چند مسئله از مهندسی کنترل در متلب و سیمولینک

برای رشته‌ی مهندسی مکانیک

ترجمه و تدوین

پویان نیّری

آرمان محمدی

فهرست مطالب

فصل اول: مفاهیم پایه

بخش اول: مفاهیم پایه‌ای متلب

فهرست مطالب بخش

- بردارها
- توابع
- رسم نمودار
- چندجمله‌ای‌ها تحت عنوان بردارها
- استفاده از متغیر s در چندجمله‌ای‌ها
- ماتریس‌ها
- چاپ
- استفاده از ام‌فایل در متلب
- بخش راهنمای متلب

متلب یک نرم‌افزار قدرتمند جهت انجام محاسبات عددی و پردازش داده‌ها است که امروزه در مهندسی کنترل، جهت طراحی و پردازش سیستم‌ها به وفور مورد استفاده قرار می‌گیرد. این نرم‌افزار، جعبه‌ابزارهای مختلفی را جهت کاربردهای گوناگون ارائه می‌نماید که در این آموزش، هدف ما بررسی جعبه‌ابزار کنترل متلب است. نرم‌افزار متلب بر روی سیستم عامل‌های یونیکس، مکینتاش و ویندوز قابل استفاده است، همچنین نسخه‌ی دانشجویی این نرم‌افزار جهت استفاده بر روی کامپیوترهای شخصی موجود است. جهت کسب اطلاعات بیشتر به وبسایت www.mathworks.com مراجعه نمایید.

دستورهای کلیدی متلب در این بخش:

```
plot , polyval , roots , conv , deconv , inv , eig , poly , tf , zero
```

بردارها

اجازه دهید این بخش را با یک مبحث ساده مثل بردار شروع کنیم. درایه‌های بردار را به ترتیب وارد کنید (بین دو براکت) و مساوی با متغیر مورد نظر قرار دهید:

```
a = [1 2 3 4 5 6 9 8 7]
```

```
a =

     1     2     3     4     5     6     9     8     7
```

حال یک بردار با مقادیر بین ۰ تا ۲۰ با فواصل دو تایی تشکیل می‌دهیم (این روش جهت تشکیل بردار زمان، بسیار مورد استفاده قرار می‌گیرد):

```
t = 0:2:20
```

```
t =

     0     2     4     6     8    10    12    14    16    18    20
```

تغییر در مقادیر درایه‌های یک بردار نیز کار ساده ایست. فرض کنید می‌خواهیم به درایه‌های یک بردار عدد ۲ را اضافه کنیم:

```
b = a + 2
```

```
b =

     3     4     5     6     7     8    11    10     9
```

حال اگر بخواهیم درایه‌های دو بردار هم‌اندازه را با یکدیگر جمع کنیم به سادگی داریم:

```
c = a + b
```

```
c =

     4     6     8    10    12    14    20    18    16
```

تفریق مقادیر دو بردار هم‌اندازه نیز به همین صورت انجام می‌گردد.

توابع

جهت سادگی کار، متلب توابع استاندارد زیادی را در بر دارد. هر تابع مجموعه‌ای از کدهاست که برای انجام وظیفه‌ی معینی استفاده می‌گردد. توابع sin, cos, log, exp, sqrt در متلب بسیار پر استفاده‌اند؛ همینطور pi تحت عنوان عدد پی، i و j تحت عنوان ریشه‌ی دوم عدد ۱- در متلب مورد استفاده قرار گرفته است.

```
sin(pi/4)
```

```
ans =

    0.7071
```

جهت بررسی نحوه بکارگیری هر تابع در متلب، عبارت `help [function name]` را در قسمت پنجره‌ی دستور متلب تایپ کنید.

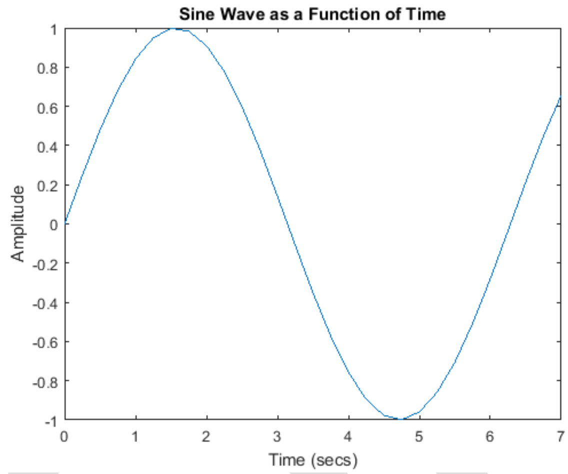
رسم نمودار

رسم منحنی در متلب کار بسیار راحتی است. فرض کنید می‌خواهید یک موج سینوسی را به صورت تابعی از زمان رسم کنید. ابتدا یک بردار زمان تولید کنید (گذاشتن نقطه ویرگول[1] در انتهای هر خط بدین معنی است که ما نمی‌خواهیم عبارت تولید شده نمایش داده شود) سپس مقدار عبارت را در هر زمان به دست آورید. دستوراتی که بعد از تابع `plot` آمده است در واقع برای نام‌گذاری نمودار و محورها در منحنی ایجاد شده می‌باشد.

```
t = 0:0.25:7;

y = sin(t);
```

[1] Semicolon

```
plot(t,y)

title('Sine Wave as a Function of Time')
xlabel('Time (secs)')
ylabel('Amplitude')
```



منحنی رسم شده تقریبا یک تناوب از یک موج سینوسی را رسم کرده است.

چندجمله‌ای‌ها تحت عنوان بردارها

در متلب چندجمله‌ای‌ها تحت عنوان بردار نمایش داده می‌شوند. این کار تنها با وارد کردن ضرایب چندجمله‌ای‌ها به ترتیب زیاد به کم انجام می‌شود. برای عنوان مثال چند جمله‌ای زیر را در نظر بگیرید:

$$s^4 + 3s^3 - 15s^2 - 2s + 9 \qquad (۱)$$

جهت ورود عبارت بالا در متلب به صورت زیر عمل می‌کنیم:

```
x = [1 3 -15 -2 9]
```

```
x =

     1     3   -15    -2     9
```

اگر هر کدام از مقادیر چندجمله‌ای وجود نداشت مقدار صفر را در بردار برای آن در نظر می‌گیریم، برای مثال:

$$s^4 + 1 \qquad (۲)$$

جهت وارد کردن عبارت بالا در متلب به صورت زیر عمل می‌کنیم:

```
y = [1 0 0 0 1]
```

```
y =

     1     0     0     0     1
```

همچنین شما می‌توانید مقدار یک چندجمله‌ای را در ازای عدد خاصی محاسبه کنید، مثلا در عبارت بالا حاصل چندجمله‌ای را در ازای s=2 محاسبه می‌کنیم:

```
z = polyval([1 0 0 0 1],2)
```

```
z =

    17
```

همچنین شما می‌توانید ریشه‌های یک چند جمله‌ای را استخراج کنید.

$$s^4 + 3s^3 - 15s^2 + 9 \quad (۳)$$

برای محاسبه ریشه‌های عبارت بالا از دستور زیر استفاده می‌کنیم:

```
roots([1 3 -15 -2 9])
```

```
ans =

   -5.5745

    2.5836

   -0.7951

    0.7860
```

حال فرض کنید که می‌خواهیم ضرب دو چندجمله‌ای را حساب کنیم. حاصل ضرب دو چندجمله‌ای معادل کانولوشن[۲] ضرایب آنهاست. دستور conv در متلب این کار را برای ما انجام می‌دهد:

```
x = [1 2];

y = [1 4 8];

z = conv(x,y)
```

```
z =

     1     6    16    16
```

تقسیم دو چندجمله‌ای نیز به همین سادگی صورت می‌گیرد. دستور deconv باقی‌مانده و خارج قسمت را نمایش می‌دهد. حال z را بر y تقسیم می‌کنیم تا x را به دست آوریم:

```
[xx, R] = deconv(z,y)
```

```
xx =
```

[۲] Convolution

```
     1     2

R =

     0     0     0     0
```

همانطور که مشاهده می‌کنید این همان مقدار x قبل می‌باشد. اگر y بر z بخش‌پذیر نبود، بردار باقی‌مانده غیر صفر به دست می‌آمد.

استفاده از متغیر s در چندجمله‌ای‌ها

نحوه‌ی دیگر نمایش چندجمله‌ای ها در متلب، استفاده از متغیر لاپلاس s می‌باشد. از این روش به طور گسترده در این کتاب استفاده خواهد شد. هم اکنون وارد جزییات مفهوم دامنه‌ی لاپلاس نمی‌شویم و تنها چندجمله‌ای را با متغیر s نمایش می‌دهیم. برای تعریف یک متغیر، دستور متلب زیر را در پنجره‌ی دستور متلب وارد کنید:

```
s = tf('s')
```

```
s =
 
 
  s
```

برای یادآوری چندجمله‌ای بالا برابر بود با:

$$s^4 + 3s^3 - 15s^2 - 2s + 9 \qquad (۴)$$

برای نمایش این چندجمله‌ای در متلب، دستور زیر را اجرا کنید:

```
polynomial = s^4 + 3*s^3 - 15*s^2 - 2*s + 9
```

```
polynomial =
 
 
  s^4 + 3 s^3 - 15 s^2 - 2 s + 9
```

در این حالت بجای استفاده از تابع `roots` جهت محاسبه ریشه‌های چندجمله‌ای، از دستور `zero` استفاده می‌کنیم:

```
zero(polynomial)
```

```
ans =

   -5.5745

    2.5836

   -0.7951

    0.7860
```

همانگونه که مشاهده می‌کنید نتایج همانند حالت قبل است.

همینطور عملیات ضرب چندجمله‌ای توسط متغیر s نیز ممکن است. x و y را دوباره تعریف می‌نماییم:

```
x = s + 2;

y = s^2 + 4*s + 8;

z = x * y
```

```
z =


  s^3 + 6 s^2 + 16 s + 16
```

همانگونه که مشاهده می‌کنید نتایج همانند استفاده از تابع `conv` در قسمت قبل است.

ماتریس‌ها

وارد کردن یک ماتریس در متلب نیز همانند وارد کردن بردار است با این تفاوت که هر سطر از ماتریس با نقطه ویرگول (;) یا کلید enter جدا می‌شود:

```
B = [1 2 3 4; 5 6 7 8; 9 10 11 12]



B = [ 1   2   3   4

      5   6   7   8

      9  10  11  12 ]
```

```
B =

     1     2     3     4

     5     6     7     8

     9    10    11    12

B =

     1     2     3     4

     5     6     7     8

     9    10    11    12
```

تغییر در مقادیر ماتریس نیز به اشکال مختلف انجام می‌گیرد. برای مثال، برای به دست آوردن ترانهاده‌ی[۳] یک ماتریس از علامت اپوستروف (′) استفاده می‌شود:

```
C = B'
```

```
C =

     1     5     9

     2     6    10

     3     7    11

     4     8    12
```

لازم به ذکر است که اگر C یک ماتریس مختلط بود، علامت اپوستروف، ماتریس ترانهاده‌ی مزدوج مختلط را نتیجه می‌داد. برای به دست آوردن ماتریس ترانهاده در این موارد، از علامت ′. استفاده می‌نماییم (اگر ماتریس مختلط نباشد این دو دستور با یکدیگر یکسان خواهند بود).

حال شما می‌توانید دو ماتریس B و C را در یکدیگر ضرب کنید. توجه کنید که ترتیب ضرب کردن دو ماتریس اهمیت دارد:

```
D = B * C


D = C * B
```

```
D =

    30    70   110

    70   174   278

   110   278   446

D =

   107   122   137   152

   122   140   158   176

   137   158   179   200

   152   176   200   224
```

گزینه‌ی دیگر برای ضرب دو ماتریس، ضرب نظیر به نظیر درایه‌های دو ماتریس توسط عملگر *. می‌باشد (دو ماتریس باید هم‌اندازه باشند):

[۳] Transpose

```
E = [1 2; 3 4]

F = [2 3; 4 5]

G = E .* F
```

```
E =

     1     2

     3     4

F =

     2     3

     4     5

G =

     2     6

    12    20
```

اگر ماتریس مورد نظر مانند ماتریس E، مربعی باشد می‌توانید با به توان رساندن آن، ماتریس را چندبار در خودش ضرب نمایید:

```
E^3
```

```
ans =

    37    54

    81   118
```

اگر قصد مکعب کردن هر درایه‌ی یک ماتریس را دارید، از کعب درایه به درایه استفاده کنید:

```
E.^3
```

```
ans =

     1     8

    27    64
```

همچنین می‌توانید معکوس یک ماتریس را محاسبه کنید:

```
X = inv(E)
```

```
X =
```

```
   -2.0000    1.0000
    1.5000   -0.5000
```

یا مقادیر ویژه‌ی آن را محاسبه کنید:

```
eig(E)
```

```
ans =
   -0.3723
    5.3723
```

حتی در متلب یک تابع جهت به دست آوردن ضرایب چندجمله‌ای مشخصه‌ی یک ماتریس وجود دارد. تابع poly برداری را تشکیل می‌دهد که ضرایب چندجمله‌ای مشخصه را شامل می‌شود:

```
p = poly(E)
```

```
p =
    1.0000   -5.0000   -2.0000
```

به یاد داشته باشید که مقادیر ویژه‌ی یک ماتریس معادل ریشه‌های چندجمله‌ای مشخصه‌ی آن است:

```
roots(p)
```

```
ans =
    5.3723
   -0.3723
```

استفاده از اِم‌فایل در متلب

برای کارکردن با ام‌فایل‌ها، تفاوتی جزئی در هر سیستم‌عامل وجود دارد:

مکینتاش

- برای ویرایش ام‌فایل‌ها یک ویرایشگر در درون متلب وجود دارد. از منوی **File** گزینه‌ی **New M-file** را انتخاب کنید. همچنین می‌توانید از هر ویرایشگر دیگر نیز استفاده نمایید.

ویندوز

- اجرای متلب در ویندوز بسیار مشابه اجرای متلب در مکینتاش می‌باشد. البته لازم است که بدانید ام‌فایل شما در کلیپ‌بورد ذخیره می‌شود بنابراین لازم است تا حتما آنرا به فرمت filename.m بر روی کامپیوتر خود ذخیره کنید.

یونیکس

- لازم است تا ویرایشگر را به طور مجزا از متلب باز کنید. بهترین روش، درست کردن یک شاخه برای تمام ام‌فایل‌های خود و سپس تغییر مسیر به این شاخه قبل از اجرای متلب و ویرایشگر می‌باشد. برای اجرای متلب از پنجره‌ی Xterm خود، تنها دستور matlab را تایپ کنید.

راهنما در متلب

نرم‌افزار متلب از یک راهنمای آنلاین بسیار خوبی برخوردار است. عبارت روبرو را جهت به دست آوردن اطلاعات بیشتر در رابطه با هر دستوری، تایپ کنید: `help [commandname]`

در این صورت لازم است که اسم دستور مورد نظر را بدانید. یک لیست از تمامی دستورات استفاده شده در این کتاب در **پیوست اول: لیست دستورات متلب** آورده شده است.

هم اکنون به ذکر چند نکته‌ی نهایی برای این بخش می‌پردازیم.

شما می‌توانید در هر لحظه به مقدار یک متغیر با تایپ نام آن دسترسی پیدا کنید:

```
B
```

```
B =

     1     2     3     4

     5     6     7     8

     9    10    11    12
```

همچنین شما می‌توانید در هر خط تا زمانی که دستورات را با نقطه ویرگول یا ویرگول جدا نمایید، بیشتر از یک دستور استفاده کنید.

همچنین ممکن است متوجه شده باشید که اگر مقداری را در متغیری ذخیره نکنید، نرم‌افزار متلب آنرا در متغیر موقتی با نام ans ذخیره می‌کند.

بخش دوم: مفاهیم پایه‌ای سیمولینک

فهرست مطالب بخش

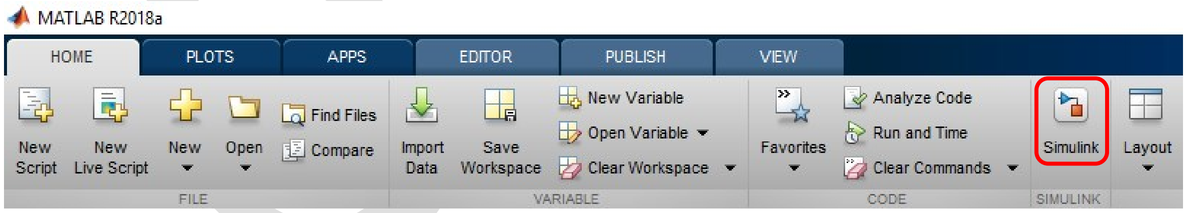
- راه‌اندازی سیمولینک
- فایل‌های مدل
- المان‌های پایه
- مثال‌های ساده
- شبیه‌سازی
- تشکیل سیستم

سیمولینک یک محیط گرافیکی متعلق به نرم‌افزار متلب است که جهت مدل‌سازی و شبیه‌سازی سیستم‌ها به کار می‌رود. یکی از مزایای اساسی نرم‌افزار سیمولینک توانایی شبیه‌سازی سیستم‌های غیرخطی است که توابع تبدیل قادر به این کار نمی‌باشند. مزیت دیگر سیمولینک دریافت مقدار اولیه است، در حالتی که در تابع تبدیل، مقدار اولیه صفر در نظر گرفته می‌شود.

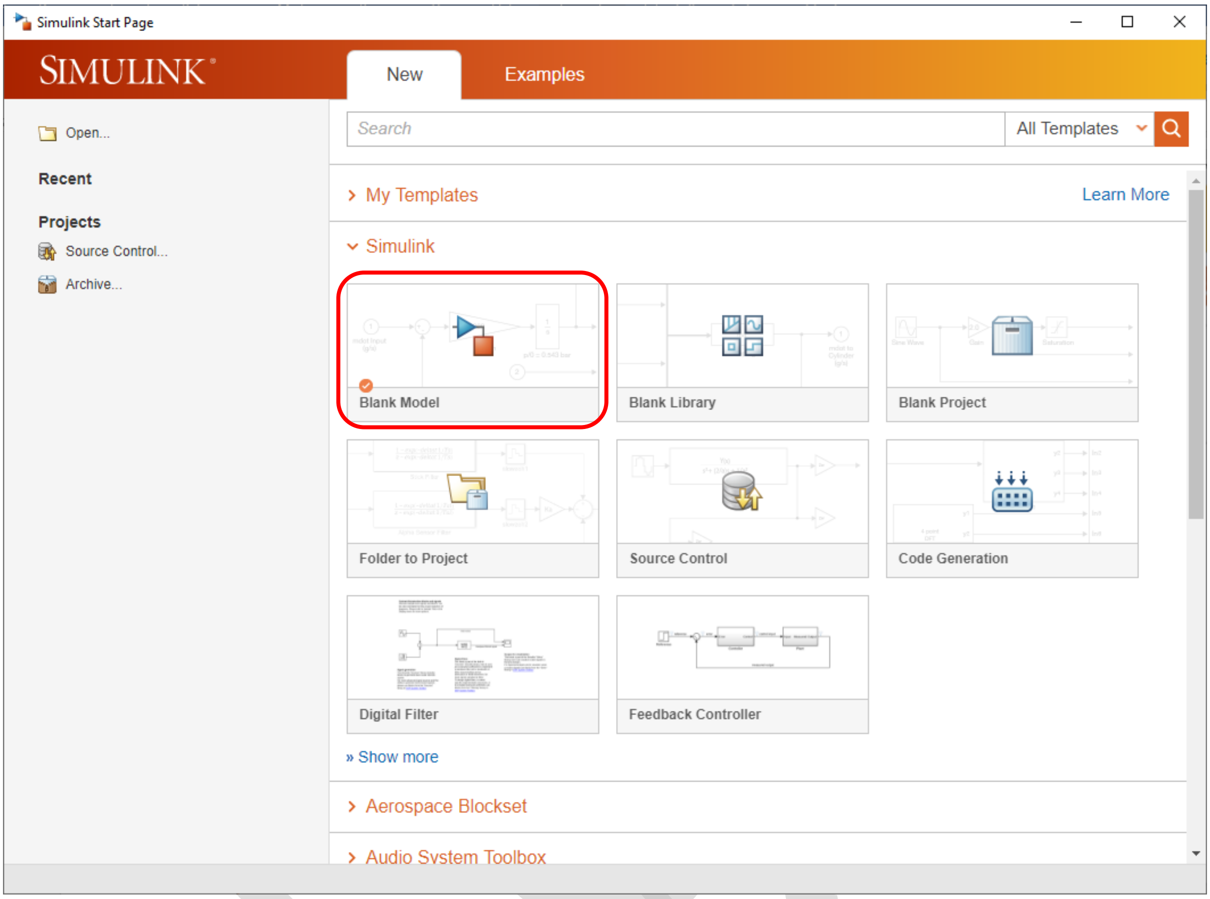
در سیمولینک سیستم‌ها توسط دیاگرام‌های بلوکی نمایش داده می‌شوند. بسیاری از المان‌ها در قالب دیاگرام‌های بلوکی از جمله توابع تبدیل، نقاط جمع و ... موجود می‌باشند. همچنین ابزارهای ورودی و خروجی مجازی مانند اسیلوسکوپ و ایجاد کننده تابع وجود دارند. سیمولینک در درون نرم‌افزار متلب نهاده شده و داده‌ها می‌توانند به سادگی در بین آن دو در تبادل باشند. در این کتاب، برای مثال‌های آورده شده در متلب، سیمولینک را برای مدل‌سازی سیستم‌ها، ساخت کنترلرها و شبیه‌سازی سیستم‌ها استفاده می‌نماییم. سیمولینک توسط محیط‌های ویندوز، مکینتاش و یونیکس پشتیبانی می‌شود و در نسخه‌ی دانشجویی نرم‌افزار متلب موجود می‌باشد. برای اطلاعات بیشتر به وبسایت اصلی MathWorks به آدرس www.MathWorks.com مراجعه نمایید.

راه‌اندازی سیمولینک

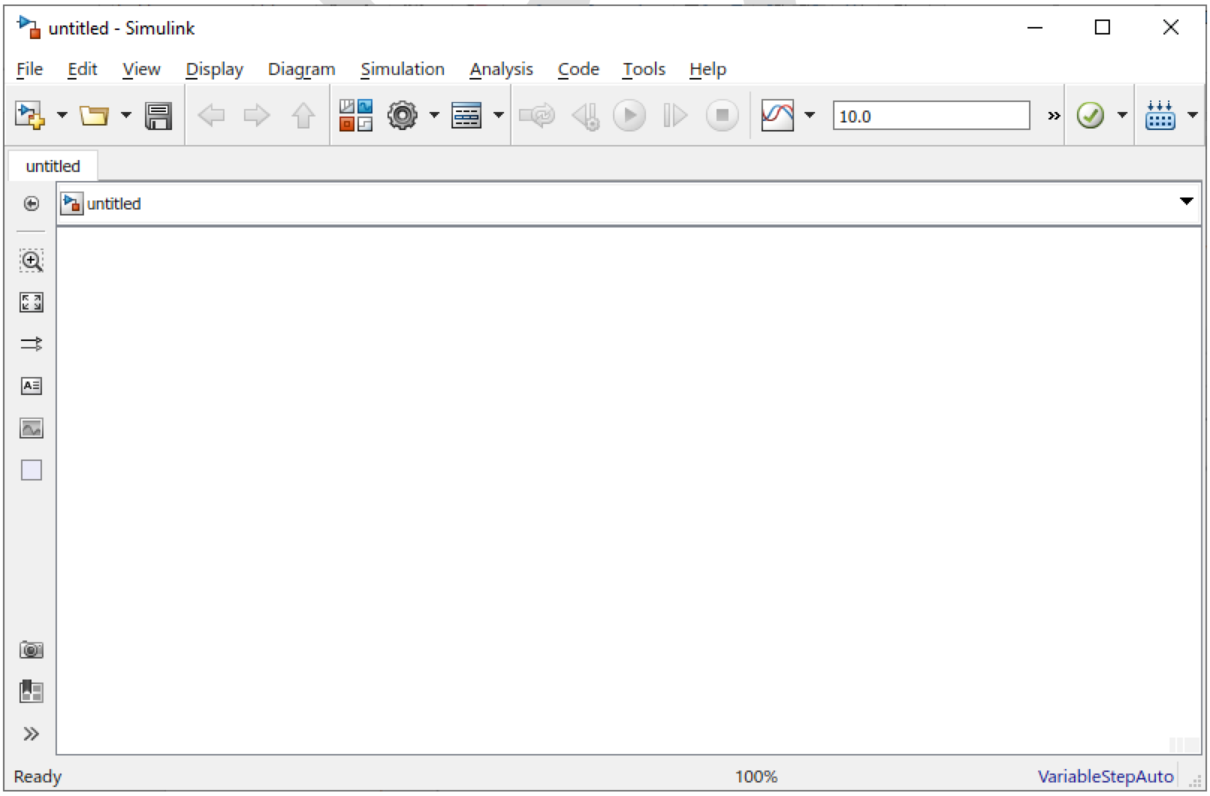
جهت راه‌اندازی سیمولینک دستور `simulink` را در پنجره‌ی دستور متلب تایپ کنید یا دکمه‌ی Simulink را در بالای صفحه فشار دهید:



بعد از راه‌اندازی سیمولینک یک صفحه با عنوان Simulink Start Page پدیدار می‌شود.



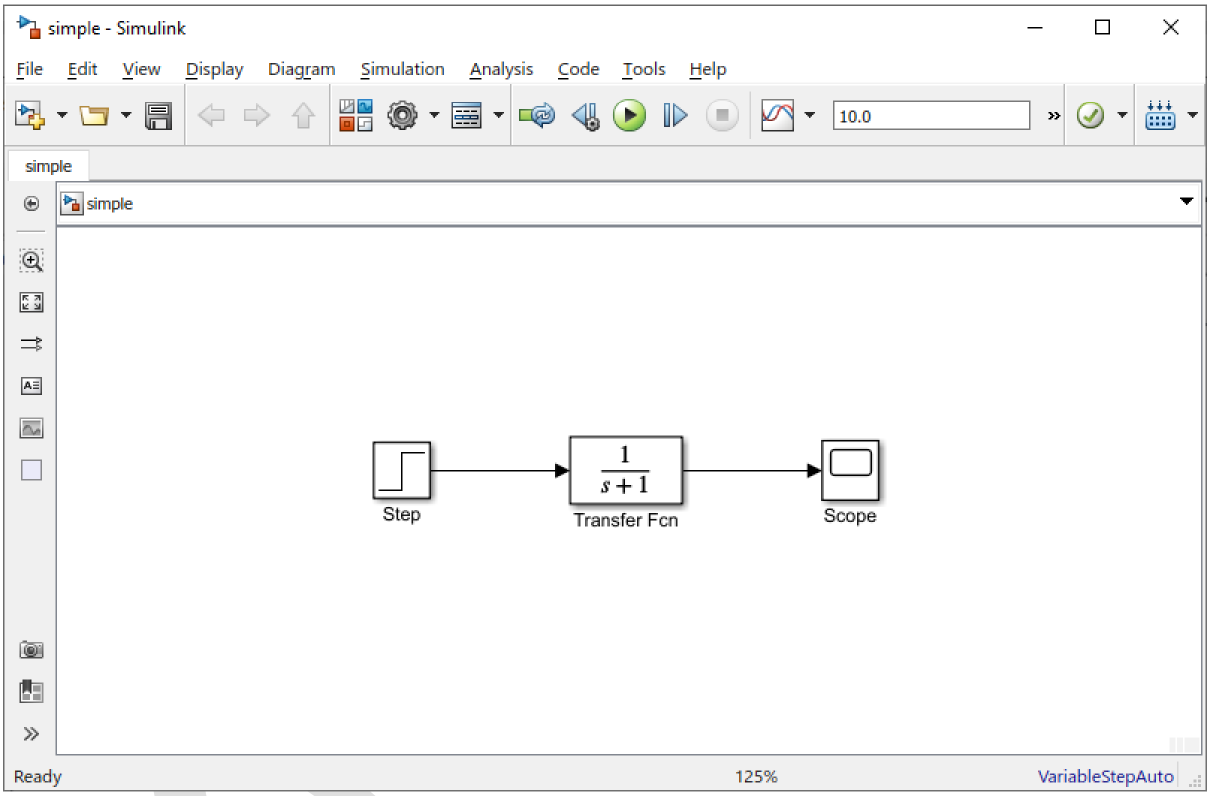
هنگامی که بر روی Blank Model کلیک کنید، پنجره‌ی جدید به شکل زیر باز می‌گردد:



فایل‌های مدل

در سیمولینک یک مدل متشکل از مجموعه‌ای از بلوک‌ها است که در کل، نمایش‌دهنده‌ی یک سیستم می‌باشد. علاوه بر ساخت یک مدل از ابتدا، می‌توان مدل‌های از پیش ساخته شده را از منوی File یا پنجره‌ی دستور متلب باز نمود. برای مثال مدل simple.slx را از سی‌دی دریافت کرده و در مسیر اجرای متلب کپی نمایید.

این فایل را با وارد کردن دستور `simple` در پنجره‌ی دستور متلب باز نمایید (یا می‌توانید از منوی File در سیمولینک و انتخاب گزینه‌ی Open این مدل را باز کنید). پنجره‌ی مدل به شکل زیر نمایش داده می‌شود:



برای ایجاد یک مدل جدید، در پنجره‌ی سیمولینک از منوی File گزینه‌ی New را انتخاب یا کلید ترکیبی Ctrl+N را بفشارید.

المان‌های پایه

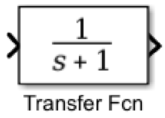
دو عنصر اساسی در محیط سیمولینک وجود دارد: بلوک‌ها و خط‌ها. بلوک‌ها جهت تولید، تنظیم و نمایش سیگنال‌ها به کار می‌رود. خط‌ها جهت انتقال سیگنال از بلوکی به بلوک دیگر به کار می‌روند.

بلوک‌ها

در کتابخانه‌ی سیمولینک چند دسته‌بندی کلی برای بلوک‌ها وجود دارد:

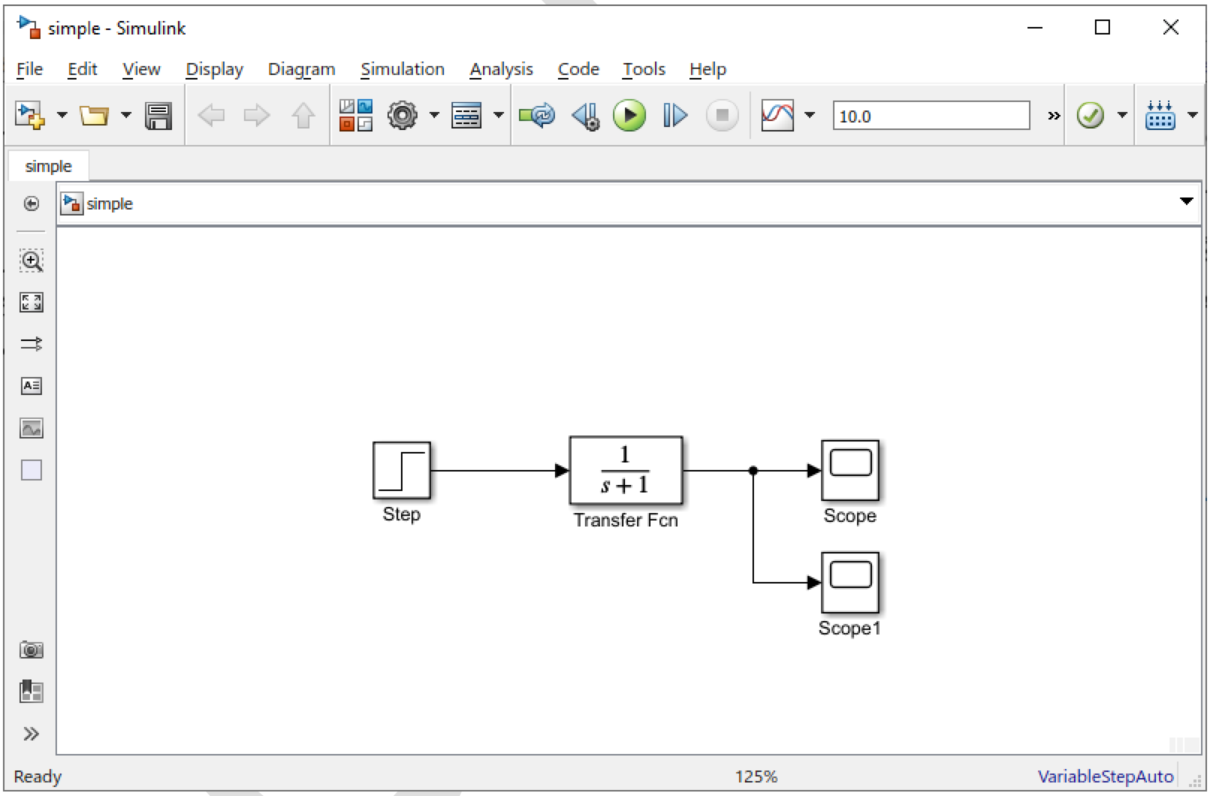
- چشمه (Sources): جهت تولید سیگنال‌ها استفاده می‌شود.
- چاه (Sinks): جهت گرفتن خروجی یا نمایش سیگنال‌ها به کار می‌رود.
- پیوسته (Continuous): المان‌های سیستم پیوسته در زمان (توابع تبدیل، مدل فضای حالت، کنترلر PID، ...)
- گسسته (Discrete): المان‌های سیستم گسسته در زمان و خطی (توابع تبدیل، مدل فضای حالت، کنترلر PID، ...)
- عملیات ریاضی (Math Operations): شامل عملیات ریاضی متداول (ضرب، جمع، قدرمطلق،...)
- درگاه‌ها و زیرسیستم‌ها (Ports & Subsystems): شامل بلوک‌های لازم جهت ساخت سیستم

بلوک‌ها ممکن است بدون درگاه یا شامل چندین درگاه ورودی یا خروجی باشند. درگاه‌های استفاده نشده به صورت نوک پیکان در اضلاع بلوک نمایش داده می‌شوند. بلوک نمایش داده شده در زیر شامل یک ورودی و یک خروجی استفاده نشده است:



خط‌ها

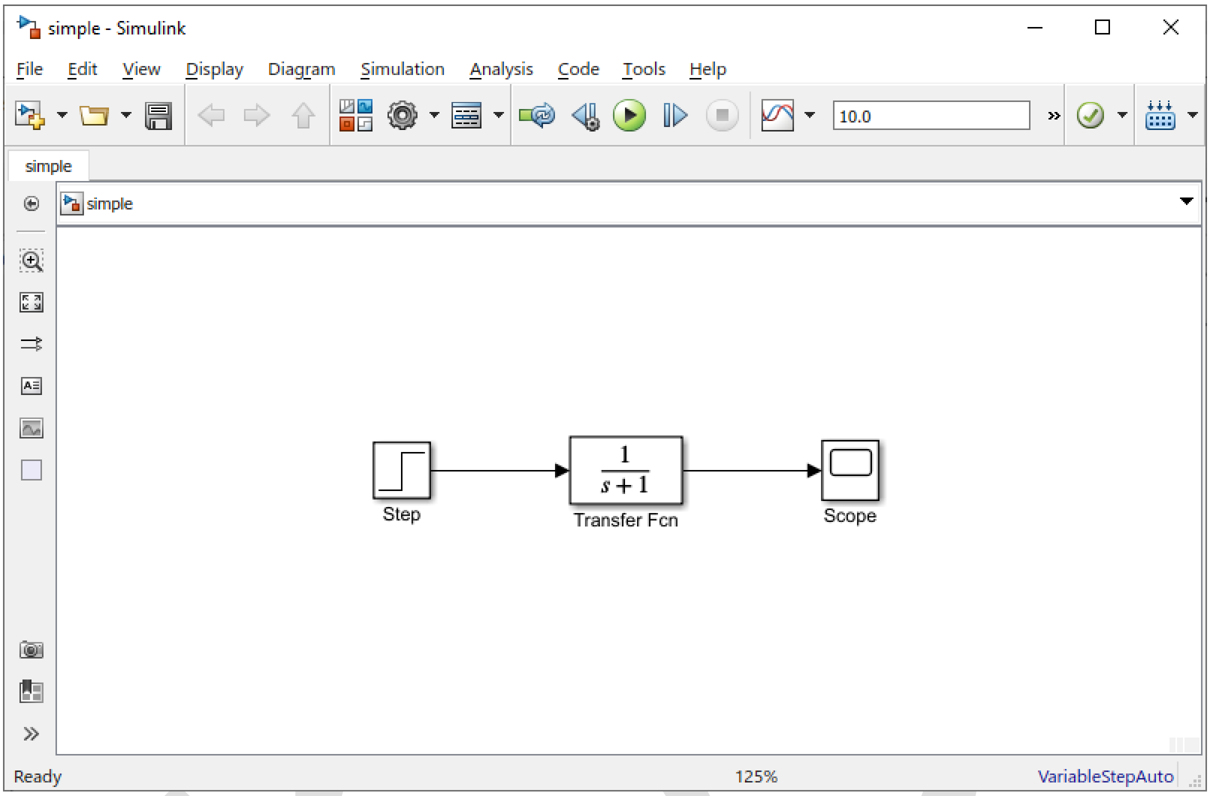
خط‌ها، سیگنال‌ها را انتقال می‌دهند. خط‌ها باید همواره سیگنال را از خروجی یک بلوک به ورودی بلوک دیگر منتقل کنند. همینطور یک سیگنال میتواند با ایجاد یک انشعاب، به صورت ورودی برای دو بلوک در نظر گرفته شود، همانطور که در شکل زیر نمایش داده شده است:



هیچگاه دو خط با یکدیگر صرفا ترکیب نمی‌شوند و حتما باید از یک بلوک مشخص همچون نقطه‌ی جمع (Summing Junction) استفاده شود.

یک سیگنال می‌تواند به صورت اسکالر یا برداری باشد. برای سیستم‌های تک ورودی تک خروجی (SISO) معمولا از سیگنال‌های اسکالر استفاده می‌شود. برای سیستم‌های چند ورودی چند خروجی (MIMO) معمولا از سیگنال‌های برداری استفاده می‌گردد که شامل دو یا چند سیگنال اسکالر است. خطوط انتقال دهنده‌ی سیگنال‌های اسکالر و برداری مشابه می‌باشند. نوع سیگنال منتقل شده توسط یک خط را بلوک‌های متصل به دو سر آن خط مشخص می‌کند.

یک مثال ساده



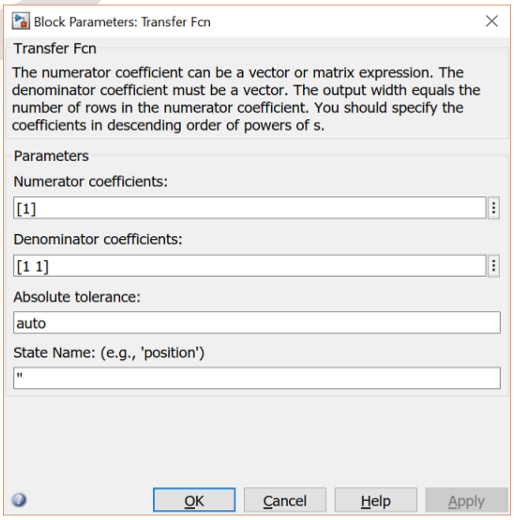
مدل ساده‌ی آمده در بالا از سه بلوک تشکیل شده است: Step, Transfer Function, Scope.

بلوک Step در واقع چشمه‌ی تولید سیگنال ورودی پله است. سیگنال تولید شده با خط به بلوک Transfer Function که یک بلوک پیوسته است انتقال یافته است. بلوک تابع تبدیل، تغییر متناسب را در سیگنال ایجاد کرده و خروجی خود را به بلوک Scope می‌فرستد. بلوک Scope یک بلوک Sink یا چاه است که برای نمایش سیگنال‌ها مانند اسیلوسکوپ به کار می‌رود.

در سیمولینک انواع مختلفی از بلوک‌ها وجود دارد که برخی از آنها در آینده مورد بحث قرار می‌گیرند. در حال حاضر تنها سه بلوک آمده در مدل بالا را مورد بررسی قرار می‌دهیم.

تنظیم بلوک‌ها

هر بلوک با دوبار کلیک کردن بر روی آن قابل تنظیم می‌گردد. برای مثال با دوبار کلیک کردن بر روی بلوک Transfer Function پنجره زیر باز می‌گردد:



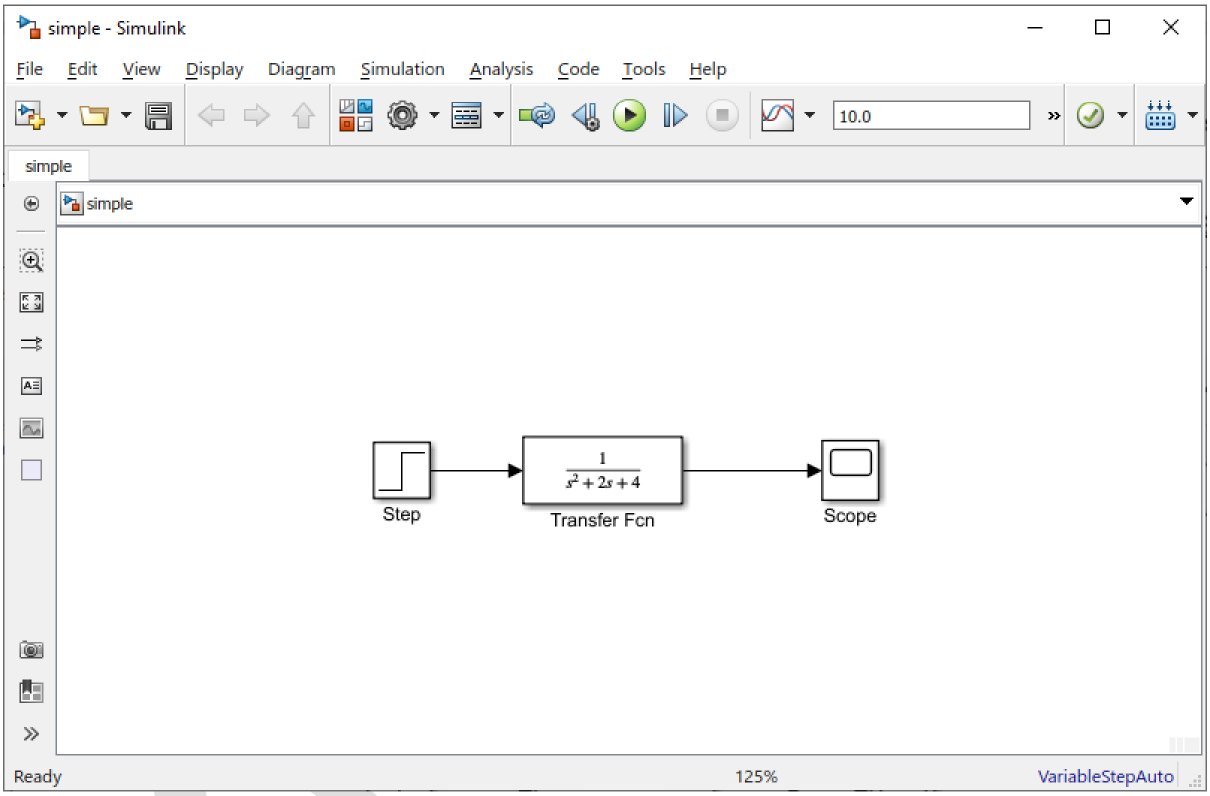
پنجره شامل قسمت‌های جهت وارد کردن ضرایب صورت و مخرج تابع تبدیل است. برای مثال جهت تغییر دادن مخرج به چند جمله‌ای زیر:

$$s^2 + 2s + 4 \tag{۱}$$

با وارد کردن عبارت زیر در بخش denominator یا مخرج:

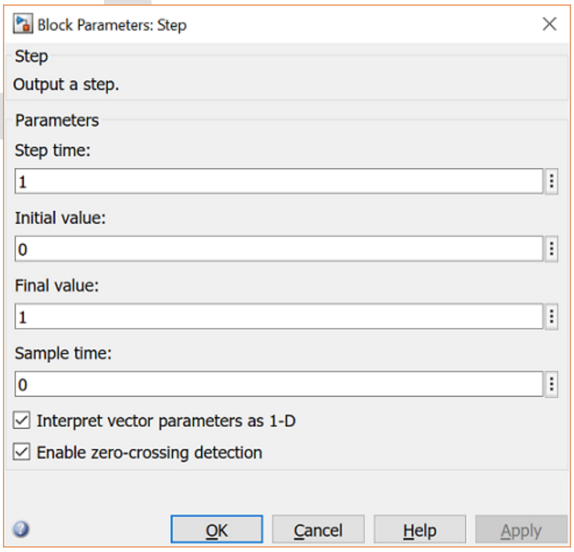
```
[1 2 4]
```

و بستن پنجره، مدل به‌صورت زیر درمی‌آید:

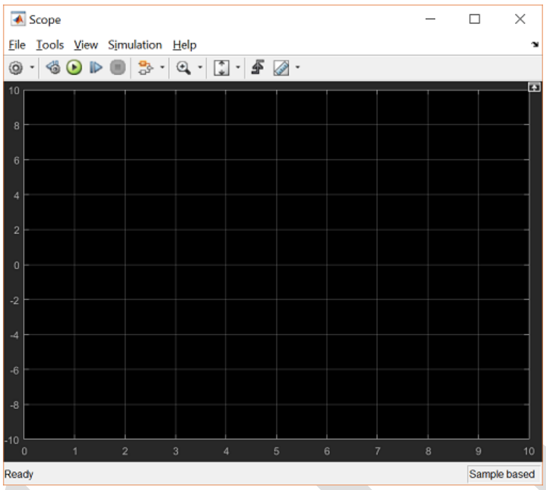


که نشان دهنده تغییر ایجاد شده در تابع تبدیل است.

با دوبار کلیک کردن بر روی بلوک Step پنجره زیر به نمایش درمی‌آید:



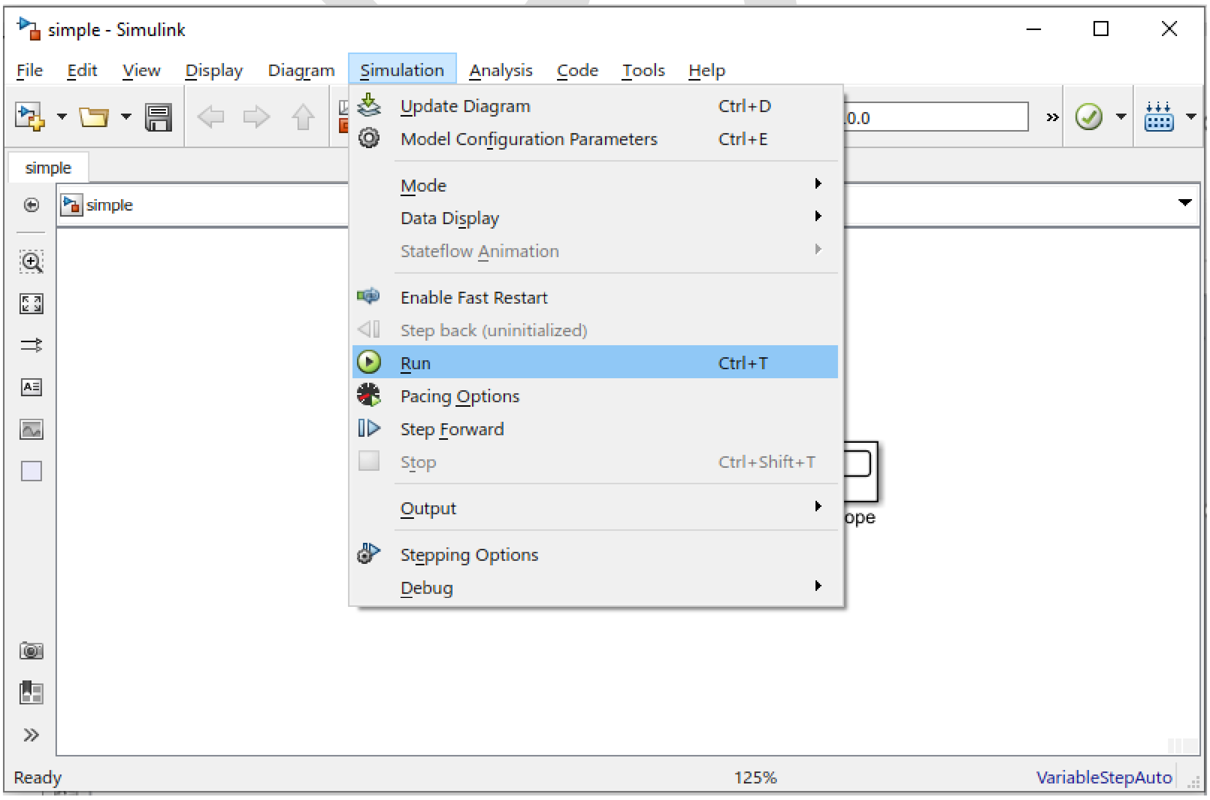
تنظیمات پیش‌فرض این بلوک به گونه‌ای است که یک سیگنال پله را در زمان ۱ ثانیه از مقدار اولیه صفر به مقدار نهایی یک می‌رساند. هر کدام از این مقادیر قابل تغییر هستند. پیچیده‌ترین این سه بلوک آورده شده، بلوک Scope است. با دوبار کلیک کردن بر روی این بلوک پنجره‌ی زیر باز می‌شود:



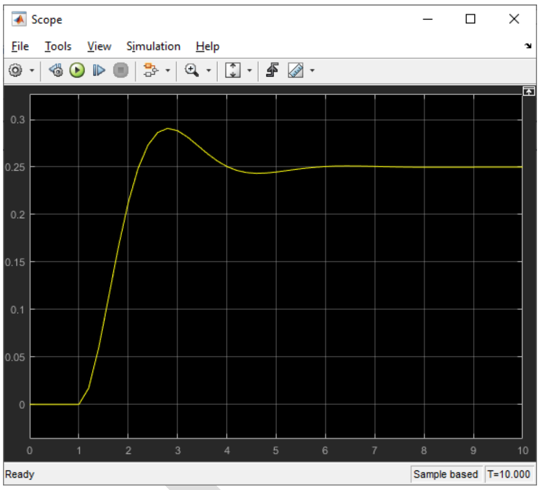
وقتی شبیه‌سازی مربوطه انجام شد، سیگنال تغذیه شده به این بلوک در این پنجره نمایش داده می‌شود. مشخصات دقیق‌تر این بلوک در این بخش بررسی نمی‌شود.

شبیه‌سازی

به مدل ساخته شده در بخش قبل مراجعه کنید. قبل از اجرا کردن برنامه، ابتدا دو بار بر روی Scope کلید کرده تا پنجره مربوطه باز شود. سپس گزینه Run را از منوی Simulation انتخاب کنید و گزینه Play را انتخاب کنید یا Ctrl-T را فشار دهید:



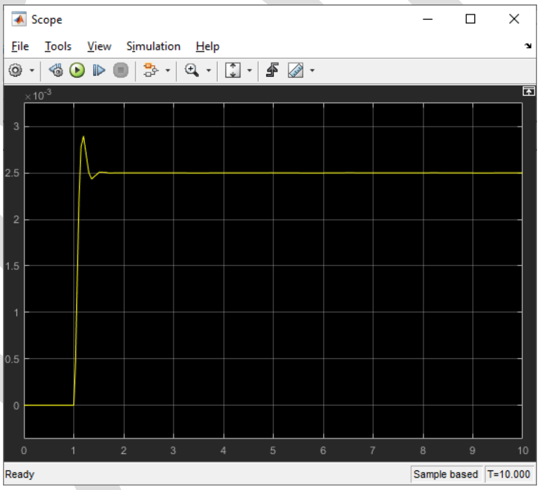
این شبیه‌سازی به سرعت صورت گرفته و پنجره Scope به صورت زیر نمایش داده شود:



توجه کنید که پاسخ سیستم تا زمان t=1 شروع نمی‌گردد که می‌توان این مورد را با دابل کلیک بر روی بلوک Step تغییر داد. با دابل کلیک بر روی تابع تبدیل و تغییر چند جمله ای مخرج به:

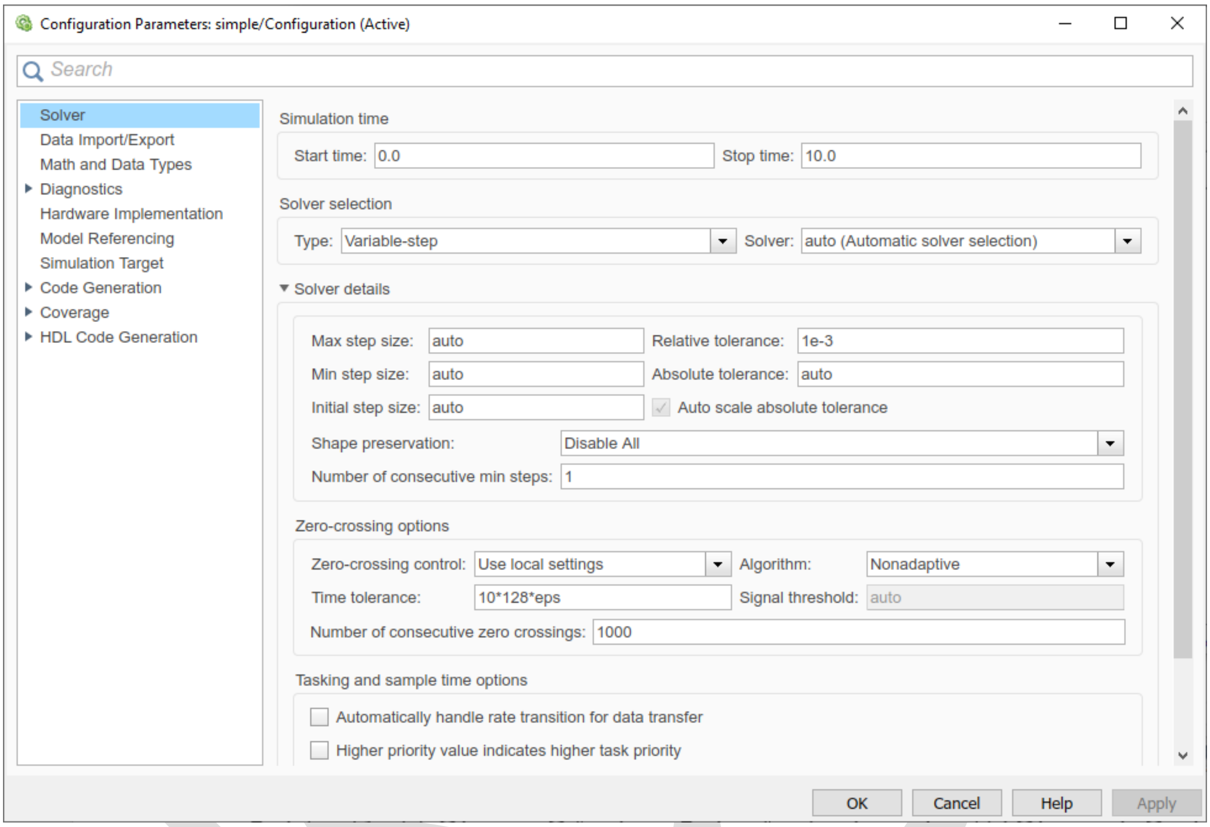
```
[1 20 400]
```

تابع تبدیل را تغییر می‌دهیم. حال با شبیه‌سازی مجدد سیستم، پاسخ سیستم به صورت زیر در می‌آید:

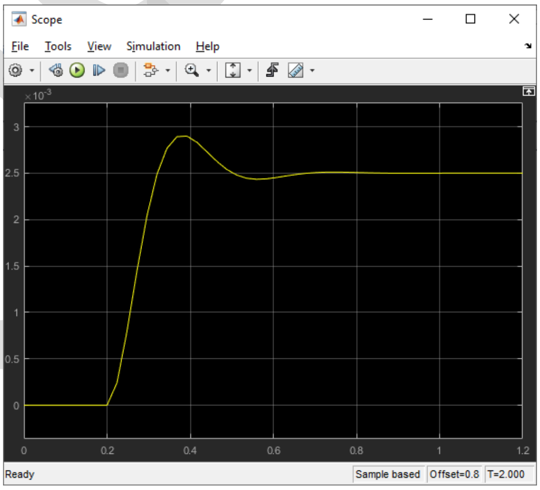


از آنجایی که تابع تبدیل جدید پاسخ خیلی سریعی از خود نشان داد، منحنی به صورت یک خط باریک فشرده شد. این مشکلی از بلوک Scope نیست اما از نظر شبیه‌سازی این نمایش صحیح نیست. سیمولینک شبیه‌سازی را برای ۱۰ ثانیه انجام داد، در حالی که سیستم خیلی در زمان بسیار کوتاه‌تری به حالت ماندگار رسیده بود.

برای حل این مشکل لازم است پارامترهای شبیه‌سازی را اصلاح کنیم. از منوی Simulation گزینه Model Configuration Parameters را انتخاب کنید، پنجره زیر باز می‌گردد:

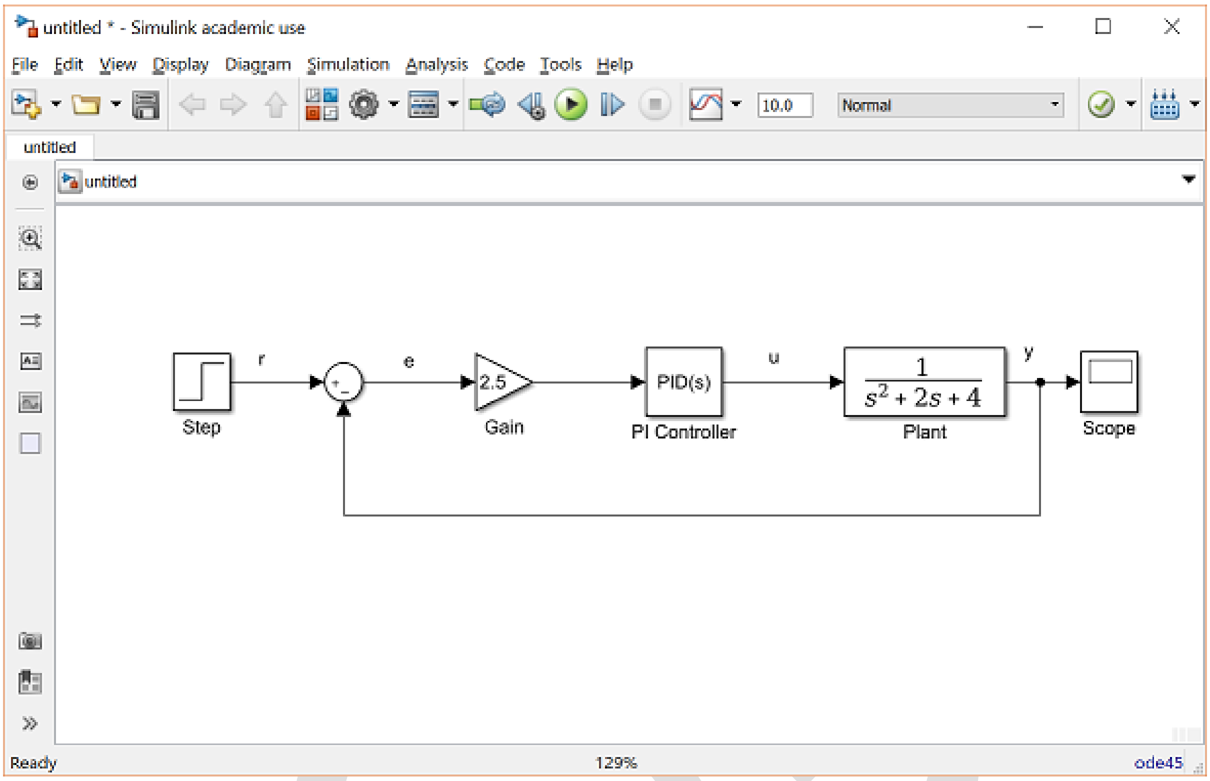


پارامترهای بسیار زیادی درشبیه‌سازی وجود دارد، ما در اینجا تنها به زمان ابتدا و انتهای شبیه‌سازی توجه می‌کنیم. زمان ابتدای شبیه‌سازی را از ۰ به ۰/۸ تغییر می‌دهیم (زیرا ورودی تا زمان ۱ ثانیه رخ نمی‌دهد). زمان پایان شبیه‌سازی را از ۱۰ ثانیه به ۲ ثانیه تغییر می‌دهیم که زمان کوتاهی پس از نشست سیستم است. هم اکنون منحنی Scope باید پاسخ به مراتب بهتری را رسم کند. شکل زیر پاسخ سیستم را نمایش می‌دهد:



تشکیل سیستم

در این بخش، یاد می‌گیریم تا چطور یک سیستم را با استفاده از بلوک‌های موجود در کتابخانه سیمولینک تشکیل دهیم. در این قسمت در نهایت سیستم زیر را تشکیل می‌دهیم:

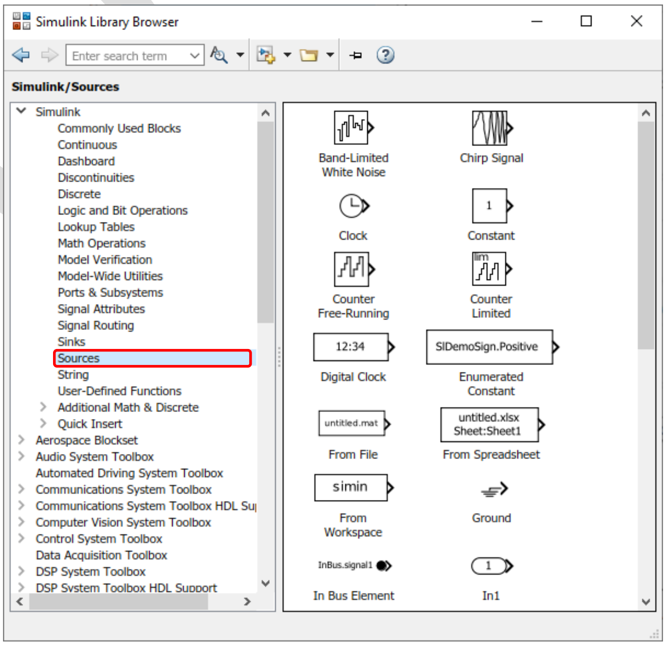


در ابتدا بلوک‌های لازم جهت تشکیل مدل را جمع‌آوری می‌کنیم. سپس بلوک‌ها را متناسب با سیستم موردنظر تنظیم می‌کنیم. سپس با خطوط، بلوک‌ها را به یکدیگر متصل کرده و سیستم را اجرا می‌کنیم تا از عملکرد آن مطمئن گردیم.

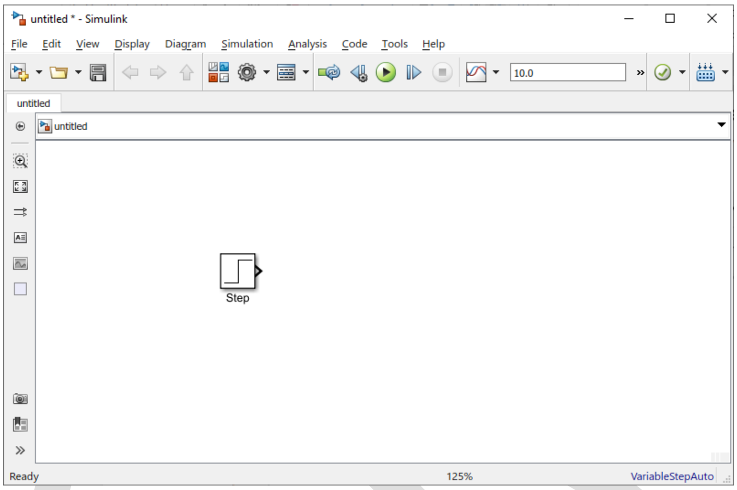
قرار دادن بلوک‌ها

جهت قرار دادن بلوک‌های لازم، مراحل زیر را طی می‌کنیم:

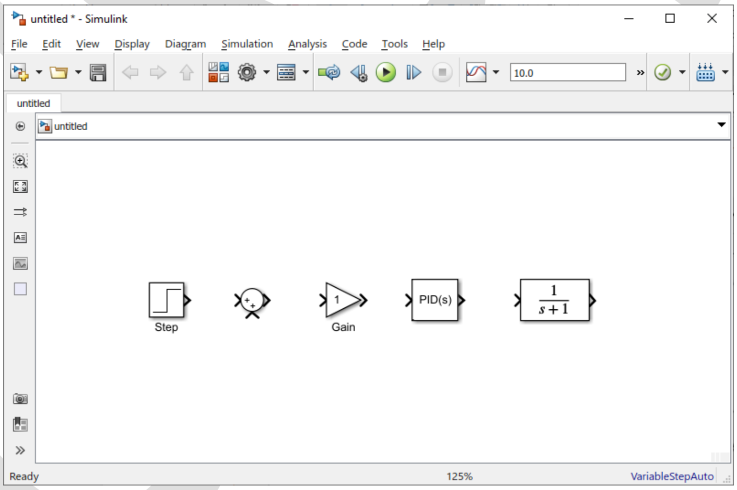
- ایجاد یک مدل جدید (انتخاب گزینه‌ی New از منوی File): حال یک مدل خالی تشکیل می‌گردد.
- بر روی منوی Tools کلیک کنید و Library Browser را انتخاب کنید.
- بر روی Sources در کتابخانه باز شده کلیک کنید.



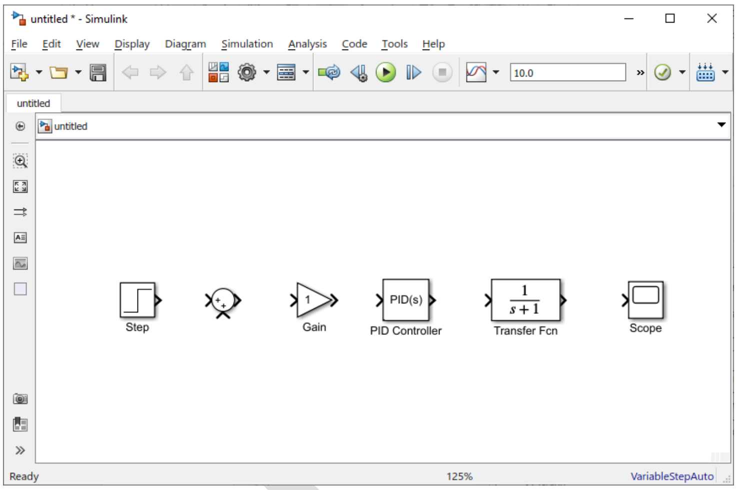
- این کار بلوک منابع را باز می‌کند تا سیگنال مورد نظر تولید گردد.
- بلوک Step را از بخش Sources بر روی صفحه‌ی ایجاد شده بکشید.



- در کتابخانه سیمولینک، بر روی Math Operations کلیک کنید.
- از کتابخانه بلوک‌های Sum و Gain را به مدل وارد کنید.
- در کتابخانه سیمولینک، بر روی Continuous کلیک کنید.
- ابتدا بلوک PID Controller را به داخل مدل وارد کنید و سمت راست بلوک Gain قرار دهید.
- از همان کتابخانه بلوک Transfer Function را وارد مدل کنید و سمت راست کنترلر قرار دهید.



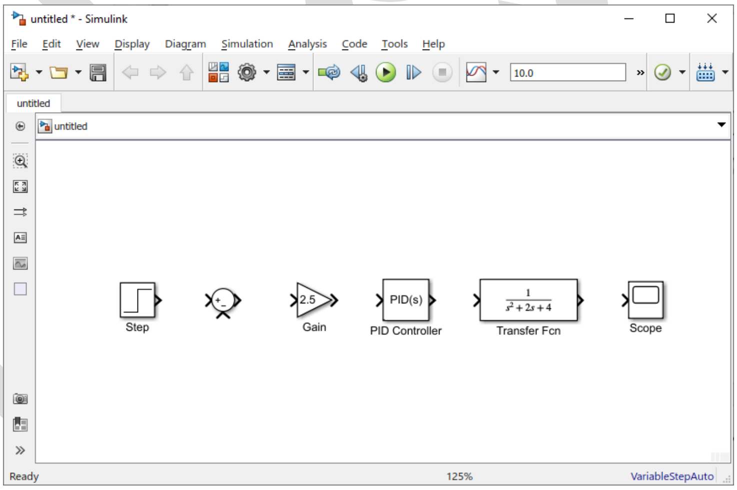
- بر روی Sinks کلیک کنید.
- بلوک Scope را وارد مدل کنید.



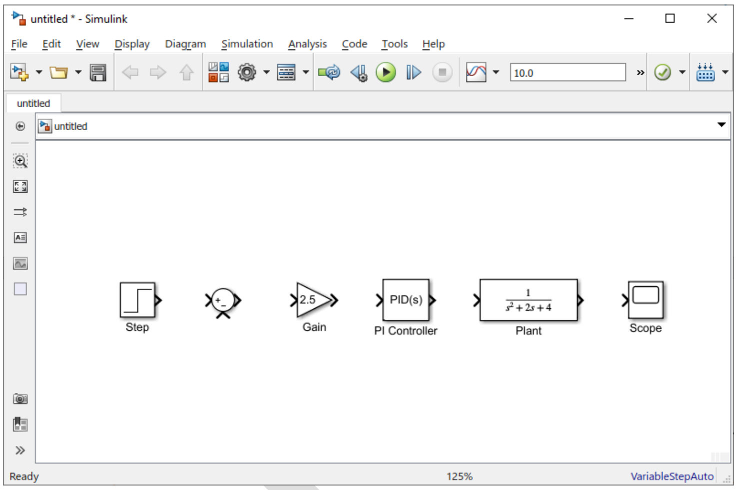
تنظیم بلوک‌ها

مراحل زیر را جهت تنظیم بلوک‌ها انجام می‌دهیم:

- دوبار بر روی بلوک Sum کلیک کنید. از آنجا که شما می‌خواهید دومین سیگنال از سیگنال دیگر کم شود، عبارت -+ را در لیست علامت‌ها وارد کنید.
- دوبار بر روی بلوک Gain کلیک کنید. ضریب مربوطه را به 2.5 تغییر دهید.
- دوبار بر روی بلوک PID Controller کلیک کنید و مقدار Proportional gain را به 1 و مقدار Integral gain را به 2 تغییر دهید.
- دوبار بر روی بلوک Transfer Function کلیک کنید. صورت را همان [1] قرار دهید و مخرج را به [1 2 4] تغییر دهید. حال مدل باید به صورت زیر در بیاید.



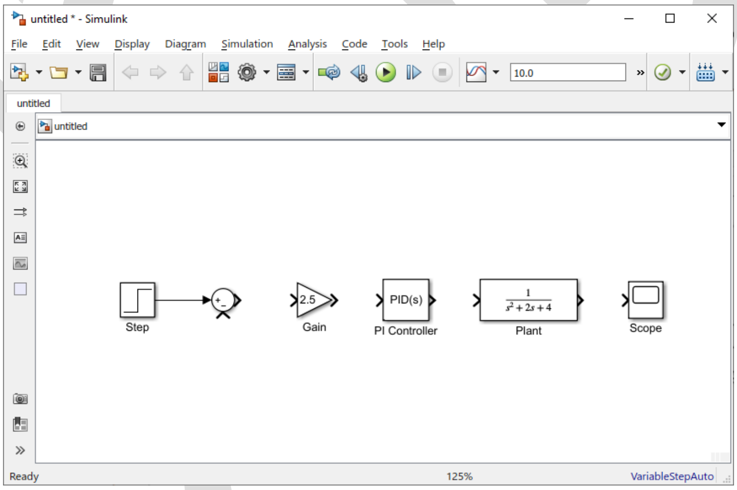
- نام بلوک کنترلر را به PI تغییر دهید (با دوبار کلیک کردن بر روی عبارت PID Controller).
- به همین صورت بلوک Transfer Functionرا به Plant تغییر دهید. حال تمام بلوک‌ها به درستی وارد شده‌اند. هم‌اکنون مدل شما باید به صورت زیر دربیاید:



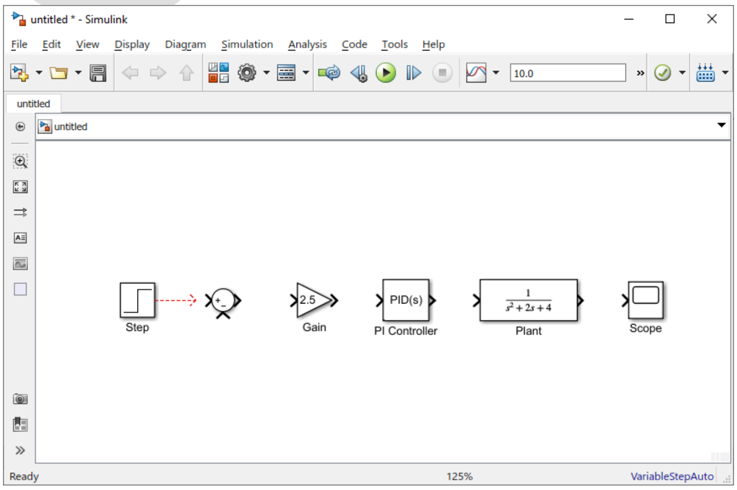
اتصال بلوک‌ها توسط خط‌ها

حال که کلیه بلوک‌ها به درستی قرار گرفته‌اند، شما باید آنها را توسط خط‌ها به یکدیگر متصل کنید. مراحل زیر را به ترتیب انجام دهید:

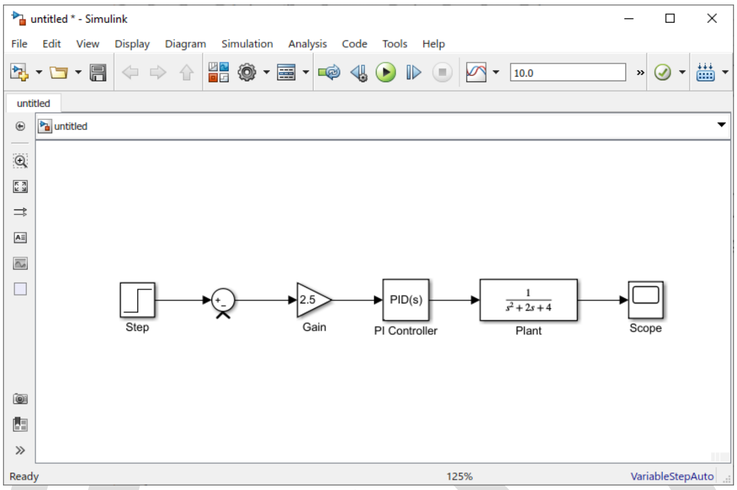
- با نگه داشتن کلیک ماوس، از خروجی بلوک Step به قسمت مثبت بلوک Sum وارد کنید و کلیک ماوس را رها کنید. حال شما باید تصویر زیر را مشاهده کنید:



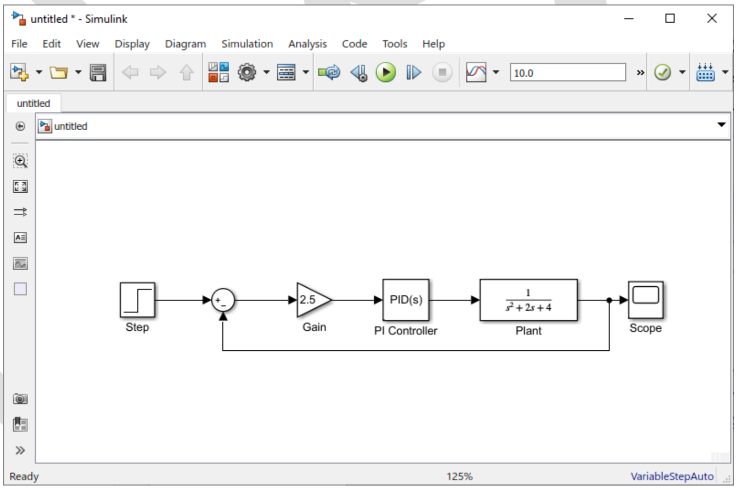
- خط رسم شده باید نوک پیکان بلوک را پر کند، اگر نوک پیکان متصل نشده و قرمز باشد، بدین معناست که خط مورد نظر به بلوکی متصل نیست.



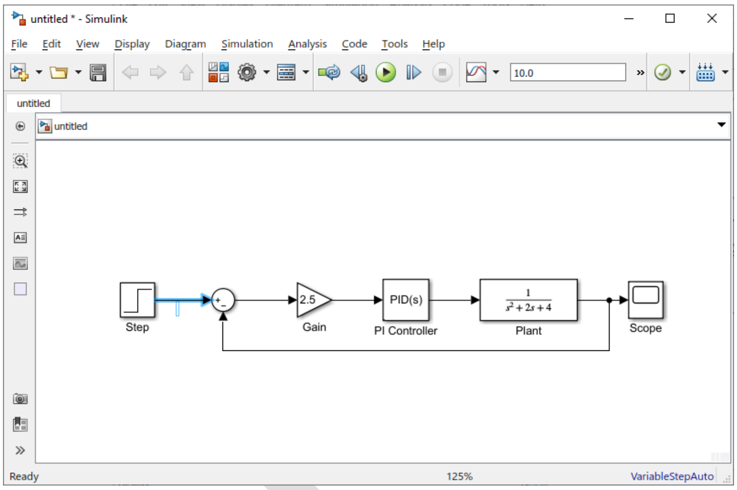
- در صورتی که قصد پاک کردن یک خط را دارید، به راحتی و تنها با کلیک بر روی آن و زدن دکمه Delete آن خط پاک می‌شود.
- به همین ترتیب یک خط از بلوک Sum به بلوک Gain متصل کنید و همینطور از Gain به PI Controller و از آن به Plant متصل کنید. خروجی Plant را به Scope متصل کرده و شکل زیر را تولید کنید:



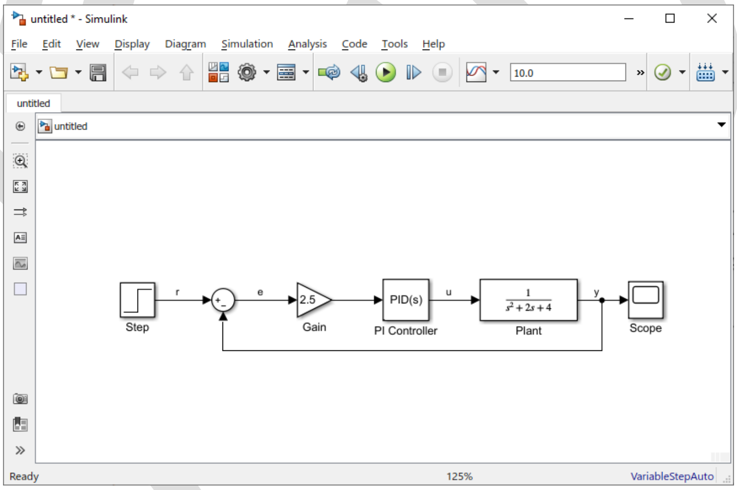
- خط باقی‌مانده، خط فیدبک سیگنال خروجی Plant به بخش منفی Sum است. این خط از دو جهت متفاوت است، اول این خط تکمیل کننده حلقه است و زاویه نود درجه دارد و دوم اینکه این خط از خروجی یک بلوک وارد نشده و از یک خط موجود شاعبه شده است.
- ماوس را از قسمت منفی Sum کشیده و به قسمتی از خط بین دو بلوک Plant و Scope متصل می‌کنیم. مدل نهایی حال باید به صورت زیر در بیاید:



- در نهایت برچسب‌های متناسب با هر سیگنال باید قرار داده شود. جهت انجام این کار بر روی هر نقطه دلخواه از صفحه دوبار کلیک کنید، حال یک بخش جهت تایپ عبارت مورد نظر همانند شکل زیر برای شما قرار داده می‌شود.



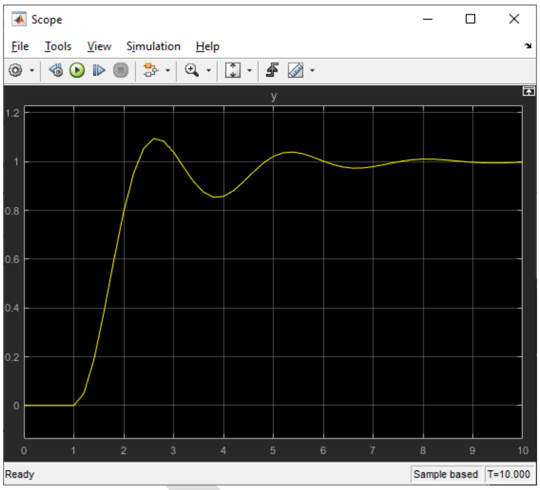
- در بخش مورد نظر r را تایپ کرده که نشان دهنده سیگنال مرجع است و سپس خارج از آن محدوده کلیک کنید تا از بخش تایپ خارج شوید.
- بر روی سیگنال خطا عبارت e در بخش کنترل u و خروجی را y نام‌گذاری کنید. شکل نهایی شما باید به صورت زیر در بیاید:



جهت ذخیره‌سازی مدل تولید شده، گزینه Save As را در منوی File انتخاب کنید و به اسم دلخواه خودتان ذخیره کنید.

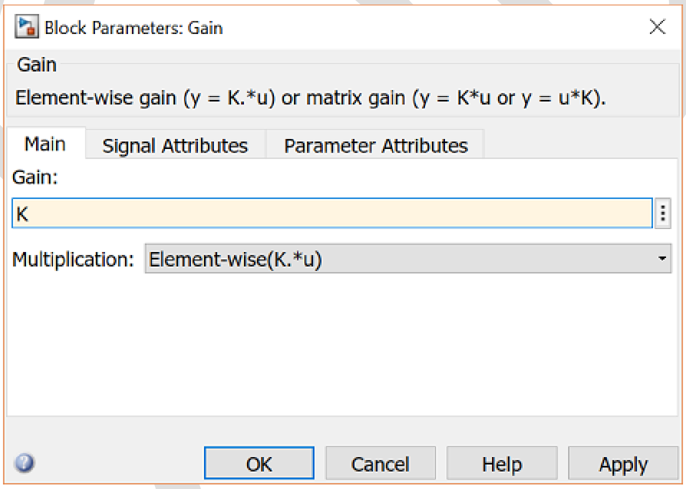
شبیه‌سازی

حال که مدل موردنظر کامل شده است، شما می‌توانید شبیه‌سازی را شروع کنید. گزینه Run را در منوی Simulation فشار دهید. دوبار بر روی بلوک Scope کلیک کنید، سیگنال خروجی باید به صورت زیر دربیاید:

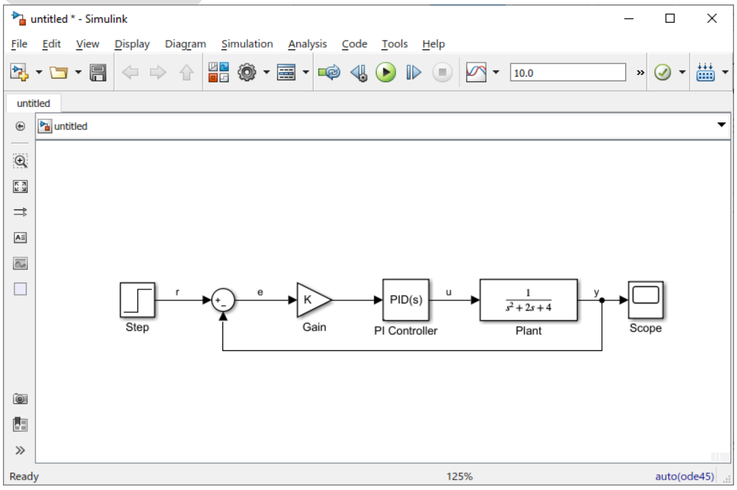


گرفتن متغیرها از متلب

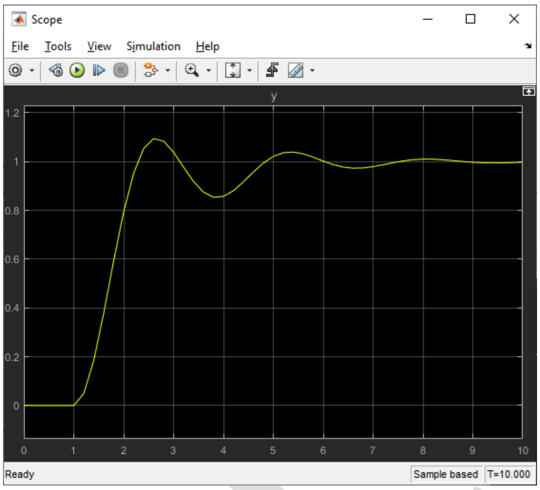
در بعضی مواقع پارامترهای مدل مثل ضرایب Gain ممکن است در محیط متلب محاسبه شده باشد و هم اکنون لازم است در محیط سیمولینک مورد استفاده قرار بگیرد. در این مواقع احتیاج نیست که مستقیما مقدار محاسبه شده وارد گردد، در این شرایط در صورتی که عبارت مورد نظر در متغیر K تحت عنوان مثال ذخیره شده باشد، با وارد کردن همین عبارت در محیط سیمولینک مقدار مورد نظر جایگزین می‌شود.



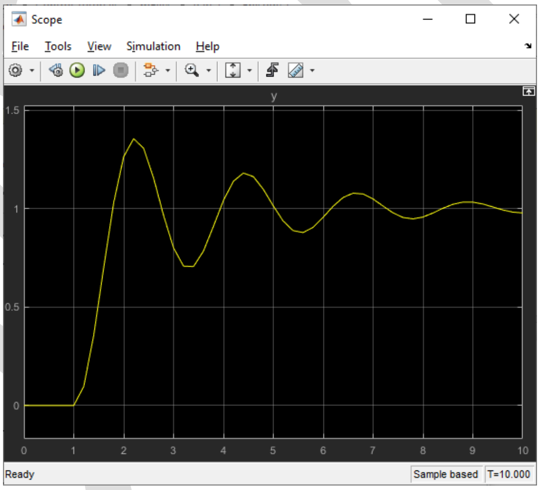
حال پنجره بالا را ببندید و توجه کنید که اینبار به جای مقدار در فیلد Gain، متغیر K قرار گرفته است. در Command Window متلب مقدار K=2.5 را وارد نمایید تا این متغیر در Workspace تعریف گردد.



حال مجددا شبیه‌سازی را انجام داده و مشاهده می‌کنیم که خروجی مورد نظر همانند حالت قبل است:



حال اگر هر گونه تغییری در محاسبات بخش متلب صورت گیرد که منجر به تغییر متغیرهای به کار رفته در سیمولینک گردد، سیمولینک از مقادیر جدید در شبیه‌سازی بعدی خود استفاده می‌کند. برای مثال در متلب عبارت K=5 را وارد کنید. حال مجددا شبیه‌سازی را تکرار کرده و بلوک Scope را باز کنید. هم اکنون منحنی خروجی را مشاهده می‌کنید که نشان دهنده ضریب بالاتر است.



علاوه بر متغیرها و سیگنال‌ها، حتی کل سیستم نیز می‌تواند بین متلب و سیمولینک جابجا شود.

فصل دوم: مقدمه

بخش اول: مدل‌سازی سیستم

فهرست مطالب بخش

اولین قدم در فرآیند طراحی کنترل، به دست آوردن مدل ریاضی مناسب برای سیستم مورد نظر می‌باشد. این مدل می‌تواند از طریق قوانین فیزیکی یا داده‌های آزمایشگاهی به دست آید. در این فصل به معرفی نمایش فضای حالت و تابع تبدیل سیستم دینامیکی می‌پردازیم. در ادامه به چند روش اصولی برای مدل‌سازی سیستم‌های مکانیکی و الکتریکی پرداخته و نحوه‌ی ساخت آنها در متلب را نشان می‌دهیم.

دستورهای کلیدی متلب در این بخش:

```
ss, tf
```

سیستم‌های دینامیکی

سیستم‌هایی که در طول زمان با یک قاعده‌ی ثابت، تغییر یا رشد می‌نمایند را سیستم‌های دینامیکی می‌نامند. برای بسیاری از سیستم‌های فیزیکی، این قاعده را می‌توان به صورت مجموعه‌ای از معادلات دیفرانسیل مرتبه اول نمایش داد:

$$\dot{x} = \frac{dx}{dt} = f(x(t), u(t), t) \qquad (۱)$$

در معادله بالا x(t) بردار حالت می‌باشد که شامل مجموعه‌ای از متغیرهای مشخصات سیستم در زمان t است. برای مثال در یک سیستم ساده‌ی مکانیکی جرم-فنر-دمپر، دو متغیر حالت می‌توانند موقعیت و سرعت جرم باشند. u(t) بردار ورودی‌های خارجی به سیستم در زمان t بوده و f یک تابع (احتمالا غیرخطی) مشتق زمانی بردار حالت، یعنی $\frac{dx}{dt}$، در زمان t می‌باشد.

حالت سیستم در هر لحظه‌ی بعد ($x(t_1)$) توسط دانستن حالت اولیه ($x(t_0)$) و تاریخچه‌ی ورودی‌ها ($u(t)$)، با انتگرال‌گیری از زمان t_0 تا زمان t_1 از معادله (۱) به دست می‌آید. اگرچه متغیرهای حالت منحصر به فرد نیستند، اما حداقل مقدار n متغیر حالت لازم است تا بتوان حالت سیستم را تعیین کرده و رفتار آینده‌ی سیستم را پیش‌بینی نمود. مقدار n برابر با مرتبه‌ی سیستم می‌باشد و نشان‌دهنده ابعاد فضای حالت است. مرتبه‌ی سیستم معمولا برابر با تعداد اجزای سیستم که قابلیت ذخیره انرژی را دارند، می‌باشد.

رابطه‌ی گفته شده در معادله (۱) بسیار کلی بوده و از آن برای تعریف طیف گسترده‌ای از سیستم‌های مختلف می‌توان استفاده نمود، هرچند که ممکن است تحلیل سیستم را بسیار دشوار کند. در حل مسائل، معادله‌ی گفته شده را می‌توان به دو صورت زیر ساده‌سازی کرد.

ساده‌سازی اول در صورتی است که تابع f به طور صریح به زمان وابسته نباشد، به این معنی که داریم $\dot{x} = f(x.u)$. در این حالت سیستم، نامتغیر با زمان[۴] نامیده می‌شود. این فرض اغلب موجه می‌باشد زیرا قوانین فیزیکی حاکم بر سیستم نیز معمولا به زمان وابسته نمی‌باشند. برای سیستم‌های نامتغیر با زمان، پارامتر یا ضرایب تابع f ثابت می‌باشد. اگرچه متغیرهای حالت $x(t)$ و ورودی‌های کنترل $u(t)$ ممکن است همچنان به زمان وابسته باشند.

فرض دوم برای ساده‌سازی سیستم، مربوط به خطی بودن سیستم می‌باشد. در واقعیت تقریبا تمامی سیستم‌های فیزیکی غیرخطی می‌باشند. به بیان دیگر، معمولا تابع f تابع پیچیده‌ای از حالت و ورودی‌ها می‌باشد. این رفتارهای غیرخطی از بسیاری از موارد مختلف نشأت می‌گیرند. یکی از شایع‌ترین عوامل غیرخطی بودن سیستم، اشباع شدن می‌باشد. هنگامی که یک سیستم اشباع می‌شود، یکی از اجزای سیستم در عملکرد خود به حد فیزیکی مشخصی می‌رسد. خوشبختانه در گستره‌ی عملیاتی کوچک (نواحی نزدیک خط مماس بر یک منحنی را در نظر بگیرید) دینامیک اکثر سیستم‌ها تقریبا خطی می‌باشد. در این صورت سیستم معادلات دیفرانسیل مرتبه اول را می‌توان توسط یک معادله‌ی ماتریسی به فرم $\dot{x} = Ax + Bu$ نمایش داد.

پیش از ظهور کامپیوترهای دیجیتال (و مدت زیادی بعد از آن) تنها تحلیل سیستم‌های خطی نامتغیر با زمان (LTI[5]) امکان پذیر بود. در نتیجه اکثر نتایج موجود در تئوری‌های کنترل بر همین اساس می‌باشند. خوشبختانه همانطور که ثابت شده است، این نتایج بسیار مفید بوده و بسیاری از مسائل مهم مهندسی نیز به کمک تکنیک‌های LTI حل می‌شوند. در اصل قدرت اصلی سیستم‌های کنترل فیدبک در کار کردن در حضور عدم قطعیت‌های اجتناب ناپذیر مدل‌سازی می‌باشد.

نمایش فضای حالت

برای سیستم‌های پیوسته خطی نامتغیر با زمان (LTI)، نمایش استاندارد فضای حالت به شرح زیر است:

$$\dot{x} = Ax + Bu \quad (۲)$$

$$y = Cx + Du \quad (۳)$$

که در آن x بردار متغیرهای حالت (با ابعاد $n \times 1$)، $\dot{x}$ مشتق زمانی بردار حالت $(n \times 1)$، u بردار ورودی یا کنترل $(p \times 1)$، y بردار خروجی $(q \times 1)$، A ماتریس سیستم $(n \times n)$، B ماتریس ورودی $(n \times p)$، C ماتریس خروجی $(q \times n)$ و D ماتریس پیشخور[۶] $(q \times p)$ می‌باشد.

