دانشکده برق و کامپیوتر دانشگاه تهران

گزارش پرژه ی درس کنترل مدرن

کنترل ربات دوچرخه

توسط:

آریا رشیدی نژاد میبدی


بسم الله الرحمن الرحیم

چکیده

در این پروژه روبات دوچرخه مورد بررسی قرار گرفته و در ابتدا معادلات حالت غیر خطی آن مطرح گردیده. سپس جواب معادلات حالت و خروجی سیستم مورد بررسی قرار گرفته. باید به این نکته توجه شود که معادلات حالت سیستم ما در سرعت بالا با دقت خوبی نزدیک به فرم خطی میباشد که نشان میدهد در این مساله با خطی سازی میتوان به جواب خوبی رسید.

کلید واژه: ، روبات دوچرخه - معادلات حالت – خطی سازی – کنترل پذیری و رویت پذیری

برای کنترل هر سیستم نیاز به معادلات آن سیستم نیاز داریم. معادلات که تغییرات متغیر های سیستم ما را در هر لحظه نشان میدهد معادلات دیفرانسیل نام دارند. در پروژه ی ما که کنترل ربات دوچرخه مد نظر میباشد ابتدا معادلات دیفرانسیل را استخراج نمودیم و برای مدل سازی و کنترل مساله در مراحل بعد از این مدل استفاده نمودیم.

معادلات دیفرانسیل غیر خطی مربوط به سیستم ما در ادامه آمده است.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = c_1 cos x_3 \dot{x}_4 + c_2 sin x_3 x_4^{\,2} + c_3 u \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = d_1 cos x_3 \dot{x}_2 + d_2 sin x_3 \end{cases} \qquad (1)$$

که در آن متغیر های حالت به صورت زیر خواهند بود. آلفا زاویه ی چرخ جلو دوچرخه و بتا زاویه ی دوچرخه با صفحه عمود بر زمین میباشد.

$$x_1 = \alpha \, , x_2 = \dot{\alpha} \, , x_3 = \beta \, , x_4 = \dot{\beta}$$

سایر مقادیر هم در زیر تعریف شده اند در آنها m1 وزن چرخ های دوچرخه m2 جرم فریم مثلثی دوچرخه r شعاع چرخ لاندا فاصله ی مرکز جرم چرخ جلو و زاویه چرخش چنگال جلو میباشد.

$$c_1 = \frac{-(m_1 r + m_2 h\sigma)\lambda}{\frac{1}{2} m_1 r^2 + m_1 \lambda^2 + m_2 \sigma^2 \lambda^2} = -c_2 \quad , \quad c_3 = \frac{1}{\frac{1}{2} m_1 r^2 + m_1 \lambda^2 + m_2 \sigma^2 \lambda^2}$$

$$d_1 = \frac{-(m_1 r + m_2 h\sigma)\lambda}{3 m_1 r^2 + 2 m_2 h^2} \, , d_2 = \frac{(m_2 h + 2 m_1 r) g}{3 m_1 r^2 + 2 m_2 h^2} \, .$$

مساله ی دوچرخه یکی از مسایل داغ در حوزه کنترل ربات میباشد زیرا این مساله در حالت ایستا ناپایدار و در حالت پویا پایدار میباشد. در واقع معادلات حالت در زمان جلو رفتن با سرعت بالا تا حدود خوبی به حالت خطی نزدیک میباشد.

معادلات دیفرانسیل موجود در صفحه قبل معادلاتی درجه یک هستند ولی معادلات حالت نیستند زیرا باید در معادلات حالت در هر جمله مشتق یکی از متغیر ها ظاهر شود که به راحتی میتوان این کار را انجام داد و معادلات حالت را به صورت زیر بدست آورد.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = [0.5 \cdot d_2 c_1 sin(2x_3) + c_2 sin x_3 x_4{}^2 + c_3 u] / (1 - d_1 c_1 cos^2 x_3) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = [0.5 \cdot d_1 c_2 sin(2x_3) x_4{}^2 + d_1 c_3 cos x_3 u + d_2 sin x_3] / (1 - d_1 c_1 cos^2 x_3) \end{cases}$$

که خروجی معادلات به صورت زیر است.

$$y = x_3$$

در ابتدا به سراغ حل معادلات حالت به صورت عددی در نرم افزار مطلب میپردازیم. برای حل از این نکته استفاده میکنیم که در نقطه های نزدیک میتوانیم متغیر حالت را با مشتق آن و مقدار آن در نقطه قبلی حساب نمود. این موضوع و روش در زیر نوشته شده.

$$x_i(t+dt) \simeq x_i(t) + dt \ \dot{x}_i(t)$$

$$x_i(0.001) = x_i(0) + 0.001\,\dot{x}_i(0)$$

$$x_i(0.002) = x_i(0.001) + 0.001\,\dot{x}_i(0.001)$$

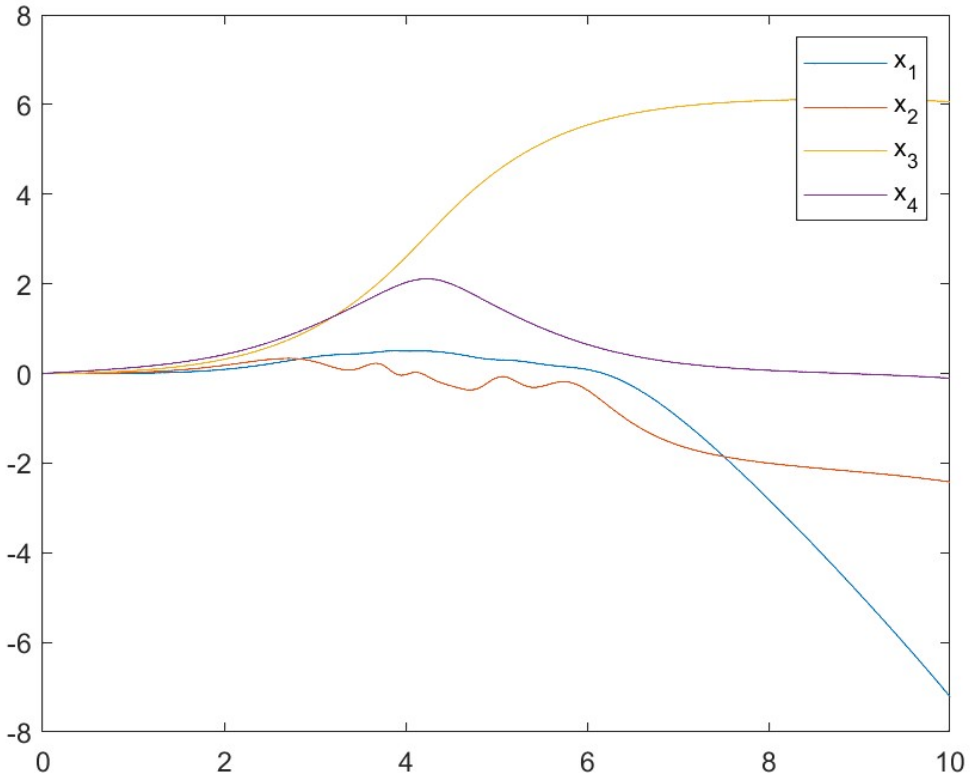
$$\vdots$$

$$x_i(10) = x_i(9.999) + 0.001\,\dot{x}_i(9.999)$$

در ادامه این روش را در نرم افزار متلب پیاده سازی نمودیم.

```
clc
clear all
%time length
t_0=0;
t_final=1;
dt=0.001;
t =t_0:dt:t_final;

%initialization
d2=1;
c1=1;
c2=-1;
c3=1;
d1=0.1;
u=1;
x1(1)=0;
x2(1)=0;
x3(1)=0;
x4(1)=0;

%for loop
for i=2:length(t)
x1(i)=dt*x2(i-1) +x1(i-1);
x2(i)=dt*((0.5*d2*c1)*sin(2*x3(i-1)) + c2*sin(x3(i-1)*x4(i-1)*x4(i-1))) /(1-d1*d2*cos(x3(i-1))*cos(x3(i-1)))+ x2(i-1);
x3(i)=dt*x4(i-1) +x3(i-1);
x4(i)=dt*(0.5*d1*c2*sin(2*x3(i-1))*x4(i-1)*x4(i-1) + d1*c3*cos(x3(i-1)*u) +d2*sin(x3(i-1)))/(1-d1*c3*(cos(x3(i-1))^2))+ x4(i-1);
end

plot(x3,t)
```

خروجی ما به ازای ورودی پله به صورت زیر بود

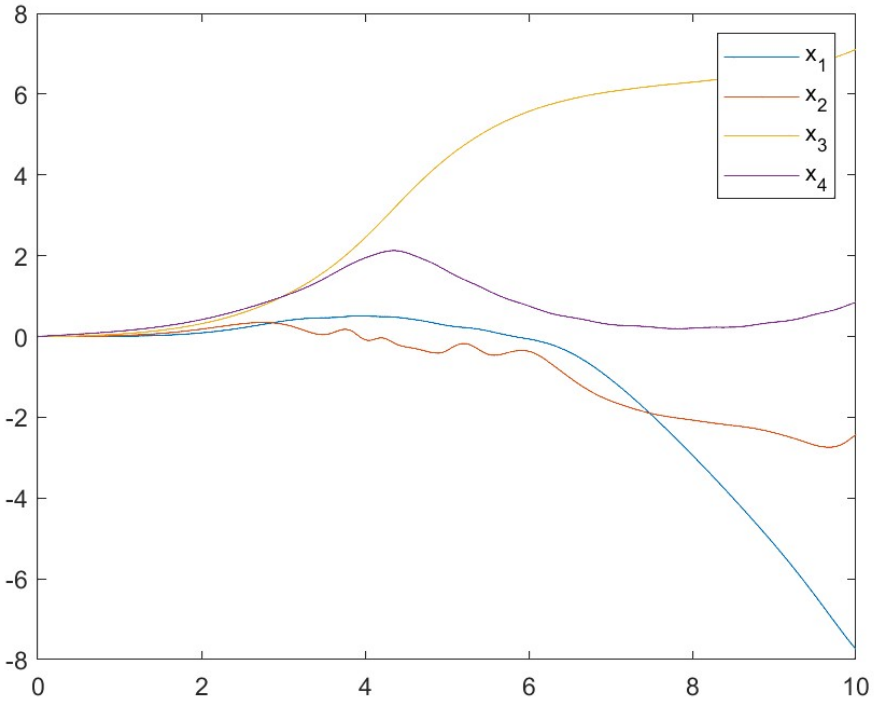


هم چنین با تابع ode45 معادلات حالت را حل نموده و به جواب های مشابهی رسیدیم.

