=chr_pid(2,1,[Kl,Ll,Tl,N,0]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=chr_pid(3,1,[Kl,Ll,Tl,N,0]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

% [Gc,H,Kp,Ti,Td]=chr_pid(4,1,[Kl,Ll,Tl,N,0]);
% out = sim('q5.slx');

```



```

% plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI','PID')

% CHR PI PID PI-D for set point and 20% overshoots
figure(6); hold on; title('CHR sp os=20%')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=chr_pid(2,1,[K1,L1,T1,N,1]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=chr_pid(3,1,[K1,L1,T1,N,1]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

% [Gc,H,Kp,Ti,Td]=chr_pid(4,1,[K1,L1,T1,N,1]);
% out = sim('q5.slx');
% plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI','PID')

% wjc
figure(7); hold on; title('wjc')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,Kp,Ti,Td]=wjcpid([K1,L1,T1,N]); H=1;
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

% CC PI PID PI-D
figure(8); hold on; title('CC')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=cohen_pid(2,1,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=cohen_pid(3,1,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=cohen_pid(4,1,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI','PID','PI-D')

% CC revisited PI PID PI-D
figure(9); hold on; title('CC revisited')

```

```

plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=cohen_pid(2,2,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=cohen_pid(3,2,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=cohen_pid(4,2,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI','PID','PI-D')

% AH PI PID for both tf and freq responses
figure(10); hold on; title('AH')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=astrom_hagglund(1,1,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=astrom_hagglund(2,1,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=astrom_hagglund(1,2,[K1,Kc,Tc,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=astrom_hagglund(2,2,[K1,Kc,Tc,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI tf','PID tf','PI freq','PID freq')

% Opt-pid PI PID PI-D for tf response and sp
figure(11); hold on; title('opt-pid sp tf')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,Kp,Ti,Td,H] = Optimum(2,1,[K1,L1,T1,N,1]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(3,1,[K1,L1,T1,N,1]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(4,1,[K1,L1,T1,N,1]);
out = sim('q5.slx');

```

```

plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(2,1,[K1,L1,T1,N,2]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(3,1,[K1,L1,T1,N,2]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(4,1,[K1,L1,T1,N,2]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(2,1,[K1,L1,T1,N,3]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(3,1,[K1,L1,T1,N,3]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(4,1,[K1,L1,T1,N,3]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','PI ISE','PID ISE','PI-D ISE','PI ISTE','PID ISTE',...
      'PI-D ISTE','PI IST2E','PID IST2E','PI-D IST2E')

% Opt-pid PI PID PI-D for freq response and sp
figure(12); hold on; title('opt-pid sp freq')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,Kp,Ti,Td,H] = Optimum(2,1,[K1,L1,T1,N,Kc,Tc,K1*Kc]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(3,1,[K1,L1,T1,N,Kc,Tc,K1*Kc]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(4,1,[K1,L1,T1,N,Kc,Tc,K1*Kc]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend(' ',' ',' ',' ','PI','PID','PI-D')

% comparing results
figure(13); hold on; title('final compare')
plot(0:1:50,ones(51,1)*0.5,'k-.')
plot(0:1:50,ones(51,1)*0.525,'k-.')
plot(0:1:50,ones(51,1)*0.475,'k-.')
plot(0:1:50,ones(51,1)*0.6,'k-.')

[Gc,H,Kp,Ti,Td]=chr_pid(3,1,[K1,L1,T1,N,1]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

```

```

[Gc,Kp,Ti,Td]=wjcpid([K1,L1,T1,N]); H=1;
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,H,Kp,Ti,Td]=astrom_hagglund(2,1,[K1,L1,T1,N]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

[Gc,Kp,Ti,Td,H] = Optimum(3,1,[K1,L1,T1,N,2]);
out = sim('q5.slx');
plot(out.simout.time,out.simout.data)

legend('',' ',' ',' ','CHR PID sp os=20%','wjc','AH PID tf',...
      'Optpid PID ISTE sp tf ')