معادله خروجی (معادله ۳) معادله‌ای حیاتی می‌باشد زیرا اغلب پیش می‌آید که متغیرهای حالت به صورت مستقیم مشاهده‌پذیر نیستند یا مطلوب ما نمی‌باشند. ماتریس خروجی C برای تعیین اینکه کدام‌یک از متغیرهای حالت (یا ترکیبی از آنها) برای استفاده‌ی کنترلر در دسترس می‌باشند استفاده می‌شود. همچنین در اکثر موارد، خروجی‌ها به طور مستقیم به ورودی‌ها وابسته نمی‌باشند (و فقط از طریق متغیرهای حالت به ورودی وابسته می‌باشد) که در این موارد ماتریس D صفر می‌باشد.

نمایش فضای حالت که به آن نمایش حوزه‌ی زمان نیز گفته می‌شود، به راحتی از طریق معادله ۱، برای سیستم‌های گوناگون از جمله سیستم‌های چند ورودی چند خروجی (MIMO[7])، با شرایط اولیه غیرصفر و غیرخطی به کار می‌رود. در نتیجه نمایش فضای حالت به طور گسترده در تئوری کنترل مدرن استفاده می‌گردد.

[۴] Time Invariant
[5] Linear Time Invariant
[۶] Feedforward
[7] Multi-Input Multi-Output

نمایش تابع تبدیل

خاصیت فوق‌العاده مهمی که در سیستم‌های LTI وجود دارد این است که اگر ورودی به سیستم سینوسی باشد، آنگاه خروجی از سیستم، سینوسی با همان فرکانس ورودی اما احتمالا با دامنه و فاز متفاوت خواهد بود. این تفاوت دامنه و فاز، تابعی از فرکانس بوده و ویژگی‌ای به نام **پاسخ فرکانسی** سیستم را ایجاد می‌کند.

با استفاده از **تبدیل لاپلاس** می‌توان نمایش حوزه‌ی زمان سیستم را به نمایش ورودی/خروجی حوزه‌ی فرکانس، که با نام **تابع تبدیل** شناخته می‌شود تبدیل نمود. با اعمال تبدیل لاپلاس، معادلات دیفرانسل حاکم بر سیستم نیز به معادلات جبری تبدیل می‌شوند که برای تحلیل بسیار ساده‌تر می‌باشد.

تبدیل لاپلاس یک تابع در حوزه‌ی زمان $f(t)$ به شکل زیر تعریف می‌شود:

$$F(s) = L\{f(t)\} = \int_0^\infty e^{-st} f(t)dt \qquad (۴)$$

که پارامتر $s = \sigma + j\omega$ متغیر فرکانس مختلط می‌باشد. بسیار نادر است که در عمل لازم باشد تا مستقیما تبدیل لاپلاس را محاسبه نمایید (اگرچه باید کاملا بلد باشید)، زیرا استفاده از جدول تبدیل لاپلاس توابع زمانی، بسیار رایج‌تر می‌باشد (به پیوست مراجعه نمایید).

تبدیل لاپلاس مشتق nام یک تابع بسیار حائز اهمیت می‌باشد که برابر است با:

$$\mathcal{L}\left\{\frac{d^n f}{dt^n}\right\} = s^n F(s) - s^{n-1} f(0) - s^{n-2} \dot{f}(0) - \cdots - f^{(n-1)}(0) \qquad (۵)$$

روش‌های حوزه‌ی فرکانسی اکثرا برای تحلیل سیستم‌های LTI تک ورودی/تک خروجی استفاده می‌شوند. سیستم‌های تک ورودی/تک خروجی دارای معادلاتی هستند که ضرایب جملات آنها یک عدد ثابت به شکل زیر می‌باشد:

$$a_n \frac{d^n y}{dt^n} + \cdots + a_1 \frac{dy}{dt} + a_0 y(t) = b_m \frac{d^m u}{dt^m} + \cdots + b_1 \frac{du}{dt} + b_0 u(t) \qquad (۶)$$

تبدیل لاپلاس معادله‌ی بالا برابر است با:

$$a_n s^n Y(s) + \cdots + a_1 s Y(s) + a_0 Y(s) = b_m s^m U(s) + \cdots + b_1 s U(s) + b_0 U(s) \qquad (۷)$$

که Y(s) و U(s) به ترتیب تبدیل لاپلاس y(t) و u(t) می‌باشند. لازم به ذکر است که در هنگام به دست آوردن تابع تبدیل، همواره فرض می‌شود تا تمامی شرایط اولیه $y(0)$، $\dot{y}(0)$ و $u(0)$ و ... مساوی با صفر می‌باشد. با این حساب تابع تبدیل خروجی Y(s) نسبت به ورودی U(s) برابر است با:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} \qquad (۸)$$

بهتر است با فاکتورگیری از ضرایب بزرگترین توان‌های صورت و مخرج، آنرا به فرم بهره-صفر-قطب[۸] نوشت:

$$G(s) = \frac{N(s)}{D(s)} = K \frac{(s - z_1)(s - z_2) \ldots (s - z_{m-1})(s - z_m)}{(s - p_1)(s - p_2) \ldots (s - p_{n-1})(s - p_n)} \qquad (۹)$$

به ریشه‌های چندجمله‌ای صورت یا در واقع همان sهایی که مقدار N(s)=0 را به دست می‌دهند، صفرهای تابع تبدیل $(z_1. \ldots . z_m)$ گفته می‌شود. همچنین به ریشه های چندجمله‌ای مخرج یا همان sهایی که رابطه‌ی D(s)=0 را ارضا می‌کنند، قطب‌های تابع تبدیل $(p_1. \ldots . p_n)$ گفته می‌شود. هر دو مقادیر صفرها و قطب‌ها می‌توانند مقادیر مختلط (دارای دو بخش حقیقی و موهومی) باشند. بهره سیستم برابر $K = b_m/a_n$ می‌باشد.

---

[۸] Gain-Zero-Pole

همچنین با کمی دقت می‌توان تابع تبدیل را مستقیما از نمایش فضای حالت به شکل زیر به دست آورد:

$$G(s) = \frac{Y(s)}{U(s)} = C(s\boldsymbol{I} - A)^{-1}B + D \qquad (۱۰)$$

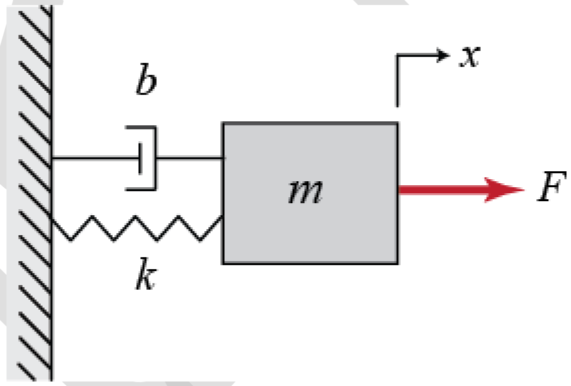
سیستم‌های مکانیکی

قوانین حرکت نیوتون پایه‌های تحلیل سیستم‌های مکانیکی را تشکیل می‌دهد. قانون دوم نیوتون (معادله ۱۱) بیانگر این است که برآیند نیروهای وارد بر یک جسم معادل با حاصلضرب جرم جسم و شتاب آن می‌باشد. همچنین قانون سوم نیوتون بیان می‌کند که دو جسمی که با یکدیگر در تماس می‌باشند، نیروی وارده به یک جسم توسط دیگری، با یکدیگر برابر و در خلاف جهت می‌باشد:

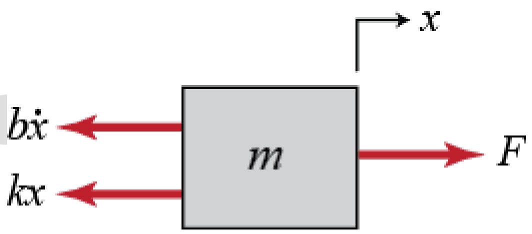
$$\Sigma \boldsymbol{F} = m\boldsymbol{a} = m\frac{d^2x}{dt^2} \qquad (۱۱)$$

بهتر است در هنگام استفاده از این معادله، دیاگرام جسم-آزاد سیستم به همراه تمامی نیروهای وارده ترسیم شود.

مثال: سیستم جرم-فنر-دمپر



دیاگرام جسم آزاد برای این سیستم در شکل زیر نشان داده شده است. نیروی فنر متناسب با جابجایی جرم (x) بوده و نیروی اصطهلاک میرایی متناسب با سرعت جرم ($v = \dot{x}$) می‌باشد. هر دوی این نیروها مخالف با جهت حرکت جرم و در نتیجه در دیاگرام جسم آزاد در جهت منفی x می‌باشند. همچنین دقت نمایید که در موقعیت $x = 0$ فنر در حالت استراحت می‌باشد.



حال برآیند نیروها را محاسبه کرده و از قانون دوم نیوتون (معادله ۱۱) در هر راستا استفاده می‌نماییم. در این مثال به علت اینکه نیرویی در راستای y وارد نمی‌شود، معادله ۱۱ را برای راستای x می‌نویسیم:

$$\Sigma F_x = F(t) - b\dot{x} - kx = m\ddot{x} \qquad (۱۲)$$

معادله‌ی به دست آمده با نام **معادله‌ی حاکم** شناخته شده و حالت دینامیکی سیستم را کاملا تعریف می‌نماید. در بخش‌های بعد خواهیم دید که چطور با استفاده از این معادله، پاسخ سیستم را به ورودی خارجی ($F(t)$) محاسبه کرده و همچنین ویژگی‌های سیستم از جمله پایداری و عملکرد را تحلیل می‌نماییم.

برای به دست آوردن نمایش فضای حالت سیستم جرم-فنر-دمپر، باید معادله حاکم مرتبه‌ی دوم به دست آمده را به مجموعه‌ی دو معادله‌ی دیفرانسیل مرتبه اول کاهش داد. بدین منظور ما موقعیت و سرعت را به عنوان متغیرهای حالت در نظر می‌گیریم:

$$x = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{۱۳}$$

متغیر موقعیت، انرژی پتانسیل ذخیره شده در فنر و متغیر سرعت، انرژی ذخیره شده در جرم را نتیجه می‌دهد. دمپر تنها انرژی را تلف کرده و چیزی را ذخیره نمی‌کند. این نکته که کدام یک از متغیرها انرژی را در سیستم ذخیره می‌کنند اغلب برای انتخاب متغیرهای حالت کمک می‌کند.

در این مثال معادله حالت به شکل زیر است:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t) \tag{۱۴}$$

برای مثال اگر بخواهیم موقعیت جرم را کنترل کنیم، آنگاه معادله خروجی سیستم برابر است با:

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{۱۵}$$

وارد کردن مدل‌های فضای حالت در متلب

در این قسمت می‌خواهیم نحوه‌ی وارد کردن معادلات به دست آمده را در یک ام‌فایل متلب نشان دهیم. علائم زیر را به هر یک از متغیرها اختصاص می‌دهیم:

```
m       mass               1.0 kg
k       spring constant    1.0 N/m
b       damping constant   0.2 Ns/m
F       input force        1.0 N
```

حال یک ام‌فایل جدید ساخته و دستورهای زیر را وارد می‌نماییم:

```
m = 1;

k = 1;

b = 0.2;

F = 1;



A = [0 1; -k/m -b/m];

B = [0 1/m]';

C = [1 0];

D = [0];



sys = ss(A,B,C,D)
```

```
sys =

  A =
         x1     x2
   x1     0      1
   x2    -1   -0.2

  B =
       u1
   x1   0
   x2   1

  C =
       x1  x2
   y1   1   0

  D =
       u1
   y1   0

Continuous-time state-space model.
```

تبدیل لاپلاس با فرض شرایط اولیه‌ی صفر برای سیستم فوق برابر است با:

$$ms^2X(s) + bsX(s) + kX(s) = F(s) \tag{۱۶}$$

در نتیجه تابع تبدیل خروجی سرعت نسبت به ورودی نیرو برابر است با:

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k} \tag{۱۷}$$

وارد کردن مدل‌های تابع تبدیل در متلب

در این بخش به ساخت مدل تابع تبدیل به دست آمده در متلب می‌پردازیم. با وارد کردن دستورهای زیر در یک ام‌فایل، و وارد کردن پارامترها داریم:

```
s = tf('s');
sys = 1/(m*s^2+b*s+k)
```

```
sys =


        1

  ---------------

  s^2 + 0.2 s + 1


Continuous-time transfer function.
```

در اینجا از متغیر s برای تعریف مدل تابع تبدیل استفاده شده است. پیشنهاد می‌شود همواره از این راه برای تعریف تابع تبدیل استفاده کنید، هرچند که ممکن است در نسخه‌های قدیمی‌تر نرم‌افزار متلب یا در هنگام اتصال به سیمولینک، لازم باشد تا مدل تابع تبدیل با استفاده از ضرایب چندجمله‌ای صورت و مخرج به صورت مستقیم تعریف شود. در این صورت از دستورهای زیر استفاده می‌کنیم:

```
num = [1];

den = [m b k];

sys = tf(num,den)
```

```
sys =


        1

  ---------------

  s^2 + 0.2 s + 1


Continuous-time transfer function.
```

شناسایی سیستم

در قسمت‌های پیش، آموختیم که چگونه از قوانین اساسی فیزیک برای مدل‌سازی سیستم‌ها استفاده نماییم. اما در واقعیت این روش‌ها به طور کامل امکان‌پذیر نمی‌باشند زیرا پارامترهای سیستم نامعین می‌باشند یا سیستم مورد نظر دارای پیچیدگی‌هایی می‌باشد. در این حالات به اندازه‌گیری‌های آزمایشگاهی و روش‌های آماری رجوع کرده و مدل سیستم را به دست می‌آوریم، به این فرآیند **شناسایی سیستم** گفته می‌شود.

شناسایی سیستم می‌تواند برای داده‌های در حوزه‌ی زمان یا حوزه‌ی فرکانس انجام شود (به **پیوست سوم: شناسایی سیستم** مراجعه نمایید).

تبدیل سیستم

اکثر عملیات انجام شده در متلب را می‌توان بر روی تابع تبدیل، مدل فضای حالت یا فرم صفر-قطب-بهره انجام داد. علاوه بر آن در صورت نیاز می‌توان هر یک از این فرم‌ها را به فرم‌های دیگر تبدیل نمود. برای یادگیری تبدیل هر یک از این فرم‌ها به فرم دیگر، به **پیوست دوم: تبدیل فرم نمایش سیستم** مراجعه کنید.

بخش دوم: تحلیل سیستم

فهرست مطالب بخش

- معرفی پاسخ زمانی
- معرفی پاسخ فرکانسی
- پایداری
- مرتبه‌ی سیستم
- سیستم‌های مرتبه اول
- سیستم‌های مرتبه دوم

بعد از به دست آوردن مدل ریاضی یک سیستم، چه به فرم فضای حالت و چه به فرم تابع تبدیل، می‌توانیم تحلیل مدل به دست آمده را برای پیش‌بینی پاسخ سیستم در دو حوزه‌ی زمان و فرکانس انجام دهیم. مباحثی که سیستم‌های کنترل بر اساس آنها طراحی می‌شوند عبارتند از افزایش پایداری، سرعت پاسخ، خطای حالت ماندگار و پیشگیری از ارتعاشات. در این بخش نشان می‌دهیم که چطور این مشخصات دینامیکی را از مدل‌های سیستم به دست آوریم.

دستورهای کلیدی متلب در این بخش:

```
tf, ssdata, pole, eig, step, pzmap, bode, linearSystemAnalyzer
```

پاسخ زمانی

پاسخ زمانی مشخص می‌نماید که در هنگام اعمال یک ورودی خاص، حالت دینامیکی سیستم در زمان چگونه تغییر می‌کند. به علت اینکه معادلات به دست آمده برای سیستم‌ها از نوع معادلات دیفرانسیل می‌باشد، برای به دست آوردن پاسخ زمانی سیستم نیاز به انتگرال گیری می‌باشد. ممکن است برای برخی از سیستم‌های ساده، جواب تحلیلی حلقه بسته وجود داشته باشد. اما برای اکثر سیستم‌ها، به خصوص سیستم‌های غیرخطی یا سیستم‌هایی که ورودی پیچیده‌ای را دریافت می‌کنند، باید این انتگرال‌گیری ها به صورت عددی انجام شود. خوشبختانه متلب بسیاری از منابع مفید برای محاسبه‌ی پاسخ‌های زمانی نسبت به ورودی های مختلف را فراهم کرده است که در ادامه به آن می‌پردازیم.

پاسخ زمانی یک سیستم دینامیکی خطی به صورت حاصل جمع **پاسخ گذرا** که وابسته به شرایط اولیه و **پاسخ حالت ماندگار** که وابسته به ورودی سیستم می‌باشد تعریف می‌شود. این پاسخ‌ها به ترتیب مربوط به حل قسمت همگن (آزاد یا ورودی صفر) و جواب‌های خاص معادله‌ی دیفرانسیل سیستم می‌باشد.

پاسخ فرکانسی

تمامی مثال‌های گفته شده در این کتاب توسط معادلات دیفرانسیل با ضرایب ثابت خطی، مدل‌سازی شده‌اند که از نوع خطی نامتغیر با زمان (LTI) حساب می‌شوند. همانطور که اشاره شد ویژگی مهم سیستم‌های LTI این است که اگر ورودی به سیستم سینوسی باشد آنگاه خروجی حالت ماندگار سیستم نیز سینوسی با همان فرکانس اما دامنه و فاز متفاوت خواهد بود. این تفاوت دامنه و فاز تابعی از فرکانس بوده که **پاسخ فرکانسی** سیستم را تشکیل می‌دهد.

پاسخ فرکانسی یک سیستم را می‌توان از تابع تبدیل آن سیستم به شرح زیر به دست آورد: برداری (مجموعه‌ای) از فرکانس‌های مختلف (از صفر یا "DC" تا بینهایت) ایجاد کرده و مقادیر تابع تبدیل سیستم را در آن فرکانس‌ها به دست می‌آوریم. اگر $G(s)$ تابع تبدیل حلقه باز سیستم و ω بردار فرکانس باشد، آنگاه می‌توان نمودار مقادیر $G(j\omega)$ نسبت به ω را رسم نمود. از آنجایی که $G(j\omega)$ اعداد مختلط را به دست می‌دهد، می‌توان هر دو نمودار اندازه و فاز (**دیاگرام بودی**[۹]) یا موقعیت آن در مختصات مختلط (**نمودار نایکوئست**) را رسم نمود. مفهوم ارائه شده در هر دوی این نمودارها یکسان بوده اما روش ارائه آنها متفاوت است.

---

[۹] Bode diagram

پایداری

در این کتاب با توجه به هدف‌هایی که دنبال می‌کنیم، از تعریف **ورودی کراندار خروجی کراندار (BIBO[10])** برای پایداری استفاده می‌کنیم. این تعریف پایداری یک سیستم را به این صورت تعریف می‌کند که اگر به ازای تمام ورودی‌های کراندار (محدود)، خروجی کراندار بماند آنگاه سیستم پایدار است. به بیان دیگر سیستم در هنگام عملکرد خود، تخریب (Blow up) نمی‌شود.

در تحلیل پایداری سیستم، نمایش تابع تبدیل سیستم بسیار مفید می‌باشد. اگر تمامی قطب‌های تابع تبدیل (مقادیری از s که چندجمله‌ای مخرج را صفر می‌کند) دارای قسمت حقیقی منفی باشند، آنگاه سیستم پایدار است. اگر هر یک از قطب‌ها دارای قسمت حقیقی مثبت باشد سیستم ناپایدار می‌باشد. اگر قطب‌ها را بر روی صفحه مختلط s نمایش دهیم آنگاه برای پایداری باید تمامی قطب‌ها در سمت چپ محور موهومی قرار بگیرند. اگر قطبی بر روی محور موهومی قرار بگیرد آنگاه سیستم پایدار مرزی (Marginally Stable) بوده و تمایل به نوسان دارد. یک سیستم با قطب‌های کاملا موهومی، پایدار BIBO محسوب نمی‌گردد. برای این سیستم‌ها ورودی محدود، پاسخ نامحدود را منجر می‌شود. قطب‌های یک سیستم LTI با استفاده از دستور `pole` در متلب به دست می‌آید، برای مثال داریم:

```
s = tf('s');
G = 1/(s^2+2*s+5)
pole(G)
```

```
G =
 
 
        1
  -------------
  s^2 + 2 s + 5
 
 
Continuous-time transfer function.

ans =

  -1.0000 + 2.0000i

  -1.0000 - 2.0000i
```

در نتیجه این سیستم پایدار می‌باشد زیرا قسمت حقیقی هر دو قطب منفی می‌باشد. پایداری یک سیستم را می‌توان از نمایش فضای حالت نیز به دست آورد. در واقع قطب‌های تابع تبدیل همان مقادیر ویژه[11] ماتریس سیستم A می‌باشد. می‌توان برای محاسبه‌ی مقادیر ویژه از دستور `eig` به طور مستقیم و یا از دستور `eig(G)` بر روی سیستم به شکل زیر استفاده نمود:

```
[A,B,C,D] = ssdata(G);

eig(A)
```

---

10 Bounded Input Bounded Output

11 eigenvalue

```
ans =

  -1.0000 + 2.0000i

  -1.0000 - 2.0000i
```

مرتبه‌ی سیستم

به بزرگترین مرتبه‌ی مشتق موجود در معادله‌ی دیفرانسیل حاکم بر سیستم، مرتبه‌ی سیستم می‌گویند. معادل آنرا می‌توان بزرگترین توان s در تابع تبدیل سیستم در نظر گرفت. ویژگی‌های مهم سیستم‌های مرتبه یک، مرتبه دو و مراتب بالاتر را در این قسمت بررسی می‌کنیم.

سیستم مرتبه اول

سیستم‌های مرتبه اول ساده‌ترین سیستم‌های دینامیکی برای تحلیل می‌باشند. مثالی که از این سیستم‌ها می‌توان مطرح کرد سیستم جرم و فنر و سیستم مدار RC است.

فرم کلی معادله دیفرانسیل مرتبه اول به شکل زیر است:

$$\dot{y} + ay = bu \quad \text{یا} \quad \tau\dot{y} + y = k_{dc}u \tag{۱}$$

فرم تابع تبدیل مرتبه اول نیز به شکل زیر است:

$$G(s) = \frac{b}{s+a} = \frac{k_{dc}}{\tau s + 1} \tag{۲}$$

که پارامترهای k_{dc} و τ رفتار این سیستم مرتبه اول را کاملا تعریف می‌کنند.

بهره‌ی DC

بهره‌ی DC یا k_{dc} نسبت اندازه‌ی حالت ماندگار پاسخ پله به اندازه‌ی ورودی پله می‌باشد. برای توابع تبدیل پایدار، با استفاده از **قضیه مقدار نهایی** می‌توان نشان داد که بهره‌ی DC همان مقدار تابع تبدیل در $s = 0$ می‌باشد. برای سیستم مرتبه‌ی اول نشان داده شده بهره DC برابر $k_{dc} = b/a$ می‌باشد.

ثابت زمانی

ثابت زمانی سیستم مرتبه اول برابر $T_c = \tau = 1/a$ می‌باشد که برابر زمانی است که پاسخ سیستم به ۶۳٪ مقدار حالت ماندگار خود برای ورودی پله (از شرایط اولیه صفر) می‌رسد. برای سیستم‌های بدون پاسخ، ثابت زمانی به صورت مقدار زمانی که پاسخ سیستم به ۳۷٪ مقدار اولیه کاهش می‌یابد تعریف می‌شود. به طور کلی مقدار ثابت زمانی، مقیاس زمانی که در آن دینامیک سیستم حائز اهمیت می‌باشد را مشخص می‌کند.

قطب‌ها/صفرها

سیستم‌های مرتبه اول دارای یک قطب حقیقی که در اینجا $s = -a$ است می‌باشند. بنابراین در صورتی که مقدار a مثبت باشد سیستم پایدار و مقدار a منفی باشد سیستم ناپایدار است. سیستم مرتبه اول استاندارد دارای صفر نمی‌باشد.

پاسخ پله[۱۲]

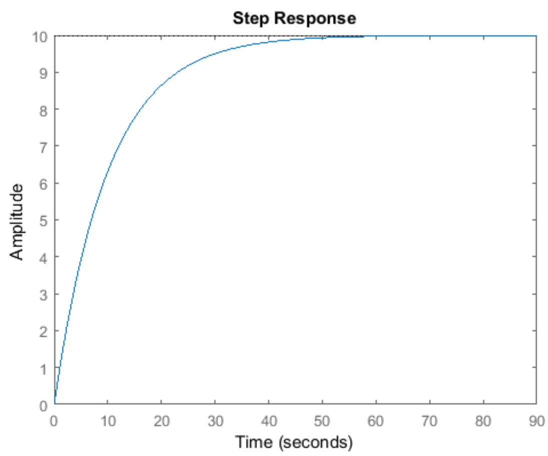
با استفاده از دستور زیر می‌توان در متلب، پاسخ زمانی سیستم به ورودی پله با اندازه u را به دست آورد:

```
k_dc = 5;
```

[۱۲] Step Response

```
Tc = 10;

u = 2;



s = tf('s');
G = k_dc/(Tc*s+1)

step(u*G)
```

```
G =


     5

  --------

  10 s + 1



Continuous-time transfer function.
```



نکته: نرم افزار متلب دارای یک رابط کاربری گرافیکی قدرتمند برای تحلیل سیستم‌های LTI می‌باشد که با استفاده از دستور `linearSystemAnalyzer('step',G)` می‌توان از آن بهره برد.

زمان نشست[۱۳]

زمان نشست T_s مدت زمانی است که خروجی سیستم در ناحیه مشخصی (برای مثلا ۲٪) از حالت ماندگار برای ورودی پله قرار می‌گیرد. زمان نشست برای سیستم مرتبه اول برای تلرانس‌های مختلف در جدول زیر آورده شده است. لازم به ذکر است که هرچه تلرانس کوچکتر باشد، زمان بیشتری لازم است تا پاسخ سیستم در ناحیه مناسب قرار بگیرد و در نتیجه زمان نشست افزایش می‌یابد.

10%	5%	2%	1%

۱۳ Settling Time

Ts=2.3/a=2.3Tc	Ts=3/a=3Tc	Ts=3.9/a=3.9Tc	Ts=4.6/a=4.6Tc

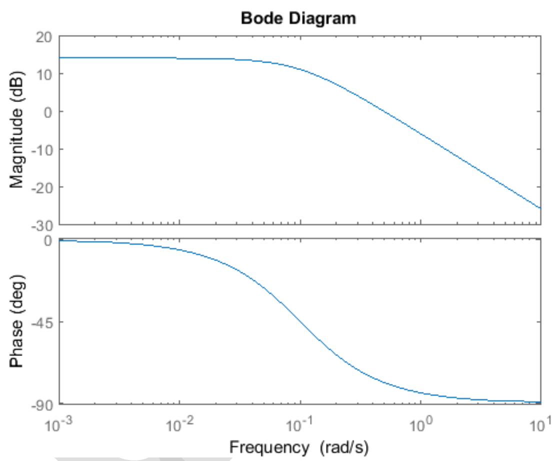
زمان نمو (رشد)[۱۴]

زمان نمو T_r مدت زمانی است که لازم است تا خروجی سیستم از مقدار پایین $x\%$ به مقدار بالاتر $y\%$ مقدار حالت ماندگار برسد. برای سیستم‌های مرتبه اول این مقادیر به ترتیب ۱۰٪ و ۹۰٪ می‌باشد.

دیاگرام بودی

دیاگرام بودی، اندازه و فاز پاسخ فرکانسی سیستم $(G(j\omega))$ را نسبت به فرکانس (ω) رسم می‌نماید. در متلب می‌توان با استفاده از دستور bode(G)، دیاگرام بودی برای سیستم G را تشکیل داد.

```
bode(G)
```



در این مبحث نیز می‌توان از رابط گرافیکی کاربری `linearSystemAnalyzer('bode',G)` برای تحلیل سیستم استفاده نمود.

دیاگرام بودی از مقیاس لگاریتمی استفاده می‌کند که سبب می‌شود تا مقادیر بزرگتر نیز قابل مشاهده باشند. همچنین در این دیاگرام اندازه با **واحد دسیبل (dB)** لگاریتمی نمایش داده می‌شود:

$$M_{dB} = 20\log_{10}(M) \quad (۳)$$

مقیاس دسیبل در محور فرکانس، این امکان را ایجاد می‌کند تا مقادیر بزرگتر نیز در یک نمودار قابل مشاهده باشند. همچنین همانطور که در تمرین‌ها خواهیم دید، وقتی که اجزای سیستم و کنترل به طور سری قرار بگیرند، تابع تبدیل کلی سیستم برابر با حاصل ضرب تمامی توابع تبدیل است. در این صورت با استفاده از مقیاس دسیبل، اندازه‌ی کل سیستم برابر با مجموع اندازه‌های هر یک از توابع تبدیل می‌باشد. همچنین فاز کل سیستم نیز برابر با مجموع فاز هر یک از توابع تبدیل است.

اندازه در فرکانس پایین در دیاگرام بودی مرتبه اول برابر $20\log(k_{dc})$ است. منحنی اندازه در فرکانس مساوی با مقدار مطلق قطب (یعنی $\omega = a$) دارای یک خمیدگی می‌باشد و با افزایش هر دهه از فرکانس، ۲۰ دسیبل کاهش می‌یابد (شیب منحنی برابر $-20\ dB/decade$ است). منحنی فاز در فرکانس پایین به صورت مجانبی به صفر میل می‌کند و در فرکانس‌های بالا به -90 درجه میل می‌کند. در فرکانس‌های بین $0.1a$ و $10a$ مقدار فاز تقریبا به ازای افزایش هر دهه از فرکانس، -45 درجه کاهش پیدا می‌کند ($-45\ degrees/decade$).

---
[۱۴] Rise Time

در بخش طراحی کنترلر با استفاده از روش‌های فرکانس خواهیم دید که چطور با استفاده از دیاگرام بودی، پایداری حلقه بسته و عملکرد فیدبک سیستم را محاسبه می‌کنیم.

سیستم‌های مرتبه دوم

در این کتاب اغلب به سیستم‌های مرتبه دوم بر خواهیم خورد که این سیستم‌ها از ساده‌ترین نوع سیستم‌های دینامیکی با رفتار نوسانی می‌باشند. مثال‌هایی که برای سیستم مرتبه دوم به کار می‌رود سیستم جرم-فنر-دمپر می‌باشد. در واقع بسیاری از سیستم‌های مرتبه بالاتر را می‌توان برای تحلیل ساده تر با سیستم‌های مرتبه دوم تقریب زد.

فرم کانونی معادلات سیستم مرتبه دوم عبارتست از:

$$m\ddot{y} + b\dot{y} + ky = f(t) \quad \text{یا} \quad \ddot{y} + 2\zeta\omega_n\dot{y} + \omega_n^2 y = k_{dc}\omega_n^2 u \tag{۴}$$

همچنین فرم کانونی تابع تبدیل مرتبه دوم که دارای دو قطب و بدون صفر می‌باشد به شکل زیر است:

$$G(s) = \frac{1}{ms^2 + bs + k} = \frac{k_{dc}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{۵}$$

پارامترهای k_{dc}، ζ و ω_n رفتار سیستم مرتبه دوم را مشخص می‌کنند.

بهره DC

بهره DC (k_{dc}) در اینجا نیز نسبت اندازه پاسخ پله حالت ماندگار به اندازه ورودی پله می‌باشد که برای سیستم‌های پایدار این مقدار از قرار دادن $s = 0$ در تابع تبدیل سیستم به دست می‌آید. فرم بهره DC به شکل زیر است:

$$k_{dc} = \frac{1}{k} \tag{۶}$$

ضریب میرایی

ضریب میرایی ζ مقداری بدون واحد است که نرخ ضعیف شدن نوسانات پاسخ سیستم که ناشی از اثرات اصطکاک ویسکوز و مقاومت الکتریکی می‌باشد را توصیف می‌کند. با توجه به تعاریف بالا داریم:

$$\zeta = \frac{b}{2\sqrt{km}} \tag{۷}$$

فرکانس طبیعی

فرکانسی که در آن بدون وجود میرایی در سیستم ($\zeta = 0$)، سیستم شروع به نوسان می‌کند به عنوان فرکانس طبیعی ω_n شناخته می‌شود:

$$\omega_n = \sqrt{\frac{k}{m}} \tag{۸}$$

قطب‌ها/صفرها

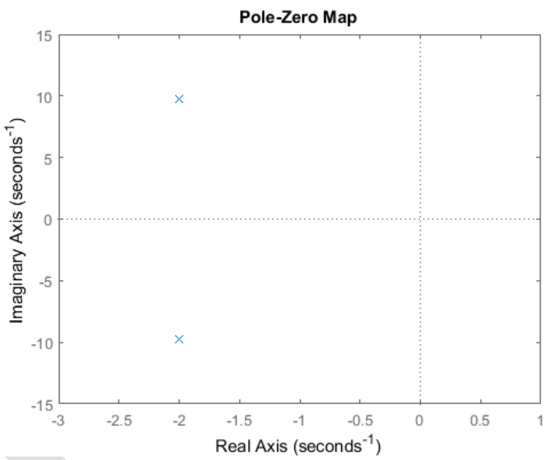
تابع تبدیل کانونی مرتبه دوم دارای دو قطب در مقادیر زیر می‌باشد:

$$s_p = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} \tag{۹}$$

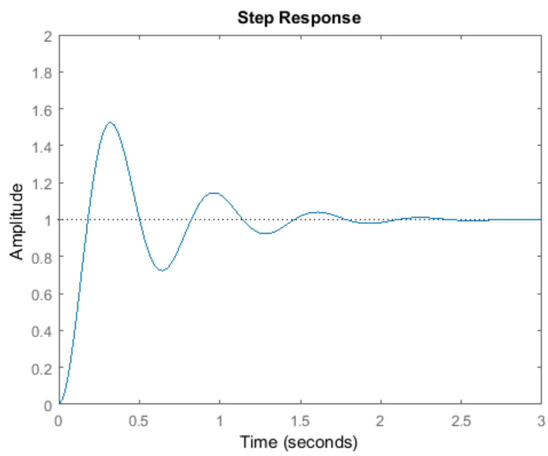
سیستم‌های زیر میرا

اگر $\zeta < 1$ آنگاه سیستم زیر میرا[15] می‌باشد. در این مورد هر دو قطب دارای مقادیر مختلطی می‌باشند که دارای بخش حقیقی منفی می‌باشد. در این حالت سیستم پایدار بوده اما در هنگام رسیدن به حالت ماندگار دارای نوسانانی می‌باشد که این نوسانات با فرکانس طبیعی میرا ($\omega_d = \omega_n\sqrt{1-\zeta^2}$) ایجاد می‌شوند.

```
k_dc = 1;

w_n = 10;

zeta = 0.2;



s = tf('s');
G1 = k_dc*w_n^2/(s^2 + 2*zeta*w_n*s + w_n^2);

pzmap(G1)
axis([-3 1 -15 15])
```




```
step(G1)

axis([0 3 0 2])
```



[15] Underdamped

زمان نشست

زمان نشست T_s زمان لازم برای خروجی سیستم است تا به درصد مشخصی از مقدار حالت ماندگار برای ورودی پله برسد. برای یک سیستم کانونی مرتبه دوم زیر میرا، زمان نشست را می‌توان با معادله زیر تقریب زد:

$$T_s = \frac{-\ln(tolerance\ fraction)}{\zeta\omega_n} \qquad (۱۰)$$

زمان نشست برای تلرانس‌های رایج در جدول زیر ارائه شده است:

10%	5%	2%	1%
$T_s = \frac{2.3}{\zeta\omega_n}$	$T_s = \frac{3}{\zeta\omega_n}$	$T_s = \frac{3.9}{\zeta\omega_n}$	$T_s = \frac{4.6}{\zeta\omega_n}$

درصد فراجهش[۱۶]

درصد فراجهش مقدار درصدی است که پاسخ پله‌ی سیستم از مقدار حالت ماندگار بیشتر می‌شود. برای یک سیستم مرتبه دو زیر میرا، درصد فراجهش M_p مستقیما به ضریب میرایی وابسته است که در معادله زیر مشاهده می‌نمایید. مقدار M_p یک عدد اعشاری است که مقدار ۱ متناسب با ۱۰۰٪ فراجهش می‌باشد:

$$M_p = e^{\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} \qquad (۱۱)$$

برای سیستم مرتبه دو زیر میرا، زمان نشست T_s، زمان نمو و درصد فراجهش M_p به ضریب میرایی و فرکانس طبیعی وابسته است:

$$T_s \approx \frac{4.6}{\zeta\omega_n} \qquad (۱۲)$$

$$T_r = \frac{1.8}{\omega_n} \qquad (۱۳)$$

$$\zeta = \frac{-\ln(M_p)}{\sqrt{\pi^2 + \ln^2(M_p)}} \qquad (۱۴)$$

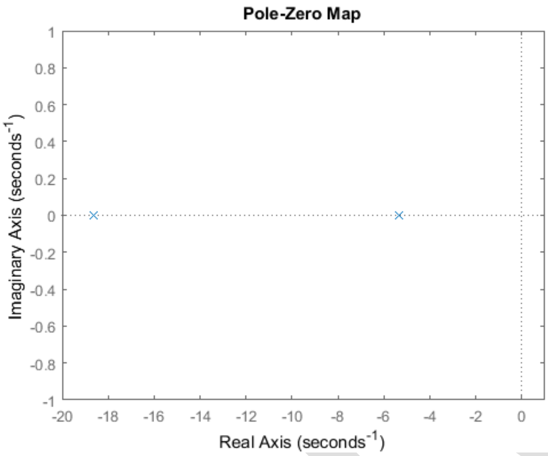
سیستم‌های فوق میرا[۱۷]

اگر $\zeta > 1$ آنگاه سیستم فوق میرا می‌باشد. در این صورت هر دو قطب حقیقی و منفی می‌باشند در نتیجه سیستم پایدار و بدون نوسان است. پاسخ پله و نقشه‌ی قطب-صفر یک سیستم فوق میرا به شکل زیر رسم می‌شود:

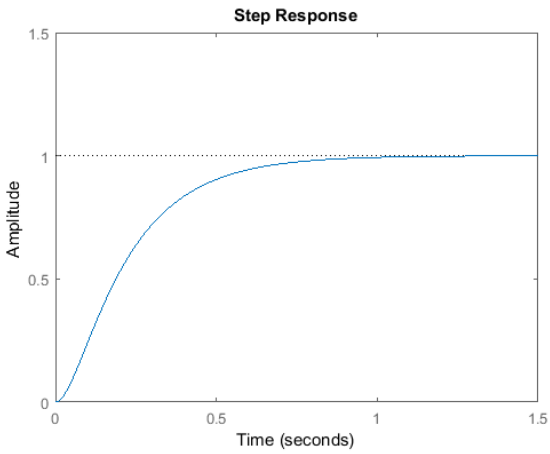
```
zeta = 1.2;

G2 = k_dc*w_n^2/(s^2 + 2*zeta*w_n*s + w_n^2);

pzmap(G2)
```

[۱۶] Percent Overshoot
[۱۷] Overdamped

```
axis([-20 1 -1 1])
```




```
step(G2)

axis([0 1.5 0 1.5])
```



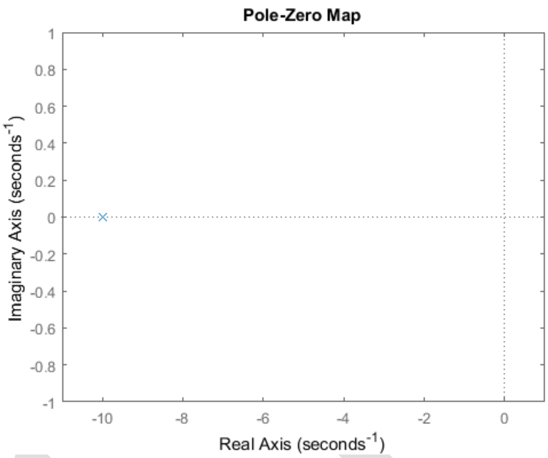
سیستم‌های میرای بحرانی[۱۸]

اگر $\zeta = 1$ آنگاه سیستم میرای بحرانی می‌باشد. در این صورت هر دو قطب حقیقی بوده و دارای اندازه یکسان $s_p = -\zeta\omega_n$ می‌باشند. برای سیستم مرتبه‌ی دوم کانونی، سریعترین زمان نشست در حالت میرایی بحرانی به دست می‌آید. با تغییر ضریب میرایی به مقدار ۱ و رسم دوباره‌ی پاسخ پله و نقشه‌ی قطب-صفر داریم:

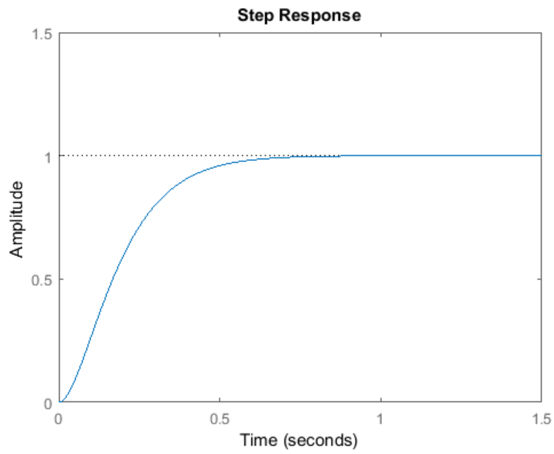
```
zeta = 1;


G3 = k_dc*w_n^2/(s^2 + 2*zeta*w_n*s + w_n^2);
```

[۱۸] Critically-Damped

```
pzmap(G3)

axis([-11 1 -1 1])
```




```
step(G3)

axis([0 1.5 0 1.5])
```

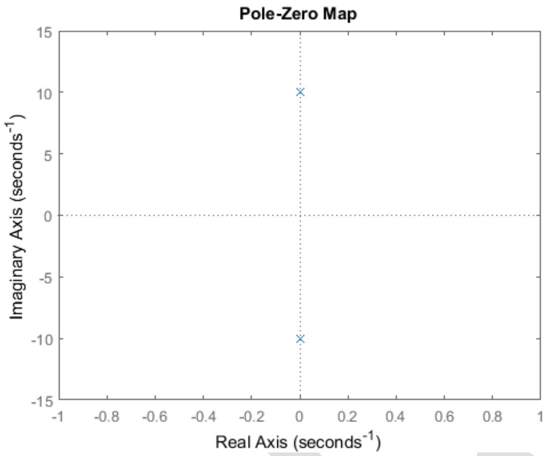


سیستم‌های بدون میرایی

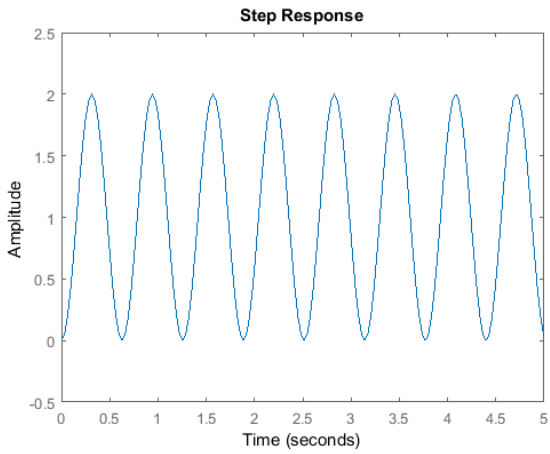
اگر $\zeta = 0$ آنگاه سیستم بدون میرایی می‌باشد. در این حالت قطب‌ها کاملا موهومی می‌باشند و در نتیجه سیستم پایدار مرزی بوده و پاسخ پله همواره نوسانی خواهد بود.

```
zeta = 0;



G4 = k_dc*w_n^2/(s^2 + 2*zeta*w_n*s + w_n^2);



pzmap(G4)
```

```
axis([-1 1 -15 15])
```



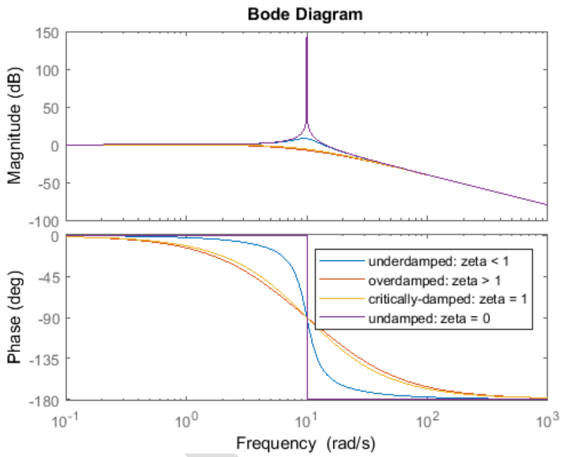

```
step(G4)

axis([0 5 -0.5 2.5])
```



دیاگرام بودی

در ادامه به رسم دیاگرام اندازه و فاز بودی برای تمامی شرایط میرایی یک سیستم مرتبه دوم می‌پردازیم:

```
bode(G1,G2,G3,G4)

legend('underdamped:  zeta  <  1','overdamped:  zeta  >  1','critically-damped:  zeta  =
1','undamped: zeta = 0')
```



منحنی اندازه دیاگرام بودی برای سیستم مرتبه دوم در هر دهه به اندازه $-40\ dB$ افت می‌کند و همچنین فاز نسبی از 0 تا -180 درجه تغییر می‌کند. همچنین برای سیستم‌های زیر میرا یک قله‌ی تشدید نزدیک فرکانس طبیعی $\omega_n = 10\ rad/s$ نیز داریم. ابعاد و تندی این قله بستگی به میرایی سیستم داشته و با یک فاکتور کیفیت یا Q-Factor مشخص می‌گردد که در زیر تعریف شده است. فاکتور کیفیت مشخصه‌ی مهمی در مباحث پردازش سیگنال می‌باشد:

$$Q = \frac{1}{2\zeta} \qquad (۱۵)$$

بخش سوم: طراحی کنترلر PID

فهرست مطالب بخش

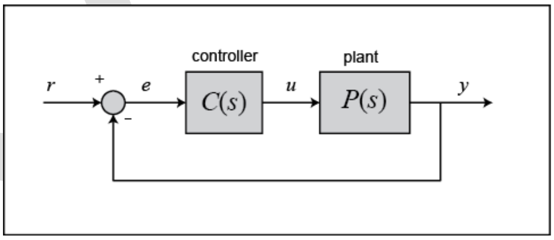
در این قسمت به معرفی یک ساختار جبران‌ساز فیدبک ساده اما توانمند که با نام کنترلر تناسبی-انتگرالی-مشتقی (PID) شناخته می‌شود می‌پردازیم. کنترلر PID به علت قابل درک بودن و کاربردی بودن به طور گسترده مورد استفاده قرار می‌گیرد. یکی از جذابیت‌های کنترلر PID این است که تمامی مهندسین مفاهیم انتگرال‌گیری و مشتق‌گیری را درک کرده و در نتیجه حتی بدون درک عمیقی از تئوری کنترل، قادر به پیاده سازی آن هستند. علاوه بر آن اگرچه کنترلر PID جبران‌ساز ساده‌ای می‌باشد اما به علت اینکه تاریخچه‌ی سیستم را ذخیره کرده (از طریق انتگرال‌گیری) و رفتار آینده‌ی سیستم را پیش بینی می‌کند (از طریق مشتق‌گیری) کنترلر پیچیده‌ای محسوب می‌گردد. در این قسمت تاثیر هر یک از پارامترهای PID را بر روی دینامیک سیستم حلقه بسته بررسی کرده و روش استفاده از یک کنترلر PID برای ارتقای عملکرد یک سیستم را نشان می‌دهیم.

دستورهای کلیدی متلب در این بخش:

```
tf, step, pid, feedback, pidtune
```

معرفی PID

در این فصل، سیستم فیدبک واحد زیر را در نظر می‌گیریم:



خروجی کنترلر PID که همان ورودی به سیستم می‌باشد را می‌توان بر حسب فیدبک واحد در حوزه‌ی زمان به شکل زیر به دست آورد:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \qquad (۱)$$

ابتدا به طرز کار کنترلر PID در سیستم حلقه بسته که در شماتیک بالا نشان داده شد نگاهی می‌اندازیم. متغیر e بیانگر خطای ردیابی است که معادل با اختلاف بین خروجی مطلوب (r) و خروجی واقعی (y) می‌باشد. سیگنال خطا (e) به کنترلر PID وارد شده و در آنجا کنترلر، مشتق و انتگرال این سیگنال خطا را بر حسب زمان حساب می‌کند. سیگنال کنترلی

(u) وارد شده به سیستم برابر با بهره تناسبی (K_p) ضریدر اندازه‌ی خطا، بعلاوه‌ی بهره انتگرال (K_i) ضریدر انتگرال خطا بعلاوه‌ی بهره مشتق (K_d) ضریدر مشتق خطا می‌باشد.

این سیگنال کنترل (u) به سیستم خورانده شده و خروجی جدید (y) به دست می‌آید. سپس خروجی جدید (y) بازخورانده شده و با مرجع مقایسه می‌شود تا سیگنال خطای جدید (e) به دست آید. کنترلر این سیگنال خطای جدید را گرفته و ورودی کنترل را بروزرسانی می‌کند. این فرآیند تا وقتی که کنترلر در جریان باشد ادامه دارد.

تابع تبدیل کنترلر PID با گرفتن تبدیل لاپلاس از معادله (۱) به دست می‌آید:

$$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \qquad (۲)$$

برای تعریف کنترلر PID در متلب مستقیما از مدل تابع تبدیل استفاده می‌کنیم، برای مثال:

```
Kp = 1;

Ki = 1;

Kd = 1;



s = tf('s');
C = Kp + Ki/s + Kd*s
```

```
C =



  s^2 + s + 1

  -----------

       s



Continuous-time transfer function.
```

متناوبا می‌توان از دستور `pid` در متلب برای ایجاد کنترلر استفاده کرد:

```
C = pid(Kp,Ki,Kd)
```

```
C =



             1

  Kp + Ki * --- + Kd * s

             s
```

```
with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
```

برای اطمینان بیشتر pid به دست آمده را به فرم تابع تبدیل درآورده تا ببینیم نتیجه‌ی یکسانی را می‌دهد:

```
tf(C)
```

```
ans =

  s^2 + s + 1
  -----------
       s

Continuous-time transfer function.
```

مشخصه‌های جملات P، I و D

افزایش بهره تناسبی K_p سبب افزایش سیگنال کنترل با توجه به خطای همان لحظه می‌شود. در این صورت کنترلر تلاش بیشتری برای کاهش خطا نشان داده که باعث عکس‌العمل سریعتر سیستم حلقه بسته خواهد شد اما فراجهش را نیز افزایش می‌دهد. تاثیر دیگر افزایش K_p تنها کاهش خطای حالت ماندگار است اما آنرا به طور کامل حذف نمی‌کند.

استفاده از جمله‌ی مشتقی کنترلر یا K_d، قابلیت پیش‌بینی خطا را زیادتر می‌کند. در کنترل ساده‌ی تناسبی، اگر K_p در زمان عملکرد کنترلر ثابت باشد، تنها راه افزایش کنترل، افزایش خطا می‌باشد. اما با کنترل مشتقی، سیگنال کنترل با بیشتر شدن شیب خطا، حتی اگر اندازه‌ی خطا هم کم باشد، بیشتر خواهد شد. این پیش‌بینی سبب افزایش میرایی سیستم شده و در نتیجه فراجهش را کاهش می‌دهد. اضافه شدن جمله‌ی مشتقی تاثیری بر روی خطای حالت ماندگار ندارد.

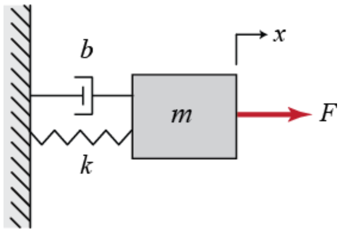
استفاده از جمله انتگرالی کنترلر یا K_i به کاهش خطای حالت ماندگار کمک می‌کند. اگر خطای ماندگار پایداری وجود داشته باشد، انتگرال‌گیر با افزایش سیگنال کنترلی آنرا کاهش خواهد داد. عیب جمله‌ی انتگرالی این است که سیستم را نوسانی می‌کند زیرا در صورت تغییر علامت سیگنال خطا، مدت زمانی طول می‌کشد تا انتگرال‌گیر به حالت قبل بازگردد.

اثر کلی هر یک از پارامترهای کنترلر (K_p, K_d, K_i) بر روی یک سیستم حلقه بسته در جدول زیر آمده است. دقت کنید که این تاثیرات در بسیاری از موارد درست بوده اما برای تمامی سیستم‌ها صادق نمی‌باشد. اگر تاثیر واقعی هر یک از پارامترها را می‌خواهید بدانید، لازم است تا تحلیل‌های بیشتری انجام داده یا بر روی سیستم واقعی امتحان نمایید.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small Change	Decrease	Decrease	No Change

مثال

فرض کنید سیستم جرم و فنر و دمپر زیر را داریم:



معادله‌ی حاکم بر این سیستم عبارتست از:

$$m\ddot{x} + b\dot{x} + kx = F \qquad (۳)$$

با گرفتن تبدیل لاپلاس از معادله‌ی حاکم داریم:

$$ms^2X(s) + bsX(s) + kX(s) = F(s) \qquad (۴)$$

تابع تبدیل بین ورودی نیرو $F(s)$ و خروجی جابجایی $X(s)$ برابر است با:

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k} \qquad (۵)$$

برای پارامترهای سیستم مقادیر زیر را در نظر می‌گیریم:

```
m = 1 kg

b = 10 N s/m

k = 20 N/m

F = 1 N
```

با جایگذاری این مقادیر در تابع تبدیل خواهیم داشت:

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20} \qquad (۶)$$

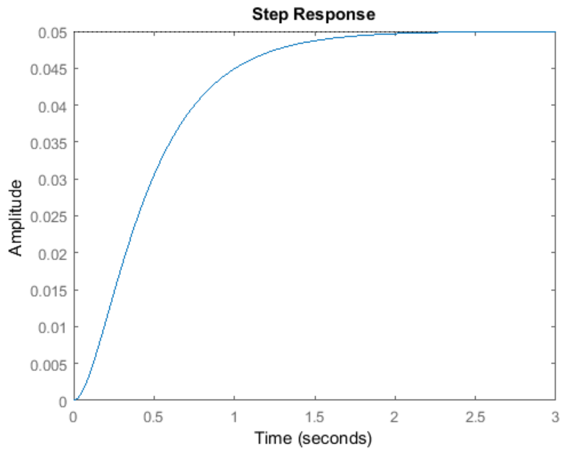
هدف از این مثال این است که تاثیر جملات K_p، K_i و K_d را برای دستیابی به موارد زیر نشان دهیم:

- زمان نمو سریع
- فراجهش مینیمم
- خطای حالت ماندگار صفر

پاسخ پله‌ی حلقه باز

ابتدا پاسخ پله‌ی سیستم حلقه باز را مشاهده می‌کنیم. یک امفایل ساخته و کد زیر را در آن اجرا کنید:

```
s = tf('s');
P = 1/(s^2 + 10*s + 20);
step(P)
```



بهره DC تابع تبدیل سیستم برابر $\frac{1}{20}$ است پس مقدار نهایی خروجی برای ورودی پله‌ی واحد برابر 0.05 می‌باشد. در این صورت خطای حالت ماندگار برابر ۰/۹۵ است که بسیار بزرگ است. علاوه بر آن زمان نمو حدود ۱ ثانیه و زمان نشست حدود ۱/۵ ثانیه می‌باشد. در ادامه کنترلی را می‌سازیم که زمان نمو را کاهش، زمان نشست را کاهش و خطای حالت ماندگار را حذف کند.

کنترل تناسبی

از جدول بالا می‌دانیم که با افزایش K_p زمان نمو کاهش، فراجهش افزایش و خطای حالت ماندگار کاهش پیدا می‌کند.

تابع تبدیل حلقه بسته برای سیستم با فیدبک واحد و کنترلر تناسبی به شکل زیر است که در آن $X(s)$ خروجی (برابر $Y(s)$) و ورودی مرجع $R(s)$ ورودی آن می‌باشد:

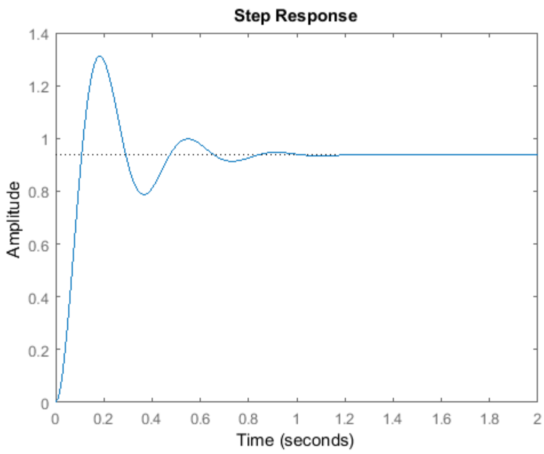
$$T(s) = \frac{X(s)}{R(s)} = \frac{K_p}{s^2 + 10s + (20 + K_p)} \tag{۷}$$

بهره K_p را برابر ۳۰۰ در نظر گرفته و امفایل را به شکل زیر تغییر دهید:

```
Kp = 300;

C = pid(Kp)

T = feedback(C*P,1)



t = 0:0.01:2;

step(T,t)
```

```
C =


  Kp = 300
```

```
P-only controller.

T =

        300
  ----------------
  s^2 + 10 s + 320

Continuous-time transfer function.
```



نمودار بالا نشان می‌دهد که کنترلر تناسبی هم زمان نمو و هم خطای حالت ماندگار را کاهش، فراجهش را افزایش و زمان نشست را به مقدار کمی کاهش داده است.

کنترل تناسبی-مشتقی

حال به کنترلر PD می‌پردازیم. از جدول بالا متوجه شدیم که اضافه شدن کنترل مشتقی K_d فراجهش و زمان نشست را کاهش می‌دهد. تابع تبدیل حلقه بسته‌ی سیستم داده شده با کنترلر PD عبارتست از:

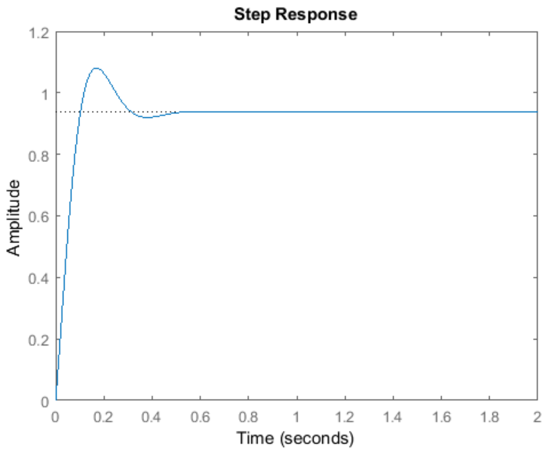
$$T(s) = \frac{X(s)}{R(s)} = \frac{K_d s + K_p}{s^2 + (10 + K_d)s + (20 + K_p)} \tag{۸}$$

مقدار K_p را مانند قبل و برابر ۳۰۰ و مقدار K_d را برابر ۱۰ در نظر بگیرید. این دستورها را در ام‌فایل وارد کرده و آنرا اجرا کنید:

```
Kp = 300;
Kd = 10;
C = pid(Kp,0,Kd)
```

```
T = feedback(C*P,1)

t = 0:0.01:2;

step(T,t)
```

```
C =

  Kp + Kd * s

  with Kp = 300, Kd = 10

Continuous-time PD controller in parallel form.

T =

     10 s + 300
  ----------------
  s^2 + 20 s + 320

Continuous-time transfer function.
```



با توجه به نمودار به دست آمده می‌توان فهمید که اضافه شدن جمله مشتقی، فراجهش و زمان نشست را کاهش داده و تاثیر ناچیزی بر روی زمان نمو و خطای حالت ماندگار داشته است.

کنترل تناسبی-انتگرالی

قبل از پرداختن به کنترل PID، کنترل PI را بررسی می‌کنیم. از جدول آورده شده می‌دانیم که اضافه کردن جمله انتگرالی K_i تمایل سیستم به کاهش زمان نمو، افزایش فراجهش و زمان نشست و کاهش خطای حالت ماندگار بیشتر می‌شود. برای سیستم داده شده، تابع تبدیل حلقه بسته با کنترلر PI برابر است با:

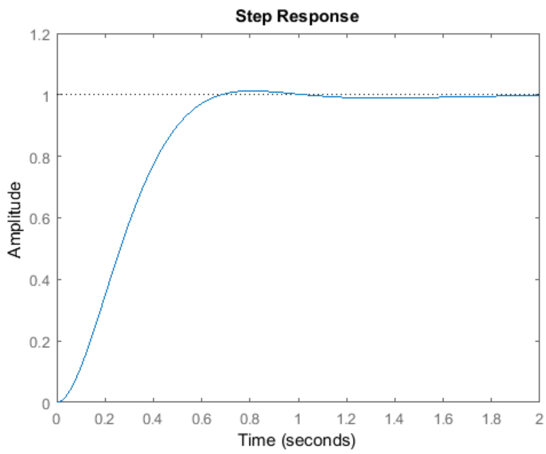
$$T(s) = \frac{X(s)}{R(s)} = \frac{K_p s + K_i}{s^3 + 10s^2 + (20 + K_p)s + K_i} \qquad (۹)$$

پارامتر K_p را به مقدار ۳۰ کاهش داده و K_i را برابر ۷۰ قرار می‌دهیم. در یک ام‌فایل جدید دستورات زیر را وارد نمایید:

```
Kp = 30;

Ki = 70;

C = pid(Kp,Ki)

T = feedback(C*P,1)



t = 0:0.01:2;

step(T,t)
```

```
C =


             1

  Kp + Ki * ---

             s
```

```
  with Kp = 30, Ki = 70

Continuous-time PI controller in parallel form.

T =

         30 s + 70
  ------------------------
  s^3 + 10 s^2 + 50 s + 70

Continuous-time transfer function.
```



با اجرای دستورات گفته شده، نمودار بالا به دست می‌آید. به علت اینکه کنترلر انتگرالی نیز زمان نشست را کاهش و فراجهش را افزایش می‌دهد و کنترلر تناسبی نیز همین تاثیر را دارد بهره‌ی تناسبی K_p را کاهش داده‌ایم. پاسخ به دست آمده نشان می‌دهد که کنترلر انتگرالی خطای حالت ماندگار را برای این مسئله به طور کامل حذف کرده است.

کنترل تناسبی-انتگرالی-مشتقی

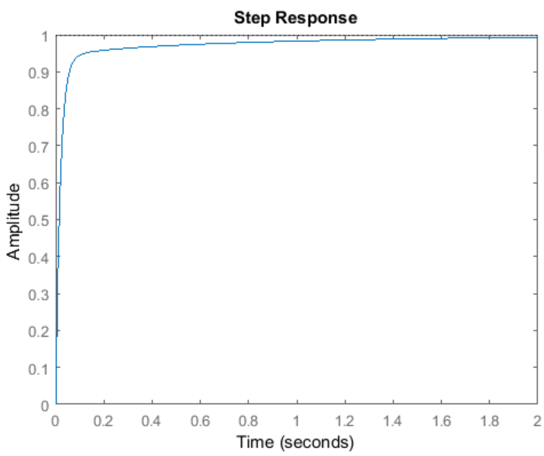
حال به بررسی کنترل PID می‌پردازیم. تابع تبدیل حلقه بسته برای سیستم داده شده به همراه کنترلر PID به شکل زیر است:

$$T(s) = \frac{X(s)}{R(s)} = \frac{K_d s^2 + K_p s + K_i}{s^3 + (10 + K_d)s^2 + \left(20 + K_p\right)s + K_i} \qquad (۱۰)$$

بعد از چند بار تنظیم این ضرایب، بهره‌های $K_p = 350$، $K_i = 300$ و $K_d = 50$ پاسخ مطلوب را نتیجه می‌دهند. برای اطمینان دستورات زیر را در یک ام‌فایل وارد کرده و آنرا اجرا کنید. در اینصورت پاسخ پله زیر را دریافت می‌کنید:

```
Kp = 350;

Ki = 300;

Kd = 50;

C = pid(Kp,Ki,Kd)

T = feedback(C*P,1);



t = 0:0.01:2;

step(T,t)
```

```
C =


             1

  Kp + Ki * --- + Kd * s

             s



  with Kp = 350, Ki = 300, Kd = 50



Continuous-time PID controller in parallel form.
```



با استفاده از این کنترلر، سیستم حلقه بسته‌ای را طراحی کرده‌ایم که بدون فراجهش، دارای زمان نمو سریع و بدون خطای حالت ماندگار می‌باشد.

نکات کلی برای طراحی کنترلر PID

برای به دست آوردن پاسخ مطلوب، در هنگام طراحی PID برای یک سیستم نکات زیر را مد نظر قرار دهید:

۱. پاسخ حلقه بسته را به دست آورده و مواردی را که نیاز به اصلاح دارند مشخص کنید.
۲. کنترل تناسبی را برای بهبود زمان نمو اضافه کنید.
۳. کنترل مشتقی را برای کاهش فراجهش اضافه کنید.
۴. کنترل انتگرالی را برای کاهش خطای حالت ماندگار اضافه کنید.
۵. هر کدام از بهره‌های K_p، K_i و K_d به نحوی تنظیم کنید تا پاسخ کلی مطلوب به دست آید. برای پیدا کردن رفتار مناسب هر کنترلر، از جدول گفته شده استفاده کنید.

در نهایت در نظر داشته باشید که لازم نیست در یک سیستم از هر سه قسمت کنترلر (تناسبی، مشتقی و انتگرالی) استفاده نمایید. برای مثال اگر یک کنترلر PI نیازهای خواسته شده را برآورده می‌کند لازم نیست که از کنترلر مشتقی استفاده شود. کنترلر را تا حد امکان ساده طراحی کنید.

تنظیم خودکار PID

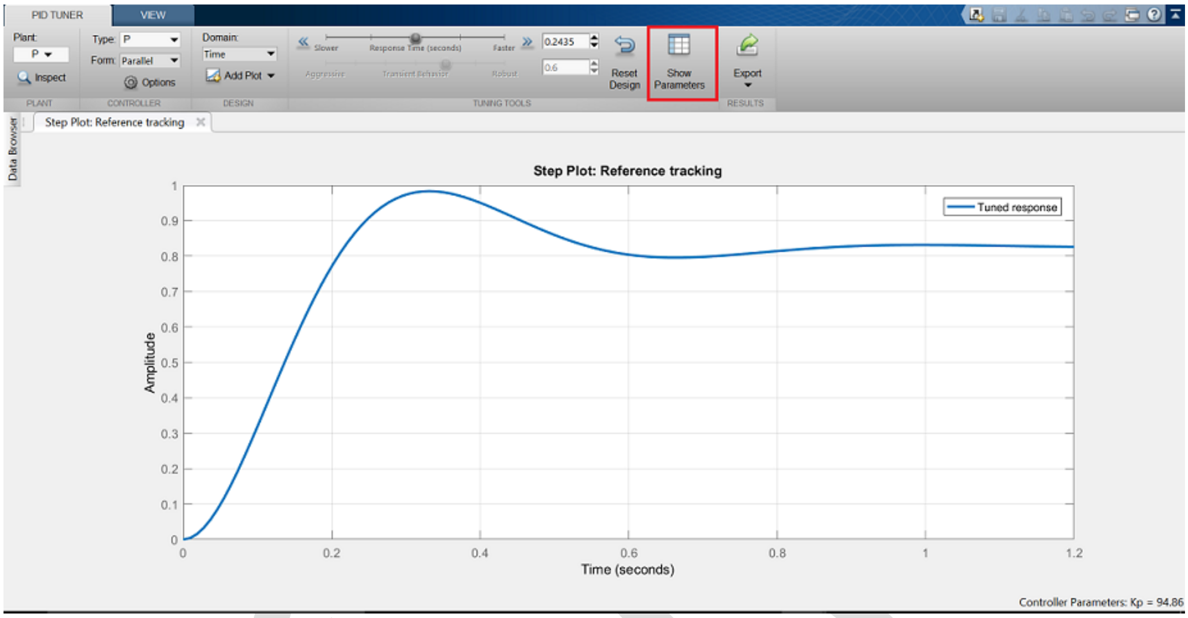
متلب ابزار تنظیم خودکار بهره‌های بهینه‌ی PID را فراهم کرده است که ما را از فرآیند سعی و خطا که در قسمت قبل گفته شد بی‌نیاز می‌کند. با دستور `pidtune` مستقیما به الگوریتم تنظیم و با دستور `pidTuner` به رابط کاربری گرافیکی دسترسی پیدا می‌کنید.

الگوریتم تنظیم خودکار متلب، بهره‌های PID را برای عملکرد متعادل (زمان پاسخ، پهنای باند) و مقاوم بودن (حدود پایداری) سیستم انتخاب می‌کند. به طور پیش فرض این الگوریتم، طراحی را برای حد فاز ۶۰ درجه انجام می‌دهد.

برای بررسی این ابزار خودکار متلب، یک کنترلر تناسبی برای سیستم جرم-فنر-دمپر با دستور زیر ایجاد می‌نماییم. در دستور نشان داده شده، P قبلا به عنوان مدل سیستم تعریف شده است و p مشخص می‌کند که کنترلر تنها تناسبی باشد.

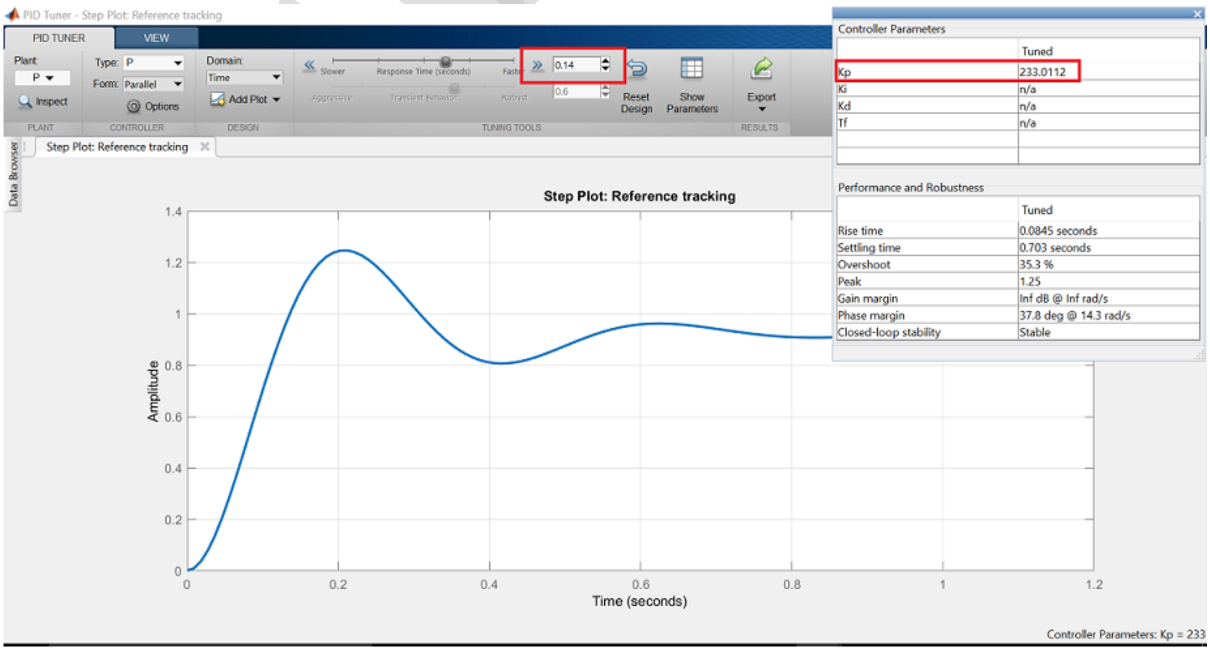
```
pidTuner(P,'p')
```

پنجره pidTuner مانند تصویر زیر نمایان می‌شود:



دقت نمایید که زمان پاسخ نشان داده شده از کنترلر تناسبی که به صورت دستی طراحی کردیم آهسته‌تر می‌باشد. حال بر روی کلید **نمایش پارامترها (Show Parameters)** در بالا و راست صفحه کلید کنید. همانطور که انتظار می‌رود بهره‌ی تناسبی K_p از مقدار انتخاب شده به صورت دستی کمتر است $K_p = 94.86 < 300$.

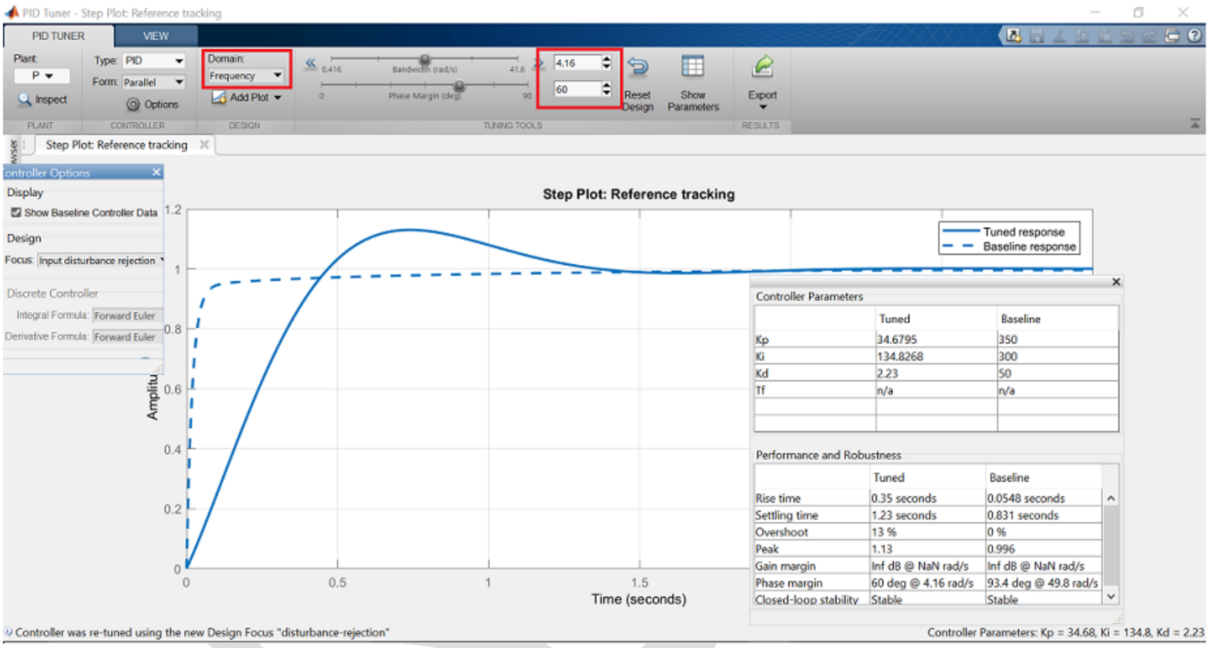
حال می‌توانیم پارامترهای کنترلر را تغییر دهیم و نتیجه را آناً در پنجره GUI مشاهده نماییم. مانند شکل، لغزنده‌ی **زمان پاسخ (Response Time)** را روی مقدار ۰٫۱۴ ثانیه قرار دهید. این کار سبب می‌شود تا زمان پاسخ سرعت گرفته و همانطور که مشاهده می‌شود مقدار K_p به سمت مقدار دستی نزدیک می‌شود. در این پنجره پارامترهای عملکرد و مقاوم بودن دیگری نیز مشاهده می‌شود. دقت کنید که قبل از تغییر لغزنده‌ی زمان پاسخ، مقدار حد فاز برابر ۶۰ تنظیم شده بود. این مقدار به طور پیش‌فرض توسط pidTuner در نظر گرفته شده که سبب تعادل خوبی بین عملکرد و مقاوم بودن سیستم می‌شود.



در قدم بعد به طراحی کنترلر PID برای سیستم می‌پردازیم. با مشخص کردن کنترلر طراحی شده از قبل (C) به عنوان ورودی دوم در دستور `pidTuner`، این ابزار کنترلر PID دیگری را طراحی کرده و پاسخ سیستم را با کنترلر داده شده مقایسه می‌نماید.

```
pidTuner(P,C)
```

مشاهده می‌شود که کنترلر طراحی شده پاسخ آهسته تری داده و دارای فراجهش بیشتری از کنترلر دستی می‌باشد. حال گزینه‌ی **Domain: Frequency** را از نوار ابزار انتخاب کرده تا پارامترهای تنظیم در حوزه‌ی فرکانس نشان داده شود.



حال برای **Bandwidth (پهنای باند)** مقدار ۳۲ رادیان/ثانیه و برای **Phase Margin (حد فاز)** مقدار **۹۰ درجه** را مشخص کرده تا کنترلری مشابه کنترلر مرجع تولید شود. پهنای باند حلقه بسته‌ی بیشتر سبب کاهش زمان نمو و فراجهش و افزایش حد فاز و پایداری سیستم می‌شود.

در نهایت باید به این نکته اشاره شود که با دستور `pidtune` به جای دستور `pidTuner` می‌توان به شکل زیر به همان کنترلر دست پیدا کرد:

```
opts = pidtuneOptions('CrossoverFrequency',32,'PhaseMargin',90);
[C, info] = pidtune(P, 'pid', opts)
```

```
C =

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

  with Kp = 320, Ki = 796, Kd = 32.2

Continuous-time PID controller in parallel form.
```

```
info =

  struct with fields:


               Stable: 1

    CrossoverFrequency: 32

           PhaseMargin: 90
```

بخش چهارم: مقدمه ای بر طراحی کنترلر با مکان هندسی ریشه‌ها

فهرست مطالب بخش

- قطب‌های حلقه بسته
- رسم مکان هندسی ریشه‌های تابع تبدیل
- انتخاب یک مقدار K از مکان هندسی ریشه‌ها
- پاسخ حلقه بسته
- استفاده از ابزار طراحی سیستم کنترل برای طراحی مکان هندسی ریشه‌ها

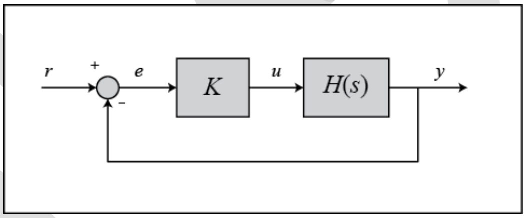
در این بخش مکان هندسی ریشه‌ها را معرفی می‌نماییم و روش ساخت آنها را با متلب نشان داده و روش طراحی کنترلرهای فیدبکی را که معیارهای مشخصی را به وسیله‌ی مکان هندسی ریشه‌ها ارضا کنند بیان می‌کنیم.

دستورهای کلیدی متلب در این بخش:

```
feedback, rlocus, step, controlSystemDesigner
```

قطب‌های حلقه بسته

مکان هندسی ریشه‌های[19] یک تابع تبدیل (حلقه باز) $H(s)$، رسم تمامی نقاط مختلف مکان ریشه‌های سیستم حلقه بسته به ازای تغییر یک پارامتر می‌باشد که این پارامتر معمولا بهره‌ی تناسبی K بوده و از صفر تا بینهایت تغییر می‌کند. شکل زیر شماتیک یک سیستم با فیدبک واحد را نشان می‌دهد اما روشی که در ادامه گفته می‌شود برای هر تابع تبدیل حلقه باز $H(s)$ مشابه می‌باشد حتی اگر برخی از اعضای سیستم حلقه باز در مسیر فیدبک قرار گرفته باشند.



تابع تبدیل حلقه بسته در این حالت برابر است با:

$$\frac{Y(s)}{R(s)} = \frac{KH(s)}{1 + KH(s)} \quad (۱)$$

در نتیجه قطب‌های سیستم حلقه بسته، مقادیری از s می‌باشند که رابطه‌ی $1 + KH(s) = 0$ را ارضا می‌کنند.

اگر فرض کنیم $H(s) = b(s)/a(s)$ آنگاه معادله‌ی گفته شده به شکل زیر بازنویسی می‌شود:

$$\Longrightarrow a(s) + Kb(s) = 0 \quad (۲)$$

$$\Longrightarrow \frac{a(s)}{K} + b(s) = 0 \quad (۳)$$

n را مرتبه‌ی چندجمله‌ای $a(s)$ و m را مرتبه‌ی چندجمله‌ای $b(s)$ در نظر می‌گیریم (مرتبه‌ی یک چندجمله‌ای برابر است با بزرگترین توان s در آن چندجمله‌ای).

[19] Root Locus

تمامی مقادیر مثبت را برای مقدار K در نظر می‌گیریم. اگر به صورت حدی $K \to 0$ آنگاه قطب‌های سیستم حلقه بسته عبارتند از جواب معادله $a(s) = 0$ (قطب‌های $H(s)$) و اگر $K \to \infty$ آنگاه قطب‌های سیستم حلقه بسته عبارتند از جواب معادله $b(s) = 0$ (صفرهای $H(s)$).

فارغ از انتخاب مقدار K، **سیستم حلقه بسته دارای n قطب می‌باشد** که در آن n تعداد قطب‌های تابع تبدیل حلقه باز $H(s)$ می‌باشد. پس **مکان هندسی ریشه‌ها دارای n شاخه می‌باشد،** هر شاخه از یک قطب $H(s)$ شروع شده و به سمت یک صفر $H(s)$ میل می‌کند. اگر تعداد قطب‌های $H(s)$ بیشتر از تعداد صفرهای آن باشد (که معمولا هم اینطور است) آنگاه $n < m$ بوده و می‌گوییم که $H(s)$ صفرهایی در بینهایت دارد. در این حالت حد $H(s)$ وقتی که $s \to \infty$ می‌رود صفر می‌باشد. تعداد صفرهای در بینهایت برابر است با $n - m$ یعنی تعداد قطب‌های حلقه باز منهای تعداد صفرهای حلقه باز، که این مقدار بیانگر تعداد شاخه‌هایی از مکان هندسی ست که به صورت مجانبی به بی‌نهایت می‌روند.

به علت اینکه مکان هندسی ریشه‌ها شامل تمامی نقاطی است که قطب‌های حلقه بسته می‌توانند اختیار کنند، مکان هندسی ریشه‌ها کمک می‌کند تا مقدار K را برای عملکرد مناسب، انتخاب نماییم. اگر هر یک از قطب‌های انتخاب شده در نیمه‌ی راست صفحه‌ی مختلط (یا سمت راست محور موهومی) بیافتند، سیستم حلقه بسته ناپایدار خواهد بود. پس حتی اگر یک سیستم دارای سه یا چهار قطب باشد با توجه به موقعیت قطب‌های غالب، همچنان رفتاری مشابه سیستم‌های مرتبه دوم یا اول را دارد.

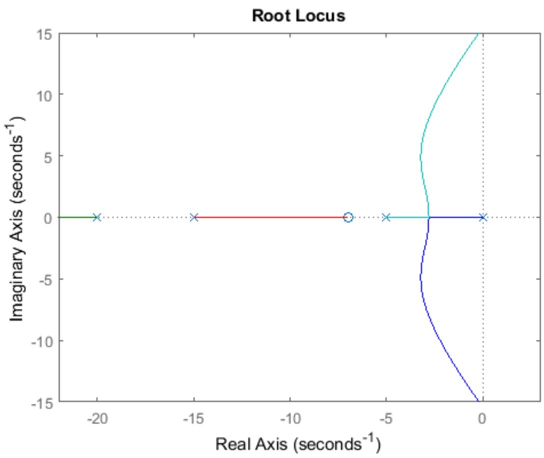
رسم مکان هندسی ریشه‌های تابع تبدیل

سیستم حلقه بازی که دارای تابع تبدیل زیر است را در نظر بگیرید:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{s+7}{s(s+5)(s+15)(s+20)} \qquad (۴)$$

چطور می‌توان با استفاده از مکان هندسی ریشه‌ها، یک کنترلر فیدبک برای این سیستم طراحی نمود؟ فرض کنیم الزامات طراحی برابر فراجهش ۵٪ و زمان نمو ۱ ثانیه باشند. امفایلی را با نام rl.m ساخته و در آن تابع تبدیل را تعریف نموده و دستور `rlocus` را اجرا کنید:

```
s = tf('s');
sys = (s + 7)/(s*(s + 5)*(s + 15)*(s + 20));
rlocus(sys)
axis([-22 3 -15 15])
```

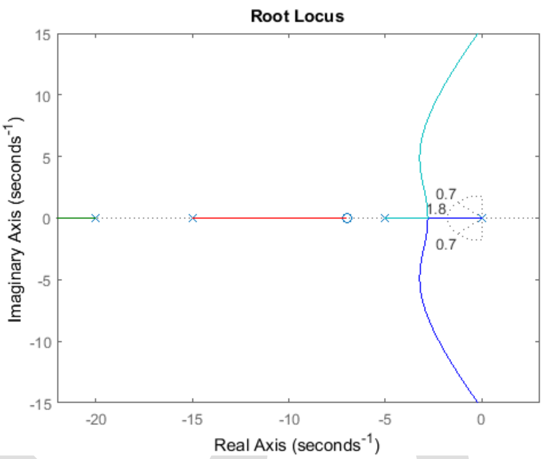


انتخاب مقدار K از مکان هندسی ریشه‌ها

نمودار بالا تمامی مکان‌های ممکن برای قطب‌های سیستم حلقه بسته با یک کنترلر تناسبی را نشان می‌دهد. در این حالت تمامی موقعیت‌های قطب‌ها معیارهای گفته شده را ارضا نمی‌کنند. برای مشخص کردن بخشی از مکان هندسی ریشه‌ها

که قابل قبول است می‌توانیم از دستور `sgrid(zeta,wn)` استفاده کنیم تا خطوط ضریب میرایی ثابت و فرکانس طبیعی ثابت را رسم کنیم. دو آرگومان این دستور عبارتند از ضریب میرایی (ζ) و فرکانس طبیعی (ω_n) (اگر معیار مورد نظر به صورت بازه باشد این دو آرگومان می‌توانند به صورت برداری تعریف شوند). در این مسئله ما به فراجهش کمتر از ۵٪ (که ضریب میرایی بزرگتر از ۰/۷ را نتیجه می‌دهد) و زمان نمو ۱ ثانیه (که فرکانس طبیعی بزرگتر از ۱/۸ را منجر می‌شود) نیاز داریم. دستورات زیر را در پنجره متلب وارد کنید:

```
zeta = 0.7;

wn = 1.8;

sgrid(zeta,wn)
```



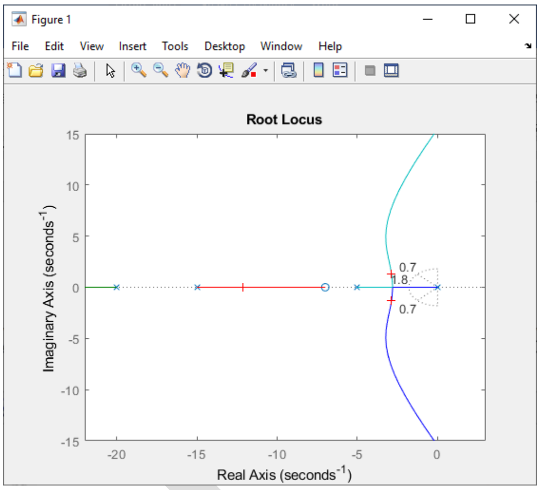
در نمودار بالا، دو خطچینی که با زاویه ۴۵ درجه رسم شده‌اند محل قطب‌های با $\zeta = 0.7$ را مشخص می‌کنند. قطب‌های قرار گرفته در میان این دو خط دارای $\zeta > 0.7$ بوده و خارج از این دو خط $\zeta < 0.7$ می‌باشد. نیم دایره‌ی رسم شده موقعیت قطب‌هایی که دارای $\omega_n = 1.8$ می‌باشند را مشخص می‌کند، پس داخل دایره $\omega_n < 1.8$ و خارج آن $\omega_n > 1.8$ می‌باشد.

برای به دست آوردن فراجهش کمتر از ۵٪، قطب‌ها باید بین دو خطچین زاویه‌دار باشند، و برای به دست آوردن زمان نمو کمتر از ۱ ثانیه باید قطب‌ها خارج از نیم‌دایره‌ی خط‌چین باشند. حال می‌دانیم که کدام بخش از مکان هندسی ریشه‌ها، شرایط گفته شده را برآورده می‌کند. تمامی قطب‌های در این ناحیه در نیم صفحه سمت چپ می‌باشد بنابراین سیستم حلقه بسته نیز پایدار می‌باشد.

در نمودار بالا مشاهده می‌شود که بخشی از مکان هندسی ریشه‌ها در منطقه‌ی مطلوب قرار گرفته است. بنابراین در این مسئله به کنترلر تناسبی نیاز داریم تا قطب‌ها را به منطقه‌ی مطلوب جابجا کند. با استفاده از دستور `rlocfind` در متلب می‌توان قطب‌های مطلوب در مکان هندسی را انتخاب نمود:

```
[k,poles] = rlocfind(sys)
```

با انتخاب بر روی هر نقطه از نمودار، قطب حلقه بسته‌ی مورد نظر انتخاب می‌گردد. با انتخاب قطب‌ها طبق نمودار زیر، شرایط مسئله ارضا می‌شود:



از آنجایی که مکان هندسی ریشه‌ها ممکن است بیش از یک شاخه داشته باشد، با انتخاب یک قطب، سایر قطب‌های حلقه بسته که همان مقدار K را دارند نیز مشخص می‌شود. دقت کنید که این قطب‌ها بر روی پاسخ نیز تاثیر می‌گذارند. در نمودار بالا مشاهده می‌کنیم که چهار قطب انتخاب شده است (با علامت "+" مشخص شده اند)، دو قطب نزدیکتر به محور موهومی در منطقه مطلوب قرار گرفته اند. چون این دو قطب، غالب می‌باشند پس می‌توان مطمئن بود که با یک کنترلر تناسبی با مقدار K مشخص شده، شرایط خواسته شده برآورده می‌شود.

پاسخ حلقه بسته

برای تغییر پاسخ پله لازم است تا تابع تبدیل حلقه بسته را داشته باشیم. این کار را می‌توان با استفاده از قوانین کاهش دیاگرام‌های بلوکی انجام داد. راه دیگر استفاده از متلب می‌باشد (اگر از دستور `rlocfind` استفاده شده است نیازی نیست تا مقدار K وارد شود):

```
K = 350;

sys_cl = feedback(K*sys,1)
```

```
sys_cl =


              350 s + 2450

  --------------------------------------

  s^4 + 40 s^3 + 475 s^2 + 1850 s + 2450
```

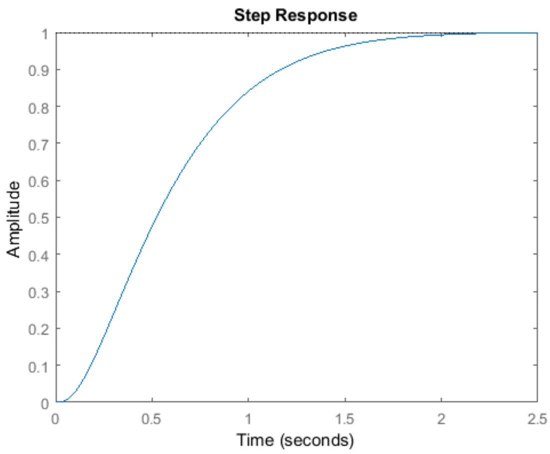
Continuous-time transfer function.

دو آرگومان وارد شده در تابع feedback، یکی تابع تبدیل در مسیر مستقیم و دیگری تابع تبدیل در مسیر فیدبک سیستم حلقه باز می‌باشد. در این مثال، سیستم ما دارای فیدبک واحد است.

اگر در سیستمی فیدبک غیر واحد داشتیم، به راهنمای تابع feedback در متلب که در آن به دست آوردن تابع تبدیل حلقه بسته با یک بهره در مسیر فیدبک را نشان داده است مراجعه کنید.

پاسخ پله برای سیستم حلقه بسته برای مقدار K انتخاب شده را بررسی می‌کنیم:

```
step(sys_cl)
```



همانطور که انتظار داشتیم، پاسخ به دست آمده دارای فراجهش کمتر از ۵٪ و زمان نمو کمتر از ۱ ثانیه می‌باشد.