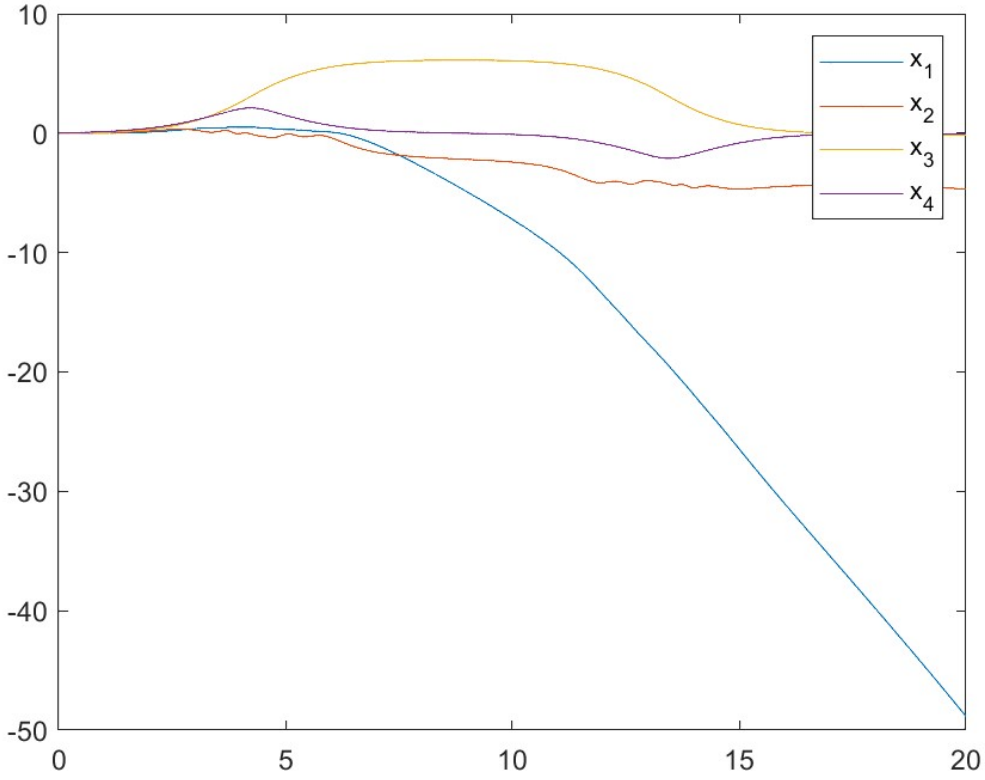
```
clear;
clc;
tspan=[0 10];
x1init=0;
x2init=0;
x3init=0;
x4init=0;


[t,x] = ode45(@bicycle ,tspan,[x1init x2init x3init x4init]);
plot(t,x(:,1),t,x(:,2),t,x(:,3),t,x(:,4))
xlabel=('time');
ylable=('x3');
legend('x_1','x_2','x_3','x_4');
```

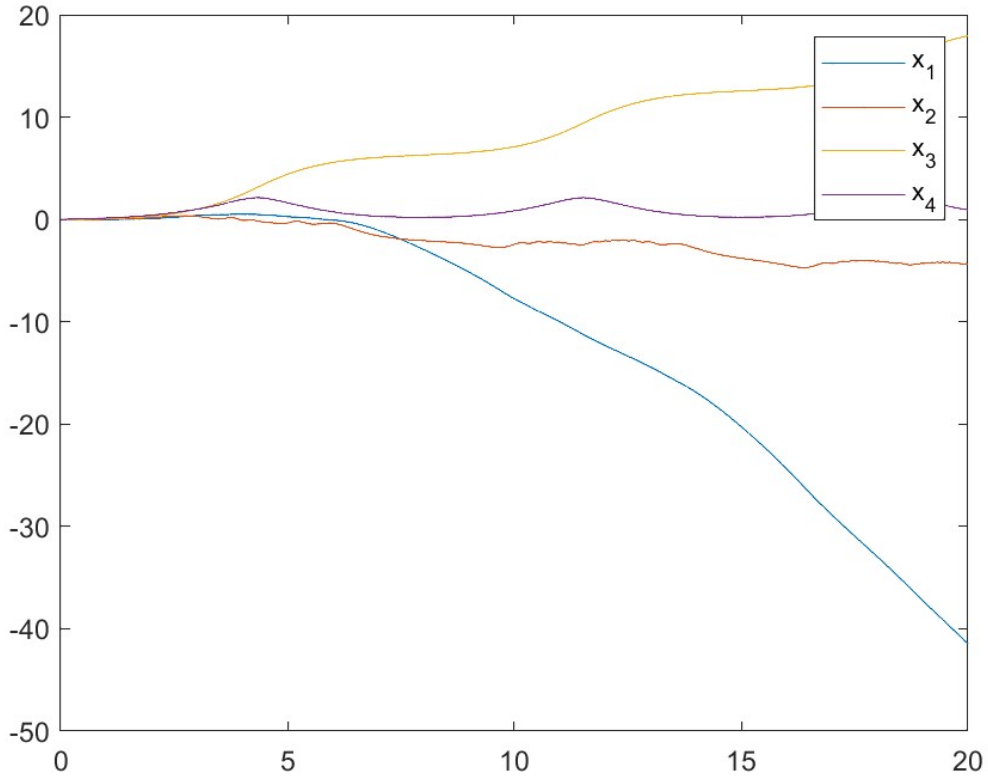
```
function dxdt=bicycle(t,x)
d2=1;
c1=1;
c2=-1;
c3=1;
d1=0.1;
u=1;

dxdt=zeros(4,1);
dxdt(1)=x(2);
dxdt(2)=( 0.5*d2*c1*sin(2*x(3)) + c2*sin(x(3)*x(4)*x(4)) +c3*u  ) / (  1-d1*c1*cos(x(3))*cos(x(3))  );
dxdt(3)=x(4);
dxdt(4)=( 0.5*d1*c2*sin(2*x(3))*x(4)*x(4) + d1*c3*cos(x(3))*u + d2*sin(x(3))  )/ ( 1-d1*c1*cos(x(3))*cos(x(3)) );
```

همچنین متغیر های حالت ما به ازای ورودی شیب واحد به صورت زیر در آمد.



برای درک بهتر در این صفحه پاسخ سیستم به ورودی پله و شیب به ترتیب آورده شده.




20
10
0
-10
-20
-30
-40
-50
0
5
10
15
20
x1
x2
x3
x4

کنترل پذیری سیستم:

در این بخش کنترل پذیری سیستم بررسی میشود. قبل از بررسی های فوق باید ابتدا نقاط تعادل سیستم را شناسایی کنیم و سپس حول نقطه تعادل معادلات دینامیکی سیستممان را خطی سازی نماییم. بعد از خطی سازی میتوانیم از آزمون های کنترل پذیری و رویت پذیری استفاده نماییم.

خطی سازی:

برای خطی سازی سیستم ابتدا فرض میکنیم که دوچرخه با سرعت حرکت کند در این حالت داریم:

X3=beta=0

با فرض بالا میتوانیم از تقریب های زیر استفاده نماییم:

Sin(x3)=0

Cos(x3)=1

حال معادلات به حالت زیر در می آیند:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = (d_2 c_1 x_3 + c_2 x_3 x_4{}^2 + c_3 u)/(1 - d_1 c_1) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = (d_1 c_2 x_3 x_4{}^2 + d_2 x_3 + d_1 c_3 u)/(1 - d_1 c_1) \end{cases}$$

با توجه به کوچک بودن x3 میتوانیم از تقریب زیر استفاده نماییم.

X3(x4^2)=0

حال معادلات حالت ما به فرم زیر در می آیند:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = (d_2 c_1 x_3 + c_3 u)/(1 - d_1 c_1) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = (d_2 x_3 + d_1 c_3 u)/(1 - d_1 c_1) \end{cases}$$

میتوانیم معادلات فوق را به فرم ماتریسی بنویسیم:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{d_2 c_1}{1-d_1 c_1} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{d_2}{1-d_1 c_1} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{c_3}{1-d_1 c_1} \\ 0 \\ \frac{d_1 c_3}{1-d_1 c_1} \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T$$

یا به عبارت دیگر:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u \\ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + Du \end{cases}$$

که ماتریس های فوق به فرم زیر خواهند بود (با در نظر گرفتن مقادیر ثابت مطابق جدول 1 در صفحه بعد:

$$\text{In which, } \boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -10.9426 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5.79882 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 7.06703 \\ 0 \\ -0.0613 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}, D = 0, \boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T, \boldsymbol{y} = x_1.$$

TABLE I

THE NUMERICAL VALUE OF THE ROBOT COEFFICIENT

Name	Physical sense	Value
m_1	Mass of the wheel	2.5kg
m_2	Mass of the triangle frame	18kg
m_3	Mass of the balance device	5kg
r	Radius of the wheel	0.33m
λ	A constant distance of steering device	0.04m
h	Height of COG	0.92m
k_1	The distance between A and COG	0.7m
k_2	The distance between A and C	1.1m
σ	$(k_2 - k_1)/k_2$	0.36

کنترل پذیری حالت با ماتریس کنترل پذیری:

برای بررسی کنترل پذیری ابتدا ماتریس کنترل پذیری را تشکیل میدهیم و رنک آن را حساب میکنیم. با محاسبه در متلب مشخص میگردد که ماتریس مورد نظر fullrank میباشد. پس سیستم ما کنترل پذیر حالت است. کد های مورد نظر در ادامه آمده است.

```
%Linear System
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [1 0 0 0];
ctrb(A,B)
rank(ctrb(A,B))
```

خروجی کد مورد نظر را در زیر مشاهده مینمایید.

```
ans =

         0    7.0670         0    0.6708
    7.0670         0    0.6708         0
         0   -0.0613         0   -0.3555
   -0.0613         0   -0.3555         0


ans =

     4
```

پس ماتریس کنترل پذیری ما به فرم زیر محاسبه گردید.

$$\boldsymbol{M} = \begin{bmatrix} 0 & 7.067 & 0 & 0.671 \\ 7.067 & 0 & 0.671 & 0 \\ 0 & -0.0613 & 0 & -0.3556 \\ -0.0613 & 0 & -0.3556 & 0 \end{bmatrix}$$

کنترل پذیری حالت با آزمون PBH :

برای استفاده از pbh کافی است فقط مقادیر ویژه در PBH قرار داده و بررسی شود.

برای مقدار ویژه اول و دوم داریم:

$$\lambda_1 = \lambda_2 = 0$$

```
%PBH Test
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [1 0 0 0];
landa=0;
PBH=[A-landa*eye B]
rank(PBH)
```

خروجی به صورت زیر خواهد بود:

```
PBH =

         0    1.0000         0         0         0
         0         0  -10.9426         0    7.0670
         0         0         0    1.0000         0
         0         0    5.7988         0   -0.0613


ans =

     4
```

```
PBH =

   -2.4081   -1.4081   -2.4081   -2.4081         0
   -2.4081   -2.4081  -13.3507   -2.4081    7.0670
   -2.4081   -2.4081   -2.4081   -1.4081         0
   -2.4081   -2.4081    3.3907   -2.4081   -0.0613


ans =

     4
```

برای مقدار ویژه 2.4081 نیز داریم:

```
PBH =

    2.4081    3.4081    2.4081    2.4081         0
    2.4081    2.4081   -8.5345    2.4081    7.0670
    2.4081    2.4081    2.4081    3.4081         0
    2.4081    2.4081    8.2069    2.4081   -0.0613


ans =

     4
```

همچنین برای مقدار ویژه 2.4081- داریم:

پس با معیار pbh نیز سیستم ما کنترل پذیر حالت میباشد.

رویت پذیری حالت با ماتریس رویت پذیری:

کد مربوط به رویت پذیری با ماتریس رویت پذیری در زیر آمده که از آن رویت پذیری حالت سی سیستم نتیجه میشود:

```
%observability
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [1 0 0 0];
obsv(A,C)
rank(obsv(A,C))
```

```
ans =

    1.0000         0         0         0
         0    1.0000         0         0
         0         0  -10.9426         0
         0         0         0  -10.9426


ans =

     4
```

خروجی به صورت زیر مشاهده گردید:

رویت پذیری حالت با آزمون PBH:

```
%PBH Observability
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [1 0 0 0];
landa=0;
PBH=[A-landa*eye;C]
rank(PBH)
```

برای مقدار ویژه 0 داریم:

خروجی برای مقدار ویژه اول و دوم:

خروجی برای مقدار ویژه 2.4081

```
PBH =

   -2.4081   -1.4081   -2.4081   -2.4081
   -2.4081   -2.4081  -13.3507   -2.4081
   -2.4081   -2.4081   -2.4081   -1.4081
   -2.4081   -2.4081    3.3907   -2.4081
    1.0000         0         0         0


ans =

     4
```

همچنین خروجی برای مقدار ویژه آخربه صورت زیر خواهد بود:

```
         0    1.0000         0         0
         0         0  -10.9426         0
         0         0         0    1.0000
         0         0    5.7988         0
    1.0000         0         0         0


ans =

     4
```

```
PBH =

    2.4081    3.4081    2.4081    2.4081
    2.4081    2.4081   -8.5345    2.4081
    2.4081    2.4081    2.4081    3.4081
    2.4081    2.4081    8.2069    2.4081
    1.0000         0         0         0


ans =

     4
```

با دو روش ثابت نمودیم سیستم ما حول نقطه کارش کنترل پذیر حالت و رویت پذیر حالت میباشد.