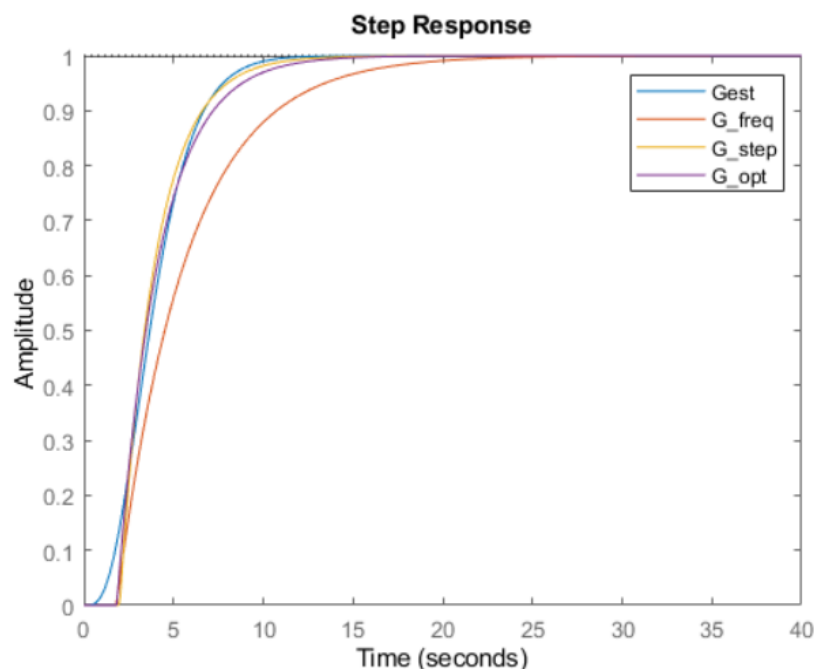
% best controller is wjc PID
[Gc,Kp,Ti,Td]=wjcpid([K1,L1,T1,N]); H=1;
out = sim('q5.slx');

```

ابتدا تابع تبدیل حلقه بسته داخل subsystem را تشکیل می دهیم که بدون در نظر گرفتن بلوک اشباع :

$$G(s) = \frac{1}{(s+1)^4}$$

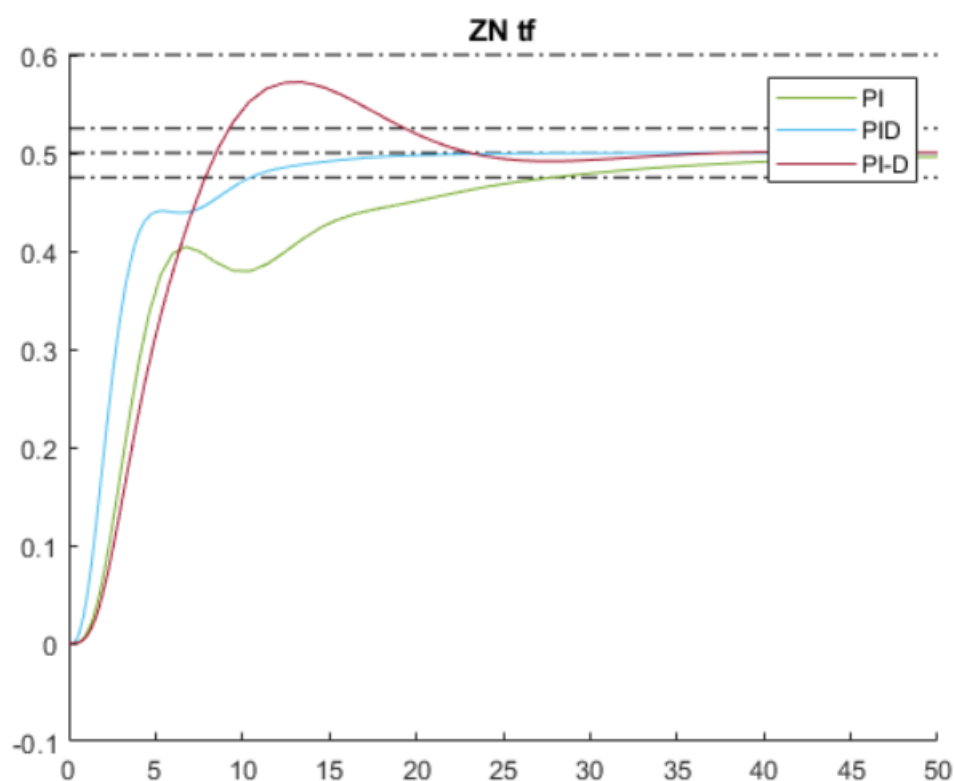
سپس به سه روش get_fod تابع تبدیل و فرکانسی و همچنین با opt_app تخمین مرتبه اول همراه با دلیلی را برای سیستم می زنیم و با مقایسه آنها می بینیم که روش تابع تبدیل و opt_app بسیار به هم نزدیک اند و به نظر می رسد که تابع تبدیل بهتر است. سپس با دستور margin مقادیر فرکانسی سیستم را نیز استخراج می کنیم و مقدمات طراحی PID فراهم می شود.

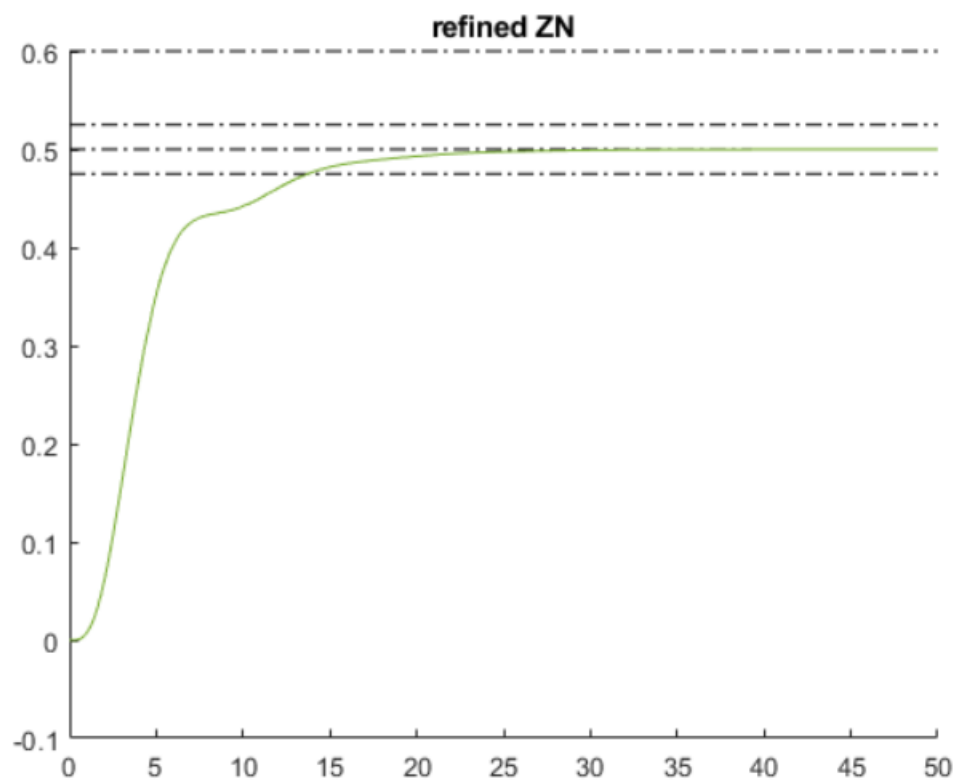
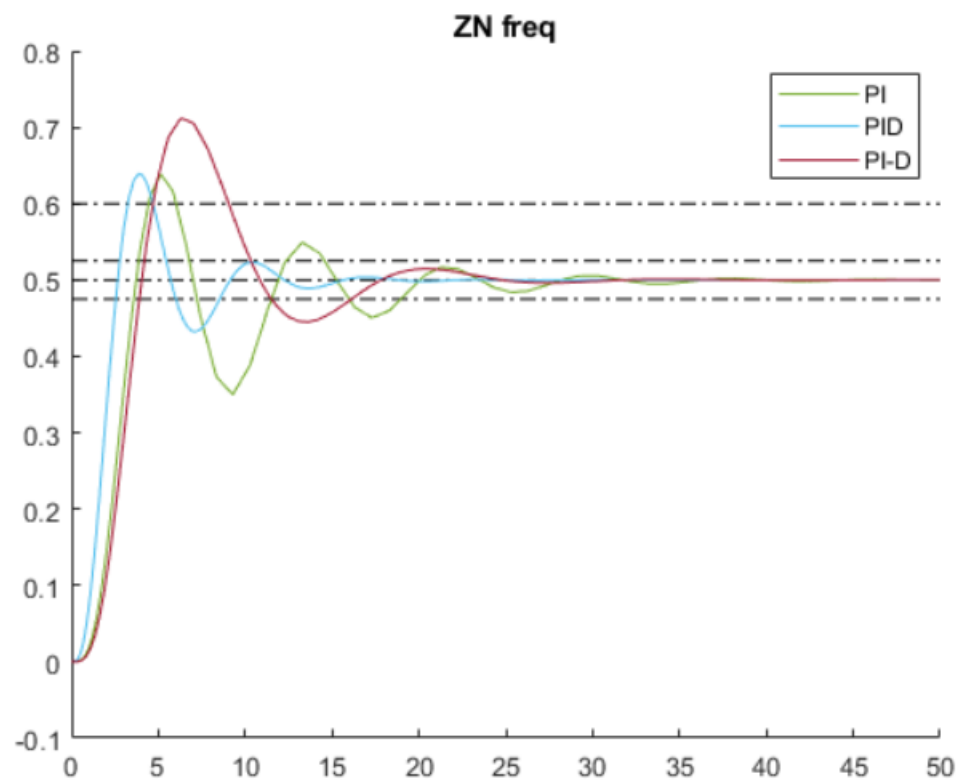


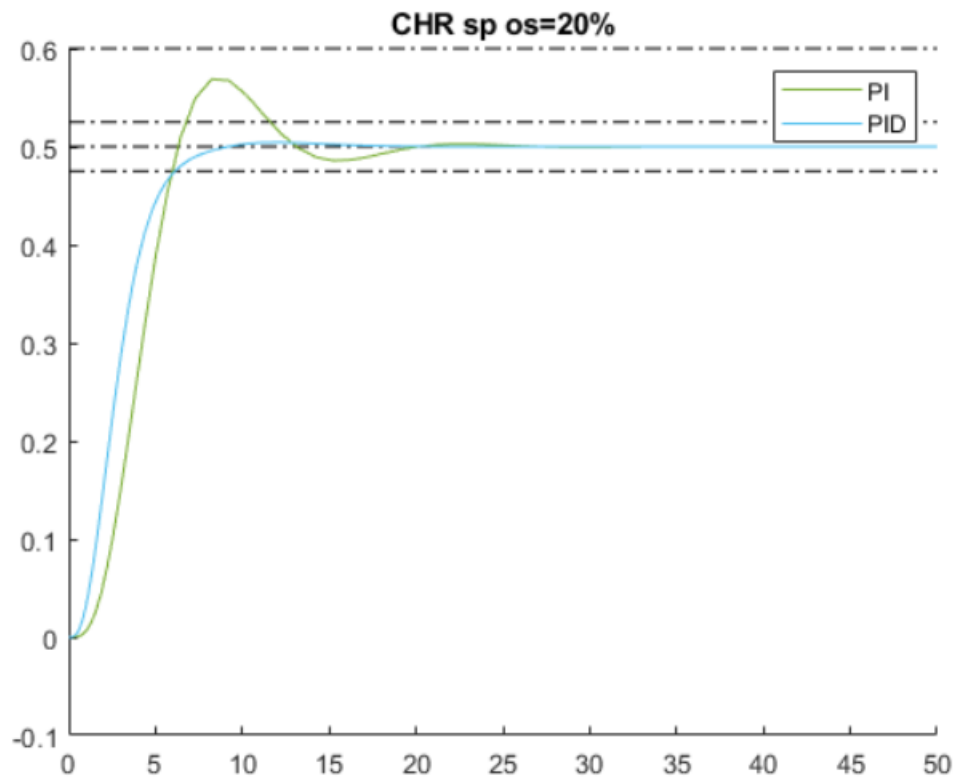
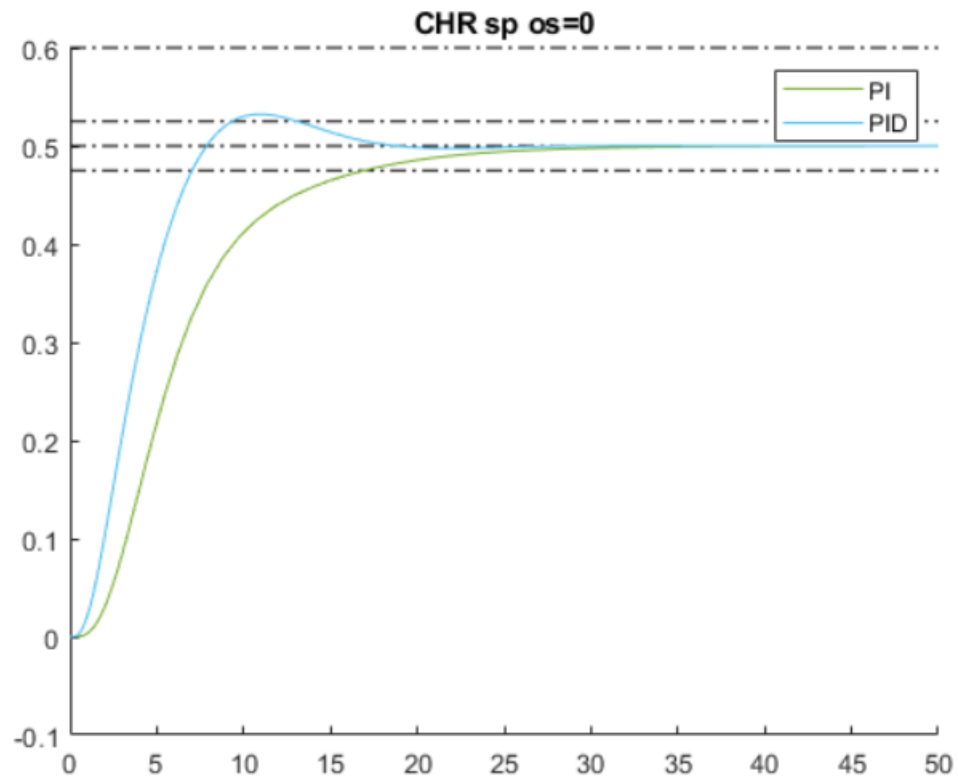
حال برای تمام روش های آموخته شده قطعه کد هایی را می زنیم که ابتدا خطوط ثابت 0.5 و 0.525 و 0.475 را رسم کند (مقدار نهایی و هدف به همراه حد 5% بالا و پایین به منظور تخمین زمان نشست) و خط ثابت 0.6 (که اورشوت 20% را بررسی کنیم). سپس با استفاده از توابع آماده کنترلر های PI و PID و PI-D را برای تمامی حالات جدولی هر یک رسم می کنیم و در پلات های جدا ذخیره می کنیم. برای خروجی گرفتن از مدل سیمولینک در طول اجرای کد از دستور sim استفاده می کنیم.

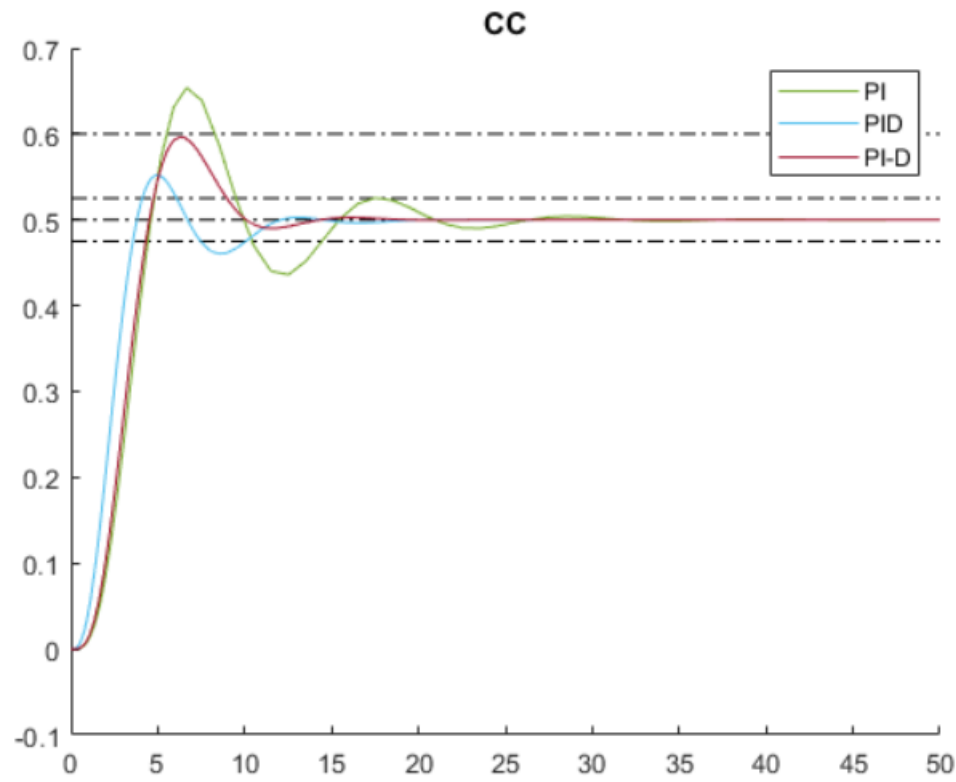
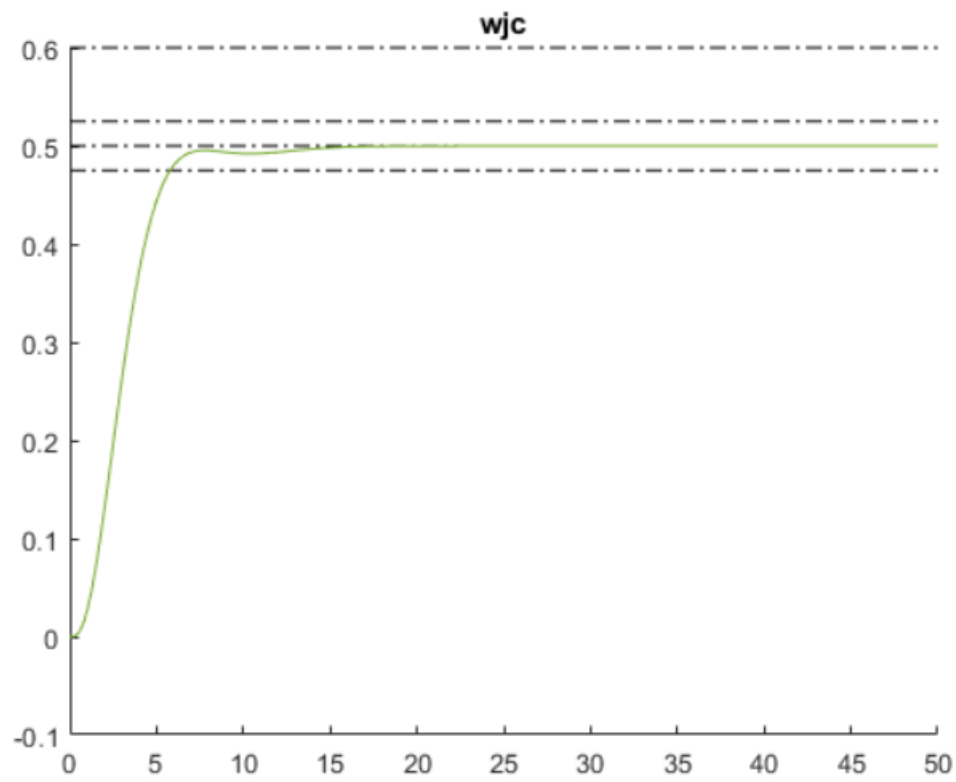
دو حالت CHR برای PI-D منجر به ارور قطب و صفر موهومی می شد که آنها را کامنت کردیم.