استفاده از ابزار طراحی سیستم کنترل برای طراحی مکان هندسی ریشه‌ها

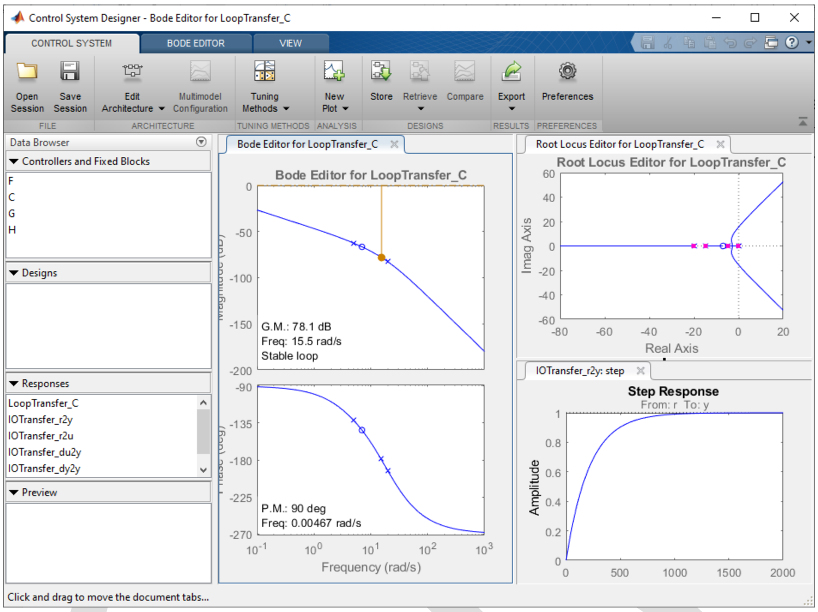
برای تکمیل قسمت قبل لازم است تا از ابزار طراحی سیستم کنترل در متلب استفاده نماییم. با استفاده از مدل قسمت قبل، سیستم $H(s)$ را به شکل زیر تعریف می‌نماییم:

```
s = tf('s');
plant = (s + 7)/(s*(s + 5)*(s + 15)*(s + 20));
```

تابع controlSystemDesigner برای طراحی و تحلیل به کار می‌رود. در این بخش از مکان هندسی ریشه‌ها برای بهبود پاسخ پله سیستم حلقه بسته به عنوان روش طراحی استفاده می‌نماییم. برای شروع، دستور زیر را در متلب وارد کنید:

```
controlSystemDesigner(plant)
```

با اجرای این دستور پنجره‌ی زیر باز می‌شود. همچنین با مراجعه به سربرگ APPS و انتخاب برنامه Control System Design and Analysis، می‌توان GUI این ابزار را نیز اجرا نمود. در این GUI می‌توانید نمودار مکان هندسی ریشه‌ها، دیاگرام بودی سیستم حلقه باز و پاسخ پله حلقه بسته برای سیستم با فیدبک واحد را با کنترلر پیش فرض $K = 1$ مشاهده نمایید.



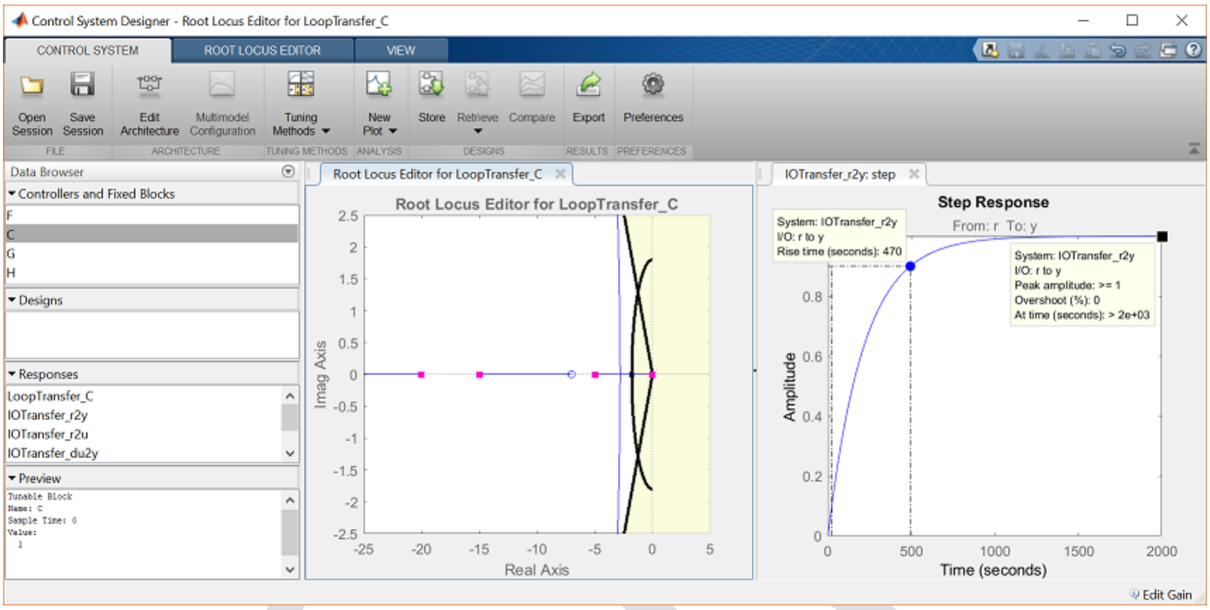
قدم بعد اضافه کردن الزامات طراحی به نمودار مکان هندسی ریشه‌ها می‌باشد. این امر با راست کلیک بر روی نمودار و انتخاب **Design Requirements, New** انجام می‌پذیرد. نیازهای طراحی را می‌توان برای زمان نشست، درصد فراجهش، ضریب میرایی، فرکانس طبیعی و قید منطقه تعریف نمود.

در اینجا نیازهای طراحی را برای ضریب میرایی و فرکانس طبیعی که در قسمت قبل با دستور `sgrid` نیز انجام شده بود تعریف می‌نماییم. برای یادآوری نیازهای طراحی مقدار $\zeta = 0.7$ و $\omega_n = 1.8$ می‌باشد. این مقادیر را در نیازهای طراحی وارد نمایید. مناطقی که در نمودار همچنان به رنگ سفید مشخص شده‌اند، مناطق قابل قبول برای قطب‌های حلقه بسته می‌باشند.

بزرگنمایی نمودار مکان هندسی ریشه‌ها را با کلیک راست بر روی نمودار و انتخاب **Properties** و در بخش **Limits** تغییر دهید. محدوده‌ی محور حقیقی را بین ۲۵- تا ۵ و محور موهومی را بین ۲/۵- تا ۲/۵ قرار دهید.

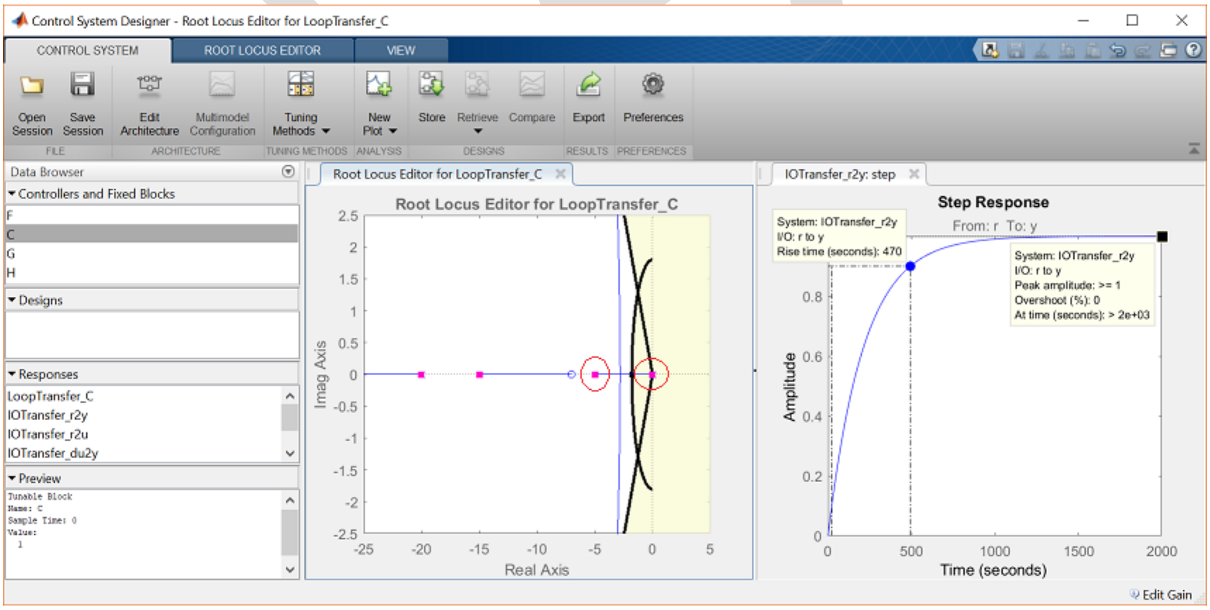
همچنین می‌توان برخی از مقادیر فعلی پارامترهای کلیدی پاسخ را مشاهده نمود. در نمودار پاسخ پله، بر روی آن کلیک راست کرده و به بخش **Characteristics** رفته و **Peak Response** را انتخاب کنید. همین کار را برای **Rise Time** انجام دهید. بعد از انتخاب این گزینه‌ها باید دو نقطه‌ی بزرگ بر روی صفحه نمایان شود. با کلیک بر روی هر یک از این نقطه‌ها، کادر اطلاعات را مشاهده می‌نمایید.

با بستن دیاگرام بودی، هر دو نمودار باید به شکل زیر درآمده باشند:

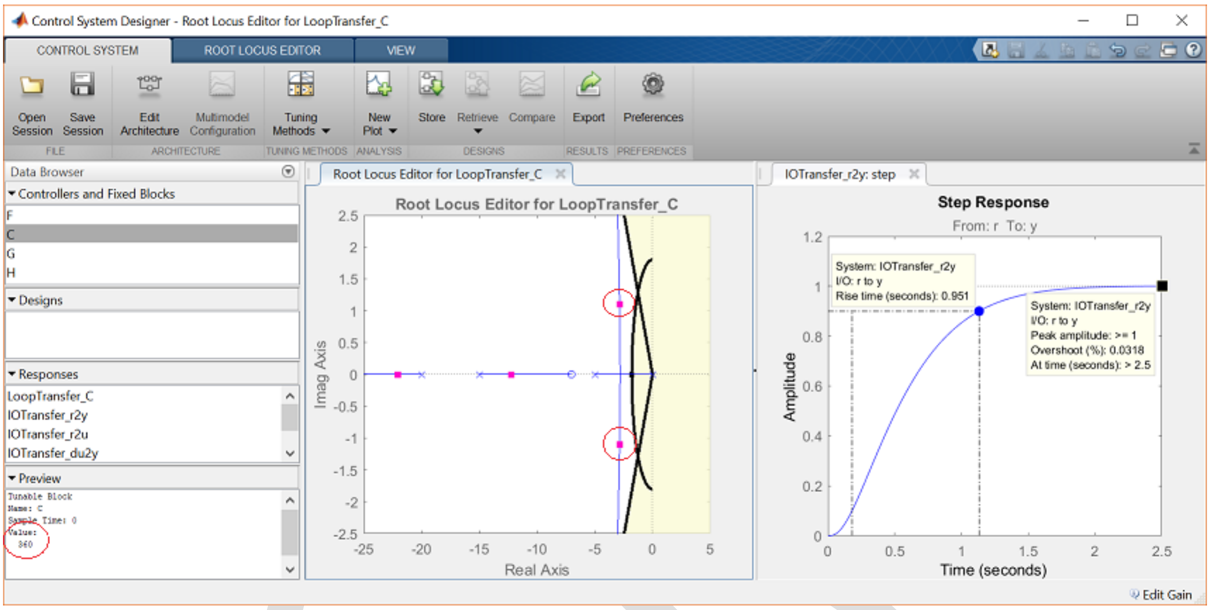


همانطور که در مشخصات پاسخ پله مشخص است، درصد فراجهش قابل قبول می‌باشد اما زمان نمو بسیار زیاد است. مربع‌های صورتی در مکان هندسی ریشه‌ها نشان‌دهنده‌ی مکان قطب‌های حلقه بسته با توجه به مقدار بهره‌ی کنترلی انتخاب شده‌ی فعلی K می‌باشند.

برای اصلاح زمان نمو، باید بهره‌ی K را تغییر داد. مشابه طرز استفاده از دستور `rlocfind`، بهره‌ی کنترلر را می‌توان مستقیما از نمودار مکان هندسی ریشه‌ها به دست آورد. بر روی نزدیکترین مربع صورتی به محور موهومی کلیک کرده و آنرا به مطقه‌ی قابل قبول که در نمودار مشخص شده است، حرکت دهید.



در پنجره‌ی **Preview** مشاهده می‌شود که بهره‌ی حلقه بسته به مقدار ۳۶۰ تغییر کرده است. با بررسی نمودار پاسخ پله حلقه بسته، هر دو شرط طراحی ارضا شده است:


Control System Designer - Root Locus Editor for LoopTransfer_C
CONTROL SYSTEM
ROOT LOCUS EDITOR
VIEW
Open Session
Save Session
Edit Architecture
Multimodel Configuration
Tuning Methods
New Plot
Store
Retrieve
Compare
Export
Preferences
FILE
ARCHITECTURE
TUNING METHODS
ANALYSIS
DESIGNS
RESULTS
PREFERENCES
Data Browser
Controllers and Fixed Blocks
F
C
G
H
Designs
Responses
LoopTransfer_C
IOTransfer_r2y
IOTransfer_r2u
IOTransfer_du2y
Preview
Root Locus Editor for LoopTransfer_C
Imag Axis
Real Axis
IOTransfer_r2y: step
Step Response
From: r To: y
System: IOTransfer_r2y
I/O: r to y
Rise time (seconds): 0.951
System: IOTransfer_r2y
I/O: r to y
Peak amplitude: >= 1
Overshoot (%): 0.0318
At time (seconds): > 2.5
Amplitude
Time (seconds)
Edit Gain

بخش پنجم: مقدمه‌ای بر طراحی کنترلر در حوزه‌ی فرکانس

فهرست مطالب بخش

روش طراحی کنترلر در حوزه‌ی فرکانس متفاوت‌تر از روش‌هاییست که تا به حال بررسی شده است. هر چند که این روش مزیت‌های خودش مخصوصا در شرایط زندگی واقعی مانند مدل‌سازی تابع تبدیل با استفاده از داده‌های فیزیکی را دارد. در این قسمت به چگونگی استفاده از پاسخ سیستم در حوزه‌ی فرکانس برای پیش‌بینی رفتار پاسخ زمانی سیستم حلقه بسته می‌پردازیم.

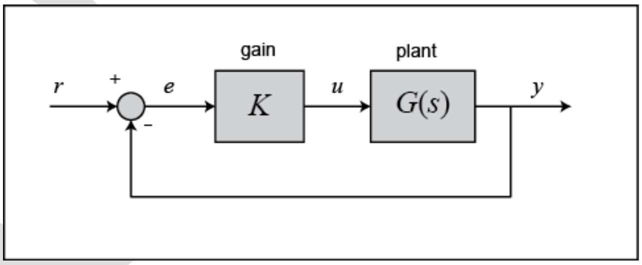
دستورهای کلیدی متلب در این بخش:

```
bode, nyquist, margin, lsim, step, feedback, controlSystemDesigner
```

حد فاز و بهره

با یادآوری از **بخش دوم: تحلیل سیستم** آموختیم که پاسخ فرکانسی یک سیستم نشان‌دهنده‌ی این است که چطور یک سیستم، ورودی سینوسی را تغییر مقیاس داده و جابجا می‌کند. این تغییر مقیاس و جابجایی سیگنال سینوسی خروجی می‌تواند بر حسب تابعی از فرکانس بیان شود که این تابع اطلاعات مفیدی را در مورد پاسخ زمانی سیستم در اختیار ما قرار می‌دهد. یکی از این اطلاعات، مشخص کردن مقاوم بودن سیستم به وسیله‌ی پاسخ فرکانسی سیستم می‌باشد. برای مثال اینکه یک سیستم چقدر به ناپایداری نزدیک است؟ در این مباحث ما از دو مقدار **حد فاز** و **حد بهره** برای نشان دادن حد سیستم قبل از رسیدن به ناپایداری استفاده می‌نماییم.

سیستم با فیدبک واحد زیر را در نظر بگیرید:

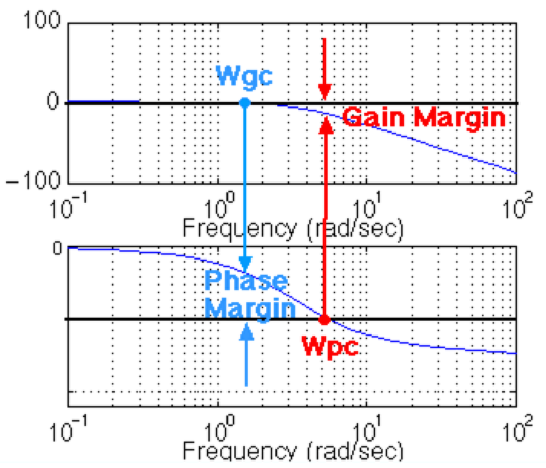


در این سیستم K بهره‌ی متغیر (یا ثابت) و $G(s)$ سیستم مورد بررسی است. حد بهره به صورت مقدار تغییری که در بهره‌ی حلقه باز لازم است تا سیستم حلقه بسته ناپایدار گردد تعریف می‌شود. سیستم‌هایی که حد بهره بالاتری دارند می‌توانند تغییرات بیشتری در پارامترهای سیستم را تحمل کرده و در چیدمان حلقه بسته دیرتر ناپایدار گردند.

حد فاز به صورت مقدار تغییر در فاز حلقه باز برای ناپایدار شدن سیستم حلقه بسته تعریف می‌شود. همچنین حد فاز مقدار مقاومت سیستم به تاخیر زمانی را نشان می‌دهد. اگر در حلقه بسته، تاخیر زمانی بیشتر از PM/ω_{gc} وجود داشته باشد (ω_{gc} فرکانس بر حسب رادیان بر ثانیه هنگامی که اندازه برابر صفر دسیبل است و PM حد فاز با واحد رادیان می‌باشد) سیستم حلقه بسته ناپایدار خواهد شد. تاخیر زمانی τ_d را می‌توان مانند یک بلوک اضافی در مسیر پیشرو فرض کرد که به سیستم تاخیر فاز داده اما تاثیری بر روی بهره نخواهد داشت. در نتیجه تاخیر زمانی را می‌توان مانند یک بلوک با اندازه‌ی ۱ و فاز $-\omega\tau_d$ در نظر گرفت.

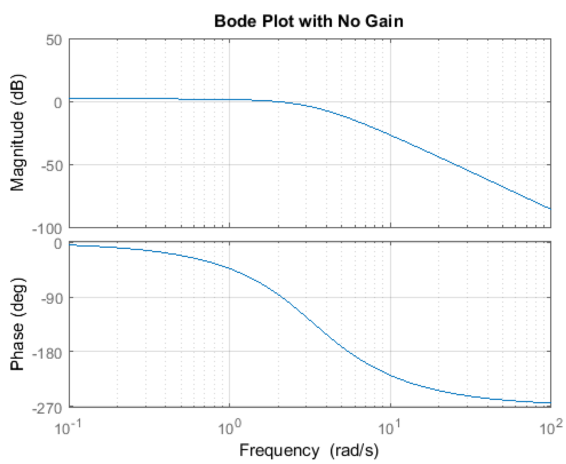
در حال حاضر روش به دست آوردن این مفاهیم اهمیتی نداشته و بر روی به دست آوردن حد بهره و فاز از روی دیاگرام بودی تمرکز می‌نماییم.

حد فاز سیستم حلقه بسته عبارتست از مقدار تاخیر فاز اضافی لازم برای رسیدن فاز سیستم حلقه باز به -180 درجه در فرکانسی که اندازه‌ی سیستم حلقه باز برابر صفر دسیبل (فرکانس گذر بهره ω_{gc}) می‌باشد. با تعریف مشابه، حد بهره مقدار بهره‌ی اضافی لازم (معمولا بر حسب دسیبل) برای رسیدن اندازه سیستم حلقه باز به صفر دسیبل در فرکانسی که فاز سیستم حلقه باز برابر ۱۸۰ درجه (فرکانس گذر فاز ω_{pc}) می‌باشد.



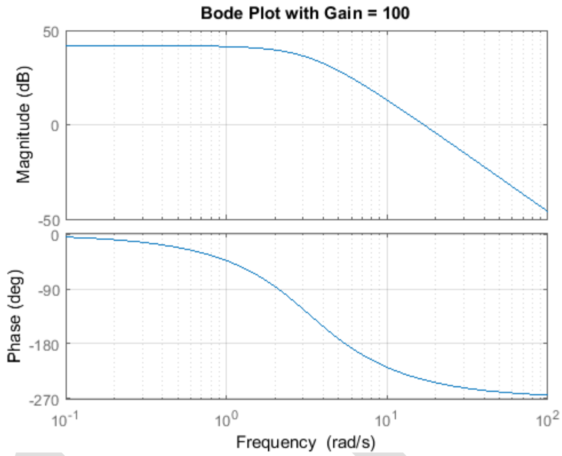
ویژگی خوبی که حد فاز دارد این است که برای به دست آوردن حد فاز جدید در هنگام تغییر بهره‌ی حلقه، نیاز به رسم دوباره‌ی دیاگرام بودی نمی‌باشد. اگر به یاد داشته باشید با اضافه کردن بهره، تنها نمودار اندازه، به سمت بالا (یا پایین) جابجا می‌شود، که معادل جابجا شدن محور y در نمودار اندازه می‌باشد. حد فاز را می‌توان به سادگی از یافتن فرکانس گذر بهره و خواندن حد فاز، به دست آورد. برای مثال، از دستورات زیر برای ترسیم دیاگرام بودی استفاده می‌نماییم:

```
s = tf('s');
G = 50/(s^3 + 9*s^2 + 30*s +40);

bode(G)
grid on
title('Bode Plot with No Gain')
```



همانطور که مشاهده می‌کنید، حد فاز حدود ۱۰۰ درجه می‌باشد. حال فرض کنید بهره‌ای با مقدار ۱۰۰ به حلقه، با دستور `bode(100*G)` اضافه شود. در این صورت نمودار زیر به دست می‌آید:

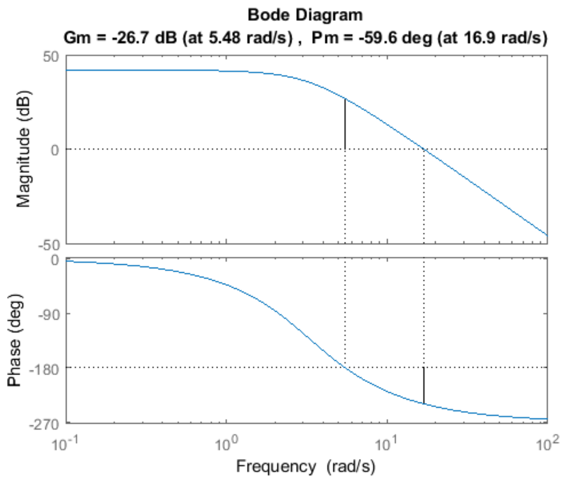
```
bode(100*G)

grid on
title('Bode Plot with Gain = 100')
```



همانطور که می‌توان دید، نمودار فاز دقیقا مشابه قبل بوده و نمودار اندازه به مقدار ۴۰ دسیبل (معادل بهره‌ی ۱۰۰) به سمت بالا جابجا شده است. حال مقدار حد فاز برابر 60- درجه است. نتایج مشابه را می‌توان با جابجا کردن محور y به مقدار ۴۰ دسیبل به سمت پایین به دست آورد. به نمودار قبل مراجعه نمایید و مقدار حد فاز در فرکانس گذر از 40- را بخوانید. این مقدار باید برابر 60- و برابر با نمودار بودی دوم باشد.

با استفاده از دستور `margin(G)` می‌توان حدود بهره و فاز را به کمک نرم افزار متلب به دست آورد. این دستور مقدار حدود فاز و بهره، فرکانس گذر بهره و فاز و نمایش این مقادیر بر روی دیاگرام بودی را ایجاد می‌کند. برای مثال داریم:

```
margin(100*G)
```



فرکانس پهنای باند

فرکانس پهنای باند، فرکانسی است که در آن اندازه‌ی سیستم حلقه بسته به مقدار ۳ دسیبل پایین تر از مقدار اندازه در DC (اندازه وقتی فرکانس به صفر میل می‌کند) می‌رسد. اگرچه وقتی با روش پاسخ فرکانسی اقدام به طراحی می‌نماییم، قصد ما به دست آوردن رفتار سیستم حلقه بسته با استفاده از پاسخ حلقه باز می‌باشد. بنابراین ما از تقریب یک سیستم مرتبه دو استفاده کرده و فرکانس پهنای باند را فرکانسی که در آن اندازه‌ی پاسخ حلقه باز بین مقادیر 6- تا 7.5- دسیبل و پاسخ

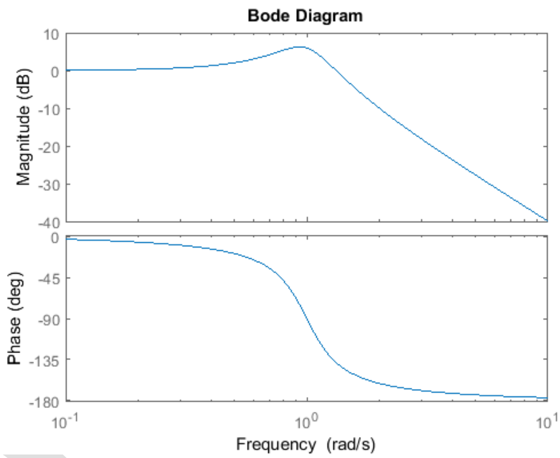
فاز حلقه باز بین 135- تا 225- درجه می‌رسد، تعریف می‌نماییم. برای توضیحات بیشتر در خصوص به دست آوردن این تقریب، به کتب مرجع مراجعه کنید.

برای نمایش اهمیت فرکانس پهنای باند، تغییرات خروجی را به ازای فرکانس‌های مختلف ورودی نشان می‌دهیم. با اینکار متوجه می‌شویم که ورودی‌های سینوسی با فرکانس کمتر از ω_{bw} (فرکانس پهنای باند) به طور معقولی توسط سیستم دنبال می‌شوند. ورودی‌های سینوسی با فرکانس بیشتر از ω_{bw}، مقدار اندازه‌ی آنها با ضریب 0.707 یا بیشتر، تضعیف می‌شوند (همچنین مقدار فاز آنها جابجا می‌گردد).

اگر یک سیستم به شکل زیر با نمایش تابع تبدیل داشته باشیم:

$$G(s) = \frac{1}{s^2 + 0.5s + 1} \quad (۱)$$

```
G = 1/(s^2 + 0.5*s + 1);

bode(G)
```

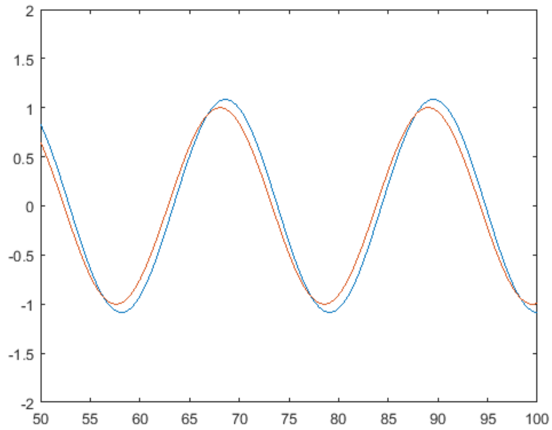


به علت اینکه دیاگرام رسم شده برای سیستم حلقه بسته می‌باشد، فرکانس پهنای باند برابر با فرکانس متناسب با بهره‌ی 3- دسیبل می‌باشد. با دقت به نمودار، این مقدار تقریبا برابر rad/s ۱/۴ می‌باشد. همچنین می‌توانیم از نمودار دریابیم که برای یک فرکانس ورودی 0.3 رادیان، خروجی سینوسی باید اندازه‌ی حدود ۱ باشد و فاز آن به مقدار چند درجه جابجا شود. برای فرکانس ورودی 3 rad/s، اندازه‌ی خروجی باید تقریبا برابر 20- dB (یا 1/10 برابر ورودی) و فاز آن تقریبا نزدیک 180- باشد. می‌توان از دستور `lsim` برای شبیه‌سازی پاسخ سیستم به ورودی سینوسی استفاده نمود.

ابتدا ورودی سینوسی با فرکانس کمتر از ω_{bw} را در نظر می‌گیریم. همچنین باید در نظر داشته باشیم که پاسخ حالت ماندگار را می‌خواهیم. بنابراین با تغییر در محورهای مختصات، پاسخ حالت ماندگار را با دقت مشاهده می‌نماییم (از پاسخ گذرا چشم‌پوشی می‌کنیم).

```
G = 1/(s^2 + 0.5*s + 1);

w = 0.3;

t = 0:0.1:100;

u = sin(w*t);

[y,t] = lsim(G,u,t);
```

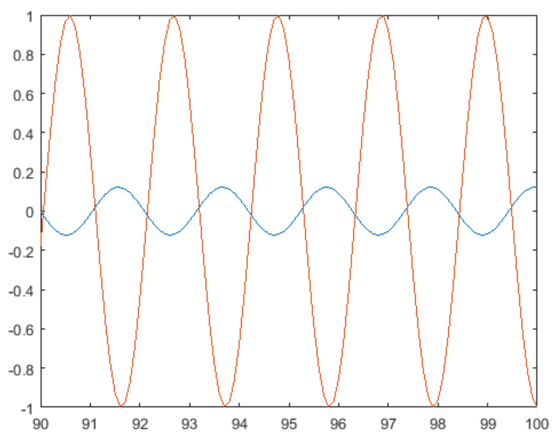
```
plot(t,y,t,u)

axis([50 100 -2 2])
```



در این حالت خروجی (خط آبی) سیگنال ورودی (خط قرمز) را به خوبی دنبال کرده و همانطور که انتظار می‌رفت چند درجه از ورودی عقب تر است.

اگرچه فرکانس ورودی را بیشتر از مقدار فرکانس پهنای باند قرار دهیم، پاسخ نامطلوبی را دریافت می‌کنیم:

```
G = 1/(s^2 + 0.5*s + 1);

w = 3;

t = 0:0.1:100;

u = sin(w*t);

[y,t] = lsim(G,u,t);

plot(t,y,t,u)

axis([90 100 -1 1])
```



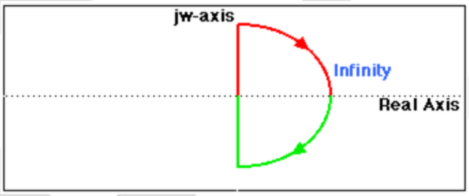
دوباره می‌توان مشاهده نمود که اندازه‌ی خروجی 1/10 ورودی بوده و تقریبا می‌توان گفت خارج از فاز (۱۸۰ درجه عقب‌تر از ورودی) است. می‌توانید با فرکانس‌های مختلف، پاسخ را مشاهده کرده و آنرا با دیاگرام بودی مقایسه کنید.

دیاگرام نایکوئست

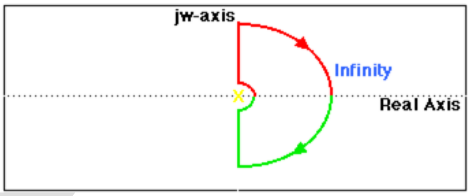
دیاگرام نایکوئست به ما قابلیت پیش بینی پایداری و عملکرد سیستم حلقه بسته، با مشاهده‌ی رفتار سیستم حلقه باز می‌دهد. شرط نایکوئست را می‌توان برای طراحی بدون توجه به پایداری حلقه باز به کار برد (به یاد داریم که روش‌های طراحی به کمک دیاگرام بودی، با فرض پایدار بودن سیستم حلقه باز انجام می‌گرفت). بنابراین ما از این شرط برای تعیین پایداری حلقه بسته در شرایطی که نمودار بودی اطلاعات گیج کننده‌ای را نمایش می‌دهد، استفاده می‌نماییم.

نکته: دستور `nyquist` در متلب، اطلاعات کافی برای سیستم‌هایی که قطب‌های حلقه باز آنها بر روی محور موهومی می‌باشد را نمایش نمی‌دهد. بنابراین پیشنهاد می‌کنیم تا فایل nyquist1.m را به عنوان یک امفایل جدید از درون سی‌دی بر روی کامپیوتر خود ذخیره کنید. این امفایل دیاگرام نایکوئست دقیق‌تری را تولید می‌کند زیرا به درستی قطب‌ها و صفرهای روی محور موهومی را در نظر می‌گیرد.

نمودار نایکوئست به طور اساسی یک نموداری از $G(j\omega)$ می‌باشد که $G(s)$ تابع تبدیل حلقه باز و ω بردار فرکانس‌های شامل تمامی نیم‌صفحه‌ی راست می‌باشد. در رسم نمودار نایکوئست، هم فرکانس‌های مثبت (از صفر تا بینهایت) و هم فرکانس‌های منفی (از منفی بینهایت تا صفر) در نظر گرفته می‌شوند. در اینجا فرکانس‌های مثبت را با قرمز و فرکانس‌های منفی را با سبز نمایش می‌دهیم. بردار فرکانسی که برای رسم نمودار نایکوئست استفاده می‌شود معمولا به شکل زیر است (تصور کنید که نمودار تا بینهایت ادامه دارد):



اگر بر روی محور موهومی صفر یا قطب حلقه باز داشته باشیم، $G(s)$ در آن نقاط تعریف نمی‌شود و باید در هنگامی که کانتور را رسم می‌کنیم، از دور آنها عبور کنیم:



در نمودار بالا به حلقه‌ی کشیده شده به دور قطب موجود بر روی محور موهومی دقت نمایید. همانطور که گفته شد، دستور `nyquist` متلب این قطب و صفرهای روی محور موهومی را در نظر نگرفته و در نتیجه نمودار نادرستی را ایجاد می‌نماید. برای رفع این مشکل از فایل nyquist1.m استفاده می‌نماییم. اگر سیستم ما دارای یک قطب بر روی محور موهومی باشد باید از `nyquist1` استفاده کنیم. اگر سیستم دارای قطب یا صفری بر روی محور موهومی نباشد یا بر روی این محور حذف صفر و قطب اتفاق افتاده باشد می‌توانیم هم از دستور `nyquist` و هم فایل nyquist1.m استفاده نماییم.

شرط کوشی[۲۰]

شرط کوشی (از تحلیل مختلط) بیان می‌کند هنگامی که یک کانتور بسته در صفحه مختلف داریم و آنرا توسط تابع مختلط $G(s)$ نگاشت می‌کنیم، تعداد دفعاتی که نمودار $G(s)$ به دور مبدا مختصات می‌چرخد برابر با تعداد صفرهای

[۲۰] Cauchy Criterion

$G(s)$ محاط شده توسط کانتور فرکانس منهای تعداد قطب‌های $G(s)$ محاط شده توسط کانتور فرکانس می‌باشد. چرخش حول مبدا در صورتی که در جهت کانتور بسته باشد مثبت و اگر در خلاف جهت باشد منفی در نظر گرفته می‌شود.

در بررسی کنترل فیدبک، برای ما $G(s)$ اهمیتی نداشته و تابع تبدیل حلقه بسته حائز اهمیت می‌باشد:

$$\frac{G(s)}{1+G(s)} \qquad (۲)$$

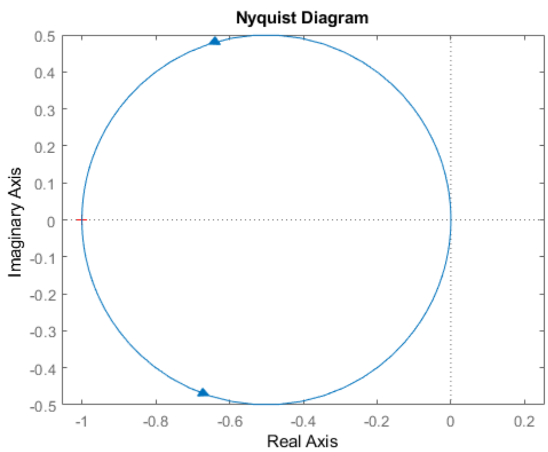
اگر $1+G(s)$ حول مبدا بچرخد، آنگاه $G(s)$ نقطه‌ی -1 را در بر می‌گیرد. از آنجایی که پایداری حلقه بسته مورد نظر ماست، هدف ما تعیین تعداد قطب‌های حلقه بسته (ریشه‌های $(1+G(s)$ در سمت راست محور موهومی می‌باشد. توضیحات بیشتر که تعداد قطب‌ها چگونه به دست می‌آید در ادامه گفته خواهد شد.

در نتیجه رفتار دیاگرام نایکوئست حول نقطه -1 بر روی محور حقیقی بسیار مهم می‌باشد. اگرچه در نمودار های رسم شده با دستور `nyquist` ممکن است جزییات نمودار در نقطه -1 کاملا واضح نباشد. برای رفع این مشکل می‌توانید تابع lnyquist.m را به فایل‌های خود اضافه کنید. دستور lnyquist.m نمودار نایکوئست را با مقیاس لگاریتمی و حفظ خصوصیات نقطه‌ی -1 رسم می‌نماید.

برای نمایش یک نمودار نایکوئست ساده در متلب، تابع تبدیل زیر را تعریف کرده و دستور رسم نمودار نایکوئست را اجرا می‌کنیم:

$$G(s)=\frac{0.5}{s-0.5} \qquad (۳)$$

```
s = tf('s');
G = 0.5/(s - 0.5);
nyquist(G)
axis equal
```



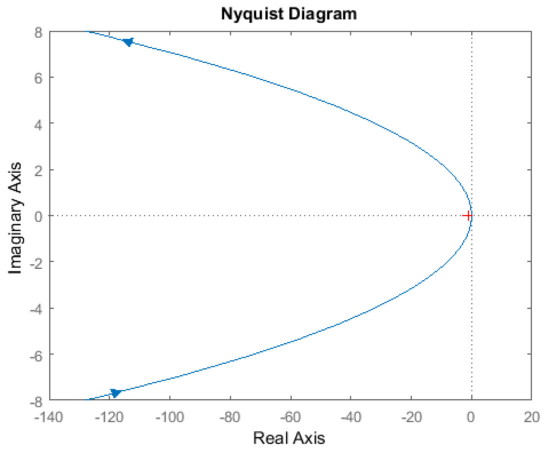
حال به نمودار نایکوئست تابع تبدیل زیر توجه کنید:

$$G(s)=\frac{s+2}{s^2} \qquad (۴)$$

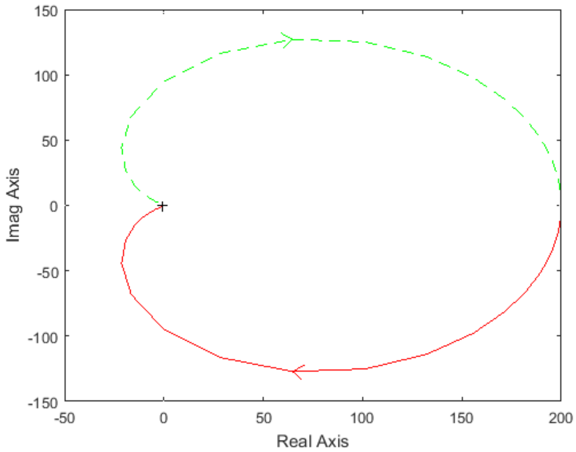
لازم به ذکر است که این نمودار دارای قطب در مبدا می‌باشد. تفاوت دستورهای `nyquist`، `nyquist1` و `lnyquist` را در این تابع تبدیل مشاهده می‌کنیم:

```
G = (s + 2)/(s^2);
```

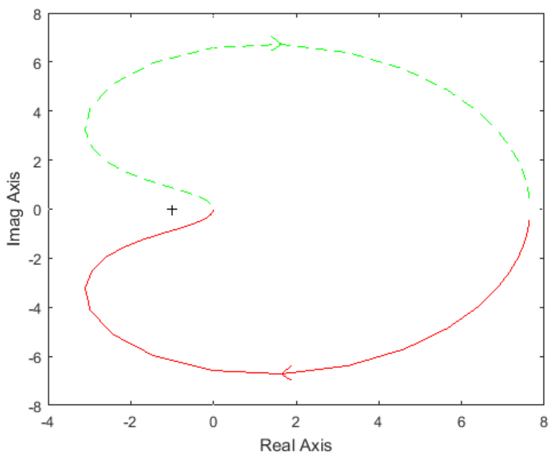
```
nyquist(G)
```




```
nyquist1(G)
```




```
lnyquist(G)
```



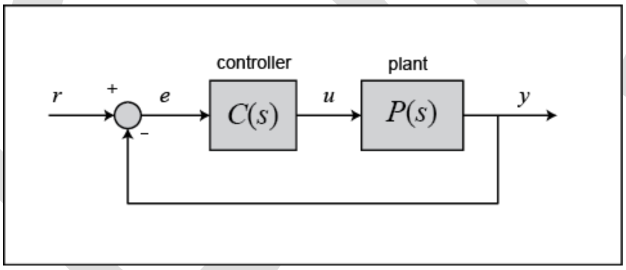
دیاگرام رسم شده توسط nyquist صحیح نمی‌باشد زیرا رفتار دیاگرام نایکوئست در شعاع به سمت بینهایت (s میل می‌کند به صفر) را بیان نمی‌کند. نمودار nyquist1 دارای شکل صحیحی بوده که به ما اجازه‌ی بررسی چرخش حول نقطه -1 و اعمال شرط نایکوئست را داده اما به سختی می‌توان جزییات نمودار را در نزدیکی نقطه -1 مشاهده کرد. تابع lnyquist از مقیاس اصلاح شده استفاده کرده و رفتار صحیح نمودار در نزدیکی نقطه -1 را با درستی شکل کلی نمودار، حفظ می‌نماید.

عملکرد حلقه بسته با استفاده از دیاگرام بودی

برای به دست آوردن عملکرد حلقه بسته با استفاده از پاسخ فرکانسی حلقه باز، باید فرضیات زیر را در نظر گرفت:

- برای استفاده از دیاگرام بودی در طراحی، سیستم حلقه باز باید پایدار باشد.
- برای سیستم‌های مرتبه دوم با فرم کانونی، در هنگامی که حد فاز بین ۰ تا ۶۰ درجه است، ضریب میرایی حلقه بسته تقریبا برابر با حد فاز تقسیم بر ۱۰۰ می‌باشد. از این فرض در مواقعی که حد فاز بیشتر از ۶۰ درجه می‌باشد می‌توان به شرط احتیاط استفاده نمود.
- برای سیستم‌های مرتبه دوم فرم کانونی، رابطه‌ی بین ضریب میرایی، فرکانس پهنای باند و زمان نشست در معادله‌ی موجود در بخش **پیوست: پهنای باند** آورده شده است.
- تقریب حدودی که می‌توان از آن استفاده کرد این است که پهنای باند تقریبا برابر فرکانس طبیعی می‌باشد.

حال با استفاده از این مفاهیم به طراحی یک کنترلر برای سیستم زیر می‌پردازیم:



که $C(s)$ کنترلر و $P(s)$ برابر:

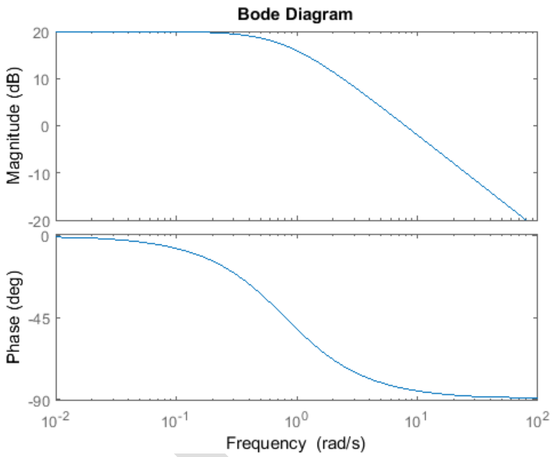
$$P(s) = \frac{10}{1.25s + 1} \qquad (۵)$$

می‌باشد. الزامات طراحی عبارتند از:

- خطای حالت ماندگار صفر
- حداکثر فراجهش کمتر از ۴۰٪
- زمان نشست کمتر از ۲ ثانیه

دو راه برای حل این مسئله وجود دارد، روش اول ترسیمی و روش دوم عددی می‌باشد. در هنگام استفاده از نرم افزار متلب، روش ترسیمی بهترین راه می‌باشد بنابراین از روش ترسیمی استفاده می‌نماییم. ابتدا به دیاگرام بودی توجه می‌کنیم. امفایلی را با کد زیر ایجاد کنید:

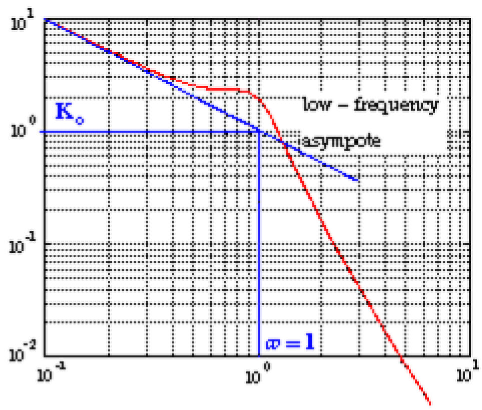
```
P = 10/(1.25*s + 1);

bode(P)
```



چندین ویژگی سیستم را می‌توان به طور مستقیم از دیاگرام بودی خواند. اول از همه، می‌توان دریافت که فرکانس پهنای باند حدود ۱۰ رادیان بر ثانیه می‌باشد. از آنجایی که فرکانس پهنای باند تقریبا برابر با فرکانس طبیعی می‌باشد (برای سیستم مرتبه اول از این نوع)، زمان نمو برابر $\frac{1.8}{BW} = \frac{1.8}{10} = 0.18$ ثانیه می‌باشد. این مقدار با تقریب زیادی به دست آمده است پس می‌توان گفت زمان نمو تقریبا برابر ۰/۲ ثانیه است.

حد فاز برای این سیستم تقریبا برابر ۹۵ درجه می‌باشد. رابطه‌ی $(PM/100 =$ ضریب میرایی$)$ تنها برای حد فاز کمتر از ۶۰ درجه قابل استفاده است. به علت اینکه سیستم مرتبه اول می‌باشد پس فراجهشی وجود نخواهد داشت.

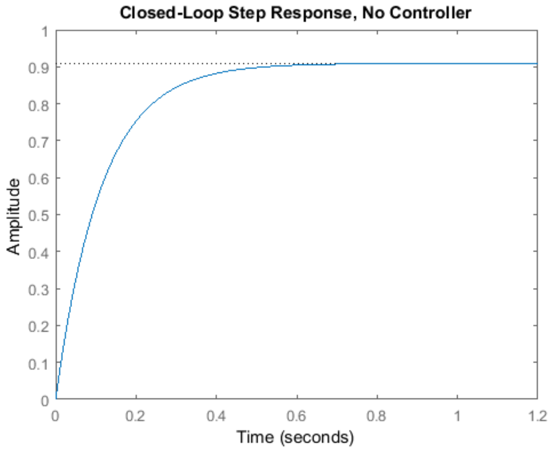
نکته‌ی مهم آخر، خطای حالت ماندگار می‌باشد. خطای حالت ماندگار را نیز می‌توان مستقیما از دیاگرام بودی به دست آورد. ثابت خطا (K_p، K_v و K_a) از تقاطع مجانب فرکانس پایین با خط $\omega = 1\ rad/sec$ به دست می‌آید. تنها کافیست خط فرکانس پایین را ادامه داده تا خط $\omega = 1$ را قطع کند. مقدار اندازه در این نقطه برابر با ثابت خطا می‌باشد. به علت اینکه دیاگرام بودی این سیستم در فرکانس پایین افقی می‌باشد (شیب مساوی صفر)، می‌دانیم این سیستم در فیدبک واحد از نوع صفر می‌باشد. بنابراین پیدا کردن این تقاطع ساده می‌باشد. بهره برابر ۲۰ دسیبل (اندازه‌ی ۱۰) می‌باشد. این بدین معناست که ثابت تابع خطا برابر ۱۰ می‌باشد. خطای حالت ماندگار عبارتست از $\frac{1}{1+K_p} = \frac{1}{1+10} = 0.091$. اگر سیستم ما به جای نوع صفر، از نوع یک بود، ثابت خطای حالت ماندگار به طرز مشابه، به شکل زیر به دست می‌آمد:



حال پیش‌بینی‌های انجام شده را بر روی پاسخ پله بررسی می‌نماییم. برای اینکار می‌توانیم دو خط زیر را به کد متلب اضافه نماییم:

```
sys_cl = feedback(P,1);

step(sys_cl)
```

```
title('Closed-Loop Step Response, No Controller')
```



همانطور که مشاهده می‌کنید، پیش‌بینی‌های انجام شده بسیار خوب بودند. زمان نمو سیستم تقریبا برابر ۰/۲ ثانیه بوده، فراجهش وجود نداشته و خطای حالت ماندگار تقریبا ۹٪ می‌باشد. حال باید کنترلری را انتخاب نماییم تا برای ما شرایط مطلوب را ایجاد کند. انتخاب ما کنترلر PI می‌باشد زیرا با استفاده از این کنترلر خطای حالت ماندگار به پاسخ پله به صفر می‌رسد. همچنین کنترلر PI دارای صفری است که می‌توان آنرا جای‌دهی کرد که به ما کمک می‌کند تا نیازهای طراحی را برآورده کنیم. برای یادآوری فرم کلی کنترلر PI به شکل زیر است:

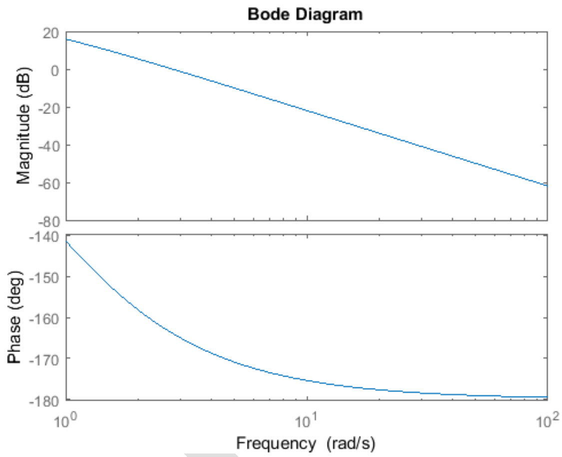
$$C(s) = \frac{K(s+a)}{s} = K + \frac{Ka}{s} \qquad (۶)$$

اولین چیزی که باید به دست آورد، ضریب میرایی برای فراجهش ۴۰٪ می‌باشد. با قرار دادن این مقدار در رابطه‌ی فراجهش و ضریب میرایی، مقدار به دست آمده برای ضریب میرایی مربوط به این مقدار فراجهش برابر با ۰/۲۸ می‌باشد. بنابراین حد فاز باید حداقل ۳۰ درجه باشد. فرکانس پهنای باند باید برابر یا بیشتر از ۱۲ باشد تا زمان نشست کمتر از ۱/۷۵ ثانیه به دست آید و نیازهای طراحی ارضا شود.

حال که مقدار مطلوب حد فاز و فرکانس پهنای باند را می‌دانیم می‌توانیم طراحی را شروع کنیم. به یاد داشته باشید که دیاگرام بودی حلقه باز را مشاهده می‌نماییم. در نتیجه فرکانس پهنای باند برابر با فرکانسی است که در آن بهره تقریبا برابر ۷- دسیبل است.

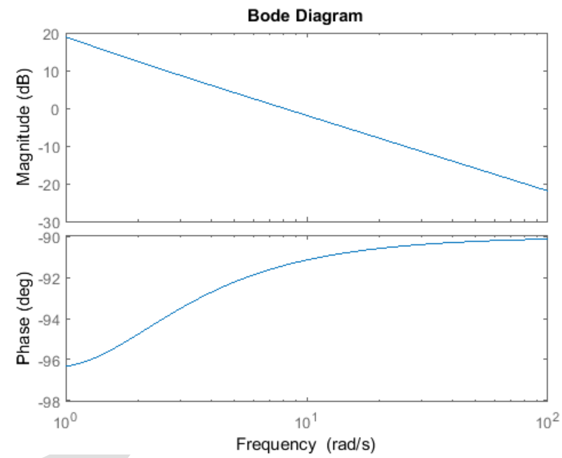
حال تاثیر قسمت انتگرالی کنترلر PI را بر روی پاسخ بررسی می‌کنیم. ام‌فایل خود را به شکل زیر تغییر دهید (با اینکار جمله‌ی انتگرالی اضافه شده اما جمله‌ی تناسبی اضافه نمی‌شود):

```
P = 10/(1.25*s + 1);

C = 1/s;

bode(C*P, logspace(0,2))
```



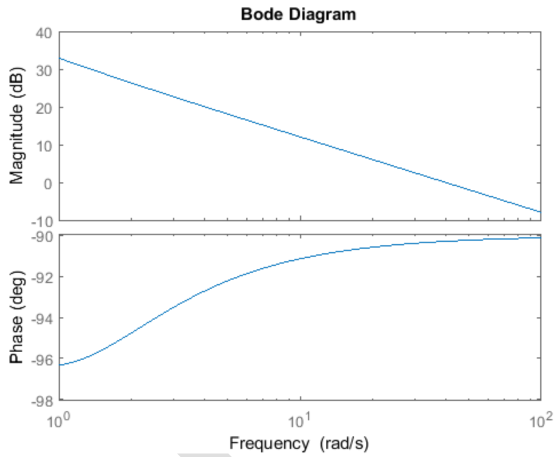
حد فاز و فرکانس پهنای باند بسیار کوچک می‌باشند. با اضافه کردن یک صفر، بهره و فاز را افزایش می‌دهیم. با اضافه کردن صفر در نقطه ۱- و اعمال تغییرات زیر در امفایل، تاثیر آنرا مشاهده می‌نماییم:

```
P = 10/(1.25*s + 1);

C = (s + 1)/s;

bode(C*P, logspace(0,2))
```



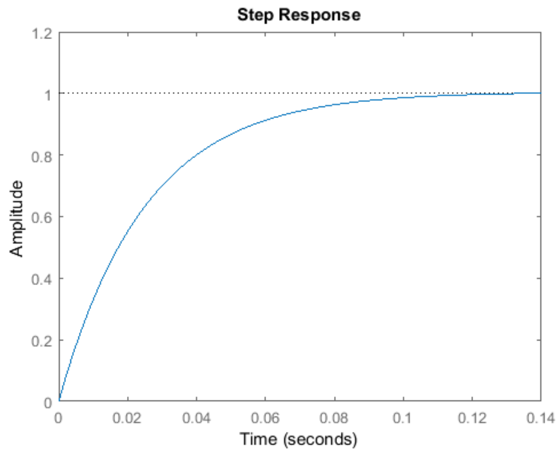
با مشاهده به دیاگرام بودی، متوجه می‌شویم اضافه کردن صفر در نقطه ۱-، به ما پاسخ مورد نظر را می‌دهد. حد فاز بزرگتر از ۶۰ درجه بوده (حتی فراجهش کمتر از حد انتظار می‌باشد) و فرکانس پهنای باند تقریبا برابر ۱۱ رادیان بر ثانیه است که پاسخ مناسب را می‌دهد. هرچند پاسخ به دست آمده رضایت بخش است اما پاسخ خوبی نمی‌باشد. بنابراین فرکانس پهنای باند بالاتری را بدون تغییر زیاد حد فاز در نظر می‌گیریم. مقدار بهره را به ۵ افزایش داده و تاثیر آنرا مشاهده می‌کنیم. با اینکار نمودار بهره جابجا شده اما نمودار فاز ثابت می‌ماند.

```
P = 10/(1.25*s + 1);

C = 5*((s + 1)/s);

bode(C*P, logspace(0,2))
```



نتیجه به دست آمده بسیار خوب است. حال به پاسخ پله به دست آمده پرداخته و نتایج خود را بررسی می‌کنیم. دو خط زیر را به ام‌فایل خود اضافه نمایید:

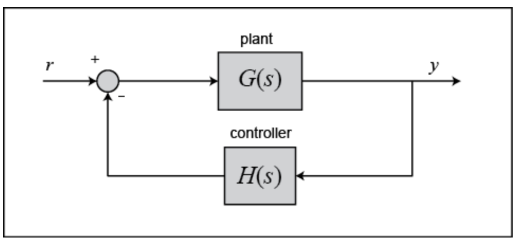
```
sys_cl = feedback(C*P,1);

step(sys_cl)
```



همانطور که مشاهده می‌کنید پاسخ به دست آمده بهتر از چیزی است که انتظار داشتیم. هرچند که همیشه پاسخ مناسب به این سادگی به دست نیامده و باید با تغییر دادن بهره و محل قطب‌ها و صفرها، پاسخ مناسب را به دست آورد.

پایداری حلقه بسته از دیاگرام نایکوئست

سیستم با فیدبک منفی زیر را در نظر بگیرید:



از شرط کوشی می‌داینم که تعداد دفعاتی (N) که منحنی $G(s)H(s)$ حول نقطه $-1+0j$ می‌چرخد برابر است با تعداد (Z) صفرهای $1+G(s)H(s)$ که توسط کانتور فرکانس محاط شده‌اند منهای تعداد (P) قطب‌های $1+G(s)H(s)$ که توسط کانتور فرکانس محاط شده اند (N=Z-P). با دقت به تابع‌های تبدیل حلقه باز و بسته و همچنین صورت مخرج آنها، می‌دانیم که:

- صفرهای $1+G(s)H(s)$ همان قطب‌های تابع تبدیل حلقه بسته‌اند.
- قطب‌های $1+G(s)H(s)$ همان قطب‌های تابع تبدیل حلقه باز هستند.

با توجه به تعریف‌های بالا داریم:

- P برابر تعداد قطب‌های حلقه باز (ناپایدار) $G(s)H(s)$ است.
- N برابر تعداد چرخش نمودار نایکوئست حول نقطه ۱- است.
- چرخش ساعتگرد حول نقطه ۱- به عنوان چرخش مثبت حساب می‌گردد.
- چرخش پادساعتگرد حول نقطه ۱- به عنوان چرخش منفی حساب می‌گردد.
- Z برابر تعداد قطب‌های سمت راست (مثبت و حقیقی) سیستم حلقه بسته می‌باشد.

بنابراین شرط پایداری نایکوئست که این سه مقدار را به یکدیگر ارتباط می‌دهد برابر است با:

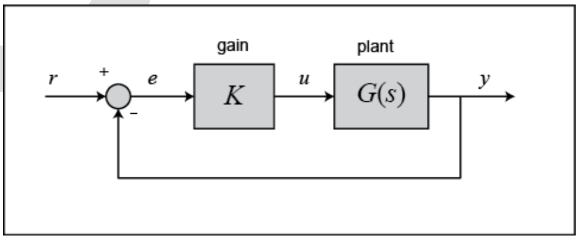
$$Z = P + N \qquad (۷)$$

نکته: این تنها یک بیان از شرط نایکوئست می‌باشد. بیان دیگر، N را به ازای چرخش پادساعتگرد حول نقطه ۱- مثبت در نظر می‌گیرد. مقادیر P و Z مانند قبل در نظر گرفته می‌شوند. در این صورت معادله به شکل $\boldsymbol{Z = P - N}$ به دست می‌آید. در این کتاب از علامت مثبت برای چرخش ساعتگرد استفاده می‌کنیم.

بسیار مهم است که شمارش تعداد دفعات چرخش منحنی نایکوئست به دور ۱- را درست انجام دهیم. برای یادگیری درست شمارش، به جزییات آن می‌پردازیم. یک روش به این صورت است که فرض می‌کنیم بر روی نقطه ۱- ایستاده و منحنی را از ابتدا تا انتها دنبال می‌کنیم. حال از خود می‌پرسیم چند بار به طور کامل ۳۶۰ درجه چرخیده‌ایم؟ اگر جهت چرخش ساعتگرد بود مقدار N مثبت و اگر چرخش پادساعتگرد بود مقدار N منفی است.

با دانستن تعداد قطب‌های سمت راست حلقه بسته (P) و تعداد دفعات چرخش دیاگرام نایکوئست حول نقطه ۱- (N) می‌توانیم پایداری سیستم حلقه بسته را تعیین کنیم. اگر Z=P+N مثبت و غیر صفر باشد، سیستم حلقه بسته ناپایدار است.

همچنین می‌توان از نمودار نایکوئست برای به دست آوردن محدوده بهره برای پایداری سیستم حلقه بسته با فیدبک واحد استفاده نمود. سیستم مورد بررسی به شکل زیر است:



که داریم:

$$G(s) = \frac{s^2 + 10s + 24}{s^2 - 8s + 15} \qquad (۸)$$

این سیستم دارای بهره K می‌باشد که می‌توان برای تغییر پاسخ سیستم حلقه بسته آنرا تغییر داد. اگرچه خواهیم دید که تنها می‌توان این بهره را در یک محدوده مشخصی تغییر داد زیرا باید پایداری حلقه بسته را نیز تضمین نمود. در نتیجه هدف ما پیدا کردن حد بهره ایست که سیستم را در حالت حلقه بسته پایدار کند.

اولین کاری که باید انجام شود، پیدا کردن مقدار قطب‌های حقیقی مثبت در تابع تبدیل حلقه باز می‌باشد:

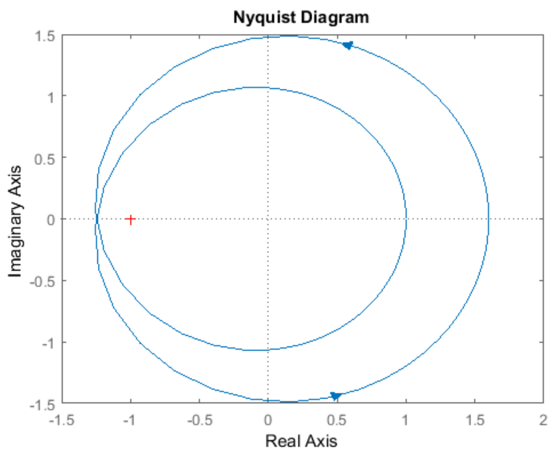
```
roots([1 -8 15])
```

```
ans =

     5

     3
```

هر دو قطب تابع تبدیل حلقه باز مثبت می‌باشند. بنابراین ما به دو چرخش پادساعتگرد (N=-2) نمودار نایکوئست حول نقطه ۱- نیاز داریم تا سیستم حلقه بسته پایدار شود (Z=P+N). اگر تعداد چرخش کمتر از ۲ باشد یا چرخش پادساعتگرد نباشد، سیستم ما ناپایدار خواهد بود.

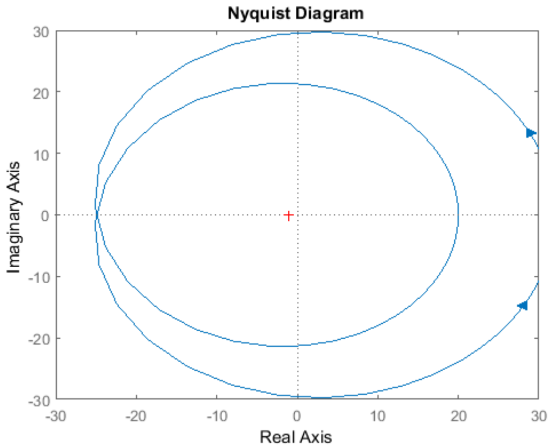
حال نمودار نایکوئست با بهره ۱ را رسم می‌نماییم:

```
G = (s^2 + 10*s + 24)/(s^2 - 8*s + 15);

nyquist(G)
```



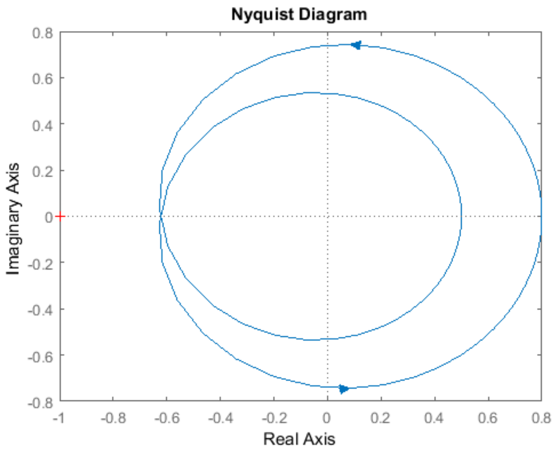
در این نمودار دارای دو چرخش پادساعتگرد حول ۱- هستیم. در نتیجه سیستم برای بهره ۱ پایدار می‌باشد. حال می‌بینیم که سیستم با افزایش بهره به مقدار ۲۰، چطور رفتار می‌کند:

```
nyquist(20*G)
```



با افزایش بهره، دیاگرام گسترش می‌یابد. در نتیجه متوجه می‌شویم که فارغ از افزایش بهره، سیستم حلقه بسته پایدار می‌ماند. اگرچه ممکن است با کاهش بهره، نمودار منقبض شود و ناپایدار گردد. نمودار را برای بهره برابر ۰/۵ رسم می‌نماییم:

```
nyquist(0.5*G)
```



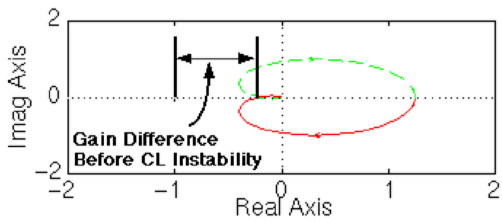
حال سیستم ما ناپایدار است. با سعی و خطا متوجه می‌شویم سیستم با بهره‌های کمتر از ۰/۸۰ ناپایدار می‌گردد. می‌توانیم این مقدار را با بزرگنمایی نمودار نایکوئست و همچنین بررسی پاسخ پله حلقه بسته برای بهره‌های ۰/۷۹، ۰/۸۰ و ۰/۸۱ بازبینی نماییم.

حد بهره

حد بهره را به صورت مقدار تغییری که لازم است در بهره حلقه باز (به دسیبل) در فاز ۱۸۰- درجه بدهیم تا اندازه حلقه باز برابر ۰ دسیبل گردد تعریف می‌نماییم. حال می‌خواهیم ریشه‌ی این تعریف را بیابیم. ابتدا سیستمی را در نظر می‌گیریم که اگر نمودار نایکوئست چرخشی حول نقطه ۱- نداشته باشد سیستم پایدار شود:

$$G(s) = \frac{50}{s^3 + 9s^2 + 30s + 40} \qquad (۹)$$

با به دست آوردن ریشه‌ها، متوجه می‌شویم قطب حلقه بازی در صفحه سمت راست نداریم. بنابراین اگر نمودار نایکوئست چرخشی حول نقطه ۱- نداشته باشد، سیستم حلقه بسته قطبی در صفحه سمت راست ندارد. حال می‌خواهیم بدانیم چقدر می‌توان بهره را تغییر داد تا سیستم حلقه بسته ناپایدار گردد؟ به شکل زیر نگاه کنید:



سیستم حلقه باز نشان داده شده در نمودار بالا، با بیشتر شدن بهره از یک مرز مشخص، در ساختار حلقه بسته ناپایدار می‌گردد. ناحیه محور حقیقی منفی بین مقدار $-1/a$ (نقطه‌ای که سیستم حلقه باز دارای فاز ۱۸۰- درجه می‌باشد که همان نقطه‌ایست که منحنی از محور حقیقی عبور می‌کند) و نقطه ۱- بیانگر مقدار مجاز افزایش بهره می‌باشد که سیستم حلقه بسته پایدار بماند.

اگر دقت کنیم، با اضافه کردن بهره‌ای به اندازه a، نمودار با نقطه ۱- تماس پیدا می‌کند:

$$G(j\omega) = \frac{-1}{a} \qquad (۱۰)$$

یا به عبارت دیگر:

$$aG(j\omega) = -1 \qquad (۱۱)$$

بنابراین می‌توان گفت که حد بهره برابر a واحد است. قبلا ذکر کردیم که حد بهره بر واحد دسیبل اندازه گیری می‌شود. در نتیجه حد بهره برابر است با:

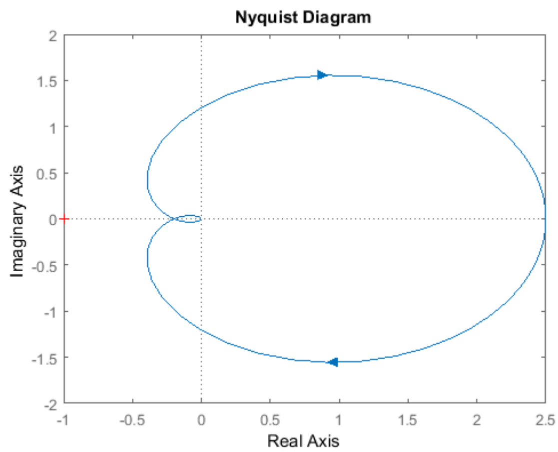
$$GM = 20\log_{10}(a)\ dB \qquad (۱۲)$$

حال حد بهره تابع تبدیل حلقه باز پایداری که در قسمت قبل به دست آورده شد را پیدا می‌کنیم. تابع تبدیل قسمت قبل به شکل زیر تعریف شد:

$$G(s) = \frac{50}{s^3 + 9s^2 + 30s + 40} \qquad (۱۳)$$

با اجرای دستور زیر، نمودار نایکوئست به دست می‌آید:

```
G = 50/(s^3 + 9*s^2 + 30*s + 20);

nyquist(G)
```



همانطور که قبلا بحث شد، برای به دست آوردن حد بهره، باید مقدار a را که در مورد آن توضیح داده شد، به دست آورد. برای این منظور باید نقطه‌ای که دقیقا ۱۸۰- درجه اختلاف فاز وجود دارد را پیدا کنیم زیرا در این نقطه تابع تبدیل تنها دارای قسمت حقیقی (بدون قسمت موهومی) است. صورت تابع تبدیل حقیقی می‌باشد، پس در نتیجه باید مخرج آنرا بررسی نمود. هنگامی که $s = j\omega$، تنها عبارات موجود در مخرج که دارای قسمت‌های موهومی می‌باشند، آنهایی هستند که دارای توان‌های فردی از s می‌باشند. پس برای اینکه $G(j\omega)$ حقیقی باشد، باید داشته باشیم:

$$-j\omega^3 + 30j\omega = 0 \qquad (۱۴)$$

که به معنی $\omega = 0$ (سمت راست ترین نقطه در نمودار نایکوئست) یا $\omega = \sqrt{30}$ می‌باشد. پس می‌توان مقدار $G(j\omega)$ در این نقطه را با استفاده از دستور `polyval` به دست آورد:

```
w = sqrt(30);

polyval(50,j*w)/polyval([1 9 30 40],j*w)
```

```
ans =

   -0.2174
```

پاسخ به دست آمده برابر $-0.2174 + 0i$ می‌باشد. قسمت موهومی برابر صفر می‌باشد پس متوجه می‌شویم جواب به دست آمده صحیح می‌باشد. این مقدار را می‌توان با مشاهده نمودار نایکوئست نیز بررسی نمود. همچنین قسمت حقیقی به دست آمده معقول می‌باشد. در مرحله بعد به به دست آوردن حد بهره می‌پردازیم.

پس می‌دانیم که اختلاف فاز ۱۸۰- درجه در نقطه $-0.2174 + 0i$ ایجاد می‌شود. این نقطه را از قبل به صورت $-1/a$ تعریف نمودیم. در نتیجه با به دست آوردن a همان حد بهره به دست می‌آید. در نهایت می‌دانیم که حد بهره بر حسب دسیبل بیان می‌شود:

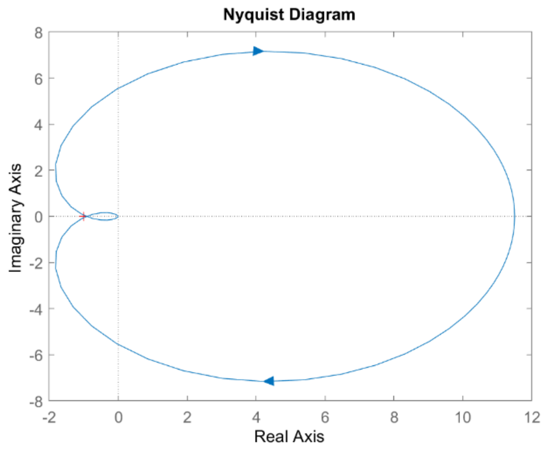
$$\frac{-1}{a} = -0.2174 \qquad (۱۵)$$

$$\Longrightarrow a = 4.6 \qquad (۱۶)$$

$$\Longrightarrow GM = 20\log_{10}(4.6) = 13.26\ dB \qquad (۱۷)$$

حال که حد بهره را داریم، دقت این جواب را با استفاده از بهره $a = 4.6$ و بزرگنمایی نمودار نایکوئست می‌سنجیم:

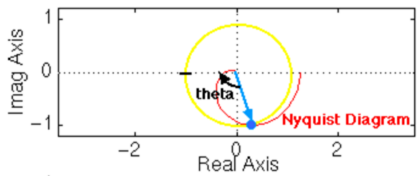
```
a = 4.6;

nyquist(a*sys)
```



در نمودار به دست آمده به نظر می‌آید که منحنی دقیقا از نقطه ۱- عبور کرده است. با استفاده از بهره‌های 4.5، 4.6 و 4.7 در نمودارهای نایکوئست بزرگنمایی شده و همچنین پاسخ پله حلقه بسته، دقت نتایج به دست آمده را بررسی نمود.

حد فاز

با بحث‌های انجام شده، اهمیت حد فاز بر ما پوشیده نمی‌باشد پس تنها به منشا مفاهیم آن می‌پردازیم. حد فاز را به صورت تغییراتی که لازم است در فاز حلقه باز با بهره واحد داده شود تا سیستم حلقه بسته ناپایدار گردد تعریف می‌نماییم. به تعریف گرافیکی این مفهوم که در زیر آمده است توجه کنید.



حال این نمودار را تحلیل می‌کنیم. از مثال قبل می‌دانیم که در صورتی که نمودار نایکوئست نقطه ۱- را دور بزند، سیستم حلقه بسته ناپایدار می‌گردد. همچنین در صورتی که منحنی به اندازه‌ی theta درجه جابجا شود، با نقطه‌ی ۱- تماس پیدا می‌کند که سبب می‌شود تا سیستم حلقه بسته پایدار مرزی شود. بنابراین مقدار زاویه لازم برای اینکه سیستم حلقه بسته پایدار مرزی شود حد فاز (بر حسب درجه) نامیده می‌شود. برای اینکه این زاویه را پیدا کنیم، دایره‌ای را به شعاع ۱ از مبدا مختصات رسم کرده و بدین صورت نقطه‌ای از منحنی نایکوئست را که دارای اندازه ۱ (بهره ۰ دسیبل) می‌باشد را می‌یابیم و مقدار جابجایی فاز از این نقطه برای رسیدن به زاویه ۱۸۰- درجه را به دست می‌آوریم.

بخش ششم: مقدمه‌ای بر طراحی کنترلر در فضای حالت

فهرست مطالب بخش

- مدل‌سازی
- پایداری
- کنترل‌پذیری و مشاهده‌پذیری
- طراحی کنترل بوسیله جایدهی قطب
- ورودی مرجع
- طراحی مشاهده‌گر

در این بخش به روش‌های فضای حالت برای طراحی کنترلر و مشاهده‌گر می‌پردازیم.

دستورهای کلیدی متلب در این بخش:

```
eig, ss, lsim, place, acker
```

مدل‌سازی

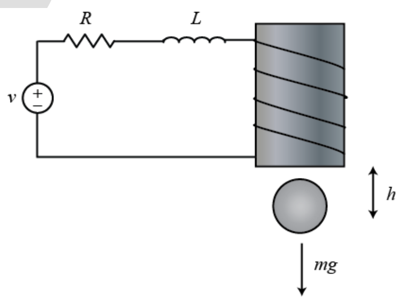
روش‌های مختلفی برای تعریف یک سیستم معادلات دیفرانسیل خطی وجود دارد. نمایش فضای حالت در **بخش اول: مدل‌سازی سیستم** معرفی گردید. برای یک سیستم SISO LTI، فرم فضای حالت به شکل زیر تعریف می‌شود:

$$\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + Bu \tag{۱}$$

$$y = C\boldsymbol{x} + Du \tag{۲}$$

که $\boldsymbol{x}$ یک بردار n در ۱ و نمایانگر متغیرهای حالت سیستم، u مقدار اسکالر ورودی و y مقدار اسکالر خروجی می‌باشد. ماتریس‌های A (n در n)، B (n در ۱) و C (۱ در n) ارتباط بین متغیرهای حالت و ورودی و خروجی را مشخص می‌کنند. در این فرم، ما دارای n معادله دیفرانسیل مرتبه اول می‌باشیم. نمایش فضای حالت را می‌توان برای سیستم‌های با چند ورودی و چند خروجی (MIMO) نیز استفاده نمود، اما در این کتاب تمرکز ما بر روی سیستم‌های تک ورودی و تک خروجی (SISO) می‌باشد.

برای معرفی روش طراحی کنترلر با فضای حالت، از مثال گوی معلق بوسیله میدان مغناطیسی استفاده می‌نماییم. جریان عبوری از سیم‌پیچ، نیروی مغناطیسی را القا می‌کند که سبب تعادل گوی (گوی از مواد مغناطیسی می‌باشد) و معلق ماندن آن در میان هوا می‌شود. مدل‌سازی این سیستم در بسیاری از کتب کنترل آورده شده است.



معادلات این سیستم به شکل زیر است:

$$m\frac{d^2h}{dt^2} = mg - \frac{Ki^2}{h} \tag{۳}$$

$$V = L\frac{di}{dt} + iR \tag{۴}$$

که h موقعیت عمودی گوی، i جریان عبوری از مدار، V ولتاژ عبوری، m جرم گوی، g شتاب گرانش، L اندوکتانس، R مقاومت و K ضریبی است که نیروی مغناطیسی اعمال شده به گوی را تعریف می‌کند، می‌باشد. برای ساده شدن مسئله، از مقادیر $m = 0.05kg$، $K = 0.0001$، $L = 0.01H$، $R = 1\ Ohm$ و $g = 9.81\ m/s^2$ استفاده می‌نماییم. هنگامی که $h = ki^2/mg$ (نقطه‌ای که $\frac{dh}{dt} = 0$ است) برقرار باشد سیستم در تعادل (گوی معلق در هوا) است. با خطی‌سازی معادلات حول نقطه $h = 0.01m$ (با جریان نامی ۷ آمپر)، معادلات فضای حالت خطی را به دست می‌آوریم:

$$\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + Bu \tag{۵}$$

$$y = C\boldsymbol{x} + Du \tag{۶}$$

که در آن:

$$x = \begin{bmatrix} \Delta h \\ \Delta \dot{h} \\ \Delta i \end{bmatrix} \tag{۷}$$

مجموعه‌ای از متغیرهای حالت سیستم (بردار ۳ در ۱)، u اختلاف ولتاژ ورودی از حالت تعادل (ΔV) و y انحراف ارتفاع توپ از ارتفاع اولیه (Δh) می‌باشد. ماتریس‌های سیستم را در یک ام‌فایل وارد می‌کنیم:

```
A = [ 0   1   0

     980  0  -2.8

      0   0  -100 ];



B = [ 0

      0

      100 ];



C = [ 1 0 0 ];
```

پایداری

اولین تحلیلی که لازم است انجام شود این است که آیا سیستم حلقه باز (بدون هیچگونه کنترل) پایدار است یا خیر. همانطور که در **بخش دوم: تحلیل سیستم** گفته شد، مقادیر ویژه ماتریس سیستم، A (معادل قطب‌های تابع تبدیل) پایداری سیستم را تعیین می‌کنند. مقادیر ویژه ماتریس A مقادیری از s می‌باشند که در معادله $\det(sI - A) = 0$ صدق می‌کنند.

```
poles = eig(A)
```

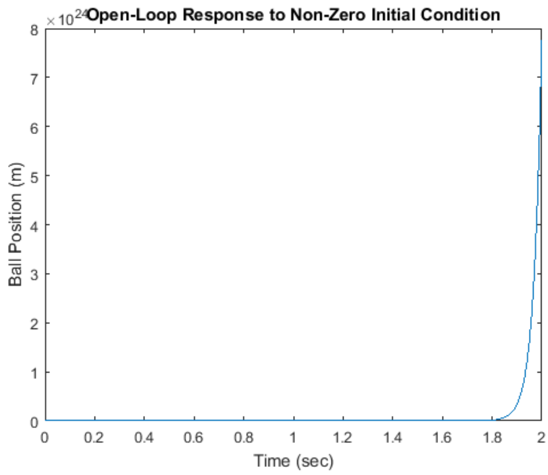
```
poles =
```

```
   31.3050

  -31.3050

 -100.0000
```

با بررسی انجام شده، درمی‌یابیم که یکی از قطب‌های سیستم در صفحه سمت راست (دارای قسمت حقیقی مثبت) قرار دارد که به معنی ناپایداری سیستم حلقه باز می‌باشد.

برای اینکه رفتار این سیستم ناپایدار را با شرایط اولیه غیر صفر مشاهده کنیم، خطوط زیر را به ام‌فایل خود اضافه می‌نماییم:

```
t = 0:0.01:2;

u = zeros(size(t));

x0 = [0.01 0 0];



sys = ss(A,B,C,0);



[y,t,x] = lsim(sys,u,t,x0);

plot(t,y)

title('Open-Loop Response to Non-Zero Initial Condition')
xlabel('Time (sec)')
ylabel('Ball Position (m)')
```



مشاهده می‌شود که فاصله‌ی بین سیم‌پیچ و گوی به سمت بینهایت میل می‌کند، که در واقعیت احتمالا توپ به زمین یا میز زیر خود برخورد می‌نماید (و احتمالا از محدوده‌ی مجاز خطی سازی خارج می‌شود).

کنترل‌پذیری و مشاهده‌پذیری

سیستمی را کنترل‌پذیر گویند که همواره ورودی کنترلی $u(t)$ وجود داشته باشد که در زمان محدود، هر یک از حالت‌های سیستم را به سایر حالت‌ها تبدیل کند. می‌توان نشان داد که یک سیستم LTI هنگامی کنترل‌پذیر است که اگر و تنها اگر

ماتریس کنترل‌پذیری آن، یعنی $\mathcal{C}$، رنک کامل باشد (به معنی این است که اگر $rank(\mathcal{C}) = n$ باشد آنگاه n برابر با تعداد متغیرهای حالت سیستم است). برای به دست آوردن رنک ماتریس کنترل‌پذیری یک سیستم LIT در متلب از دستور `rank(ctrb(A,B))` یا `rank(ctrb(sys))` استفاده می‌نماییم.

$$\mathcal{C} = [B \quad AB \quad A^2B \quad \cdots \quad A^{(n-1)}B] \tag{۸}$$

ممکن است در یک سیستم تمامی متغیرهای حالت آن به طور مستقیم قابل اندازه‌گیری نباشند. در این موارد لازم است تا مقادیر متغیرهای حالت داخلی مجهول با استفاده از خروجی‌های در دسترس سیستم تخمین زده شود.

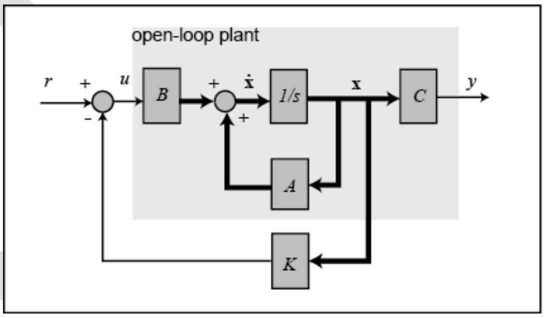
سیستمی مشاهده‌پذیر است که حالت اولیه آن ($x(t_0)$) را بتوان با استفاده از ورودی ($u(t)$) و خروجی ($y(t)$) سیستم و در بازه‌ی زمانی محدود $t_0 < t < t_f$ به دست آورد. برای سیستم‌های LTI، سیستم مشاهده‌پذیر است اگر و تنها اگر ماتریس مشاهده‌پذیری $\mathcal{O}$ دارای رنک کامل باشد (به معنی این است که اگر $\mathrm{rank}(\mathcal{O}) = \mathrm{n}$ باشد آنگاه n برابر با تعداد متغیرهای حالت سیستم است). مشاهده‌پذیری یک مدل LTI را می‌توان با استفاده از دستور `rank(obsv(A,C))` یا `rank(obsv(sys))` در متلب به دست آورد.

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{۹}$$

مفاهیم کنترل‌پذیری و مشاهده‌پذیری دو مفهوم جدا نشدنی هستند. یک سیستم (A و B) کنترل‌پذیر است اگر و تنها اگر یک سیستم (A′ و B′) مشاهده‌پذیر باشد. از این قضیه در طراحی مشاهده‌گر استفاده خواهیم نمود.

طراحی کنترل بوسیله جایدهی قطب

در این قسمت برای سیستم زیر با روش جایدهی قطب، کنترلی را طراحی می‌نماییم. شماتیک سیستم با فیدبک از تمامی حالت‌ها را در زیر مشاهده می‌نمایید. منظور از تمامی حالت‌ها این است که تمامی متغیرهای حالت در تمامی زمان‌ها در اختیار کنترلر قرار دارند. برای این سیستم به سنسوری برای اندازه‌گیری موقعیت گوی، سنسور دیگری برای اندازه‌گیری سرعت گوی و سومین سنسور برای اندازه‌گیری جریان موجود در سیم‌پیچ نیاز داریم.



برای ساده‌سازی، فرض می‌کنیم ورودی مرجع صفر ($r = 0$) می‌باشد. در نتیجه ورودی برابر است با:

$$u = -Kx \tag{۱۰}$$

معادلات فضای حالت برای سیستم فیدبک حلقه بسته عبارتند از:

$$\dot{x} = Ax + B(-Kx) = (A - BK)x \tag{۱۱}$$

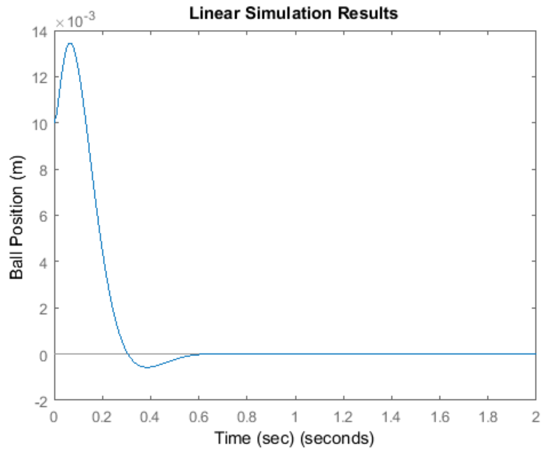
$$y = Cx \tag{۱۲}$$

پایداری و عملکرد در حوزه زمان این سیستم فیدبک حلقه بسته توسط موقعیت مقادیر ویژه ماتریس $A - BK$ به دست می‌آیند که این مقادیر همان قطب‌های سیستم حلقه بسته می‌باشند. از آنجایی که ماتریس‌های A و BK هر دو ۳x۳ می‌باشند این سیستم دارای ۳ قطب است. با انتخاب مناسب ماتریس بهره فیدبک K، می‌توان قطب‌های حلقه بسته را

در هر مکان دلخواه (زیرا سیستم کنترل‌پذیر است) قرار داد. برای پیدا کردن ماتریس بهره فیدبک K که قطب‌های حلقه بسته مطلوب را ایجاد می‌کنند، می‌توان از تابع `place` در متلب استفاده نمود.

قبل از اجرای این روش، باید تصمیم بگیریم که موقعیت قطب‌های حلقه بسته در چه مکانی باشد. نیازهای طراحی را برابر با زمان نشست کمتر از ۰/۵ ثانیه و فراجهش کمتر از ۵٪ در نظر بگیرید. در این صورت ما می‌خواهیم دو قطب غالب در $-10 \pm 10i$ (یعنی در $\zeta = 0.7$ یا زاویه ۴۵ درجه و $\sigma = 10 > 4.6 * 2$) قرار گیرند. همچنین برای شروع قطب سوم را در -50 در نظر می‌گیریم (زیرا به اندازه کافی سریع باشد تا تاثیری بر روی پاسخ نگذارد) و بعدا با توجه به رفتار حلقه بسته به دست آمده، آنرا تغییر می‌دهیم. از دستور `lsim` به بعد را از ام‌فایل خود حذف کرده و خطوط زیر را به آن اضافه کنید:

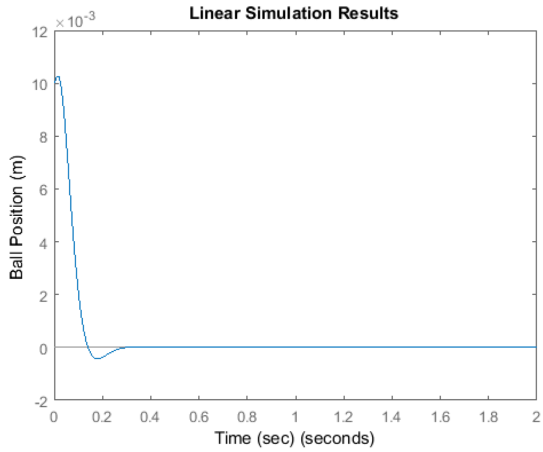
```
p1 = -10 + 10i;

p2 = -10 - 10i;

p3 = -50;



K = place(A,B,[p1 p2 p3]);

sys_cl = ss(A-B*K,B,C,0);



lsim(sys_cl,u,t,x0);

xlabel('Time (sec)')
ylabel('Ball Position (m)')
```



با بررسی نمودار بالا می‌توان دید که مقدار فراجهش بسیار زیاد است (در تابع تبدیل چند صفر وجود دارد که بر روی فراجهش تاثیر می‌گذارند و در معادلات فضای حالت این صفرها صریحا دیده نمی‌شوند). قطب‌ها را کمی به سمت چپ تغییر داده و تاثیر آنرا بر روی پاسخ گذرا مشاهده می‌کنیم (با اینکار پاسخ سریعتر می‌شود).

```
p1 = -20 + 20i;

p2 = -20 - 20i;

p3 = -100;
```

```
K = place(A,B,[p1 p2 p3]);

sys_cl = ss(A-B*K,B,C,0);

lsim(sys_cl,u,t,x0);

xlabel('Time (sec)')
ylabel('Ball Position (m)')
```



این بار مقدار فراجهش کاهش پیدا کرد. برای اطلاعات بیشتر برای انتخاب قطب‌های حلقه بسته‌ی مطلوب، به کتب مرجع مراجعه کنید.

در هر دو حالت تلاش کنترلی[۲۱] (u) را به دست آورید. به طور کلی هرچه قطب‌ها به سمت چپ بروند تلاش کنترلی بیشتری مورد نیاز است.

نکته: اگر می‌خواهید دو یا چند قطب را در یک نقطه قرار دهید نمی‌توانید از دستور `place` استفاده کنید. برای این منظور از تابع `acker` استفاده کنید (این دستور مانند دستور `place` می‌باشد با این تفاوت که از لحاظ عددی ضعیف‌تر است):

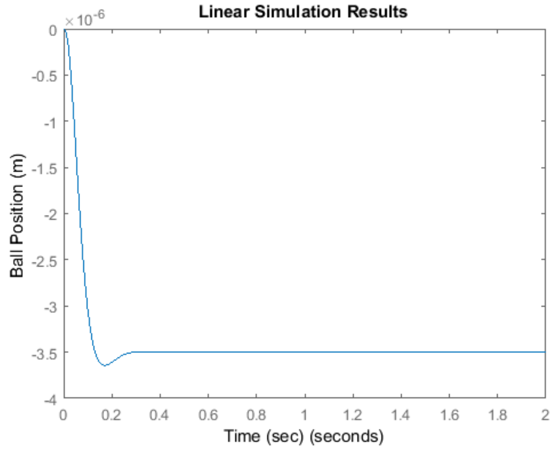
```
K = acker(A,B,[p1 p2 p3])
```

ورودی مرجع

سیستم گفته شده در قبل را در نظر می‌گیریم و ورودی پله را اعمال می‌کنیم (مقدار پله را کم انتخاب می‌کنیم تا سیستم در ناحیه خطی‌سازی شده باقی بماند). مقدار t، u و lsim را در امفایل به شکل زیر جاگذاری کنید:

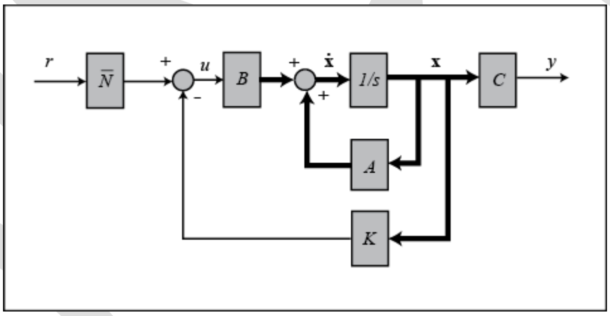
```
t = 0:0.01:2;

u = 0.001*ones(size(t));

sys_cl = ss(A-B*K,B,C,0);

lsim(sys_cl,u,t);

xlabel('Time (sec)')
ylabel('Ball Position (m)')
axis([0 2 -4E-6 0])
```

[۲۱] Control Effort



سیستم مورد نظر ورودی پله را به خوبی دنبال نمی‌کند. خروجی سیستم نه تنها مقدار آن یک نمی‌باشد بلکه علامت آن به جای مثبت، منفی است!