بررسی رویت پذیری سیستم با در نظر گرفتن متغیر های مختلف جهت اندازه گیری:

در قسمت مشاهده کردیم به ازای y=x1 سیستم ما مشاهده پذیر میباشد. در این قسمت با قرار دادن y=x2 یا y=x3 یا y=x4 مشاهده پذیری سیستم را بررسی مینماییم. بدین منظور از ماتریس مشاهده پذیری استفاده مینماییم.

حالت y=x2 :

```
%observability
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [0 1 0 0];
obsv(A,C)
rank(obsv(A,C))
```

```
ans =

         0    1.0000         0         0
         0         0  -10.9426         0
         0         0         0  -10.9426
         0         0  -63.4542         0


ans =

     3
```

در این حالت سیستم ما مشاهده پذیر نمیباشد.

حالت y=x3 :

```
%observability
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [0 0 1 0];
obsv(A,C)
rank(obsv(A,C))
```

```
ans =

         0         0    1.0000         0
         0         0         0    1.0000
         0         0    5.7988         0
         0         0         0    5.7988


ans =

     2
```

در این حالت نیز سیستم ما مشاهده پذیر نمیباشد.

```
%observability
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [0 0 0 1];
obsv(A,C)
rank(obsv(A,C))
```

حالت y=x4 :

```
ans =

         0         0         0    1.0000
         0         0    5.7988         0
         0         0         0    5.7988
         0         0   33.6263         0


ans =

     2
```

در این حالت نیز سیستم ما مشاهده پذیر نمیباشد. پس مشخص است که باید از خروجی استفاده که در آن حالت سیستم قابل مشاهده باشد.

طراحی کنترلر تناسبی انتگرالی مشتقگیر(PID):

برای طراحی کنترلر PID از فیدبک خروجی استفاده میکنیم و ضریبی از خود خروجی مشتق خروجی و انتگرال خروجی را به ورودی سیستم فیدبک میدهیم. در این حالت خواص داخلی سیستم اهمیتی ندارد و فقط رابطه ی ورودی خروجی مهم است. یعنی تابع تبدیل اهمیت دارد.

در متلب ابتدا تابع تبدیل را بدست می آوریم و سپس ضرایب را با pid tune بدست می آوریم.

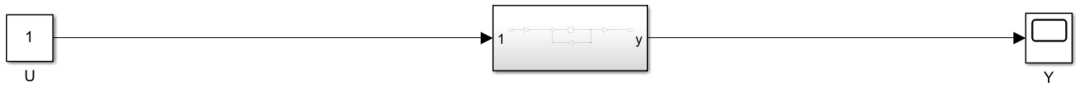
```
%PID Controller
clear;
clc;
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [1 0 0 0];
D=0;
%sys=ss(A,B,C,D);
%sys_as_tf = tf(sys);
[b,a] = ss2tf(A,B,C,D);
sys = tf(b,a);
%[C_pid,info] = pidtune(sys,'PID')
%T_pid = feedback(C_pid*sys, 1);
%step(T_pid)
[C_pid,info] = pidtune(sys,'PID')
```

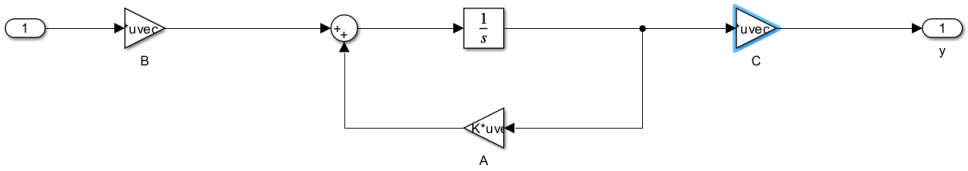
پس ضرایب ترم های مشتق گیر – انتگرال گیر و تناسبی بدست می آیند.

```
C_pid =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = -0.00502, Ki = -4.39e-05, Kd = -0.143

Continuous-time PID controller in parallel form.


info =

  struct with fields:

               Stable: 0
    CrossoverFrequency: 1
          PhaseMargin: NaN
```

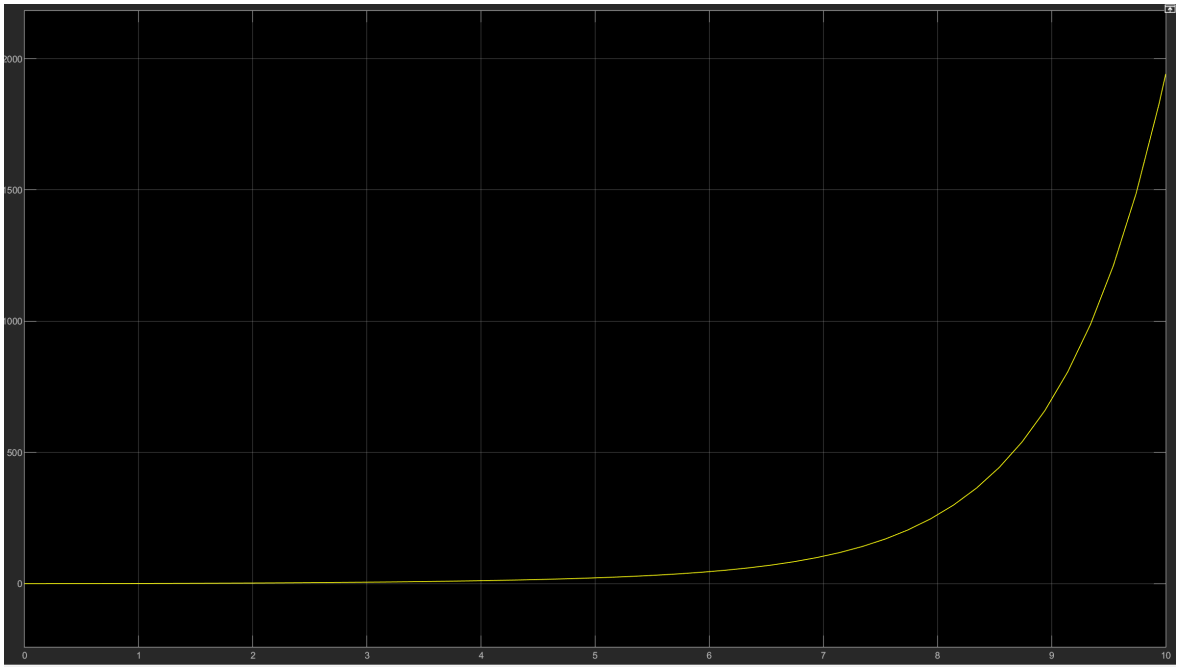
Implementing PID using Simulink

برای طراحی در سیمولینک ابتدا پاسخ سیستم حلقه بسته را میابیم.

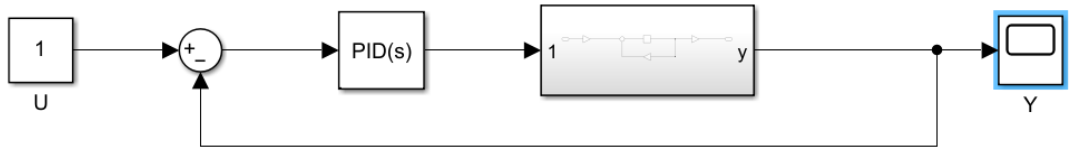




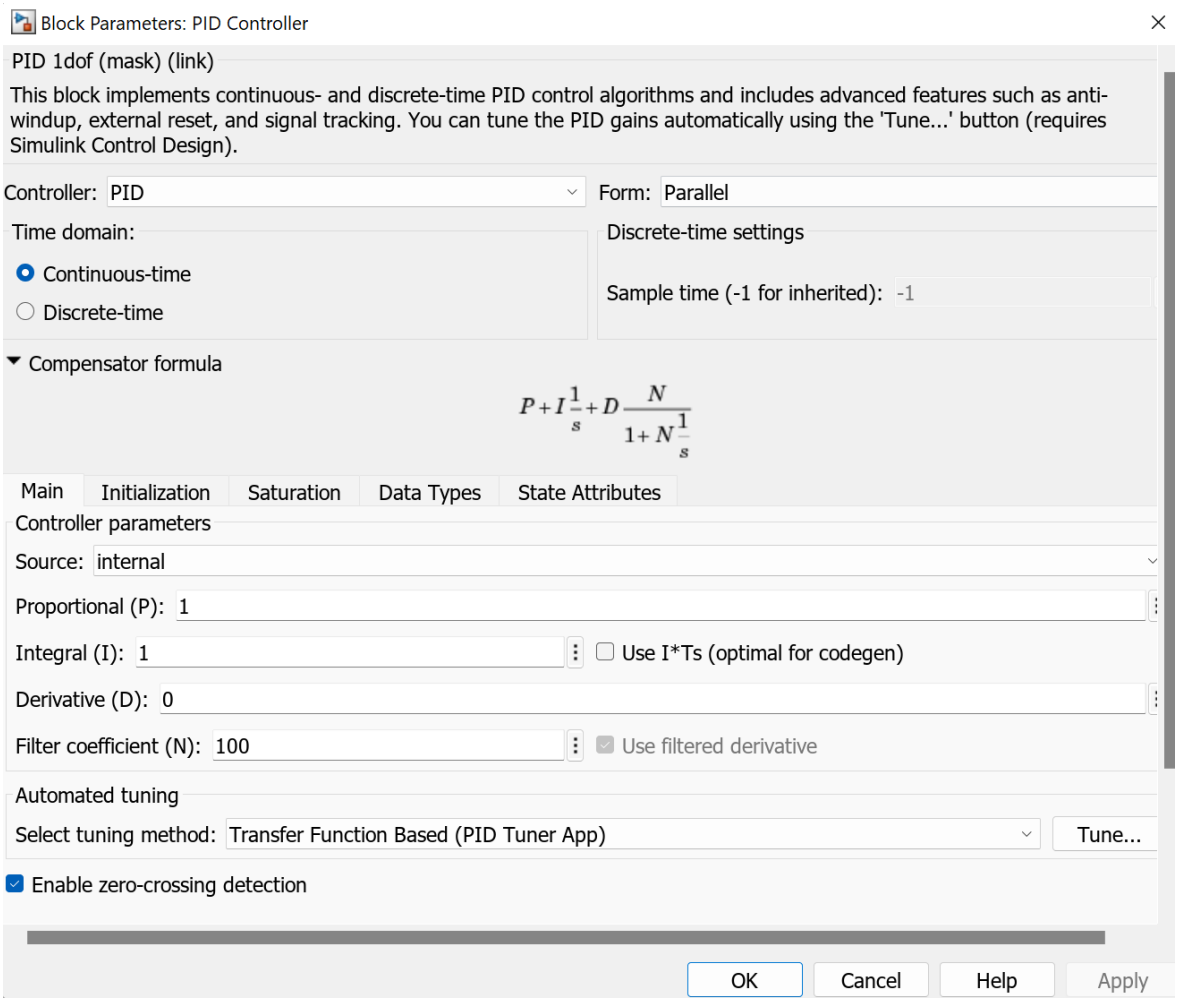
خروجی سیستم حلقه باز به صورت زیر بدست می آید.



که ناپایدار است. برای طراحی کنترلر از سلف تیونینگ سیمولینک بهره میگیریم.