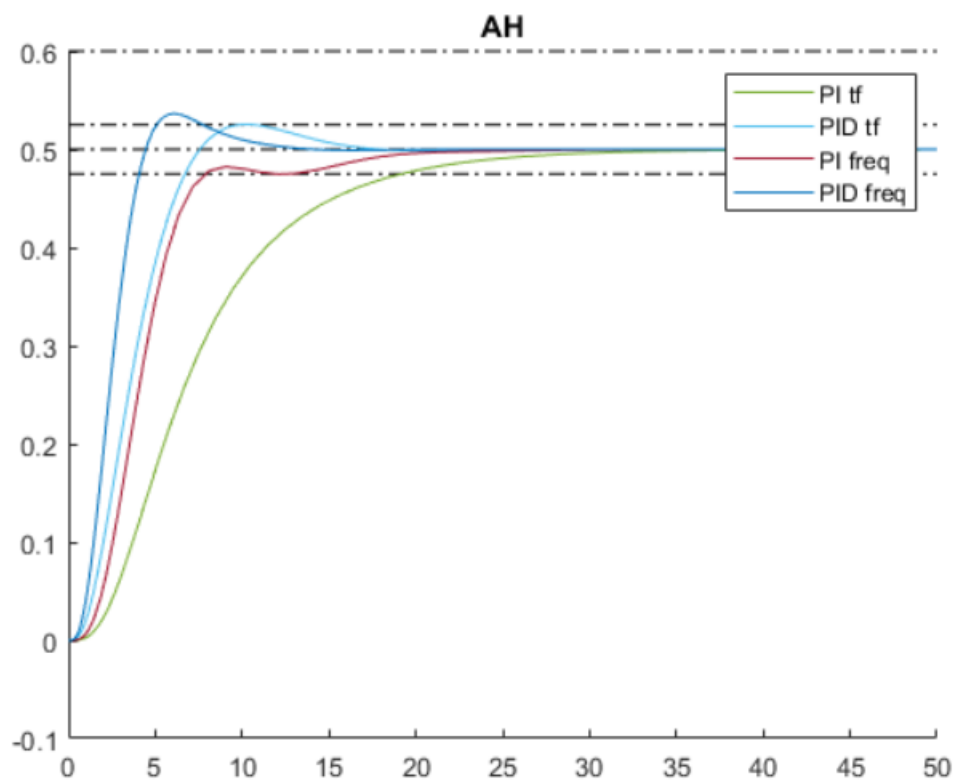
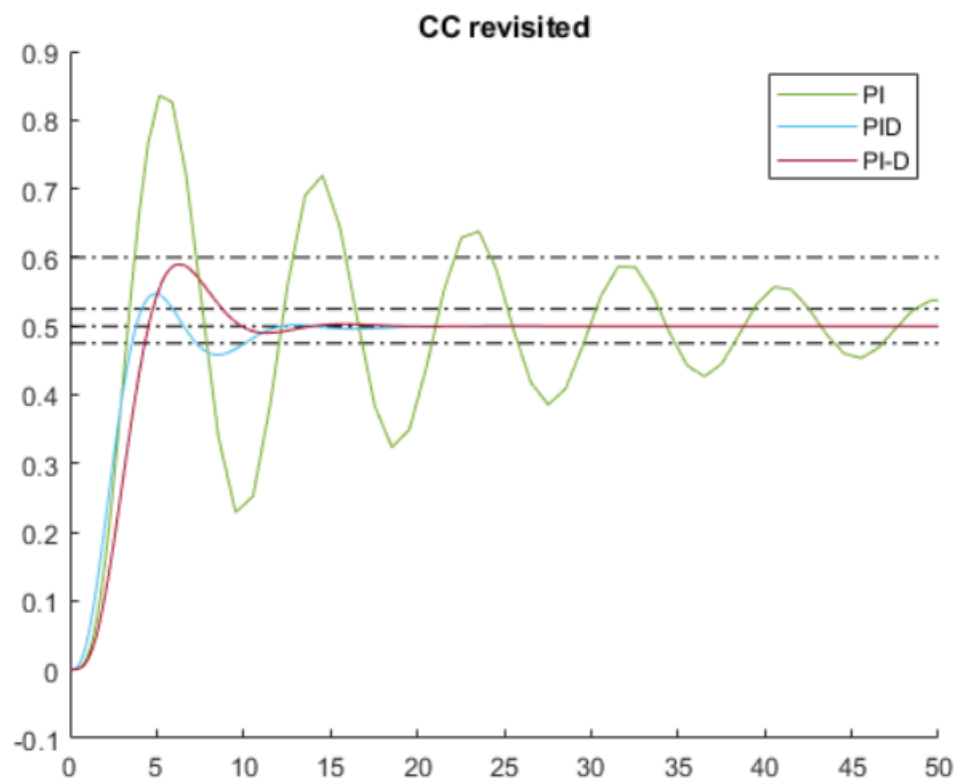
نتایج هر روش برای تمام حالات به صورت زیر است :

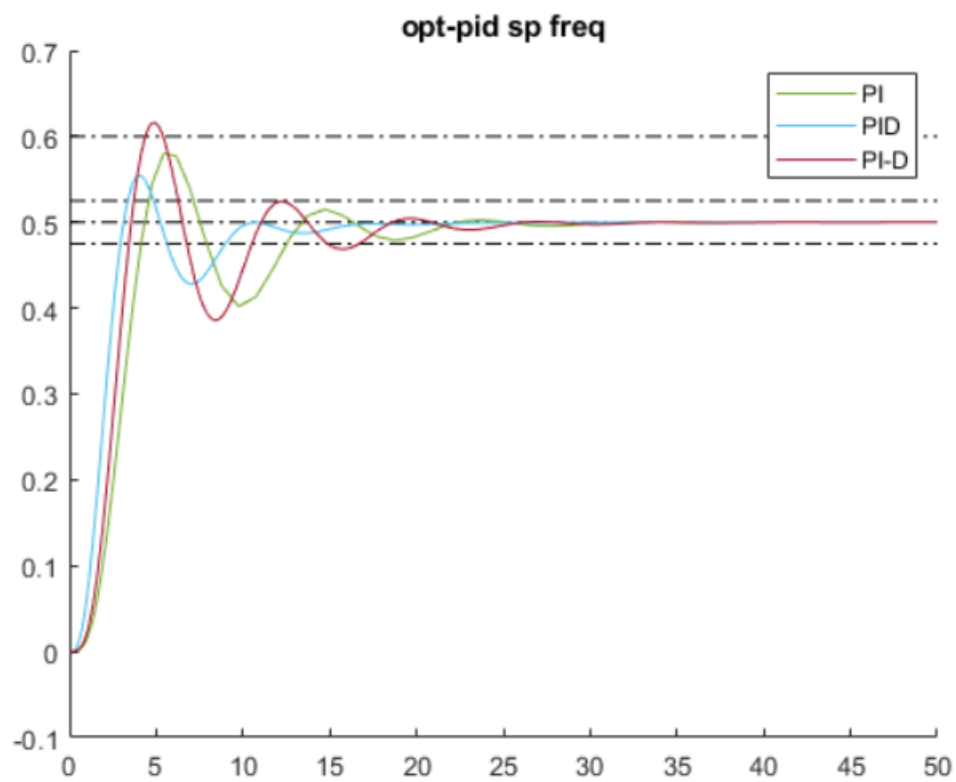
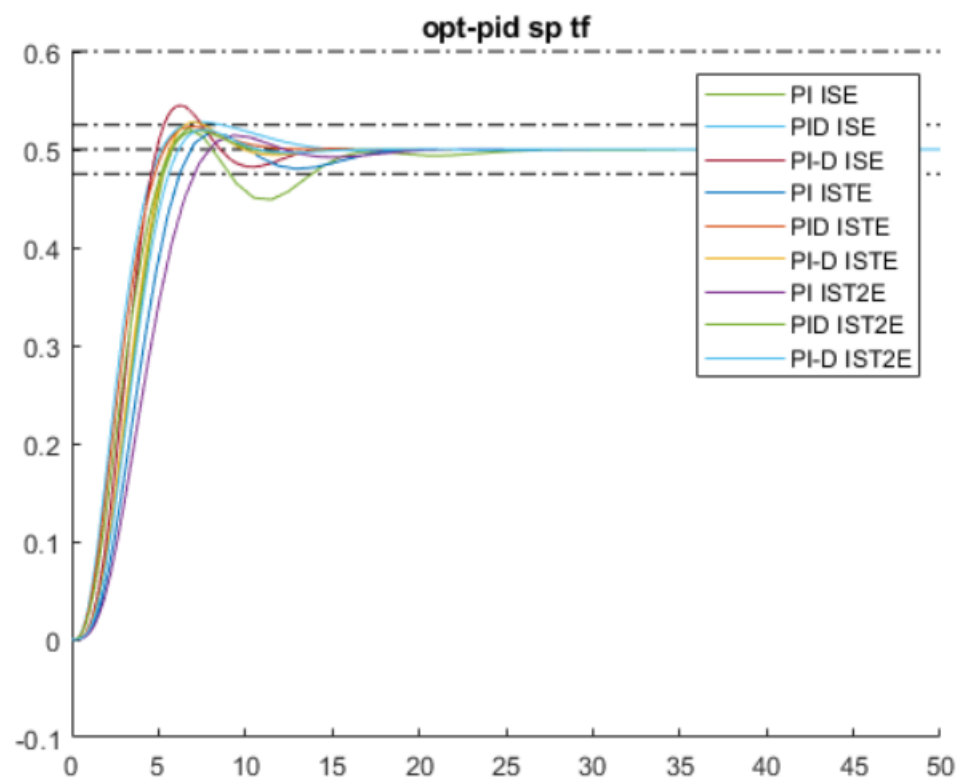




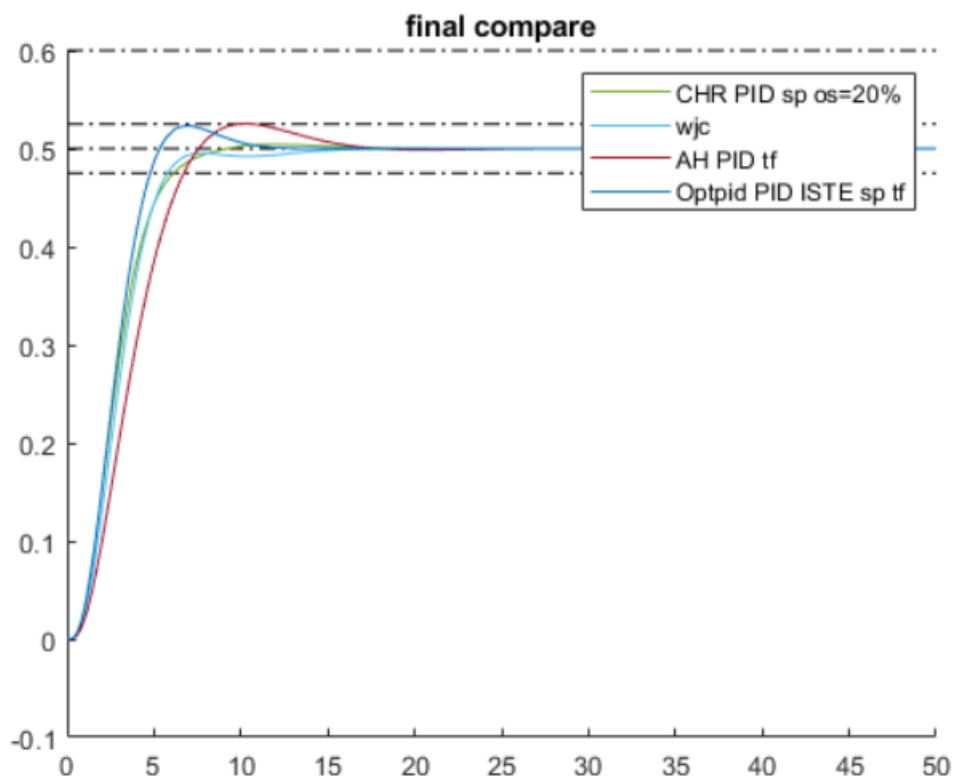








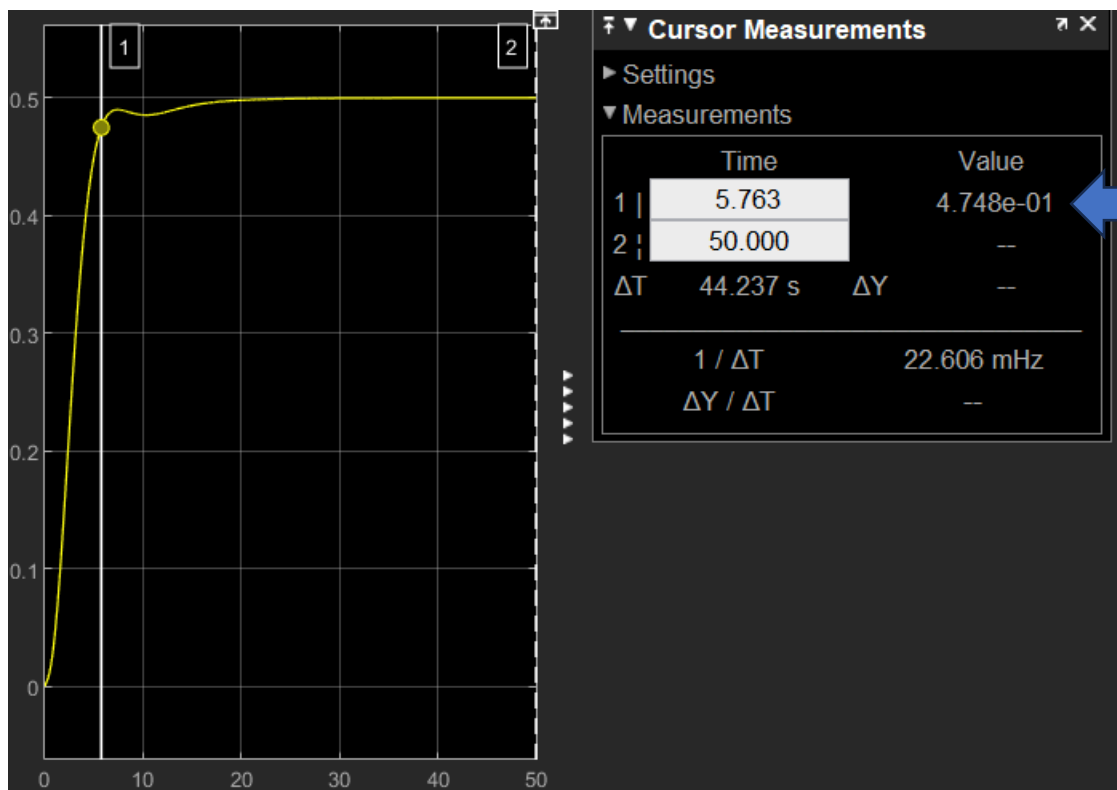
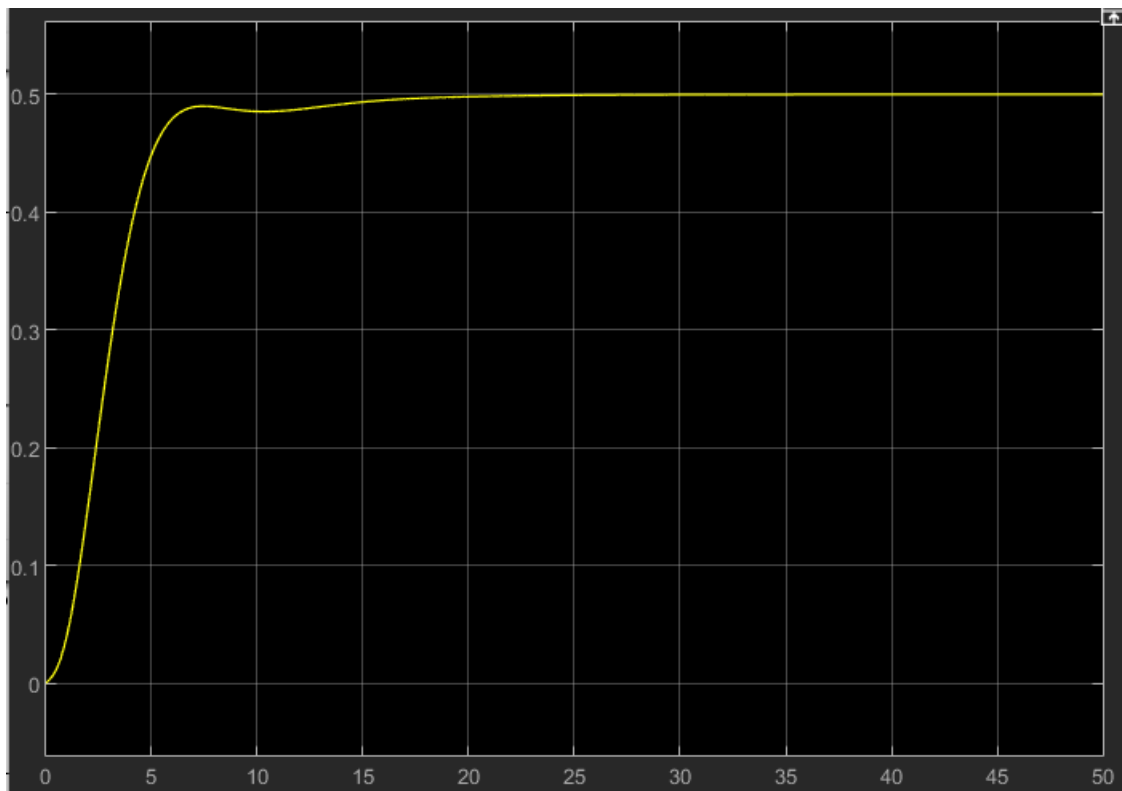
با بررسی تمامی کنترلر های فوق و انتخاب بهترین کنترلر از بین هر پلات که نتیجه را نیز ارضا کند به این نتیجه می رسیم که 4 کنترلر از بین آنها بهترین اند که این 4 کنترلر را با هم مقایسه می کنیم:



که مشاهده می شود با در نظر گرفتن مقاوم بودن کنترلر *wjc* در حین نداشتن اورشوت کمترین زمان نشست را دارد و در مجموع بهترین کنترلر از بین تمامی روش های آموخته شده است.

در نهایت آخرین خط کد مقادیر *Gc* و *H* را برای *wjc* تنظیم می کند و به این صورت در سیمولینک با *uncomment* کردن بخش مدل واقعی می توان نتیجه را روی مدل واقعی نیز تست کرد.

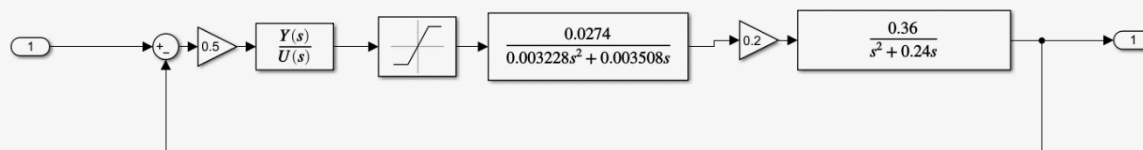
خروجی مدل واقعی پس از ورودی دادن مقدار 0.5 به صورت زیر است که مشاهده می شود مقادیر زمان نشست حدود 5.8 ثانیه و بدون اورشوت است که مطلوب است:



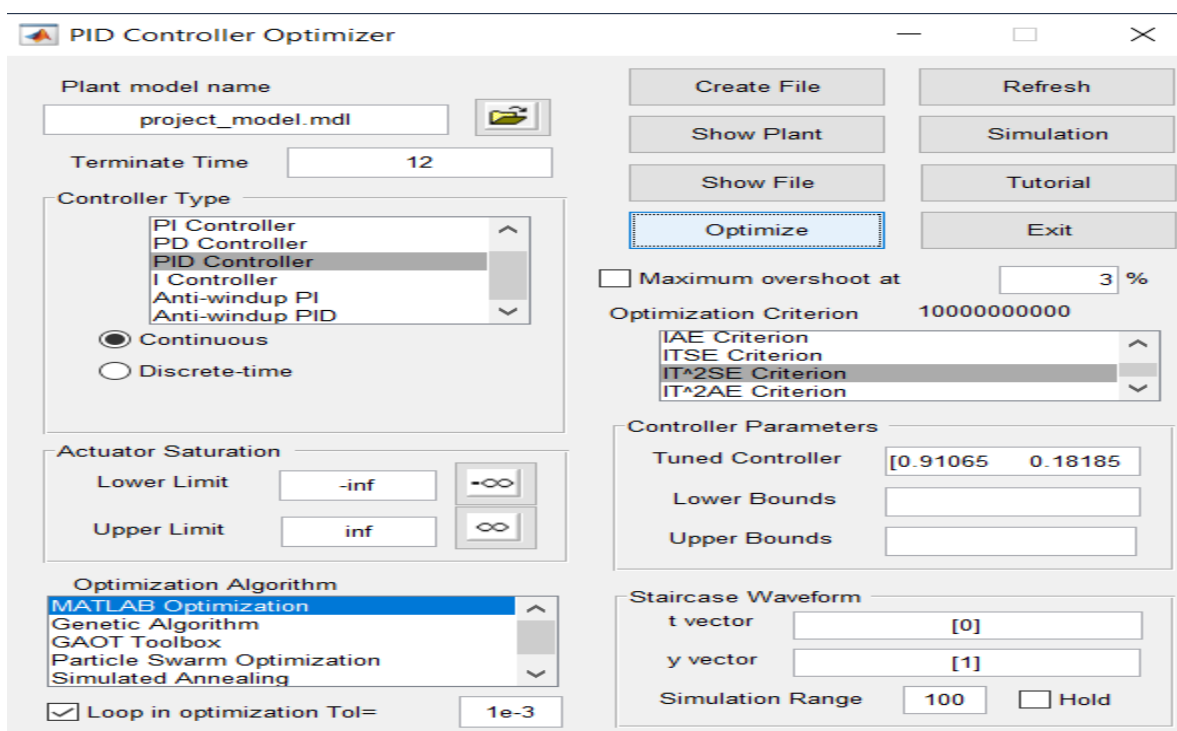
(6)

۶. با استفاده از ابزار optim pid کنترلی از خانواده‌ی PID طراحی کنید بگونه‌ای که شرایط فوق حاصل گردد.

در این سوال مدل مدار باز سیستم را ، پلنت اصلی همراه پایدار ساز در نظر می گیریم:

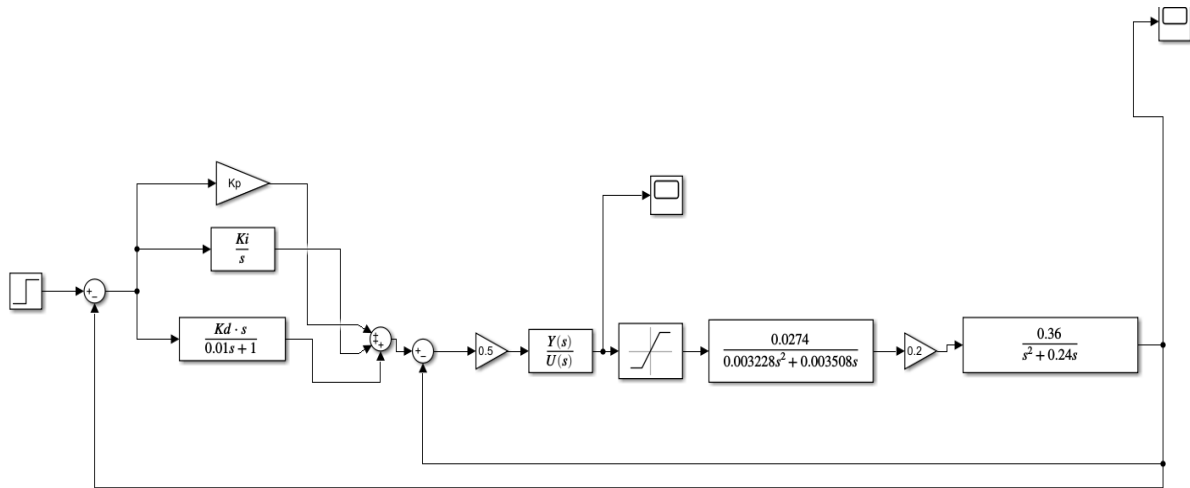


سپس با دستور optimpid ، ابزار مورد نظر ما باز می شود و این مدل را در آن لود می کنیم :