اگر به شماتیک دقت کنیم متوجه می‌شویم که ما خروجی را با مرجع مقایسه نمی‌کنیم، بلکه ما تمامی حالت‌ها را اندازه گرفته و در بردار بهره K ضرب کرده و آنگاه حاصل آنرا از مرجع کم می‌کنیم. دلیلی ندارد تا مقدار Kx برابر با مقدار خروجی مطلوب شود. برای رفع این مشکل، می‌توان ورودی مرجع را متناسب با Kx در حالت ماندگار نماییم. ضریب تناسب[۲۲] $\bar{N}$ در شماتیک زیر نشان داده شده است:



در متلب با دستور `rscale` (موجود در سی‌دی) مقدار $\bar{N}$ را حساب می‌کنیم (خط زیر را بعد از کد ... = K قرار دهید):

```
Nbar = rscale(sys,K)
```

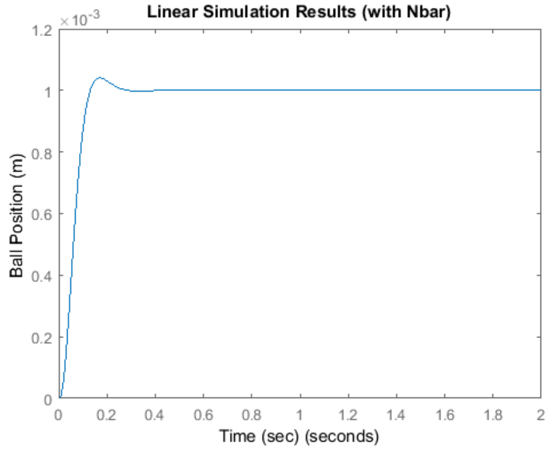
```
Nbar =

 -285.7143
```

لازم به ذکر است که این تابع، یک تابع استاندارد متلب نمی‌باشد و لازم است تا آنرا از سی‌دی کپی کرده و در مسیر فضای کاری کنونی خود ذخیره کنید. برای به دست آوردن پاسخ سیستم با این ضریب تناسب، می‌توان ورودی را در مقدار $\bar{N}$ ضرب نمود:

```
lsim(sys_cl,Nbar*u,t)

title('Linear Simulation Results (with Nbar)')
xlabel('Time (sec)')
ylabel('Ball Position (m)')
```

[۲۲] Scale Factor

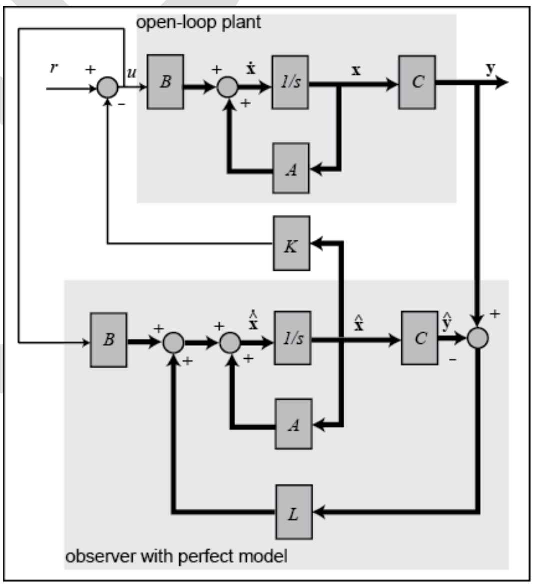
```
axis([0 2 0 1.2*10^-3])
```



در این حالت، پله به خوبی دنبال می‌شود. محاسبه‌ی ضریب تناسب به شناخت صحیحی از سیستم نیاز دارد. اگر خطایی در مدل ما وجود داشته باشد، آنگاه ورودی با مقدار نادرستی متناسب می‌شود. روش دیگر که در بخش معرفی کنترل PID گفته شد، اضافه کردن یک متغیر حالت برای انتگرال‌گیری از خطای خروجی می‌باشد. با اینکار اثر اضافه نمودن جمله‌ی انتگرالی را به کنترلر ایجاد می‌کنیم که همانطور که می‌دانیم سبب کاهش خطای حالت ماندگار می‌شود.

طراحی مشاهده‌گر

زمانی که قادر به اندازه‌گیری تمامی متغیرهای حالت x نمی‌باشیم (معمولا در عمل به این مشکل بر خواهیم خورد) می‌توان از یک **مشاهده‌گر** و تنها اندازه‌گیری خروجی سیستم $y = Cx$، برای تخمین متغیرهای حالت استفاده نمود. در مثال گوی مغناطیسی، به سیستم سه متغیر حالت تخمین زده شده ($\hat{x}$) اضافه می‌نماییم. شماتیک این مسئله به شکل زیر در می‌آید:



اساسا یک مشاهده‌گر، مشابه سیستم می‌باشد زیرا هر دوی آنها دارای یک ورودی و تقریبا یک معادله دیفرانسیل می‌باشند. در مشاهده‌گر یک عبارت اضافی وجود دارد که مقدار خروجی واقعی اندازه‌گیری شده y را با مقدار خروجی تخمین زده شده $\hat{y} = C\hat{x}$ مقایسه می‌نماید. این عبارت به تصحیح حالت‌های تخمین زده شده $\hat{x}$ کمک نموده و سبب میل کردن مقادیر آنها به مقادیر حالت واقعی x می‌شود (با فرض خطای کوچک در اندازه‌گیری):

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \quad (۱۳)$$

$$\hat{y} = C\hat{x} \tag{۱۴}$$

دینامیک خطای مشاهده‌گر توسط قطب‌های معادله $A - LC$ به دست می‌آید:

$$\dot{e} = \dot{x} - \dot{\hat{x}} = (A - LC)e \tag{۱۵}$$

در قدم اول باید بهره‌ی مشاهده‌گر L را تعیین نمود. برای اینکه می‌خواهیم دینامیک مشاهده‌گر سریعتر از خود سیستم باشد، باید قطب‌های آن حداقل ۵ برابر دورتر از قطب‌های غالب سیستم باشد. همانطور که ذکر شد در استفاده از دستور `place`، نباید چند قطب در یک نقطه باشد، پس:

```
op1 = -100;

op2 = -101;

op3 = -102;
```

به خاطر وابستگی بین کنترل‌پذیری و مشاهده‌پذیری، می‌توان از تکنیک استفاده شده برای به دست آوردن ماتریس کنترل استفاده نمود و به جای ماتریس B، ماتریس C را قرار داده و از ترانهاده‌ی آنها استفاده نمود:

```
L = place(A',C',[op1 op2 op3])';
```

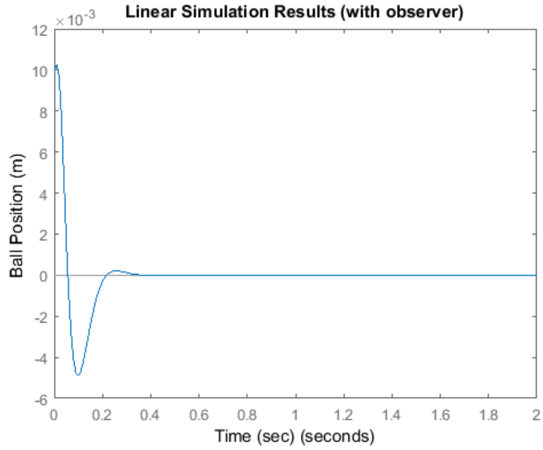
معادلات موجود در دیاگرام بلوکی بالا برای تخمین $\hat{x}$ به دست آمده‌اند. حال باید معادلات ترکیبی سیستم و مشاهده‌گر را با استفاده از معادلات حالت اصلی و خطای تخمین زده شده $e = x - \hat{x}$ نوشت. به علت اینکه لزوما تمامی متغیرهای حالت اندازه‌گیری نشده‌اند، ما از حالت تخمین زده شده برای فیدبک $u = -K\hat{x}$ استفاده می‌نماییم. بعد از چند مرحله محاسبه‌ی جبری، به معادلات ترکیبی حالت و خطا برای سیستم با مشاهده‌گر و فیدبک تمامی حالات خواهیم رسید.

```
At = [ A-B*K             B*K

       zeros(size(A))    A-L*C ];



Bt = [    B*Nbar

       zeros(size(B)) ];



Ct = [ C    zeros(size(C)) ];
```

برای مشاهده‌ی پاسخ به شرایط اولیه غیر صفر و بدون ورودی مرجع، خطوط زیر را به ام‌فایل خود اضافه کنید. در اینجا فرض می‌کنیم مشاهده‌گر با تخمین اولیه برابر با صفر شروع به کار کرده که در این صورت خطای تخمین زده‌ی اولیه برابر با بردار حالت اولیه $e = x$ خواهد شد.

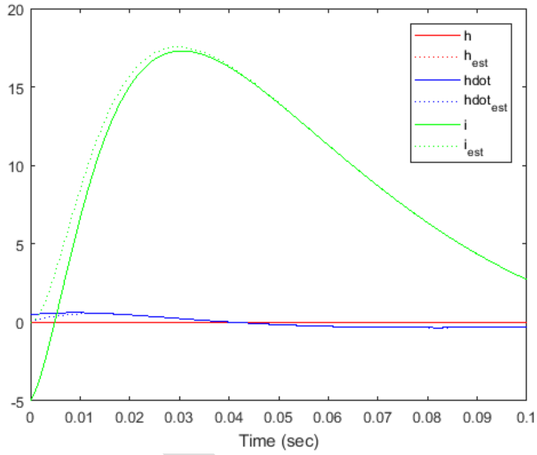
```
sys = ss(At,Bt,Ct,0);

lsim(sys,zeros(size(t)),t,[x0 x0]);
```

```
title('Linear Simulation Results (with observer)')
xlabel('Time (sec)')
ylabel('Ball Position (m)')
```



پاسخ تمامی حالات در نمودار زیر رسم شده است. یادآوری می‌کنیم که دستور `lsim` به ما x و e را می‌دهد و برای به دست آوردن $\hat{x}$ باید از معادله $x - e$ استفاده کرد.

```
t = 0:1E-6:0.1;

x0 = [0.01 0.5 -5];

[y,t,x] = lsim(sys,zeros(size(t)),t,[x0 x0]);

n = 3;

e = x(:,n+1:end);

x = x(:,1:n);

x_est = x - e;

% Save state variables explicitly to aid in plotting
h = x(:,1); h_dot = x(:,2); i = x(:,3);
h_est = x_est(:,1); h_dot_est = x_est(:,2); i_est = x_est(:,3);

plot(t,h,'-r',t,h_est,':r',t,h_dot,'-b',t,h_dot_est,':b',t,i,'-g',t,i_est,':g')
legend('h','h_{est}','hdot','hdot_{est}','i','i_{est}')
xlabel('Time (sec)')
```



از نمودار بالا مشاهده می‌شود که تخمین‌های مشاهده‌گر به سرعت به سمت مقادیر واقعی متغیرهای حالت همگرا شده و در حالت ماندگار به خوبی متغیرهای حالت را دنبال می‌کنند.

بخش هفتم: مقدمه‌ای بر طراحی کنترلر دیجیتال

فهرست مطالب بخش

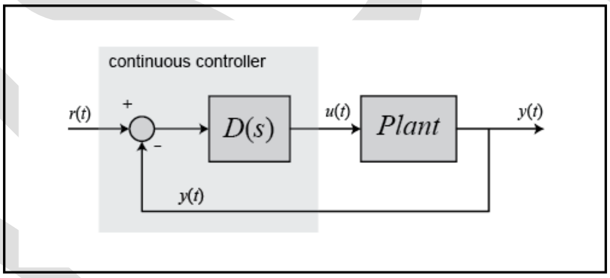
در این بخش به تبدیل مدل‌های پیوسته به مدل‌های گسسته (یا تبدیل معادله دیفرانسیل آنها به یکدیگر) می‌پردازیم. همچنین تبدیل z را معرفی نموده و روش استفاده از آن برای تحلیل و طراحی کنترلرهای گسسته را نشان می‌دهیم.

دستورهای کلیدی متلب در این بخش:

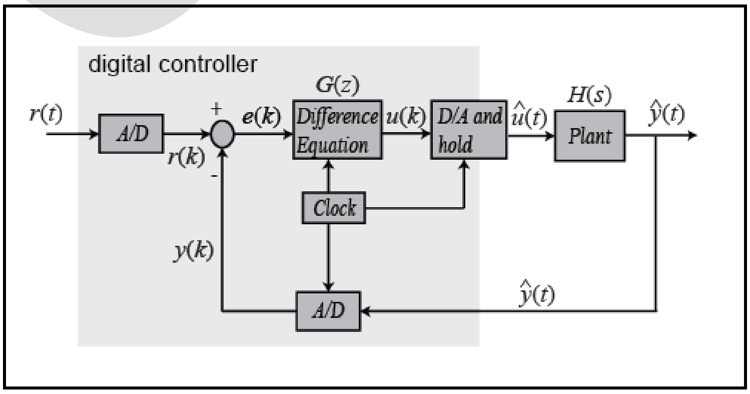
```
c2d, pzmap, zgrid, step, rlocus
```

مقدمه

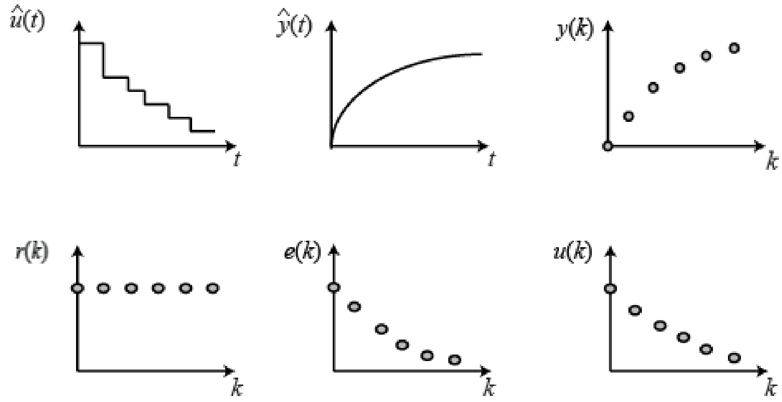
در شکل زیر یک سیستم فیدبک پیوسته که تا به حال در نظر می‌گرفتیم را مشاهده می‌کنید. تقریبا تمامی کنترلرهای پیوسته را می‌توان به وسیله‌ی قطعات الکترونیک آنالوگ پیاده‌سازی نمود.



کنترلر پیوسته که در شکل بالا در کادر تیره رنگ قرار دارد را می‌توان مانند شکل زیر با یک کنترلر دیجیتال جایگزین نمود که عملکرد این کنترلر مشابه کنترلر پیوسته می‌باشد. تفاوت اصلی این دو کنترلر این است که سیستم دیجیتال به جای سیگنال‌های پیوسته، به وسیله‌ی سیگنال‌های گسسته (نمونه‌هایی از سیگنال خوانده شده) کار می‌کند. اگر بخواهیم الگوریتم کنترل را در یک نرم افزار و بر روی یک کامپیوتر دیجیتال (که معمولا نیز به همین صورت است) اجرا کنیم لازم است تا این تغییرات اعمال شوند.



سیگنال‌های مختلف سیستم دیجیتال بالا را می‌توان در شکل زیر مشاهده نمود.

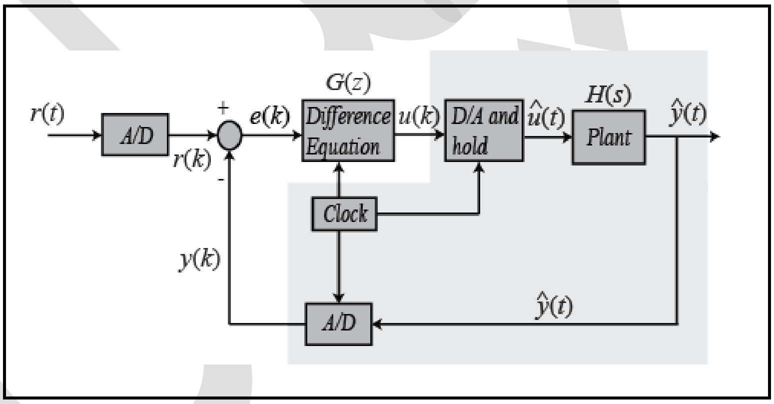


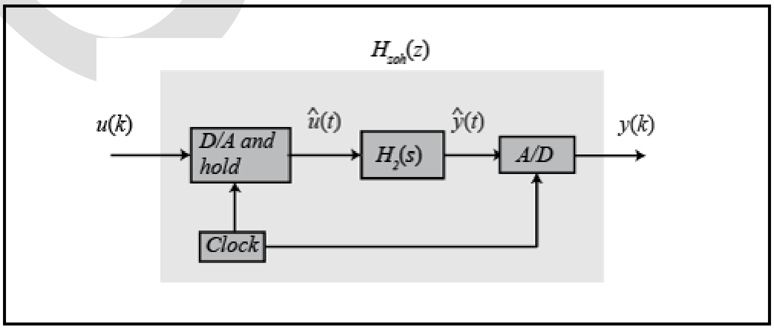
هدف از این بخش نشان دادن استفاده از متلب برای طراحی سیستم‌های کنترل دیجیتال در هنگام سر و کار داشتن با توابع گسسته، چه به شکل تابع تبدیل و چه به فرم فضای حالت می‌باشد.

معادل نگهدارنده‌ی صفر[۲۳]

در شکل قبل می‌بینیم که سیستم دارای بخش‌های گسسته و پیوسته می‌باشد. در حالت کلی سیستم‌های موجود در دنیای واقعی با استفاده از سیگنال‌های پیوسته با زمان کار کرده و پاسخ می‌دهند، در حالی که ممکن است الگوریتم کنترل بر روی یک کامپیوتر دیجیتال پیاده‌سازی شود. در هنگام طراحی یک سیستم کنترل دیجیتال ابتدا باید معادل گسسته‌ی قسمت‌های پیوسته سیستم را به دست آورد.

برای این کار، قسمت زیر از سیستم کنترل دیجیتال را در نظر گرفته و به شکل زیر مرتب می‌نماییم:





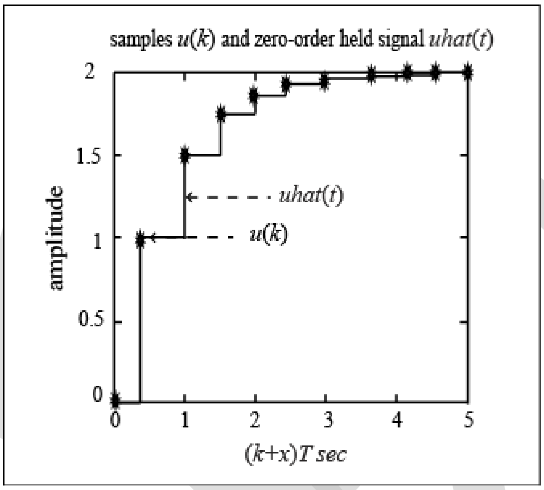
ساعت (Clock) متصل به **مبدل‌های A/D و D/A** (مبدل آنالوگ به دیجیتال و دیجیتال به آنالوگ) در هر T ثانیه یک پالس تولید کرده و هر مبدل D/A و A/D با دریافت یک پالس، سیگنالی را ارسال می‌نمایند. هدف از ایجاد این پالس این

---

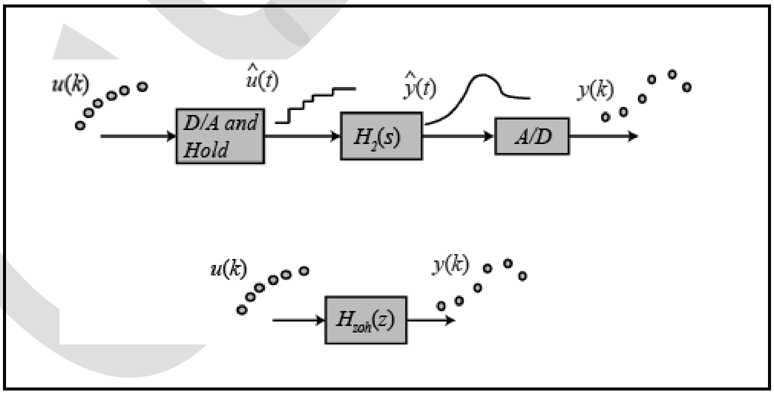
[۲۳] Zero-Hold Equivalence

است که $H_{zoh}(z)$ تنها در نمونه‌های ورودی تناوبی $u(k)$ عمل می‌کند و در نتیجه‌ی آن خروجی تناوبی $y(k)$ را تنها در بازه‌های گسسته‌ای از زمان تولید می‌کند. بنابراین $H_{zoh}(z)$ را می‌توان یک تابع گسسته دانست.

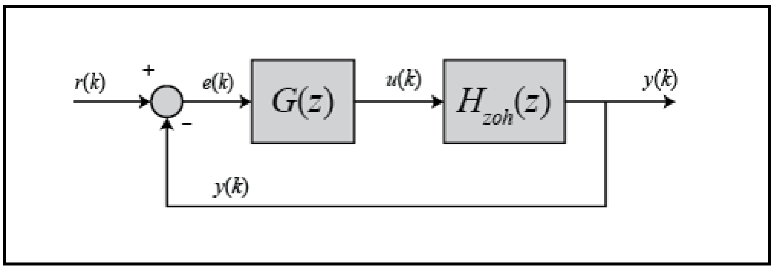
فلسفه‌ی طراحی این است که ما می‌خواهیم تابع گسسته $H_{zoh}(z)$ را به گونه‌ای پیدا کنیم که برای ورودی ثابت تکه‌ای به سیستم پیوسته $H(s)$، خروجی نمونه‌برداری شده از سیستم پیوسته برابر با خروجی سیستم گسسته باشد. فرض کنید سیگنال $u(k)$ بیانگر نمونه‌ای از سیگنال ورودی باشد. روش‌های مختلفی برای دریافت این نمونه $u(k)$ و تولید سیگنال پیوسته $\hat{u}(t)$ وجود دارد. در ترسیم زیر یک مثال از ورودی پیوسته $\hat{u}(t)$ که در هر نمونه در بازه kT تا $(k+1)T$، ثابت نگه داشته شده است $u(k)$ را مشاهده می‌کنید. به عملیات ثابت نگه داشتن $\hat{u}(t)$ در دوره‌ی نمونه‌برداری، روش **نگهدارنده‌ی مرتبه صفر**[۲۴] گفته می‌شود.



سپس سیگنال نگه داشته شده $\hat{u}(t)$ به $H_2(s)$ داده شده و مبدل A/D خروجی $y(k)$ را تولید می‌کند. این خروجی $y(k)$ تولید شده مشابه خروجی تولید شده از دادن سیگنال گسسته $u(k)$ به $H_{zoh}(z)$ می‌باشد.



شماتیک را با استفاده از $H_{zoh}(z)$ رسم می‌کنیم:



---

[۲۴] Zero-Order Hold

حال می‌توانیم سیستم کنترل دیجیتالی را طراحی کنیم که تنها با توابع گسسته سروکار دارد.

نکته: در مواردی پاسخ‌های گسسته با پاسخ‌های پیوسته تولید شده با مدار نگه‌دارنده استفاده شده در سیستم کنترل دیجیتال همخوانی ندارند. برای اطلاعات بیشتر به **پیوست چهارم: تاخیر ناشی از نگهدارنده** مراجعه کنید.

تبدیل با استفاده از c2d

با استفاده از تابع c2d در متلب می‌توان یک سیستم پیوسته (چه تابع تبدیل و چه فضای حالت) را به یک سیستم گسسته با روش نگهدارنده مرتبه صفر که در بالا توضیح داده شد، تبدیل کرد. دستور اصلی برای اینکار در متلب به شکل زیر است:

```
sys_d = c2d(sys,Ts,'zoh')
```

زمان نمونه‌برداری (Ts بر حسب sec/sample) باید کمتر از $1/(30BW)$ باشد که منظور از BW فرکانس پهنای باند سیستم حلقه بسته می‌باشد.

مثال: جرم-فنر-دمپر

تابع تبدیل

مدل تابع تبدیل پیوسته زیر را در نظر بگیرید:

$$\frac{X(s)}{F(S)}=\frac{1}{ms^2+bs+k} \tag{۱}$$

فرض می‌کنیم فرکانس پهنای باند حلقه بسته بیشتر از 1 rad/sec است. پس باید زمان نمونه برداری (Ts) را برابر 1/100 sec در نظر بگیریم. حال یک ام‌فایل با محتویات زیر ایجاد کنید:

```
m = 1;

b = 10;

k = 20;



s = tf('s');
sys = 1/(m*s^2+b*s+k);

Ts = 1/100;
sys_d = c2d(sys,Ts,'zoh')
```

```
sys_d =


  4.837e-05 z + 4.678e-05

  -----------------------

  z^2 - 1.903 z + 0.9048


Sample time: 0.01 seconds
```

```
Discrete-time transfer function.
```

فضای حالت

مدل فضای حالت پیوسته این سیستم به شکل زیر است:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t) \qquad (۲)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \qquad (۳)$$

تمامی مقادیر ثابت مانند قبل می‌باشند. ام‌فایل زیر مدل فضای حالت پیوسته بالا را به مدل فضای حالت گسسته تبدیل می‌کند:

```
A = [0    1; -k/m    -b/m];
B = [0;   1/m];
C = [1 0];
D = [0];

Ts = 1/100;

sys = ss(A,B,C,D);

sys_d = c2d(sys,Ts,'zoh')
```

```
sys_d =


  A =

               x1         x2

   x1       0.999   0.009513

   x2     -0.1903     0.9039


  B =

                u1

   x1   4.837e-05

   x2    0.009513


  C =

         x1   x2

   y1     1    0
```

```
  D =

       u1

   y1   0

Sample time: 0.01 seconds

Discrete-time state-space model.
```

با استفاده از ماتریس‌های به دست آمده، فضای حالت گسسته را می‌توان به شکل زیر نوشت:

$$\begin{bmatrix} x(k) \\ v(k) \end{bmatrix} = \begin{bmatrix} 0.9990 & 0.0095 \\ -0.1903 & 0.9039 \end{bmatrix} \begin{bmatrix} x(k-1) \\ v(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0.0095 \end{bmatrix} F(k-1) \tag{۴}$$

$$y(k-1) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x(k-1) \\ v(k-1) \end{bmatrix} \tag{۵}$$

حال مدل فضای حالت گسسته به دست آمده است.

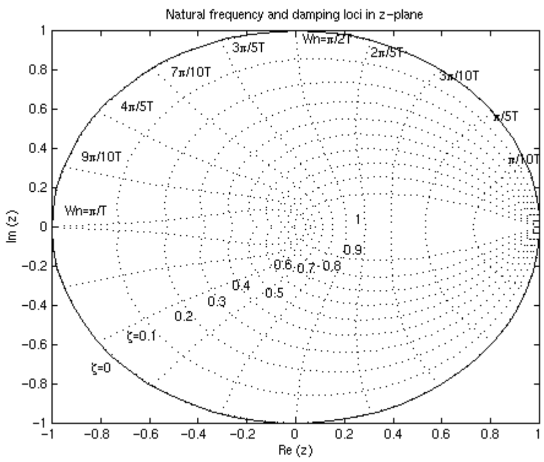
پاسخ گذرا و ماندگار

می‌دانیم برای سیستم‌های پیوسته به ازای موقعیت قطب‌ها در صفحه s، رفتارهای مختلفی به دست می‌آید. برای مثال در صورتی که هر یک از قطب‌های سیستم در سمت راست محور موهومی قرار بگیرد سیستم ناپایدار است. برای سیستم‌های گسسته می‌توانیم رفتار سیستم را بر اساس موقعیت قطب‌ها در صفحه z تحلیل کنیم. با تعاریف زیر می‌توان ویژگی‌های صفحه z را به ویژگی‌های صفحه s مرتبط نمود.

$$z = e^{sT} \tag{۶}$$

- T برابر زمان نمونه برداری (بر حسب sec/sample)
- s برابر موقعیت در صفحه s
- z برابر موقعیت در صفحه z

در شکل زیر خطوط با ضریب میرایی ثابت (ζ) و فرکانس طبیعی (ω_n) که از صفحه s به صفحه z نگاشت شده است را مشاهده می‌کنید.



می‌توان مشاهده نمود که مرز پایداری در صفحه z دیگر محور موهومی نمی‌باشد بلکه دایره‌ای به شعاع ۱ و به مرکز مبدا ($|z| = 1$) بوده که به نام دایره واحد از آن یاد می‌شود. یک سیستم گسسته زمانی پایدار است که تمامی قطب‌های آن در درون دایره واحد قرار گرفته و زمانی ناپایدار است که هر یک از قطب‌های آن بیرون از دایره قرار گرفته باشد.

برای بررسی پاسخ گذرا با توجه به موقعیت قطب‌ها در صفحه z، از سه معادله‌ی زیر که از سیستم‌های پیوسته داشتیم و در اینجا نیز صادق است استفاده می‌کنیم:

$$\zeta\omega_n \geq \frac{4.6}{T_s} \tag{۷}$$

$$\omega_n \geq \frac{1.8}{T_r} \tag{۸}$$

$$\zeta = \frac{-\ln(M_p)}{\sqrt{\pi^2 + \ln^2(M_p)}} \tag{۹}$$

که در آن:

- ζ برابر ضریب میرایی
- ω_n برابر فرکانس طبیعی (rad/sec)
- T_s زمان نشست ۱٪
- T_r زمان نمو ۱۰-۹۰٪
- M_p ماکزیمم فراجهش

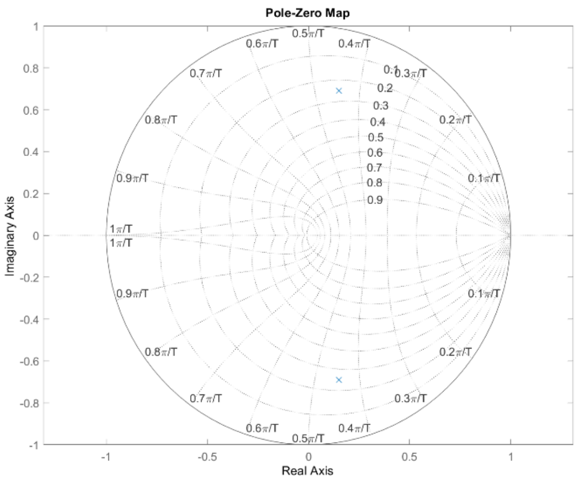
نکته مهم: فرکانس طبیعی (ω_n) در صفحه z بر واحد rad/sample رسم شده است اما در هنگام استفاده از معادلات بالا، فرکانس طبیعی باید بر واحد rad/sec استفاده شود.

فرض کنید تابع تبدیل گسسته زیر را داریم:

$$\frac{Y(z)}{F(z)} = \frac{1}{z^2 - 0.3z + 0.5} \tag{۱۰}$$

یک ام‌فایل جدید ساخته و دستورات زیر را در آن وارد می‌نماییم. با اجرای این ام‌فایل، نمودار زیر که دارای خطوط ضریب میرایی و فرکانس طبیعی ثابت است به دست می‌آید.

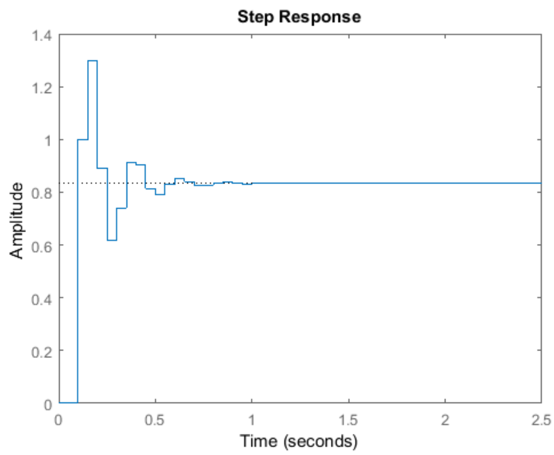
```
numDz = 1;
denDz = [1 -0.3 0.5];
sys = tf(numDz,denDz,-1); % the -1 indicates that the sample time is undetermined

pzmap(sys)
axis equal
zgrid
```



در این نمودار می‌توان مشاهده کرد که قطب‌ها تقریبا در فرکانس طبیعی $9\pi/20T$ (rad\sample) و ضریب میرایی ۰/۲۵ قرار گرفته‌اند. با در نظر گرفتن زمان نمونه برداری 1/20 sec (که مقدار $\omega_n = 28.2\ rad/sec$ را به دست می‌دهد) و با استفاده از سه معادله‌ی بالا، می‌توان دریافت که زمان نمو برابر ۰/۰۶ ثانیه، زمان نشست برابر ۰/۶۵ ثانیه و ماکزیمم درصد فراجهش برابر ۴۵٪ (۰/۴۵ برابر بیشتر از مقدار حالت ماندگار) می‌باشد.

حال برای سنجیدن مقادیر به دست آمده، پاسخ پله‌ی سیستم را رسم می‌نماییم. خطوط زیر را به ام‌فایل اضافه کرده و آنرا اجرا کنید. در این صورت پاسخ پله‌ی زیر به دست می‌آید.

```
sys = tf(numDz,denDz,1/20);

step(sys,2.5);
```



همانطور که می‌توان از نمودار دید زمان نمو، زمان نشست و فراجهش مطابق انتظار به دست آمد. با اینکار یاد گرفتیم که چطور با استفاده از موقعیت قطب‌ها و سه معادله‌ی گفته شده، پاسخ گذرای یک سیستم گسسته را تحلیل کنیم.

مکان هندسی گسسته ریشه‌ها

مکان هندسی ریشه‌ها عبارتست از مکان هندسی نقاطی که ریشه‌ها می‌توانند با تغییر دادن یک پارامتر از صفر تا بینهایت در آن قرار بگیرند. معادله مشخصه‌ی سیستم با فیدبک واحد با یک بهره‌ی تناسبی K برابر است با:

$$1 + KG(z)H_{zoh}(z) = 0 \quad (۱۱)$$

که $G(z)$ کنترلر دیجیتال و $H_{zoh}(z)$ تابع تبدیل سیستم در حوزه z (که با روش نگهدارنده مرتبه صفر به دست آمده است) می‌باشد.

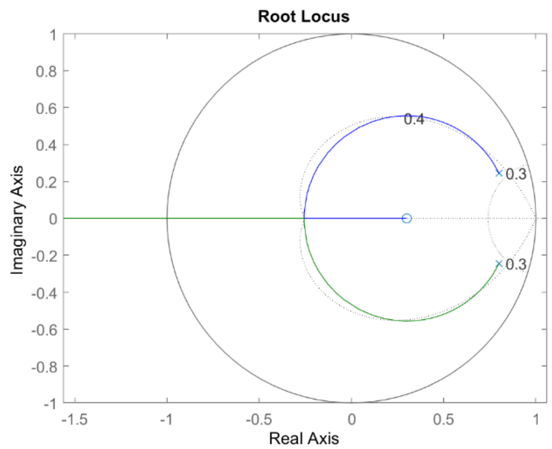
روش رسم مکان هندسی در صفحه z مانند روش استفاده شده در صفحه s می‌باشد. برای یادآوری بخش آموزش مکان هندسی ریشه‌ها، از دستور `sgrid` در متلب برای پیدا کردن نواحی مکان هندسی ریشه‌ها که بهره‌ی قابل قبول (K) را می‌دهد استفاده می‌کردیم. برای تحلیل مکان هندسی گسسته ریشه‌ها، از دستور `zgrid` که همان عملکرد `sgrid` را دارد استفاده می‌کنیم. دستور `zgrid(zeta ,Wn)` خطوط ضریب میرایی ثابت (zeta) و فرکانس طبیعی ثابت (Wn) را رسم می‌کند.

تابع تبدیل زیر را در نظر بگیرید:

$$\frac{Y(z)}{F(z)} = \frac{z-0.3}{z^2-1.6z+0.7} \tag{۱۲}$$

نیازهای طراحی، ضریب میرایی بزرگتر از ۰/۶ و فرکانس طبیعی بزرگتر از ۰/۴ rad/sample می‌باشد. دستورات زیر، مکان هندسی ریشه با خطوط ضریب میرایی ثابت و فرکانس طبیعی ثابت را رسم می‌کند. امفایل جدیدی را ساخته و دستورات زیر را وارد کنید. با اجرای این امفایل باید نمودار مکان هندسی ریشه‌ها به دست آید.

```
numDz = [1 -0.3];
denDz = [1 -1.6 0.7];
sys = tf(numDz,denDz,-1);

rlocus(sys)

axis equal

zeta = 0.4;
Wn = 0.3;
zgrid(zeta,Wn)
```



از این نمودار می‌توان مشاهده کرد که برای بعضی از مقادیر K سیستم ما پایدار است زیرا بخش‌هایی از مکان هندسی ریشه‌ها وجود دارد که هر دو شاخه‌ی آن درون دایره‌ی واحد قرار گرفته است. فرکانس طبیعی در خارج خط فرکانس طبیعی ثابت، بزرگتر از ۰/۳ بوده و ضریب میرایی درون خط ضریب میرایی ثابت، بزرگتر از ۰/۴ می‌باشد. در این مثال بخش‌هایی از مکان هندسی ریشه‌های به دست آمده، درون ناحیه مطلوب قرار گرفته‌اند. بنابراین بهره‌ی K را به گونه‌ای انتخاب می‌کنیم که دو قطب حلقه بسته درون ناحیه‌ی مطلوب که پاسخ رضایت بخشی را می‌دهند قرار بگیرند.

بخش هشتم: مقدمه‌ای بر مدل‌سازی سیمولینک

مدل‌سازی و شبیه‌سازی یک مدل ریاضی از سیستم فیزیکی در سیمولینک کار بسیار ساده‌ای می‌باشد. در محیط سیمولینک، مدل‌ها به طور گرافیکی با دیاگرام‌های بلوکی نمایش داده می‌شوند. گستره‌ی وسیعی از بلوک‌ها در قالب کتابخانه‌های طبقه‌بندی شده در انواع مختلف در اختیار کاربر قرار گرفته است. یکی از مزیت‌های اصلی استفاده از سیمولینک (و به طور کلی شبیه‌سازی) در تحلیل سیستم‌های دینامیکی این است که اجازه‌ی تحلیل سریع پاسخ سیستم‌های پیچیده که حل تحلیلی آنها بسیار دشوار است را می‌دهد. سیمولینک قادر است تا پاسخ‌های عددی تقریبی برای مدل‌های ریاضی را که ما نمی‌توانیم یا نمی‌خواهیم با دست محاسبه کنیم به دست آورد.

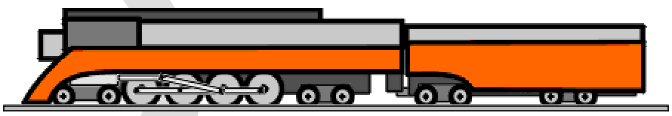
به طور کلی معادلات ریاضی یک سیستم مشخص که به عنوان قدم اول در مدل‌سازی سیمولینک استفاده می‌شود را می‌توان از قوانین فیزیکی استخراج نمود. در این بخش نشان می‌دهیم که چگونه این مدل ریاضی استخراج شده و سپس این مدل در سیمولینک وارد می‌شود. این مدل در فصل مقدمه‌ای بر کنترل سیمولینک برای نمایش طراحی و شبیه‌سازی کنترل در سیمولینک استفاده می‌گردد.

فهرست مطالب بخش

سیستم قطار

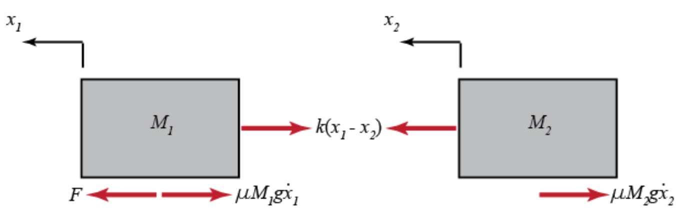
در این مثال یک قطار اسباب بازی را که دارای لوکوموتیو و واگن می‌باشد در نظر می‌گیریم. فرض کنیم که قطار تنها در یک بعد حرکت می‌کند (در راستای مسیر ریل). حال می‌خواهیم قطار را به گونه‌ای کنترل کنیم که به آرامی شروع به حرکت کرده و به آرامی متوقف گردد و همچنین بتواند با کمترین خطای حالت ماندگار، با یک سرعت ثابت حرکت نماید.

جرم لوکوموتیو و واگن را با M_1 و M_2 نمایش می‌دهیم. همچنین اتصال بین لوکوموتیو و واگن توسط اتصالی با سختی k انجام شده است. به بیان دیگر، این اتصال به عنوان یک فنر با ثابت k مدل شده است. نیروی F بیانگر نیروی اعمال شده بین چرخ‌های لوکوموتیو و ریل می‌باشد که μ ضریب اصطکاک غلتشی است.



دیاگرام جسم آزاد و قانون دوم نیوتن

اولین قدم برای به دست آوردن معادلات ریاضی حاکم بر سیستم فیزیکی، رسم دیاگرام جسم آزاد سیستم است. برای سیستم قطار، دیاگرام جسم آزاد در زیر رسم شده است.



از قانون دوم نیوتن می‌دانیم که برآیند نیروهای وارد بر یک جسم معادل است با حاصلضرب جرم جسم در شتاب آن. در این مسئله، نیروهای افقی وارد بر لوکوموتیو (M_1) شامل نیروی فنر، نیروی مقاومت غلتشی و نیروی حاصل از تماس

چرخ با ریل می‌باشد. نیروهای افقی وارد بر واگن (M_2) شامل نیروی فنر و نیروی مقاومت غلتشی می‌باشد. در راستای عمودی نیروی وزن با نیروی عمودی سطح در تعادل است ($N = mg$). بنابراین در راستای عمودی شتابی وجود ندارد.

فنر را به صورت نیرویی که به طور خطی با تغییر طول فنر تغییر می‌کند در نظر می‌گیریم یعنی $k(x_1 - x_2)$ که x_1 و x_2 به ترتیب جابجایی لوکوموتیو و واگن می‌باشد. در اینجا فرض می‌کنیم که در صورتی که x_1 و x_2 برابر با صفر باشند فنر در حالت استراحت قرار دارد. نیروی مقاومت غلتشی نیز به صورت نیرویی که به طور خطی از حاصلضرب سرعت و نیروی عمودی سطح (که همان نیروی وزن است) به دست می‌آید مدل شده است.

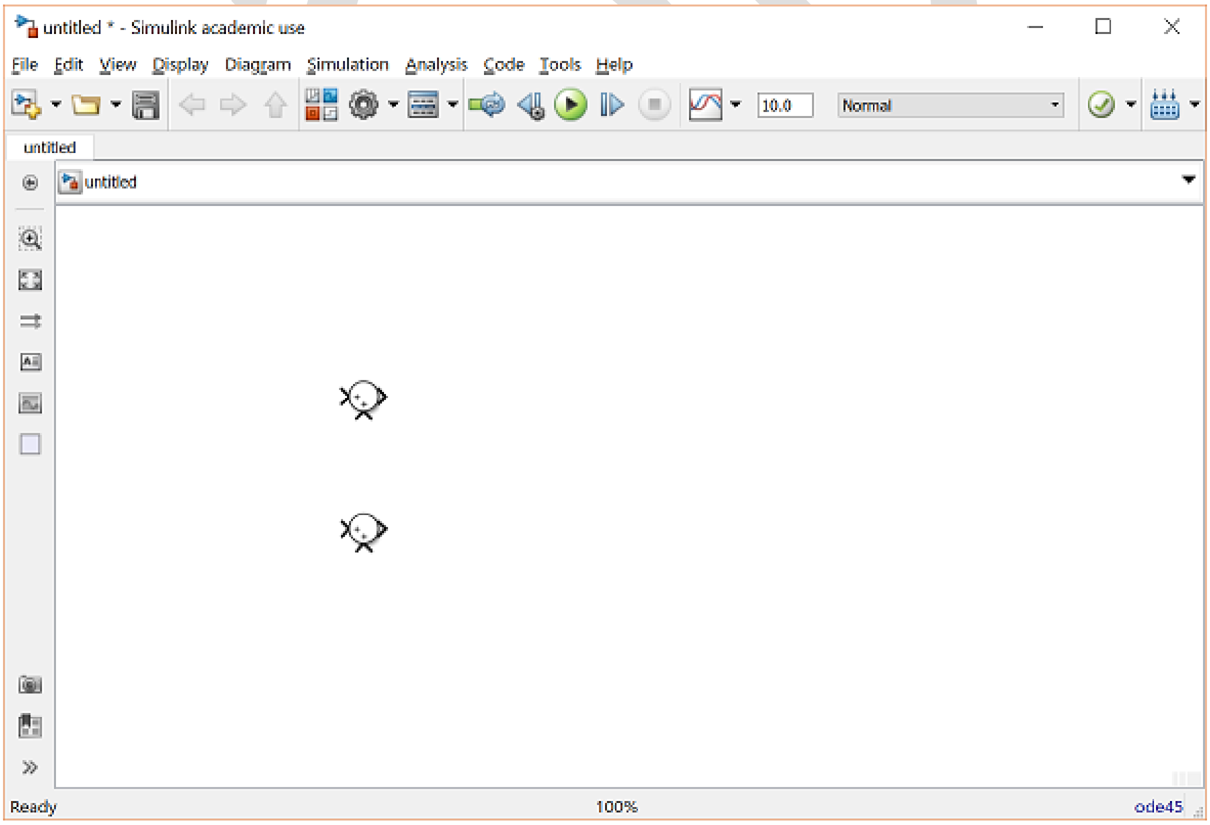
با اعمال قانون دوم نیوتن در راستای افقی و با توجه به دیاگرام‌های جسم آزاد رسم شده، می‌توان قوانین حاکم بر این سیستم را به دست آورد:

$$\Sigma F_1 = F - k(x_1 - x_2) - \mu M_1 g \dot{x}_1 = M_1 \ddot{x}_1 \qquad (۱)$$

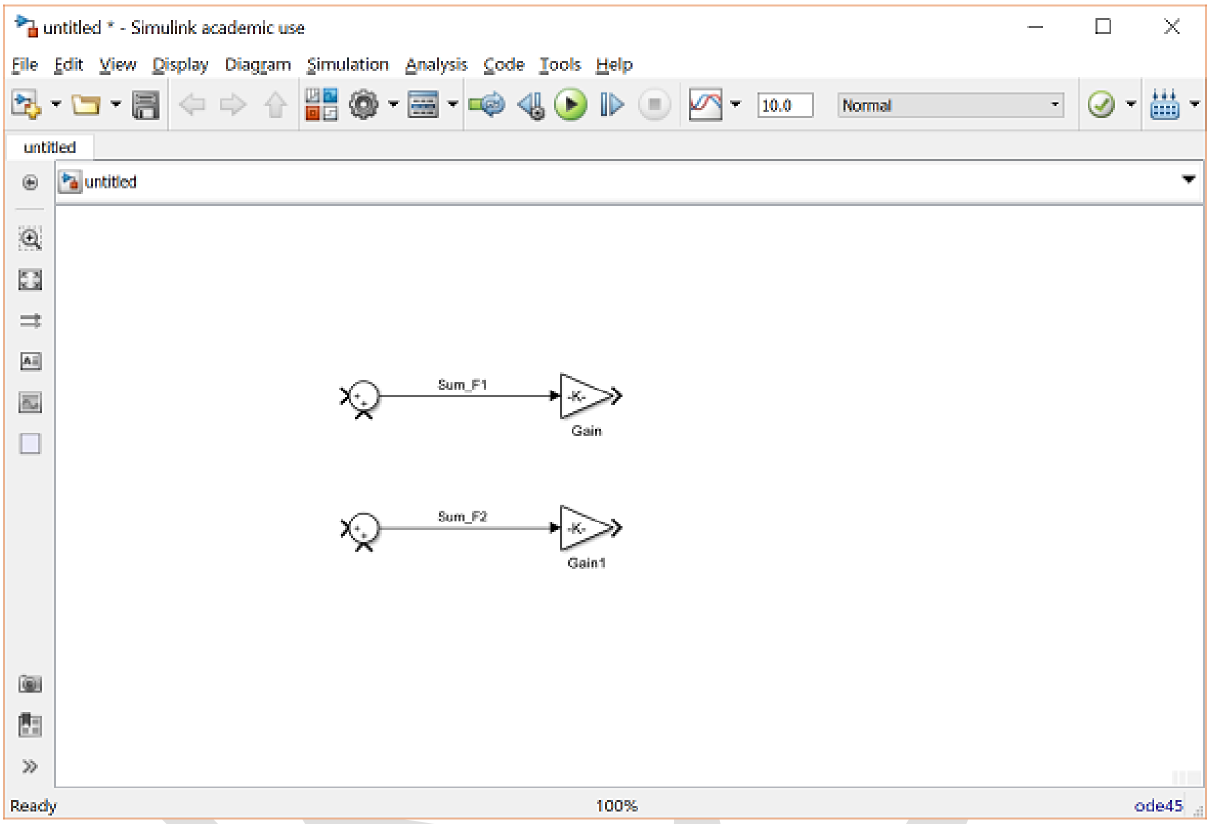
$$\Sigma F_2 = k(x_1 - x_2) - \mu M_2 g \dot{x}_2 = M_2 \ddot{x}_2 \qquad (۲)$$

ساخت مدل سیمولینک

می‌توان مجموعه‌ی قوانین حاکم بر سیستم را به صورت گرافیکی نمایش داد. در اینجا برای هر یک از جرم‌ها، معادله‌ی کلی $\Sigma F = ma$ یا $a = (\Sigma F)/m$ را تشکیل می‌دهیم. ابتدا سیمولینک را باز کرده و مدل جدیدی را ایجاد کنید. در قدم بعد دو بلوک Sum (از کتابخانه Math Operations) را به مدل خود اضافه کرده و آنها را تقریبا مانند شکل قرار دهید:

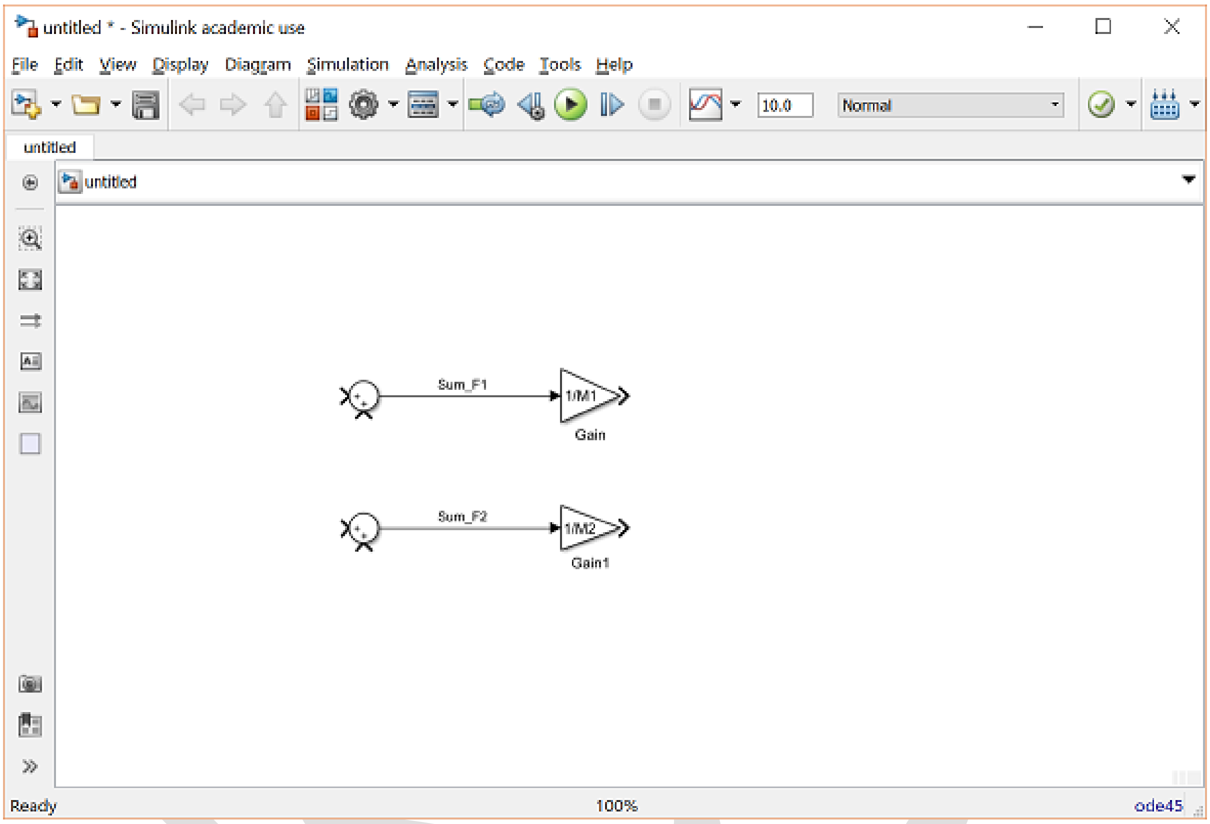


خروجی هریک از این بلوک‌های Sum بیانگر برآیند نیروهای وارد بر هر جرم می‌باشد. با ضرب این حاصلجمع در $\frac{1}{M}$، شتاب هر جرم به دست می‌آید. حال دو بلوک Gain (از کتابخانه Math Operations) را وارد مدل خود کرده و آنها را به خروجی بلوک‌های Sum متصل کنید. برای گویا بودن مدل ساخته شده، این دو سیگنال را با عناوین “Sum_F1” و “Sum_F2” نام‌گذاری کنید. برای نام‌گذاری بر روی خط هر سیگنال دابل کلیک کرده و نام دلخواه را تایپ کنید.

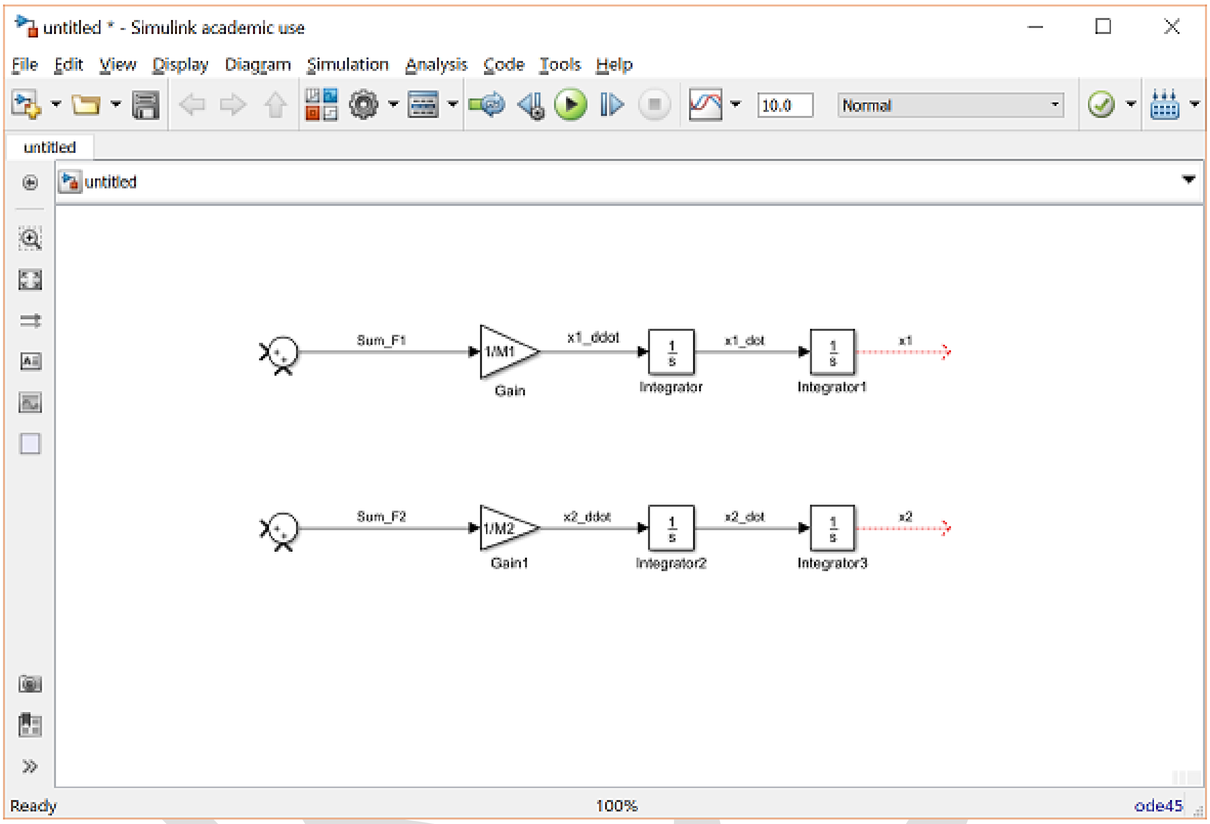


بلوک‌های Gain باید عبارت $1/M$ را داشته باشند. ما متغیرهای M1 و M2 را در فضای کار متلب تعریف کرده‌ایم پس تنها لازم است تا نام این متغیرها را در بلوک‌های بهره وارد نماییم. بر روی بلوک Gain بالایی دابل کلیک کرده و مقدار "1/M1" را در فیلد **Gain** وارد کنید. به طور مشابه مقدار "1/M2" در فیلد **Gain** بلوک Gain دوم وارد کنید.

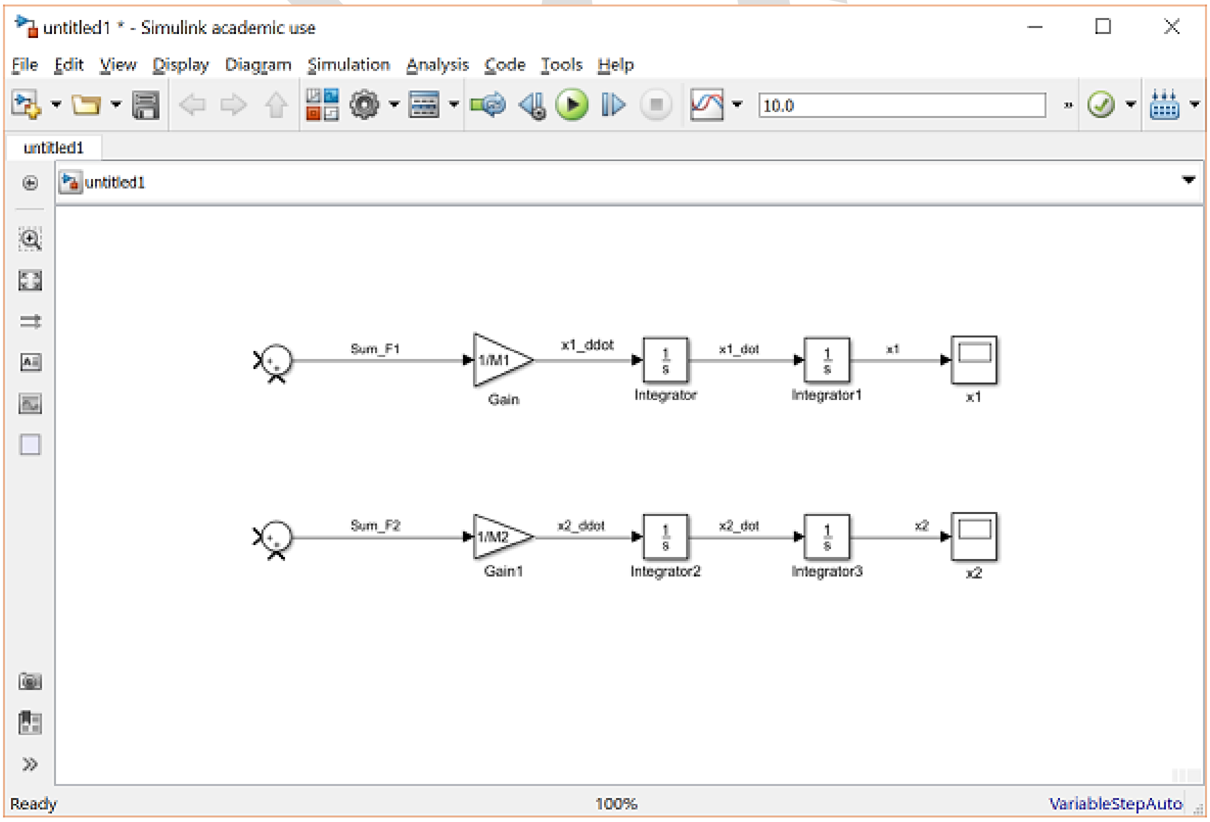
بعد از وارد کردن این مقادیر، همچنان بلوک‌ها مقدار -k- نمایش می‌دهند که به علت کوچک بودن ابعاد آنهاست، با تغییر ابعاد بلوک‌ها می‌توان مقادیر واقعی هر بهره را نمایش داد. برای تغییر اندازه بلوک، یکبار بر روی آن کلیک کرده تا مربع‌های کوچک در گوشه‌های بلوک نمایش داده شود، سپس با کشیدن هر یک از این مربع‌ها، می‌توان اندازه بلوک را تغییر داد. حال مدل ما به شکل زیر است:



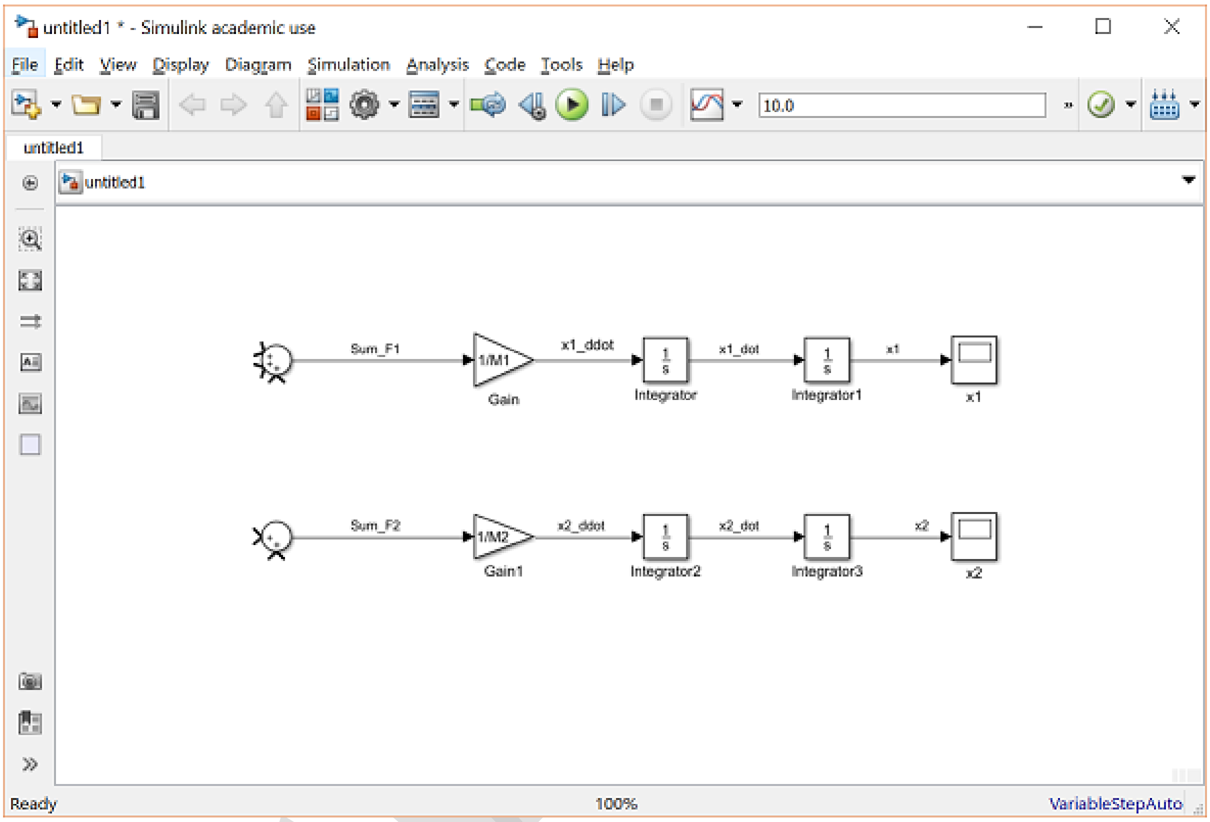
خروجی بلوک‌های Gain، شتاب هر یک از جرم‌ها (لوکوموتیو و واگن) می‌باشد. معادلات حاکم به دست آمده وابسته به سرعت و جابجایی جرم‌ها می‌باشد. از آنجایی که سرعت را می‌توان از انتگرال‌گیری از شتاب و جابجایی را از انتگرال‌گیری از سرعت به دست آورد، می‌توانیم برای به دست آوردن این مقادیر از بلوک‌های انتگرال‌گیر استفاده نماییم. در مجموع چهار بلوک Integrator نیاز است که می‌توان آنها را از کتابخانه Continuous به مدل خود اضافه نمود. بلوک‌ها را به شکل زیر قرار داده و نام‌گذاری کنید. انتگرال‌گیر اول، شتاب جرم ۱ به عنوان ورودی گرفته ("x1_ddot") و سرعت جرم ۱ ("x1_dot") را تولید می‌کند. انتگرال‌گیر دوم از سرعت انتگرال گرفته و جابجایی جرم ۱ ("x1") را به عنوان خروجی می‌دهد. این روش برای انتگرال‌گیرهای جرم دوم نیز استفاده می‌شود.



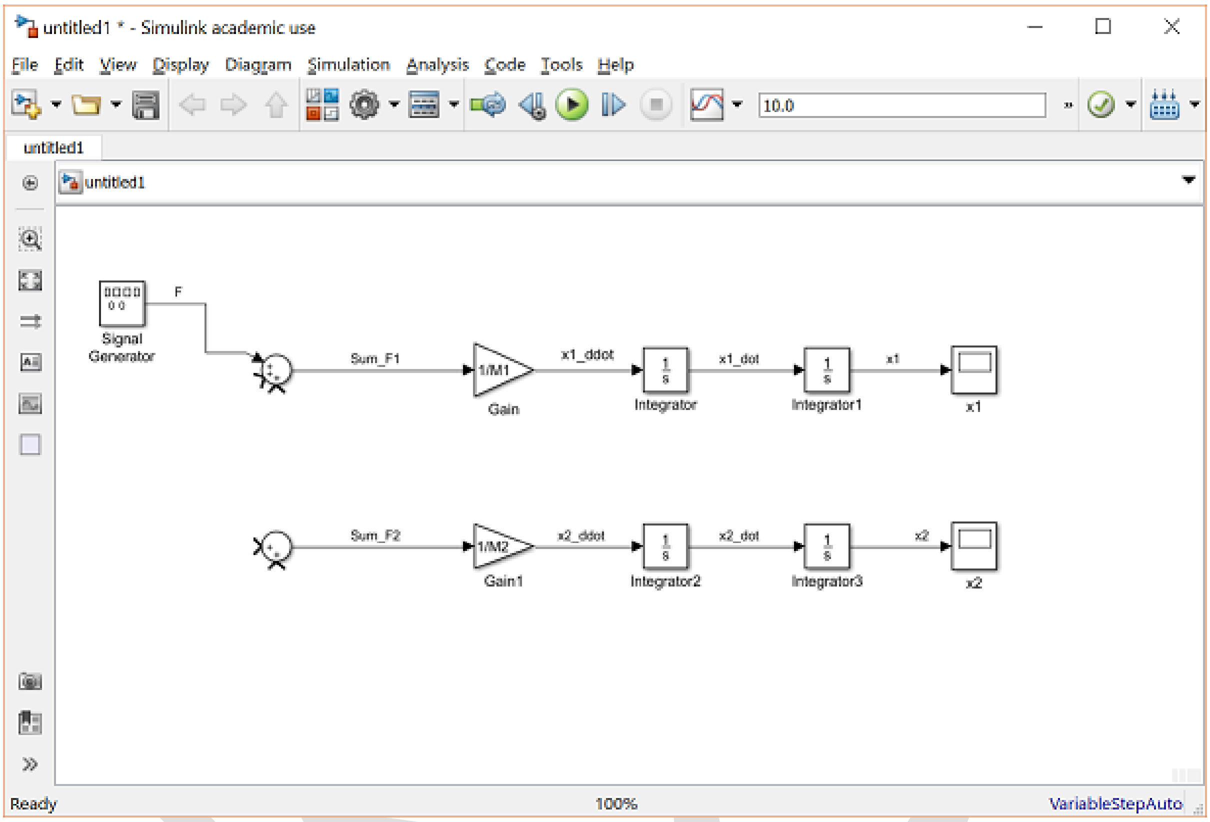
حال دو بلوک Scope را از کتابخانه Sinks به مدل خود اضافه کرده و آنها را به خروجی Integrator متصل کنید. این دو سیگنال را ”x1“ و ”x2“ نام‌گذاری کنید.



حال مدل ما برای اضافه کردن نیروهای وارد بر هر جرم آماده است. ابتدا لازم است تا ورودی به هر بلوک Sum را به گونه‌ای تنظیم کنیم تا بیانگر تعداد درستی از نیروها باشد (در مورد علامت این نیروها بعدا صحبت می‌کنیم). از آنجایی که در مجموع سه نیرو بر جرم ۱ اعمال می‌شوند، بر روی بلوک Sum دابل کلیک کرده و فیلد لیست علائم را به "++|+" تغییر دهید. کاراکتر "|" به عنوان فاصله استفاده می‌شود. به جرم ۲ تنها دو نیرو وارد می‌شود پس نیازی به تغییر بلوک Sum آن نمی‌باشد.



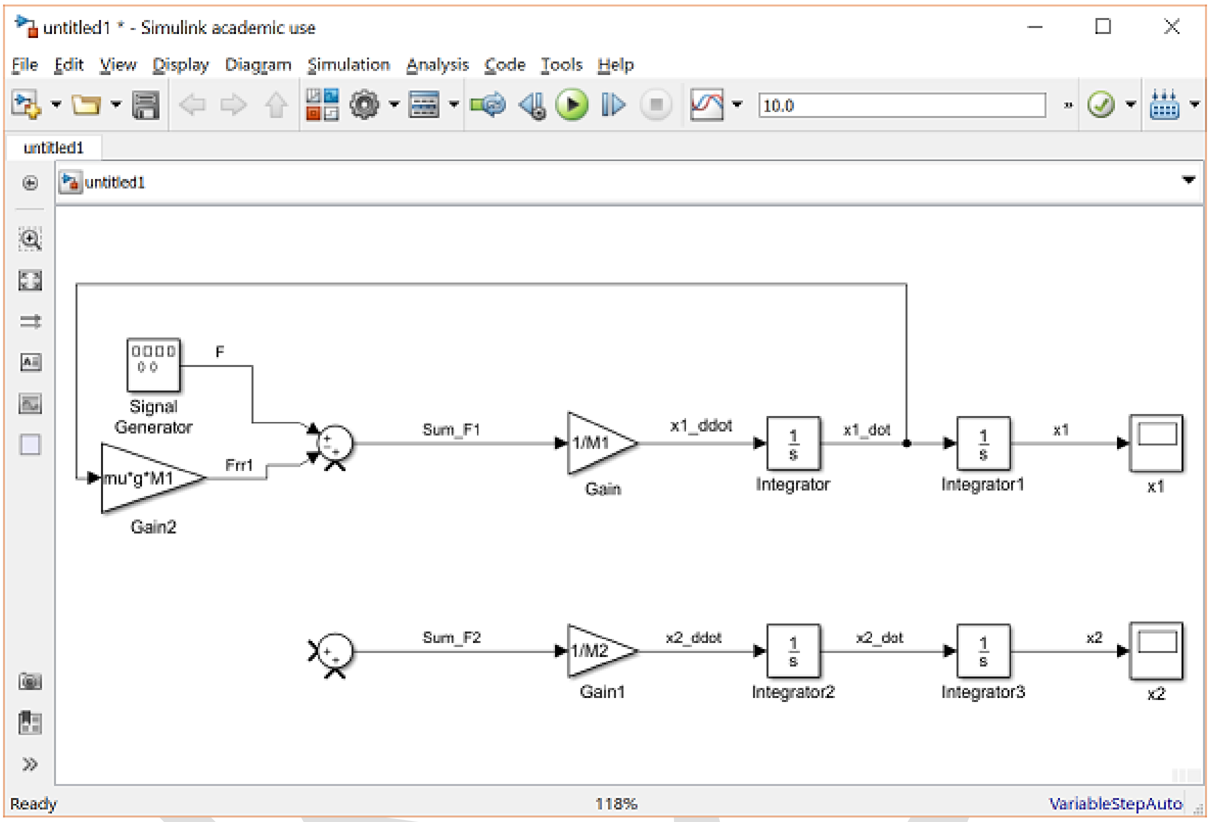
نیروی اولی که به جرم ۱ وارد می‌شود نیروی ورودی F می‌باشد. یک بلوک Signal Generator را از کتابخانه Sources انتخاب کرده و آنرا به ورودی بالای بلوک Sum متصل کنید. این سیگنال را "F" نام‌گذاری کنید.



نیروی بعدی وارد شده به جرم ۱، نیروی مقاومت غلتشی می‌باشد. برای یادآوری این نیرو به شکل زیر مدل‌سازی شده است:

$$F_{rr,1} = \mu g M_1 \dot{x}_1 \qquad (۳)$$

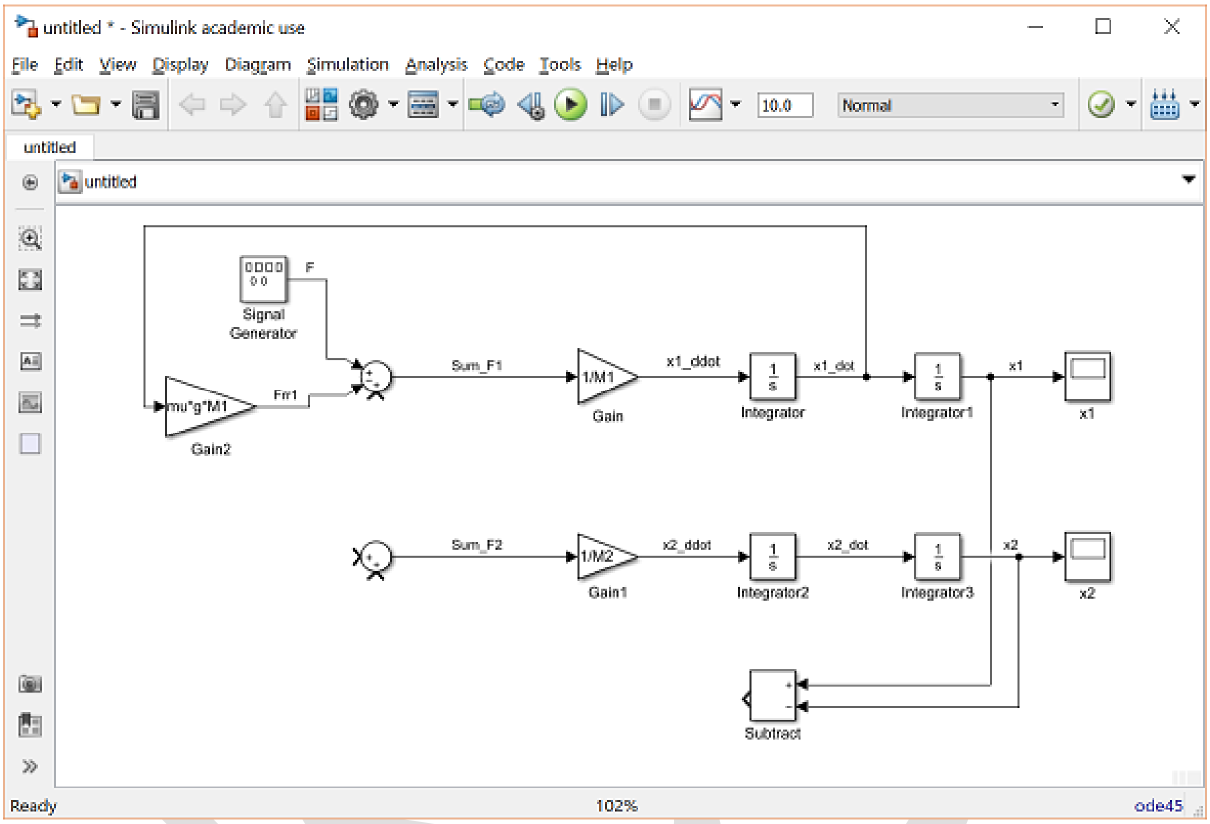
برای تولید این نیرو، می‌توانیم سیگنال سرعت را گرفته و آنرا در بهره‌ی مناسب ضرب نماییم. یک بلوک Gain به مدل اضافه می‌نماییم. یک انشعاب از سیگنال ”x1_dot“ گرفته و آنرا به بلوک بهره‌ای که اضافه نمودیم وصل کنید. خروجی بلوک Gain را به ورودی دوم بلوک Sum متصل کنید. بر روی بلوک Gain دابل کلیک کرده و مقدار ”mu*g*M1“ را در فیلد **Gain** وارد کنید. نیروی مقاومت غلتشی در جهت عکس حرکت اعمال می‌شود بنابراین از لیست علائم بلوک Sum، فیلد لیست علائم را به ”+-|“ تغییر می‌دهیم. سپس اندازه بلوک Gain را تغییر داده تا مقدار بهره نمایش داده شده و خروجی آنرا با عنوان ”Frr1“ نام‌گذاری کنید. حال مدل ما به شکل زیر است:



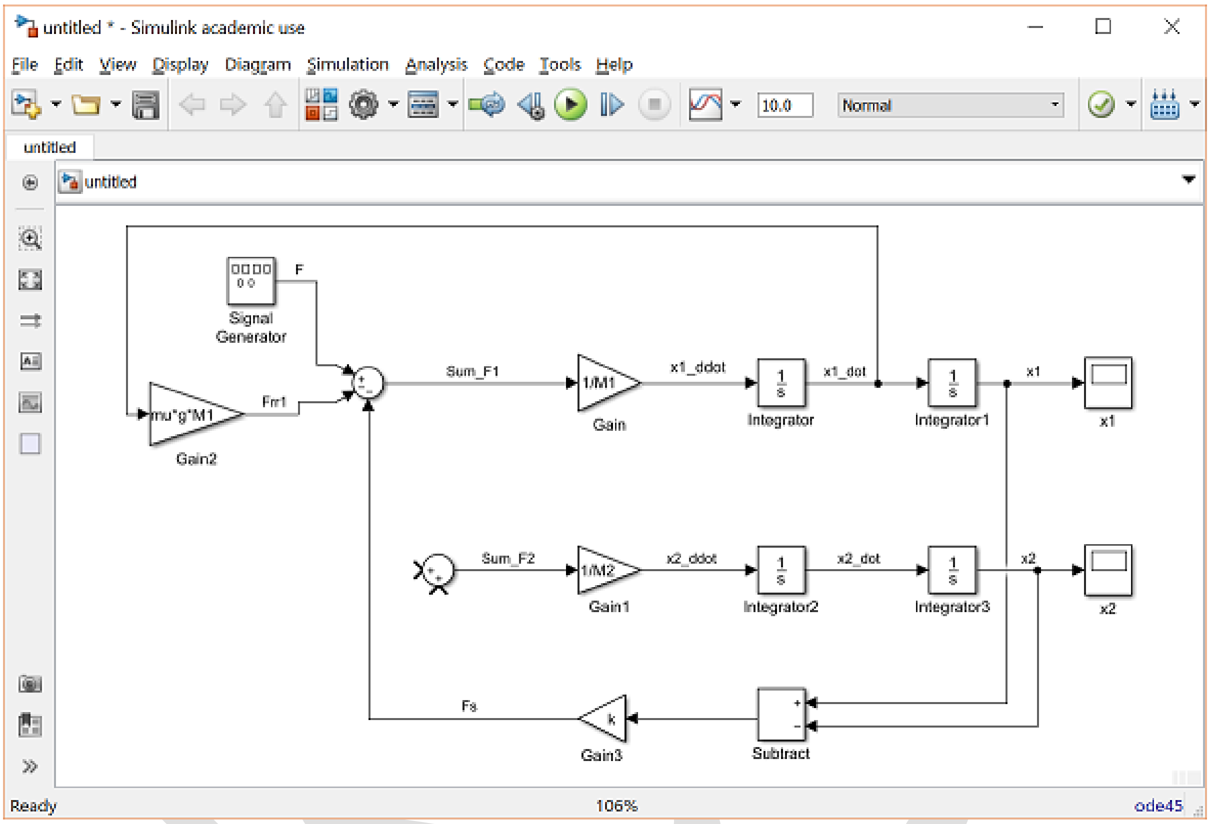
نیروی آخر وارد شده به جرم ۱، نیروی فنر می‌باشد. این نیرو را به شکل زیر مدل‌سازی نمودیم:

$$F_s = k(x_1 - x_2) \tag{۴}$$

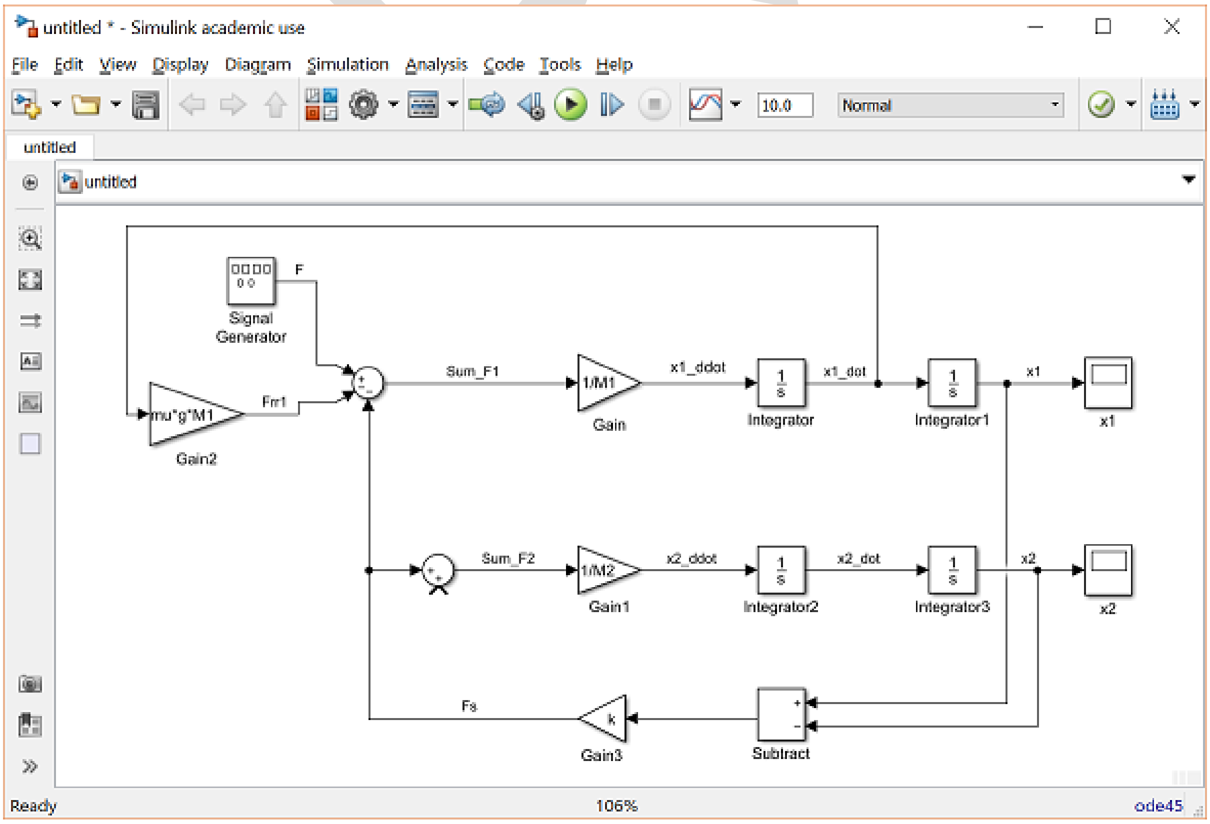
پس لازم است تا سیگنال $(x_1 - x_2)$ را ایجاد کنیم و در بهره k (ثابت فنر) ضرب نماییم. بلوک Subtract (یا بلوک Sum) را به مدل ساخته شده، اضافه می‌نماییم. برای تغییر جهت این بلوک بر روی آن راست کلیک کرده و گزینه **Rotate & Flip > Flip Block** را انتخاب کنید. همچنین می‌توانید برای این کار، بلوک را انتخاب کرده و کلیدهای ترکیبی Ctrl+I را فشار دهید. یک انشعاب از سیگنال "x2" گرفته و آنرا به ورودی منفی بلوک Subtract متصل کنید. همچنین یک انشعاب از سیگنال "x1" گرفته و آن را به ورودی مثبت متصل کنید. ممکن است با اینکار خطوط اتصال بین بلوک‌ها با یکدیگر تقاطع داشته باشند اما در واقع آنها با هم تماسی ندارند. تنها وقتی که در محل تقاطع خطوط یک نقطه‌ی کوچک ایجاد شود اتصال بین خطوط وجود دارد.



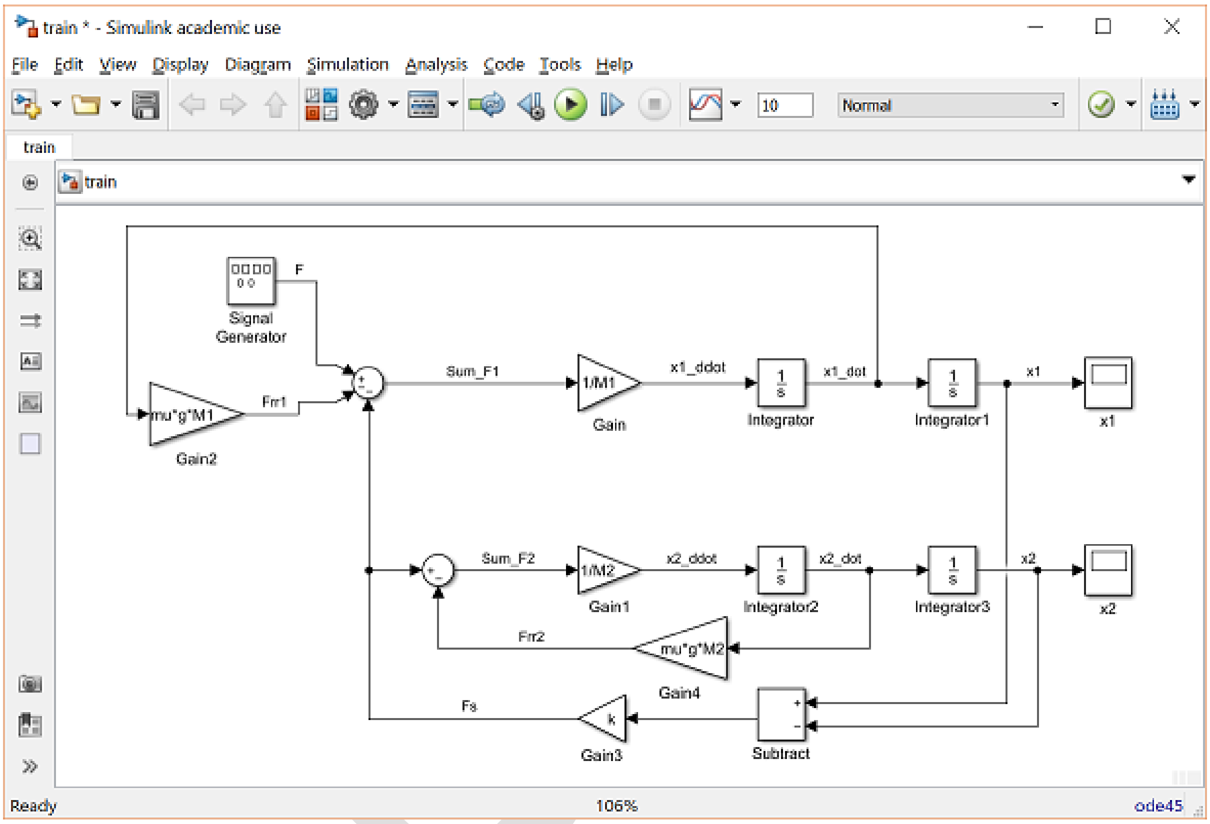
حال با ضرب این تفاضل در ثابت فنر، می‌توان نیروی فنر را به دست آورد. یک بلوک Gain را به مدل خود و در سمت چپ بلوک تفاضل اضافه کنید. مقدار بلوک بهره را مقدار "k" که همان ثابت فنر است تعریف کرده و خروجی بلوک تفاضل را به ورودی آن متصل کنید. سپس خروجی بلوک Gain را به سومین ورودی بلوک Sum برای جرم ۱ متصل کرده و آنرا "Fs" نامگذاری نمایید. به علت اینکه نیروی فنر در خلاف جهت حرکت جرم ۱ وارد می‌شود، لازم است تا دوباره لیست علائم را در بلوک Sum به "--|+" تغییر دهیم. حال مدل ما به شکل زیر است:



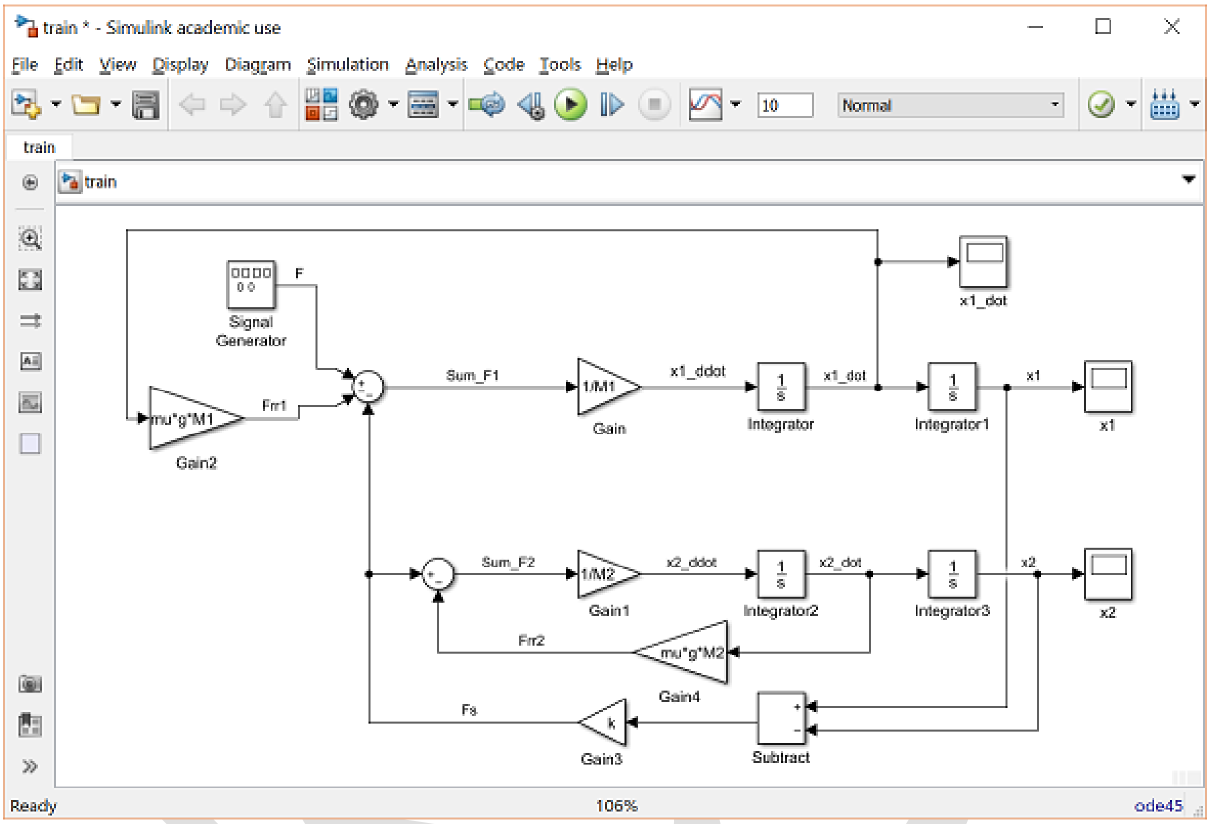
نوبت به اعمال نیرو به جرم ۲ می‌باشد. برای نیروی اول از همان نیروی فنری که برای جرم ۱ ایجاد کردیم استفاده کرده و تنها تفاوت آن علامت مثبت آن برای جرم ۲ می‌باشد. از سیگنال ”Fs“ یک انشعاب گرفته و به ورودی اول بلوک Sum جرم ۲ وصل کنید.



آخرین نیروی اعمالی نیروی مقاومت غلتشی جرم ۲ می‌باشد. این نیرو به روش مشابه نیروی مقاومت غلتشی جرم ۱ به دست می‌آید. از سیگنال "x2_dot" یک انشعاب گرفته و آنرا با استفاده از بلوک Gain در مقدار "mu*g*M2" ضرب می‌نماییم. خروجی بلوک Gain را به ورودی دوم بلوک Sum جرم ۲ داده و آن سیگنال را "Frr2" نامگذاری می‌کنیم. با تغییر دادن علامت ورودی دوم در بلوک Sum به منفی، مدل ما به شکل زیر درآمده است.



مدل ما به طور کامل ساخته شده است و تنها لازم است تا ورودی مناسب را ایجاد کرده و خروجی مورد نظر را استخراج کنیم. ورودی سیستم نیروی F تولید شده توسط لوکوموتیو است. در مدل سیمولینک نیروی F را از خروجی بلوک تولید سیگنال تعریف کرده‌ایم. خروجی سیستم که باید مشاهده و کنترل شود، سرعت لوکوموتیو قطار می‌باشد. یک Scope دیگر را به مدل اضافه کرده و انشعابی از سیگنال "x1_dot" به آن متصل کنید. این سیگنال را "x1_dot" نامگذاری کنید. حال مدل ما به شکل زیر درآمده است.



مدل ما آماده شده است و باید آنرا ذخیره نمود.