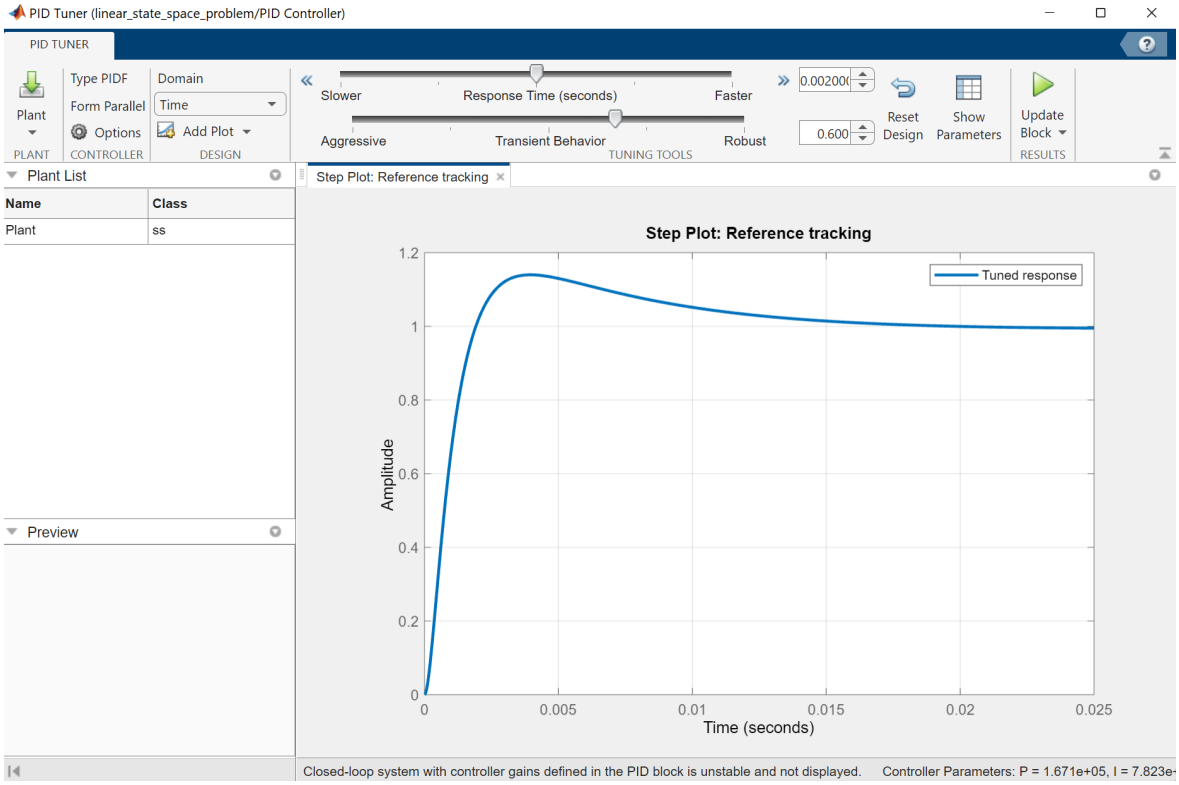


حال با کلیک بر روی PID صفحه زیر باز میشود



که نشان میدهد ضرایب هنوز تیون نشده با کلیک بر روی تیون ضرایب بدست می آید.

به صورت پیش فرض هدف کنترلی در تیونر reference tracking میباشد که با هدف ما ساز گاری دارد.



با کلیک بر روی update block میتوانیم ضرایب بلوک پی آی دی را تست کنیم.

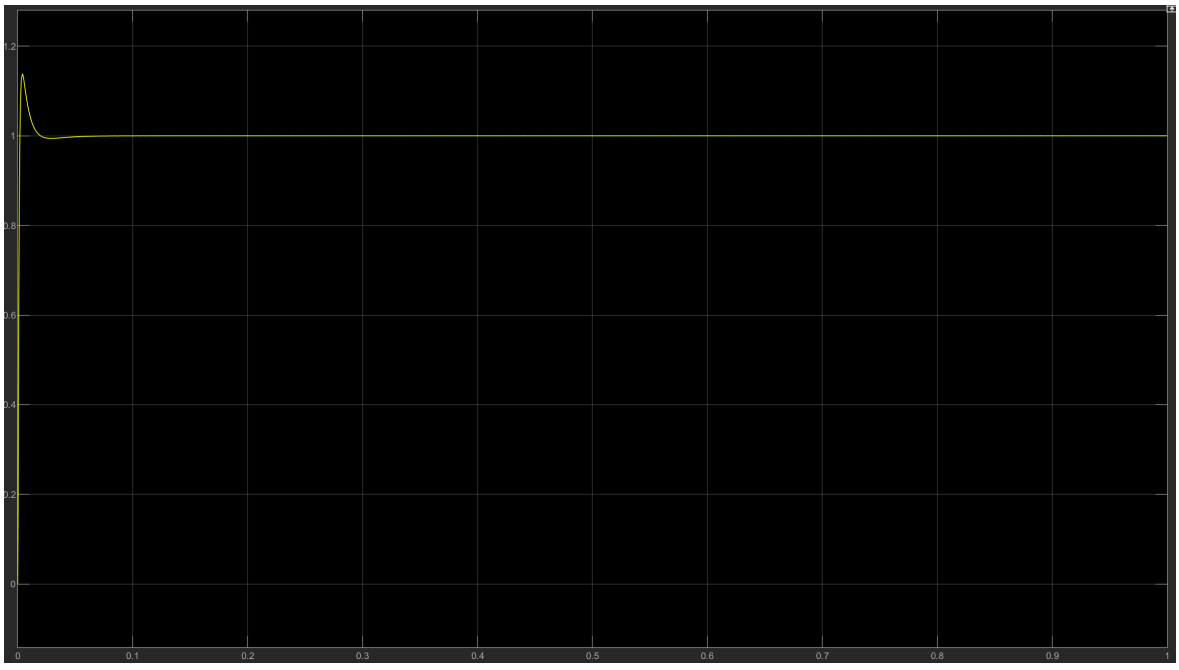
حال میبینیم در صفحه قبل تنظیمات عوض شده.

Proportional (P): 167111.522623289

Integral (I): 7823288.87960466

Derivative (D): 876.529857359357

حال دوباره سیستم را اجرا مینماییم و متوجه میشویم با این ضرایب هدف کنترلی ما محقق شده است.



طراحی کنترلر با استفاده از فیدبک حالت

با استفاده از دستور eig در متلب میتوانیم مقادیر ویژه ی سیستم را بیابیم.

مقادیر ویژه ی سیستم ما به صورت زیر در می آید.

```
%State Feedback Controler
clear;
clc;
A = [0 1 0 0;0 0 -10.9426 0;0 0 0 1;0 0 5.79882 0];
B = [0;7.06703;0;-0.0613];
C = [1 0 0 0];
D=0;
eig(A)
```

مقادیر ویژه به صور زیر در می آید.

```
ans =

         0
         0
    2.4081
   -2.4081
```

مشاهده میشود سیستم ما ناپایدار میباشد. پس باید با استفاده از فیدبک حالت مقادیر ویژه آن را به مقادیر ویژه دلخواهمان ببریم.

چون سیستم ما کنترل پذیر میباشد میتوانیم مقادیر ویژه آن را به مقادیر ویژه دلخواهمان تغییر دهیم. در واقع شرط استفاده از فیدبک حالت در کنترل سیستم و جایابی مناسب قطب ها کنترل پذیری حالت سیستم میباشد.

ما مقادیر ویژه خود را به -6 -5 -4 -1 میبریم .

```
%State Feedback Controler
clear;
clc;
A=readmatrix('A_barr');
B=readmatrix('B_barr');
C = [1 0 0 0];
D=0;
%controlability
rank(ctrb(A,B))
%our eigen values
eig(A);
%designing K
eigs = [-1;-4;-5;-6]; %desired eigen values we want!
K = place (A,B,eigs);
eig(A-B*K)
```

برای طراحی K بدین منظور از دستور place استفاده میکنیم. در انتها نیز برای تست عملکرد Kبدست آمده مقادیر ویژه ی A-BK را بدست می آوریم.

```
-6.0000
-5.0000
-4.0000
-1.0000
```

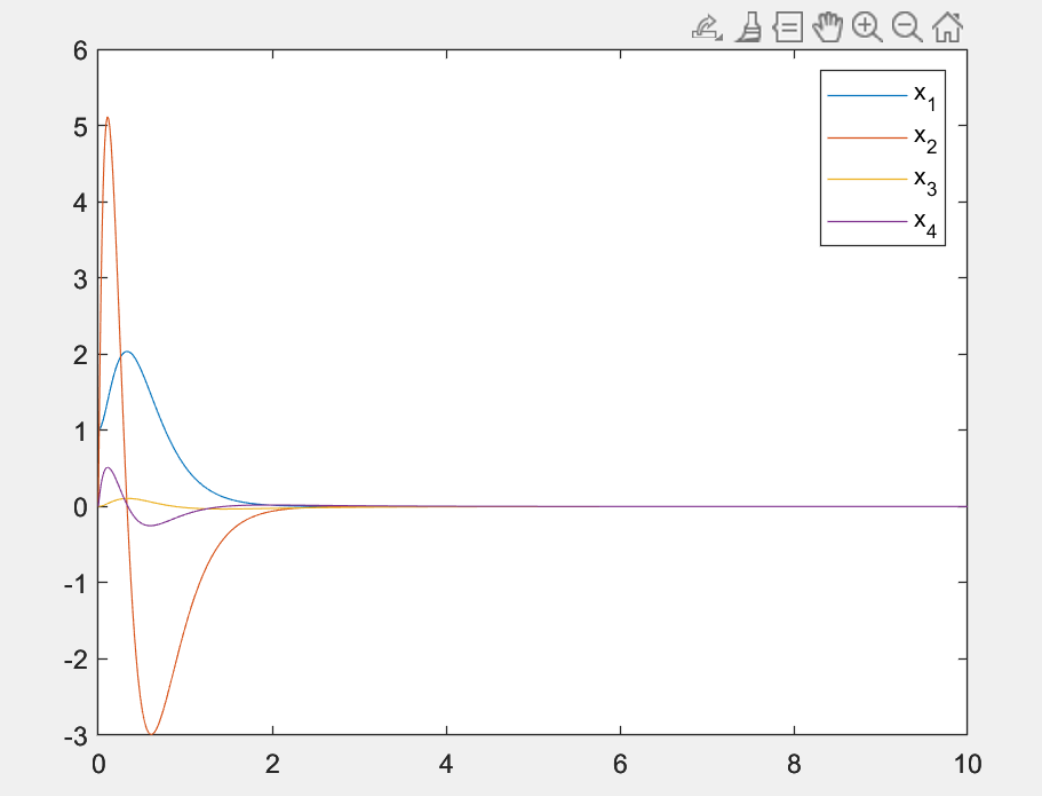
مشاهده میشود مقادیر ویژه ی سیستم جدید به مقادیر ویژه دلخواهمان رسیده است.

حال که K را بدست آورده ایم از آن بر روی سیستم غیر خطی مان استفاده میکنیم تا عملکرد آن را در حضور فیدبک حالت بررسی نماییم.

بدین منظور از دستور ode45 استفاده مینماییم.