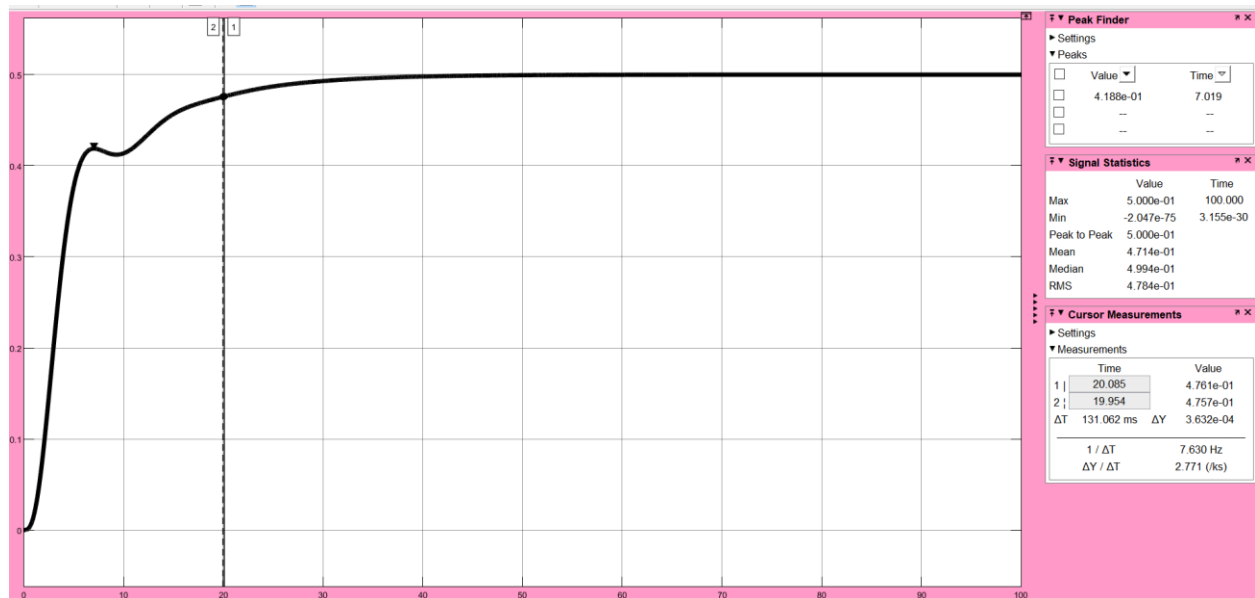


Optimization های مختلف و terminate تایم های متفاوت را امتحان می کنیم تا نهایت به بهترین ضرایب برسیم طوری که پاسخ پله ما شرایط مطلوب را داشته باشد. در نهایت معیار IT^2SE با $terminate\ time = 12$ پی ای دی با شرایط مطلوب تر را به ما می دهد. ضرایب را به شکل زیر در سیستم مدار بسته جای گذاری می کنیم:

Kd	0.2638
keyCriterion	5
Ki	0.1818
Kp	0.9106



پاسخ پله می شود:



که ستلینگ تایم حدود 20 ثانیه دارد و اورشوت نمی زند. برای اینکه به پاسخ پله مطلوب برسیم ضرایب را کمی تغییر می دهیم و fine tune می کنیم.

The block diagram illustrates a control system for a motor. The system consists of the following components and connections:

- Reference Input:** A step function input is fed into a summing junction.
- Summing Junction 1:** The reference input is subtracted from the feedback signal (from the output) to produce the error signal.
- Feedforward Path:** The error signal is fed into a block with a gain of 0.7 and a transfer function $\frac{0.3}{s}$.
- Feedback Path:** The error signal is also fed into a block with a transfer function $\frac{0.3s}{0.01s+1}$.
- Summing Junction 2:** The outputs of the feedforward and feedback paths are summed to produce the control signal.
- Controller:** The control signal is fed into a block with a transfer function $\frac{Y(s)}{U(s)}$.