اجرای مدل

قبل از اجرای مدل، لازم است تا مقادیر عددی را به متغیرهای تعریف شده در مدل اختصاص دهیم. برای این سیستم، مقادیر زیر را انتخاب می‌نماییم:

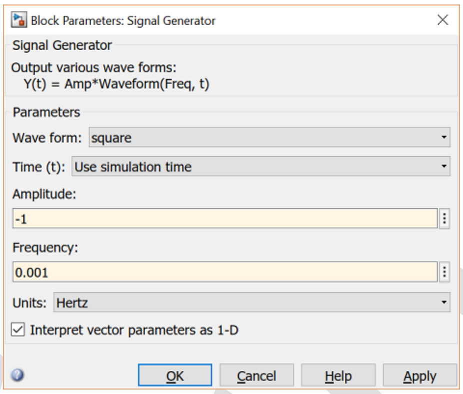
- M_1 = 1 kg
- M_2 = 0.5 kg
- k = 1 N/sec
- F = 1 N
- μ = 0.02 sec/m
- g = 9.8 m/s^2

یک ام‌فایل ساخته و دستورات زیر را در آن وارد کنید:

```
M1 = 1;

M2 = 0.5;

k  = 1;

F  = 1;

mu = 0.02;

g  = 9.8;
```

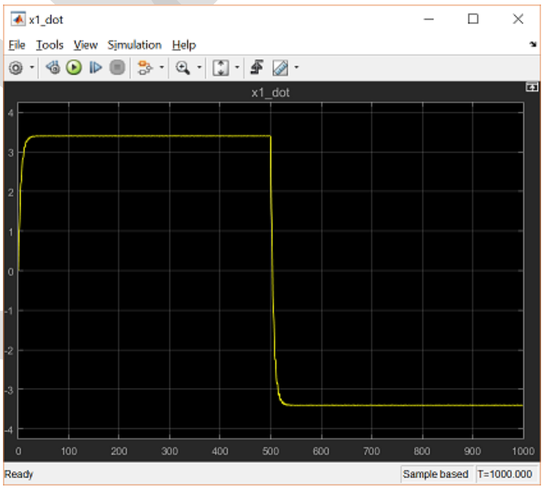
این ام‌فایل را در متلب اجرا کرده تا متغیرها تعریف شوند. با این کار سیمولینک به مقادیر این متغیرها دسترسی خواهد داشت.

اکنون باید ورودی مناسبی را برای سیستم تولید کنیم. بر روی بلوک Signal Generator دابل کلیک کرده و از منوی کشویی Wave form مورد square را انتخاب نموده و مقدار Frequency را برابر "0.001" قرار دهید. واحدها را بر مبنای Hertz قرار دهید. همچنین برای فیلد Amplitude مقدار "-1" را وارد نمایید (مقادیر مثبت دامنه باعث می‌شود ابتدا سیگنال پله منفی شود و سپس مثبت شود).



آخرین قدم قبل از اجرای شبیه‌سازی، انتخاب زمان شبیه‌سازی مناسب است. برای مشاهده‌ی یک سیکل کامل سیگنال مربعی با فرکانس ۰/۰۰۱ هرتز لازم است تا مدل ما برای ۱۰۰۰ ثانیه شبیه‌سازی گردد. در بالای پنجره و از منوی Simulation بر روی Model Configuration Parameters کلیک کرده و مقدار Stop Time را برابر "1000" قرار دهید. پنجره‌ی باز شده را ببندید.

حال شبیه‌سازی را اجرا کرده و Scope "x1_dot" را باز کرده تا خروجی سرعت را بررسی کنیم. ورودی به سیستم یک موج مربعی با دو پله بوده است که پله اول مثبت و پله دوم منفی می‌باشد. به طور فیزیکی به این معناست که ابتدا لوکوموتیو به سمت جلو حرکت کرده و سپس به سمت عقب حرکت می‌کند. خروجی سرعت در زیر رسم شده است.



در این فصل به استخراج مدل ریاضی سیستم قطار پرداخته و معادلات به دست آمده را در سیمولینک پیاده‌سازی نمودیم. روش دیگر برای اینکار استفاده از ابزار مدلسازی فیزیکی Simscape می‌باشد. Simscape به کاربر اجازه می‌دهد تا یک سیستم را با استفاده از بلوک‌هایی که بیانگر المان‌ها و اجزای فیزیکی هستند (مانند اینرسی و مفاصل یا مقاومت و سلف) مدلسازی نماید. با استفاده از Simscape می‌توان یک سیستم فیزیکی را بدون استخراج معادلات ریاضی حاکم بر آن، شبیه‌سازی نمود.

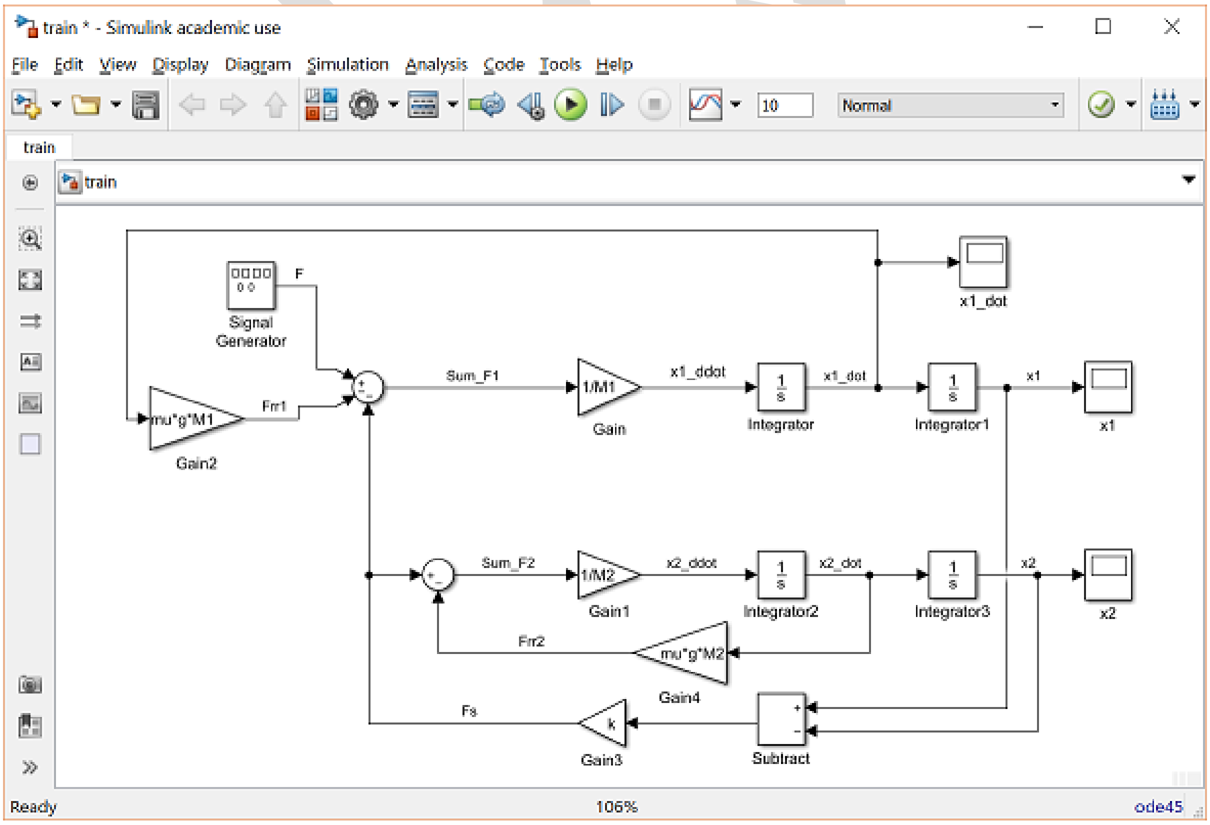
بخش نهم: مقدمه‌ای بر طراحی کنترلر در سیمولینک

فهرست مطالب بخش

مدل حلقه باز سیستم

در فصل پیش دیدیم که چطور می‌توان یک سیستم فیزیکی را در سیمولینک شبیه‌سازی نمود. به طور کلی سیمولینک می‌تواند کل سیستم کنترل یعنی علاوه بر سیستم فیزیکی، الگوریتم کنترل را نیز شبیه‌سازی نماید. همانطور که ذکر شد سیمولینک به طور ویژه برای به دست آوردن جواب‌های تقریبی مدل‌های ریاضی که به سختی می‌توان آنها را با محاسبات دستی به دست آوردن مفید است. برای مثال یک سیستم غیر خطی را در نظر بگیرید. یکی از روش‌های کنترل این سیستم، خطی سازی سیستم و سپس استفاده از مدل خطی‌سازی شده برای طراحی یک کنترلر به روش تحلیلی می‌باشد. سپس می‌توان از سیمولینک برای پیاده‌سازی این کنترلر بر روی مدل غیر خطی استفاده نمود. از سیمولینک می‌توان برای تولید سیستم خطی‌سازی شده و از متلب برای طراحی کنترلر که قبلا توضیح داده شد استفاده کرد. بسیاری از امکانات طراحی کنترلر موجود در متلب نیز به طور مستقیم از سیمولینک قابل دسترسی می‌باشند. در این فصل هر دو روش گفته شده را توضیح می‌دهیم.

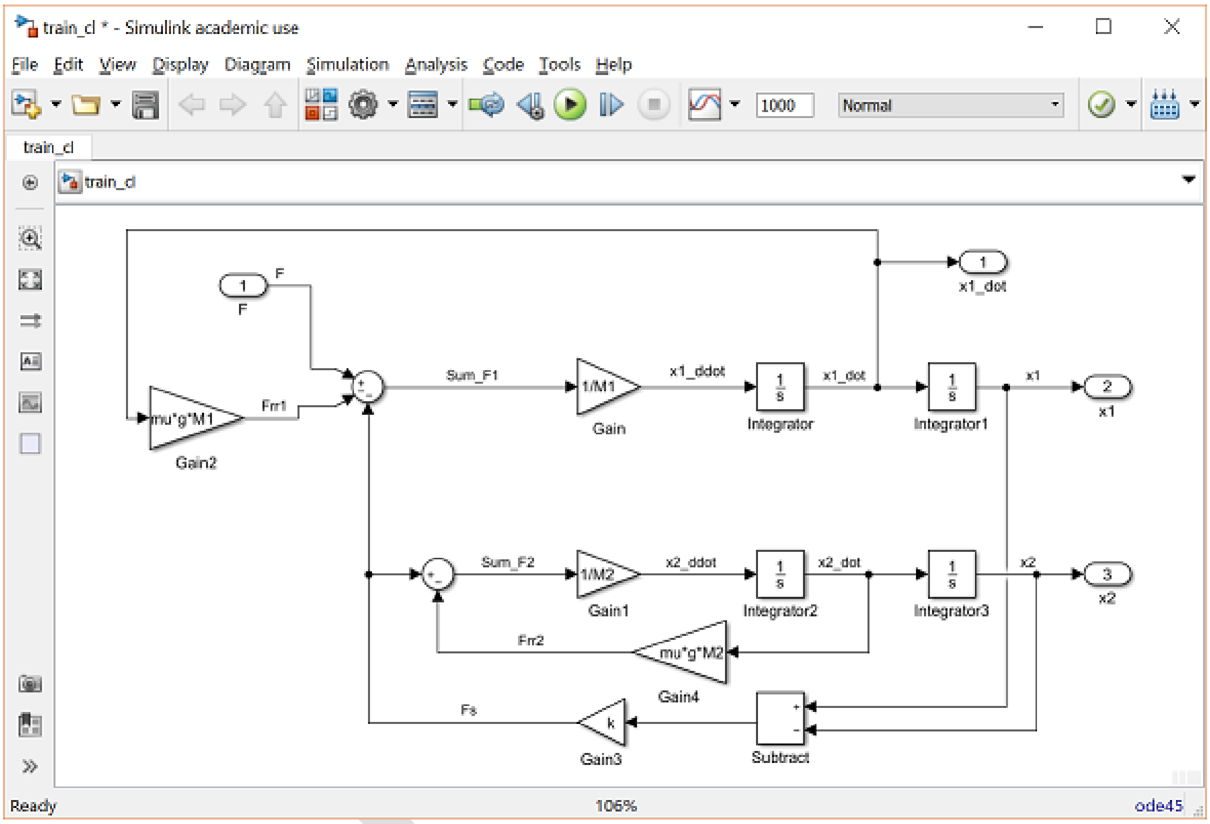
مثال سیستم قطار اسباب بازی گفته شده در فصل قبل را در نظر می‌گیریم.



با فرض اینکه قطار تنها در یک بعد حرکت می‌کند، می‌خواهیم کنترلی را به سیستم اعمال کنیم تا سیستم به آرامی شروع به حرکت کرده و به نرمی متوقف شود و همچنین بتواند سرعت ثابت را با حداقل خطای ماندگار دنبال کند.

پیاده‌سازی کنترلر PID در سیمولینک

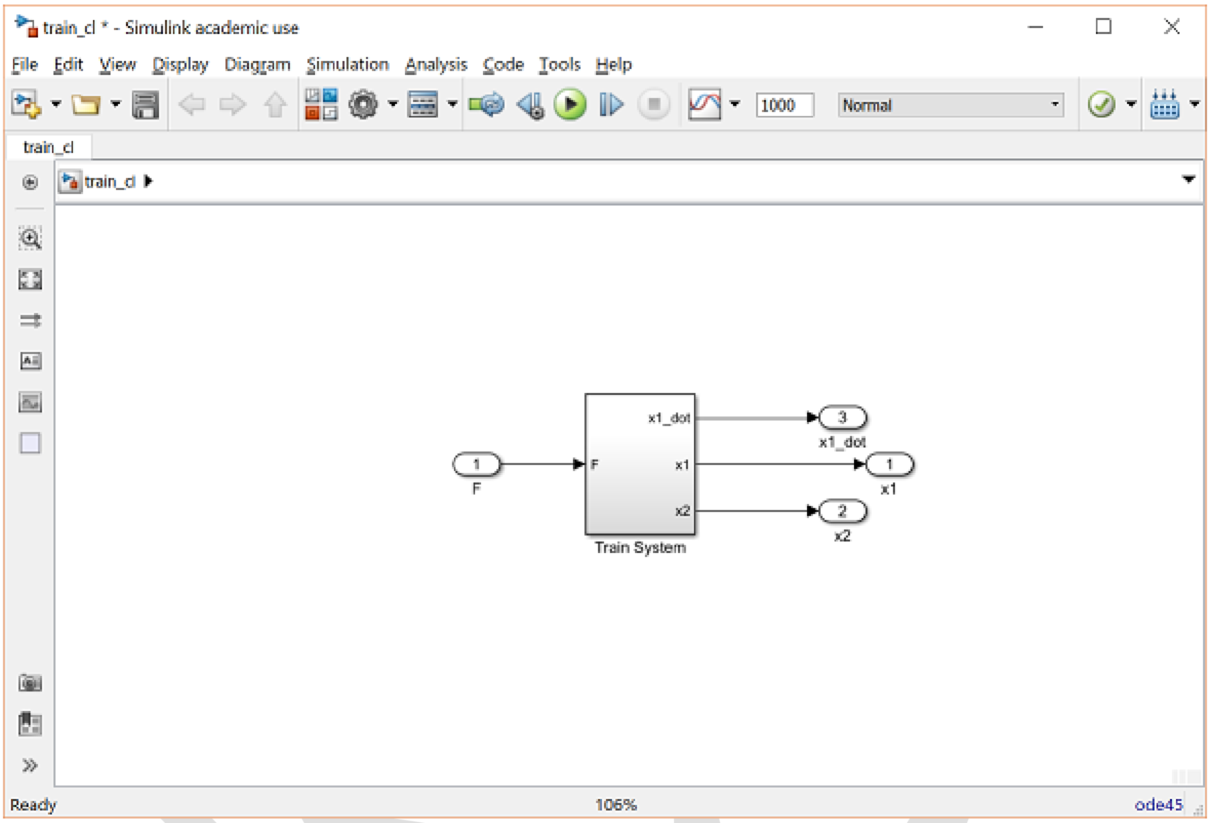
ابتدا ساختار سیستم قطار با فیدبک واحد و کنترلر PID را ایجاد می‌کنیم. برای اینکه مدل سیمولینک واضح باشد مدل قطار را در یک بلوک زیرسیستم[۲۵] ذخیره می‌نماییم. برای این کار سه Scope قرار داده شده را حذف نموده و هر سه سیگنال را به سه بلوک Out از کتابخانه Sinks متصل می‌کنیم. هر بلوک Out را با نام مناسب آن متغیر نامگذاری نمایید. همچنین بلوک Signal Generator را حذف کرده و به جای آن یک بلوک In از کتابخانه Sources قرار دهید. این ورودی را "F" نامیده و بیانگر نیروی ایجاد شده بین لوکوموتیو و ریل می‌باشد. حال مدل ما به شکل زیر است:



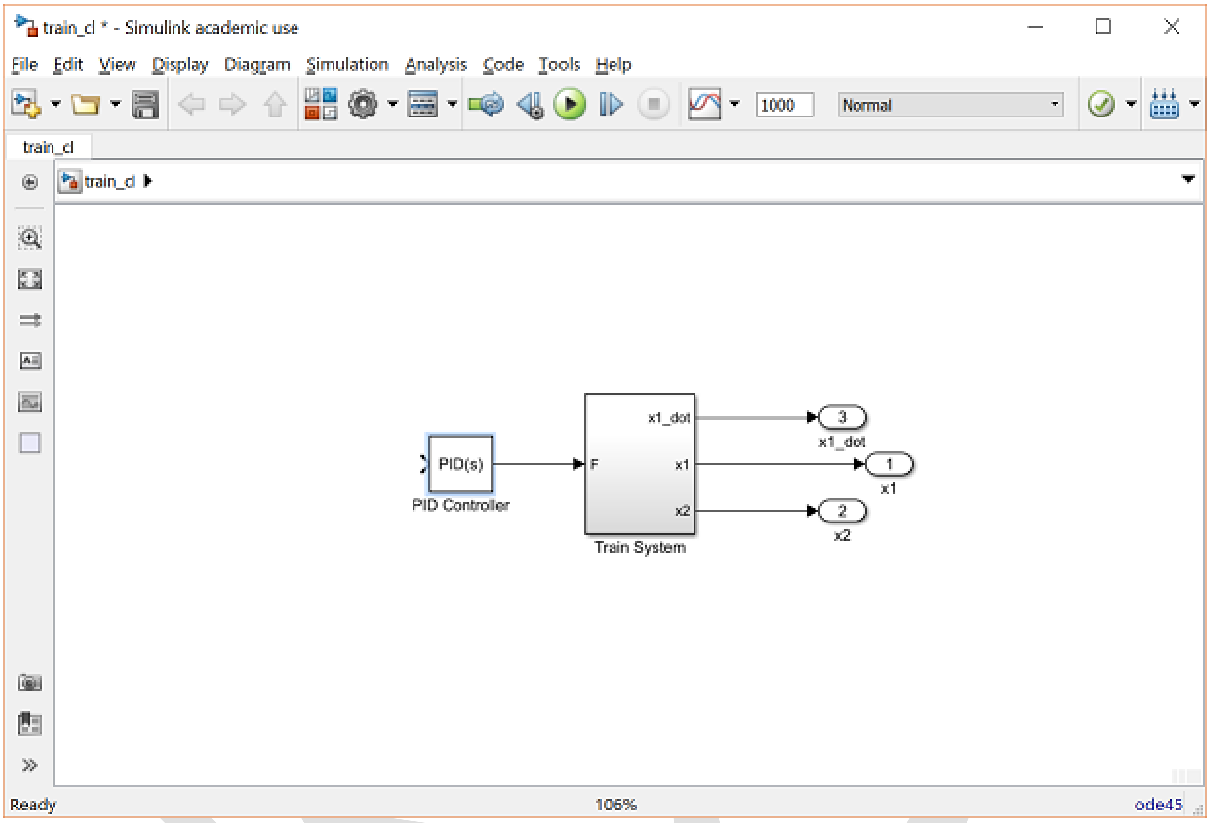
در قدم بعد تمامی بلوک‌ها را انتخاب کرده (با کلید ترکیبی Ctrl+A) و با راست کلیک بر روی این صفحه، گزینه Create Subsystem From Selection را انتخاب کنید. با کمی تغییر چیدمان و نامگذاری، مدل ما به شکل زیر درآمده است:

---

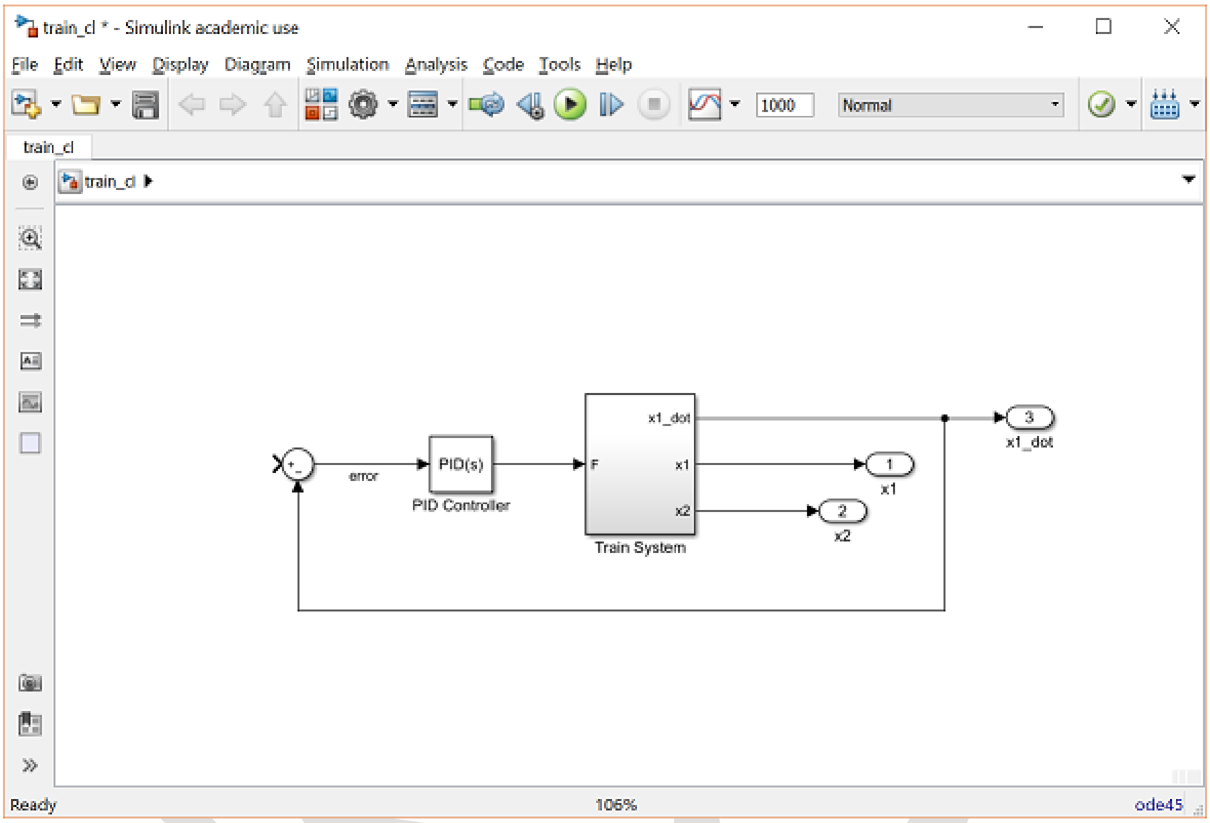
۲۵ Subsystem



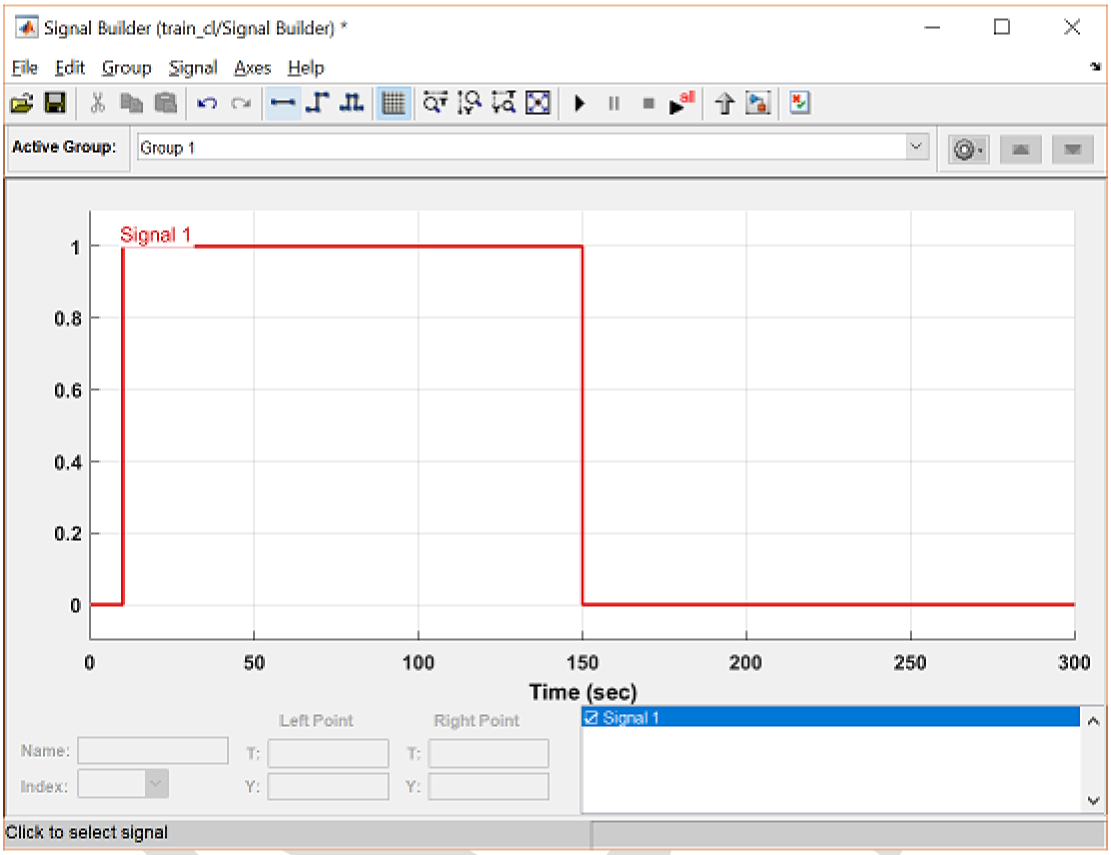
حال می‌توان کنترلر را به سیستم اضافه نمود. ما از کنترلر PID که توسط بلوک کنترلر PID در کتابخانه Continuous پیاده‌سازی می‌شود استفاده می‌کنیم. با سری کردن این بلوک با زیرسیستم قطار، مدل به شکل زیر در می‌آید. در ادامه ما کنترلر را به گونه‌ای طراحی می‌کنیم تا مستقیما نیروی F را تولید کند، با اینکار از دینامیک تولید گشتاور توسط موتور و همچنین دینامیک تولید نیرو توسط تماس چرخ و ریل صرف نظر کرده‌ایم. این ساده‌سازی از این جهت صورت گرفته است که قصد ما در اینجا تنها معرفی کاربردهای اساسی سیمولینک برای طراحی کنترلر و تحلیل می‌باشد.



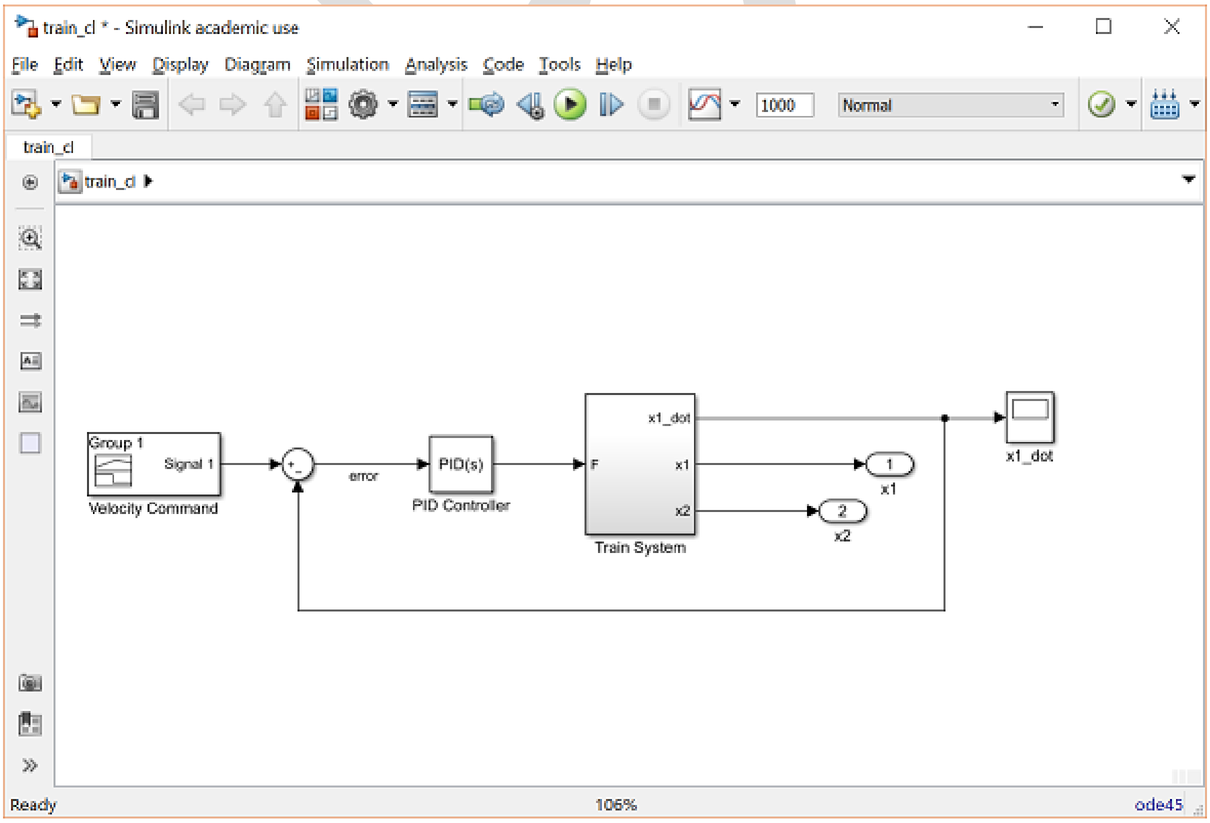
با راست کلیک بر روی بلوک کنترلر PID، ابتدا بهره‌ی انتگرالی را برابر ۰ و بهره‌های تناسبی و مشتقی را مانند پیش‌فرض به ترتیب برابر ۱ و ۰ قرار می‌دهیم. سپس یک بلوک Sum از کتابخانه عملیات ریاضی به مدل اضافه می‌کنیم. بر روی این بلوک راست کلیک کرده و لیست علائم را به "-+|" تغییر می‌دهیم. به علت اینکه هدف ما کنترل سرعت قطار می‌باشد، از سیگنال سرعت لوکوموتیو فیدبک می‌گیریم. برای اینکار یک انشعاب از سیگنال "x1_dot" گرفته و آنرا به ورودی با علامت منفی بلوک Sum وصل می‌کنیم. خروجی بلوک Sum باید برابر با خطای سرعت قطار بوده و خروجی آن به ورودی کنترلر متصل شود. با اتصال بلوک‌های گفته شده و نام‌گذاری سیگنال‌ها، مدل ما به شکل زیر می‌باشد.



در قدم بعد از بلوک Signal Builder در کتابخانه Sources به عنوان سرعت مطلوب قطار استفاده می‌نماییم. چون می‌خواهیم کنترلری طراحی کنیم تا قطار را به آرامی به حرکت درآورده و سپس به نرمی متوقف کند، سرعت مطلوب را ابتدا به صورت یک پله به ارتفاع 1 m/s و سپس یک پله برای بازگشت به مقدار 0 m/s در نظر می‌گیریم. برای تولید این سیگنال، بر روی بلوک Signal Builder دابل کلیک کرده و از منوی Axes در بالای صفحه گزینه Change time range را انتخاب می‌کنیم. فیلد Max time را برابر "300" ثانیه قرار می‌دهیم. سپس پله را طوری طراحی می‌کنیم تا در زمان ۱۰ ثانیه اتفاق افتاده و در زمان ۱۵۰ ثانیه به صفر بازگردد. برای اینکار بر روی قسمت‌های مختلف نمودار کلیک نموده یا خط سیگنال را به موقعیت مناسب کشیده یا زمان مورد نظر را در فیلد **T** در پایین صفحه وارد می‌کنیم. در نهایت سیگنال ما به شکل زیر است:



همچنین یک بلوک Scope از کتابخانه Sinks به مدل اضافه نموده و به جای بلوک خروجی سرعت قطار قرار دهید. با نام گذاری مجدد بلوک‌ها، مدل ما به شکل زیر است.



اکنون مدل حلقه بسته‌ی ما برای شبیه‌سازی آماده است.

اجرای مدل حلقه بسته

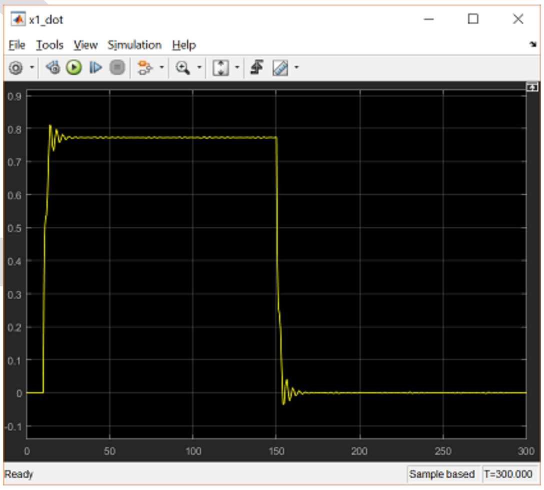
قبل از اجرای مدل، باید مقادیر عددی را به متغیرها اختصاص دهیم. برای سیستم قطار از مقادیر زیر استفاده می‌کنیم:

- M_1 = 1 kg
- M_2 = 0.5 kg
- k = 1 N/sec
- F = 1 N
- μ = 0.02 sec/m
- g = 9.8 m/s^2

در یک ام‌فایل دستورات زیر را وارد کنید:

```
M1 = 1;

M2 = 0.5;

k  = 1;

F  = 1;

mu = 0.02;

g  = 9.8;
```

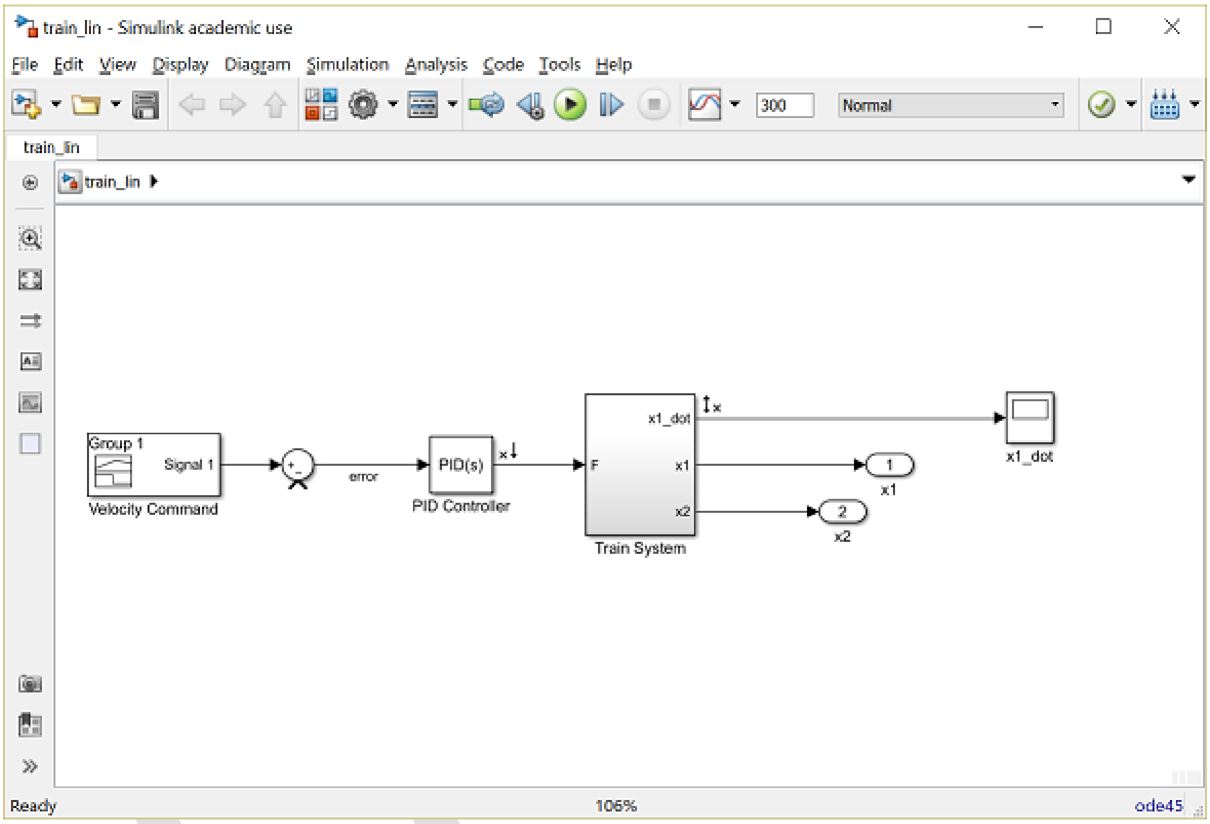
با اجرای این ام‌فایل در محیط متلب، این مقادیر تعریف می‌شوند. سیمولینک از مقادیر این متغیرها در مدل خود استفاده می‌کند. در قدم بعد باید زمان شبیه‌سازی را تنظیم کنیم تا سازگار با گستره‌ی زمانی تعریف شده در بلوک Signal Builder باشد. برای اینکار از منوی Simulation گزینه Model Configuration Parameters را انتخاب کرده و در بالای پنجره‌ی باز شده، مقدار "۳۰۰" را در فیلد Stop Time وارد کنید. حال شبیه‌سازی را اجرا کرده و Scope "x1_dot" را برای نمایش خروجی سرعت باز کنید. نتیجه‌ی به دست آمده که در پایین آمده است، نشان می‌دهد که سیستم حلقه بسته با این کنترلر، پایدار می‌باشد.



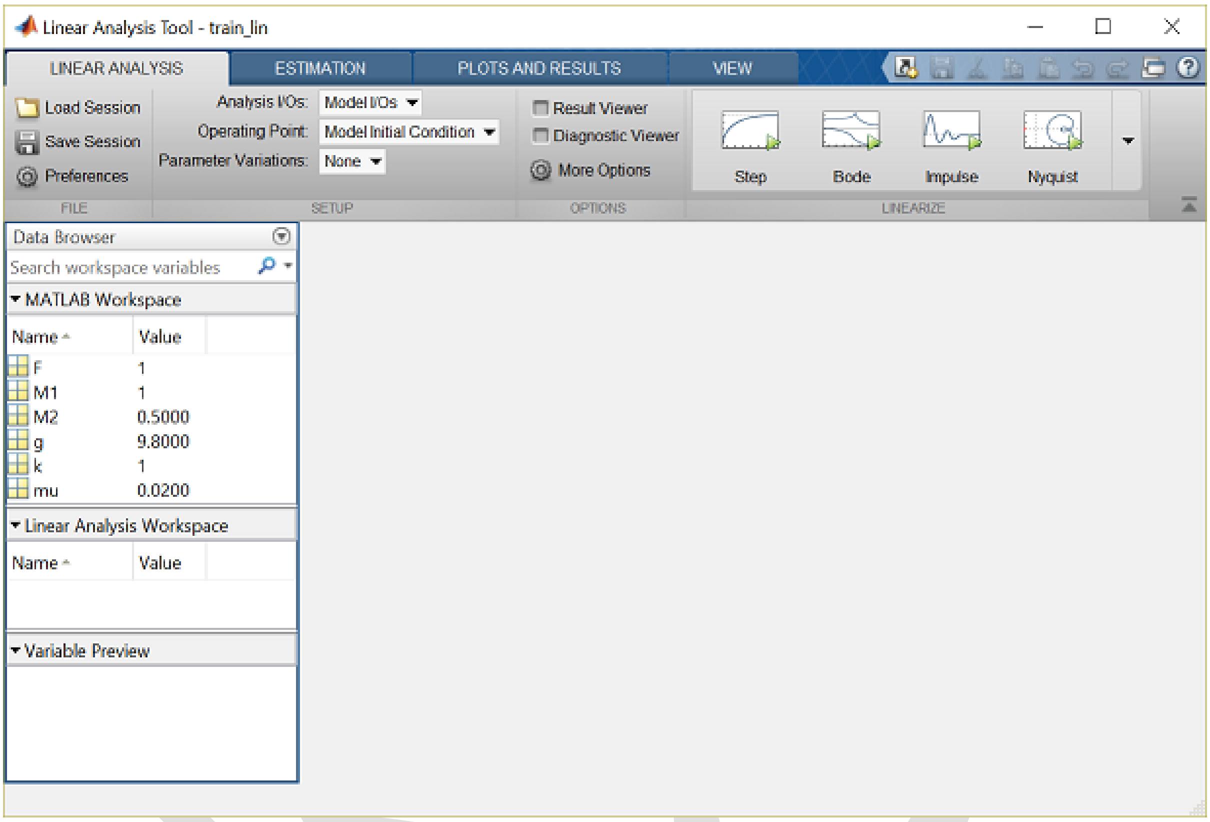
به دلیل اینکه عملکرد سیستم حلقه بسته رضایت بخش نمی‌باشد (خطای حالت ماندگار زیاد)، در ادامه به طراحی دوباره‌ی کنترلر این سیستم می‌پردازیم. ابتدا نشان می‌دهیم تا چگونه مدل را از سیمولینک به متلب، جهت تحلیل و طراحی، منتقل کنیم سپس چگونگی طراحی به طور مستقیم در سیمولینک را نشان می‌دهیم.

استخراج مدل به متلب

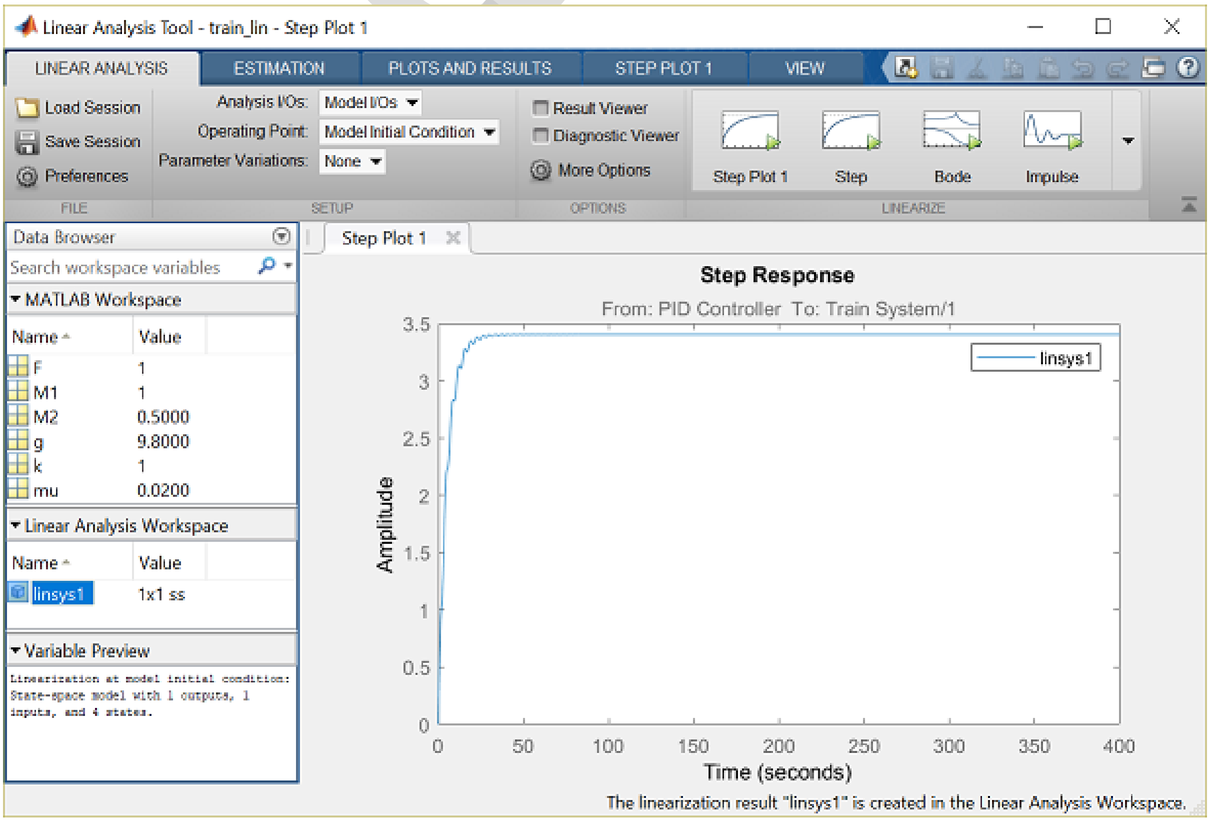
جعبه ابزار طراحی کنترل سیمولینک به ما قابلیت استخراج مدل از سیمولینک به مدل را می‌دهد. این ویژگی به خصوص برای سیستم‌های پیچیده یا غیرخطی مفید می‌باشد. همچنین برای ایجاد مدل گسسته (نمونه‌برداری شده) نیز از این قابلیت استفاده می‌شود. برای این مثال، مدل پیوسته برای زیر سیستم قطار را استخراج می‌کنیم. در قدم اول باید ورودی‌ها و خروجی‌ها را مشخص کنیم. ورودی سیستم قطار، نیروی F می‌باشد. برای تخصیص ورودی، بر روی سیگنال F راست کلیک کرده (خروجی بلوک PID) و گزینه Linear Analysis Points > Open-loop Input را انتخاب کنید. به همین منوال می‌توان خروجی سیستم قطار را با راست کلیک بر روی سیگنال "x1_dot" و انتخاب Linear Analysis Points > Open-loop Output تخصیص داد. این ورودی و خروجی‌ها توسط یک فلش کوچک در شکل نمایش داده می‌شود. از آنجایی که می‌خواهیم تنها مدل خود قطار (بدون کنترل) را به دست آوریم، باید سیگنال فیدبک را حذف کنیم وگرنه مدل حلقه بسته سیستم از F به $\dot{x}_1$ به دست می‌آید. حال مدل ما به شکل زیر است:



حال می‌توانیم با بازکردن Linear Analysis Tool، مدل را استخراج کنیم. از منوی Analysis گزینه Control Design > Linear Analysis را انتخاب کنید. با دنبال کردن مراحل ذکر شده، پنجره‌ی زیر به نمایش در می‌آید:



این ابزار، از یک مدل سیمولینک (احتمالا غیرخطی) یک شئ LTI می‌سازد که حول یک نقطه‌ی مشخص خطی‌سازی شده است. چون مدل سیمولینک ما خطی می‌باشد، انتخاب نقطه‌ی کاری تاثیری نخواهد داشت (زیرا خطی‌سازی لازم نیست) پس این مقدار را به طور پیش‌فرض Model Initial Condition قرار می‌دهیم. برای تولید مدل خطی‌سازی شده، بر روی دکمه‌ی Step بالای شکل کلیک کنید. حال پنجره Linear Analysis Tool به شکل زیر است:



در حال حاضر پاسخ پله سیستم خطی‌سازی شده به طور خودکار ایجاد می‌شود. با مقایسه‌ی پاسخ پله تولید شده توسط شبیه‌سازی سیستم حلقه باز در قسمت مقدمه‌ای بر مدل‌سازی سیمولینک، می‌توان دید که پاسخ‌های به دست آمده مشابه هستند. این تشابه کاملا معقول می‌باشد زیرا مدل ما خطی است. علاوه بر آن، فرآیند خطی‌سازی، شئ linsys1 را در فضای کاری Linear Analysis تولید کرده است. این شئ LTI را می‌توان با کشیدن و رها کردن در فضای کاری متلب در پنجره Linear Analysis Tool، به محیط متلب وارد نمود.

با استخراج این مدل، می‌توان از تمام قابلیت‌های متلب برای طراحی کنترلر استفاده نمود. برای مثال دستورات زیر را برای تحلیل و ایجاد سیستم حلقه بسته موجود در سیمولینک ساخته شده، اجرا کنید.

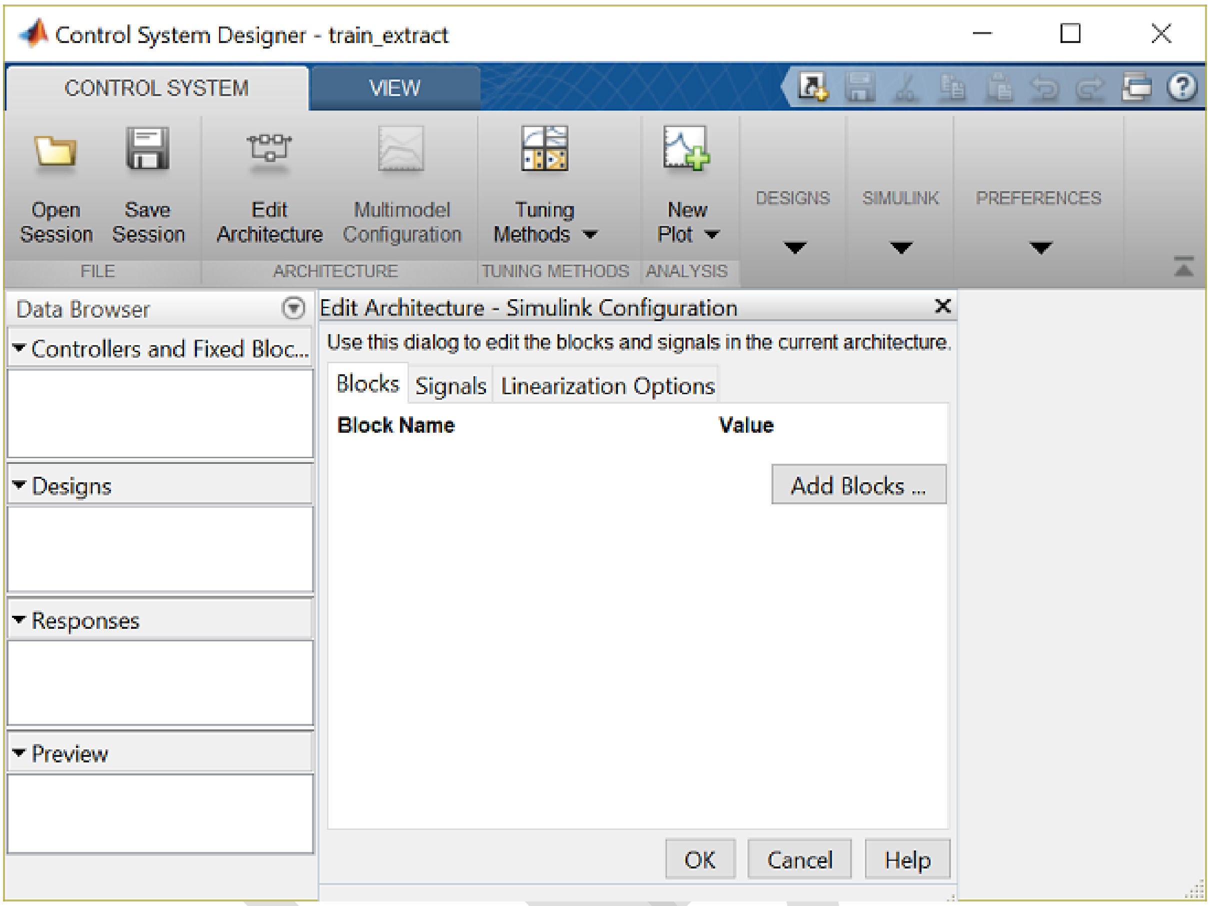
```
sys_cl = feedback(linsys1,1);

p = pole(sys_cl)

z = zero(sys_cl)
```

```
p =
-0.9237 + 0.0000i
0.0000 + 0.0000i
-0.2342 + 1.6574i
-0.2342 - 1.6574i


z =
-0.0980 + 1.4108i
-0.0980 - 1.4108i
0.0000 + 0.0000i
```

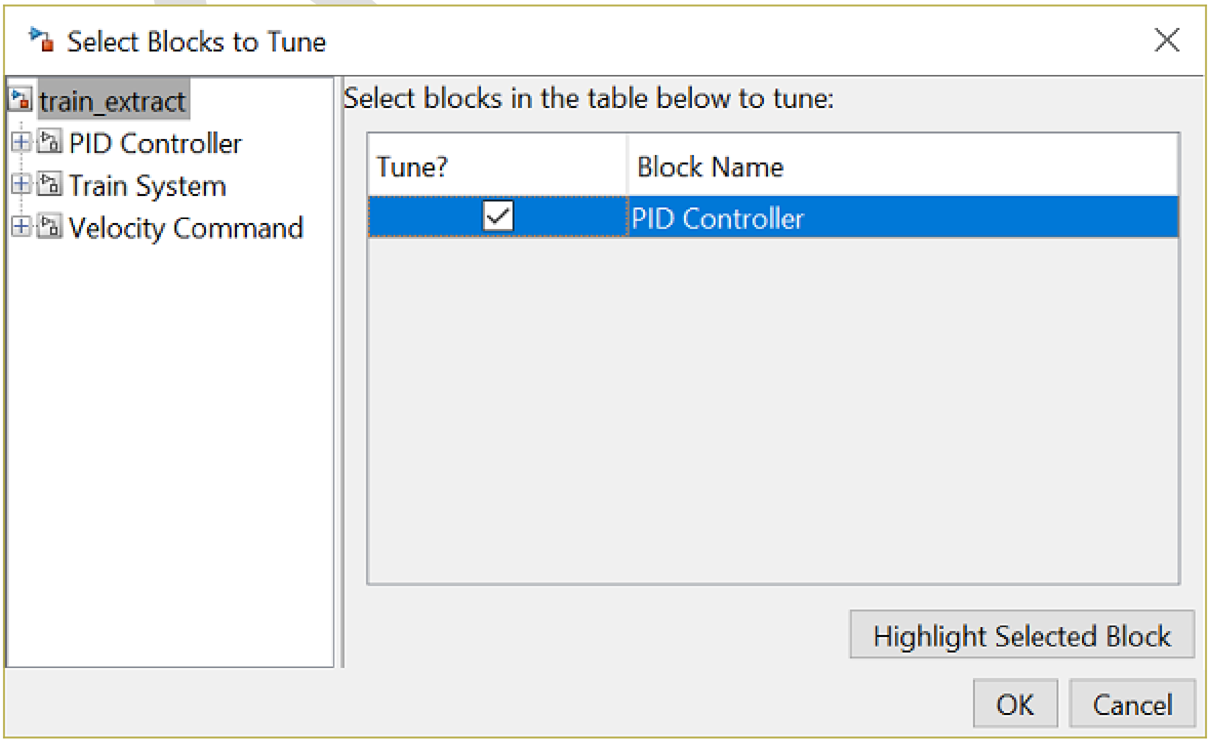
با بررسی نتیجه به دست آمده، متوجه حذف صفر و قطب در مبدا می‌شویم. بعلاوه قطب‌های باقی مانده قسمت حقیقی بزرگی منفی دارند و آهسته ترین قطب‌های آنها مختلط هستند. این امر نشان‌دهنده‌ی این است که سیستم حلقه بسته‌ی کنونی پایدار بوده و قطب‌های غالب آن، زیرمیرا هستند. این موارد با نتایج به دست آمده در بالا همخوانی دارند. در قدم بعد می‌توانیم از متلب برای طراحی کنترلر جدید مثلا برای کاهش نوسانات پاسخ سیستم استفاده نمود. در این بخش به روش طراحی کنترلر به طور مستقیم از سیمولینک می‌پردازیم.

طراحی کنترلر در سیمولینک

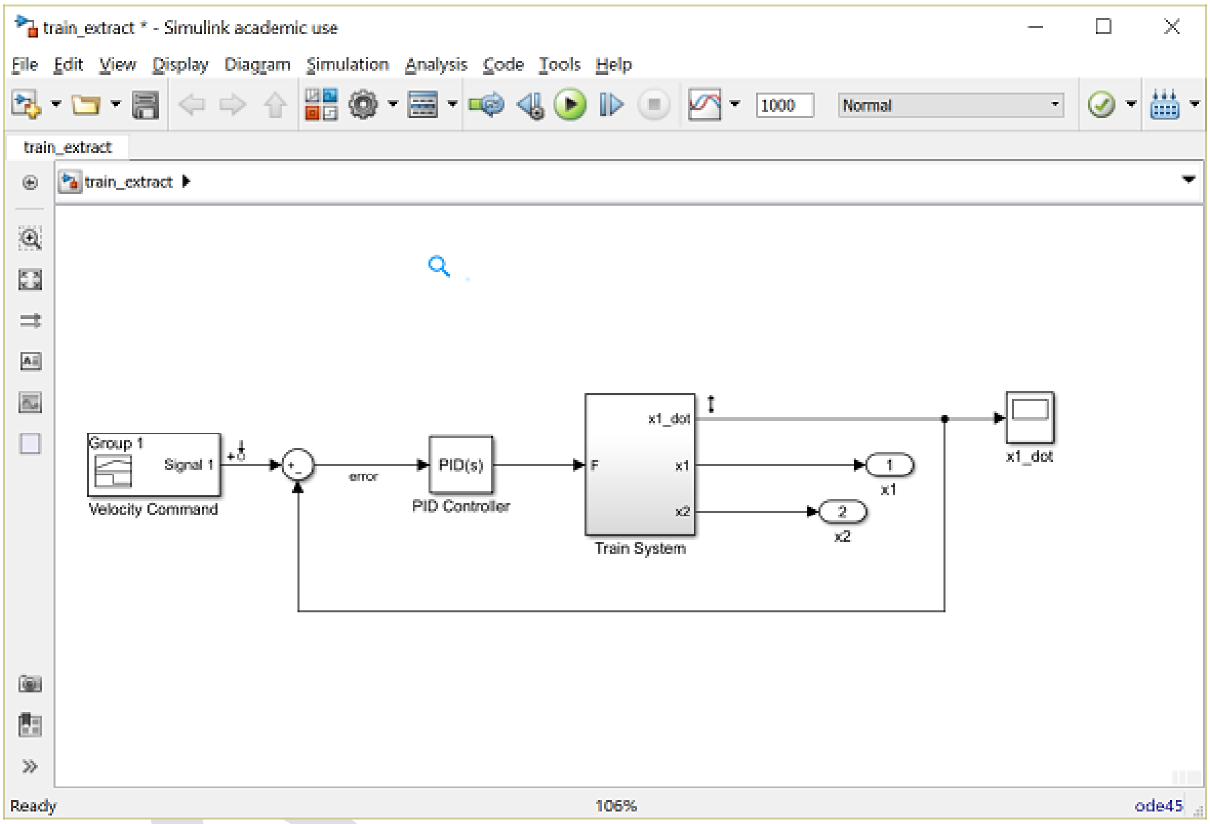
برای طراحی کنترلر در سیمولینک می‌توان از ابزارهای گوناگونی استفاده نمود. یکی از روش‌ها برای اینکار، کلیک راست بر روی بلوک کنترلر PID و انتخاب دکمه Tune می‌باشد که سبب اجرای ابزار PID Tuner می‌شود. ما در اینجا به جای اینکار از ابزاری کلی‌تری با نام Control System Designer استفاده می‌کنیم. برای اجرای این ابزار از منوی Analysis گزینه‌ی Control Design > Control System Designer را انتخاب کنید. با اینکار پنجره‌ی زیر باز می‌شود.



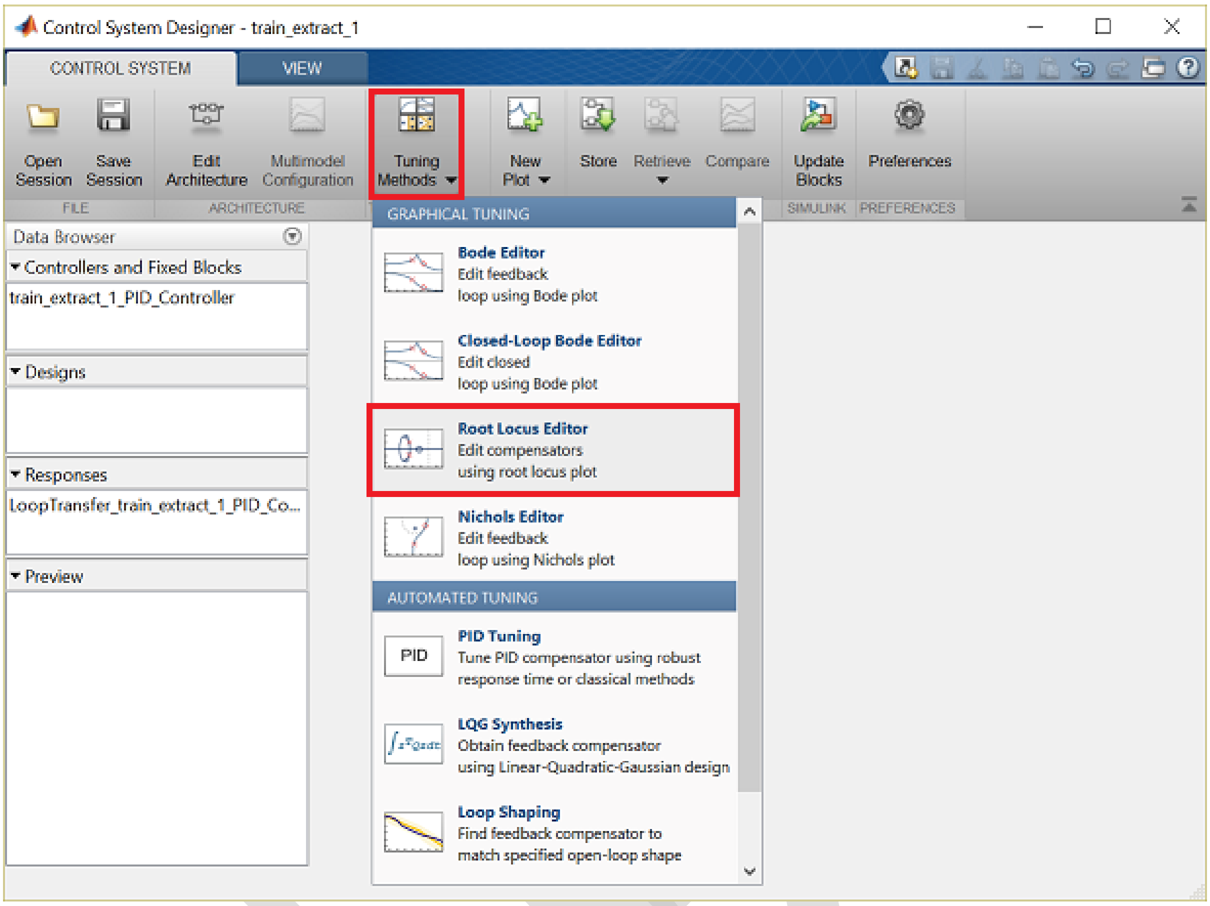
اولین کاری که باید انجام داد، تعیین بلوک کنترلری است که قرار است تنظیم شود. برای تعیین بلوک کنترلر، ابتدا بر روی دکمه Add Blocks کلیک کرده و سپس از پنجره‌ی نشان داده شده، بلوک PID Controller را انتخاب کنید. لازم به ذکر است که کنترلرهایی که به شکل بلوک‌های دیگر نمایش داده می‌شوند (تابع تبدیل، فضای حالت و ...) را نیز با این روش می‌توان تنظیم کرد.



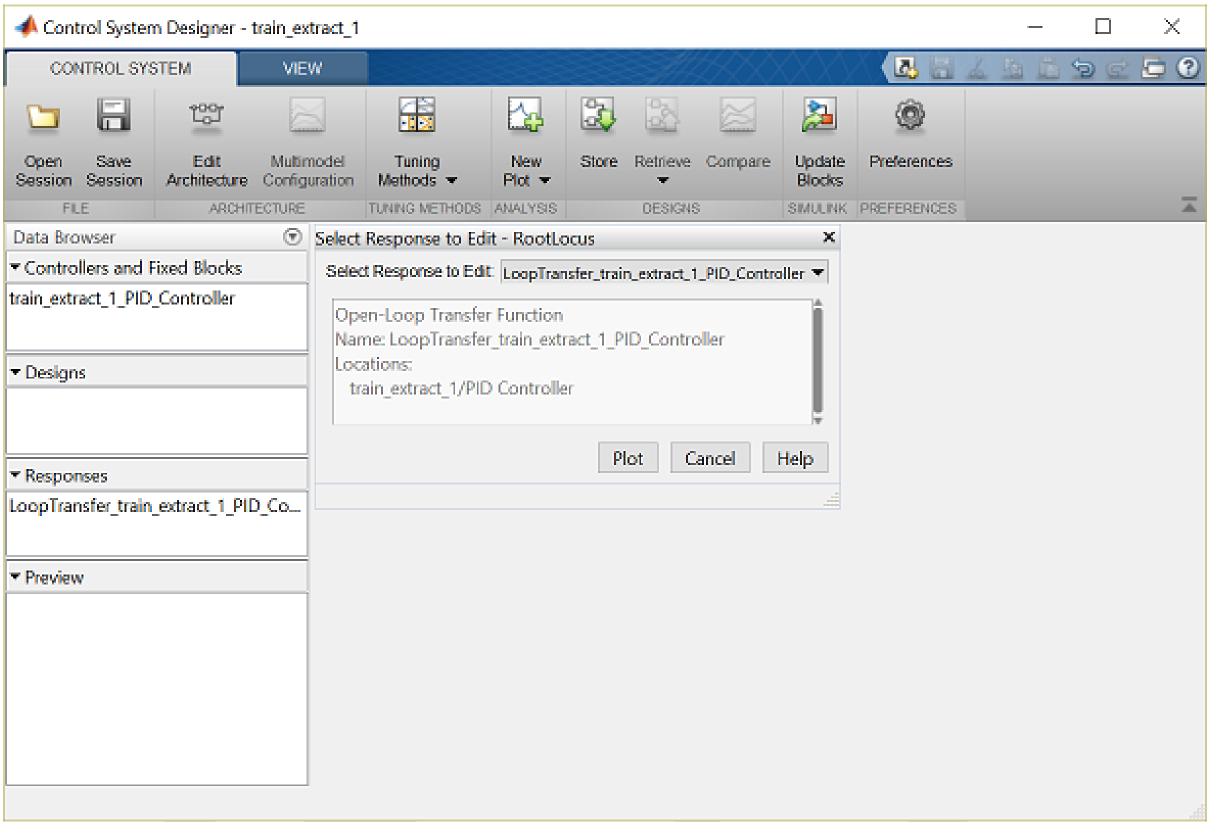
قبل از تنظیم کنترلر، ابتدا باید ورودی و خروجی سیستم حلقه بسته را که قصد تحلیل آنرا داریم تعیین کنیم. اینکار مشابه‌ی کاری است که در قسمت استخراج مدل خطی‌سازی شده به متلب انجام دادیم. بر روی سیگنال دستور سرعت کلیک راست کرده (سیگنال خروجی از بلوک Signal Builder) و گزینه‌ی Linear Analysis Points > Input Perturbation را انتخاب کنید تا ورودی سیستم حلقه بسته تعیین گردد. سپس بر روی سیگنال سرعت لوگوموتیو ("x1_dot") کلیک راست کرده و گزینه Linear Analysis Points > Output Measurement را انتخاب کنید. حال مدل ما به شکل زیر که دارای فلش‌های کوچک بر روی ورودی و خروجی می‌باشد درآمده است.



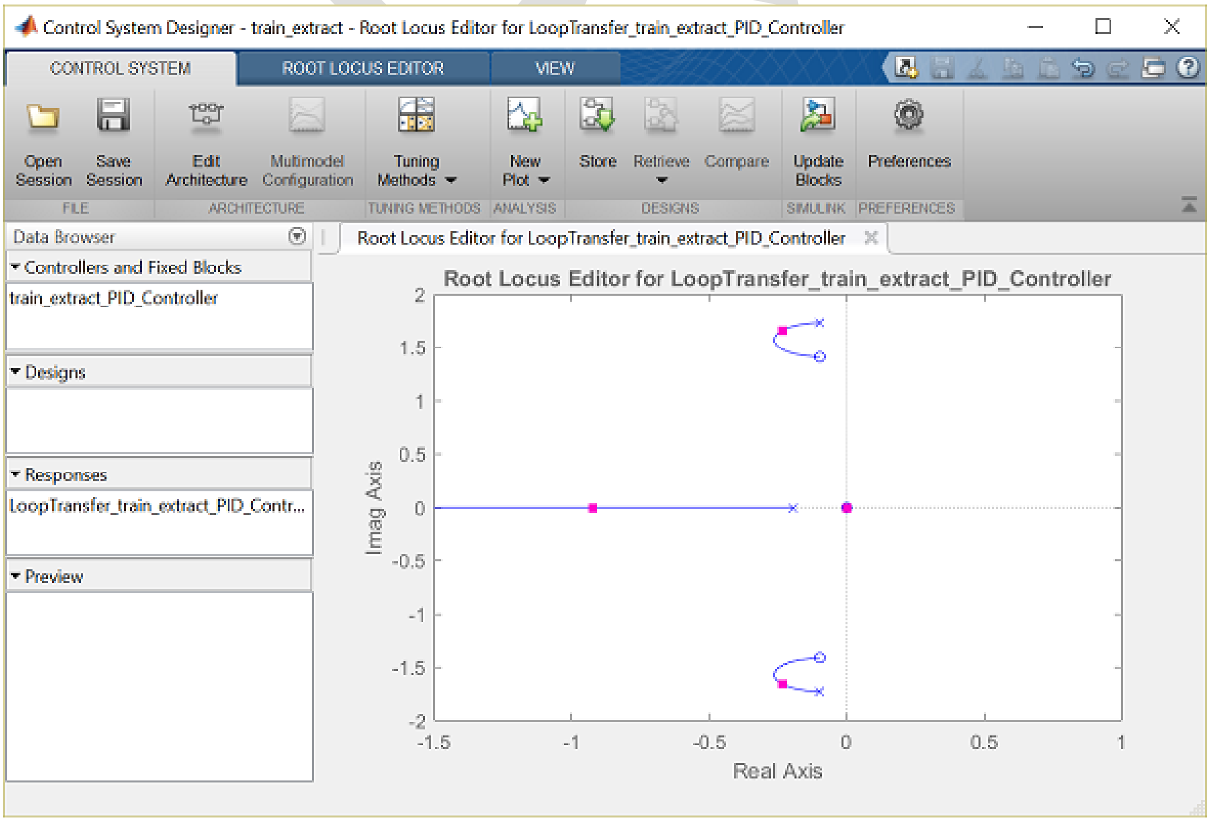
اکنون بلوک کنترلر و سیگنال‌های ورودی و خروجی سیستم تعیین شده‌اند و نوبت به تنظیم کنترلر است. بر روی دکمه OK در پنجره‌ی Edit Architecture کلیک کنید. حال پنجره‌ی زیر نمایان می‌شود.



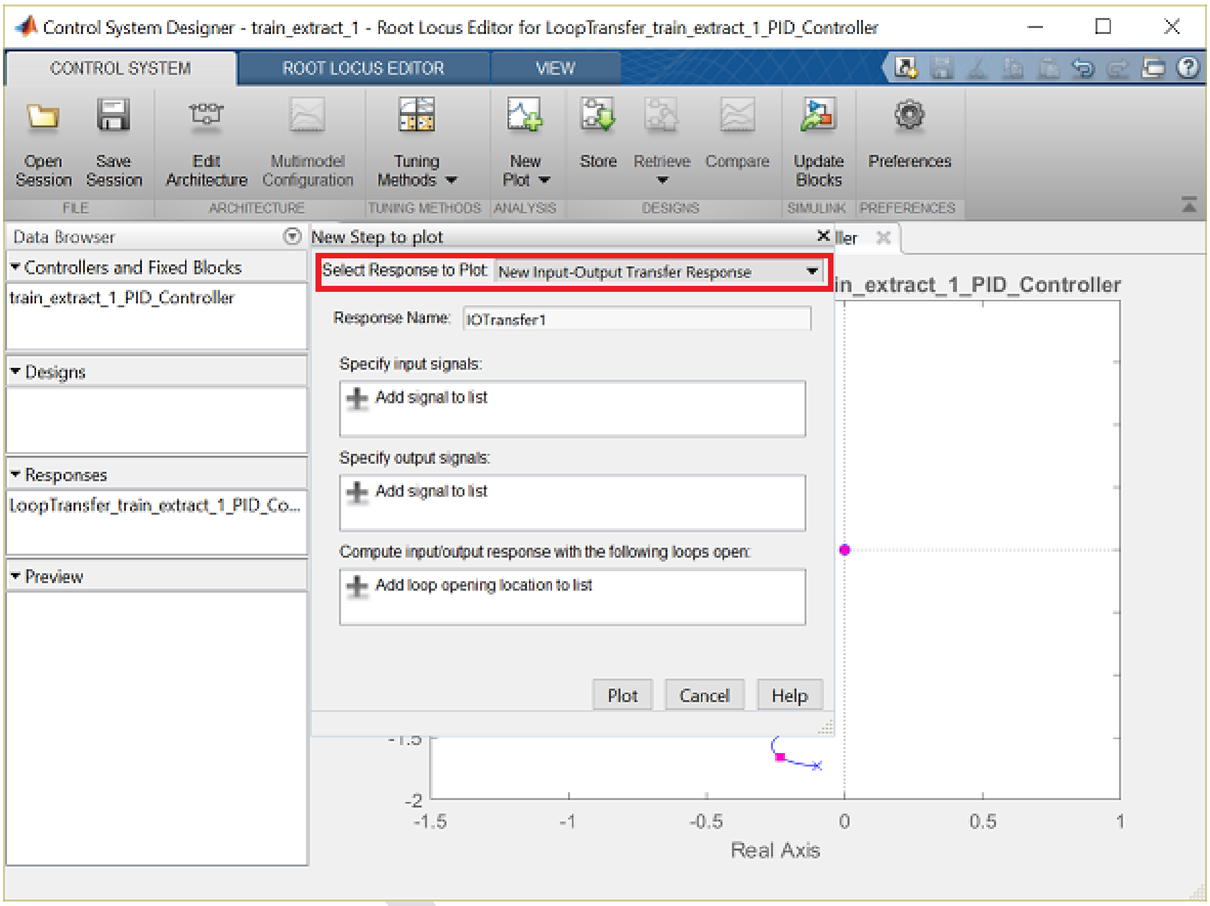
با کلیک بر روی گزینه Tuning Methods، نمودار طراحی که قصد داریم از آن برای طراحی استفاده کنیم را انتخاب می‌کنیم. در این مثال از روش طراحی مکان هندسی ریشه‌ها استفاده می‌کنیم پس در بخش Graphical Tuning گزینه‌ی Root Locus Editor را انتخاب کنید. روش طراحی به کمک مکان هندسی ریشه‌ها، از یک نمودار که در آن تمامی موقعیت‌های قطب‌های حلقه بسته بر حسب بهره‌ای که از صفر تا بینهایت تغییر می‌کند نمایش داده شده است استفاده می‌کند. از پنجره‌ی Select Response to Edit دکمه Plot را انتخاب کنید.



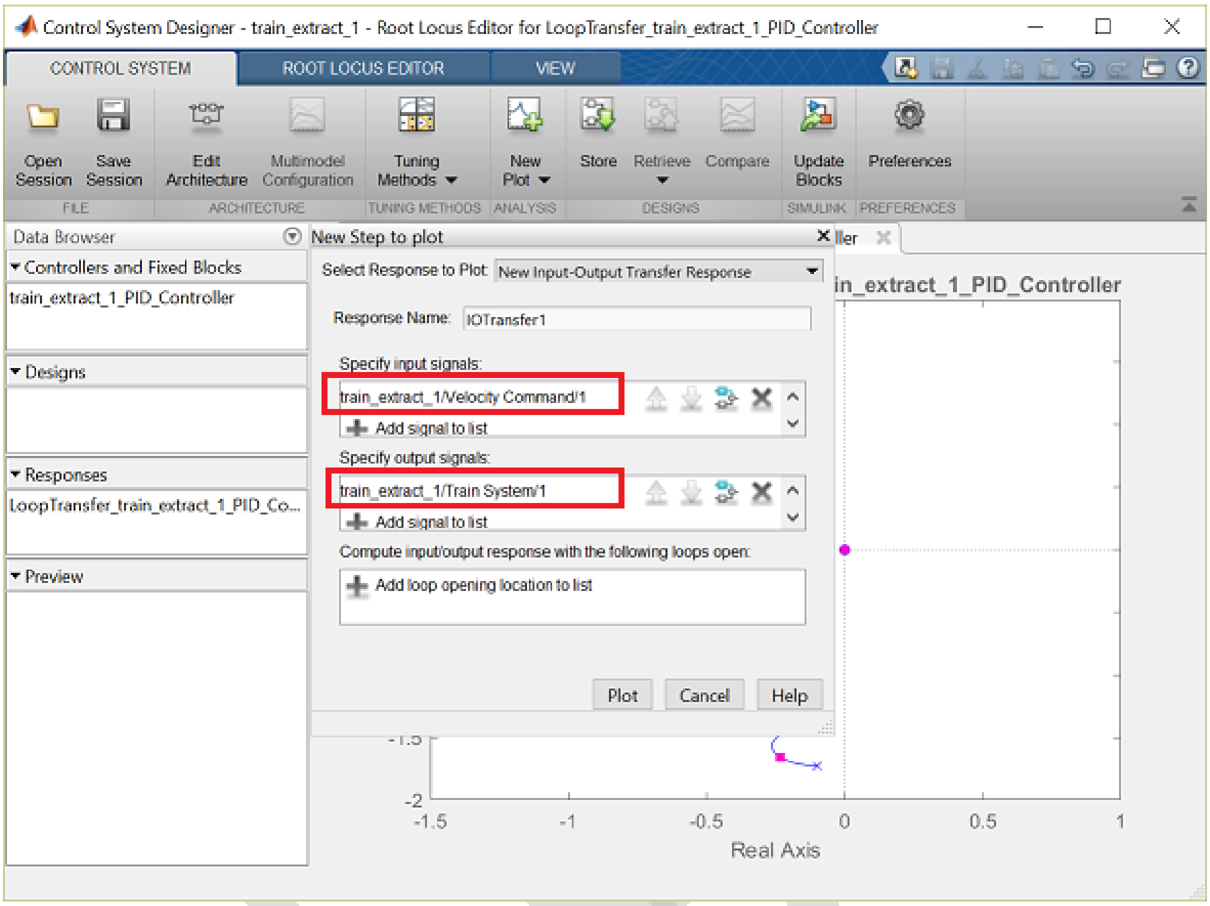
حال باید نمودار مکان هندسی ریشه‌ها مانند زیر نمایش داده شود. با بررسی نمودار می‌توان دریافت که به ازای تمامی مقادیر بهره، قطب‌های سیستم حلقه بسته در سمت چپ محور موهومی قرار می‌گیرد که به معنی پایدار بودن پاسخ سیستم است.



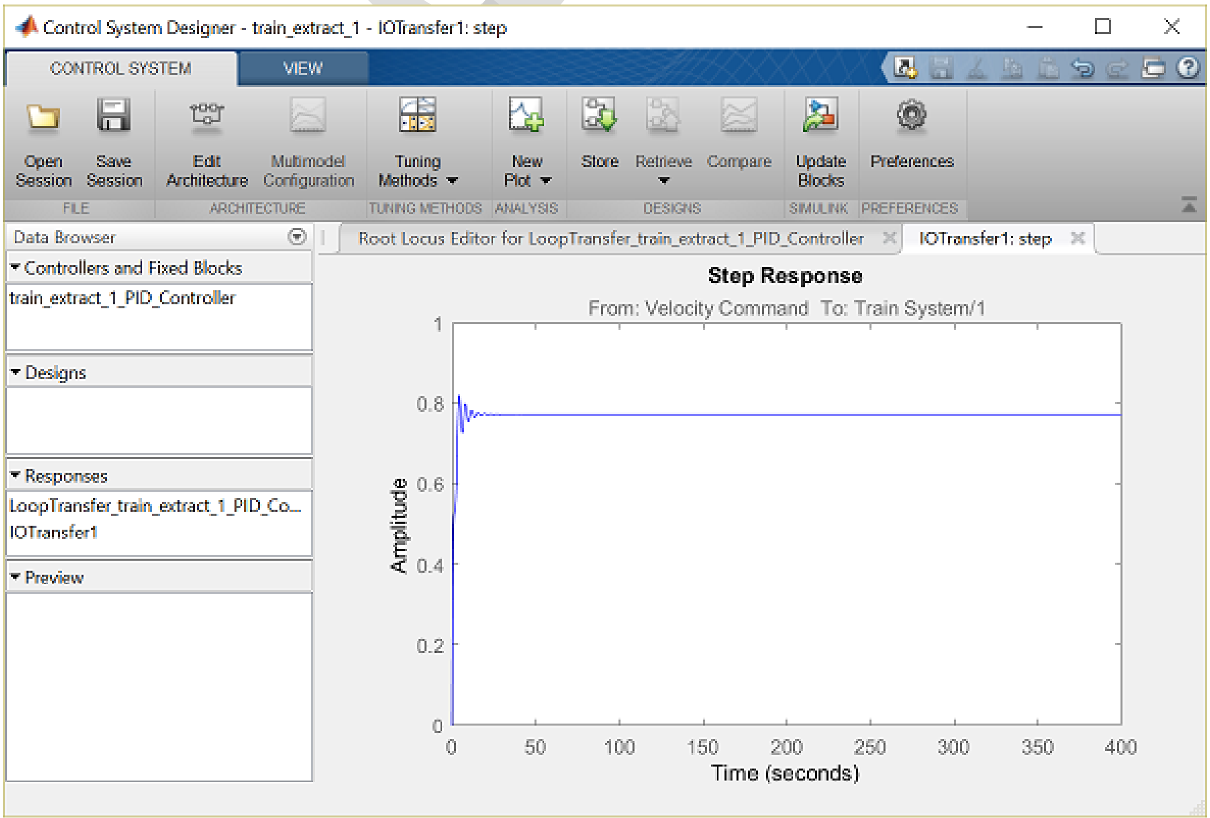
با کاهش قابل توجه بهره‌ی حلقه، می‌توانیم قطب‌های حلقه بسته را از محور موهومی دورتر کرده و در نتیجه عملکرد سیستم را تغییر دهیم. برای اینکار از نمودار گرافیکی نشان داده شده نقاط مربعی کوچک صورتی رنگ را با ماوس گرفته و آنها را به موقعیت قطب‌های حلقه باز ببرید (در نمودار با x نمایش داده شده‌اند). برای بررسی پاسخ پله سیستم حلقه بسته‌ی متناسب با قطب‌های انتخاب شده، در زبانه‌ی Control System بر روی New Plot کلیک کرده و آنگاه گزینه‌ی New Step را انتخاب کنید. با نمایش این پنجره، از منوی کشویی مقابل Select Response to Plot عبارت New Input-Output Transfer Response را انتخاب می‌کنیم.



سپس سیگنال‌های ورودی و خروجی را در پنجره‌ی باز شده‌ی بالا انتخاب می‌نماییم.



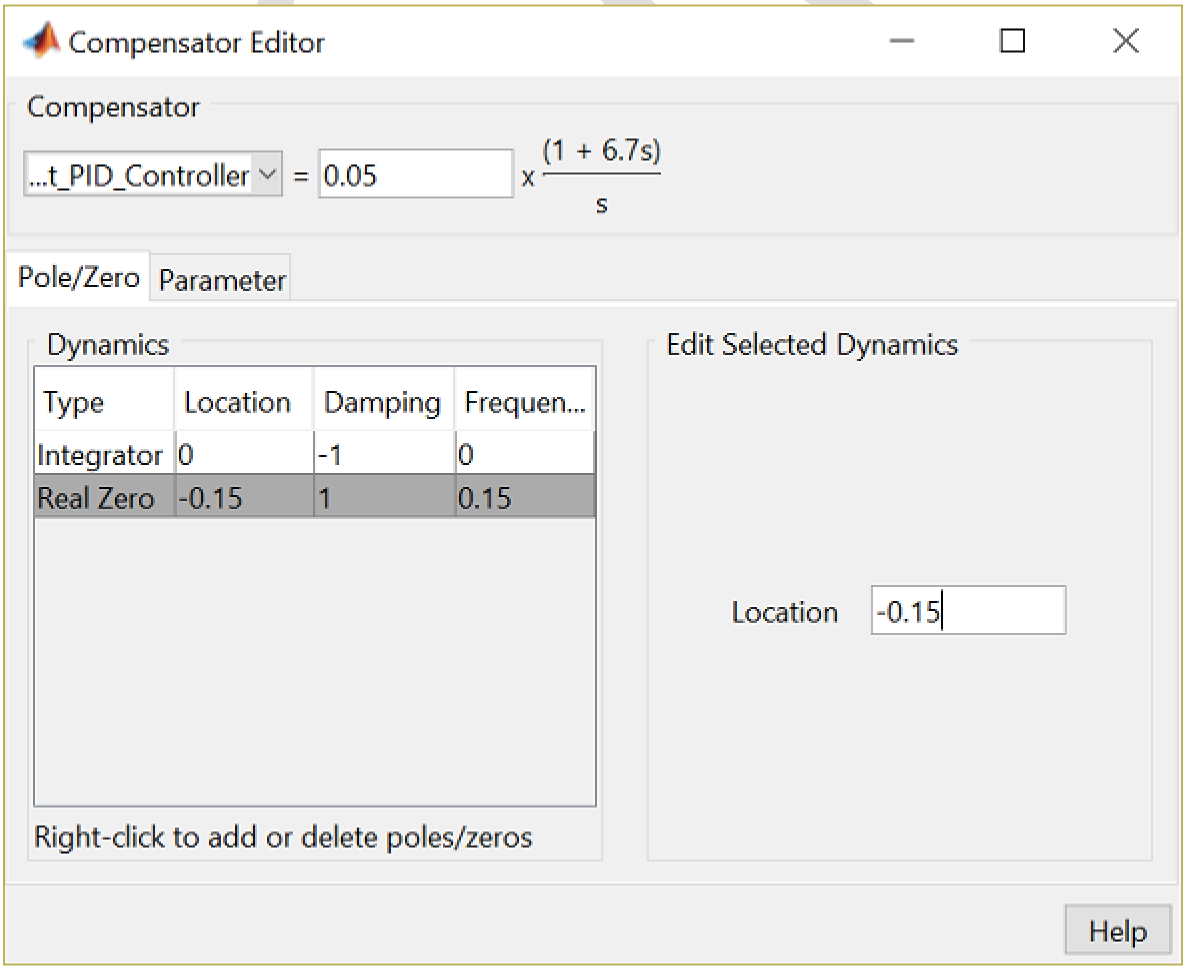
سپس بر روی کلید Plot کلیک کنید. از پاسخ پله‌ی حلقه بسته به دست آمده می‌توان مشاهده کرد که پاسخ پایدار می‌باشد اما دارای خطای حالت ماندگار است.



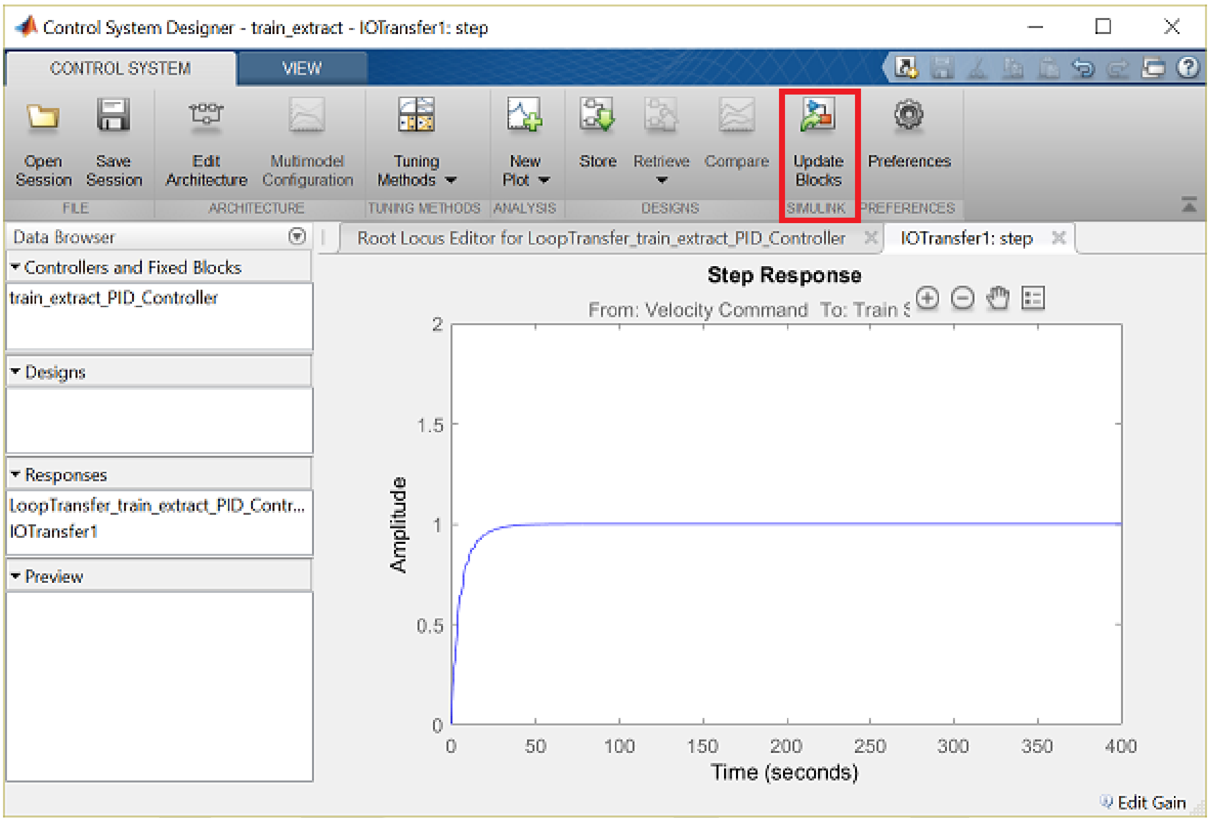
همانطور که گفته شده بود، اضافه کردن کنترل انتگرالی سبب کاهش خطای حالت ماندگار در سیستم حلقه بسته می‌گردد. در این موارد، اضافه کردن انتگرال‌گیر به کنترلر، سیستم را از نوع ۱ می‌کند و سیستم‌های نوع ۱ می‌توانند ورودی پله را بدون خطای حالت ماندگار دنبال کنند. فرم کلی کنترلر PI به شکل زیر است:

$$C(s) = K_p + \frac{K_i}{s} = \frac{K_p s + K_i}{s} \qquad (۱)$$

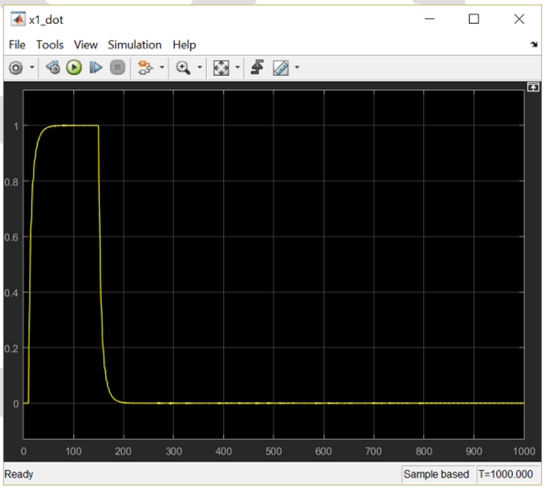
با بررسی این معادله متوجه می‌شویم که کنترلر PI یک انتگرال‌گیر و یک صفر به سیستم حلقه باز اضافه می‌کند. برای اضافه کردن انتگرال‌گیر به سیستم، بر روی فضای نمودار مکان هندسی ریشه‌ها راست کلیک کرده و از منوی نشان داده شده گزینه Add Pole/Zero > Integrator را انتخاب کنید. به طور مشابه برای اضافه کردن صفر، بر روی نمودار راست کلیک کرده و گزینه Add Pole/Zero > Real Zero را انتخاب کنید و سپس بر روی مکانی از محور حقیقی که قصد اضافه کردن صفر را دارید کلیک کنید. برای جابجا کردن این صفر می‌توانید آنرا با کلیک ماوس گرفته و به موقعیت دیگر بکشید. کنترلر (یا جبران‌ساز) را می‌توان به طور مستقیم با وارد کردن موقعیت‌های صفر و قطب ویرایش کرد. برای اینکار در محیط نمودار مکان هندسی ریشه‌ها راست کلیک کرده و گزینه Edit Compensator را انتخاب کنید. پنجره‌ی باز شده به شکل زیر است. یک انتگرال‌گیر و صفر حقیقی در 0.15- را اضافه شده و بهره‌ی حلقه را برابر ۰/۰۵ قرار می‌دهیم.



پاسخ پله سیستم حلقه بسته‌ی حاصل در نمودار زیر نشان می‌دهد که لوکوموتیو قطار به آرامی شروع به حرکت کرده و دستور سرعت ثابت را بدون خطای حالت ماندگار دنبال می‌کند.