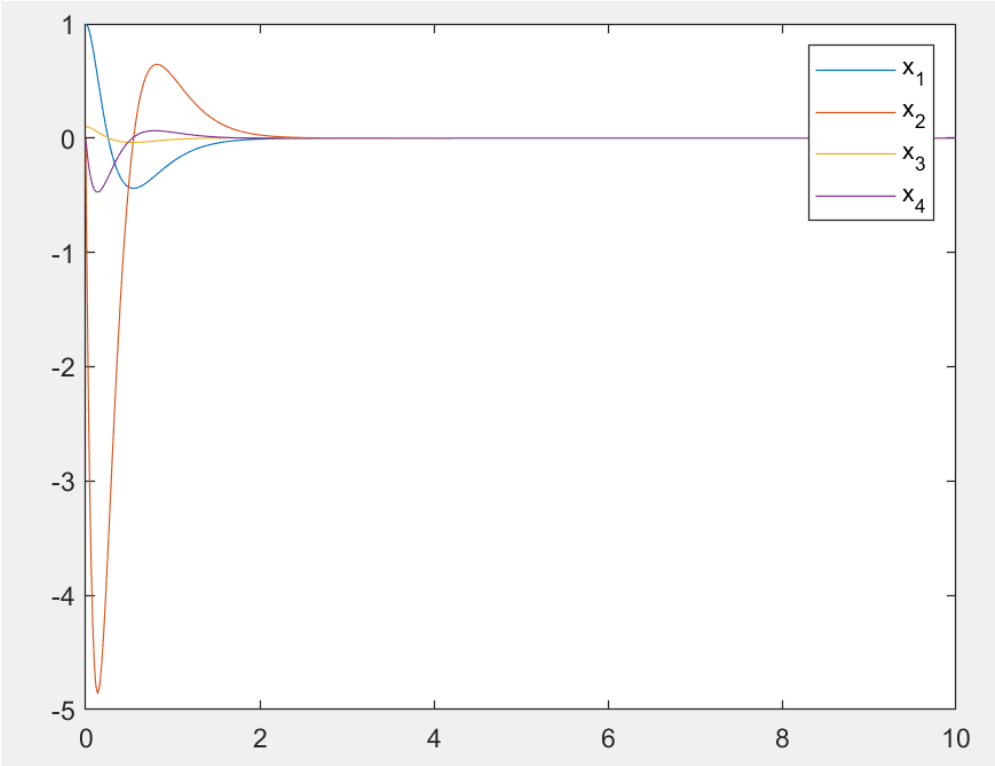
```
%testing on our nonlinear system
tspan=[0 10];
yinit=[1 0 0 0];
[t,y] = ode45(@(t,y)bicycle(y,-K*y),tspan,yinit);
plot(t,y(:,1),t,y(:,2),t,y(:,3),t,y(:,4))
xlabel=('time');
ylable=('x3');
legend('x_1','x_2','x_3','x_4');
```

با شرط اولیه 1 برای x1 و صفر برای باقی متغیر ها خروجی سیستم به صورت زیر در می آید.



نتیجه ی فوق قابل انتظار بود زیرا set point خاصی به سیستم ندادیم و با پایدار سازی باید مقادیر به صفر همگرا شود که این اتفاق افتاده.

در تست دوم شرط اولیه x3 را نیز 0.1 در نظر میگیریم که خروجی تغییر میکند.

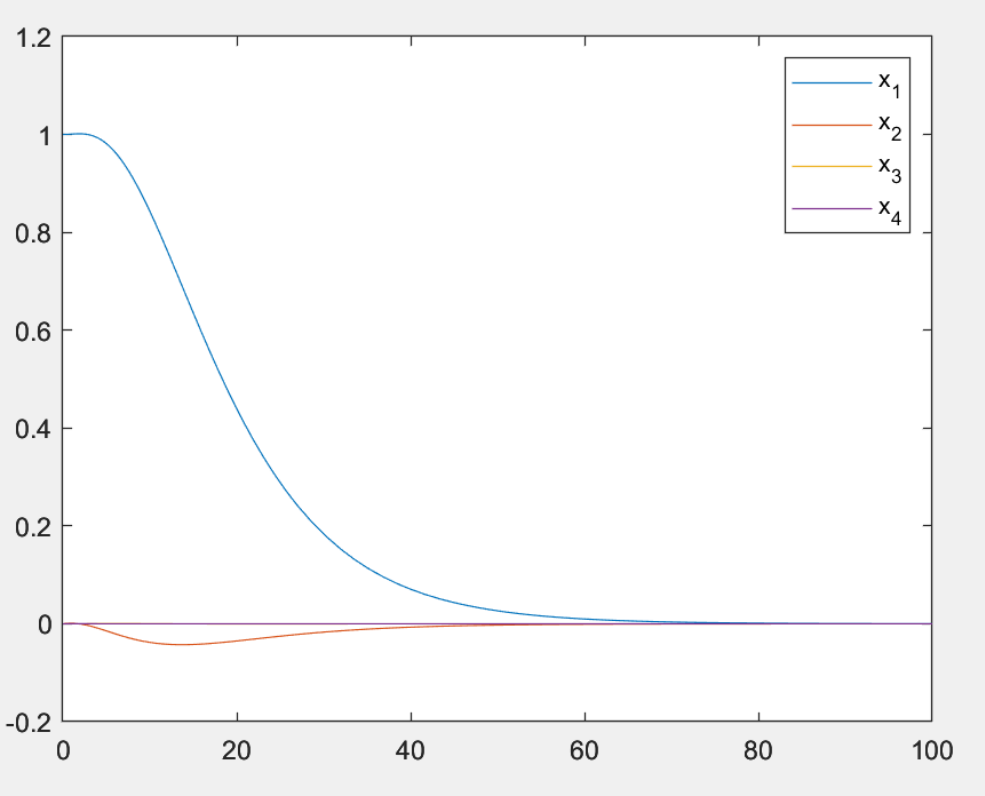


مشاهده میشود این بار به شکل دیگری سیستم کنترل میشود. به ازای بعضی شرایط اولیه نیز به سختی خروجی گرفته میشود. با انتخاب بهتر مقادیر ویژه میتوانیم نتایج بهتری بگیریم که بهترین حالت استفاده از Linear Quadratic Regulator میباشد. هر چه مقادیر ویژه منفی تر باشد زود تر سیستم ما پایدار میشود ولی actuator توان این کار را ممکن است نداشته باشد. و همچنین actuator توانایی ایجاد قطب های کوچک را دارد ولی قطب های کوچک زمان زیادی برای پایدار سازی لازم دارند و در بعضی حالات شاید این توانایی را نداشته باشند.

برای تست تاثیر ضعیف بودن قطب ها بر سیستم از قطب های مطلوب زیر استفاده میکنیم.

```
eigs = [-0.1;-0.2;-0.3;-0.4];
```

در این حالت مشاهده میشود زمانی بالای 60 ثانیه لازم است تا سیستم را پایدار نماید.



ردیابی با پیش جبرانساز استاتیکی

در این قسمت از پیش جبرانساز استاتیکی برای مساله ی ردیابی استفاده مینماییم.

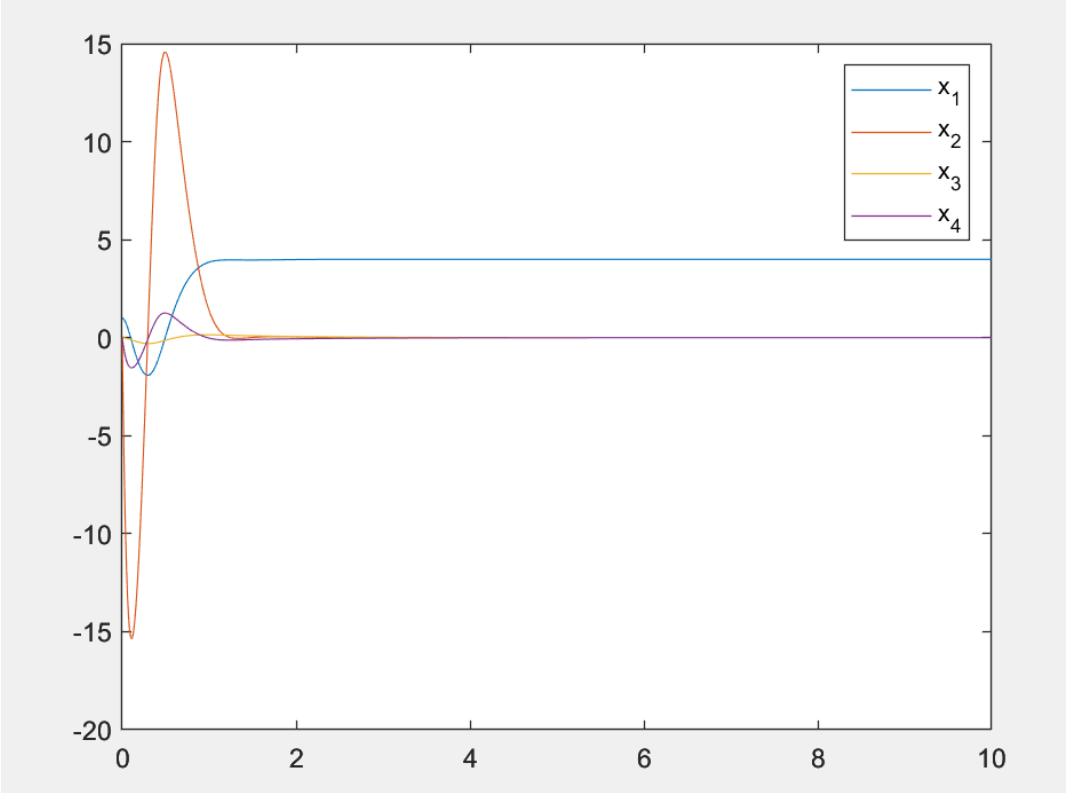
ابتدا GCL را پیاده سازی مینماییم.

```
%GCL
GCLZ=C*inv(B*K-A)*B;
```

حال از آن استفاده مینماییم تا خروجی سیستم خود را به عدد4 برسانیم.

```
%testing on our nonlinear system
tspan=[0 10];
yinit=[1 0 0 0];
r=4;
[t,y] = ode45(@(t,y)bicycle(y,-K*y+r*inv(GCLZ)),tspan,yinit);
plot(t,y(:,1),t,y(:,2),t,y(:,3),t,y(:,4))
xlabel=('time');
ylable=('x3');
legend('x_1','x_2','x_3','x_4');
```

خروجی به صورت زیر در می آید که مورد نظر ما میباشد.



مشاهده میشود در استفاده همزمان از فیدبک حالت و پیش جبرانساز استاتیکی توانستیم در مدت اندکی خروجی سیستم را به مقدار دلخواهمان برسانیم.

ردیابی با فیدبک حالت و کنترلر انتگرالی

برای استفاده از روش کنترلر انتگرالی باید یک متغیر حالت جدید به سیستم اضافه نماییم.