- Plant:** The output of the controller is fed into a block with a transfer function $\frac{0.0274}{0.003228s^2 + 0.003508s}$.
- Output:** The output of the plant is fed into a summing junction that subtracts it from the reference input to produce the error signal.
- Summing Junction 3:** The output of the plant is also fed into a summing junction that subtracts it from the output of the controller to produce the feedback signal.

The screenshot displays a software interface for signal analysis. The main plot shows a signal that rises sharply from 0 to about 5.2 within the first 10 units of time, then levels off. Two vertical dashed lines are positioned at approximately 7.5 and 8.9 on the x-axis, labeled '1' and '2' respectively.

On the right, there are two panels:

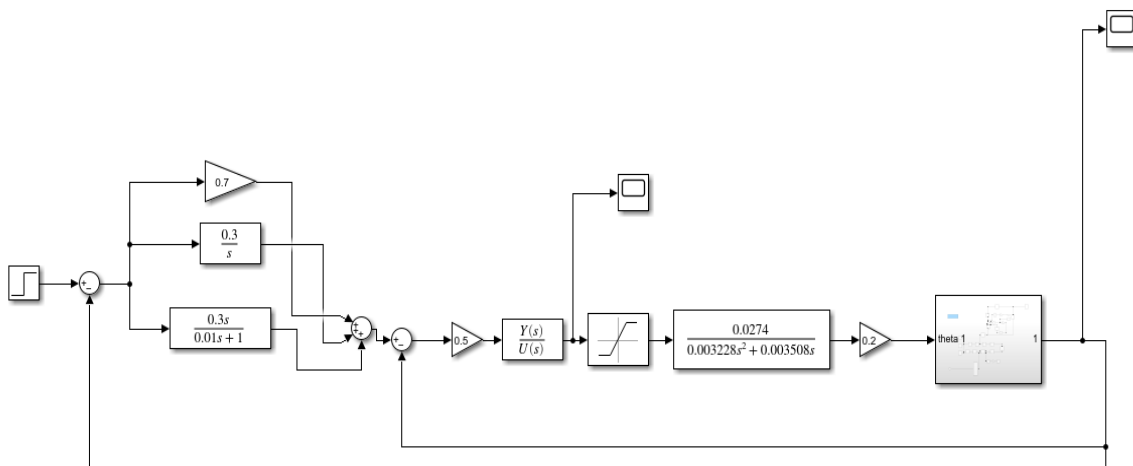
- Peak Finder:** This panel shows settings for peak detection and a table of detected peaks.

Value	Time
5.135e-01	8.963
--	--
--	--
- Cursor Measurements:** This panel shows settings for cursor measurements and a table of measured values at the two cursor positions.

Time	Value
6.586	4.778e-01
9.010	5.135e-01
ΔT	2.425 s
ΔY	3.571e-02
$1 / \Delta T$	412.432 mHz
$\Delta Y / \Delta T$	14.730 (ks)

Mp=2.7%

حال این کنترلر طراحی شده را روی سیستم اصلی تست می کنیم:



پاسخ پله سیستم اصلی همراه کنترلر طراحی شده:



$T_s=6.17s$

$M_p=0.46\%$

سیگنال کنترلی



سیگنال کنترلی در مدل تقریبی و مدل واقعی اورشوت زیادی نمی زند و پاسخ پله نیز به مقدار مطلوب رسیده بنابراین پدیده وینداپ رخ نداده و نیازی به ضد کوک انتگرال نیست.