بهره‌های کنترل انتخاب شده را می‌توان با کلیک بر روی دکمه Update Blocks در زبانه‌ی Control System به مدل سیمولینک اعمال کرد. با اینکار شبیه‌سازی‌های انجام شده با کنترلر تنظیم شده انجام می‌گردد. با کلیک بر روی Scope سرعت قطار، نمودار مشابه نمودار بالا به دست می‌آید.



به نظر می‌رسد که این پاسخ، اهداف ما را که شروع حرکت به آرامی و توقف نرم قطار است ارضا کرده و بدون خطای حالت ماندگار می‌باشد. این پاسخ مشابه پاسخ تولید شده توسط Control System Designer است زیرا هر دوی آنها از یک مدل خطی استفاده کرده‌اند.

فصل سوم: کنترل کروز خودرو

بخش اول: مدل‌سازی سیستم

فهرست مطالب بخش

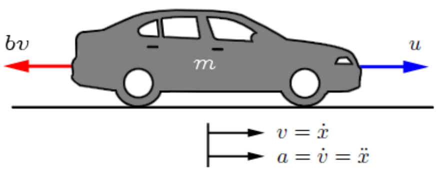
- سیستم فیزیکی
- معادلات سیستم
- پارامترهای سیستم
- مدل فضای حالت
- مدل تابع تبدیل

دستورهای کلیدی متلب در این بخش:

```
ss, tf
```

سیستم فیزیکی

سیستم کنترل سرعت اتوماتیک خودرو (کروز) یک مثال بارز از سیستم کنترل فیدبک می‌باشد که بر روی بسیاری از خودروها استفاده می‌شود. هدف سیستم کنترل کروز، ثابت نگه داشتن سرعت خودرو فارغ از اغتشاشات خارجی وارد شده مانند تغییر شدت و جهت باد و شیب جاده می‌باشد. با اندازه‌گیری سرعت خودرو و مقایسه‌ی آن با سرعت مطلوب یا مرجع، می‌توان به طور خودکار شدت گاز خودرو را با توجه به قوانین کنترل، تنظیم نمود.



در اینجا یک مدل ساده‌ی دینامیکی خودرو را در نظر می‌گیریم که دیاگرام جسم آزاد آن را در تصویر بالا مشاهده می‌کنید. خودرو با جرم m، تحت نیروی کنترلی u قرار دارد. نیروی u بیانگر نیروی تولید شده توسط اتصال چرخ با زمین می‌باشد. برای این مدل ساده، فرض می‌کنیم مستقیما می‌توانیم این نیرو را کنترل کنیم و از دینامیک سیستم انتقال قدرت خودرو، لاستیک‌ها و ... صرف نظر می‌کنیم. نیروی مقاومت bv ناشی از مقاومت غلتشی و نیروی پسای[۲۶] باد می‌باشد. این نیرو خطی و متناسب با سرعت خودرو و در جهت خلاف حرکت خودرو در نظر گرفته می‌شود.

معادلات سیستم

با این فرضیات، سیستم ما مانند یک سیستم جرم و دمپر مرتبه اول می‌باشد. با استفاده از قانون دوم نیوتن در راستای x، معادلات سیستم به دست می‌آید:

$$m\dot{v} + bv = u \qquad (۱)$$

چون می‌خواهیم سرعت خودرو را کنترل کنیم، خروجی معادله را سرعت خودرو در نظر می‌گیریم یعنی:

$$y = v \qquad (۲)$$

پارامترهای سیستم

[۲۶] Drag

برای این مثال، فرض کنید که پارامترهای سیستم به شکل زیر است:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

مدل فضای حالت

سیستم‌های مرتبه اول تنها دارای یک منبع ذخیره انرژی می‌باشد که در این مثال انرژی جنبشی خودرو این نقش را بازی می‌کند. برای این سیستم‌ها تنها یک متغیر حالت نیاز می‌باشد. در نتیجه نمایش فضای حالت به شکل زیر می‌باشد:

$$\dot{\mathbf{x}} = [\dot{v}] = \left[\frac{-b}{m}\right][v] + \left[\frac{1}{m}\right][u] \tag{۳}$$

$$y = [1][v] \tag{۴}$$

این معادلات فضای حالت را با دستورات زیر در متلب وارد می‌کنیم:

```
m = 1000;
b = 50;

A = -b/m;
B = 1/m;
C = 1;
D = 0;

cruise_ss = ss(A,B,C,D);
```

مدل تابع تبدیل

با گرفتن تبدیل لاپلاس از معادلات دیفرانسیل حاکم بر سیستم و در نظر گرفتن شرایط اولیه صفر، تابع تبدیل برای سیستم کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms + b} \quad \left[\frac{m/s}{N}\right] \tag{۵}$$

با اجرای دستور زیر، این تابع تبدیل وارد متلب می‌گردد:

```
s = tf('s');
P_cruise = 1/(m*s+b);
```

بخش دوم: تحلیل سیستم

فهرست مطالب بخش

- مدل سیستم و پارامترها
- ویژگی‌های عملکرد
- پاسخ پله‌ی حلقه باز
- قطب‌ها/صفرهای حلقه باز
- دیاگرام بودی حلقه باز

دستورهای کلیدی متلب در این بخش:

```
ss, step
```

مدل سیستم و پارامترها

مدل تابع تبدیل برای سیستم کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \qquad (۱)$$

پارامترهای استفاده شده در این مثال عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

ویژگی‌های عملکرد

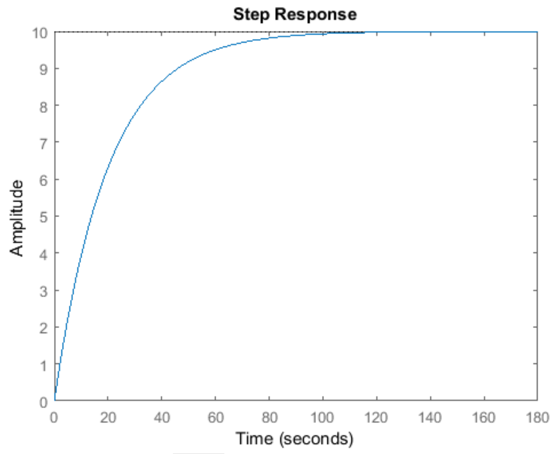
برای قدم بعد باید نیازهای طراحی را در نظر بگیریم که سیستم کنترل شده بتواند آنرا ارضا کند. وقتی که موتور نیروی ۵۰۰ نیوتن را تولید می‌کند، خودرو به حداکثر سرعت 10 m/s می‌رسد که در نمودار پاسخ پله‌ی حلقه باز می‌توان مشاهده نمود. خودرو باید بتواند در کمتر از ۵ ثانیه به این سرعت برسد. با در نظر گرفتن شرط ذکر شده، نیازهای طراحی این مسئله را به شکل زیر عنوان می‌کنیم:

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

پاسخ پله‌ی حلقه باز

پاسخ پله به نیروی ۵۰۰ نیوتنی بدون هیچگونه کنترل فیدبک، با استفاده از اجرای دستورات زیر به دست می‌آید:

```
m = 1000;
b = 50;
u = 500;


s = tf('s');
P_cruise = 1/(m*s+b);

step(u*P_cruise)
```

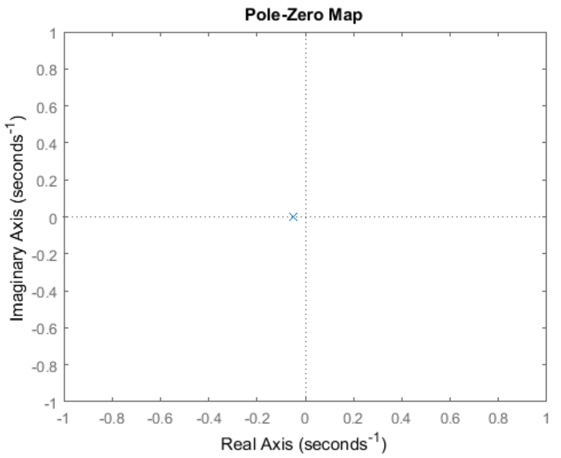


مشاهده می‌شود که سیستم حلقه باز هیچگونه فراجهش یا نوسانی ندارد (از ویژگی‌های سیستم‌های مرتبه اول) و در حالت ماندگار به سرعت مطلوب 10 m/s نمی‌رسد. همچنین زمان نمو بسیار آهسته (تقریبا برابر ۶۰ ثانیه) است. بنابراین باید کنترلر فیدبکی طراحی کنیم تا سریعا سرعت را بالا برده و بر روی سایر فاکتورهای عملکرد سیستم نیز تاثیر منفی نگذارد.

قطب‌ها/صفرهای حلقه باز

سیستم کنترل کروز دارای یک قطب در s=-b/m می‌باشد که می‌توان با دستور زیر، آنها را در صفحه‌ی مختلط s نمایش داد:

```
pzmap(P_cruise)

axis([-1 1 -1 1])
```

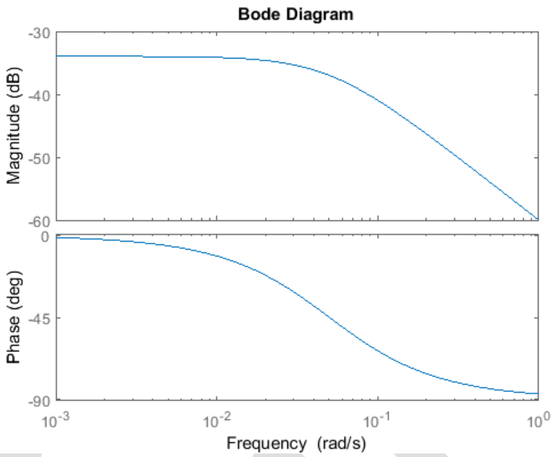


مشاهده می‌کنیم که سیستم حلقه باز پایدار بوده و به دلیل اینکه قطب سیستم، حقیقی و منفی می‌باشد نوسانی در پاسخ وجود ندارد. علاوه بر آن سرعت پاسخ سیستم توسط اندازه‌ی قطب آن یعنی b/m تعیین می‌گردد. هرچه این اندازه بزرگتر باشد، پاسخ سیستم سریع‌تر به حالت ماندگار می‌رسد. چون ما برای تغییر پاسخ سیستم، قادر به تغییر پارامترهای سیستم نمی‌باشیم و باید کنترلری را طراحی کنیم تا صفرها و قطب‌های سیستم حلقه بسته به گونه‌ای قرار بگیرند تا عملکرد سیستم مطلوب شود.

دیاگرام بودی حلقه باز

برای طراحی کنترلر، یکی از اطلاعات مفیدی که می‌توان از سیستم داشت، پاسخ فرکانسی حلقه باز سیستم می‌باشد که با دستور متلب زیر به دست می‌آید:

```
bode(P_cruise)
```



مشاهده می‌شود که در دیاگرام بودی، ویژگی‌های سیستم‌های مرتبه اول مانند اندازه ‎-3 db و فاز ‎-45 deg در فرکانس گوشه‌ی برابر w=b/m=0.05 rad/s و همچنین شیب ‎-20 db/dec در فرکانس‌های بالا نمایان شده است.

بخش سوم: طراحی کنترلر PID

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
tf, step, feedback
```

مدل سیستم و پارامترها

مدل تابع تبدیل برای مسئله کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \tag{۱}$$

پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

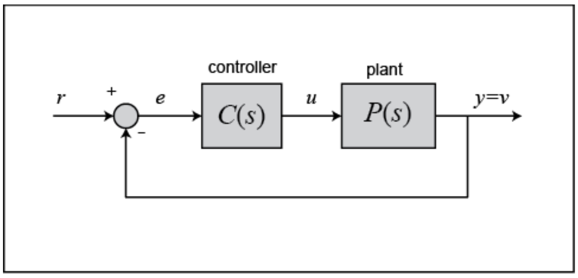
(u) نیروی کنترلی = ۵۰۰ نیوتن

ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

کلیات PID

دیاگرام بلوکی یک سیستم با فیدبک واحد در تصویر زیر نشان داده شده است:



برای یادآوری، تابع تبدیل کنترلر PID عبارتست از:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \tag{۲}$$

با دستور زیر می‌توان کنترلر PID را در متلب مستقیماً به فرم تابع تبدیل تعریف نمود:

```
Kp = 1;
Ki = 1;
Kd = 1;

s = tf('s');
C = Kp + Ki/s + Kd*s
```

```
C =

  s^2 + s + 1
  -----------
       s

Continuous-time transfer function.
```

همچنین می‌توان از شئ pid controller متلب بهره برده و کنترلر پیوسته با زمان PID را تعریف نمود:

```
C = pid(Kp,Ki,Kd)
```

```
C =

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

  with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
```

کنترل تناسبی

اولین قدم در این مسئله، پیدا کردن تابع تبدیل حلقه بسته سیستم با یک کنترل تناسبی $(C = K_p)$ می‌باشد.

تابع تبدیل سیستم با کنترل تناسبی عبارتست از:

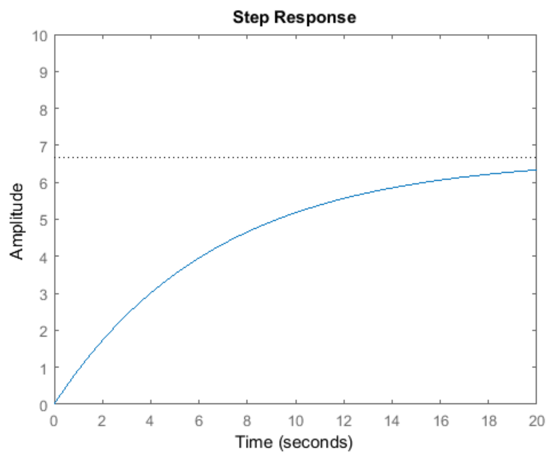
$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_p}{ms + b + K_p} \qquad (٣)$$

برای یادآوری لازم به ذکر است که کنترلر تناسبی K_p، زمان نمو را کاهش می‌دهد که در اینجا مطلوب می‌باشد.

حال از ضریب تناسبی K_p برابر ۱۰۰ و سرعت مرجع برابر 10 m/s استفاده می‌کنیم. در یک ام‌فایل جدید، دستورات زیر را وارد کنید:

```
m = 1000;
b = 50;
r = 10;

s = tf('s');
P_cruise = 1/(m*s + b);

Kp = 100;
C = pid(Kp);

T = feedback(C*P_cruise,1)

t = 0:0.1:20;
step(r*T,t)
axis([0 20 0 10])
```

```
T =

        100
  ------------
  1000 s + 150

Continuous-time transfer function.
```



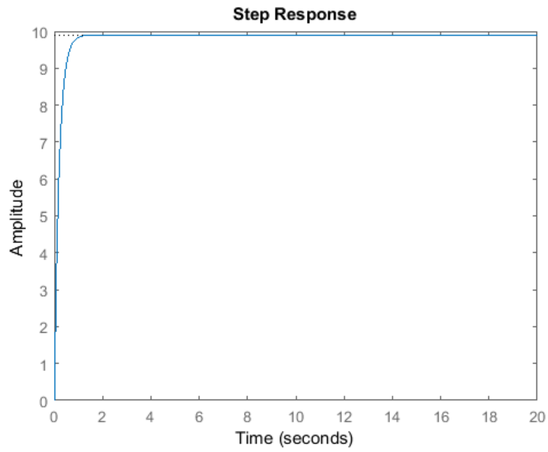
در کد بالا، از دستور `feedback` استفاده شده است که این دستور دیاگرام بلوکی را به فرم حلقه بسته تبدیل می‌کند. برای اطمینان از نتیجه‌ی به دست آمده، می‌توانید تابع تبدیل حلقه بسته‌ی T به دست آمده را به طور دستی حساب کنید.

با اجرای این امفایل در متلب، پاسخ پله‌ی فوق به دست می‌آید. همانطور که از نمودار مشخص است، نه زمان نمو و نه خطای حالت ماندگار رضایت‌بخش نیستند.

برای کاهش زمان نمو و خطای حالت ماندگار، می‌توان K_p را افزایش داد. مقدار K_p را در امفایل به ۵۰۰۰ تغییر داده و آنرا دوباره اجرا کنید. با اینکار نمودار زیر به دست می‌آید.

```
Kp = 5000;
C = pid(Kp);
T = feedback(C*P_cruise,1);
```

```
step(r*T,t)
axis([0 20 0 10])
```



حال خطای حالت ماندگار صفر می‌باشد و زمان نمو نیز کاهش چشمگیری داشته است. اگرچه این پاسخ غیر واقعی می‌باشد زیرا یک سیستم کنترل کروز واقعی نمی‌تواند در کمتر از ۰/۵ ثانیه، سرعت خودرو را از 0 به 10 m/s برساند زیرا محدودیت‌های زیادی برای موتور و سیستم انتقال قدرت وجود دارد.

محدودیت‌های عملگر یکی از مواردی است که اغلب در مسائل واقعی مهندسی به آن بر می‌خوریم و در نتیجه در هنگام طراحی کنترل جدید باید آنرا در نظر گرفت. در فصل‌های بعد به این مسئله بیشتر خواهیم پرداخت.

راه حل این مسئله، انتخاب بهره‌ی تناسبی کمتر که زمان نشست معقول را ایجاد کند می‌باشد. همچنین برای حذف خطای حالت ماندگار از کنترلر انتگرالی استفاده می‌کنیم.

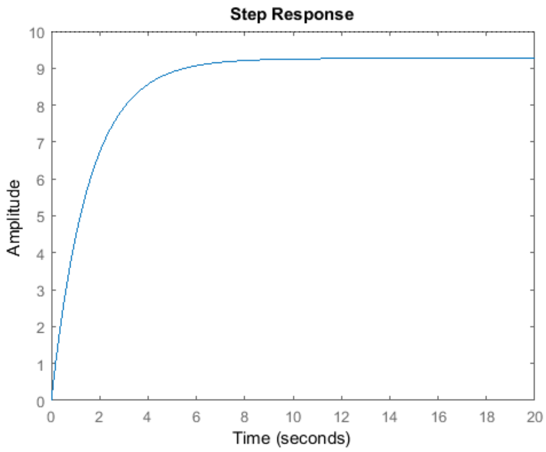
کنترل PI

تابع تبدیل سیستم حلقه بسته با کنترلر PI ($C = K_p + K_i/s$) به شکل زیر است:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_P s + K_i}{ms^2 + (b + K_p)s + K_i} \qquad (۴)$$

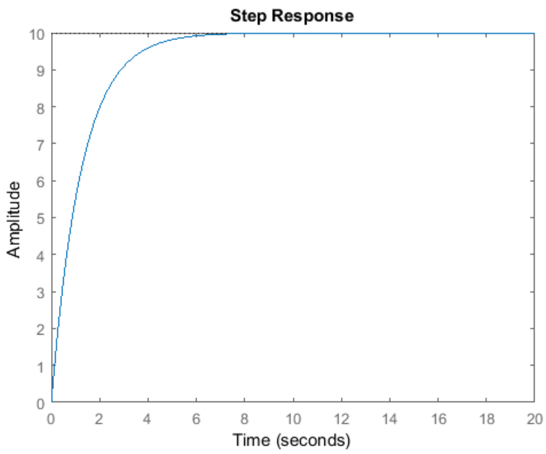
به یاد داریم که با اضافه شدن کنترلر انتگرالی به سیستم، خطای حالت ماندگار حذف می‌گردد. حال K_p را برابر ۶۰۰ و K_i را برابر ۱ قرار دهید تا پاسخ جدید به دست آید.

```
Kp = 600;
Ki = 1;

C = pid(Kp,Ki);
T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```



حال باید دو بهره‌ی تناسبی K_p و بهره‌ی انتگرالی K_i را تنظیم نمود تا پاسخ مطلوب حاصل شود. در هنگام تنظیم بهره‌ی انتگرالی K_i توصیه می‌شود که از مقادیر کوچک شروع کنید زیرا مقادیر بزرگ K_i می‌تواند سیستم را ناپایدار سازد. با قرار دادن $K_p = 800$ و $K_i = 40$، پاسخ پله سیستم حلقه بسته به شکل زیر در می‌آید:

```
Kp = 800;
Ki = 40;

C = pid(Kp,Ki);
T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```



کنترل PID

برای این مثال، نیازی به کنترل مشتق برای رسیدن به خروجی مطلوب نمی‌باشد اما برای نشان دادن روش کار کنترل PID در این بخش به آن می‌پردازیم. تابع تبدیل سیستم به همراه کنترلر PID ($C = K_p + K_i/s + K_d s$) به شکل زیر است:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_d s^2 + K_P s + K_i}{(m + K_d)s^2 + \left(b + K_p\right)s + K_i} \qquad (۵)$$

مقادیر بهره‌های کنترلر را با دستورات زیر وارد متلب کنید:

```
Kp = 1;
Ki = 1;
Kd = 1;

C = pid(Kp,Ki,Kd);
T = feedback(C*P_cruise,1);
```

پاسخ پله را رسم کرده و ضرایب K_p، K_i و K_d را به گونه‌ای تنظیم می‌کنیم تا نتیجه‌ی رضایت‌بخش حاصل شود. این قسمت را به عنوان تمرین بر عهده‌ی خواننده می‌گذاریم.

توصیه: معمولا برای انتخاب بهره‌های مناسب، نیاز به سعی و خطا می‌باشد. بهترین روش برای اینکار، تنظیم کردن هر یک از بهره‌ها به نوبت و مشاهده تغییرات حاصل در خروجی سیستم می‌باشد. مشخصات بهره‌های K_p، K_i و K_d در **فصل اول - بخش سوم: کنترلر PID** آمده است.

بخش چهارم: طراحی کنترلر با مکان هندسی ریشه‌ها

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
tf, rlocus, feedback, step
```

مدل سیستم

مدل تابع تبدیل برای مسئله‌ی کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \qquad (۱)$$

پارامترهای سیستم

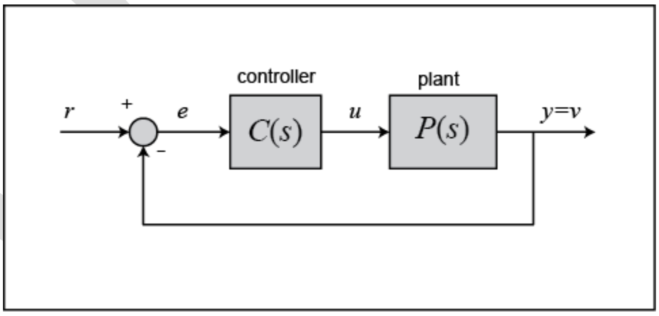
پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

دیاگرام بلوکی سیستم با فیدبک واحد به شکل زیر است:



ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

کنترل تناسبی

از فصل مقدمه‌ای بر طراحی کنترلر با مکان هندسی ریشه‌ها به خاطر داریم که نمودار مکان هندسی ریشه‌ها، موقعیت تمامی قطب‌های حلقه بسته‌ی ممکن را وقتی بهره‌ی سیستم از صفر تا بینهایت تغییر می‌کند را نشان می‌دهد. بنابراین

برای حل این مسئله تنها از یک کنترلر تناسبی K_p استفاده می‌گردد. در این صورت تابع تبدیل حلقه بسته به شکل زیر در می‌آید:

$$\frac{Y(s)}{R(s)}=\frac{K_p}{ms+(b+K_p)} \tag{۲}$$

همچنین می‌دانیم با دستور `sgrid` می‌توانیم نواحی مطلوب نمودار مکان هندسی ریشه‌ها را مشخص کنیم. برای استفاده از دستور `sgrid` لازم است تا ضریب میرایی ζ و فرکانس طبیعی ω_n مشخص باشند. با استفاده از دو معادله‌ی زیر می‌توان ضریب میرایی و فرکانس طبیعی را به دست آورد:

$$\omega_n \geq \frac{1.8}{T_r} \tag{۳}$$

$$\zeta \geq \sqrt{\frac{\ln^2(M_p)}{\pi^2+\ln^2(M_p)}} \tag{۴}$$

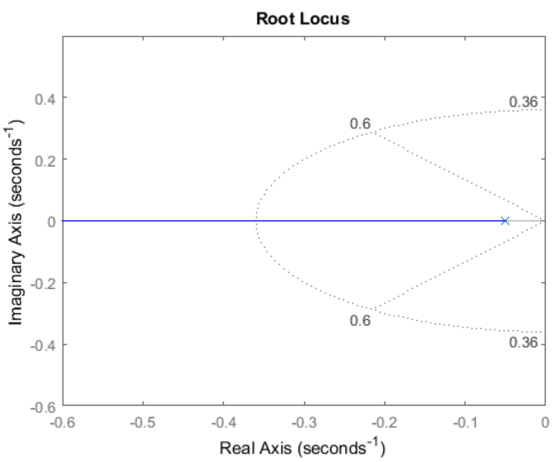
در این معادلات داریم:

- ω_n فرکانس طبیعی بر حسب rad/s
- ζ ضریب میرایی
- Tr زمان نمو بر حسب ثانیه
- Mp ماکزیمم فراجهش

یکی از نیازهای طراحی، زمان نمو کمتر از ۵ ثانیه می‌باشد. با استفاده از معادله‌ی اول فرکانس طبیعی باید بزرگتر از ۰/۳۶ باشد. همچنین از معادله‌ی دوم در می‌یابیم ضریب میرایی باید بیشتر از ۰/۶ باشد تا ماکزیمم فراجهش از ۱۰ ٪ کمتر شود.

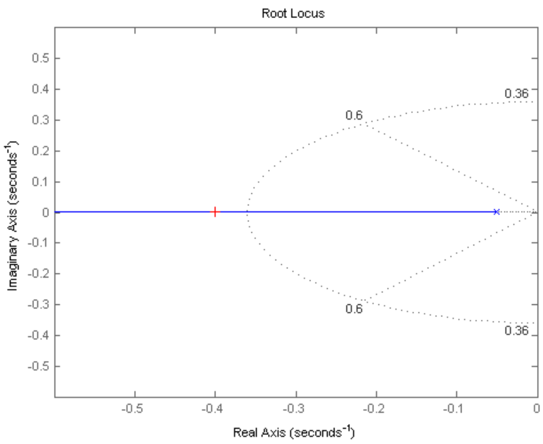
اکنون می‌توان نمودار مکان هندسی ریشه‌ها را با استفاده از دستور `sgrid` برای مشخص کردن نواحی قابل قبول، رسم نمود. در یک ام‌فایل جدید دستورات زیر را وارد کنید:

```
m = 1000;
b = 50;
r = 10;

s = tf('s');
P_cruise = 1/(m*s+b);

rlocus(P_cruise)
axis([-0.6 0 -0.6 0.6]);
sgrid(0.6, 0.36)
```



خطوط خط‌چین زاویه دار، موقعیت ضریب میرایی ثابت ($\zeta = 0.6$) را مشخص می‌کند. در بین این دو خط، ضریب میرایی بزرگ‌تر از ۰/۶ و در خارج از آنها کمتر از ۰/۶ می‌باشد. نیم‌بیضی خط‌چین نشان‌دهنده موقعیت فرکانس طبیعی ثابت ($\omega_n = 0.36$) می‌باشد. نقاط داخل آن دارای فرکانس طبیعی کمتر از ۰/۳۶ و خارج آن دارای فرکانس طبیعی بیشتر از ۰/۳۶ می‌باشند.

با داشتن ناحیه‌ی مطلوب، می‌توانیم با دستور `rlocfind` بهره‌ای را پیدا کنیم تا قطب‌های حلقه بسته در ناحیه‌ی مطلوب قرار بگیرند. کد `[Kp,poles]=rlocfind(P_cruise)` را در انتهای ام‌فایل خود اضافه کنید تا بهره‌ی مورد نظر را به دست آورید. با اجرای این دستور، پیامی را مشاهده می‌کنید که از شما می‌خواهد نقطه‌ای را در نمودار مکان هندسی ریشه‌ها انتخاب کنید. نقطه‌ی انتخاب شده باید بین خطوط خط‌چین ($\zeta > 0.6$) و خارج از نیم‌بیضی ($\omega_n > 0.36$) باشد. در نقطه‌ای خارج از نیم‌بیضی و بر روی محور حقیقی (تقریبا در 0.4-) نقطه‌ای را انتخاب کنید.

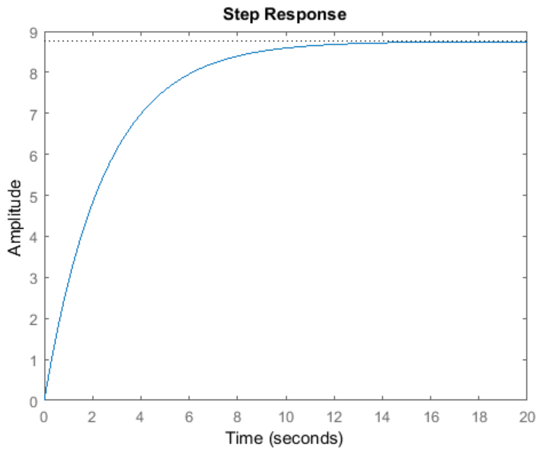


با این کار، خروجی زیر در متلب نمایش داده می‌شود:

```
Select a point in the graphics window

         selected_point =


         -0.4002 + 0.0019i

         Kp =


         350.2419

         poles =


         -0.4002
```

با استفاده از بهره‌ی به دست آمده، پاسخ پله‌ی سیستم حلقه بسته به شکل زیر در می‌آید:

```
Kp = 350.2419;

sys_cl = feedback(Kp*P_cruise,1);

t = 0:0.1:20;

step(r*sys_cl,t)
```



با بهره‌ی K_p استفاده شده، هر دو شرط زمان نمو و فراجهش ارضا شده‌اند، هرچند که خطای حالت ماندگار بیش از ۱۰٪ وجود دارد.

کنترلر پس‌فاز[۲۷]

برای کاهش خطای حالت ماندگار، به سیستم، کنترلر پس‌فاز را اضافه می‌نماییم. تابع تبدیل کنترلر پس‌فاز عبارتست از:

$$C(s) = \frac{s + z_0}{s + p_0} \qquad (۵)$$

در اینصورت تابع تبدیل سیستم (بدون عبارت K_p) به شکل:

$$\frac{Y(s)}{R(s)} = \frac{s + z_0}{ms^2 + (b + mp_0)s + bp_0} \qquad (۶)$$

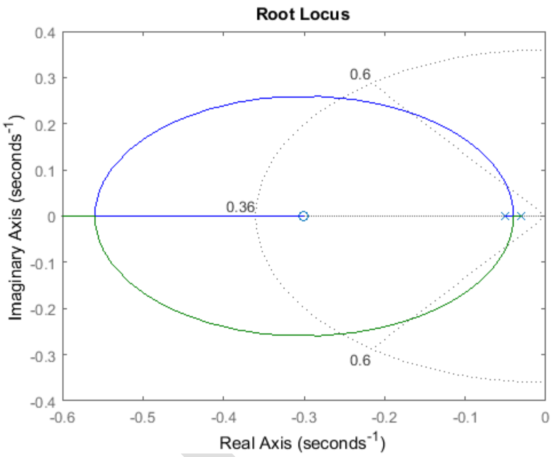
در می‌آید. در نهایت با اضافه کردن بهره‌ی K_p، تابع تبدیل نهایی به شکل زیر به دست می‌آید:

$$\frac{Y(s)}{U(s)} = \frac{K_p s + K_p z_0}{ms^2 + \left(b + mp_0 + K_p\right)s + \left(bp_0 + K_p z_0\right)} \qquad (۷)$$

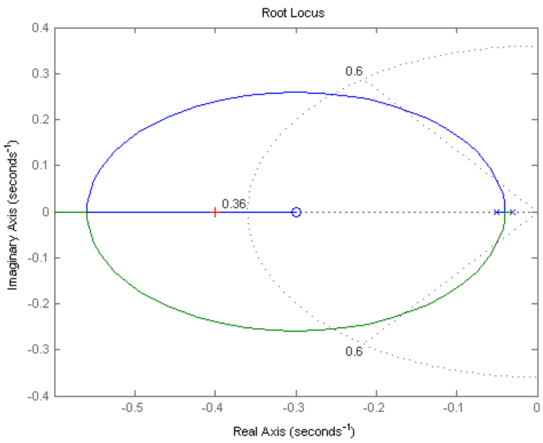
در **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز**، گفته شد که باید صفر و قطب کنترلر پس‌فاز در نزدیکی یکدیگر قرار بگیرند. همچنین گفته شد که خطای حالت ماندگار به نسبت z_0/p_0 کاهش می‌یابد. به همین دلیل z_0 را برابر ۰/۳ و p_0 را برابر ۰/۰۳ در نظر می‌گیریم. با ساخت و اجرای یک امفایل جدید نتیجه‌ی زیر حاصل می‌گردد:

```
zo = 0.3;
po = 0.03;

s = tf('s');
C_lag = (s+zo)/(s+po);

rlocus(C_lag*P_cruise);
axis([-0.6 0 -0.4 0.4])
sgrid(0.6,0.36);
```

[۲۷] Lag Controller



دوباره با اجرای دستور `rlocfind`، بهره‌ی جدید K_p به دست می‌آید. کد `[Kp,poles]=rlocfind(C_lag*P_cruise)` را اجرا کرده و نقطه‌ای در نزدیکی 0.4- بر روی محور حقیقی را انتخاب کنید.

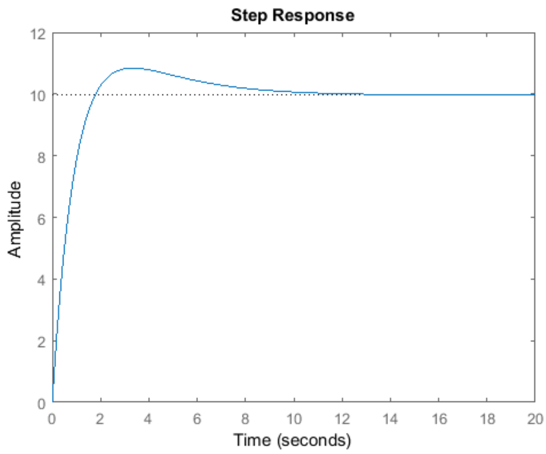


با انتخاب این نقطه، نتایج زیر حاصل می‌گردد:

```
Select a point in the graphics window

        selected_point =

        -0.4002 - 0.0012i

        Kp =

        1.2936e+03

        poles =

        -0.9733
        -0.4003
```

برای به دست آوردن پاسخ پله‌ی حلقه بسته‌ی جدید، کد زیر را اجرا کنید:

```
Kp = 1293.6;

sys_cl = feedback(Kp*C_lag*P_cruise,1);

t = 0:0.1:20;

step(r*sys_cl,t)

axis([0 20 0 12])
```



همانطور که مشاهده می‌شود خطای حالت ماندگار به نزدیکی صفر کاهش پیدا کرده است. فراجهش ایجاد شده ناشی از اضافه کردن صفر توسط کنترلر پس‌فاز می‌باشد. حال تمامی الزامات طراحی ارضا شده است اما برای تمرین، سایر مقادیر z_0 و p_0 را آزموده و تاثیر آنرا بر روی پاسخ سیستم حلقه بسته مشاهده کنید.

بخش پنجم: طراحی کنترلر در حوزه‌ی فرکانس

فهرست مطالب بخش

- مدل سیستم
- پارامترهای سیستم
- ویژگی‌های عملکرد
- دیاگرام بودی و پاسخ حلقه باز
- کنترلر تناسبی
- جبران‌ساز لگ

دستورهای کلیدی متلب در این بخش:

```
tf, feedback, step
```

مدل سیستم

مدل تابع تبدیل برای مسئله کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \qquad (۱)$$

پارامترهای سیستم

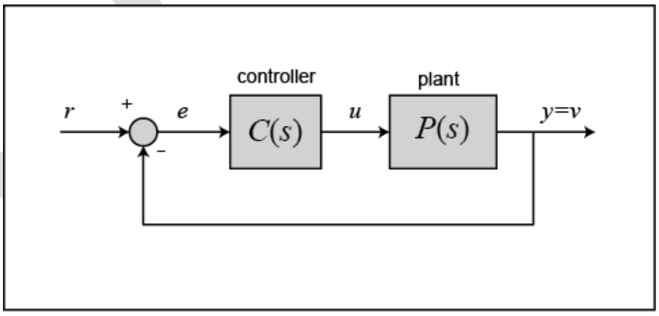
پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

دیاگرام بلوکی سیستم با فیدبک واحد به شکل زیر است:

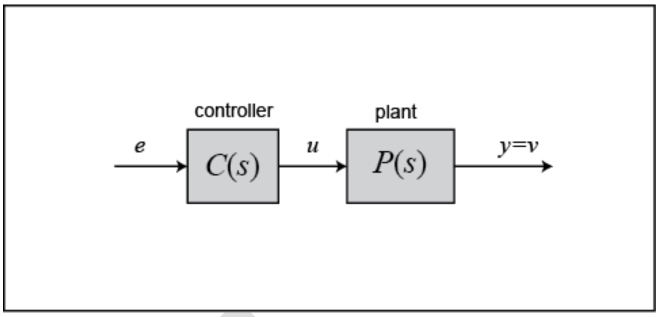


ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

دیاگرام بودی و پاسخ حلقه باز

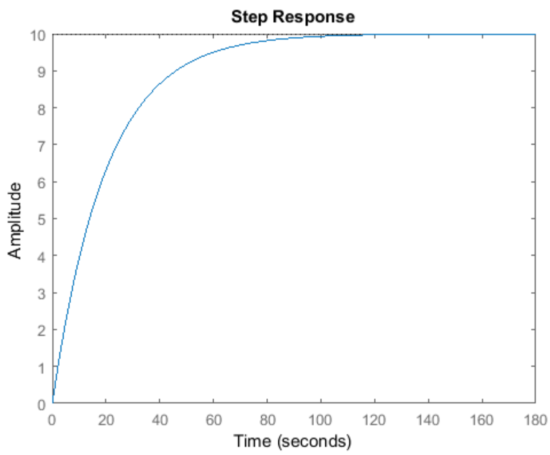
اولین قدم برای حل این مسئله از طریق پاسخ فرکانسی، مشخص کردن تابع تبدیل حلقه باز است. در این روش هم مانند روش مکان هندسی ریشه‌ها، از یک کنترل تناسبی برای حل مسئله استفاده خواهد شد. دیاگرام بلوکی و تابع تبدیل حلقه باز به شرح زیر است:



$$\frac{Y(s)}{E(s)} = \frac{K_p}{ms+b} \qquad (۲)$$

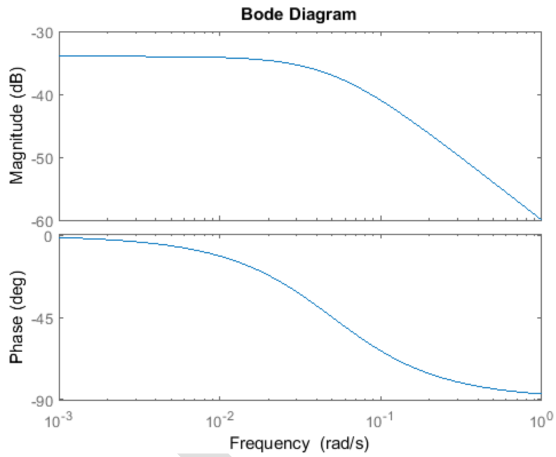
برای استفاده از دیاگرام بودی، پاسخ حلقه باز باید پایدار باشد. مقدار K_p را برابر ۱ در نظر گرفته و پاسخ حلقه باز را بررسی می‌کنیم. امفایل جدیدی را ساخته و دستورات زیر را در آن ذخیره و اجرا کنید:

```
m = 1000;
b = 50;
u = 500;

Kp = 1;

s = tf('s');
P_cruise = 1/(m*s+b);
C = Kp;
step(u*C*P_cruise)
```



همانطور که مشاهده می‌شود، سیستم حلقه باز پایدار است. بنابراین می‌توانیم دیاگرام بودی را رسم کنیم. امفایل بالا را تغییر داده و به جای دستور step، دستور زیر را قرار دهید و اجرا کنید:

```
bode(C*P_cruise);
```



کنترل تناسبی

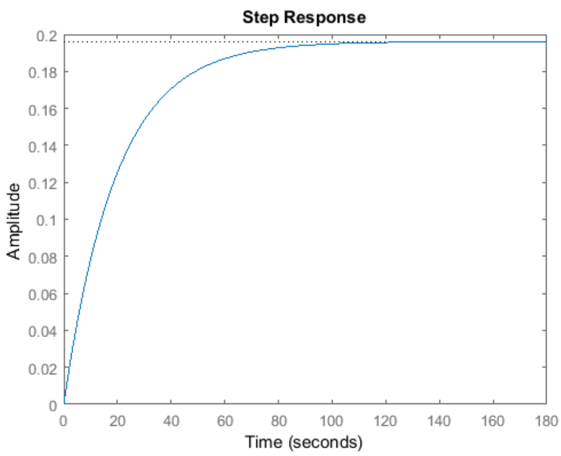
در **فصل دوم - بخش پنجم: مقدمه‌ای بر طراحی کنترلر در حوزه‌ی فرکانس**، گفته شد که چه ویژگی‌هایی از سیستم را می‌توان از دیاگرام بودی استنتاج کرد.

خطای حالت ماندگار از رابطه‌ی زیر به دست می‌آید:

$$\text{خطای حالت ماندگار} = \frac{1}{1+M_{\omega\to 0}} \times 100\% \qquad (٣)$$

برای این سیستم، بهره‌ی فرکانس پایین برابر 0.02 = 34db- می‌باشد در نتیجه خطای حالت ماندگار برابر ٩٨٪ به دست می‌آید. برای صحت این مطلب، می‌توانیم پاسخ پله حلقه بسته را رسم کنیم:

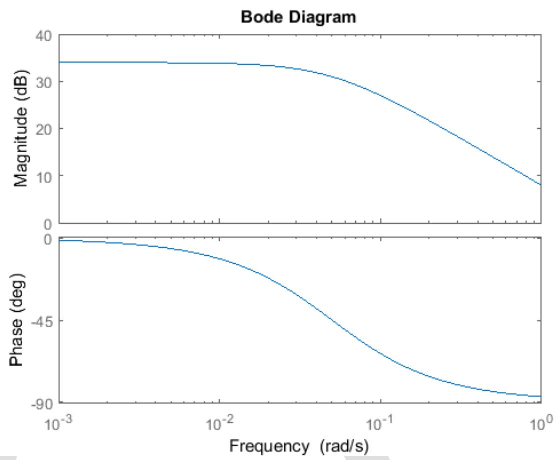
```
r = 10;
sys_cl = feedback(C*P_cruise,1);

step(r*sys_cl);
```



برای بهبود خطای حالت ماندگار باید بهره‌ی فرکانس پایین را افزایش دهیم. خطای حالت ماندگار باید کمتر از ٢٪ باشد، بنابراین داریم $1/(1+M_{\omega=0}) < 0.02$ پس $M_{\omega=0} > 49 = 33.8db$. چون بهره‌ی فرکانس پایین برابر 34db- می‌باشد و برای رسیدن به مقدار مطلوب خطای حالت ماندگار باید بهره‌ی فرکانس پایین برابر ٣٣/٨ دسیبل داشته باشیم، با

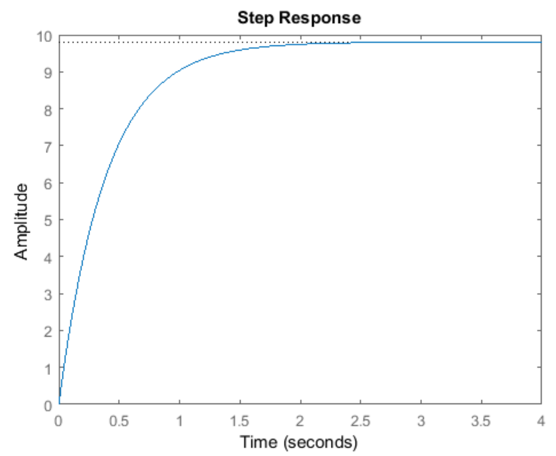
استفاده از تنها یک کنترلر تناسبی، لازم است که $K_p > (34db + 33.8db) = 67.8db = 2455$ باشد. حال با این بهره‌ی تناسبی، دیاگرام بودی سیستم حلقه باز را رسم می‌کنیم:

```
Kp = 2500;
C = Kp;

bode(C*P_cruise);
```



همانطور که از دیاگرام بودی بالا مشاهده می‌کنید، اندازه فرکانس پایین برابر ۳۴ دسیبل می‌باشد. حال با این بهره، پاسخ پله‌ی سیستم حلقه بسته را رسم می‌کنیم:

```
sys_cl = feedback(C*P_cruise,1);

step(r*sys_cl);
```



مقدار خطای حالت ماندگار به حد مطلوب رسیده است. هرچند که زمان نمو بسیار کمتر از مقدار خواسته شده و در این مثال غیر معقول می‌باشد زیرا خودرو نمی‌تواند در ۲ ثانیه به سرعت 10 m/s برسد. در نتیجه از یک بهره‌ی تناسبی کوچک‌تر به همراه یک جبران‌ساز پس‌فاز استفاده می‌کنیم.

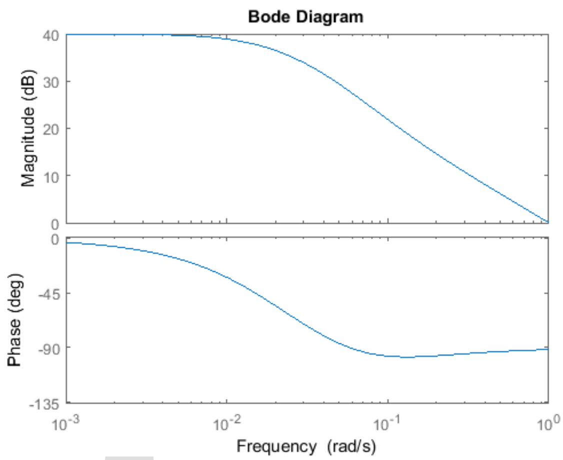
جبران‌ساز پس‌فاز

در **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز** گفته شد که جبران‌ساز پس‌فاز بهره‌ی فرکانس پایین را اضافه می‌کند اما فرکانس پهنای باند را ثابت نگه می‌دارد. در این مسئله دقیقاً به جبران‌ساز پس‌فاز نیاز داریم زیرا بهره‌ی فرکانس پایین بزرگ‌تر، خطای حالت ماندگار کمتری را نتیجه داده اما فرکانس پهنای باند ثابت می‌ماند که سبب ثابت ماندن زمان نمو می‌شود. تابع تبدیل کنترلر پس‌فاز به شکل زیر است:

$$C_{\text{پس‌فاز}}(s) = \frac{s + z_0}{s + p_0} \qquad (۴)$$

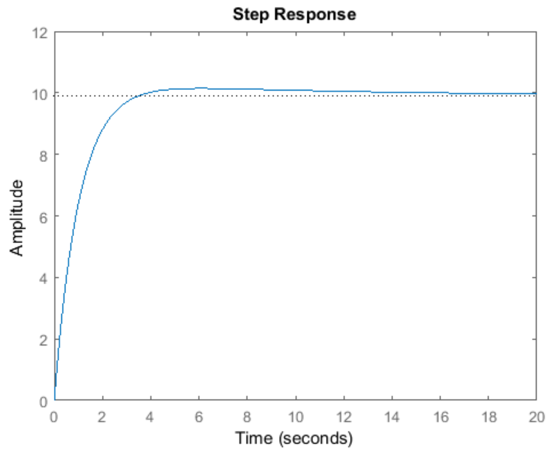
در **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز** گفته شد که در کنترلر پس‌فاز، قطب و صفر باید در نزدیکی هم قرار بگیرند و همچنین در این صورت خطای حالت ماندگار به نسبت z_0/p_0 کاهش می‌یابد. به همین دلیل مقدار z_0 را برابر ۰/۱ و مقدار p_0 را برابر ۰/۰۲ قرار می‌دهیم. بهره‌ی تناسبی $K_p = 1000$ به وسیله‌ی سعی و خطا انتخاب شده است:

```
Kp = 1000;
zo = 0.1;
po = 0.02;

C_lag = (s+zo)/(s+po);

bode(Kp*C_lag*P_cruise);
```



برای اطمینان از عملکرد سیستم، پاسخ پله‌ی حلقه بسته را رسم می‌کنیم:

```
sys_cl = feedback(Kp*C_lag*P_cruise,1);

t = 0:0.1:20;

step(r*sys_cl,t);
```



همانطور که مشاهده می‌کنید مقدار کمی فراجهش وجود داشته، خطای حالت ماندگار نزدیک صفر و زمان نمو کمتر از ۵ ثانیه است. حال سیستم ما تمامی خواسته‌های طراحی را دارا می‌باشد و نیازی و اقدامات بیشتر نمی‌باشد.

بخش ششم: طراحی کنترلر در فضای حالت

فهرست مطالب بخش

در این بخش با استفاده از مدل فضای حالت به طراحی کنترلر و مشاهده‌گر برای سیستم کنترل کروز می‌پردازیم.

دستورهای کلیدی متلب در این بخش:

```
ss, feedback
```

معادلات فضای حالت

معادلات حرکت به فرم فضای حالت به شرح زیر است:

$$[\dot{v}] = \left[\frac{-b}{m}\right][v] + \left[\frac{1}{m}\right][u] \qquad (۱)$$

$$y = [1][v] \qquad (۲)$$

که در آن:

(m) جرم خودرو ۱۰۰۰ کیلوگرم

(b) ضریب میرایی 50 N.s/m

(u) نیروی کنترل نامی ۵۰۰ نیوتن

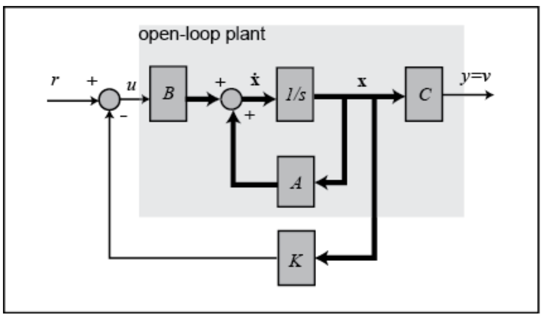
(v) سرعت خودرو که خروجی سیستم است عبارتست از y = v

الزامات طراحی

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

طراحی کنترل با استفاده از جایدهی قطب

شماتیک سیستم با فیدبک تمام حالات به شکل زیر است:



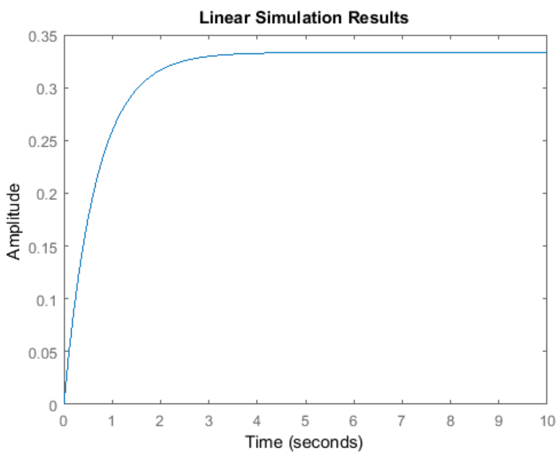
در این شماتیک، K ماتریس بهره فیدبک حالت‌ها و u ورودی کنترلی که برابر $u = r - kx = r - kv$ است می‌باشد.

در **فصل دوم - بخش ششم: مقدمه‌ای بر طراحی کنترلر در فضای حالت،** گفته شد که می‌توان از تکنیک جایدهی قطب برای به دست آوردن خروجی مطلوب استفاده نمود. قطب‌های سیستم حلقه بسته را می‌توان از معادله‌ی مشخصه به دست آورد که در فرم فضای حالت، معادله‌ی مشخصه برابر دترمینان ماتریس $[sI - (A - B \times K)]$ می‌باشد. اگر بتوان قطب‌های سیستم را با ماتریس کنترل مناسب K در موقعیت مطلوب قرار داد پس می‌توان خروجی مطلوب را به دست آورد. در این بخش ابتدا قطب‌ها انتخاب شده و با استفاده از متلب، ماتریس کنترل مربوطه‌ی K را پیدا می‌کنیم.

حال باید موقعیت قطب‌ها را مشخص کنیم. چون ماتریس $[sI - (A - B \times K)]$ ماتریس 1x1 می‌باشد، تنها لازم است تا یک قطب را جایدهی کنیم. فرض کنید می‌خواهیم قطب در 1.5- (موقعیت دلخواه) باشد. همانطور که در فصل اول گفته شد، از دستور place در متلب برای به دست آوردن ماتریس کنترل K استفاده می‌کنیم. یک ام‌فایل جدید ساخته و دستورات زیر را در آن وارد کنید. با اجرای این ام‌فایل، ماتریس کنترل و پاسخ پله نمایش داده می‌شود.

```
m = 1000;
b = 50;
t = 0:0.1:10;
u = 500*ones(size(t));

A = [-b/m];
B = [1/m];
C = [1];
D = [0];

sys = ss(A,B,C,D);

x0 = [0];

p1 = -1.5;

K = place(A,B,[p1])

sys_cl = ss(A-B*K,B,C,D);
lsim(sys_cl,u,t,x0);
axis([0 10 0 0.35])
```

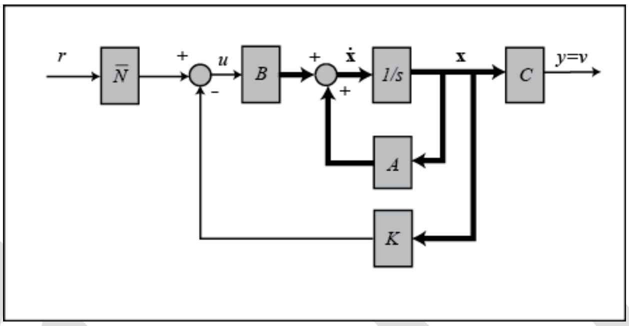
```
K =

        1450
```



از نمودار مشاهده می‌کنیم که زمان نمو رضایت‌بخش اما خطای حالت ماندگار بسیار زیاد است.

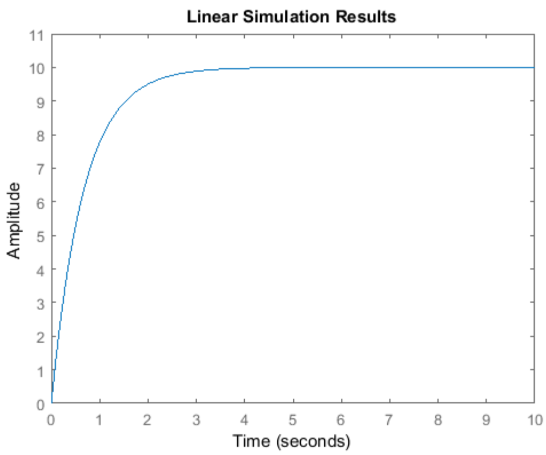
ورودی مرجع

در **فصل اول – بخش ششم: فضای حالت**، ضریب بزرگنمایی Nbar (در شماتیک زیر مشخص شده است) معرفی شد که می‌توان از آن برای حذف خطای حالت ماندگار استفاده کرد. از دستور `rscale` برای محاسبه‌ی ضریب بزرگنمایی استفاده می‌نماییم. از ام‌فایل rscale.m موجود در سی‌دی استفاده کنید. در حال حاضر ورودی در عدد ۵۰۰ ضرب شده و هدف ما سرعت حالت ماندگار 10 m/s می‌باشد.



کدهای زیر را در یک ام‌فایل وارد کرده و آنرا اجرا کنید. به این ترتیب پاسخ پله نمایش داده می‌شود:

```
Nbar = rscale(sys,K)*10/500;

sys_cl = ss(A-B*K,B*Nbar,C,D);

lsim(sys_cl,u,t,x0);

axis([0 10 0 11])
```



از پاسخ پله‌ی مشاهده شده می‌توان دید که خطای حالت ماندگار از بین رفته است. زمان نمو کمتر از ۵ ثانیه و فراجهش صفر می‌باشد. تمامی نیازهای طراحی برآورده شده است.

بخش هفتم: طراحی کنترلر دیجیتال

فهرست مطالب بخش

در این بخش قصد داریم از روش مکان هندسی ریشه‌ها برای طراحی کنترلر دیجیتال استفاده کنیم.

دستورهای کلیدی متلب در این بخش:

```
tf, c2d, rlocus, zgrid, feedback, step
```

مدل سیستم

مدل تابع تبدیل برای مسئله‌ی کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \tag{۱}$$

پارامترهای سیستم

پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

تابع تبدیل گسسته

اولین قدم در تحلیل سیستم به طور گسسته، به دست آوردن تابع تبدیل گسسته‌ی سیستم پیوسته است. با استفاده از دستور c2d، تابع تبدیل بالا ((Y(s)/U(s)) را به تابع تبدیل گسسته تبدیل می‌نماییم. برای استفاده از این تابع، نیاز به سه آرگومان یا ورودی داریم: سیستم، زمان نمونه‌برداری (Ts) و روش ('method'). زمان نمونه‌برداری (Ts) که بر حسب sec/sample بیان می‌شود، باید کمتر از $1/(30BW)$ باشد (BW فرکانس پهنای باند حلقه بسته است). برای آرگومان روش، از روش نگهدارنده مرتبه صفر ('zoh') استفاده می‌کنیم. زمان نمونه برداری را 1/50 ثانیه در نظر می‌گیریم. با توجه به فرکانس پهنای باند 1 rad/sec، این مقدار به اندازه‌ی کافی سریع می‌باشد. حال کدهای زیر را در یک ام‌فایل وارد کرده و آنرا اجرا کنید:

```
m = 1000;
b = 50;
u = 500;

s = tf('s');
P_cruise = 1/(m*s+b);

Ts = 1/50;

dP_cruise = c2d(P_cruise,Ts,'zoh')
```

```
dP_cruise =

   1.999e-05

  ---------

  z - 0.999

Sample time: 0.02 seconds

Discrete-time transfer function.
```

مکان هندسی ریشه‌ها در صفحه z

در **فصل دوم - بخش هفتم: مقدمه‌ای بر طراحی کنترلر دیجیتال**، گفته شد که از تابع `zgrid` برای به دست آوردن نواحی قابل قبول مکان هندسی ریشه‌ها در حالت گسسته استفاده می‌شود که از آن می‌توان مقدار بهره‌ی مطلوب K را به دست آورد. دستور `zgrid` نیاز به دو آرگومان دارد: فرکانس طبیعی (ω_n) و ضریب میرایی (ζ). این دو آرگومان را می‌توان از زمان نمو و فراجهش مطلوب سیستم، با معادلات زیر به دست آورد.

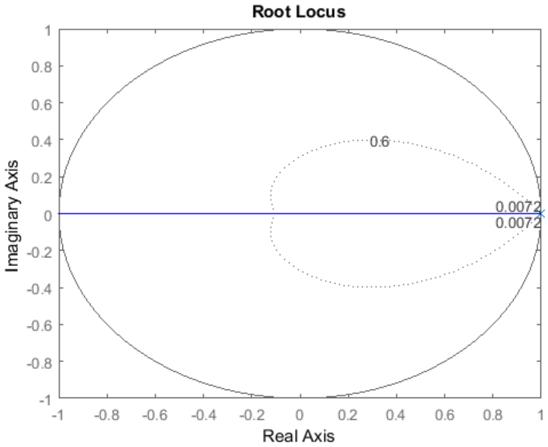
$$\omega_n \geq \frac{1.8}{T_r} \qquad (۲)$$

$$\zeta \geq \sqrt{\frac{\ln^2(M_p)}{\pi^2 + \ln^2(M_p)}} \qquad (۳)$$

زمان نمو مطلوب ۵ ثانیه و فراجهش مطلوب ۱۰٪ است، در این صورت باید فرکانس طبیعی بزرگتر از 0.36 rad/sec و ضریب میرایی بزرگتر از 0.6 باشد.

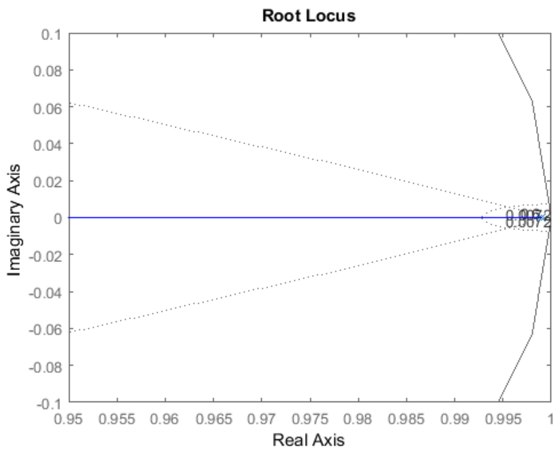
مکان هندسی ریشه‌ها را رسم کرده و با استفاده از دستور `zgrid` نواحی قابل قبول را مشخص می‌کنیم. اما پیش از آن، همانطور که گفته شده بود، در دستور `zgrid` فرکانس طبیعی باید بر واحد rad/sample باشد پس با تبدیل واحد، فرکانس طبیعی برابر $\omega_n = 0.36T_s = 0.0072$ می‌باشد. حال دستورات زیر را در ام‌فایل وارد و آنرا دوباره اجرا کنید تا نمودار زیر را به دست آورید.

```
Wn = 0.0072;
zeta = 0.6;

rlocus(dP_cruise)

zgrid(zeta, Wn)
axis ([-1 1 -1 1])
```



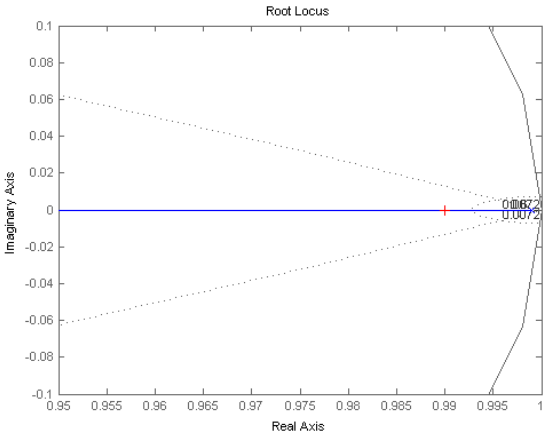
نواحی قابل قبول صفحه‌ی مختلط در نزدیکی نقطه (۱،۰) می‌باشد. برای بررسی بهتر، این نقطه را بزرگنمایی می‌کنیم. دستور زیر را در انتهای ام‌فایل وارد کرده و آنرا دوباره اجرا کنید:

```
axis ([0.95 1 -.1 .1])
```



خطوط خط‌چین در سمت راست، نقاط فرکانس طبیعی ثابت را مشخص می‌کند، در خارج از این دو خط‌چین فرکانس طبیعی بزرگتر از ۰/۰۰۷۲ می‌باشد. دیگر خطوط خط‌چین نشان‌دهنده نقاط ضریب میرایی ثابت می‌باشند که در داخل آنها، ضریب میرایی بزرگتر از ۰/۶ است. خط‌های توپر عمودی، در واقع بخشی از دایره واحد می‌باشند که به علت بزرگنمایی، به صورت شکسته نمایش داده شده‌اند.

در نمودار بالا، بخشی از مکان هندسی ریشه‌ها که در ناحیه‌ی مطلوب قرار دارد را مشاهده می‌کنید. با استفاده از دستور `rlocfind` مقدار بهره‌ی K را به دست آورده و سپس پاسخ پله‌ی مربوطه را رسم می‌کنیم. با اجرای دستور `[K,poles] = rlocfind(dP_cruise)` در متلب، پنجره‌ای باز شده و نقطه‌ی مورد نظر از شما درخواست می‌گردد. دقت کنید که اگر قطبی را که خیلی از دایره‌ی واحد فاصله دارد انتخاب کنید، پاسخ پله‌ی سیستم بسیار سریع شده و به طور فیزیکی نامعقول می‌گردد. بنابراین باید قطبی را که در نزدیکی تقاطع خطوط فرکانس طبیعی ثابت و محور حقیقی می‌باشد را انتخاب کنید. نقطه‌ای را در نزدیکی ۰/۹۹ که در شکل زیر با علامت بعلاوه مشخص شده است، انتخاب کنید.

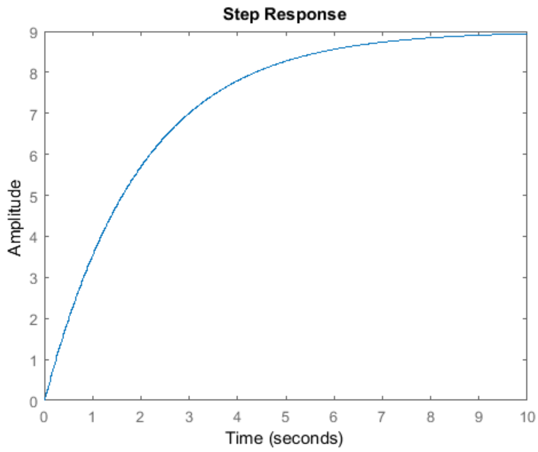


بعد از انتخاب این نقطه، خروجی زیر را در پنجره‌ی دستور متلب مشاهده می‌کنید. این خروجی به شما نقطه‌ی انتخاب شده و بهره‌ی K را اعلام می‌کند.

```
Select a point in the graphics window

        selected_point =

        0.9900 - 0.0003i

        K =

        451.1104

        poles =

        0.9900
```

سپس برای پاسخ پله‌ی حلقه بسته، کد زیر را در ام‌فایل وارد کنید:

```
K = 451.1104;

sys_cl = feedback(K*dP_cruise,1);

r = 10;

figure

step(r*sys_cl,10);
```



این پاسخ، شروط زمان نمو و فراجهش را ارضا می‌کند اما خطای حالت ماندگار حدود ۱۱٪ می‌باشد. برای به دست آوردن خطای حالت ماندگار مطلوب، در قسمت بعد کنترلر دیجیتال را اصلاح می‌کنیم.

جبران‌ساز با استفاده از کنترلر دیجیتال

با یادآوری از بخش‌های قبل، جبران‌ساز پس‌فاز برای به دست آوردن پاسخ مطلوب به سیستم اضافه می‌شد. در حالت کنترل دیجیتال برای مسئله کنترل کروز، کنترلر دیجیتال فعلی را با اضافه کردن جبران‌ساز پس‌فاز، اصلاح می‌کنیم:

$$C_{\text{پس‌فاز}}(z) = K_d \frac{z - z_0}{z - z_p} \qquad (۵)$$

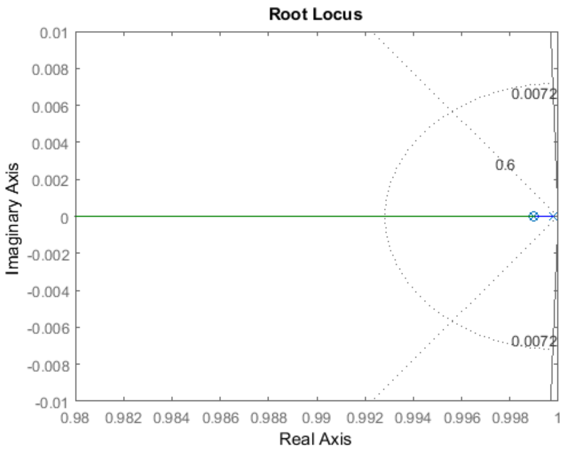
روش‌هایی برای طراحی جبران‌سازهای دیجیتال پیش‌فاز و پس‌فاز و همچنین برای طراحی جبران‌سازهای پیوسته پیش‌فاز و پس‌فاز وجود دارد. در روش طراحی گسسته، صفر جبران‌ساز پس‌فاز باید به گونه‌ای انتخاب شود تا یکی از قطب‌های سیستم را خنثی کند. البته باید دقت شود تا جبران‌ساز ناپایدار نگردد. بنابراین در اینجا صفر را در $z_0 = 0.999$ انتخاب می‌کنیم.

برای کاهش خطای حالت ماندگار، لازم به ذکر است که بهره فرکانس پایین سیستم گسسته با جبران‌ساز پس‌فاز، به نسبت $(1 - z_0)/(1 - z_p)$ افزایش می‌یابد. برای کاهش خطای حالت ماندگار با نسبت ۵، z_p را برابر ۰/۹۹۹۸ انتخاب می‌کنیم. برای داشتن بهره‌ی ۱ در فرکانس صفر، قبل از رسم مکان هندسی ریشه‌ها، صورت کسر را در $K_d = \frac{1-z_p}{1-z_0} = 0.2$ ضرب می‌کنیم. در نهایت باید کل جبران‌ساز را در بهره‌ی به دست آمده از مکان هندسی ریشه‌ها ضرب نمود.

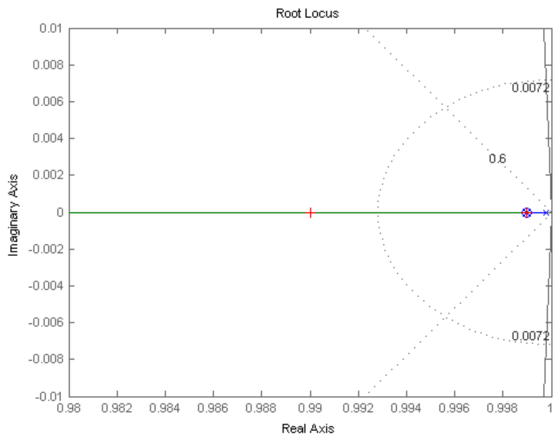
حال که تابع تبدیل جبران‌ساز گسسته را داریم، مکان هندسی ریشه‌ها را رسم کرده و پاسخ پله را به دست می‌آوریم. ابتدا یک ام‌فایل جدید ساخته و دستورات زیر را وارد کنید.

```
m = 1000;
b = 50;
u = 500;

s = tf('s');
P_cruise = 1/(m*s+b);
Ts = 1/50;
dP_cruise = c2d(P_cruise,Ts,'zoh');

z = tf('z',Ts);
C = 0.2*(z - 0.999)/(z - 0.9998);

Wn = 0.0072;
zeta = 0.6;

rlocus(C*dP_cruise)
```

```
zgrid(zeta, Wn)
axis([0.98 1 -0.01 0.01])
```



با اجرای دستور `[K,poles] = rlocfind(C*dP_cruise)` در متلب، دوباره نقطه‌ای در نزدیکی ۰/۹۹ مانند شکل زیر انتخاب کنید.

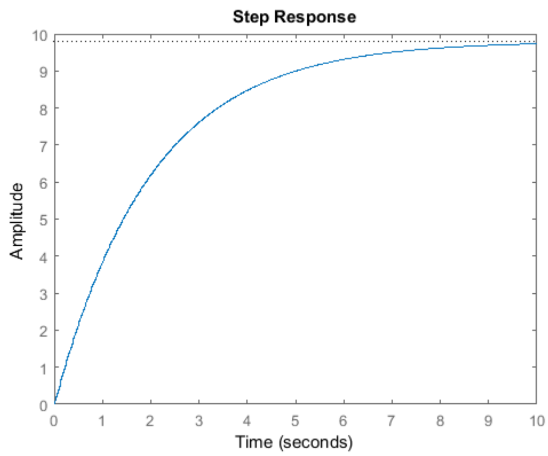


بعد از انتخاب این نقطه، خروجی زیر را در متلب مشاهده می‌کنید:

```
Select a point in the graphics window

        selected_point =

        0.9900 - 0.0000i

        K =

        2.4454e+03

        poles =

        0.9900
        0.9900
```

در نهایت برای مشاهده‌ی پاسخ پله‌ی حلقه بسته، دستور زیر را اجرا می‌کنیم:

```
K = 2.4454e+03;

sys_cl = feedback(K*C*dP_cruise,1);

r = 10;

step(r*sys_cl,10);
```



پاسخ به دست آمده تقریبا سرعتی مانند سرعت پاسخ قبل داشته اما خطای حالت ماندگار آن به ۲٪ کاهش یافته است. این سیستم تمامی نیازهای طراحی را ارضا کرده و پاسخ معقولی دارد.

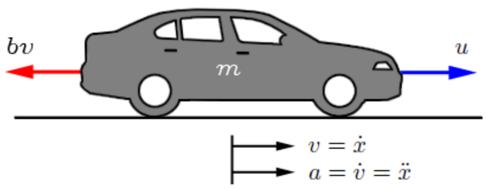
نکته: مسئله طراحی ممکن است تنها یک جواب نداشته باشد. برای تمرین، سایر جبران‌سازها را آزموده تا پاسخ بهتری را به دست آورید.

بخش هشتم: مدل‌سازی سیمولینک

فهرست مطالب بخش

سیستم فیزیکی و معادلات آن

معادلات سیستم کنترل کروز نسبتاً ساده می‌باشند. اگر مقاومت غلتشی و درگ هوا را متناسب با سرعت خودرو در نظر بگیریم، مسئله مانند یک مسئله‌ی ساده‌ی جرم و دمپر تبدیل می‌شود.



با استفاده از قانون دوم نیوتن، معادلات حاکم بر سیستم عبارتند از:

$$m\dot{v} = u - bv \qquad (۱)$$

که u نیروی ایجاد شده از تماس چرخ و زمین است که به طور مستقیم کنترل می‌شود. برای این مثال مقادیر زیر را در نظر بگیرید:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

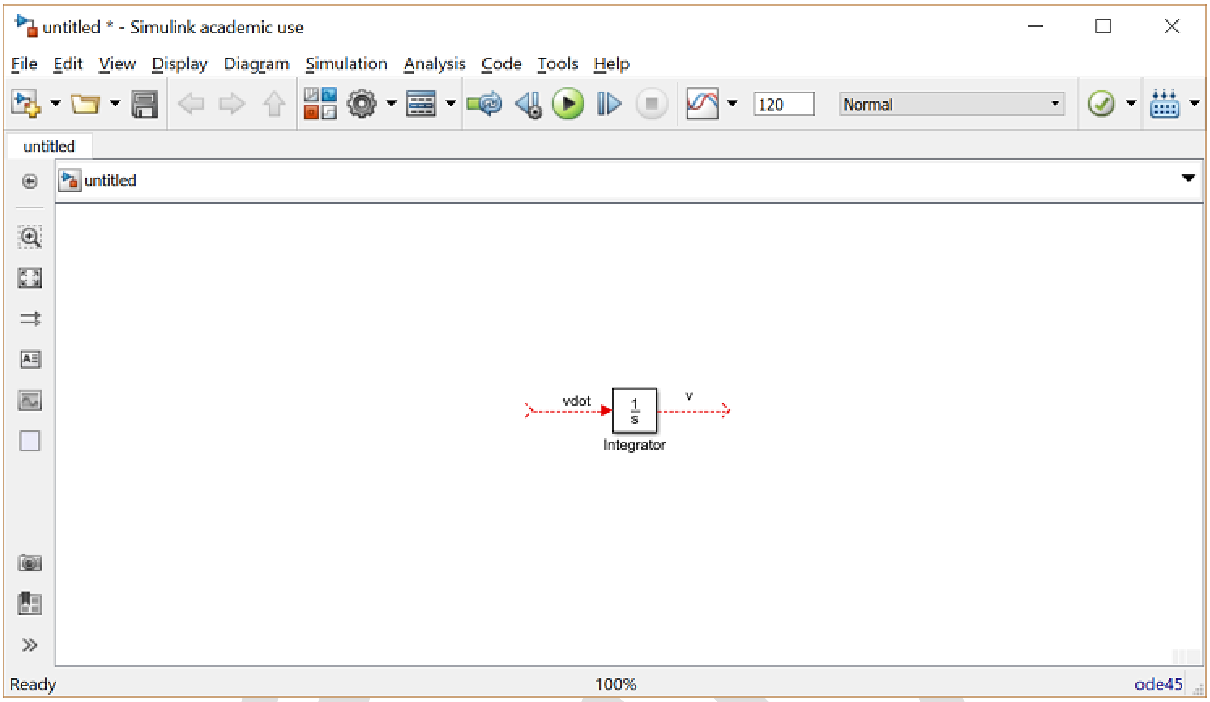
(u) نیروی کنترلی = ۵۰۰ نیوتن

ساخت مدل

مدل این سیستم از مجموع نیروهای وارد شده به جرم و انتگرال‌گیری از شتاب (که سرعت را نتیجه می‌دهد) به دست می‌آید:

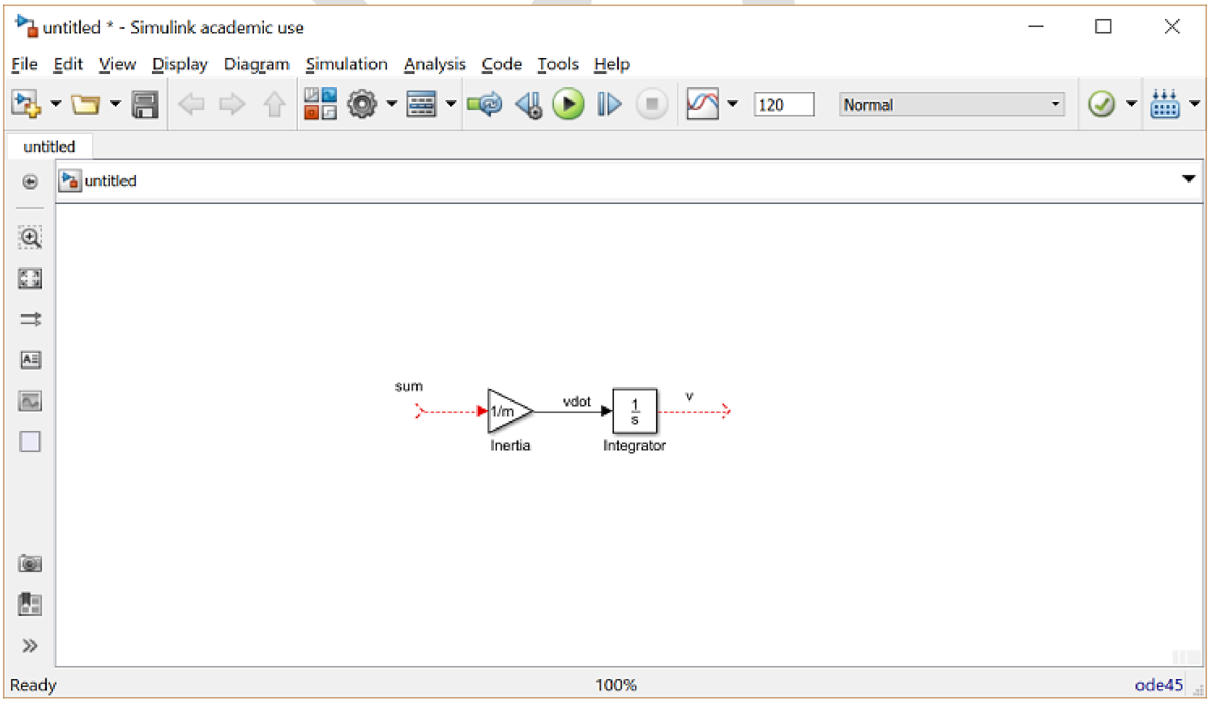
$$\int \frac{dv}{dt} dt = v \qquad (۲)$$

- یک بلوک Integrator (از کتابخانه Continuous) به مدل اضافه کرده و خطوط را به ورودی و خروجی آن متصل کنید.
- خط ورودی را "vdot" و خروجی را "v" نام‌گذاری کنید. برای نام‌گذاری، بر روی فضای بالای هر خط، دبل کلیک کنید.



به علت اینکه شتاب (dv/dt) برابر مجموع نیروها تقسیم بر جرم می‌باشد، سیگنال ورودی را بر جرم تقسیم می‌کنیم:

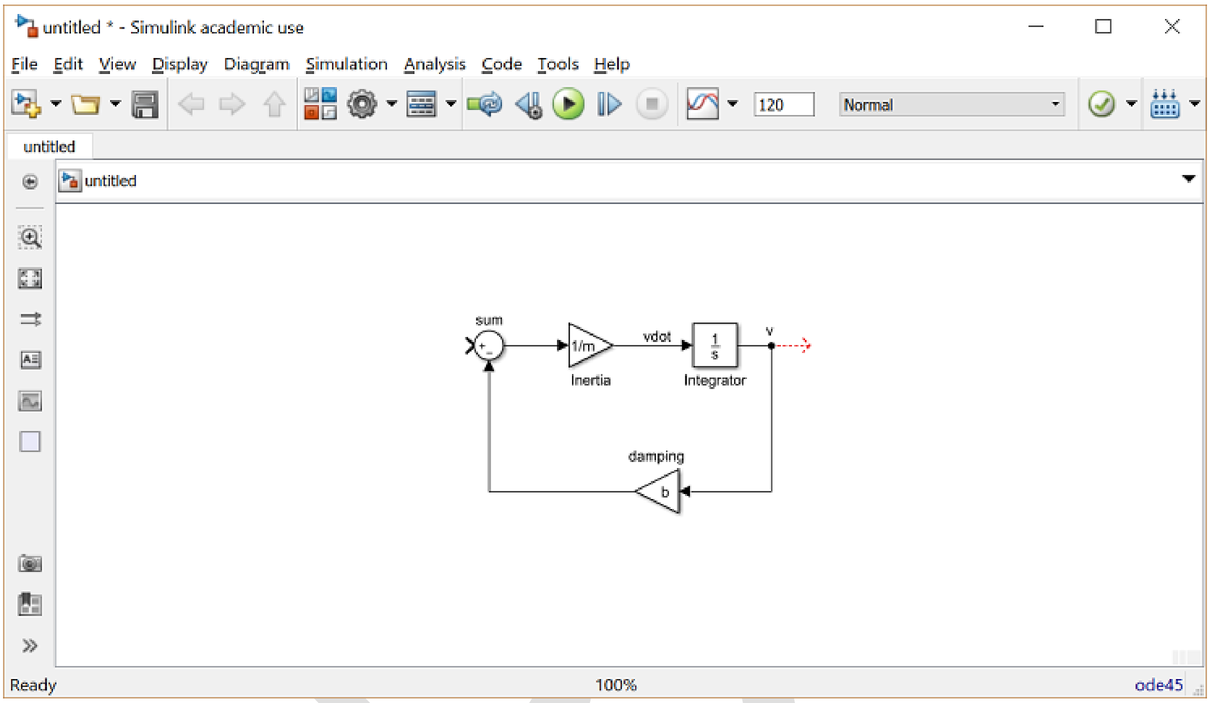
- یک بلوک Gain (از کتابخانه‌ی Math Operations) به ورودی بلوک Integrator متصل کرده و یک خط نیز به ورودی بلوک Gain متصل کنید.
- مقدار بلوک Gain را با دبل کلیک به ”1/m“ تغییر دهید.
- لیبل بلوک Gain را با کلیک بر روی کلمه ”Gain“ در پایین بلوک، به ”inertia“ تغییر دهید.



حال نوبت به اضافه کردن نیروهایی است که در معادله (۱) آورده شده است. ابتدا نیروی میرایی را اضافه می‌کنیم.

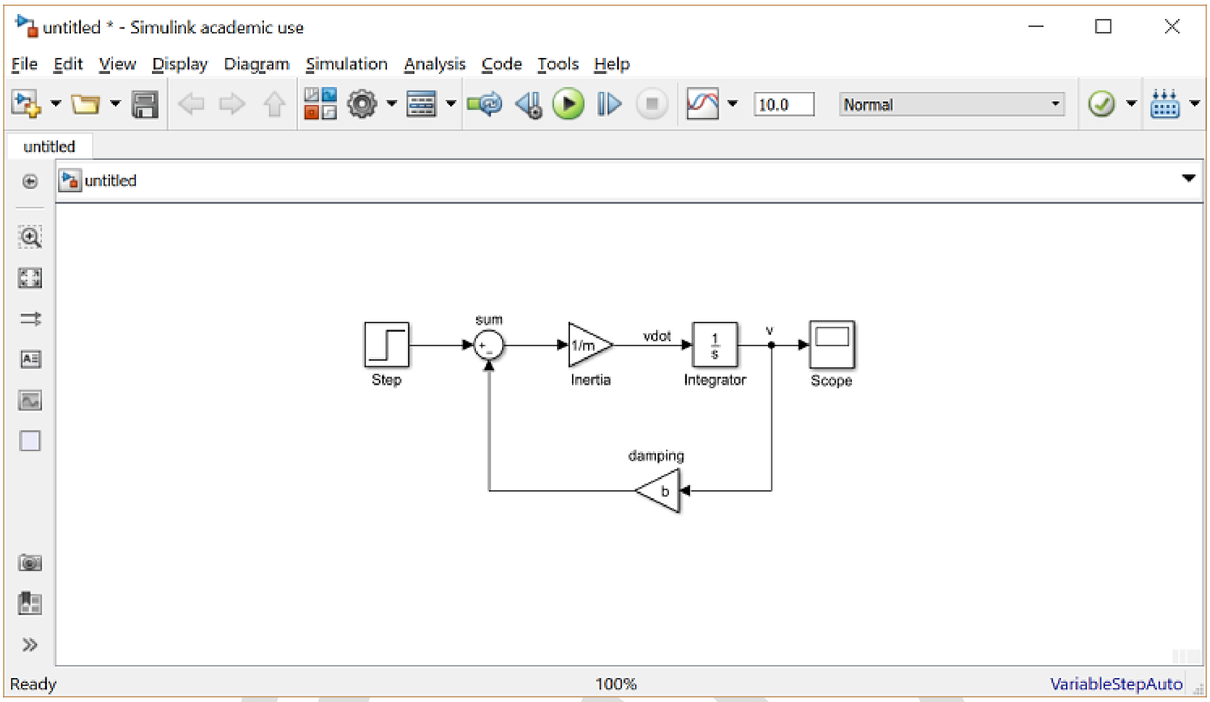
- یک بلوک Sum (از کتابخانه Math Operations) به خط ورودی به بلوک Gain اینرسی وصل کنید.
- علائم بلوک Sum را به ”+-“ تغییر دهید.

- یک بلوک Gain در زیر بلوک اینرسی قرار داده و آنرا با کلیک ماوس انتخاب کنید و از منوی **Rotate & Flip** گزینه‌ی **Flip Block** (یا کلید ترکیبی **Ctrl+I**) را انتخاب کنید تا بلوک Gain دوران پیدا کند.
- مقدار بلوک را برابر "b" قرار داده و نام آنرا به "damping" تغییر دهید.
- یک خط (با نگه داشتن کلید Ctrl) از خروجی بلوک Integrator ایجاد کرده و آنرا به ورودی بلوک بهره‌ی damping متصل کنید.
- یک خط از خروجی بلوک بهره‌ی damping به ورودی منفی بلوک Sum متصل کنید.

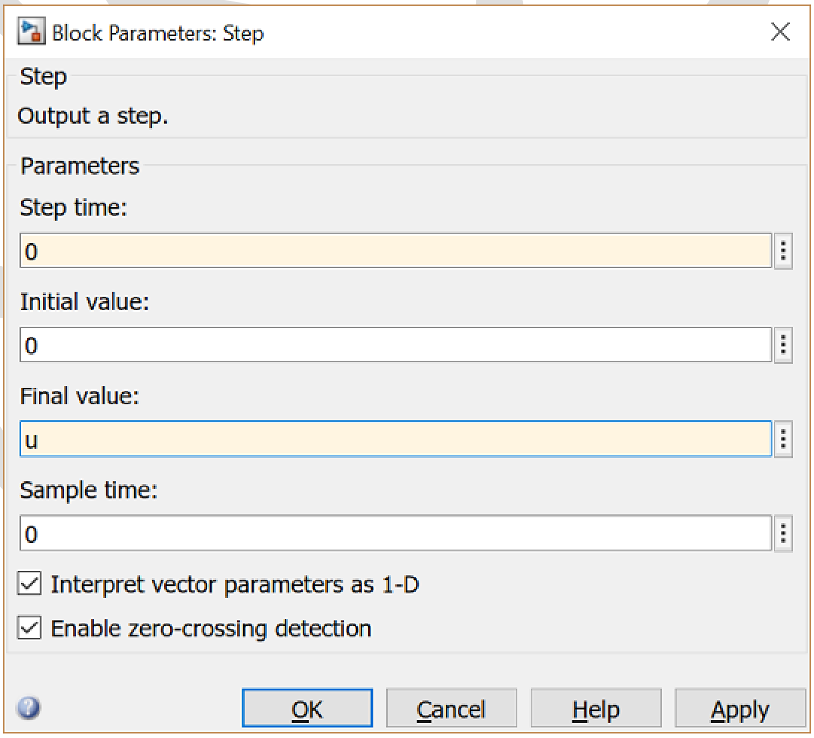


نیروی دوم وارد بر جرم، ورودی کنترل u می‌باشد. برای این نیرو از ورودی پله استفاده می‌نماییم.

- یک بلوک Step (از کتابخانه Sources) برداشته و آنرا به ورودی مثبت بلوک Sum متصل کنید.
- برای مشاهده‌ی سرعت خروجی، یک بلوک Scope (از کتابخانه Sinks) برداشته و به خروجی Integrator متصل کنید.



• برای تولید ورودی پله‌ی مناسب با مقدار ۵۰۰ در زمان صفر، بر روی بلوک Step دبل کلیک کرده و Step Time را برابر ”0“ و Final Value را برابر ”u“ قرار دهید.

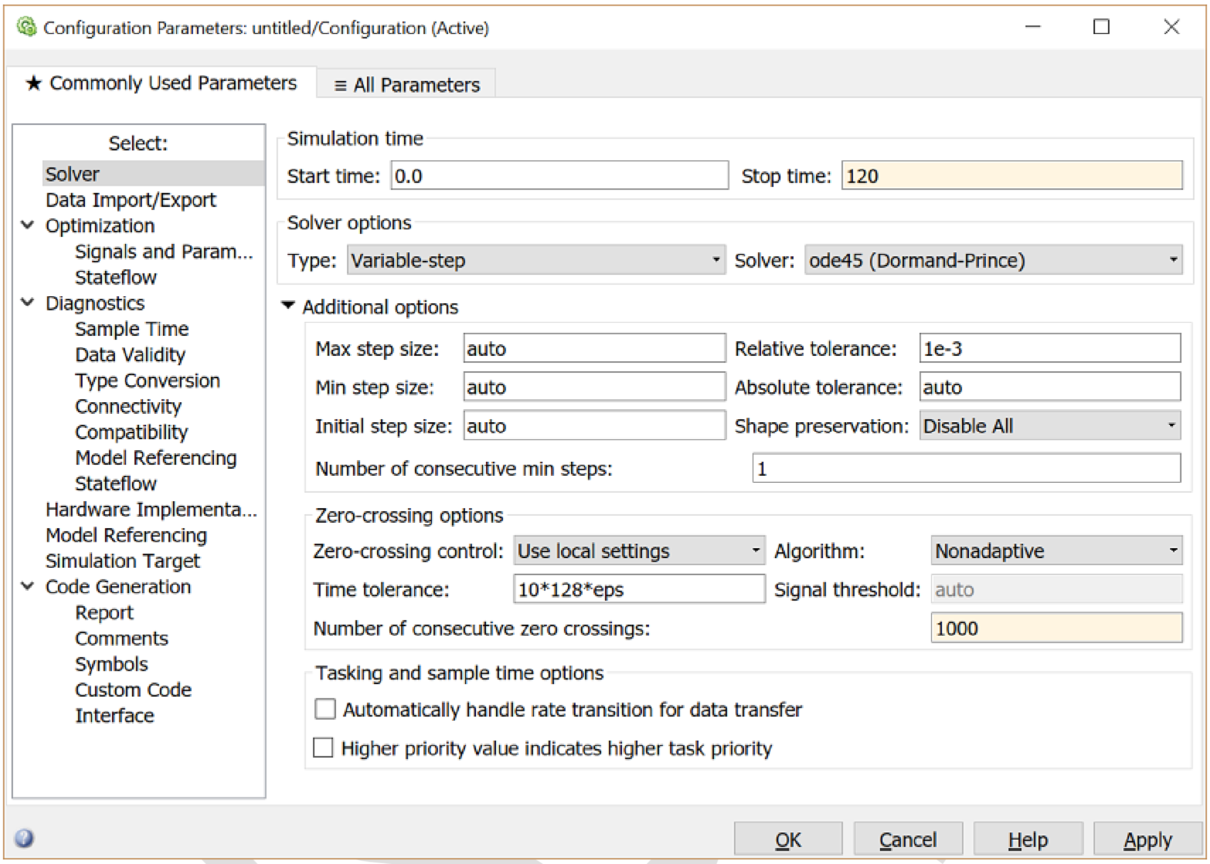


می‌توانید مدل کامل سیستم را از سی‌دی کپی کرده و آنرا بر روی کامپیوتر خود ذخیره نمایید.

پاسخ حلقه باز

برای شبیه‌سازی این سیستم، ابتدا باید زمان شبیه‌سازی مناسب را تنظیم نماییم.

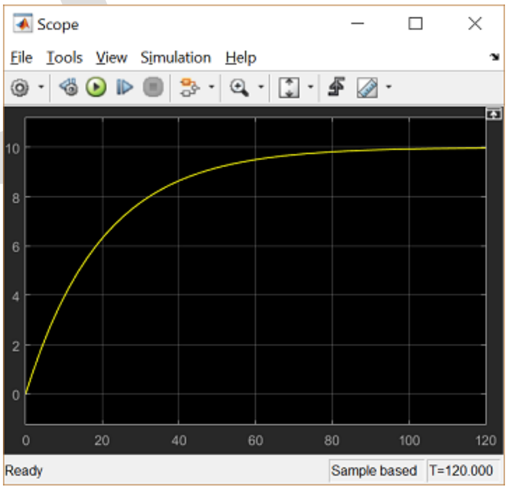
• گزینه‌ی Parameters را از منوی Simulation انتخاب کرده و در فیلد Stop Time مقدار ”120“ را وارد کنید. ۱۲۰ ثانیه زمان کافی برای مشاهده‌ی پاسخ حلقه باز می‌باشد.