لذا تابع bicycle را تغییر میدهیم. متغیر حالت بعدی همان Q میباشد.

```
function dy=bicycle_integral(y,u)
d2=1;
c1=1;
c2=-1;
c3=1;
d1=0.1;
C=[1 0 0 0];
r=4;
%dy=zeros(4,1);
dy(1,1)=y(2);
dy(2,1)=( 0.5*d2*c1*sin(2*y(3)) + c2*sin(y(3)*y(4)*y(4)) +c3*u  ) / (
dy(3,1)=y(4);
dy(4,1)=( 0.5*d1*c2*sin(2*y(3))*y(4)*y(4) + d1*c3*cos(y(3))*u + d2*si
dy(5,1)=r-C*y(1:4);
```

برای بررسی کنترل پذیری ماتریس افزوده ماتریس زیر را بررسی مینماییم.

```
%Controlability of augmented matrices
rank([B A;0 -C])
```

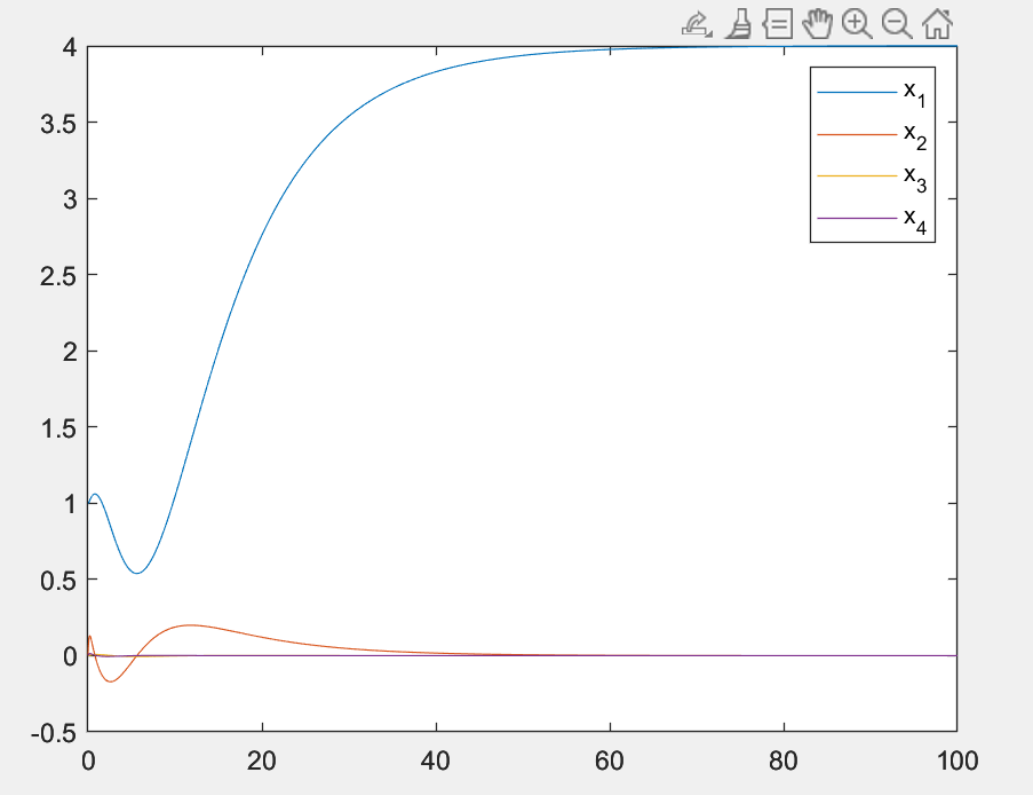
حال K جدیدی را طراحی مینماییم

```
%new matrices
new_A=[A [0;0;0;0];-C 0];
new_B=[B;0];
new_C=[C 0];
%Controlability of augmented matrices
rank([B A;0 -C])
%designing K
eigs = [-.1;-.4;-.5;-.6;-10];
K = place(new_A,new_B,eigs);
eig(new_A-new_B*K)
```

حال برای حل معادله جدید از کد زیر بهره میبریم.

```
%testing on our nonlinear system
tspan=[0 100];
yinit=[1 0 0 0 0];
[t,y] = ode45(@(t,y)bicycle_integral(y,-K*y),tspan,yinit);
  plot(t,y(:,1),t,y(:,2),t,y(:,3),t,y(:,4))
  xlabel=('time');
  ylable=('x3');
  legend('x_1','x_2','x_3','x_4');
```

جواب به صورت زیر بدست می آید. که چون r را برابر 4 قرار دادیم خروجی ما یعنی x1 به 4 همگرا میشود.



Linear Quadratic Regulator

در قسمت جایابی قطب ها مشکل انتخاب قطب های مطلوب را بیان نمودیم. میدانیم با انتخاب قطب های سیستم نزدیک به مبدا سیستم ما در پایدارسازی متغیر ها کند عمل میکند. همچنین اگر قطب ها خیلی بزرگ باشند عملا عملگر های ما توانایی جایابی قطب ها را ندارند. در اینجا نیاز به روشی داریم که بتواند بهترین جایابی در قطب ها را داشته باشیم که هم سیستم پایدار باشد و هم خوب به اهدافمان برسیم.

بدین منظور ما برای پیدا کردن ماتریس K از کد زیر استفاده مینماییم.

```
%designing K
Q=[10 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
R=0.01;
K = lqr(A,B,Q,R);
eig(A-B*K) %new eigen values after using linear quadratic regulator
```

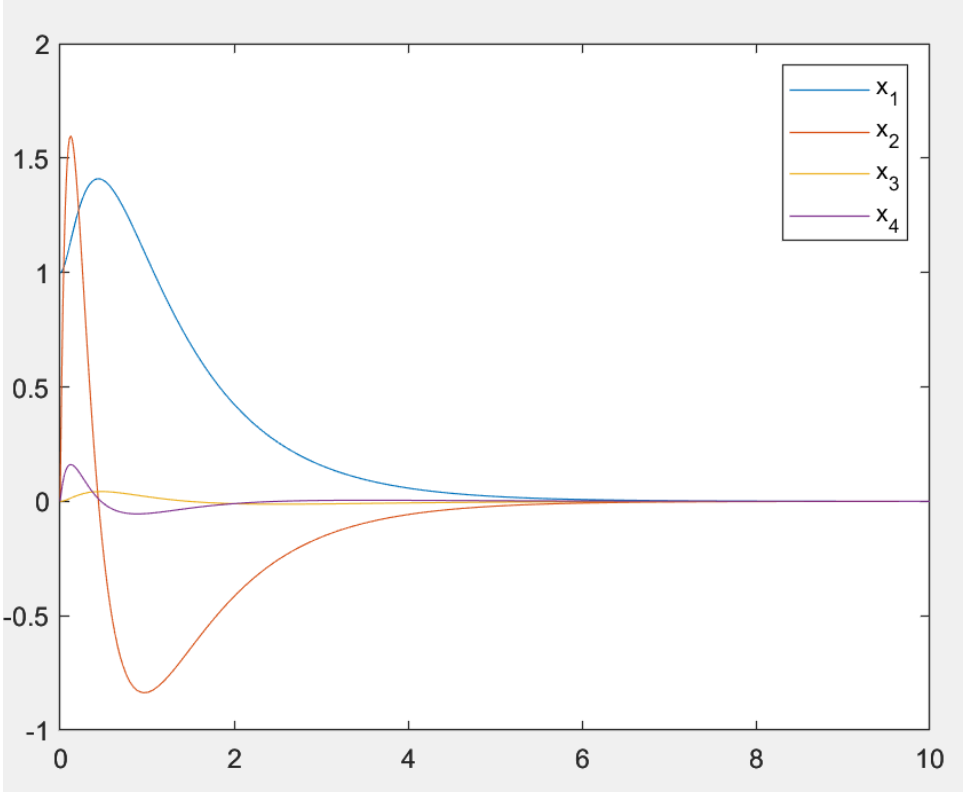
روش LQR قطب های سیستم حلقه بسته ما را به نقاط زیر تغییر میدهد.

```
-10.6821 + 0.0000i
 -3.2871 + 0.0000i
 -1.0003 + 0.0016i
 -1.0003 - 0.0016i
```

برای ماتریس Q از ماتریس زیر استفاده مینماییم زیرا کنترل x1 برای ما از بقیه متغیر ها اهمیت بالاتری دارد.

```
10     0     0     0
 0     1     0     0
 0     0     1     0
 0     0     0     1
```

بدین ترتیب خروجی سیستم حلقه بسته با جایابی قطب ها به روش LQR به صورت زیر در می آید.



این خروجی نه خیلی کند است و نه خیلی سریع و بهترین روش برای طراحی و جایابی قطب ها میباشد.

طراحی سیستم تخمین گر

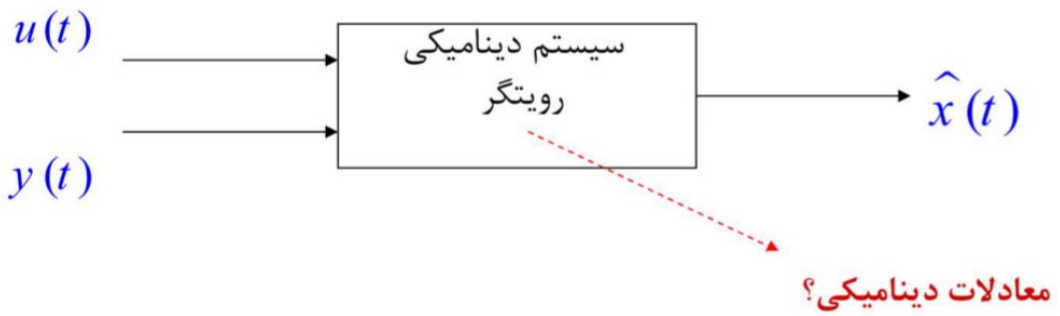
همان طور که میدانید در بسیاری از سیستم های واقعی فیدبک گرفتن از تمام متغیر های حالت به شدت هزینه بر میباشد و یا در بسیاری از موارد که ویژگی مورد نظر غیر فیزیکی میباشد این کار امکان پذیر نمیباشد. لذا نیاز به روشی داریم تا این متغیر ها را از خروجی های قابل اندازه گیری سیستم تخمین بزنیم.

قبل از هر چیز شرط طراحی تخمینگر رویت پذیری سیستم میباشد که آنرا بررسی مینماییم.

```
%observability
rank(obsv(A,C))
```

که خروجی آن به صورت زیر است که نشان دهنده ی رویت پذیری سیستم ما میباشد.

```
ans =

     4
```



$$\dot{\hat{x}}(t) = \hat{A}\hat{x}(t) + \hat{B}u(t) + Ly(t)$$

ماتریس های Ahat Bhat L را به گونه ای میابیم که خطای سیستم تخمینگر به صورت مجانبی صفر گردد.

$$\hat{A} = A - LC$$
$$\hat{B} = B$$

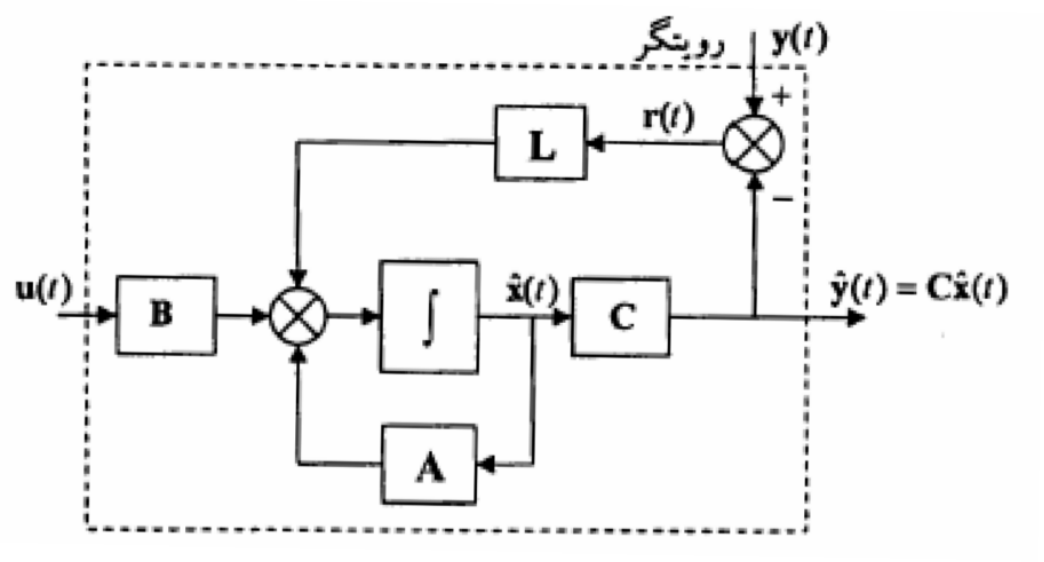
ماتریس L را چنان میابیم که مقادیر ویژه Ahat سمت چپ قرار گیرند.

$$\dot{\hat{x}}(t) = (A - LC)\hat{x}(t) + \begin{bmatrix} B & L \end{bmatrix} \begin{bmatrix} u(t) \\ y(t) \end{bmatrix}$$