با این کنترلر نیز به شرایط مطوب مسئله رسیدیم.

فایل های مورد استفاده:

Project_model

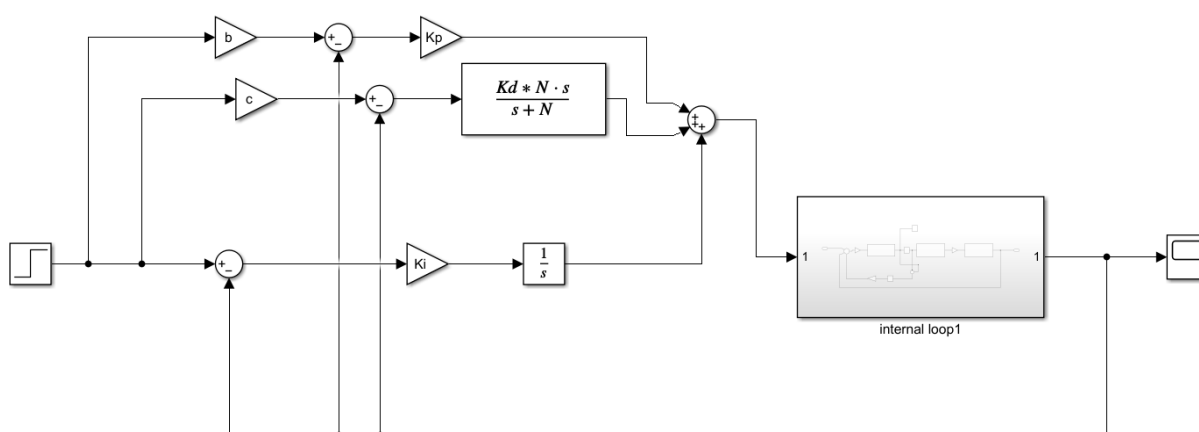
Model6 , q2

۷. یک کنترلر PID دو درجه آزادی برای رسیدن به خواسته‌های مسئله طراحی کنید و این کنترلر را با کنترلرهای طراحی شده در قسمت‌های قبلی مقایسه نمایید.

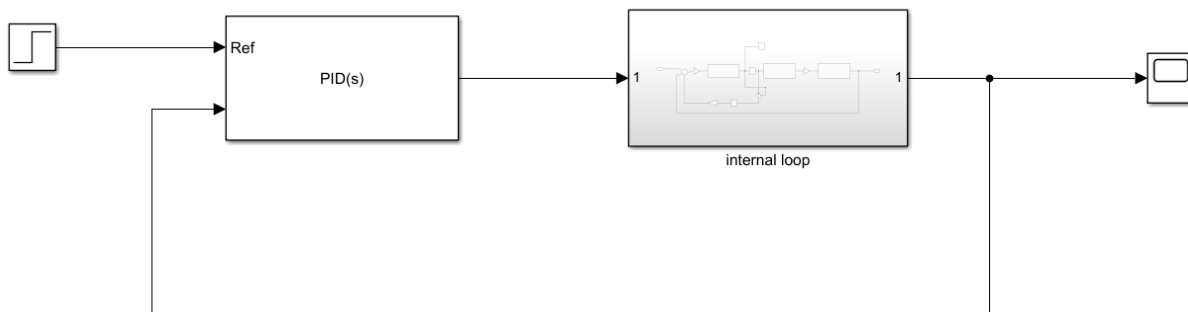
برای کنترلر دو درجه آزادی از بلوک PID controller (2DOF) استفاده می‌کنیم. این بلوک سیگنال هدف (مقدار مطلوب) را به نام ref می‌گیرد و سیگنال خروجی را نیز به صورت فیدبک به آن ورودی می‌دهیم. عملکرد آن به این صورت است که ضریبی از ورودی را برای تناسب و مشتق در نظر می‌گیرد و با تنظیم این ضرایب به عنوان دو پارامتر طراحی اضافه به کنترل سیستم می‌پردازد. معادله عملکرد این کنترلر به این صورت است :

$$P(b \cdot r - y) + I \frac{1}{s}(r - y) + D \frac{N}{1 + N \frac{1}{s}}(c \cdot r - y)$$

شماتیک آن به صورت زیر است :

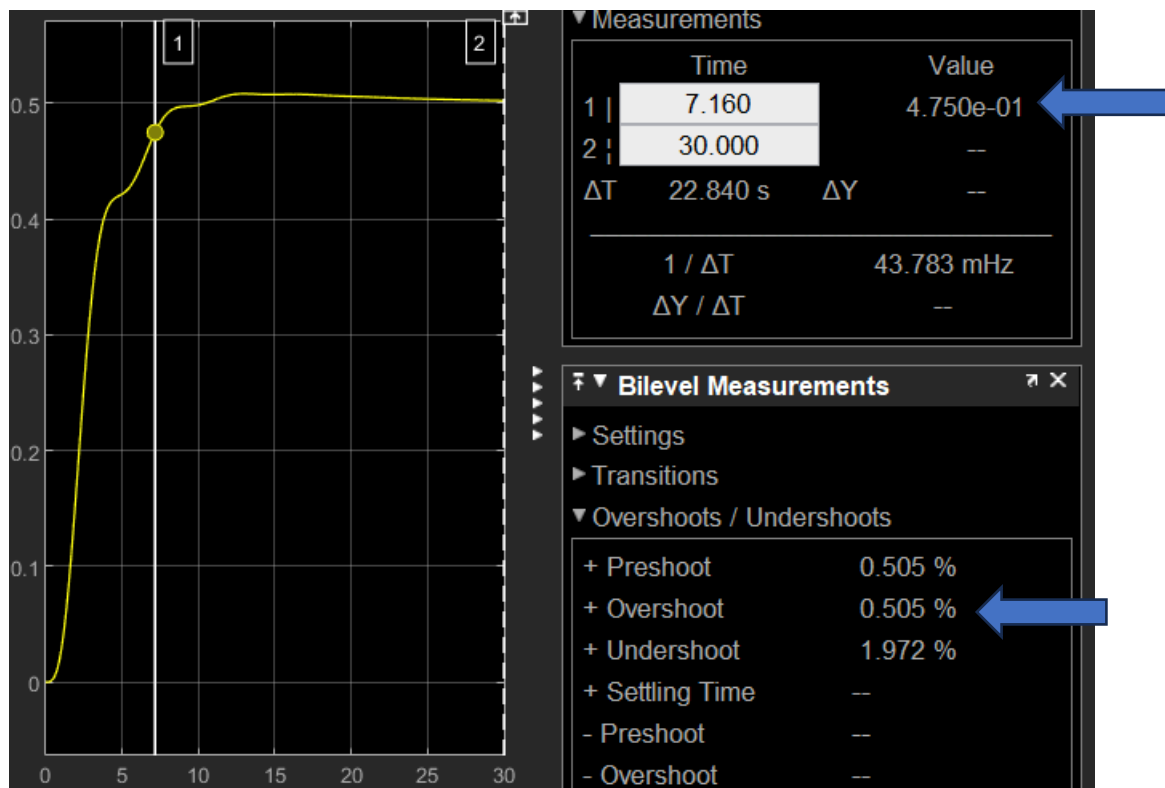
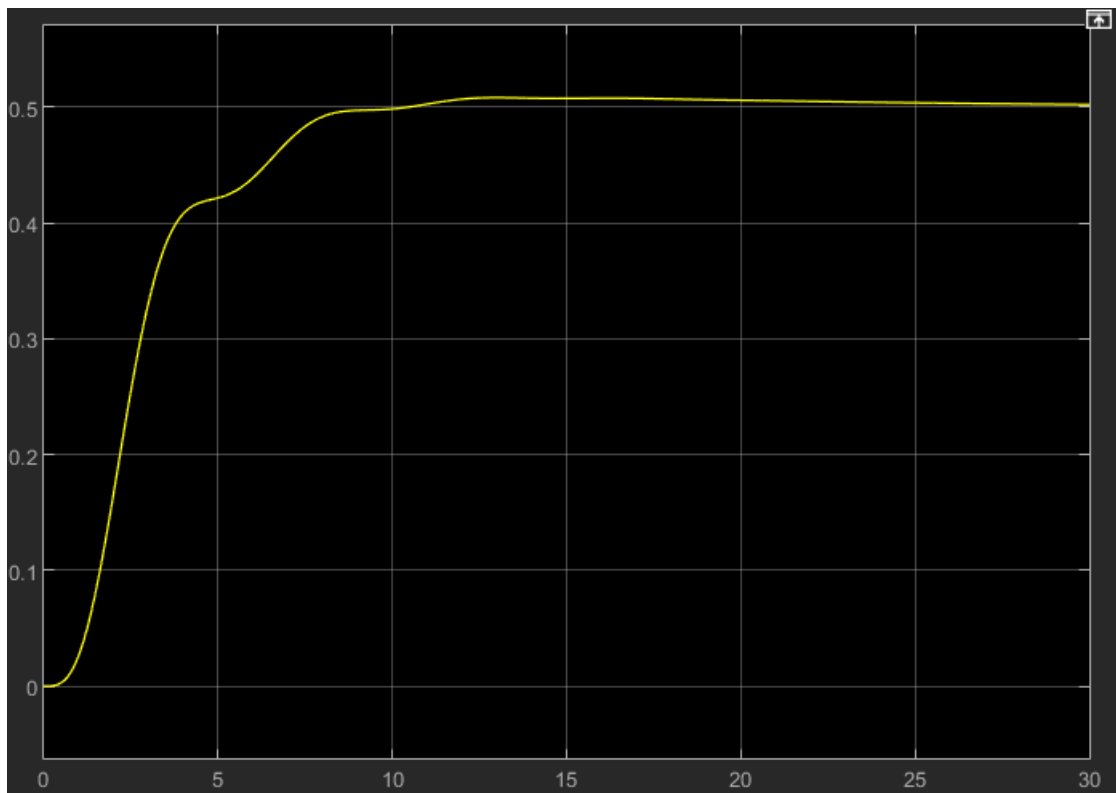


با استفاده از بلوک PID دو درجه آزادی سیستم زیر را می‌بندیم و پس از تیون کردن آن به صورت دستی با جابجایی اسلایدر ها در نهایت به مقادیر زیر می‌رسیم که رفتار خروجی سیستم نیز در زیر آمده است و شرایط مسئله را ارضا می‌کند : (فایل q7)

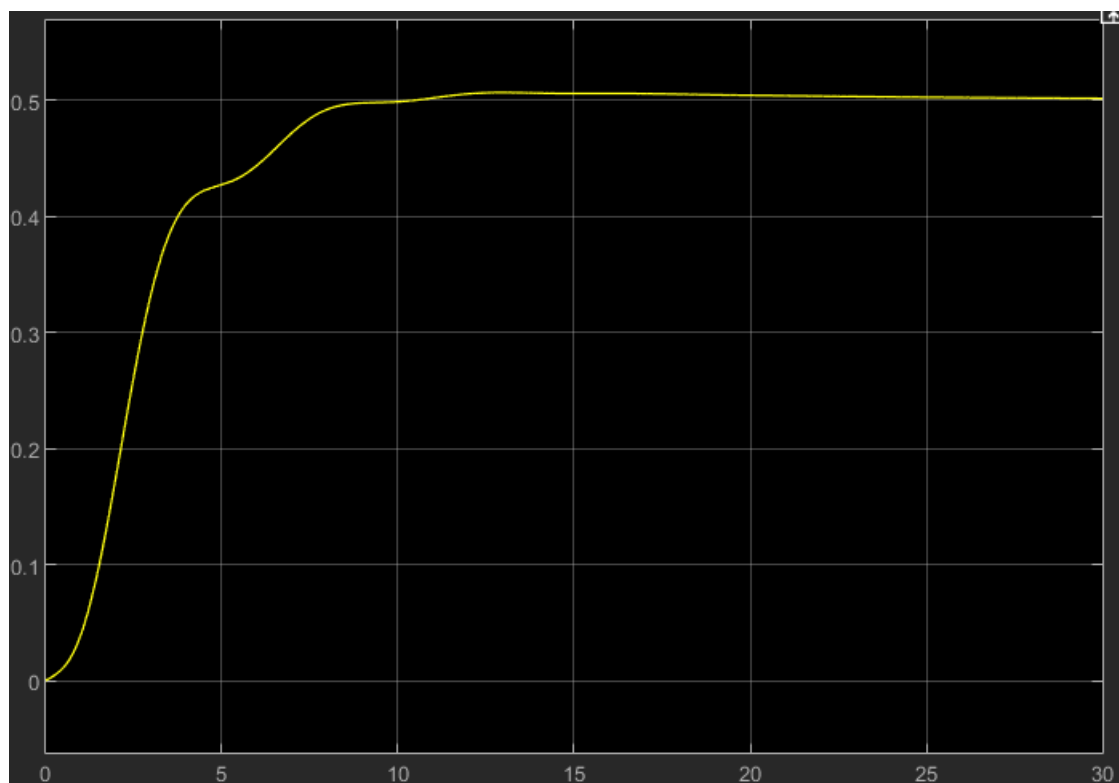


Main	Initialization	Output Saturation	Data Types	State Attributes
Controller parameters				
Source: internal				
Proportional (P): 1.91206426119348				