حال باید پارامترهای فیزیکی سیستم را وارد نماییم. دستورات زیر را در محیط متلب وارد کنید:

```
m = 1000;
b = 50;
u = 500;
```

شبیه‌سازی را اجرا نمایید (با کلید ترکیبی **Ctrl+T** یا انتخاب **Run** از منوی **Simulation**). پس از اتمام شبیه‌سازی، خروجی زیر را مشاهده می‌نمایید:



با توجه به نمودار بالا، هدف ما بهبود پاسخ سیستم کنترل کروز می‌باشد. مدل ساخته شده در این بخش را در بخش بعد برای طراحی و تحلیل کنترلر استفاده خواهیم نمود.

بخش نهم: طراحی کنترلر در سیمولینک

فهرست مطالب بخش

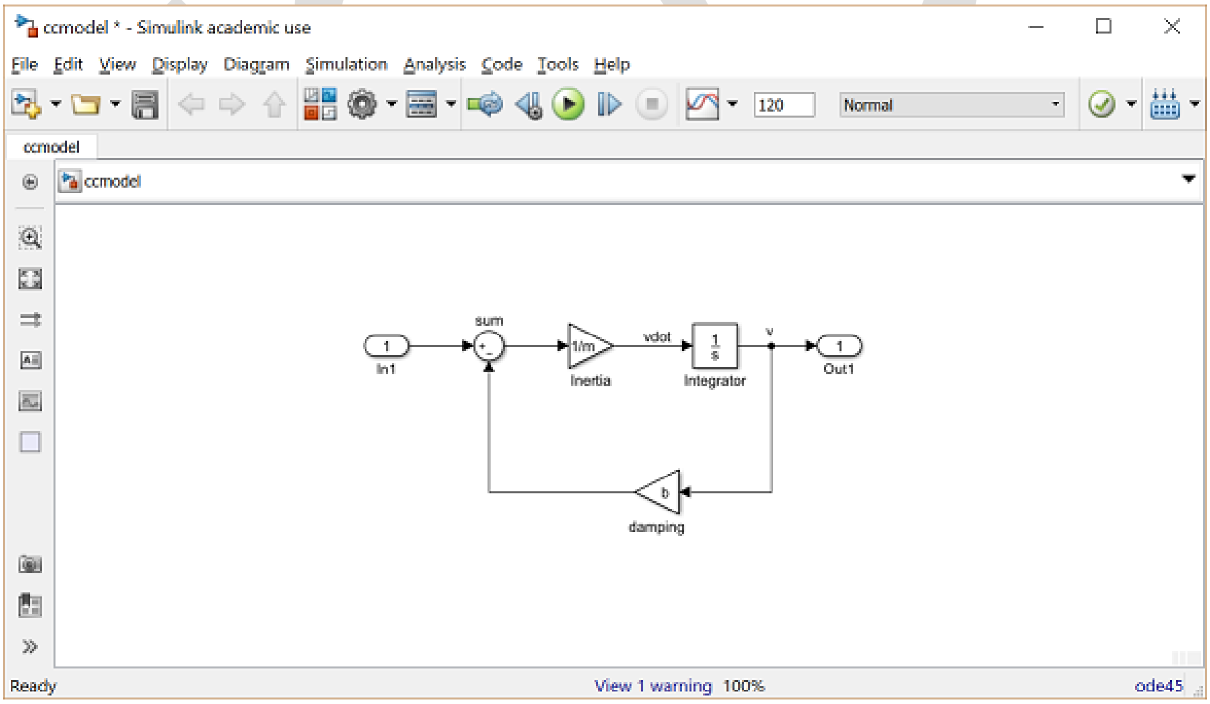
- استخراج یک مدل خطی به متلب
- پیاده‌سازی کنترل PI
- پاسخ حلقه بسته

در بخش قبل مدل سیمولینک سیستم کنترل کروز را تشکیل دادیم. این مدل را می‌توانید از سی‌دی بر روی کامپیوتر خود ذخیره نمایید. در این بخش، نشان می‌دهیم که چگونه یک کنترلر فیدبک را در سیمولینک پیاده‌سازی کرده و عملکرد سیستم را بهبود دهیم.

استخراج یک مدل خطی به متلب

سیمولینک این قابلیت را به کاربر می‌دهد تا از مدل ساخته شده در سیمولینک، یک مدل خطی (به هر دو فرم فضای حالت و تابع تبدیل) به متلب استخراج کند. برای اینکار از بلوک‌های In1 و Out1 و تابع متلب `linmod` استفاده خواهیم نمود.

- بلوک Step و Scope را به ترتیب با بلوک‌های In1 و Out1 جایگزین کنید (این بلوک‌ها در کتابخانه Ports & Subsystems می‌باشند). بدین صورت ورودی و خروجی سیستم برای فرآیند استخراج تعریف می‌شود.



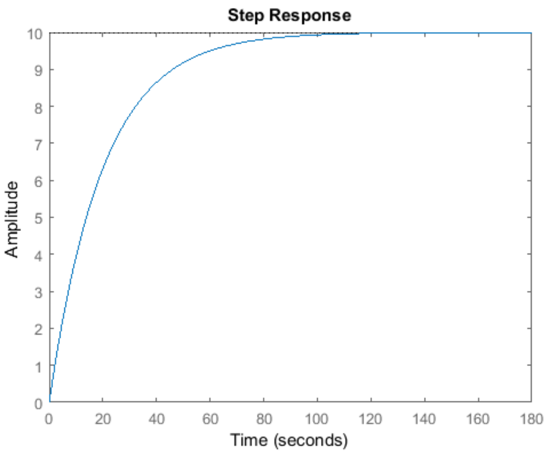
مدل خود را با نام "ccmodel.slx" ذخیره نمایید. متلب از فایل مدل ذخیره شده برای استخراج مدل خطی استفاده می‌نماید. در محیط متلب، دستورات زیر را وارد نمایید:

```
m = 1000;
b = 50;
u = 500;

[A,B,C,D] = linmod('ccmodel')
cruise_ss = ss(A,B,C,D);
```

```
A =

   -0.0500

B =

   1.0000e-03

C =

     1

D =

     0
```

برای تایید از صحت استخراج مدل، پاسخ پله حلقه باز را برای تابع تبدیل استخراج شده در متلب محاسبه می‌نماییم. صورت تابع تبدیل را در ۵۰۰ ضرب کرده تا ورودی پله ۵۰۰ نیوتن را ایجاد نماییم. دستور زیر را در متلب وارد کنید:

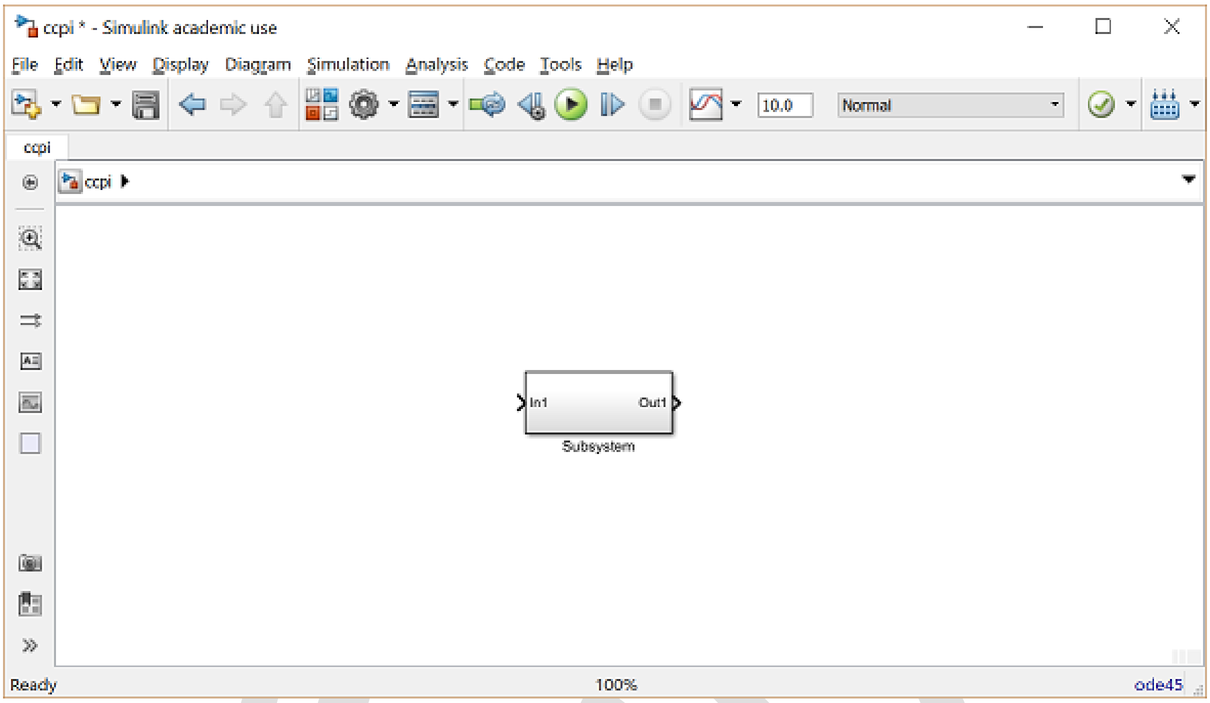
```
step(u*cruise_ss)
```



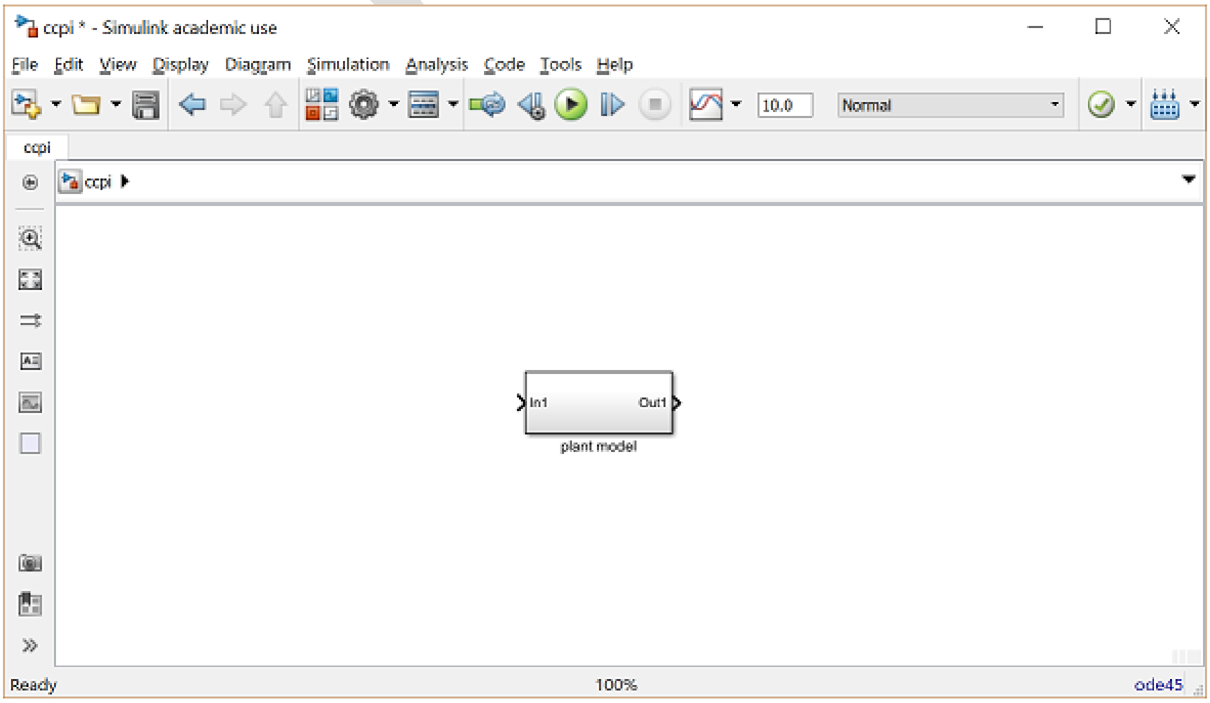
پیاده‌سازی کنترل PI

در **بخش سوم: طراحی کنترلر PID** یک کنترلر PI با ضرایب $K_p = 800$ و $K_i = 40$ برای دستیابی به پاسخ مطلوب، طراحی گردید. در این بخش این کار را در سیمولینک برای سیستم حلقه باز پیاده خواهیم کرد.

- یک پنجره‌ی مدل جدید ایجاد کنید.
- یک بلوک Subsystem از کتابخانه‌ی Ports & Subsystems در درون مدل جدید خود قرار دهید.

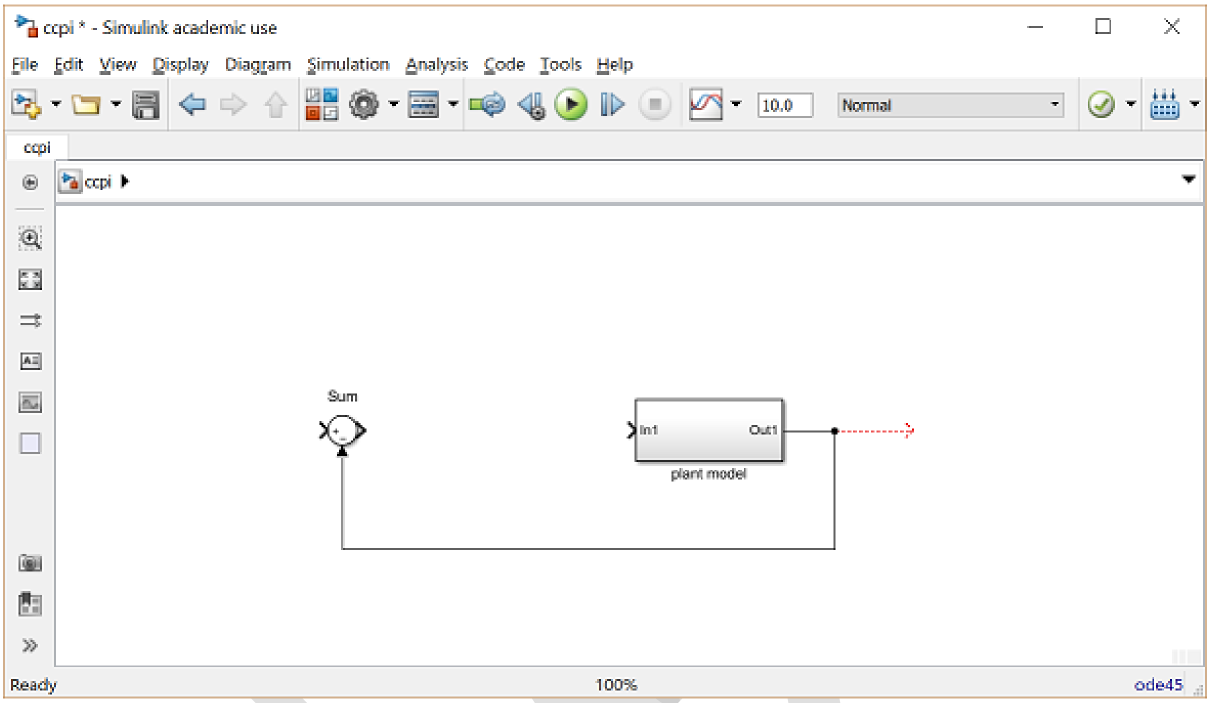


- بر روی این بلوک دبل کلیک کرده تا پنجره‌ی خالی محتویات Subsystem نمایش داده شود (که در حال حاضر خالی می‌باشد).
- مدل سیستم کنترل کروز پیشین خود که با نام ccmodel.slx ذخیره نمودید را باز نمایید.
- از منوی Edit گزینه‌ی **Select All** (یا کلید ترکیبی **Ctrl+A**) را انتخاب کرده و از منوی **Edit** گزینه‌ی **Copy** (یا کلید ترکیبی **Ctrl+C**) را انتخاب کنید.
- حال پنجره‌ی Subsystem خالی مدل جدید را انتخاب کرده و از منوی **Edit** گزینه‌ی **Paste** (یا کلید ترکیبی **Ctrl+V**) را انتخاب کنید. حال باید سیستم اصلی خود را در پنجره‌ی Subsystem جدید مشاهده کنید. این پنجره را ببندید.
- حال باید ترمینال‌های ورودی و خروجی را بر روی بلوک Subsystem مشاهده کنید. این بلوک را "plant model" نامگذاری کنید.



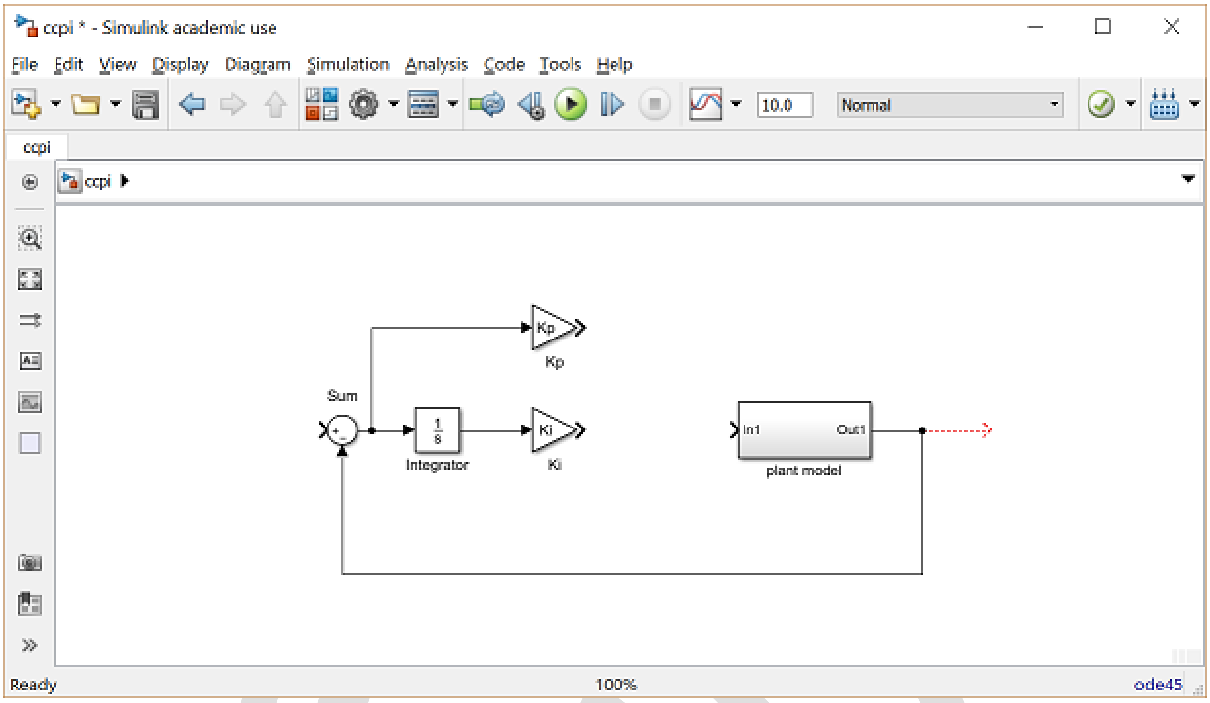
در قدم بعد به ساخت کنترلر PI برای مدل سیستم می‌پردازیم. ابتدا از خروجی سیستم فیدبک می‌گیریم:

- یک خط از خروجی سیستم ایجاد کنید.
- یک بلوک Sum با ورودی "+-" قرار دهید.
- خط ایجاد شده از خروجی سیستم را به ورودی منفی بلوک Sum متصل کنید.



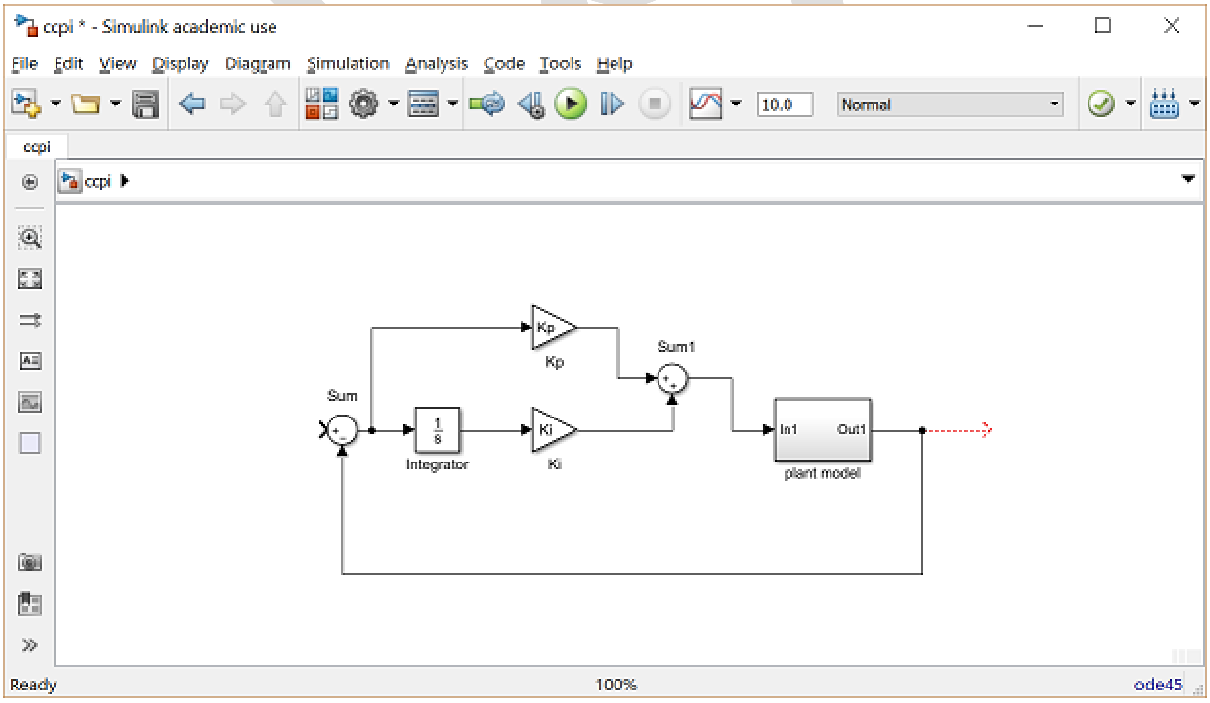
خروجی بلوک Sum، سیگنال خطا را تشکیل می‌دهد. با استفاده از این سیگنال، جملات تناسبی و انتگرالی را می‌سازیم.

- یک بلوک Integrator را بعد از بلوک Sum قرار داده و آنها را به یکدیگر متصل کنید.
- یک بلوک Gain را به عنوان بهره انتگرالی، بعد از بلوک Integrator قرار داده و آنها را به یکدیگر متصل کنید.
- بهره‌ی انتگرال‌گیر را "Ki" نامگذاری کرده و و مقدار بهره‌ی آنرا برابر "Ki" قرار دهید.
- یک بلوک بهره‌ی جدید در مدل خود قرار داده و آنرا به خط خروجی از بلوک Sum متصل کنید.
- این بلوک را "Kp"نامیده و مقدار آنرا برابر "Kp" قرار دهید.



حال جملات تناسبی و انتگرالی را به یکدیگر اضافه کرده و حاصل جمع آنها را به سیستم اعمال می‌کنیم.

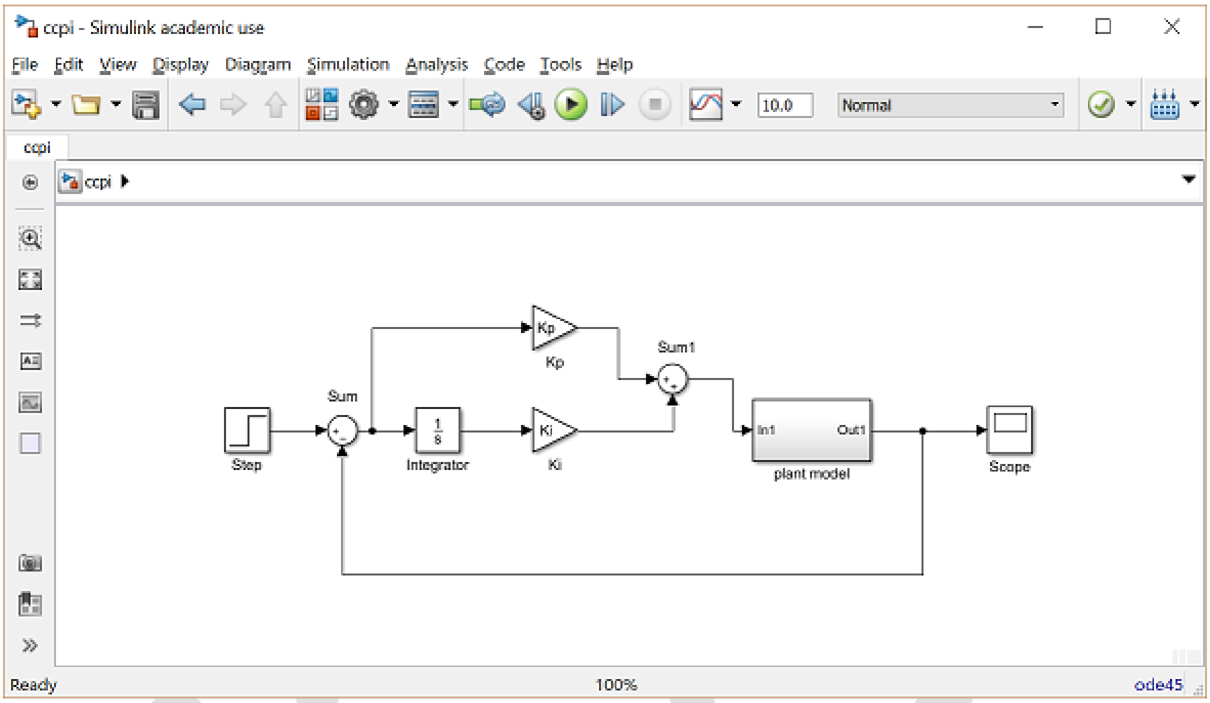
- یک بلوک Sum بین بلوک Ki و بلوک سیستم قرار داده و دو خروجی بلوک‌های Gain را به ورودی‌های بلوک Sum متصل کنید.
- خروجی بلوک Sum را به ورودی بلوک سیستم متصل کنید.



در نهایت ورودی پله را به سیستم اعمال کرده و خروجی را در بلوک Scope مشاهده می‌نماییم.

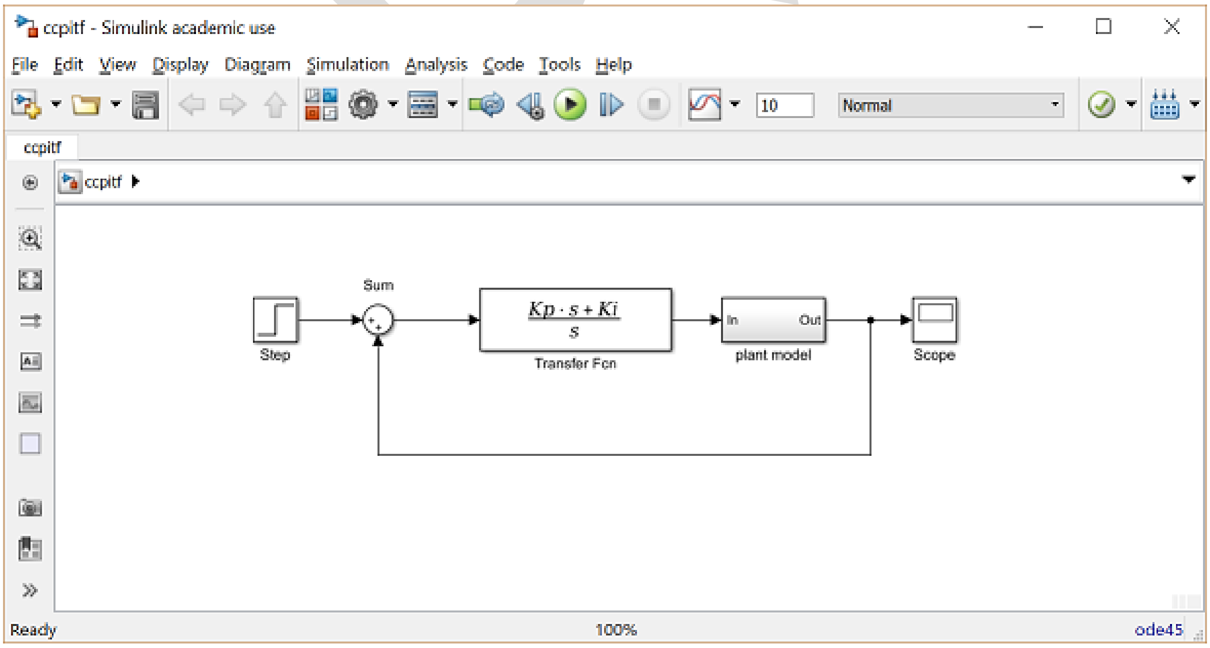
- یک بلوک Step را به ورودی آزاد بلوک Sum فیدبک متصل کنید.
- یک بلوک Scope را به خروجی سیستم متصل کنید.

- بر روی بلوک Step دبل کلیک کرده و Step Time را برابر "0" و Final Value را برابر "u" قرار دهید. با این کار امکان تغییر اندازه‌ی پله از خارج از سیمولینک را داریم.



می‌توانید مدل سیستم حلقه بسته را از داخل سی‌دی کپی نمایید.

در این مثال یک کنترلر PI را با استفاده از بلوک‌های پایه ساختیم. روش دیگری که می‌توان استفاده کرد، استفاده از بلوک Transfer Fcn (از کتابخانه Continuous) برای ساخت کنترلر PI در یک قدم مانند شکل زیر می‌باشد:



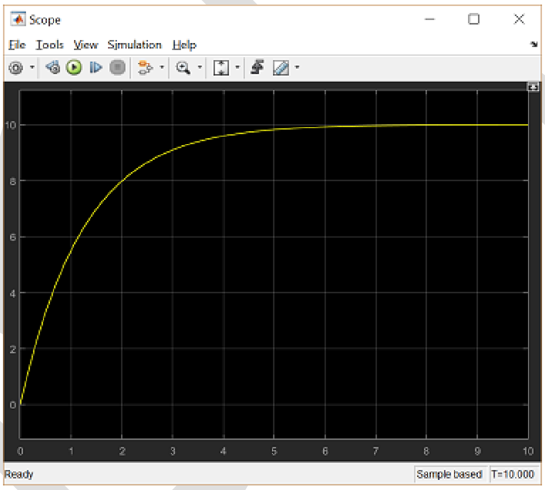
می‌توانید به این مدل نیز از داخل سی‌دی دسترسی داشته باشید.

پاسخ حلقه بسته

برای شبیه‌سازی این سیستم ابتدا باید زمان شبیه‌سازی مناسب تعیین شود. از منوی **Simulation** گزینه‌ی **Parameters** را انتخاب نمایید و در فیلد Stop Time مقدار "10" را وارد کنید. نیاز طراحی عبارتست از زمان نمو کمتر از ۵ ثانیه، پس می‌توان برای ۱۰ ثانیه شبیه‌سازی را انجام داده و خروجی را مشاهده کرد. پیش از آن باید پارامترهای فیزیکی را تعیین کنیم. دستورات زیر را در محیط متلب وارد کنید:

```
m = 1000;
b = 50;
r = 10;
Kp = 800;
Ki = 40;
```

شبیه‌سازی را اجرا کنید (با کلید ترکیبی **Ctrl+T** یا انتخاب **Run** از منوی **Simulation**). پس از اتمام شبیه‌سازی، خروجی زیر را مشاهده می‌کنیم:



فصل چهارم: کنترل کروز خودرو

بخش اول: مدل‌سازی سیستم

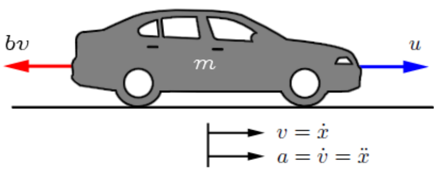
فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
ss, tf
```

سیستم فیزیکی

سیستم کنترل سرعت اتوماتیک خودرو (کروز) یک مثال بارز از سیستم کنترل فیدبک می‌باشد که بر روی بسیاری از خودروها استفاده می‌شود. هدف سیستم کنترل کروز، ثابت نگه داشتن سرعت خودرو فارغ از اغتشاشات خارجی وارد شده مانند تغییر شدت و جهت باد و شیب جاده می‌باشد. با اندازه‌گیری سرعت خودرو و مقایسه‌ی آن با سرعت مطلوب یا مرجع، می‌توان به طور خودکار شدت گاز خودرو را با توجه به قوانین کنترل، تنظیم نمود.



در اینجا یک مدل ساده‌ی دینامیکی خودرو را در نظر می‌گیریم که دیاگرام جسم آزاد آن را در تصویر بالا مشاهده می‌کنید. خودرو با جرم m، تحت نیروی کنترلی u قرار دارد. نیروی u بیانگر نیروی تولید شده توسط اتصال چرخ با زمین می‌باشد. برای این مدل ساده، فرض می‌کنیم مستقیما می‌توانیم این نیرو را کنترل کنیم و از دینامیک سیستم انتقال قدرت خودرو، لاستیک‌ها و ... صرف نظر می‌کنیم. نیروی مقاومت bv ناشی از مقاومت غلتشی و نیروی پسای[۲۸] باد می‌باشد. این نیرو خطی و متناسب با سرعت خودرو و در جهت خلاف حرکت خودرو در نظر گرفته می‌شود.

معادلات سیستم

با این فرضیات، سیستم ما مانند یک سیستم جرم و دمپر مرتبه اول می‌باشد. با استفاده از قانون دوم نیوتن در راستای x، معادلات سیستم به دست می‌آید:

$$m\dot{v} + bv = u \qquad (۱)$$

چون می‌خواهیم سرعت خودرو را کنترل کنیم، خروجی معادله را سرعت خودرو در نظر می‌گیریم یعنی:

$$y = v \qquad (۲)$$

پارامترهای سیستم

[۲۸] Drag

برای این مثال، فرض کنید که پارامترهای سیستم به شکل زیر است:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

مدل فضای حالت

سیستم‌های مرتبه اول تنها دارای یک منبع ذخیره انرژی می‌باشد که در این مثال انرژی جنبشی خودرو این نقش را بازی می‌کند. برای این سیستم‌ها تنها یک متغیر حالت نیاز می‌باشد. در نتیجه نمایش فضای حالت به شکل زیر می‌باشد:

$$\dot{\mathbf{x}} = [\dot{v}] = \left[\frac{-b}{m}\right][v] + \left[\frac{1}{m}\right][u] \tag{۳}$$

$$y = [1][v] \tag{۴}$$

این معادلات فضای حالت را با دستورات زیر در متلب وارد می‌کنیم:

```
m = 1000;
b = 50;

A = -b/m;
B = 1/m;
C = 1;
D = 0;

cruise_ss = ss(A,B,C,D);
```

مدل تابع تبدیل

با گرفتن تبدیل لاپلاس از معادلات دیفرانسیل حاکم بر سیستم و در نظر گرفتن شرایط اولیه صفر، تابع تبدیل برای سیستم کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \qquad \left[\frac{m/s}{N}\right] \tag{۵}$$

با اجرای دستور زیر، این تابع تبدیل وارد متلب می‌گردد:

```
s = tf('s');
P_cruise = 1/(m*s+b);
```

بخش دوم: تحلیل سیستم

فهرست مطالب بخش

- مدل سیستم و پارامترها
- ویژگی‌های عملکرد
- پاسخ پله‌ی حلقه باز
- قطب‌ها/صفرهای حلقه باز
- دیاگرام بودی حلقه باز

دستورهای کلیدی متلب در این بخش:

```
ss, step
```

مدل سیستم و پارامترها

مدل تابع تبدیل برای سیستم کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \qquad (۱)$$

پارامترهای استفاده شده در این مثال عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

ویژگی‌های عملکرد

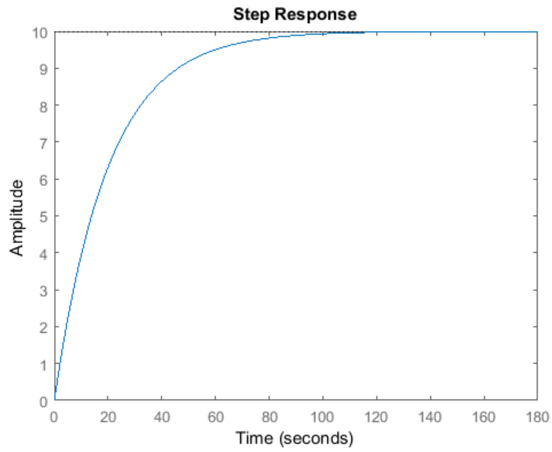
برای قدم بعد باید نیازهای طراحی را در نظر بگیریم که سیستم کنترل شده بتواند آنرا ارضا کند. وقتی که موتور نیروی ۵۰۰ نیوتن را تولید می‌کند، خودرو به حداکثر سرعت 10 m/s می‌رسد که در نمودار پاسخ پله‌ی حلقه باز می‌توان مشاهده نمود. خودرو باید بتواند در کمتر از ۵ ثانیه به این سرعت برسد. با در نظر گرفتن شرط ذکر شده، نیازهای طراحی این مسئله را به شکل زیر عنوان می‌کنیم:

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

پاسخ پله‌ی حلقه باز

پاسخ پله به نیروی ۵۰۰ نیوتنی بدون هیچگونه کنترل فیدبک، با استفاده از اجرای دستورات زیر به دست می‌آید:

```
m = 1000;
b = 50;
u = 500;


s = tf('s');
P_cruise = 1/(m*s+b);

step(u*P_cruise)
```

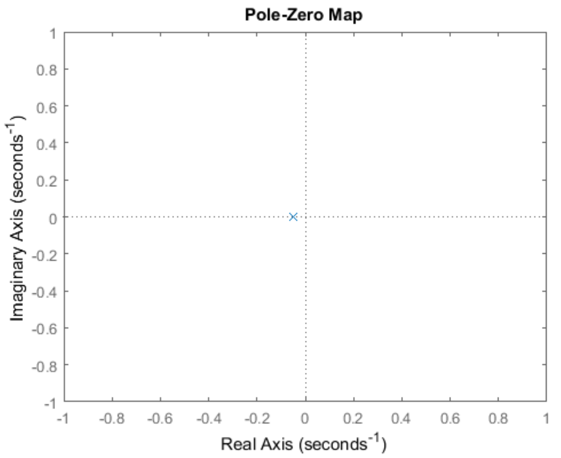


مشاهده می‌شود که سیستم حلقه باز هیچگونه فراجهش یا نوسانی ندارد (از ویژگی‌های سیستم‌های مرتبه اول) و در حالت ماندگار به سرعت مطلوب 10 m/s نمی‌رسد. همچنین زمان نمو بسیار آهسته (تقریبا برابر ۶۰ ثانیه) است. بنابراین باید کنترلر فیدبکی طراحی کنیم تا سریعا سرعت را بالا برده و بر روی سایر فاکتورهای عملکرد سیستم نیز تاثیر منفی نگذارد.

قطب‌ها/صفرهای حلقه باز

سیستم کنترل کروز دارای یک قطب در s=-b/m می‌باشد که می‌توان با دستور زیر، آنها را در صفحه‌ی مختلط s نمایش داد:

```
pzmap(P_cruise)

axis([-1 1 -1 1])
```

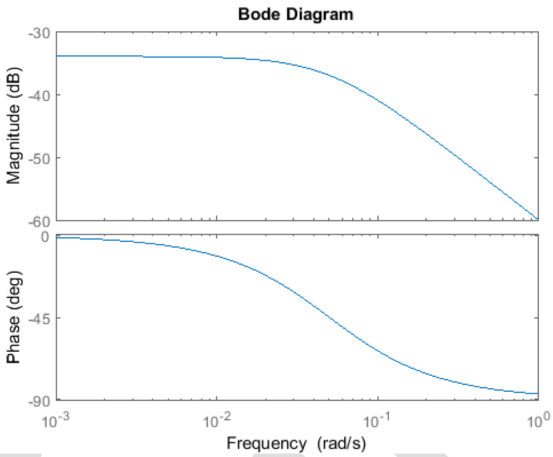


مشاهده می‌کنیم که سیستم حلقه باز پایدار بوده و به دلیل اینکه قطب سیستم، حقیق و منفی می‌باشد نوسانی در پاسخ وجود ندارد. علاوه بر آن سرعت پاسخ سیستم توسط اندازه‌ی قطب آن یعنی b/m تعیین می‌گردد. هرچه این اندازه بزرگتر باشد، پاسخ سیستم سریع‌تر به حالت ماندگار می‌رسد. چون ما برای تغییر پاسخ سیستم، قادر به تغییر پارامترهای سیستم نمی‌باشیم و باید کنترلری را طراحی کنیم تا صفرها و قطب‌های سیستم حلقه بسته به گونه‌ای قرار بگیرند تا عملکرد سیستم مطلوب شود.

دیاگرام بودی حلقه باز

برای طراحی کنترلر، یکی از اطلاعات مفیدی که می‌توان از سیستم داشت، پاسخ فرکانسی حلقه باز سیستم می‌باشد که با دستور متلب زیر به دست می‌آید:

```
bode(P_cruise)
```



مشاهده می‌شود که در دیاگرام بودی، ویژگی‌های سیستم‌های مرتبه اول مانند اندازه 3 db- و فاز 45 deg- در فرکانس گوشه‌ی برابر w=b/m=0.05 rad/s و همچنین شیب 20 db/dec- در فرکانس‌های بالا نمایان شده است.

بخش سوم: طراحی کنترلر PID

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
tf, step, feedback
```

مدل سیستم و پارامترها

مدل تابع تبدیل برای مسئله کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \tag{۱}$$

پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

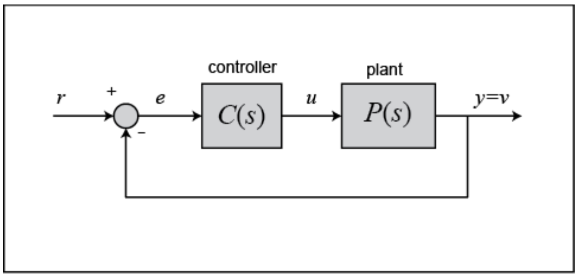
(u) نیروی کنترلی = ۵۰۰ نیوتن

ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

کلیات PID

دیاگرام بلوکی یک سیستم با فیدبک واحد در تصویر زیر نشان داده شده است:



برای یادآوری، تابع تبدیل کنترلر PID عبارتست از:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \tag{۲}$$

با دستور زیر می‌توان کنترلر PID را در متلب مستقیماً به فرم تابع تبدیل تعریف نمود:

```
Kp = 1;
Ki = 1;
Kd = 1;

s = tf('s');
C = Kp + Ki/s + Kd*s
```

```
C =

  s^2 + s + 1
  -----------
       s

Continuous-time transfer function.
```

همچنین می‌توان از شئ pid controller متلب بهره برده و کنترلر پیوسته با زمان PID را تعریف نمود:

```
C = pid(Kp,Ki,Kd)
```

```
C =

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

  with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
```

کنترل تناسبی

اولین قدم در این مسئله، پیدا کردن تابع تبدیل حلقه بسته سیستم با یک کنترل تناسبی $(C = K_p)$ می‌باشد.

تابع تبدیل سیستم با کنترل تناسبی عبارتست از:

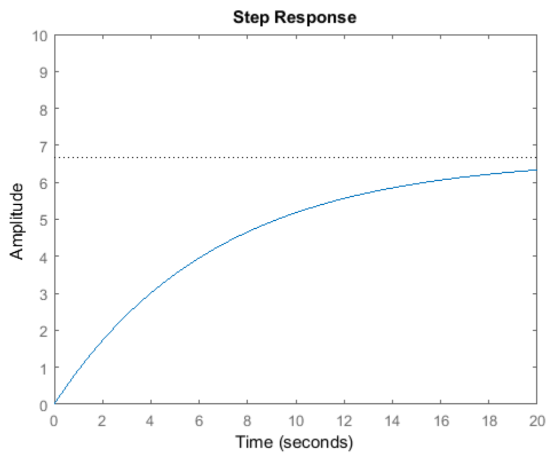
$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_p}{ms + b + K_p} \qquad (۳)$$

برای یادآوری لازم به ذکر است که کنترلر تناسبی K_p، زمان نمو را کاهش می‌دهد که در اینجا مطلوب می‌باشد.

حال از ضریب تناسبی K_p برابر ۱۰۰ و سرعت مرجع برابر 10 m/s استفاده می‌کنیم. در یک امفایل جدید، دستورات زیر را وارد کنید:

```
m = 1000;
b = 50;
r = 10;

s = tf('s');
P_cruise = 1/(m*s + b);

Kp = 100;
C = pid(Kp);

T = feedback(C*P_cruise,1)

t = 0:0.1:20;
step(r*T,t)
axis([0 20 0 10])
```

```
T =

       100

  ------------

  1000 s + 150

 Continuous-time transfer function.
```



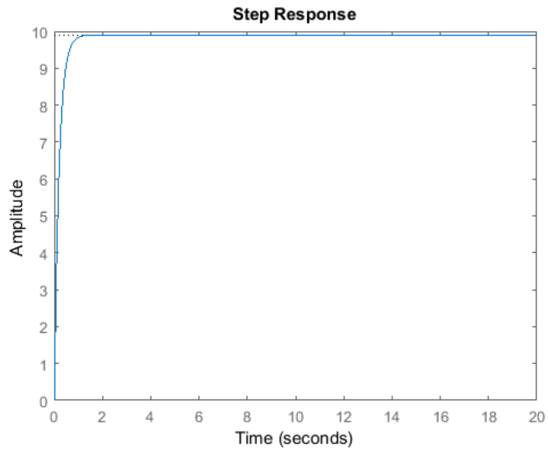
در کد بالا، از دستور `feedback` استفاده شده است که این دستور دیاگرام بلوکی را به فرم حلقه بسته تبدیل می‌کند. برای اطمینان از نتیجه‌ی به دست آمده، می‌توانید تابع تبدیل حلقه بسته‌ی T به دست آمده را به طور دستی حساب کنید.

با اجرای این امفایل در متلب، پاسخ پله‌ی فوق به دست می‌آید. همانطور که از نمودار مشخص است، نه زمان نمو و نه خطای حالت ماندگار رضایت‌بخش نیستند.

برای کاهش زمان نمو و خطای حالت ماندگار، می‌توان K_p را افزایش داد. مقدار K_p را در امفایل به ۵۰۰۰ تغییر داده و آنرا دوباره اجرا کنید. با اینکار نمودار زیر به دست می‌آید.

```
Kp = 5000;
C = pid(Kp);
T = feedback(C*P_cruise,1);
```

```
step(r*T,t)
axis([0 20 0 10])
```



حال خطای حالت ماندگار صفر می‌باشد و زمان نمو نیز کاهش چشمگیری داشته است. اگرچه این پاسخ غیر واقعی می‌باشد زیرا یک سیستم کنترل کروز واقعی نمی‌تواند در کمتر از ۰/۵ ثانیه، سرعت خودرو را از 0 به 10 m/s برساند زیرا محدودیت‌های زیادی برای موتور و سیستم انتقال قدرت وجود دارد.

محدودیت‌های عملگر یکی از مواردی است که اغلب در مسائل واقعی مهندسی به آن بر می‌خوریم و در نتیجه در هنگام طراحی کنترل جدید باید آنرا در نظر گرفت. در فصل‌های بعد به این مسئله بیشتر خواهیم پرداخت.

راه حل این مسئله، انتخاب بهره‌ی تناسبی کمتر که زمان نشست معقول را ایجاد کند می‌باشد. همچنین برای حذف خطای حالت ماندگار از کنترلر انتگرالی استفاده می‌کنیم.

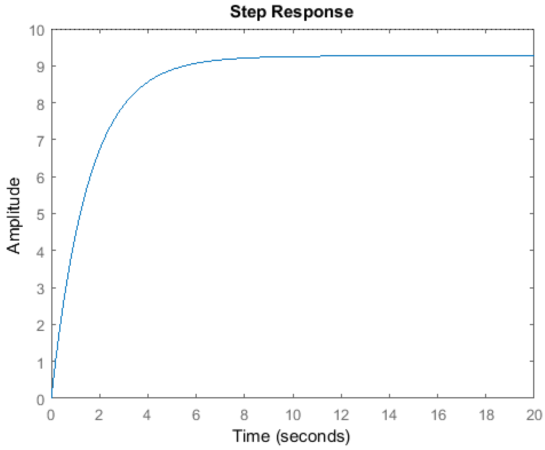
کنترل PI

تابع تبدیل سیستم حلقه بسته با کنترلر PI ($C = K_p + K_i/s$) به شکل زیر است:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_P s + K_i}{ms^2 + (b + K_p)s + K_i} \quad (۴)$$

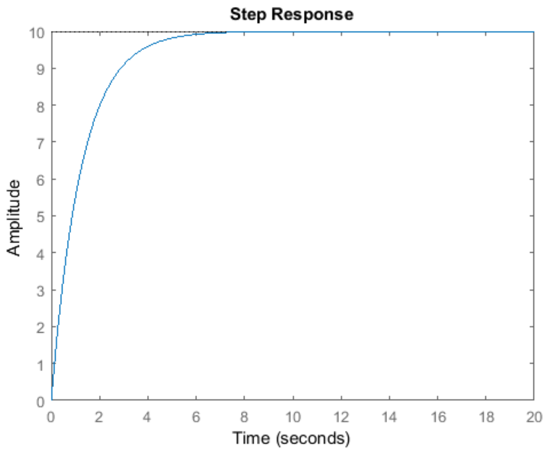
به یاد داریم که با اضافه شدن کنترلر انتگرالی به سیستم، خطای حالت ماندگار حذف می‌گردد. حال K_p را برابر ۶۰۰ و K_i را برابر ۱ قرار دهید تا پاسخ جدید به دست آید.

```
Kp = 600;
Ki = 1;

C = pid(Kp,Ki);
T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```



حال باید دو بهره‌ی تناسبی K_p و بهره‌ی انتگرالی K_i را تنظیم نمود تا پاسخ مطلوب حاصل شود. در هنگام تنظیم بهره‌ی انتگرالی K_i توصیه می‌شود که از مقادیر کوچک شروع کنید زیرا مقادیر بزرگ K_i می‌تواند سیستم را ناپایدار سازد. با قرار دادن $K_p = 800$ و $K_i = 40$، پاسخ پله سیستم حلقه بسته به شکل زیر در می‌آید:

```
Kp = 800;
Ki = 40;

C = pid(Kp,Ki);
T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```



کنترل PID

برای این مثال، نیازی به کنترل مشتق برای رسیدن به خروجی مطلوب نمی‌باشد اما برای نشان دادن روش کار کنترل PID در این بخش به آن می‌پردازیم. تابع تبدیل سیستم به همراه کنترلر PID ($C = K_p + K_i/s + K_d s$) به شکل زیر است:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_d s^2 + K_p s + K_i}{(m + K_d)s^2 + (b + K_p)s + K_i} \qquad (۵)$$

مقادیر بهره‌های کنترلر را با دستورات زیر وارد متلب کنید:

```
Kp = 1;
Ki = 1;
Kd = 1;

C = pid(Kp,Ki,Kd);
T = feedback(C*P_cruise,1);
```

پاسخ پله را رسم کرده و ضرایب K_p، K_i و K_d را به گونه‌ای تنظیم می‌کنیم تا نتیجه‌ی رضایت‌بخش حاصل شود. این قسمت را به عنوان تمرین بر عهده‌ی خواننده می‌گذاریم.

توصیه: معمولا برای انتخاب بهره‌های مناسب، نیاز به سعی و خطا می‌باشد. بهترین روش برای اینکار، تنظیم کردن هر یک از بهره‌ها به نوبت و مشاهده تغییرات حاصل در خروجی سیستم می‌باشد. مشخصات بهره‌های K_p، K_i و K_d در **فصل اول - بخش سوم: کنترلر PID** آمده است.

بخش چهارم: طراحی کنترلر با مکان هندسی ریشه‌ها

فهرست مطالب بخش

- مدل سیستم
- پارامترهای سیستم
- ویژگی‌های عملکرد
- کنترل تناسبی
- کنترلر لَگ

دستورهای کلیدی متلب در این بخش:

```
tf, rlocus, feedback, step
```

مدل سیستم

مدل تابع تبدیل برای مسئله‌ی کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \tag{۱}$$

پارامترهای سیستم

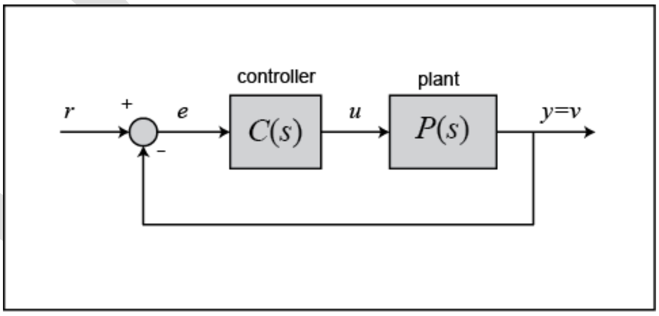
پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

دیاگرام بلوکی سیستم با فیدبک واحد به شکل زیر است:



ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

کنترل تناسبی

از فصل مقدمه‌ای بر طراحی کنترلر با مکان هندسی ریشه‌ها به خاطر داریم که نمودار مکان هندسی ریشه‌ها، موقعیت تمامی قطب‌های حلقه بسته‌ی ممکن را وقتی بهره‌ی سیستم از صفر تا بینهایت تغییر می‌کند را نشان می‌دهد. بنابراین

برای حل این مسئله تنها از یک کنترلر تناسبی K_p استفاده می‌گردد. در این صورت تابع تبدیل حلقه بسته به شکل زیر در می‌آید:

$$\frac{Y(s)}{R(s)} = \frac{K_p}{ms + (b + K_p)} \tag{۲}$$

همچنین می‌دانیم با دستور `sgrid` می‌توانیم نواحی مطلوب نمودار مکان هندسی ریشه‌ها را مشخص کنیم. برای استفاده از دستور `sgrid` لازم است تا ضریب میرایی ζ و فرکانس طبیعی ω_n مشخص باشند. با استفاده از دو معادله‌ی زیر می‌توان ضریب میرایی و فرکانس طبیعی را به دست آورد:

$$\omega_n \geq \frac{1.8}{T_r} \tag{۳}$$

$$\zeta \geq \sqrt{\frac{\ln^2(M_p)}{\pi^2 + \ln^2(M_p)}} \tag{۴}$$

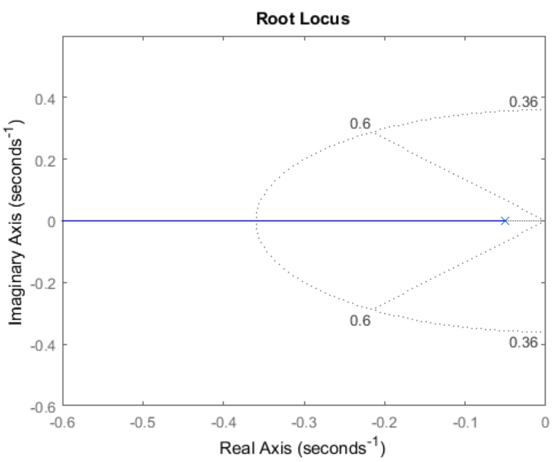
در این معادلات داریم:

- ω_n فرکانس طبیعی بر حسب rad/s
- ζ ضریب میرایی
- Tr زمان نمو بر حسب ثانیه
- Mp ماکزیمم فراجهش

یکی از نیازهای طراحی، زمان نمو کمتر از ۵ ثانیه می‌باشد. با استفاده از معادله‌ی اول فرکانس طبیعی باید بزرگتر از ۰/۳۶ باشد. همچنین از معادله‌ی دوم در می‌یابیم ضریب میرایی باید بیشتر از ۰/۶ باشد تا ماکزیمم فراجهش از ۱۰ ٪ کمتر شود.

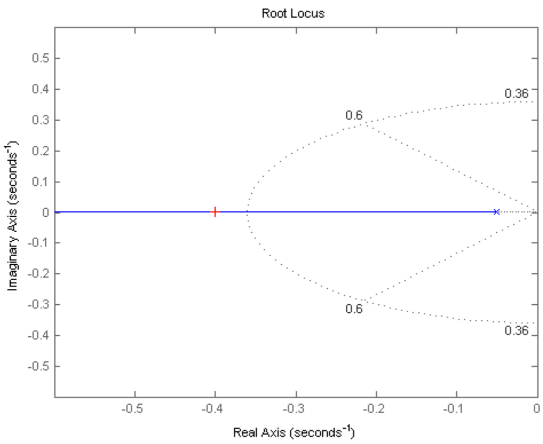
اکنون می‌توان نمودار مکان هندسی ریشه‌ها را با استفاده از دستور `sgrid` برای مشخص کردن نواحی قابل قبول، رسم نمود. در یک ام‌فایل جدید دستورات زیر را وارد کنید:

```
m = 1000;
b = 50;
r = 10;

s = tf('s');
P_cruise = 1/(m*s+b);

rlocus(P_cruise)
axis([-0.6 0 -0.6 0.6]);
sgrid(0.6, 0.36)
```



خطوط خط‌چین زاویه دار، موقعیت ضریب میرایی ثابت ($\zeta = 0.6$) را مشخص می‌کند. در بین این دو خط، ضریب میرایی بزرگتر از ۰/۶ و در خارج از آنها کمتر از ۰/۶ می‌باشد. نیم‌بیضی خط‌چین نشان‌دهنده موقعیت فرکانس طبیعی ثابت ($\omega_n = 0.36$) می‌باشد. نقاط داخل آن دارای فرکانس طبیعی کمتر از ۰/۳۶ و خارج آن دارای فرکانس طبیعی بیشتر از ۰/۳۶ می‌باشند.

با داشتن ناحیه‌ی مطلوب، می‌توانیم با دستور `rlocfind` بهره‌ای را پیدا کنیم تا قطب‌های حلقه بسته در ناحیه‌ی مطلوب قرار بگیرند. کد `[Kp,poles]=rlocfind(P_cruise)` را در انتهای ام‌فایل خود اضافه کنید تا بهره‌ی مورد نظر را به دست آورید. با اجرای این دستور، پیامی را مشاهده می‌کنید که از شما می‌خواهد نقطه‌ای را در نمودار مکان هندسی ریشه‌ها انتخاب کنید. نقطه‌ی انتخاب شده باید بین خطوط خط‌چین ($\zeta > 0.6$) و خارج از نیم‌بیضی ($\omega_n > 0.36$) باشد. در نقطه‌ای خارج از نیم‌بیضی و بر روی محور حقیقی (تقریبا در 0.4-) نقطه‌ای را انتخاب کنید.

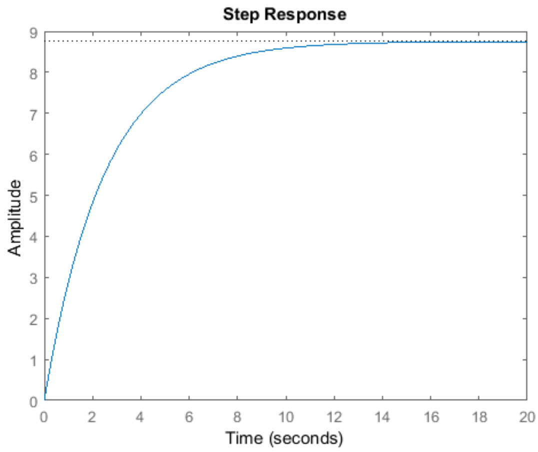


با این کار، خروجی زیر در متلب نمایش داده می‌شود:

```
Select a point in the graphics window

         selected_point =


         -0.4002 + 0.0019i

         Kp =


         350.2419

         poles =


         -0.4002
```

با استفاده از بهره‌ی به دست آمده، پاسخ پله‌ی سیستم حلقه بسته به شکل زیر در می‌آید:

```
Kp = 350.2419;

sys_cl = feedback(Kp*P_cruise,1);

t = 0:0.1:20;

step(r*sys_cl,t)
```



با بهره‌ی K_p استفاده شده، هر دو شرط زمان نمو و فراجهش ارضا شده‌اند، هرچند که خطای حالت ماندگار بیش از ۱۰٪ وجود دارد.

کنترلر پس‌فاز[۲۹]

برای کاهش خطای حالت ماندگار، به سیستم، کنترلر پس‌فاز را اضافه می‌نماییم. تابع تبدیل کنترلر پس‌فاز عبارتست از:

$$C(s) = \frac{s+z_0}{s+p_0} \tag{۵}$$

در اینصورت تابع تبدیل سیستم (بدون عبارت K_p) به شکل:

$$\frac{Y(s)}{R(s)} = \frac{s+z_0}{ms^2+(b+mp_0)s+bp_0} \tag{۶}$$

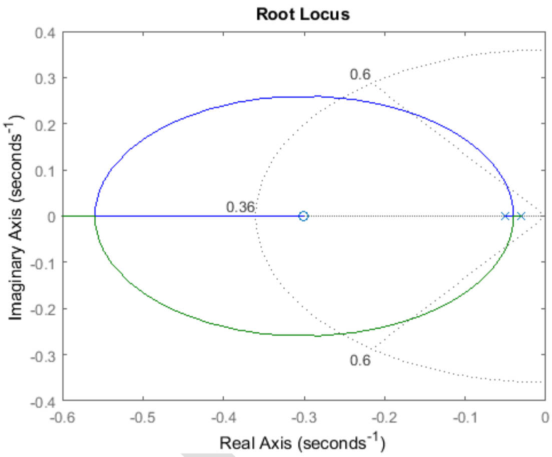
در می‌آید. در نهایت با اضافه کردن بهره‌ی K_p، تابع تبدیل نهایی به شکل زیر به دست می‌آید:

$$\frac{Y(s)}{U(s)} = \frac{K_p s+K_p z_0}{ms^2+\left(b+mp_0+K_p\right)s+\left(bp_0+K_p z_0\right)} \tag{۷}$$

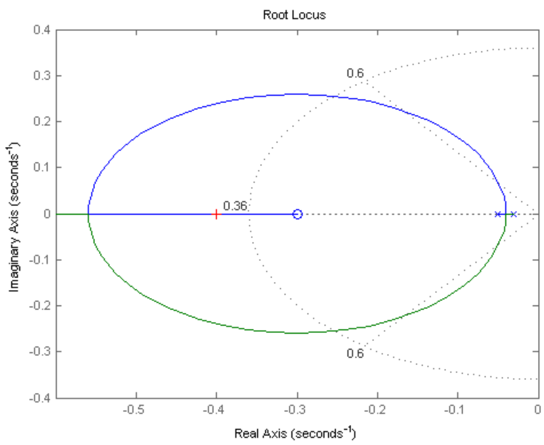
در **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز**، گفته شد که باید صفر و قطب کنترلر پس‌فاز در نزدیک یکدیگر قرار بگیرند. همچنین گفته شد که خطای حالت ماندگار به نسبت z_0/p_0 کاهش می‌یابد. به همین دلیل z_0 را برابر ۰/۳ و p_0 را برابر ۰/۰۳ در نظر می‌گیریم. با ساخت و اجرای یک ام‌فایل جدید نتیجه‌ی زیر حاصل می‌گردد:

```
zo = 0.3;
po = 0.03;

s = tf('s');
C_lag = (s+zo)/(s+po);

rlocus(C_lag*P_cruise);
axis([-0.6 0 -0.4 0.4])
sgrid(0.6,0.36);
```

[۲۹] Lag Controller



دوباره با اجرای دستور rlocfind، بهره‌ی جدید K_p به دست می‌آید. کد `[Kp,poles]=rlocfind(C_lag*P_cruise)` را اجرا کرده و نقطه‌ای در نزدیکی 0.4- بر روی محور حقیقی را انتخاب کنید.

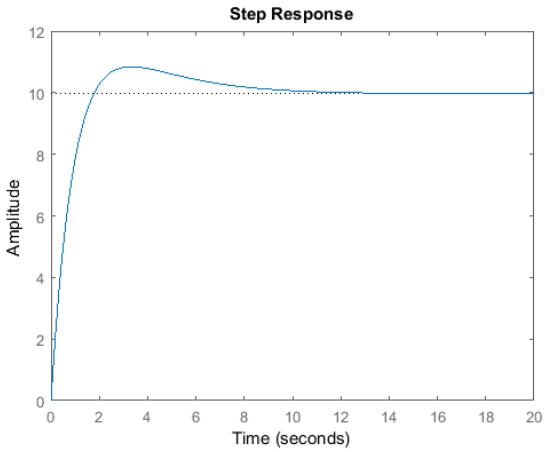


با انتخاب این نقطه، نتایج زیر حاصل می‌گردد:

```
Select a point in the graphics window

         selected_point =

         -0.4002 - 0.0012i

         Kp =

         1.2936e+03

         poles =

         -0.9733
         -0.4003
```

برای به دست آوردن پاسخ پله‌ی حلقه بسته‌ی جدید، کد زیر را اجرا کنید:

```
Kp = 1293.6;

sys_cl = feedback(Kp*C_lag*P_cruise,1);

t = 0:0.1:20;

step(r*sys_cl,t)

axis([0 20 0 12])
```



همانطور که مشاهده می‌شود خطای حالت ماندگار به نزدیکی صفر کاهش پیدا کرده است. فراجهش ایجاد شده ناشی از اضافه کردن صفر توسط کنترلر پس‌فاز می‌باشد. حال تمامی الزامات طراحی ارضا شده است اما برای تمرین، سایر مقادیر z_0 و p_0 را آزموده و تاثیر آنرا بر روی پاسخ سیستم حلقه بسته مشاهده کنید.

بخش پنجم: طراحی کنترلر در حوزه فرکانس

فهرست مطالب بخش

- مدل سیستم
- پارامترهای سیستم
- ویژگی‌های عملکرد
- دیاگرام بودی و پاسخ حلقه باز
- کنترلر تناسبی
- جبران‌ساز لگ

دستورهای کلیدی متلب در این بخش:

```
tf, feedback, step
```

مدل سیستم

مدل تابع تبدیل برای مسئله کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \tag{۱}$$

پارامترهای سیستم

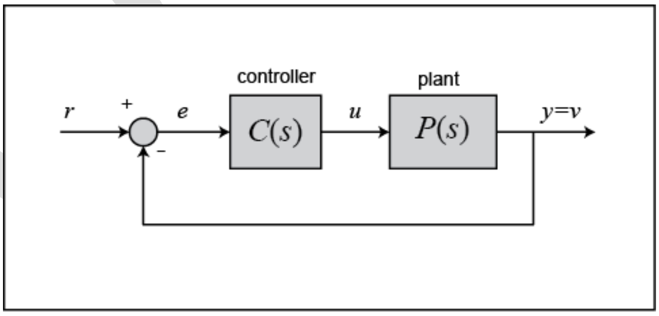
پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

دیاگرام بلوکی سیستم با فیدبک واحد به شکل زیر است:

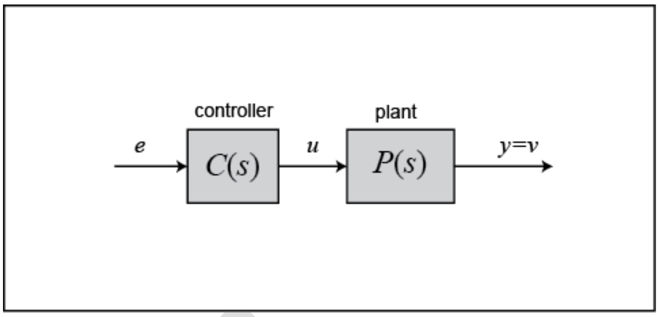


ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

دیاگرام بودی و پاسخ حلقه باز

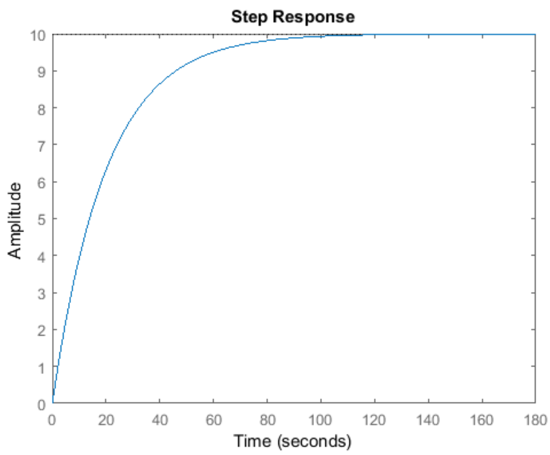
اولین قدم برای حل این مسئله از طریق پاسخ فرکانسی، مشخص کردن تابع تبدیل حلقه باز است. در این روش هم مانند روش مکان هندسی ریشه‌ها، از یک کنترل تناسبی برای حل مسئله استفاده خواهد شد. دیاگرام بلوکی و تابع تبدیل حلقه باز به شرح زیر است:



$$\frac{Y(s)}{E(s)} = \frac{K_p}{ms + b} \qquad (۲)$$

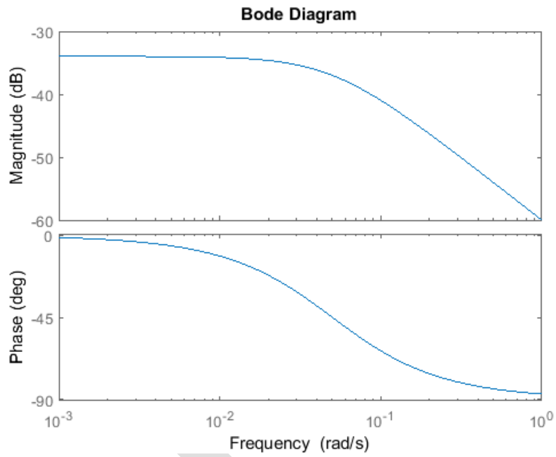
برای استفاده از دیاگرام بودی، پاسخ حلقه باز باید پایدار باشد. مقدار K_p را برابر ۱ در نظر گرفته و پاسخ حلقه باز را بررسی می‌کنیم. امفایل جدیدی را ساخته و دستورات زیر را در آن ذخیره و اجرا کنید:

```
m = 1000;
b = 50;
u = 500;

Kp = 1;

s = tf('s');
P_cruise = 1/(m*s+b);
C = Kp;
step(u*C*P_cruise)
```



همانطورکه مشاهده می‌شود، سیستم حلقه باز پایدار است. بنابراین می‌توانیم دیاگرام بودی را رسم کنیم. امفایل بالا را تغییر داده و به جای دستور step، دستور زیر را قرار دهید و اجرا کنید:

```
bode(C*P_cruise);
```



کنترل تناسبی

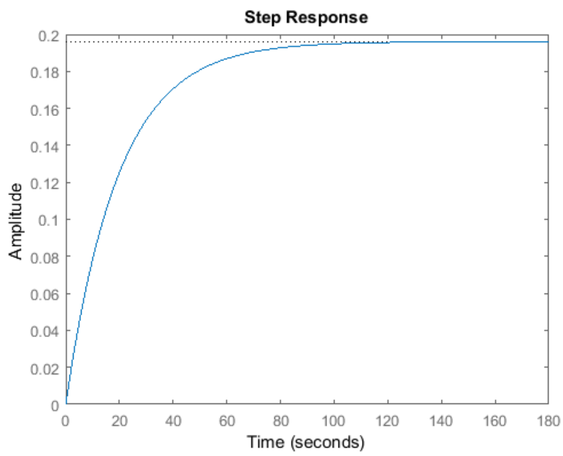
در **فصل دوم - بخش پنجم: مقدمه‌ای بر طراحی کنترلر در حوزه‌ی فرکانس**، گفته شد که چه ویژگی‌هایی از سیستم را می‌توان از دیاگرام بودی استنتاج کرد.

خطای حالت ماندگار از رابطه‌ی زیر به دست می‌آید:

$$\text{خطای حالت ماندگار} = \frac{1}{1 + M_{\omega \to 0}} \times 100\% \qquad (۳)$$

برای این سیستم، بهره‌ی فرکانس پایین برابر 0.02 = 34db- می‌باشد در نتیجه خطای حالت ماندگار برابر ۹۸٪ به دست می‌آید. برای صحت این مطلب، می‌توانیم پاسخ پله حلقه بسته را رسم کنیم:

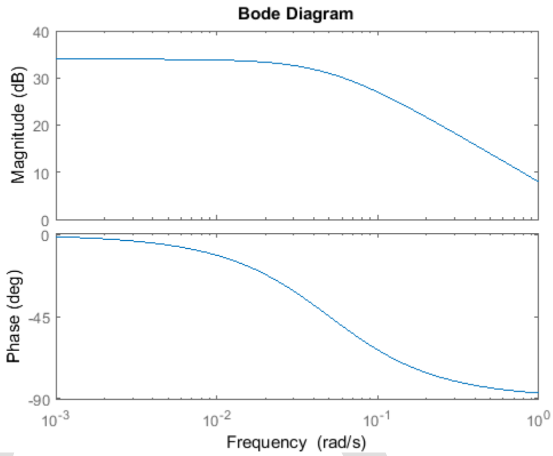
```
r = 10;
sys_cl = feedback(C*P_cruise,1);

step(r*sys_cl);
```



برای بهبود خطای حالت ماندگار باید بهره‌ی فرکانس پایین را افزایش دهیم. خطای حالت ماندگار باید کمتر از ۲٪ باشد، بنابراین داریم $1/(1 + M_{\omega=0}) < 0.02$ پس $M_{\omega=0} > 49 = 33.8db$. چون بهره‌ی فرکانس پایین برابر 34db- می‌باشد و برای رسیدن به مقدار مطلوب خطای حالت ماندگار باید بهره‌ی فرکانس پایین برابر ۳۳/۸ دسیبل داشته باشیم، با

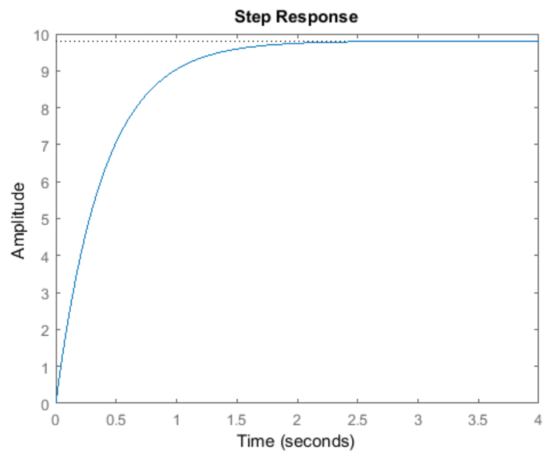
استفاده از تنها یک کنترلر تناسبی، لازم است که $K_p > (34db + 33.8db) = 67.8db = 2455$ باشد. حال با این بهره‌ی تناسبی، دیاگرام بودی سیستم حلقه باز را رسم می‌کنیم:

```
Kp = 2500;
C = Kp;

bode(C*P_cruise);
```



همانطور که از دیاگرام بودی بالا مشاهده می‌کنید، اندازه فرکانس پایین برابر ۳۴ دسیبل می‌باشد. حال با این بهره، پاسخ پله‌ی سیستم حلقه بسته را رسم می‌کنیم:

```
sys_cl = feedback(C*P_cruise,1);

step(r*sys_cl);
```



مقدار خطای حالت ماندگار به حد مطلوب رسیده است. هرچند که زمان نمو بسیار کمتر از مقدار خواسته شده و در این مثال غیر معقول می‌باشد زیرا خودرو نمی‌تواند در ۲ ثانیه به سرعت 10 m/s برسد. در نتیجه از یک بهره‌ی تناسبی کوچک‌تر به همراه یک جبران‌ساز پس‌فاز استفاده می‌کنیم.

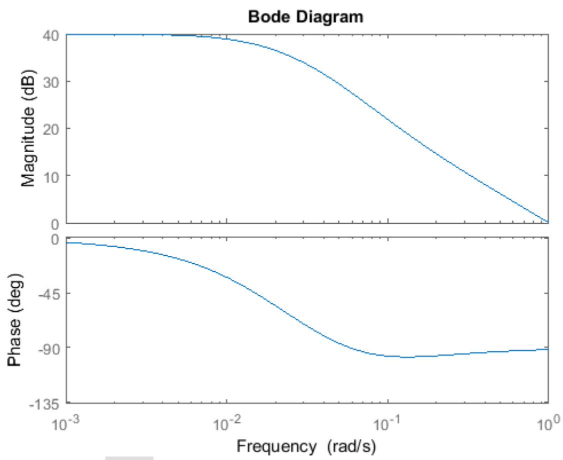
جبران‌ساز پس‌فاز

در **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز** گفته شد که جبران‌ساز پس‌فاز بهره‌ی فرکانس پایین را اضافه می‌کند اما فرکانس پهنای باند را ثابت نگه می‌دارد. در این مسئله دقیقاً به جبران‌ساز پس‌فاز نیاز داریم زیرا بهره‌ی فرکانس پایین بزرگ‌تر، خطای حالت ماندگار کمتری را نتیجه داده اما فرکانس پهنای باند ثابت می‌ماند که سبب ثابت ماندن زمان نمو می‌شود. تابع تبدیل کنترلر پس‌فاز به شکل زیر است:

$$C_{\text{پس‌فاز}}(s) = \frac{s+z_0}{s+p_0} \qquad (۴)$$

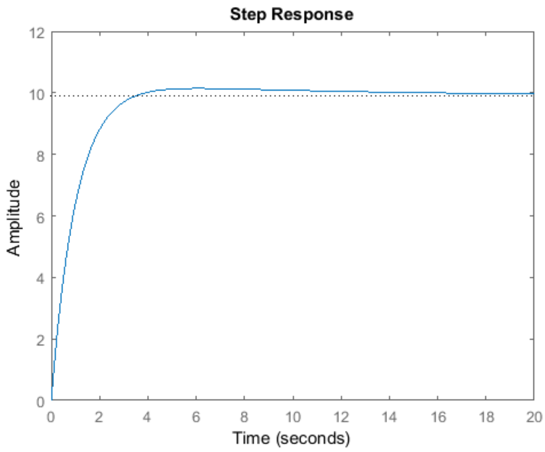
در **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز** گفته شد که در کنترلر پس‌فاز، قطب و صفر باید در نزدیکی هم قرار بگیرند و همچنین در این صورت خطای حالت ماندگار به نسبت z_0/p_0 کاهش می‌یابد. به همین دلیل مقدار z_0 را برابر ۰/۱ و مقدار p_0 را برابر ۰/۰۲ قرار می‌دهیم. بهره‌ی تناسبی $K_p = 1000$ به وسیله‌ی سعی و خطا انتخاب شده است:

```
Kp = 1000;
zo = 0.1;
po = 0.02;

C_lag = (s+zo)/(s+po);

bode(Kp*C_lag*P_cruise);
```



برای اطمینان از عملکرد سیستم، پاسخ پله‌ی حلقه بسته را رسم می‌کنیم:

```
sys_cl = feedback(Kp*C_lag*P_cruise,1);

t = 0:0.1:20;

step(r*sys_cl,t);
```



همانطور که مشاهده می‌کنید مقدار کمی فراجهش وجود داشته، خطای حالت ماندگار نزدیک صفر و زمان نمو کمتر از ۵ ثانیه است. حال سیستم ما تمامی خواسته‌های طراحی را دارا می‌باشد و نیازی و اقدامات بیشتر نمی‌باشد.

بخش ششم: طراحی کنترلر در فضای حالت

فهرست مطالب بخش

- معادلات فضای حالت
- نیازهای طراحی
- طراحی کنترل با استفاده از جایدهی قطب
- ورودی مرجع

در این بخش با استفاده از مدل فضای حالت به طراحی کنترلر و مشاهده‌گر برای سیستم کنترل کروز می‌پردازیم.

دستورهای کلیدی متلب در این بخش:

```
ss, feedback
```

معادلات فضای حالت

معادلات حرکت به فرم فضای حالت به شرح زیر است:

$$[\dot{v}] = \left[\frac{-b}{m}\right][v] + \left[\frac{1}{m}\right][u] \qquad (۱)$$

$$y = [1][v] \qquad (۲)$$

که در آن:

(m) جرم خودرو ۱۰۰۰ کیلوگرم

(b) ضریب میرایی 50 N.s/m

(u) نیروی کنترل نامی ۵۰۰ نیوتن

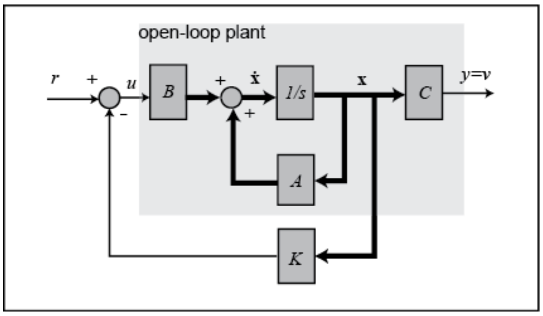
(v) سرعت خودرو که خروجی سیستم است عبارتست از y = v

الزامات طراحی

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

طراحی کنترل با استفاده از جایدهی قطب

شماتیک سیستم با فیدبک تمام حالات به شکل زیر است:



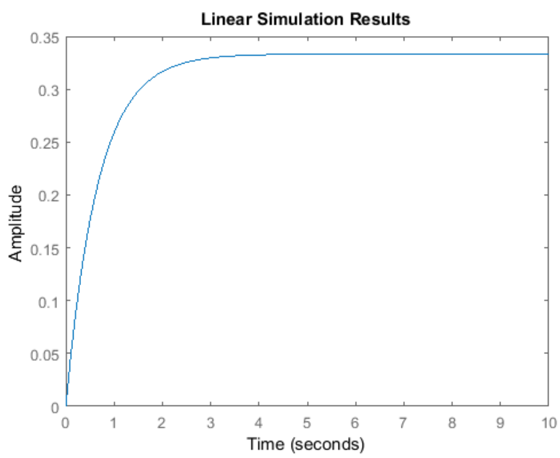
در این شماتیک، K ماتریس بهره فیدبک حالت‌ها و u ورودی کنترلی که برابر $u = r - kx = r - kv$ است می‌باشد.

در **فصل دوم - بخش ششم: مقدمه‌ای بر طراحی کنترلر در فضای حالت،** گفته شد که می‌توان از تکنیک جایدهی قطب برای به دست آوردن خروجی مطلوب استفاده نمود. قطب‌های سیستم حلقه بسته را می‌توان از معادله‌ی مشخصه به دست آورد که در فرم فضای حالت، معادله‌ی مشخصه برابر دترمینان ماتریس $[sI - (A - B \times K)]$ می‌باشد. اگر بتوان قطب‌های سیستم را با ماتریس کنترل مناسب K در موقعیت مطلوب قرار داد پس می‌توان خروجی مطلوب را به دست آورد. در این بخش ابتدا قطب‌ها انتخاب شده و با استفاده از متلب، ماتریس کنترل مربوطه‌ی K را پیدا می‌کنیم.

حال باید موقعیت قطب‌ها را مشخص کنیم. چون ماتریس $[sI - (A - B \times K)]$ ماتریس 1x1 می‌باشد، تنها لازم است تا یک قطب را جایدهی کنیم. فرض کنید می‌خواهیم قطب در 1.5- (موقعیت دلخواه) باشد. همانطور که در فصل اول گفته شد، از دستور place در متلب برای به دست آوردن ماتریس کنترل K استفاده می‌کنیم. یک امفایل جدید ساخته و دستورات زیر را در آن وارد کنید. با اجرای این امفایل، ماتریس کنترل و پاسخ پله نمایش داده می‌شود.

```
m = 1000;
b = 50;
t = 0:0.1:10;
u = 500*ones(size(t));

A = [-b/m];
B = [1/m];
C = [1];
D = [0];

sys = ss(A,B,C,D);

x0 = [0];

p1 = -1.5;

K = place(A,B,[p1])

sys_cl = ss(A-B*K,B,C,D);
lsim(sys_cl,u,t,x0);
axis([0 10 0 0.35])
```

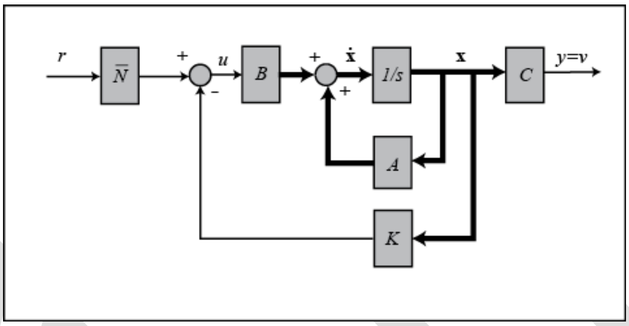
```
K =

        1450
```



از نمودار مشاهده می‌کنیم که زمان نمو رضایت‌بخش اما خطای حالت ماندگار بسیار زیاد است.

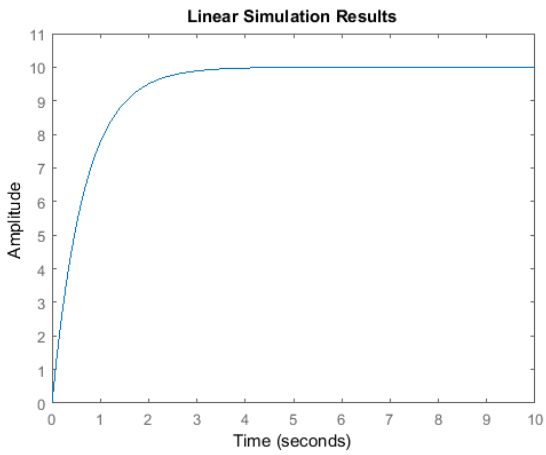
ورودی مرجع

در **فصل دوم – بخش ششم: مقدمه‌ای بر طراحی کنترلر در فضای حالت،** ضریب بزرگنمایی Nbar (در شماتیک زیر مشخص شده است) معرفی شد که می‌توان از آن برای حذف خطای ماندگار استفاده کرد. از دستور `rscale` برای محاسبه‌ی ضریب بزرگنمایی استفاده می‌نماییم. از ام‌فایل rscale.m موجود در سی‌دی استفاده کنید. در حال حاضر ورودی در عدد ۵۰۰ ضرب شده و هدف ما سرعت حالت ماندگار 10 m/s می‌باشد.



کدهای زیر را در یک ام‌فایل وارد کرده و آنرا اجرا کنید. به این ترتیب پاسخ پله نمایش داده می‌شود:

```
Nbar = rscale(sys,K)*10/500;

sys_cl = ss(A-B*K,B*Nbar,C,D);

lsim(sys_cl,u,t,x0);

axis([0 10 0 11])
```



از پاسخ پله‌ی مشاهده شده می‌توان دید که خطای حالت ماندگار از بین رفته است. زمان نمو کمتر از ۵ ثانیه و فراجهش صفر می‌باشد. تمامی نیازهای طراحی برآورده شده است.

بخش هفتم: طراحی کنترلر دیجیتال

فهرست مطالب بخش

در این بخش قصد داریم از روش مکان هندسی ریشه‌ها برای طراحی کنترلر دیجیتال استفاده کنیم.

دستورهای کلیدی متلب در این بخش:

```
tf, c2d, rlocus, zgrid, feedback, step
```

مدل سیستم

مدل تابع تبدیل برای مسئله‌ی کنترل کروز به شکل زیر است:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms+b} \quad \left[\frac{m/s}{N}\right] \tag{۱}$$

پارامترهای سیستم

پارامترهای استفاده شده در این مسئله عبارتند از:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

(u) نیروی کنترلی = ۵۰۰ نیوتن

ویژگی‌های عملکرد

- زمان نشست کمتر از ۵ ثانیه
- فراجهش کمتر از ۱۰٪
- خطای حالت ماندگار کمتر از ۲٪

تابع تبدیل گسسته

اولین قدم در تحلیل سیستم به طور گسسته، به دست آوردن تابع تبدیل گسسته‌ی سیستم پیوسته است. با استفاده از دستور c2d، تابع تبدیل بالا (Y(s)/U(s)) را به تابع تبدیل گسسته تبدیل می‌نماییم. برای استفاده از این تابع، نیاز به سه آرگومان یا ورودی داریم: سیستم، زمان نمونه‌برداری (Ts) و روش ('method'). زمان نمونه‌برداری (Ts) که بر حسب sec/sample بیان می‌شود، باید کمتر از $1/(30BW)$ باشد (BW فرکانس پهنای باند حلقه بسته است). برای آرگومان روش، از روش نگهدارنده مرتبه صفر ('zoh') استفاده می‌کنیم. زمان نمونه برداری را 1/50 ثانیه در نظر می‌گیریم. با توجه به فرکانس پهنای باند 1 rad/sec، این مقدار به اندازه‌ی کافی سریع می‌باشد. حال کدهای زیر را در یک ام‌فایل وارد کرده و آنرا اجرا کنید:

```
m = 1000;
b = 50;
u = 500;

s = tf('s');
P_cruise = 1/(m*s+b);

Ts = 1/50;

dP_cruise = c2d(P_cruise,Ts,'zoh')
```

```
dP_cruise =

   1.999e-05

  ---------

  z - 0.999

Sample time: 0.02 seconds

Discrete-time transfer function.
```

مکان هندسی ریشه‌ها در صفحه z

در **فصل دوم - بخش هفتم: مقدمه‌ای بر طراحی کنترلر دیجیتال،** گفته شد که از تابع `zgrid` برای به دست آوردن نواحی قابل قبول مکان هندسی ریشه‌ها در حالت گسسته استفاده می‌شود که از آن می‌توان مقدار بهره‌ی مطلوب K را به دست آورد. دستور `zgrid` نیاز به دو آرگومان دارد: فرکانس طبیعی (ω_n) و ضریب میرایی (ζ). این دو آرگومان را می‌توان از زمان نمو و فراجهش مطلوب سیستم، با معادلات زیر به دست آورد.

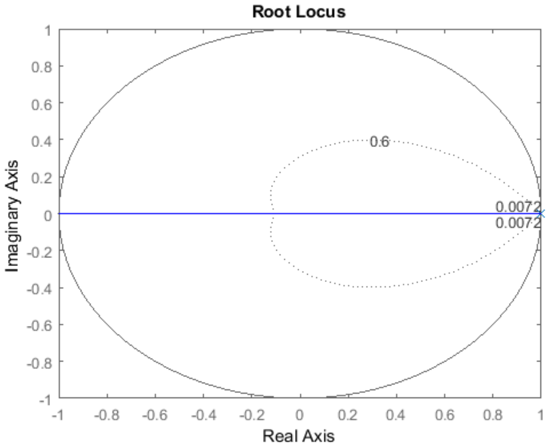
$$\omega_n \geq \frac{1.8}{T_r} \qquad (۲)$$

$$\zeta \geq \sqrt{\frac{\ln^2(M_p)}{\pi^2 + \ln^2(M_p)}} \qquad (۳)$$

زمان نمو مطلوب ۵ ثانیه و فراجهش مطلوب ۱۰٪ است، در این صورت باید فرکانس طبیعی بزرگتر از 0.36 rad/sec و ضریب میرایی بزرگتر از 0.6 باشد.

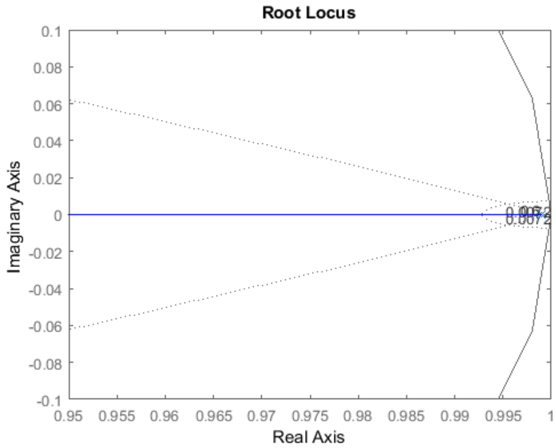
مکان هندسی ریشه‌ها را رسم کرده و با استفاده از دستور `zgrid` نواحی قابل قبول را مشخص می‌کنیم. اما پیش از آن، همانطور که گفته شده بود، در دستور `zgrid` فرکانس طبیعی باید بر واحد rad/sample باشد پس با تبدیل واحد، فرکانس طبیعی برابر $\omega_n = 0.36T_s = 0.0072$ می‌باشد. حال دستورات زیر را در ام‌فایل وارد و آنرا دوباره اجرا کنید تا نمودار زیر را به دست آورید.

```
Wn = 0.0072;
zeta = 0.6;

rlocus(dP_cruise)

zgrid(zeta, Wn)
axis ([-1 1 -1 1])
```



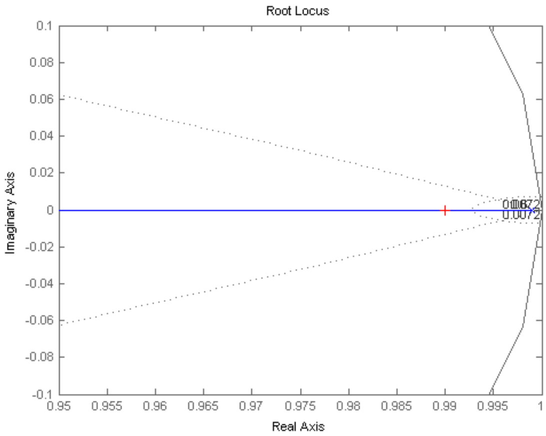
نواحی قابل قبول صفحه‌ی مختلط در نزدیکی نقطه (۱،۰) می‌باشد. برای بررسی بهتر، این نقطه را بزرگنمایی می‌کنیم. دستور زیر را در انتهای ام‌فایل وارد کرده و آنرا دوباره اجرا کنید:

```
axis ([0.95 1 -.1 .1])
```



خطوط خط‌چین در سمت راست، نقاط فرکانس طبیعی ثابت را مشخص می‌کند، در خارج از این دو خط‌چین فرکانس طبیعی بزرگتر از ۰/۰۰۷۲ می‌باشد. دیگر خطوط خط‌چین نشان‌دهنده نقاط ضریب میرایی ثابت می‌باشند که در داخل آنها، ضریب میرایی بزرگتر از ۰/۶ است. خط‌های توپر عمودی، در واقع بخشی از دایره واحد می‌باشند که به علت بزرگنمایی، به صورت شکسته نمایش داده شده‌اند.