معادلات دینامیکی رویتگر

or

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L[y(t) - C\hat{x}(t)]$$



حال L را به گونه ای حساب میکنیم که قطب های A-LC را به مقادیر 4- 3- 2- 1- ببرد

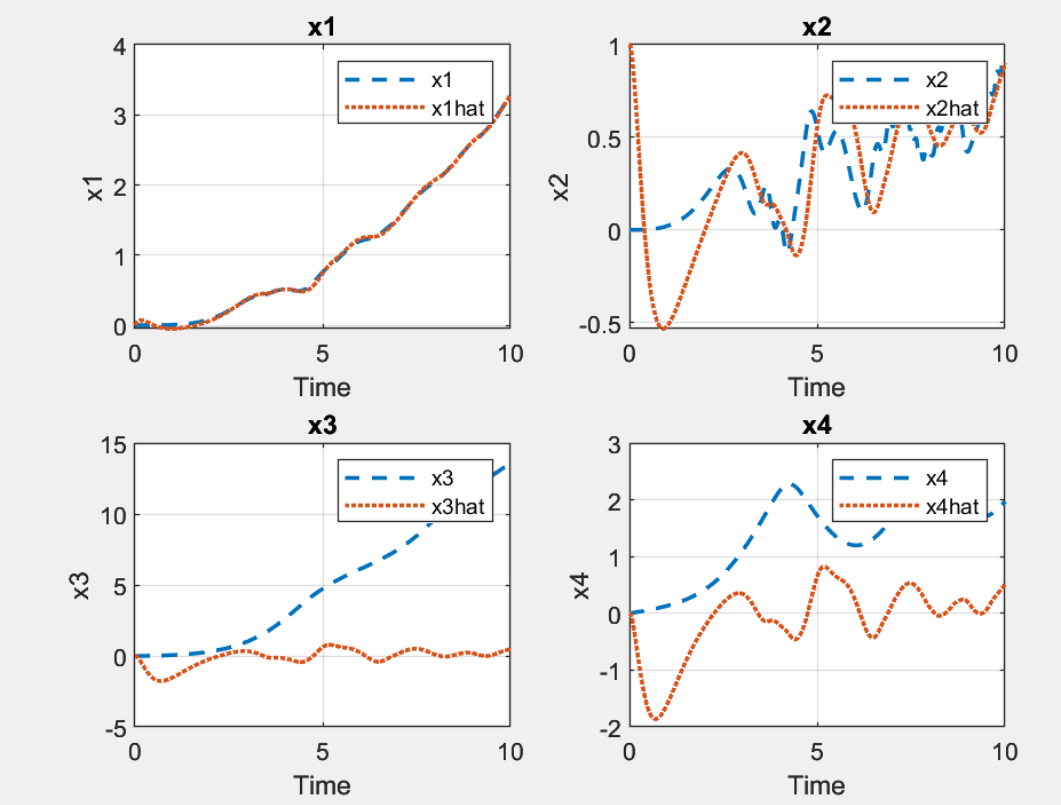
```
%designing L
observer_eigs=[-1 -2 -3 -4];
L=(place(A',C',observer_eigs))';
```

L به صورت زیر محاصبه میگردد .

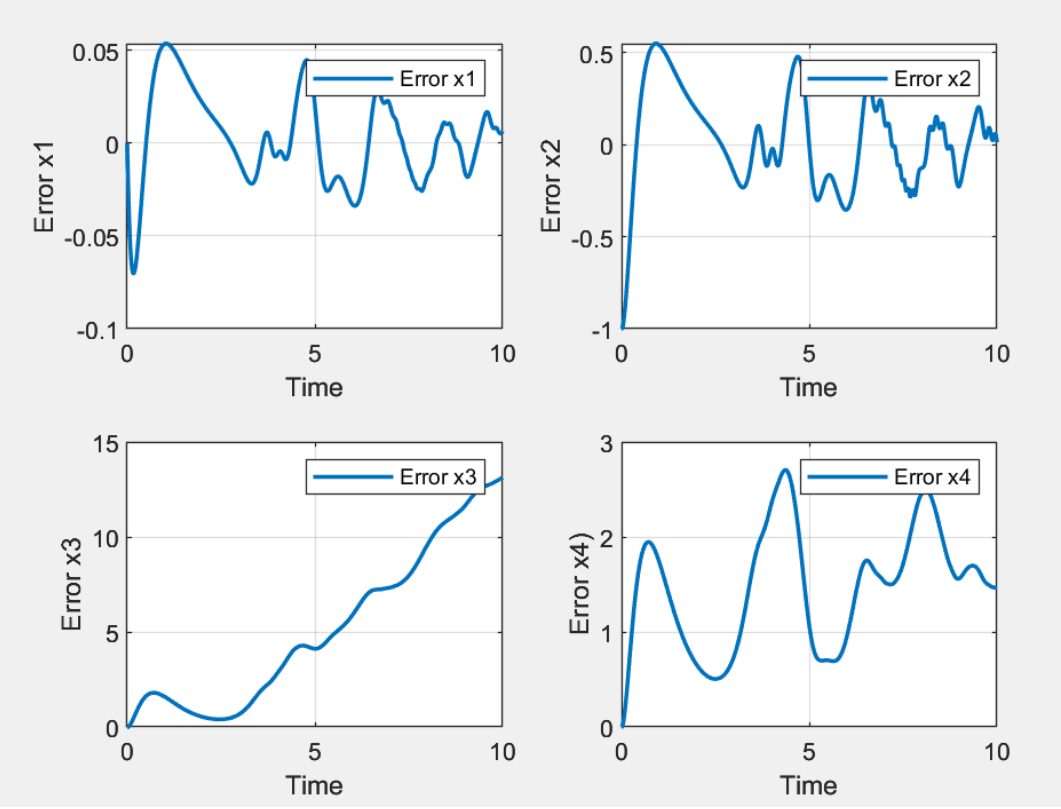
```
L =

   10.0000
   36.1111
   55.0000
   57.7111
```

حال باید رویتگر خود را طراحی نماییم.

برای پیاده سازی این قسمت از for loop استفاده مینماییم یکی برای محاسبه ی x معادله اول و دیگری برای xhat نتایج به صورت زیر حاصل میگردد.



نمودار error نیز به صورت زیر حاصل میگردد.



برای حل معادله اصلی از کد زیر بهره میگیریم

```
for i=2:length(t)
    u(i)=0;

x1(i)=dt*x2(i-1) +x1(i-1);
x2(i)=dt*((0.5*d2*c1)*sin(2*x3(i-1)) + c2*sin(x3(i-1)*x4(i-1)*x4(i-1))) /(1-d1*d2*cos(x3(i-1))*cos(x3(i-1)))+ x2(i-1);
x3(i)=dt*x4(i-1) +x3(i-1);
x4(i)=dt*(0.5*d1*c2*sin(2*x3(i-1))*x4(i-1)*x4(i-1) + d1*c3*cos(x3(i-1)*u(i)) +d2*sin(x3(i-1)))/(1-d1*c3*(cos(x3(i-1))^2))

    x(:,i)=[x1(i);x2(i);x3(i);x4(i)];
end

y=C*[x1;x2;x3;x4];
```

همچنین برای حل معادله مربوط به رویتگر از کد زیر استفاده میکنیم.

```
for i=2:length(t)
    u(i)=0;
    U_1(:,i)=[u(i);y(i)];
    x1_hat (i) = x1_hat(i-1)+ dt*(A_1(1,:)*x_hat(:,i-1)+B_1(1,:)*U_1(:,i-1));
    x2_hat (i) = x2_hat(i-1)+ dt*(A_1(2,:)*x_hat(:,i-1)+B_1(2,:)*U_1(:,i-1));
    x3_hat (i) = x3_hat(i-1)+ dt*(A_1(3,:)*x_hat(:,i-1)+B_1(3,:)*U_1(:,i-1));
    x4_hat (i) = x4_hat(i-1)+ dt*(A_1(4,:)*x_hat(:,i-1)+B_1(4,:)*U_1(:,i-1));
    x_hat(:,i)=[x1_hat(i);x2_hat(i);x3_hat(i);x4_hat(i)];
end
```

ارور را نیز با کد زیر می یابیم

```
e=[x1-x1_hat;x2-x2_hat;x3-x3_hat;x4-x4_hat];
N1_2=norm(e(1,:),2);
N2_2=norm(e(2,:),2);
N3_2=norm(e(3,:),2);
N4_2=norm(e(4,:),2);

N1_INF=norm(e(1,:),inf);
N2_INF=norm(e(2,:),inf);
N3_INF=norm(e(3,:),inf);
N4_INF=norm(e(4,:),inf);
```

رویتگر کاهش یافته لیونبرگر

برای رویت گر کاهش یافته C را در سیستم تغییر میدهیم تا سیستم دو خروجی شود. میتوان خروجی دوم را x3 قرار دهیم.کد های ما به صورت زیر میباشند.

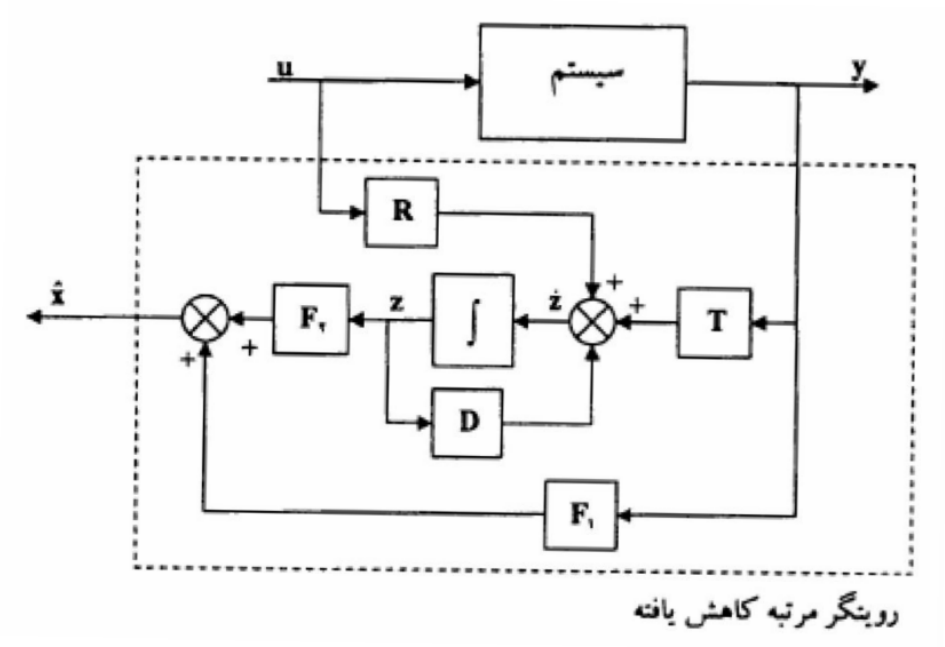
```
n=rank(ctrb(A,B));
l=rank(C);

D=[-7 0;0 -10];
Q=[0 0 1 0;1 1 1 1;0 0 0 1;1 0 0 1];

A_t=inv(Q)*A*Q;

A_t_11=A_t(1:(n-l),1:(n-l));
A_t_12=A_t(1:(n-l),(n-l)+1:n);
A_t_21=A_t((n-l)+1:n,1:(n-l));
A_t_22=A_t((n-l)+1:n,(n-l)+1:n);

L_t=(D-A_t_11)*inv(A_t_21);
L=[eye(n-l) L_t]*inv(Q);
T=A_t_12+L_t*A_t_22-D*L_t;
R=L*B;
```

```
z1(1)=0;
z2(1)=0;
z(:,1)=[z1(1);z2(1)];
y(:,1)=[x1(1);x3(1)];
for i=2:length(t)
    u(i)=0;
    z1(i)=z1(i-1)+ dt*(D(1,:)*z(:,i-1)+T(1,:)*y(:,i-1)+R(1)*u(i-1));
    z2(i)=z2(i-1)+ dt*(D(2,:)*z(:,i-1)+T(2,:)*y(:,i-1)+R(2)*u(i-1));
    z(:,i)=[z1(i);z2(i)];
    y(:,i)=[x1(i);x3(i)];
end

z=[z1;z2];
x_hat=inv([C;L])*[y;z];
```