Integral (I): 0.218784094899679				
Derivative (D): 4.04275038096098				
<input checked="" type="checkbox"/> Use filtered derivative				
Filter coefficient (N): 121.379989923856				
Setpoint weight (b): 1.20927644511706				
Setpoint weight (c): 0.0366462486007094				
Automated tuning				
Select tuning method: Transfer Function Based (PID Tuner App) Tune...				

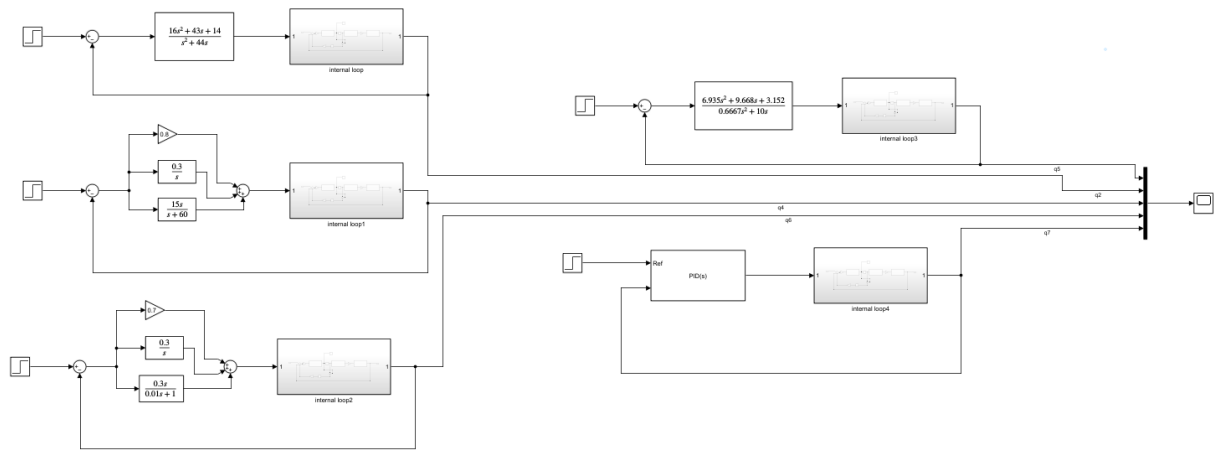
مشاهده می شود که رفتار خروجی دارای حدود 7.2 ثانیه زمان نشست و 0.5% اورشوت است که با
مطلوبات صورت مسئله همخوانی دارد :



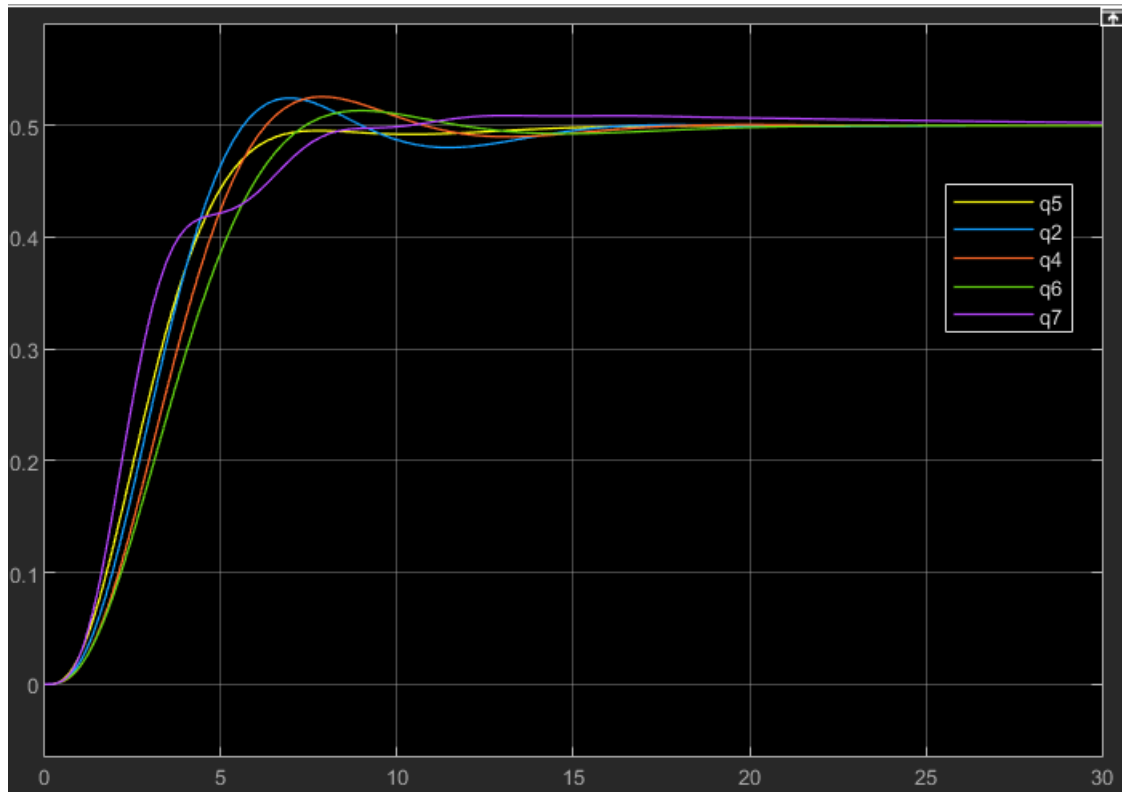
نتیجه اعمال این کنترلر روی مدل واقعی :



برای مقایسه این کنترلر با کنترلر های طراحی شده در قسمت های قبل سیمولینک زیر را می زنیم (به نام q7_compare):



و مقایسه به صورت زیر است :



مشاهده می شود که کنترلر سوال شماره 5 (wjc) زمان نشست کمتری نسبت به بقیه (بعد از q2) دارد اما q2 اورشوت زیادی دارد و در نتیجه همچنان wjc به نظر بهینه ترین کنترلر برای این سیستم می رسد.

برای اجرای سیمولینک های q7 و q7_compare ابتدا کد code ران شود تا متغیر های استفاده شده ایجاد شوند.

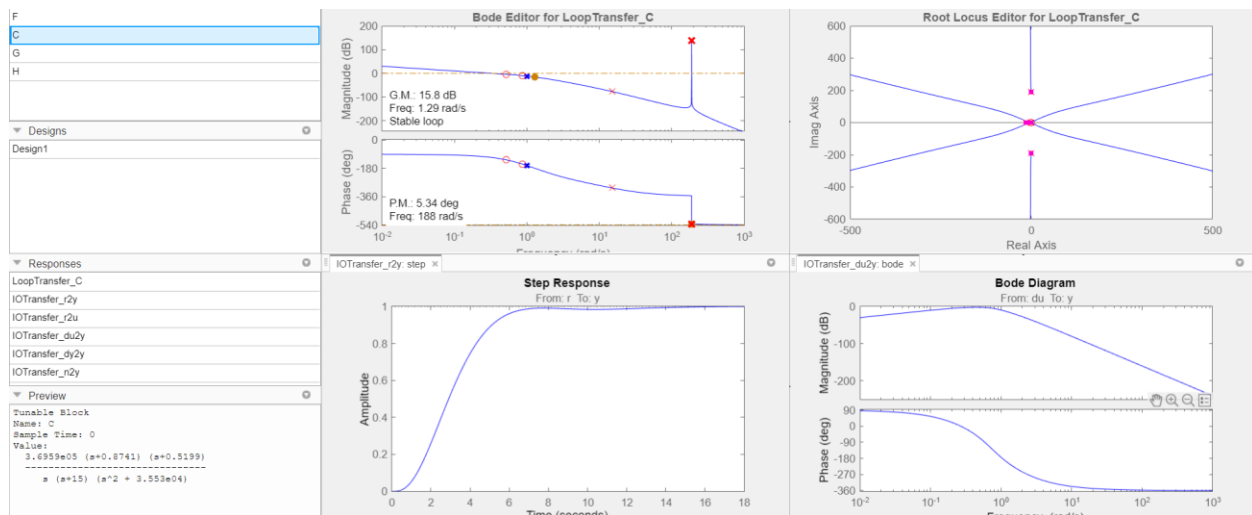
۸. کدامیک از کنترلرهای طراحی شده در قسمت‌های قبل، نسبت به اغتشاشی با فرکانس ۳۰ هرتز مقاوم است؟ اگر هیچکدام از کنترلرهای قبلی این شرایط را ندارند، کنترلری (به روش دلخواه) طراحی کنید که علاوه بر خواسته‌های گفته شده، دامنه‌ی نوسانات نهایی را به کمتر از ۳ درصد دامنه‌ی اغتشاش برساند.