ارور نیز به صورت زیرمحاسبه میشود.

```
e=[x1-x_hat(1,:);x2-x_hat(2,:);x3-x_hat(3,:);x4-x_hat(4,:)];

N2_2=norm(e(2,:),2);
N4_2=norm(e(4,:),2);
N2_INF=norm(e(2,:),inf);
N4_INF=norm(e(4,:),inf);
```

بلوک دیاگرام رویتگر این بار در شکل زیر آمده است.



$$\begin{bmatrix} C \\ L \end{bmatrix}^{-1} = \begin{bmatrix} F_1 & F_2 \end{bmatrix}$$

$$\hat{x}(t) = \begin{bmatrix} F_1 & F_2 \end{bmatrix} \begin{bmatrix} y(t) \\ z(t) \end{bmatrix}$$

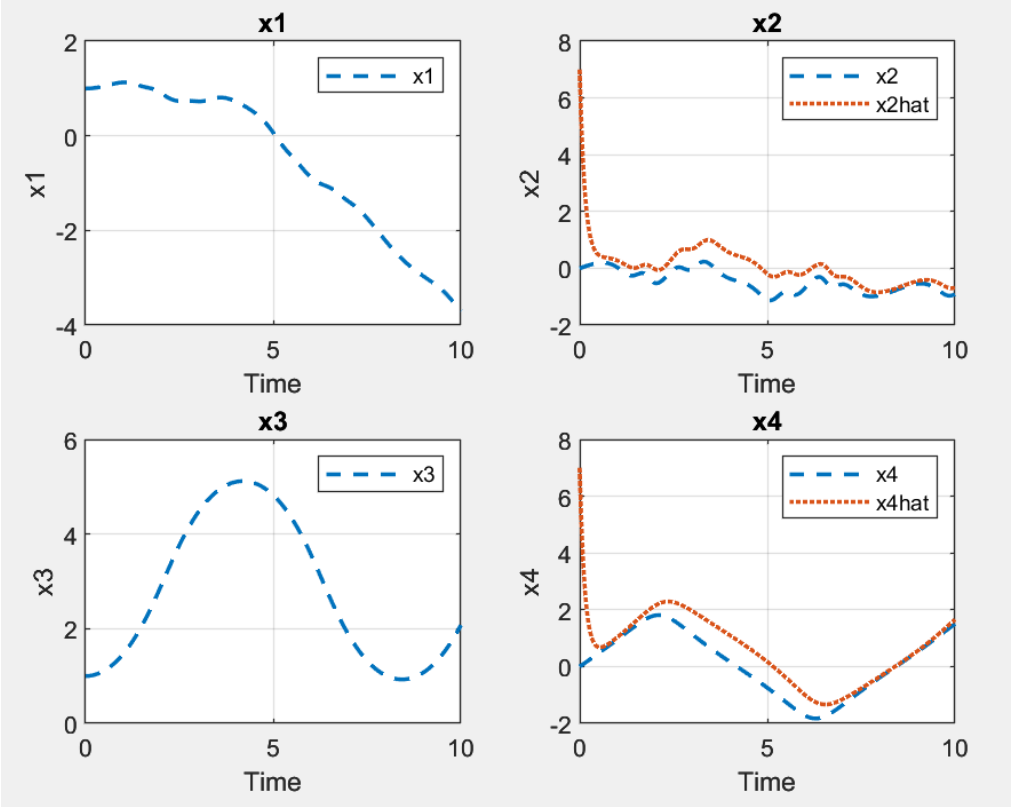
$$\dot{z}(t) = Dz(t) + Ru(t) + Ty(t)$$

$$LA - DL = TC$$

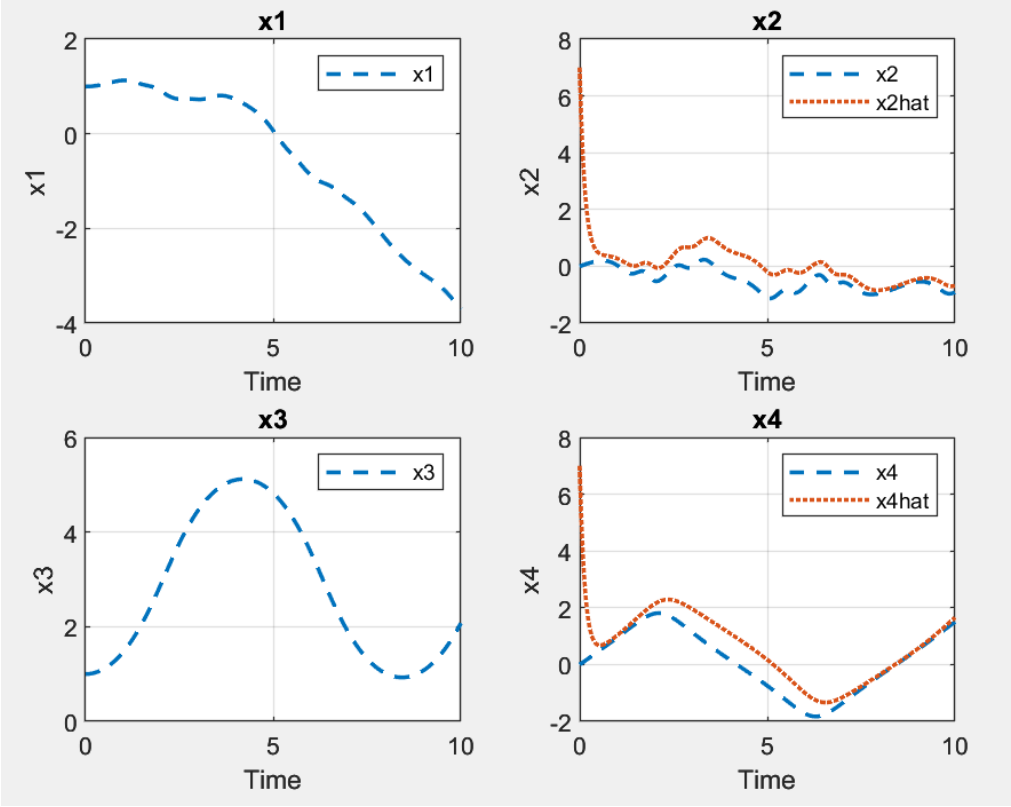
$$R = LB$$

خروجی های رویتگر و خروجی های واقعی با هم در شکل زیر رسم شده اند

.



و ارور های موجود نیز به شکل زیر رسم شده اند.



استفاده از رویتگر مرتبه کامل لیونبرگر در کنترل فیدبک حالت

در این قسمت از تابع ode استفاده میکنیم. ولی معادله دیفرانسیل (فضای حالت) جدید تعریف میکنیم که در بردارنده ی متغیر های x1 x2 x3 x4 x1hat x2hat x3hat x4hatباشد و آن را حل مینماییم.

```
function dy=Bicycle_With_Full_Observer(y,u)
d2=1;
c1=1;
c2=-1;
c3=1;
d1=0.1;
A=readmatrix('A_barr');
B=readmatrix('B_barr');
C = [1 0 0 0];
D=0;
%
A_0=A';
B_1=C';
f=[-1 -2 -3 -4];
L=(acker(A_0,B_1,f))';
A_1=A-L*C;
B_1=[B L];
%dy=zeros(4,1);
dy(1,1)=y(2);
dy(2,1)=( 0.5*d2*c1*sin(2*y(3)) + c2*sin(y(3)*y(4)*y(4)) +c3*u  ) / (  1-d1*c1*cos(y(3))*cos(y(3))
dy(3,1)=y(4);
dy(4,1)=( 0.5*d1*c2*sin(2*y(3))*y(4)*y(4) + d1*c3*cos(y(3))*u + d2*sin(y(3))  )/ ( 1-d1*c1*cos(y(3))
dy(5:8,1)=A_1*y(1:4) + B_1*[u;C*y(1:4)];
```

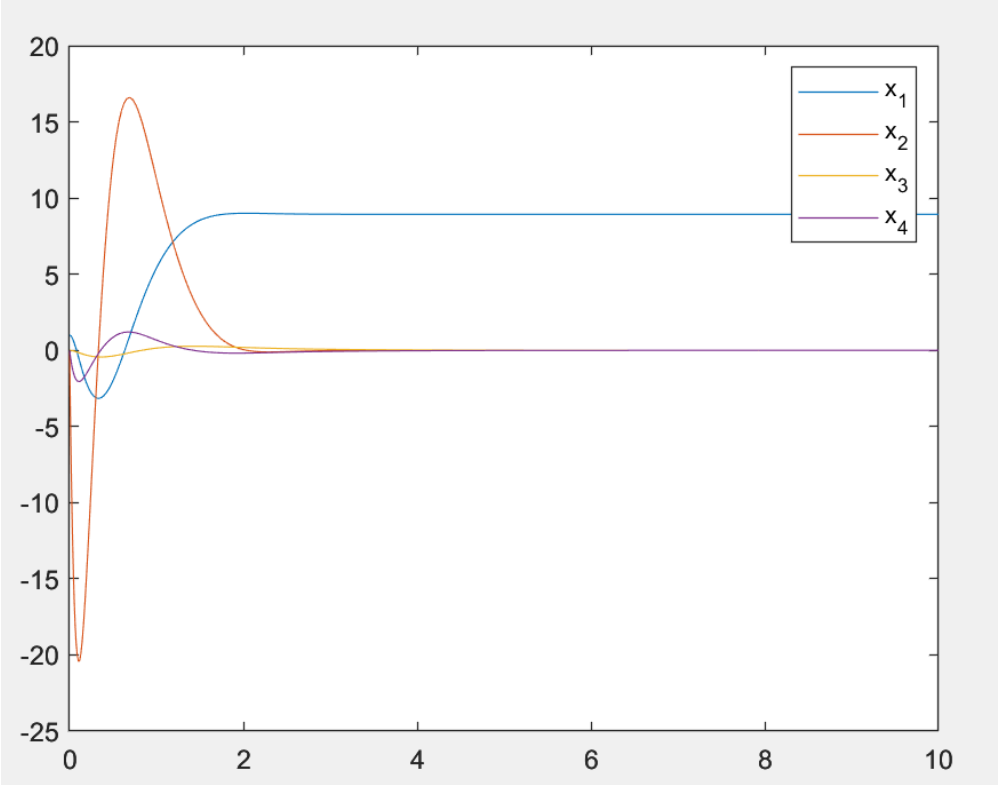
مشاهده میشود 4 ستون آخر آن اضافه معادله دیفرانسیل تخمینگر به معادله ی اصلی است.

از این تابع در جدید در روش قبل استفاده میکنیم

```
%Designing Full State Estimator!
clear;
clc;
A=readmatrix('A_barr');
B=readmatrix('B_barr');
C = [1 0 0 0];
D=0;
%controlability
rank(ctrb(A,B)) %because our system is controlable we can design state feed_back contr
%observability
rank(obsv(A,C)) %because our system is observable we can design state estimator!
%eigen values
eig(A)
%designing K
eigs = [-1;-4;-5;-6];
K = place (A,B,eigs);
eig(A-B*K)
%GCL
GCLZ=(C/(B*K-A))*B;%using A/B instead of A*inv(B) is really faster way!
%designing L
observer_eigs=[-1 -2 -3 -4];
L=(place(A',C',observer_eigs))';
%kallman filter-> kf=(lqr(A',C',Vd,Vn))';       Vd=0.1*eye(4);    Vn=1;
%testing on our nonlinear system
tspan=[0 10];
yinit=[1 0 0 0 0 0 0 0];
r=4;
[t,y] = ode45(@(t,y)Bicycle_With_Full_Observer(y,-K*y(5:8)+r*inv(GCLZ)),tspan,yinit);
```

مشاهده میکنید اینبار به جای k*y از k*y(5:8) که همان y*xhat میباشد استفاده نمودیم

نتایج به صورت زیر در می آید



مشاهده میشود که این بار static compensator درست عمل نمیکند.