می‌دانیم بنا بر قضیه اصل مدل داخلی برای اینکه سیستمی نسبت به اغتشاش مقاوم باشد باید قطب‌های ناپایدار اغتشاش داخل تابع تبدیل کنترلر وجود داشته باشد (اغتشاش ورودی فرض کرده ایم). با فرض اینکه سیگنال ورودی به موتور اشباع نشود (یعنی از ۲۰ همواره کمتر باشد) تابع تبدیل حلقه بسته داخلی بدست آورده بودیم که فقط ۴ قطب در -۱ دارد و در نتیجه باید در مخرج کنترلر ترم $s^2 + w^2$ وجود داشته باشد که w برابر است با :

$$\omega = 2\pi f = 2\pi \times 30 = 60\pi$$

اما این ترم در مخرج هیچکدام از کنترلر ها نیست.

در فایل سیسو q8 این ترم را به کنترلر wjc طراحی شده اضافه می‌کنیم و می‌بینیم که همچنان زمان نشست و اوروشت به همان صورت است :



کنترلر طراحی شده :

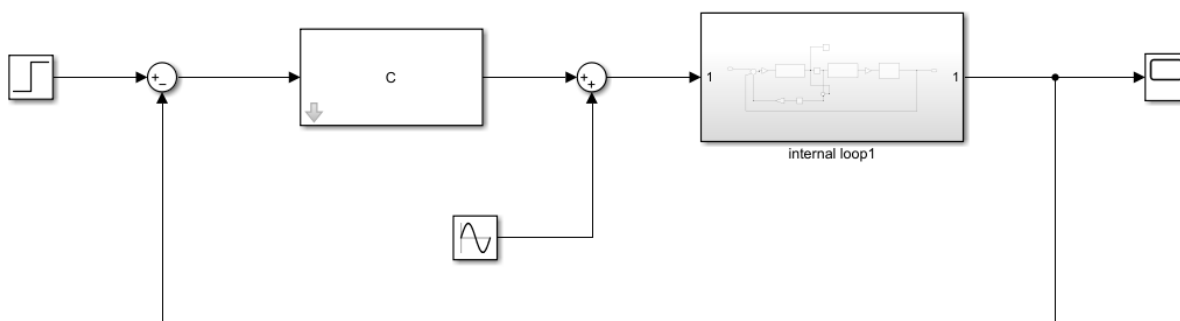
$$G(s) = \frac{369590(s + 0.8741)(s + 0.5199)}{s(s + 15)(s^2 + 35530)}$$

در مدل q8_new این کنترلر را برای دامنه های مختلف تست می کنیم و می بینیم که با وجود اغتشاش با دامنه $60 \cdot \pi$ رادیان بر ثانیه برای دامنه های تا حدود 5 علاوه بر حذف اثر اغتشاش زمان نشست و اورشوت نیز همان شرایط مسئله باقی مانده اند. (به دلیل وجود قطب فوق در کنترلر همواره این فرکانس از اغتشاش را دمپ می کند . بحث بر سر دامنه اغتشاشی است که سیستم برای رفتار گذرای خود نیز به آن مقاوم است).

برای اجرای این کد نیز ابتدا code را اجرا کنید.

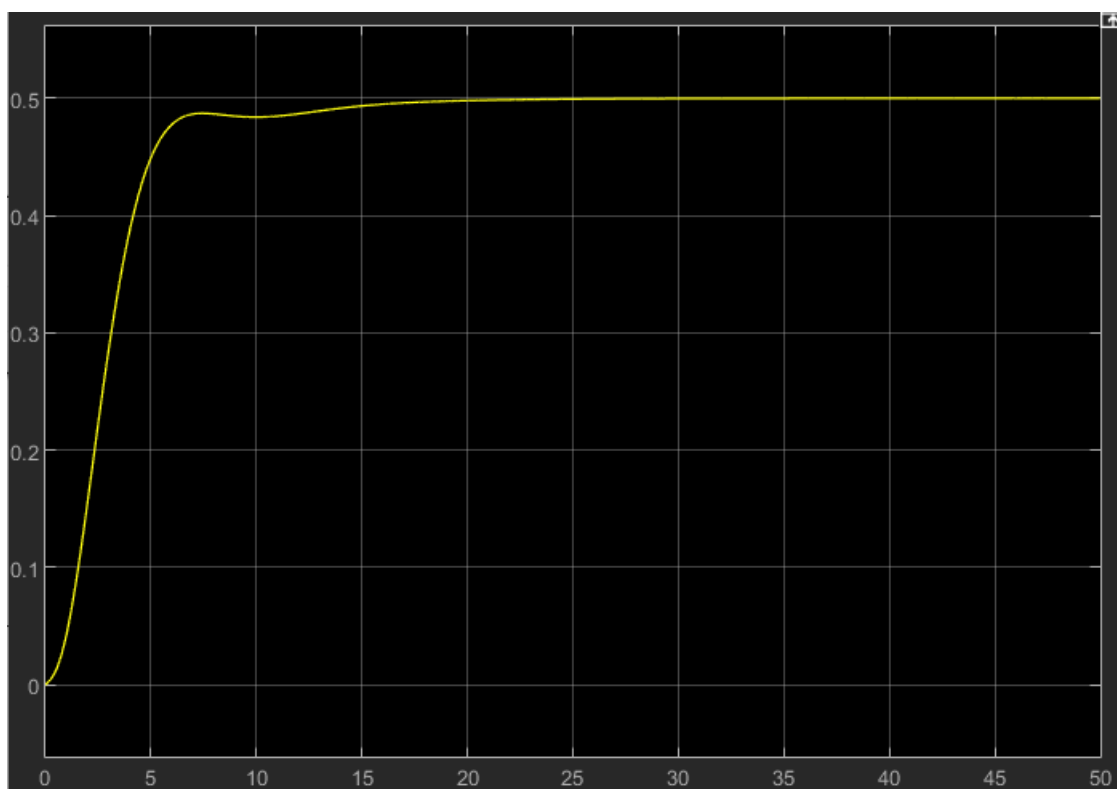
همچنین برای اجرای سیمولینک C را از سیسو export کنید.

رفتار سیستم به ازای دامنه 5 و 7 :



در دامنه 7 می توان دید که سیستم شروع به تغییر زمان نشست می کند.

به ازای دامنه 5



و دامنه 7