در نمودار بالا، بخشی از مکان هندسی ریشه‌ها که در ناحیه‌ی مطلوب قرار دارد را مشاهده می‌کنید. با استفاده از دستور `rlocfind` مقدار بهره‌ی K را به دست آورده و سپس پاسخ پله‌ی مربوطه را رسم می‌کنیم. با اجرای دستور `[K,poles] = rlocfind(dP_cruise)` در متلب، پنجره‌ای باز شده و نقطه‌ی مورد نظر از شما درخواست می‌گردد. دقت کنید که اگر قطبی را که خیلی از دایره‌ی واحد فاصله دارد انتخاب کنید، پاسخ پله‌ی سیستم بسیار سریع شده و به طور فیزیکی نامعقول می‌گردد. بنابراین باید قطبی را که در نزدیکی تقاطع خطوط فرکانس طبیعی ثابت و محور حقیقی می‌باشد را انتخاب کنید. نقطه‌ای را در نزدیکی ۰/۹۹ که در شکل زیر با علامت بعلاوه مشخص شده است، انتخاب کنید.

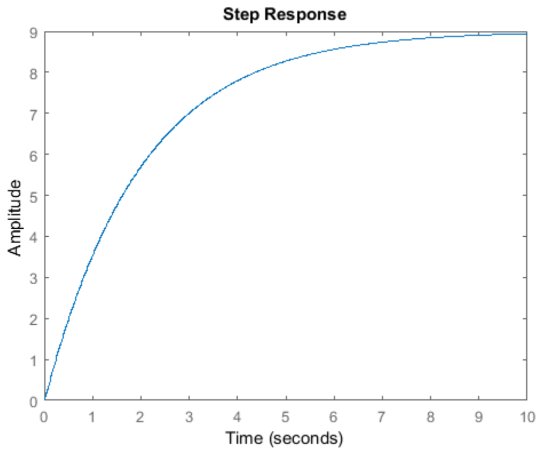


بعد از انتخاب این نقطه، خروجی زیر را در پنجره‌ی دستور متلب مشاهده می‌کنید. این خروجی به شما نقطه‌ی انتخاب شده و بهره‌ی K را اعلام می‌کند.

```
Select a point in the graphics window

        selected_point =

        0.9900 - 0.0003i

        K =

        451.1104

        poles =

        0.9900
```

سپس برای پاسخ پله‌ی حلقه بسته، کد زیر را در ام‌فایل وارد کنید:

```
K = 451.1104;

sys_cl = feedback(K*dP_cruise,1);

r = 10;

figure

step(r*sys_cl,10);
```



این پاسخ، شروط زمان نمو و فراجهش را ارضا می‌کند اما خطای حالت ماندگار حدود ۱۱٪ می‌باشد. برای به دست آوردن خطای حالت ماندگار مطلوب، در قسمت بعد کنترلر دیجیتال را اصلاح می‌کنیم.

جبران‌ساز با استفاده از کنترلر دیجیتال

با یادآوری از بخش‌های قبل، جبران‌ساز پس‌فاز برای به دست آوردن پاسخ مطلوب به سیستم اضافه می‌شد. در حالت کنترل دیجیتال برای مسئله کنترل کروز، کنترلر دیجیتال فعلی را با اضافه کردن جبران‌ساز پس‌فاز، اصلاح می‌کنیم:

$$C_{\text{پس‌فاز}}(z) = K_d \frac{z - z_0}{z - z_p} \qquad (۵)$$

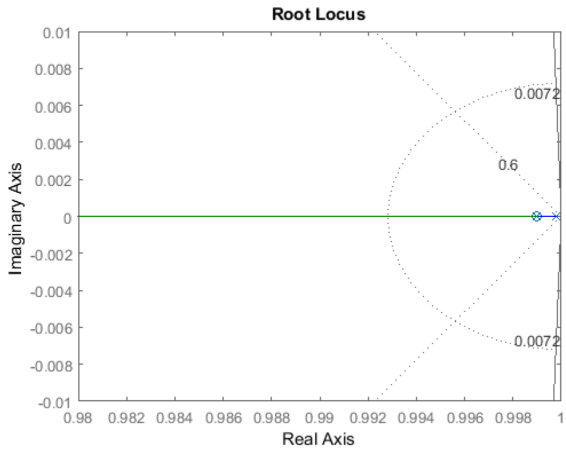
روش‌هایی برای طراحی جبران‌سازهای دیجیتال پیش‌فاز و پس‌فاز و همچنین برای طراحی جبران‌سازهای پیوسته پیش‌فاز و پس‌فاز وجود دارد. در روش طراحی گسسته، صفر جبران‌ساز پس‌فاز باید به گونه‌ای انتخاب شود تا یکی از قطب‌های سیستم را خنثی کند. البته باید دقت شود تا جبران‌ساز ناپایدار نگردد. بنابراین در اینجا صفر را در $z_0 = 0.999$ انتخاب می‌کنیم.

برای کاهش خطای حالت ماندگار، لازم به ذکر است که بهره فرکانس پایین سیستم گسسته با جبران‌ساز پس‌فاز، به نسبت $(1 - z_0)/(1 - z_p)$ افزایش می‌یابد. برای کاهش خطای حالت ماندگار با نسبت ۵، z_p را برابر ۰/۹۹۹۸ انتخاب می‌کنیم. برای داشتن بهره‌ی ۱ در فرکانس صفر، قبل از رسم مکان هندسی ریشه‌ها، صورت کسر را در $K_d = \frac{1-z_p}{1-z_0} =$ 0.2 ضرب می‌کنیم. در نهایت باید کل جبران‌ساز را در بهره‌ی به دست آمده از مکان هندسی ریشه‌ها ضرب نمود.

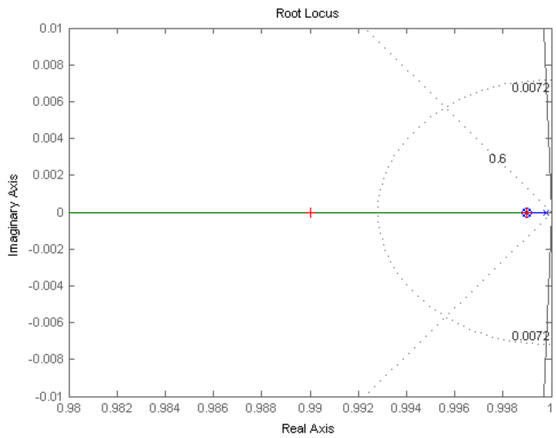
حال که تابع تبدیل جبران‌ساز گسسته را داریم، مکان هندسی ریشه‌ها را رسم کرده و پاسخ پله را به دست می‌آوریم. ابتدا یک ام‌فایل جدید ساخته و دستورات زیر را وارد کنید.

```
m = 1000;
b = 50;
u = 500;

s = tf('s');
P_cruise = 1/(m*s+b);
Ts = 1/50;
dP_cruise = c2d(P_cruise,Ts,'zoh');

z = tf('z',Ts);
C = 0.2*(z - 0.999)/(z - 0.9998);

Wn = 0.0072;
zeta = 0.6;

rlocus(C*dP_cruise)
```

```
zgrid(zeta, Wn)
axis([0.98 1 -0.01 0.01])
```



با اجرای دستور `[K,poles] = rlocfind(C*dP_cruise)` در متلب، دوباره نقطه‌ای در نزدیکی ۰/۹۹ مانند شکل زیر انتخاب کنید.

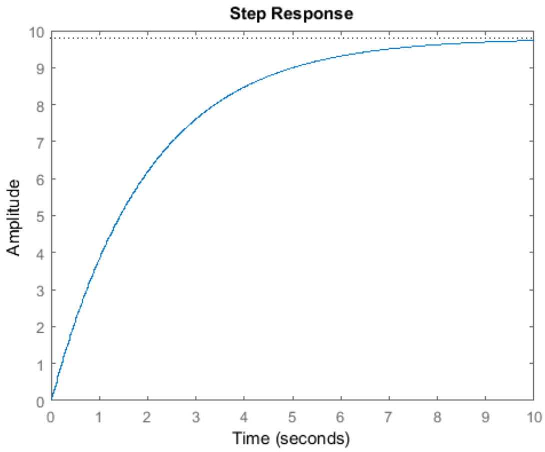


بعد از انتخاب این نقطه، خروجی زیر را در متلب مشاهده می‌کنید:

```
Select a point in the graphics window

        selected_point =

        0.9900 - 0.0000i

        K =

        2.4454e+03

        poles =

        0.9900
        0.9900
```

در نهایت برای مشاهده‌ی پاسخ پله‌ی حلقه بسته، دستور زیر را اجرا می‌کنیم:

```
K = 2.4454e+03;

sys_cl = feedback(K*C*dP_cruise,1);

r = 10;

step(r*sys_cl,10);
```



پاسخ به دست آمده تقریبا سرعتی مانند سرعت پاسخ قبل داشته اما خطای حالت ماندگار آن به ۲٪ کاهش یافته است. این سیستم تمامی نیازهای طراحی را ارضا کرده و پاسخ معقولی دارد.

نکته: مسئله طراحی ممکن است تنها یک جواب نداشته باشد. برای تمرین، سایر جبران‌سازها را آزموده تا پاسخ بهتری را به دست آورید.

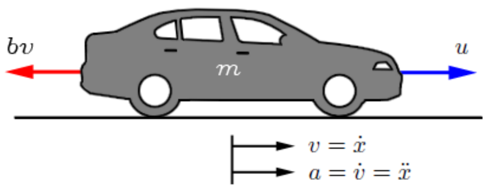
بخش هشتم: مدل‌سازی سیمولینک

فهرست مطالب بخش

- سیستم فیزیکی و معادلات آن
- ساخت مدل
- پاسخ حلقه باز

سیستم فیزیکی و معادلات آن

معادلات سیستم کنترل کروز نسبتاً ساده می‌باشند. اگر مقاومت غلتشی و درگ هوا را متناسب با سرعت خودرو در نظر بگیریم، مسئله مانند یک مسئله‌ی ساده‌ی جرم و دمپر تبدیل می‌شود.



با استفاده از قانون دوم نیوتن، معادلات حاکم بر سیستم عبارتند از:

$$m\dot{v} = u - bv \qquad (۱)$$

که u نیروی ایجاد شده از تماس چرخ و زمین است که به طور مستقیم کنترل می‌شود. برای این مثال مقادیر زیر را در نظر بگیرید:

(m) جرم خودرو = ۱۰۰۰ کیلوگرم

(b) ضریب میرایی = 50 N.s/m

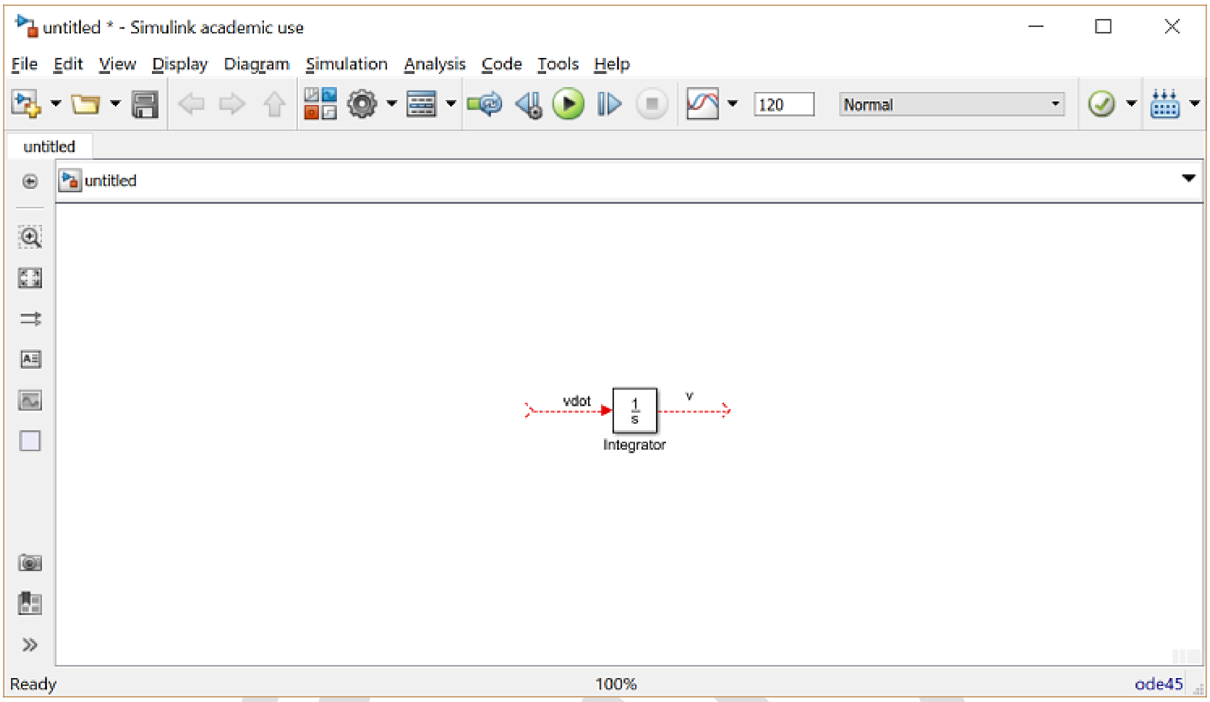
(u) نیروی کنترلی = ۵۰۰ نیوتن

ساخت مدل

مدل این سیستم از مجموع نیروهای وارد شده به جرم و انتگرال‌گیری از شتاب (که سرعت را نتیجه می‌دهد) به دست می‌آید:

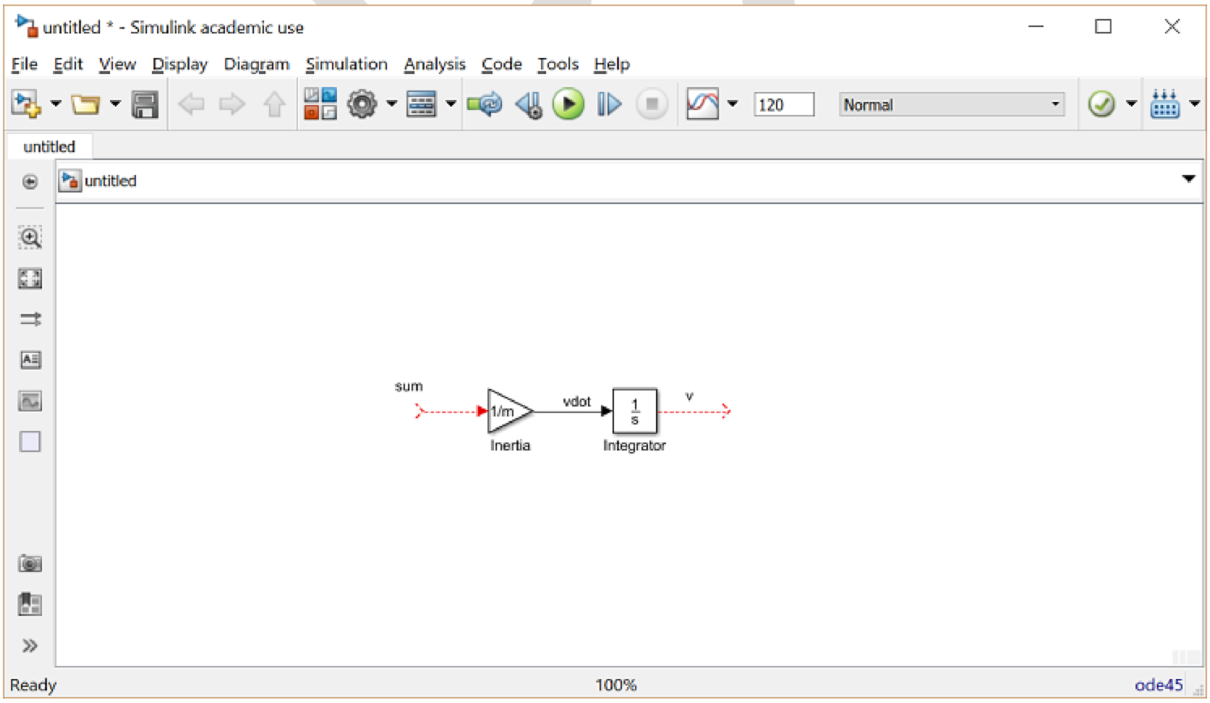
$$\int \frac{dv}{dt} dt = v \qquad (۲)$$

- یک بلوک Integrator (از کتابخانه Continuous) به مدل اضافه کرده و خطوط را به ورودی و خروجی آن متصل کنید.
- خط ورودی را "vdot" و خروجی را "v" نام‌گذاری کنید. برای نام‌گذاری، بر روی فضای بالای هر خط، دبل کلیک کنید.



به علت اینکه شتاب (dv/dt) برابر مجموع نیروها تقسیم بر جرم می‌باشد، سیگنال ورودی را بر جرم تقسیم می‌کنیم:

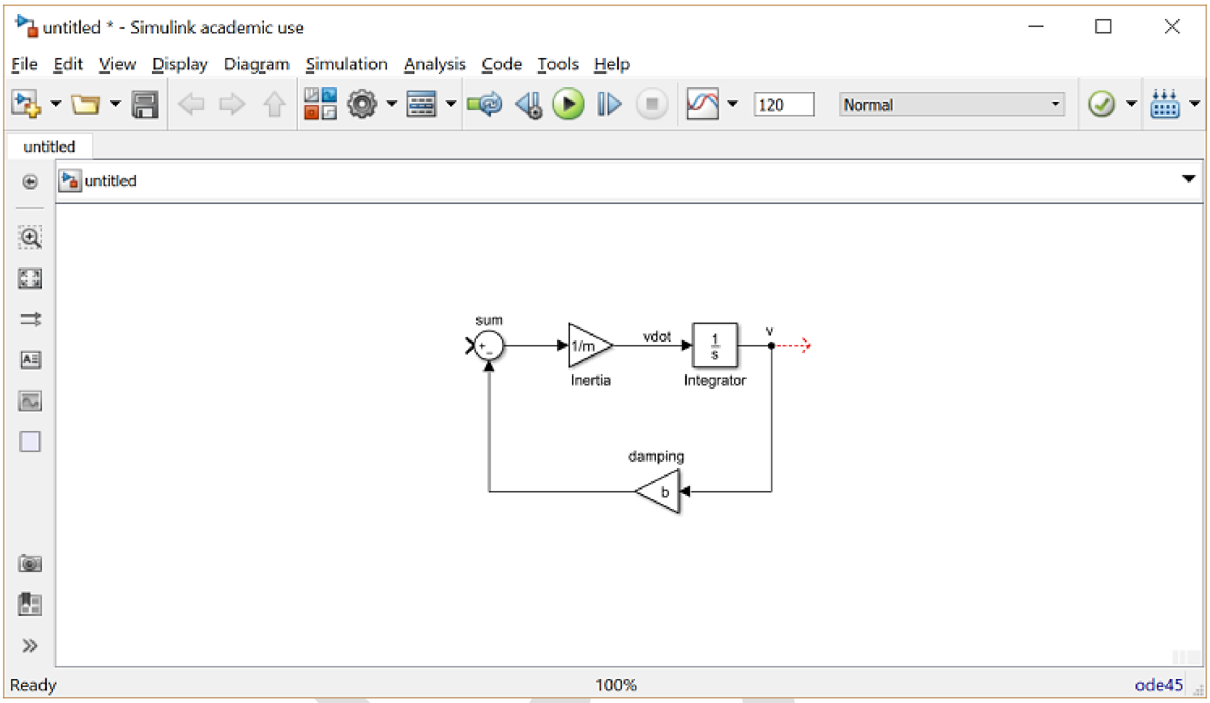
- یک بلوک Gain (از کتابخانه‌ی Math Operations) به ورودی بلوک Integrator متصل کرده و یک خط نیز به ورودی بلوک Gain متصل کنید.
- مقدار بلوک Gain را با دبل کلیک به "1/m" تغییر دهید.
- لیبل بلوک Gain را با کلیک بر روی کلمه "Gain" در پایین بلوک، به "inertia" تغییر دهید.



حال نوبت به اضافه کردن نیروهایی است که در معادله (۱) آورده شده است. ابتدا نیروی میرایی را اضافه می‌کنیم.

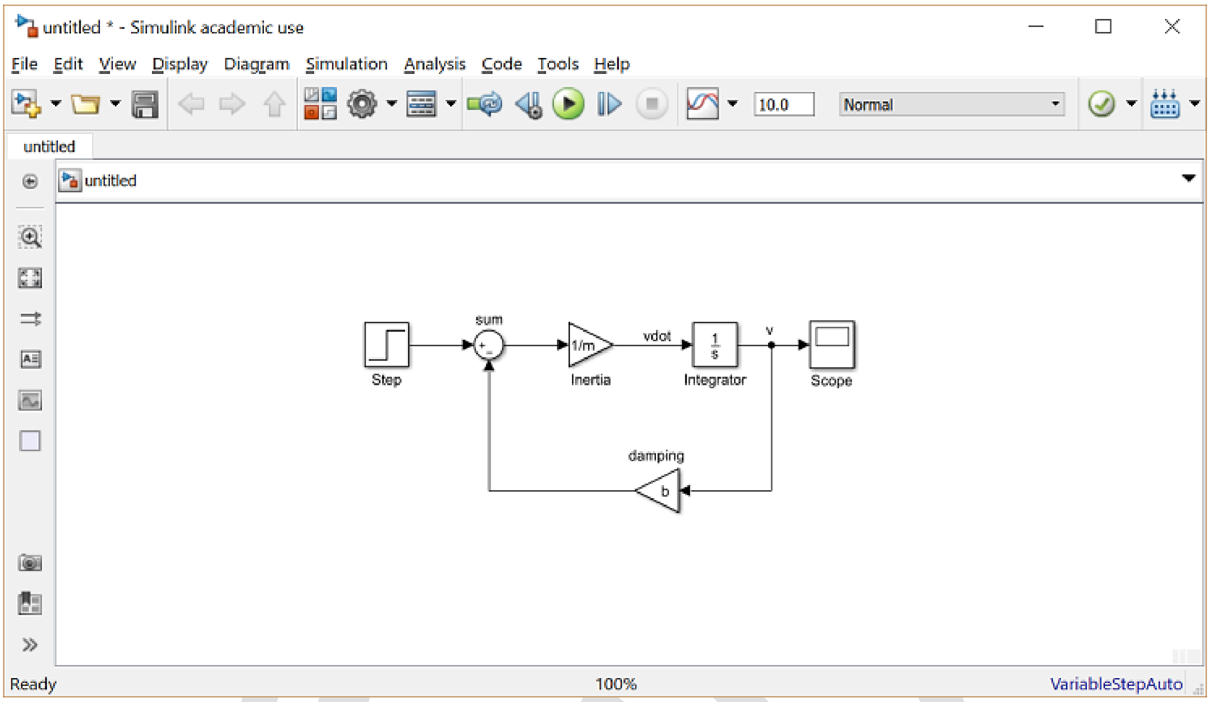
- یک بلوک Sum (از کتابخانه Math Operations) به خط ورودی به بلوک Gain اینرسی وصل کنید.
- علائم بلوک Sum را به "+-" تغییر دهید.

- یک بلوک Gain در زیر بلوک اینرسی قرار داده و آنرا با کلیک ماوس انتخاب کنید و از منوی **Rotate & Flip** گزینه‌ی **Flip Block** (یا کلید ترکیبی **Ctrl+I**) را انتخاب کنید تا بلوک Gain دوران پیدا کند.
- مقدار بلوک را برابر "b" قرار داده و نام آنرا به "damping" تغییر دهید.
- یک خط (با نگه داشتن کلید Ctrl) از خروجی بلوک Integrator ایجاد کرده و آنرا به ورودی بلوک بهره‌ی damping متصل کنید.
- یک خط از خروجی بلوک بهره‌ی damping به ورودی منفی بلوک Sum متصل کنید.

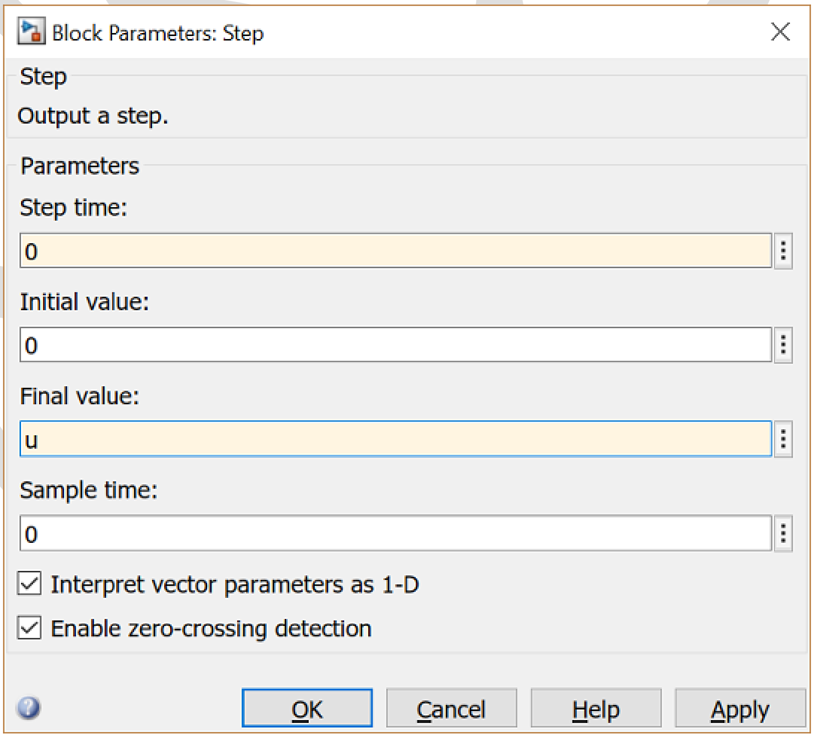


نیروی دوم وارد بر جرم، ورودی کنترل u می‌باشد. برای این نیرو از ورودی پله استفاده می‌نماییم.

- یک بلوک Step (از کتابخانه Sources) برداشته و آنرا به ورودی مثبت بلوک Sum متصل کنید.
- برای مشاهده‌ی سرعت خروجی، یک بلوک Scope (از کتابخانه Sinks) برداشته و به خروجی Integrator متصل کنید.



• برای تولید ورودی پله‌ی مناسب با مقدار ۵۰۰ در زمان صفر، بر روی بلوک Step دبل کلیک کرده و Step Time را برابر "0" و Final Value را برابر "u" قرار دهید.

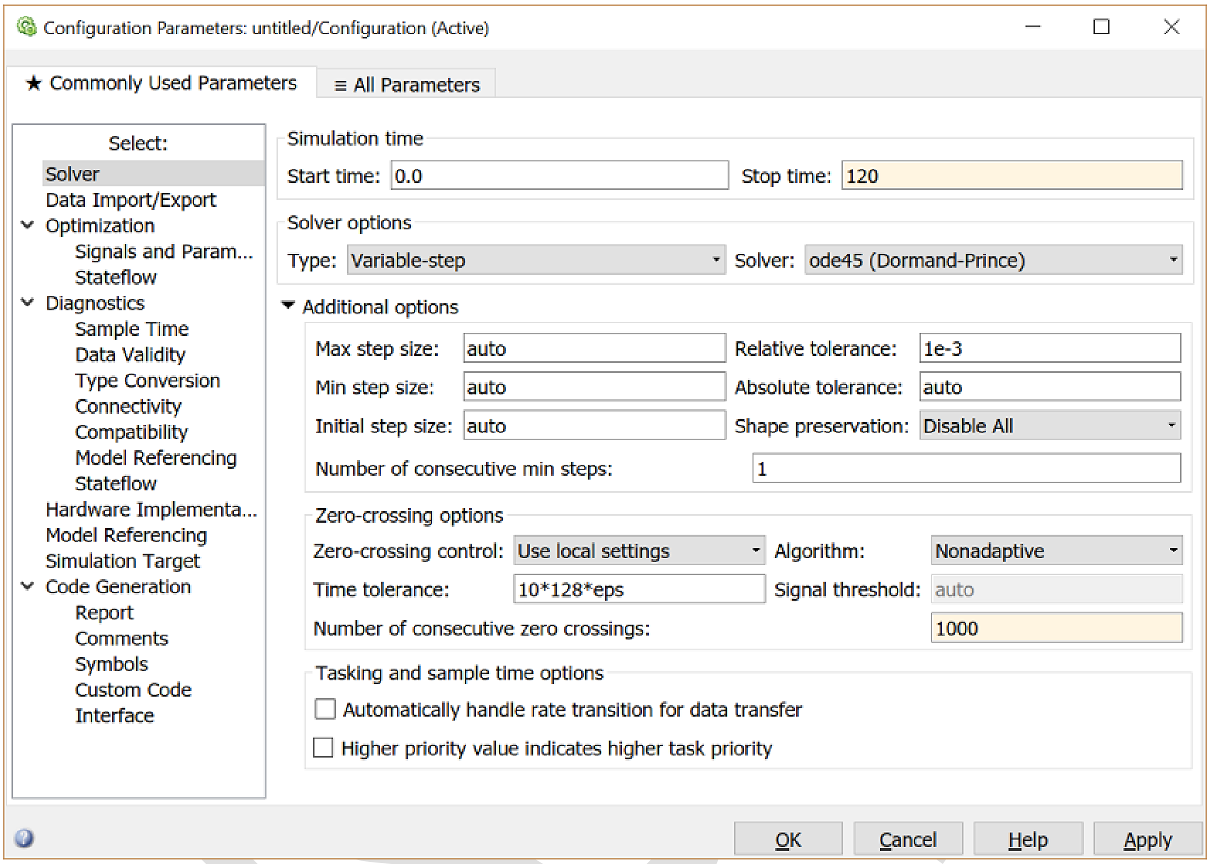


می‌توانید مدل کامل سیستم را از سی‌دی کپی کرده و آنرا بر روی کامپیوتر خود ذخیره نمایید.

پاسخ حلقه باز

برای شبیه‌سازی این سیستم، ابتدا باید زمان شبیه‌سازی مناسب را تنظیم نماییم.

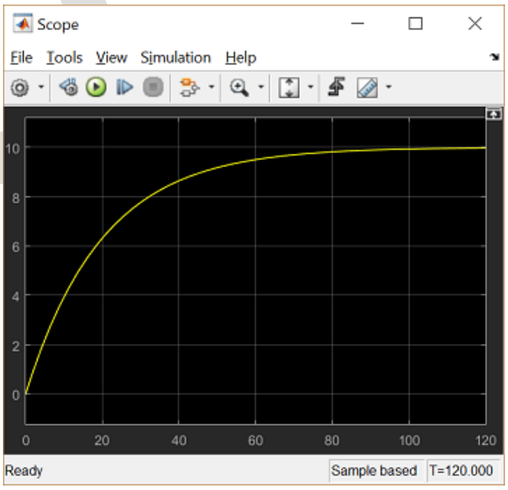
• گزینه‌ی Parameters را از منوی Simulation انتخاب کرده و در فیلد Stop Time مقدار "120" را وارد کنید. ۱۲۰ ثانیه زمان کافی برای مشاهده‌ی پاسخ حلقه باز می‌باشد.



حال باید پارامترهای فیزیکی سیستم را وارد نماییم. دستورات زیر را در محیط متلب وارد کنید:

```
m = 1000;
b = 50;
u = 500;
```

شبیه‌سازی را اجرا نمایید (با کلید ترکیبی **Ctrl+T** یا انتخاب **Run** از منوی **Simulation**). پس از اتمام شبیه‌سازی، خروجی زیر را مشاهده می‌نمایید:



با توجه به نمودار بالا، هدف ما بهبود پاسخ سیستم کنترل کروز می‌باشد. مدل ساخته شده در این بخش را در بخش بعد برای طراحی و تحلیل کنترلر استفاده خواهیم نمود.

بخش نهم: طراحی کنترلر در سیمولینک

فهرست مطالب بخش

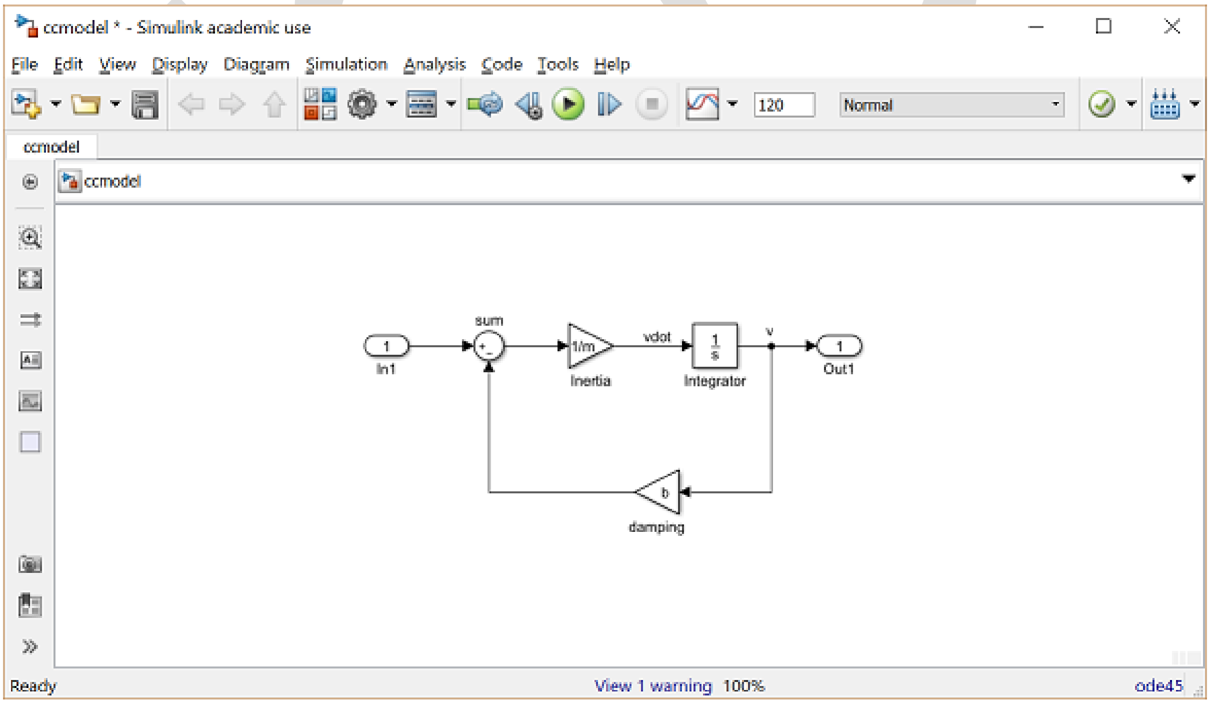
- استخراج یک مدل خطی به متلب
- پیاده‌سازی کنترل PI
- پاسخ حلقه بسته

در بخش قبل مدل سیمولینک سیستم کنترل کروز را تشکیل دادیم. این مدل را می‌توانید از سی‌دی بر روی کامپیوتر خود ذخیره نمایید. در این بخش، نشان می‌دهیم که چگونه یک کنترلر فیدبک را در سیمولینک پیاده‌سازی کرده و عملکرد سیستم را بهبود دهیم.

استخراج یک مدل خطی به متلب

سیمولینک این قابلیت را به کاربر می‌دهد تا از مدل ساخته شده در سیمولینک، یک مدل خطی (به هر دو فرم فضای حالت و تابع تبدیل) به متلب استخراج کند. برای اینکار از بلوک‌های In1 و Out1 و تابع متلب `linmod` استفاده خواهیم نمود.

- بلوک Step و Scope را به ترتیب با بلوک‌های In1 و Out1 جایگزین کنید (این بلوک‌ها در کتابخانه Ports & Subsystems می‌باشند). بدین صورت ورودی و خروجی سیستم برای فرآیند استخراج تعریف می‌شود.



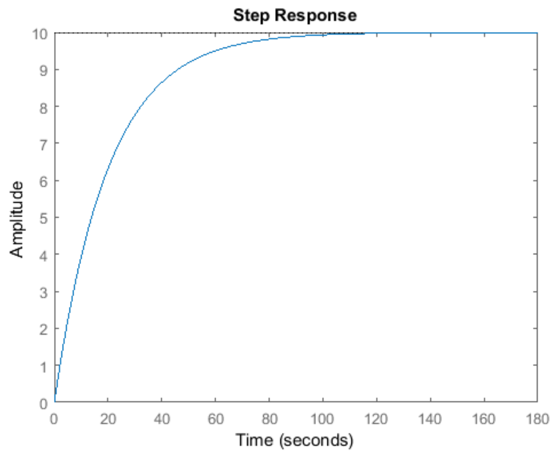
مدل خود را با نام "ccmodel.slx" ذخیره نمایید. متلب از فایل مدل ذخیره شده برای استخراج مدل خطی استفاده می‌نماید. در محیط متلب، دستورات زیر را وارد نمایید:

```
m = 1000;
b = 50;
u = 500;

[A,B,C,D] = linmod('ccmodel')
cruise_ss = ss(A,B,C,D);
```

```
A =

   -0.0500

B =

   1.0000e-03

C =

     1

D =

     0
```

برای تایید از صحت استخراج مدل، پاسخ پله حلقه باز را برای تابع تبدیل استخراج شده در متلب محاسبه می‌نماییم. صورت تابع تبدیل را در ۵۰۰ ضرب کرده تا ورودی پله ۵۰۰ نیوتن را ایجاد نماییم. دستور زیر را در متلب وارد کنید:

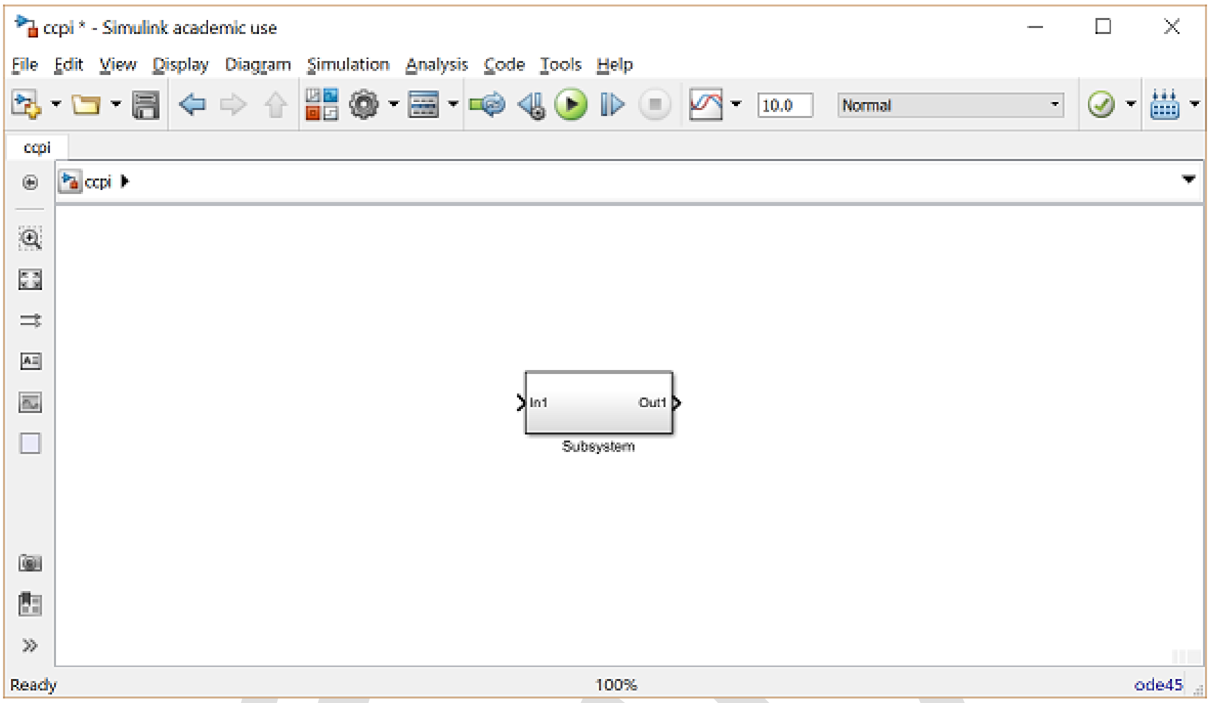
```
step(u*cruise_ss)
```



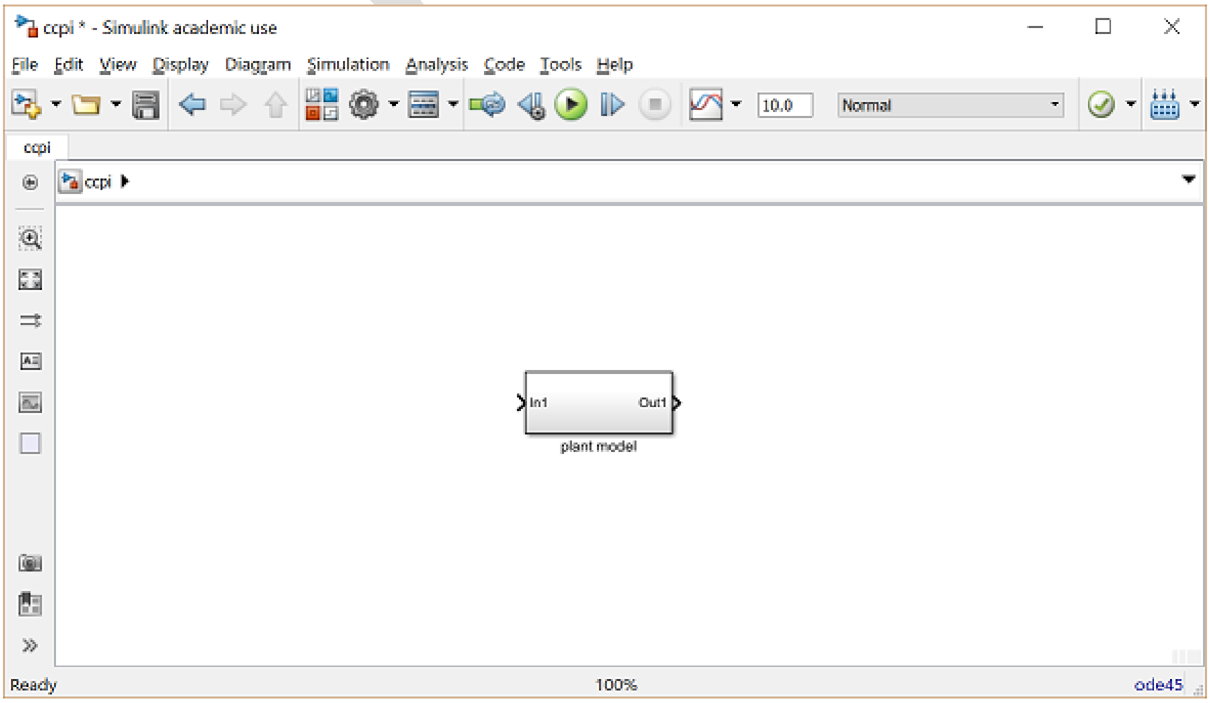
پیاده‌سازی کنترل PI

در **بخش سوم: طراحی کنترلر PID** یک کنترلر PI با ضرایب $K_p = 800$ و $K_i = 40$ برای دستیابی به پاسخ مطلوب، طراحی گردید. در این بخش این کار را در سیمولینک برای سیستم حلقه باز پیاده خواهیم کرد.

- یک پنجره‌ی مدل جدید ایجاد کنید.
- یک بلوک Subsystem از کتابخانه‌ی Ports & Subsystems در درون مدل جدید خود قرار دهید.

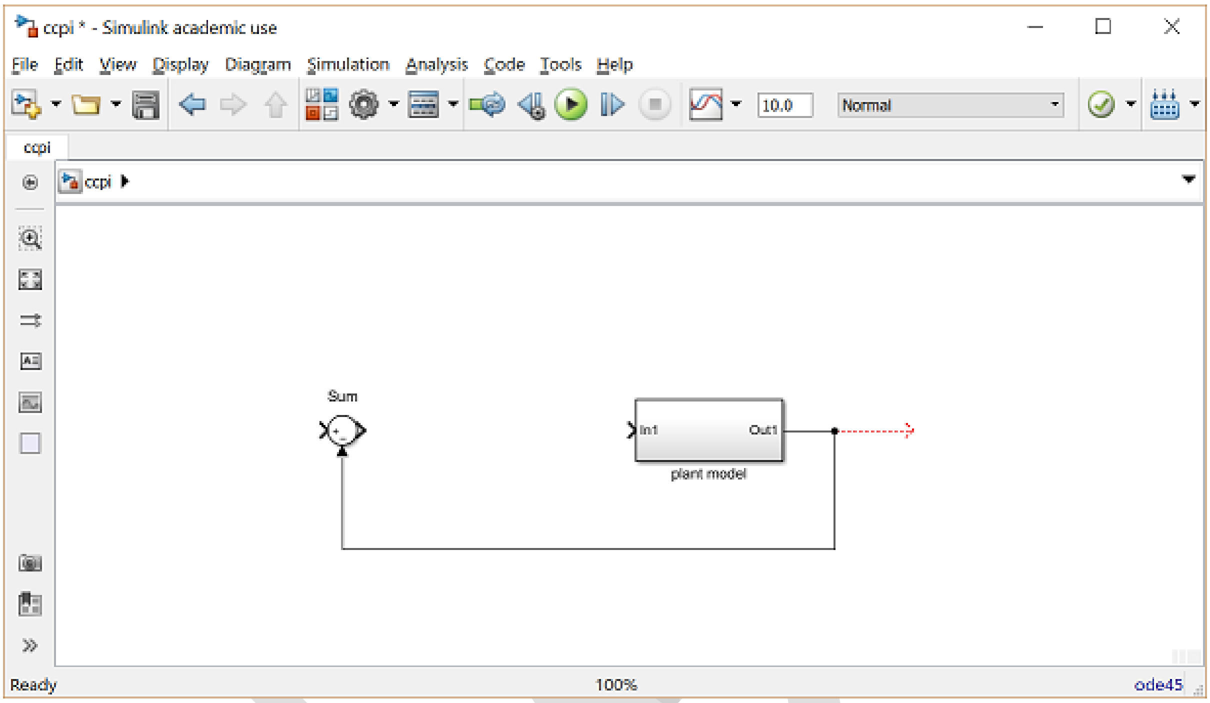


- بر روی این بلوک دبل کلیک کرده تا پنجره‌ی خالی محتویات Subsystem نمایش داده شود (که در حال حاضر خالی می‌باشد).
- مدل سیستم کنترل کروز پیشین خود که با نام ccmodel.slx ذخیره نمودید را باز نمایید.
- از منوی **Edit** گزینه‌ی **Select All** (یا کلید ترکیبی **Ctrl+A**) را انتخاب کرده و از منوی **Edit** گزینه‌ی **Copy** (یا کلید ترکیبی **Ctrl+C**) را انتخاب کنید.
- حال پنجره‌ی Subsystem خالی مدل جدید را انتخاب کرده و از منوی **Edit** گزینه‌ی **Paste** (یا کلید ترکیبی **Ctrl+V**) را انتخاب کنید. حال باید سیستم اصلی خود را در پنجره‌ی Subsystem جدید مشاهده کنید. این پنجره را ببندید.
- حال باید ترمینال‌های ورودی و خروجی را بر روی بلوک Subsystem مشاهده کنید. این بلوک را "plant model" نامگذاری کنید.



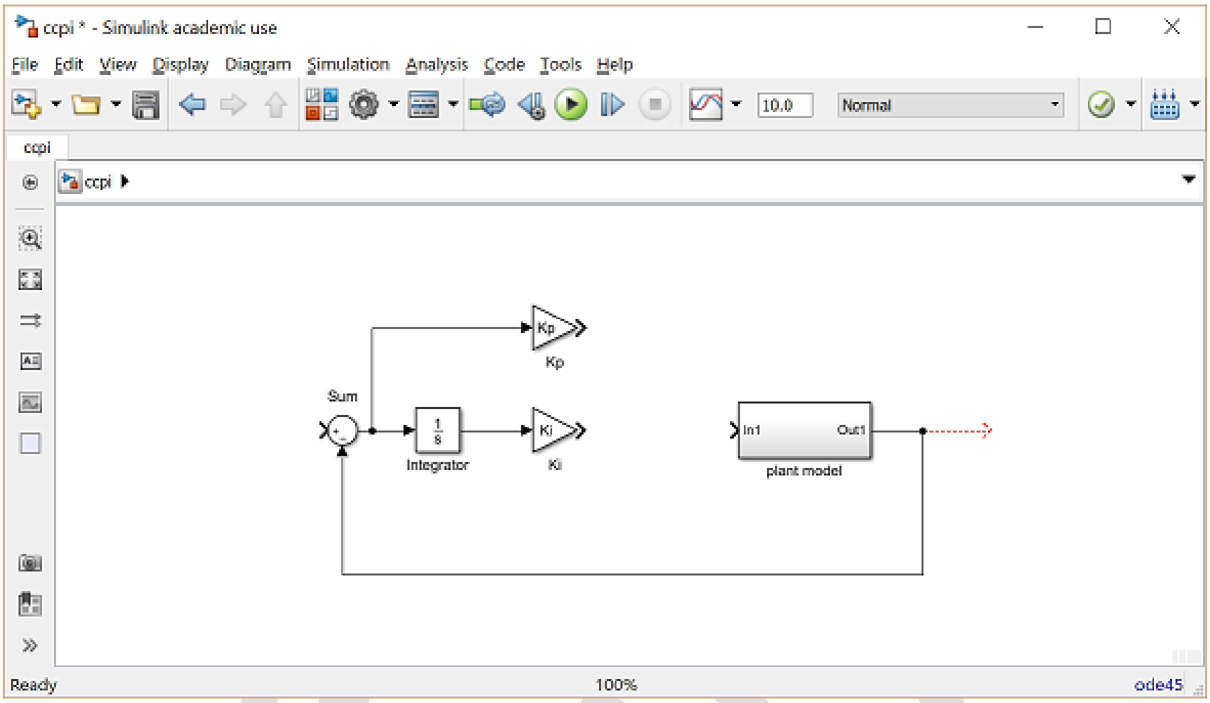
در قدم بعد به ساخت کنترلر PI برای مدل سیستم می‌پردازیم. ابتدا از خروجی سیستم فیدبک می‌گیریم:

- یک خط از خروجی سیستم ایجاد کنید.
- یک بلوک Sum با ورودی "-+" قرار دهید.
- خط ایجاد شده از خروجی سیستم را به ورودی منفی بلوک Sum متصل کنید.



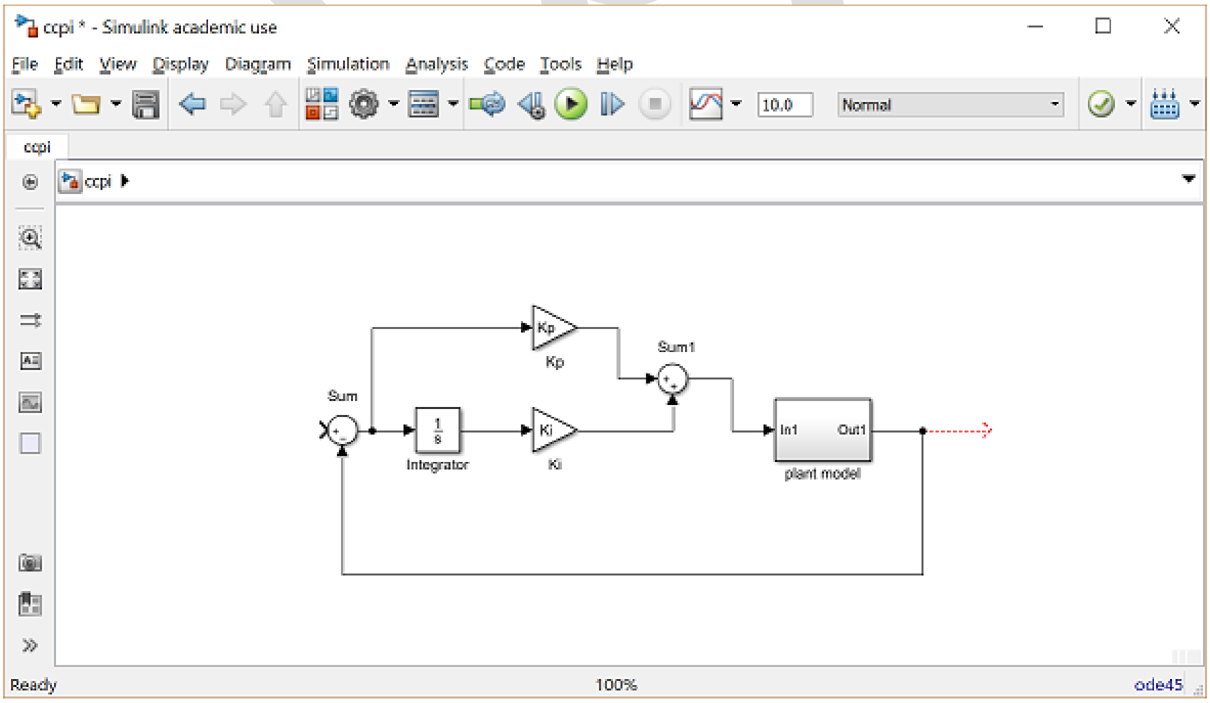
خروجی بلوک Sum، سیگنال خطا را تشکیل می‌دهد. با استفاده از این سیگنال، جملات تناسبی و انتگرالی را می‌سازیم.

- یک بلوک Integrator را بعد از بلوک Sum قرار داده و آنها را به یکدیگر متصل کنید.
- یک بلوک Gain را به عنوان بهره انتگرالی، بعد از بلوک Integrator قرار داده و آنها را به یکدیگر متصل کنید.
- بهره‌ی انتگرال‌گیر را "Ki" نامگذاری کرده و و مقدار بهره‌ی آنرا برابر "Ki" قرار دهید.
- یک بلوک بهره‌ی جدید در مدل خود قرار داده و آنرا به خط خروجی از بلوک Sum متصل کنید.
- این بلوک را "Kp"نامیده و مقدار آنرا برابر "Kp" قرار دهید.



حال جملات تناسبی و انتگرالی را به یکدیگر اضافه کرده و حاصل جمع آنها را به سیستم اعمال می‌کنیم.

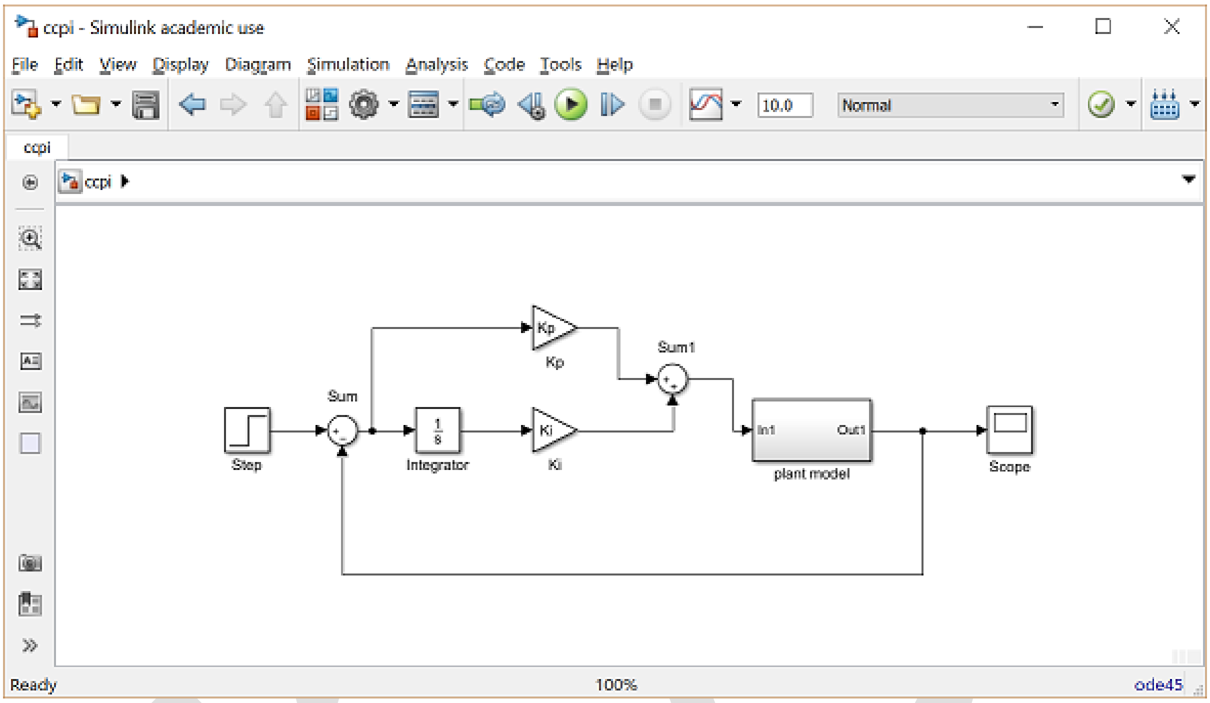
- یک بلوک Sum بین بلوک Ki و بلوک سیستم قرار داده و دو خروجی بلوک‌های Gain را به ورودی‌های بلوک Sum متصل کنید.
- خروجی بلوک Sum را به ورودی بلوک سیستم متصل کنید.



در نهایت ورودی پله را به سیستم اعمال کرده و خروجی را در بلوک Scope مشاهده می‌نماییم.

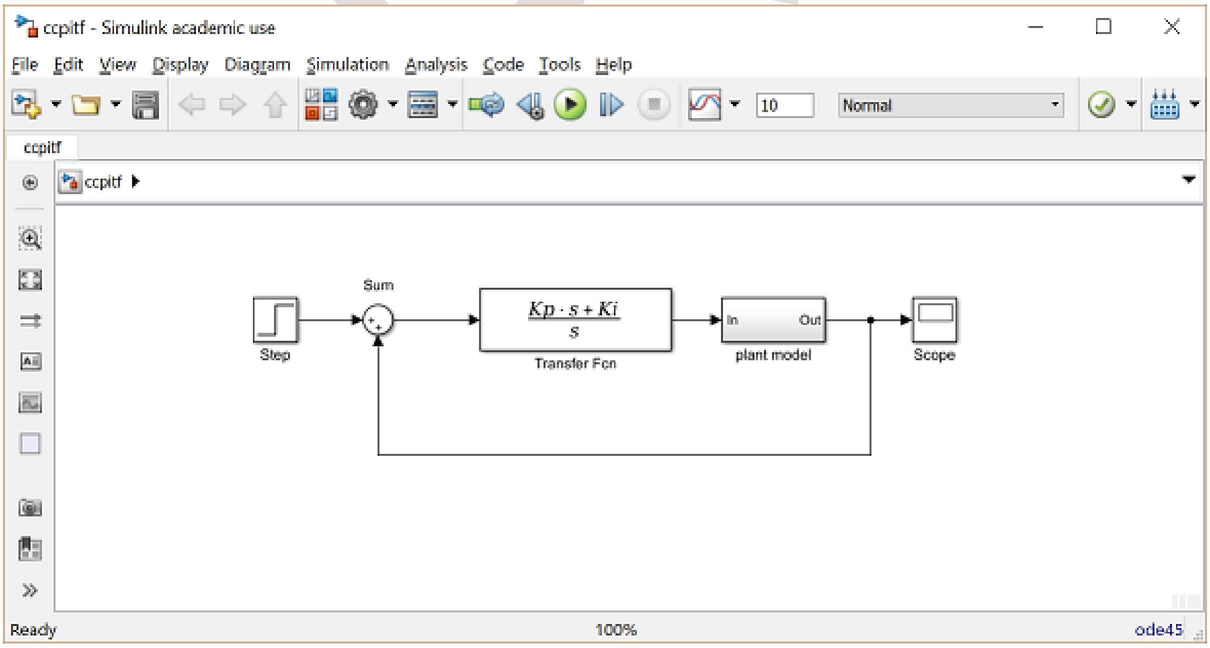
- یک بلوک Step را به ورودی آزاد بلوک Sum فیدبک متصل کنید.
- یک بلوک Scope را به خروجی سیستم متصل کنید.

- بر روی بلوک Step دبل کلیک کرده و Step Time را برابر "0" و Final Value را برابر "u" قرار دهید. با این کار امکان تغییر اندازه‌ی پله از خارج از سیمولینک را داریم.



می‌توانید مدل سیستم حلقه بسته را از داخل سی‌دی کپی نمایید.

در این مثال یک کنترلر PI را با استفاده از بلوک‌های پایه ساختیم. روش دیگری که می‌توان استفاده کرد، استفاده از بلوک Transfer Fcn (از کتابخانه Continuous) برای ساخت کنترلر PI در یک قدم مانند شکل زیر می‌باشد:



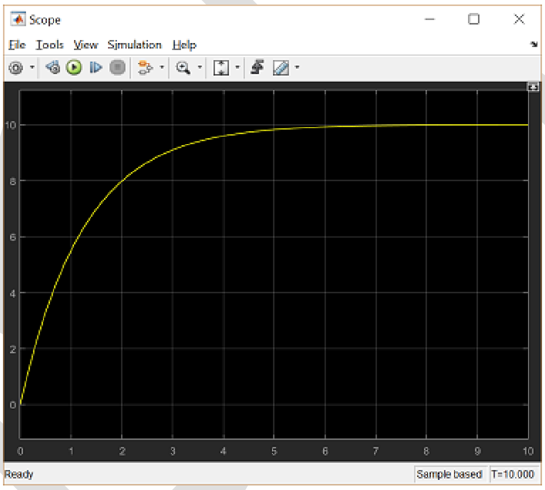
می‌توانید به این مدل نیز از داخل سی‌دی دسترسی داشته باشید.

پاسخ حلقه بسته

برای شبیه‌سازی این سیستم ابتدا باید زمان شبیه‌سازی مناسب تعیین شود. از منوی **Simulation** گزینه‌ی **Parameters** را انتخاب نمایید و در فیلد Stop Time مقدار "10" را وارد کنید. نیاز طراحی عبارتست از زمان نمو کمتر از ۵ ثانیه، پس می‌توان برای ۱۰ ثانیه شبیه‌سازی را انجام داده و خروجی را مشاهده کرد. پیش از آن باید پارامترهای فیزیکی را تعیین کنیم. دستورات زیر را در محیط متلب وارد کنید:

```
m = 1000;
b = 50;
r = 10;
Kp = 800;
Ki = 40;
```

شبیه‌سازی را اجرا کنید (با کلید ترکیبی **Ctrl+T** یا انتخاب **Run** از منوی **Simulation**). پس از اتمام شبیه‌سازی، خروجی زیر را مشاهده می‌کنیم:



فصل پنجم: هواپیما

بخش اول: مدل‌سازی سیستم

فهرست مطالب بخش

- تجهیز فیزیکی و معادلات سیستم
- تابع تبدیل و مدل فضای حالت
- الزامات طراحی
- نمایش در محیط متلب

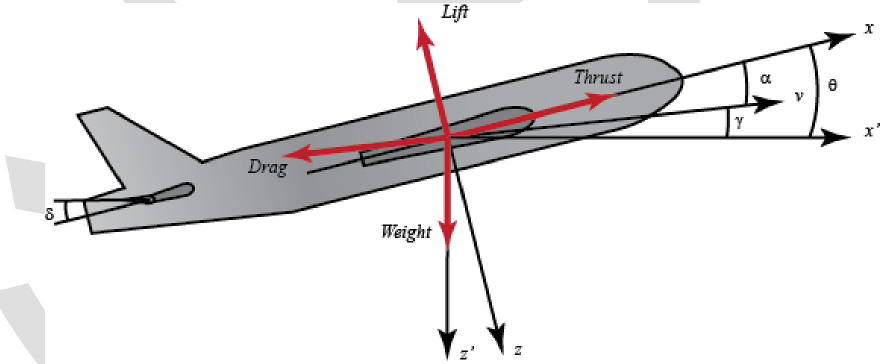
دستورهای کلیدی متلب در این بخش:

```
ss, tf
```

تجهیز فیزیکی و معادلات سیستم

معادلات حاکم بر هواپیما شامل شش معادله‌ی پیچیده‌ی غیر خطی کوپله است. اما با فرض شرایط خاصی می‌تواند به فرم خطی و دی کوپله و به معادلات طولی و جانبی در آید. زاویه‌ی حمله‌ی[۳۰] هواپیما با دینامیک طولی تعریف می‌شود. در این مثال یک خلبان خودکار جهت کنترل زاویه‌ی حمله‌ی هواپیما طراحی می‌شود.

در شکل زیر محورهای اصلی و نیروهای وارد بر هواپیما مشخص شده‌اند:



فرض می‌کنیم که هواپیما در یک ارتفاع و سرعت ثابت قرار دارند. در نتیجه نیروهای پیش‌رانش[۳۱] و برآر[۳۲] و پسا[۳۳] و وزن همدیگر را بالانس می‌کنند. همچنین فرض بر این است که با تغییر زاویه‌ی حمله‌ی هواپیما، سرعت هواپیما تحت هیچ شرایطی تغییر نمی‌کند (فرض غیر واقعی اما ساده کننده). با این فرضیات معادلات سیستم به صورت زیر در می‌آیند:

$$\dot{\alpha} = \mu\Omega\sigma\left[-(C_L + C_D)\alpha + \frac{1}{\mu - C_L}q - (C_W sin\gamma)\theta + C_L\right] \qquad (۱)$$

$$\dot{q} = \frac{\mu\Omega}{2i_{yy}}\left[[C_M - \eta(C_L + C_D)]\alpha + [C_M + \sigma C_M(1 - \mu C_L)]q + (\mu C_W sin\gamma)\delta\right] \qquad (۲)$$

$$\dot{\theta} = \Omega q \qquad (۳)$$

---

[۳۰] Pitch – زاویه‌ی هواپیما حول محور عرضی.
[۳۱] Thrust
[۳۲] Lift
[۳۳] Drag

جهت به دست آوردن اطلاعات بیشتر در رابطه با استخراج معادلات بالا، به یکی از کتب مرجع مرتبط با هواپیما مراجعه نمایید.

در این سیستم ورودی زاویه بالابرنده[۳۴] δ و خروجی زاویه‌ی حمله‌ی هواپیما θ است.

مدل‌های تابع تبدیل و فضای حالت

قبل از پیدا کردن مدل‌های تابع تبدیل و فضای حالت، یک ساده‌سازی عددی برای معادلات بالا انجام می‌دهیم:

$$\dot{\alpha} = -0.313\alpha + 56.7q + 0.232\delta \qquad (۴)$$

$$\dot{q} = -0.0139\alpha - 0.426q + 0.0203\delta \qquad (۵)$$

$$\dot{\theta} = 56.7q \qquad (۶)$$

این مقادیر از هواپیمای تجاری بویینگ استخراج شده‌اند.

۱. تابع تبدیل

جهت به دست آوردن تابع تبدیل لازم است از معادلات بالا تبدیل لاپلاس بگیریم. جهت انجام این کار از شرایط اولیه صفر استفاده می‌کنیم. تبدیل لاپلاس معادلات بالا در زیر آورده شده است:

$$sA(s) = -0.313A(s) + 56.7Q(s) + 0.232\Delta(s) \qquad (۷)$$

$$sQ(s) = -0.0139A(s) - 0.426Q(s) + 0.0203\Delta(s) \qquad (۸)$$

$$s\Theta(\mathrm{s}) = 56.7\mathrm{Q}(\mathrm{s}) \qquad (۹)$$

بعد از چند عملیات جبری، به رابطه‌ی زیر می‌رسیم:

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s} \qquad (۱۰)$$

۲. فضای حالت

لازم به ذکر است که معادلات بالا خود به فرم متغیر حالت نوشته شده‌اند، پس می‌توانیم معادلات را به صورت ماتریسی زیر بازنویسی کنیم:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix} [\delta] \qquad (۱۱)$$

با توجه به خروجی مورد نظر ما که زاویه‌ی حمله‌ی هواپیما می‌باشد، معادله‌ی خروجی عباتست از:

$$y = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} \qquad (۱۲)$$

الزامات طراحی

مرحله‌ی بعد انتخاب چندین معیار طراحی است. در این مثال هدف طراحی کنترلر فیدبکی است که با دریافت دستور پله برای زاویه‌ی حمله، فراجهش سیستم کمتر از ۱۰٪، زمان نمو کمتر از ۲ ثانیه، زمان نشست کمتر از ۱۰ ثانیه و خطای حالت ماندگار کمتر از ۲٪ باشد. برای مثال اگر مقدار مرجع ۰/۲ رادیان (۱۱ درجه) باشد، زاویه‌ی حمله نباید از حدود

[۳۴] Elevator

۰/۲۲ رادیان بیشتر شود و باید در کمتر از ۲ ثانیه از ۰/۰۲ به ۰/۱۸ رسیده و در ۱۰ ثانیه با تلرانس ۲٪ به مقدار نهایی خود و بین مقادیر ۰/۱۹۶ تا ۰/۲۰۴ رادیان قرار بگیرد.

به طور خلاصه ملاحظات طراحی به شرح زیر است:

فراجهش کمتر از ۱۰٪

زمان نمو کمتر از ۲ ثانیه

زمان نشست کمتر از ۱۰ ثانیه

خطای حالت پایا ۲٪

نمایش در متلب

حال ما آماده‌ایم تا سیستم را در متلب نمایش دهیم. با دستور کد زیر در متلب، تابع تبدیل حلقه باز سیستم تعریف می‌گردد:

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s)
```

```
P_pitch =


      1.151 s + 0.1774

  -------------------------

  s^3 + 0.739 s^2 + 0.921 s



Continuous-time transfer function.
```

برای تعریف مدل فضای حالت آمده در بالا، کد زیر را وارد کنید:

```
A = [-0.313 56.7 0; -0.0139 -0.426 0; 0 56.7 0];

B = [0.232; 0.0203; 0];

C = [0 0 1];

D = [0];

pitch_ss = ss(A,B,C,D)
```

```
pitch_ss =


  A =

             x1        x2        x3
```

```
  x1   -0.313     56.7        0

  x2  -0.0139   -0.426        0

  x3        0     56.7        0


 B =

         u1

  x1   0.232

  x2  0.0203

  x3       0


 C =

      x1   x2   x3

  y1   0    0    1


 D =

      u1

  y1   0


Continuous-time state-space model.
```

نکته: می‌توان با کمک متلب مدل فضای حالت را به فرم تابع تبدیل (یا بالعکس) درآورد . جهت اطلاعات بیشتر به **پیوست دوم: تبدیل فرم نمایش سیستم** مراجعه نمایید.

بخش دوم: تحلیل سیستم

فهرست مطالب بخش

- پاسخ حلقه باز
- پاسخ حلقه بسته

دستورهای کلیدی متلب در این بخش:

```
tf, step, pole, zero, feedback, residue
```

معادلات دینامیکی در حوزه‌ی لاپلاس و تابع تبدیل حلقه باز برای زاویه‌ی حمله‌ی هواپیما به شکل زیر است:

$$sA(s) = -0.313A(s) + 56.7Q(s) + 0.232\Delta(s) \quad (۱)$$

$$sQ(s) = -0.0139A(s) - 0.426Q(s) + 0.0203\Delta(s) \quad (۲)$$

$$s\Theta(s) = 56.7Q(s) \quad (۳)$$

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s} \quad (۴)$$

همچنین الزامات طراحی برای ورودی مرجع پله، عبارتست از:

- فراجهش کمتر از ۱۰٪
- زمان برخاست کمتر از ۲ ثانیه
- زمان نشست کمتر از ۱۰ ثانیه
- خطای حالت پایا کمتر از ۲٪

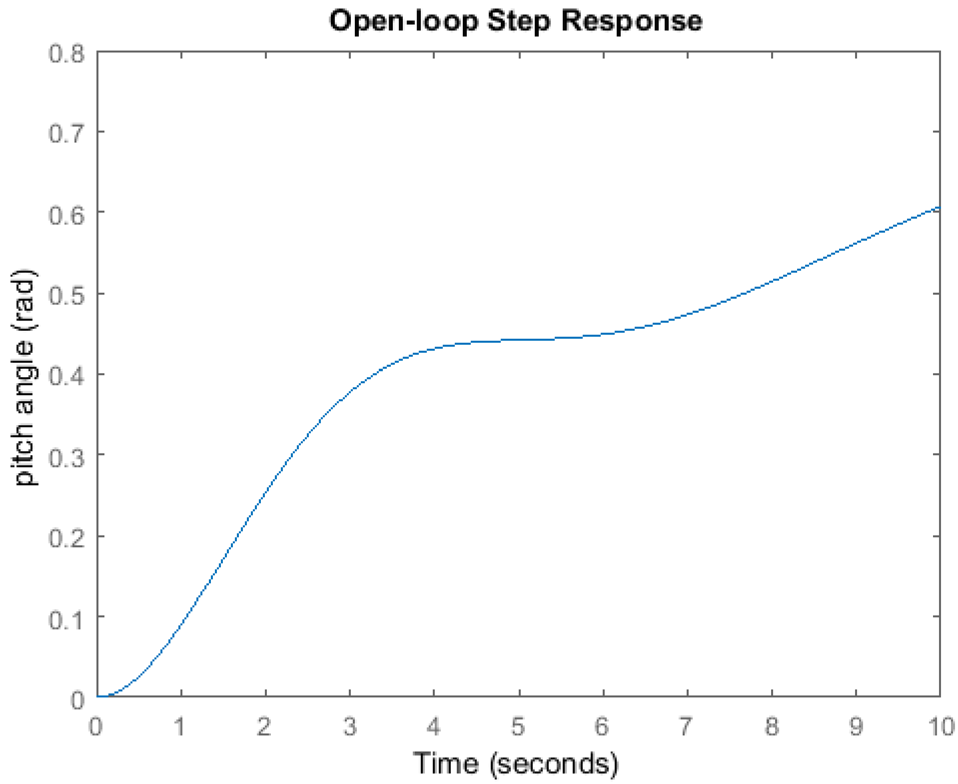
پاسخ حلقه باز

ابتدا یک امفایل ساخته و کد زیر را در آن وارد کنید:

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s);
```

حال به پاسخ سیستم حلقه باز بدون کنترل می‌پردازیم. یعنی از دستور step برای اعمال ورودی پله و تحلیل پاسخ پله‌ی حلقه باز استفاده می‌نماییم. برای اینکه به زاویه‌ی بالابرنده (δ) فرمان 0.2 رادیان (۱۱ درجه) را بدهیم از ضریب 0.2 استفاده می‌کنیم. کد زیر را به ام فایل خود اضافه کرده و آنرا در متلب اجرا کنید:

```
t = [0:0.01:10];

step(0.2*P_pitch,t);

axis([0 10 0 0.8]);

ylabel('pitch angle (rad)');
title('Open-loop Step Response');
```



از نمودار بالا مشخص است که پاسخ حلقه باز شرایط لازم را ارضا نمی‌کند. در واقع پاسخ حلقه باز سیستم ناپایدار می‌باشد. ناپایداری سیستم با استفاده از مکان یابی قطب‌های سیستم مشخص می‌شود و برای اینکار از دستور pole استفاده می‌نماییم:

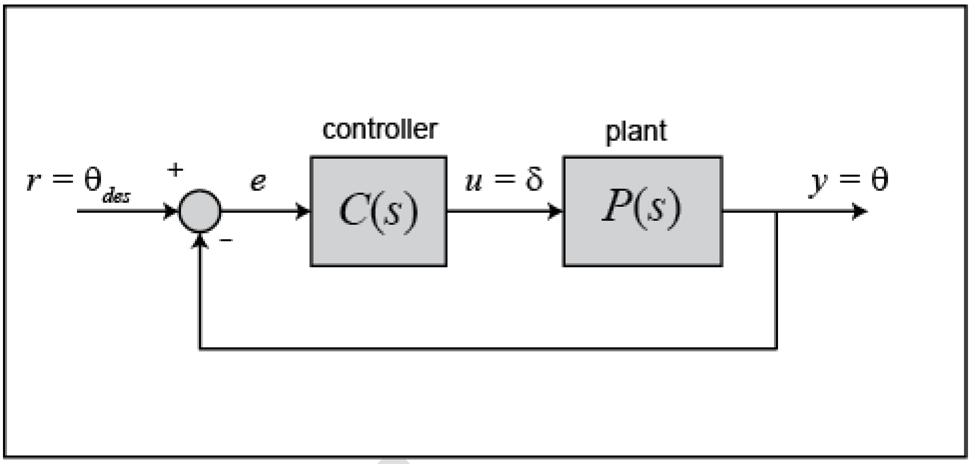
```
pole(P_pitch)
```

```
ans =

   0.0000 + 0.0000i

  -0.3695 + 0.8857i

  -0.3695 - 0.8857i
```

همانگونه که مشاهده می‌شود یکی از قطب‌ها بر روی محور موهومی و دو قطب دیگر در سمت چپ محور موهومی قرار دارند. قطب موجود بر روی محور موهومی نشان می‌دهد که پاسخ سیستم بی‌کران نمی‌شود اما به صفر نیز نمی‌رسد. اگر چه پاسخ حالت آزاد سیستم به صورت کران‌دار است اما پاسخ یک سیستم با یک قطب بر روی محور موهومی حتی اگر ورودی کراندار باشد می‌تواند بی‌کران شود. این موضوع با نتیجه‌ی به دست آمده سازگار می‌باشد. در این مثال قطب بر روی محور موهومی همانند یک انتگرال‌گیر عمل می‌کند. بنابراین با ورودی پله، خروجی سیستم به صورت نامحدود رشد می‌کند؛ همان گونه که انتگرال یک عدد ثابت هنگامی که حد بالای انتگرال بزرگ باشد، به بی‌نهایت میل می‌کند.

پاسخ حلقه بسته

جهت پایداری سیستم سیستم و رسیدن به معیارهای مطلوب، از یک کنترلر فیدبک استفاده می‌کنیم. شکل زیر شماتیک کنترلر را نمایش می‌دهد:



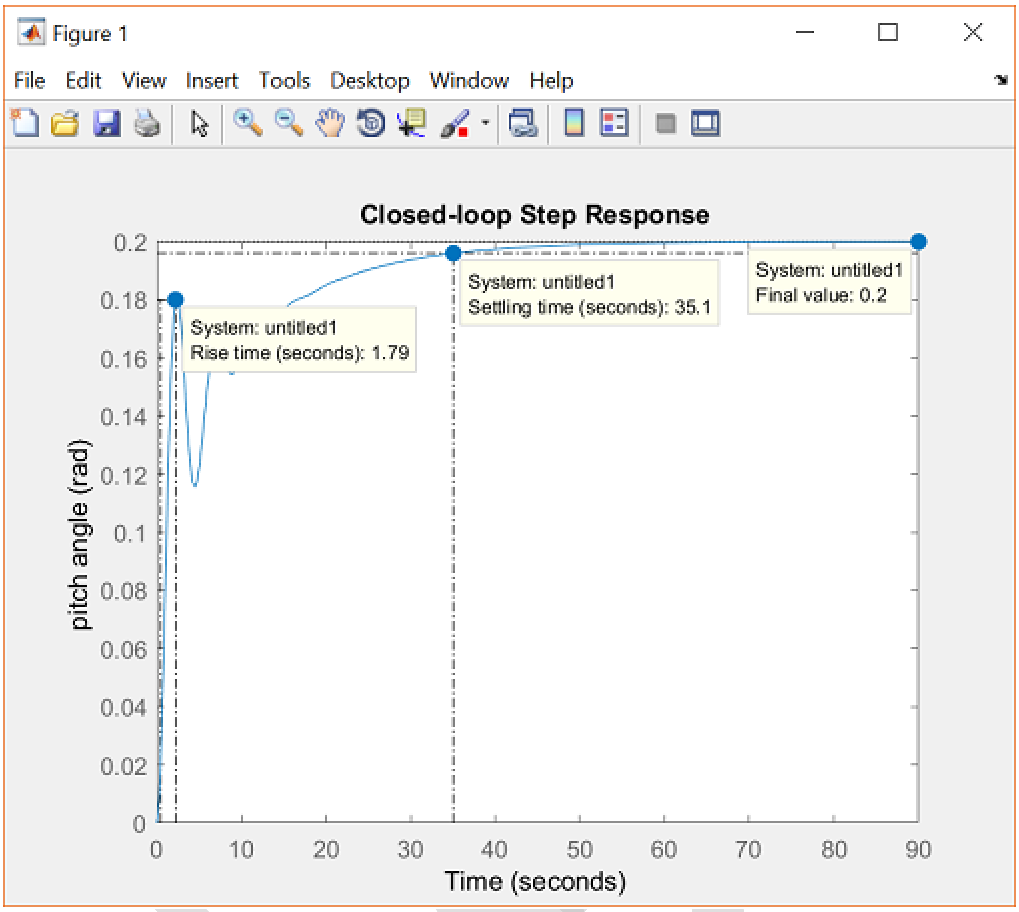
سیستم حلقه بسته بالا با کنترلر C(s) با استفاده از کد متلب زیر به راحتی تشکیل می‌شود:

```
sys_cl = feedback(P_pitch,1)
```

```
sys_cl =


          1.151 s + 0.1774

  ----------------------------------

  s^3 + 0.739 s^2 + 2.072 s + 0.1774



Continuous-time transfer function.
```

پاسخ سیستم حلقه بسته با اضافه کردن کدهای زیر به ام‌فایل خود به دست می‌آید. توجه شود که برای ورودی مرجع پله برای زاویه‌ی حمله، از ضریب 0.2 برای فرمان 0.2 رادیان (۱۱ درجه) استفاده شده است. با اجرای کد زیر، نمودار پاسخ پله‌ی سیستم به دست می‌آید که می‌توانید با راست کلیک بر روی نمودار و انتخاب منوی Characteristics، زمان نمو، زمان نشست و مقدار نهایی را بر روی نمودار نمایش دهید:

```
   step(0.2*sys_cl);

         ylabel('pitch angle (rad)');

         title('Closed-loop Step Response');
```



بررسی پاسخ حلقه بسته‌ی بالا نشان می‌دهد که اضافه کردن فیدبک به سیستم، آن را پایدار کرده است. به بیان دیگر فراجهش سیستم به صفر رسیده اما زمان نمو و زمان نشست سیستم در محدوده مجاز نمی‌باشند. خصوصیات پاسخ سیستم از مکان قطب‌ها و صفرهای تابع تبدیل سیستم مشخص می‌شوند. با استفاده از دستور pole و zero می‌توان قطب‌ها و صفرهای سیستم حلقه بسته را به دست آورد:

```
poles = pole(sys_cl)

zeros = zero(sys_cl)
```

```
poles =

  -0.3255 + 1.3816i

  -0.3255 - 1.3816i

  -0.0881 + 0.0000i

zeros =

   -0.1541
```

نتایج بالا نشان می‌دهد که تابع تبدیل حلقه بسته از مرتبه سوم به همراه یک صفر است. اکثر معادلات برای پیش‌بینی رفتار پاسخ پله‌ی سیستم برای یک سیستم زیر میرا از مرتبه دوم و بدون صفر است. پس نمی‌توانیم از این معادلات استفاده کنیم. اما از طرفی می‌توانیم خروجی سیستم را به حوزه‌ی زمان برگردانیم تا تابع زمانی برای پاسخ سیستم به دست

آورده و درکی از تاثیر موقعیت صفر و قطب‌های حلقه بسته بر روی پاسخ سیستم داشته باشیم. با فرض فرم $Y(s)/R(s)$ برای تابع تبدیل حلقه بسته، با فرض ورودی $R(s)$ برابر پله‌ای با اندازه‌ی 0.2، خروجی $Y(s)$ در حوزه‌ی لاپلاس را محاسبه می‌نماییم:

$$Y(s) = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 2.072s + 0.1774} R(s) = \frac{0.2(1.151s + 0.1774)}{s^4 + 0.739s^3 + 2.072s^2 + 0.1774s} \quad (۵)$$

می‌توانیم از تفکیک جزئی کسر برای تبدیل عبارت بالا به جملات ساده‌تر که قابل درک بوده و می‌توانیم آنرا از حوزه‌ی لاپلاس به حوزه‌ی زمان تبدیل کنیم استفاده نماییم. ابتدا از دستور zpk خروجی سیستم را به عبارات ساده‌تر تبدیل می‌نماییم:

```
R = 0.2/s;

Y = zpk(sys_cl*R)
```

```
Y =


          0.2302 (s+0.1541)

  -------------------------------------

  s (s+0.08805) (s^2 + 0.6509s + 2.015)



Continuous-time zero/pole/gain model.
```

با توجه به نتیجه‌ی بالا، مخرج خروجی Y(s) را می‌توان به صورت یک جمله‌ی مرتبه اول و یک جمله‌ی مرتبه دوم و یک قطب در مبدا فاکتور گرفت. جمله‌ی مرتبه اول دارای یک قطب حقیقی، جمله‌ی مرتبه دوم دارای یک جفت قطب مختلط می‌باشند. بنابراین می‌توانیم خروجی Y(s) را به شکل زیر نمایش دهیم:

$$Y(s) = \frac{0.2302(s + 0.1541)}{s(s + 0.08805)(s^2 + 0.6509s + 2.105)} = \frac{A}{s} + \frac{B}{s + 0.08805} + \frac{Cs + D}{s^2 + 0.6509s + 2.015} \quad (۶)$$

مقادیر ثابت A، B، C و D را می‌توان با محاسبات دستی یا با کمک دستور residue در متلب محاسبه نمود. در دستور `[r,p,k] = residue(num,den)` مقدار num و den بردارهای ضریب صورت و مخرج تابع تبدیل تفکیک شده می‌باشند. لازم به ذکر است که در انتهای بردار den باید یک درایه‌ی صفر در نظر گرفت زیرا این چندجمله‌ای دارای عدد ثابت نمی‌باشد.

```
[r,p,k] = residue(0.2*[1.151 0.1774],[1 0.739 2.072 0.1774 0])
```

```
r =

  -0.0560 + 0.0160i

  -0.0560 - 0.0160i

  -0.0879 + 0.0000i
```

```
   0.2000 + 0.0000i

p =

  -0.3255 + 1.3816i

  -0.3255 - 1.3816i

  -0.0881 + 0.0000i

   0.0000 + 0.0000i

k =

     []
```

در نتیجه‌ی به دست آمده، r بردار باقی‌مانده‌های تفکیک جزئی کسر یا همان صورت کسرهای تفکیک شده، می‌باشد. بردار p شامل قطب‌های مربوط به هر یک از باقی‌مانده‌ها می‌باشد. در این مورد ماتریس k خالی است زیرا مرتبه‌ی صورت از مرتبه‌ی مخرج کوچکتر می‌باشد. با توجه به روابط بالا، مقدار A و B به ترتیب برابر 0.2, -0.0881 است. مقادیر C , D از ترکیب باقی‌مانده‌های مربوط به قطب‌های مختلط به شکل زیر به دست می‌آید:

```
[num,den] = residue(r(1:2),p(1:2),k);

tf(num,den)
```

```
ans =


   -0.1121 s - 0.08071

  ----------------------

  s^2 + 0.6509 s + 2.015


Continuous-time transfer function.
```

با توجه به بالا C=-0.1121 و D=-0.08071 است. در نتیجه تفکیک جزئی تابع تبدیل به صورت زیر به دست می‌آید:

$$Y(s) = \frac{0.2}{s} - \frac{0.0881}{s + 0.08805} - \frac{0.1121s + 0.08071}{s^2 + 0.6509s + 2.015} \qquad (۷)$$

با استفاده از یک جدول تبدیل لاپلاس می‌توان تبدیل لاپلاس معکوس جملات بالا را حساب کرده و پاسخ را در حوزه‌ی زمان به دست آورد. اگر جعبه ابزار Symbolic Math بر روی متلب شما نصب باشد میتوانید از دستور laplace برای به دست آوردن تبدیل لاپلاس معکوس استفاده کنید:

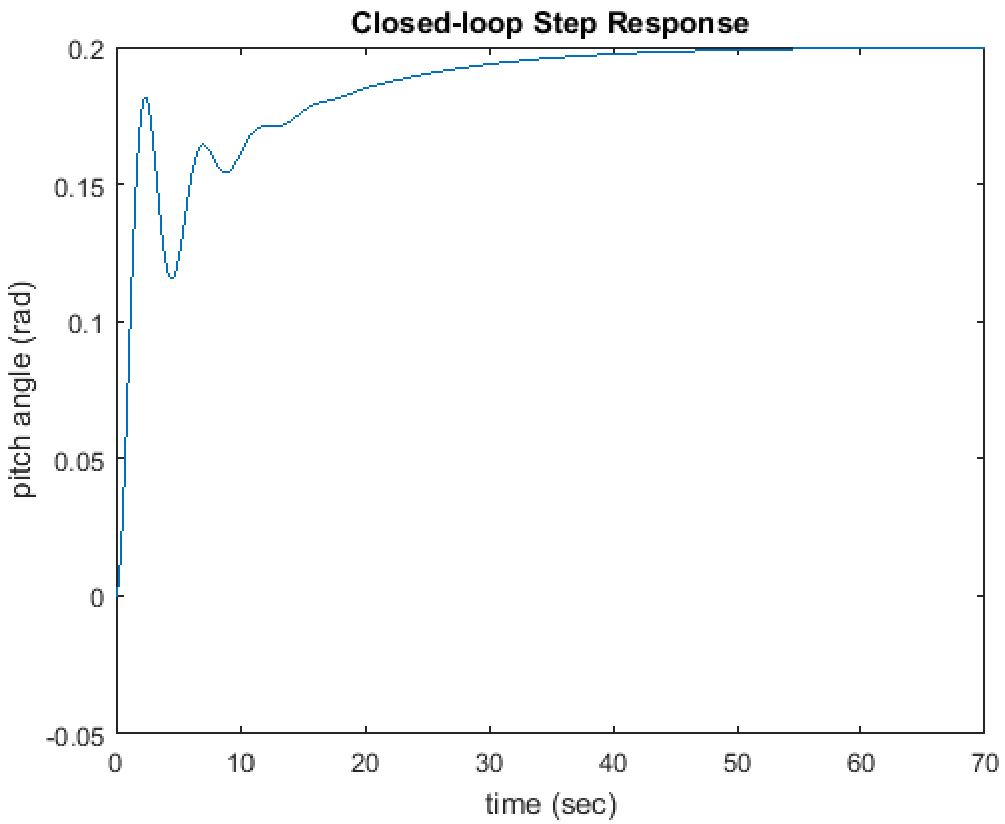
$$y(t) = 0.2 - 0.0881e^{-0.08805t} - e^{-0.3255t}(0.1121\cos(1.3816t) + 0.0320\sin(1.3815t)) \qquad (۸)$$

با بررسی عبارت به دست آمده، هر جمله مربوط به یکی از قطب‌های Y(s) می‌باشد که قسمت حقیقی قطب بیانگر تضغیف (یا رشد) نمایی پاسخ و قسمت موهومی بیانگر فرکانس نوسانات پاسخ می‌باشد. تاثیر صفرها بر روی ضرایب هر

جمله می‌باشد. به عبارت دیگر صفرها سهم نسبی هر پاسخ را مشخص می‌کنند. مثال بالا کمک می‌کند تا درکی از تاثیر صفر و قطب‌ها در حوزه‌ی لاپلاس بر روی رفتار سیستم در حوزه‌ی زمان داشته باشیم.

با اجرای کد زیر در متلب، نمودار پاسخ سیستم به دست می‌آید که با تقریب خوبی مشابه نمودار به دست آمده با دستور step می‌باشد.

```
t = [0:0.1:70];

y = 0.2 - 0.0881*exp(-0.08805*t) - exp(-
0.3255*t).*(0.1121*cos(1.3816*t)+0.0320*sin(1.3816*t));

plot(t,y)

xlabel('time (sec)');
ylabel('pitch angle (rad)');
title('Closed-loop Step Response');
```



شکل بالا نشان می‌دهد که سیستم حلقه بسته الزامات طراحی را ارضا نمی‌کند. در ادامه کنترلرهای مختلف جهت دریافت پاسخ مناسب را بررسی می‌کنیم.

بخش سوم: طراحی کنترلر PID

دستورهای کلیدی متلب در این بخش:

```
controlSystemDesigner
```

فهرست مطالب بخش

- کنترل تناسبی
- کنترل PI
- کنترل PID

با توجه به مسئله‌ی اصلی، تابع تبدیل حلقه باز برای دینامیک زاویه‌ی حمله‌ی هواپیما به شکل زیر است:

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s} \quad (۱)$$

در حالتی که ورودی زاویه‌ی جابجایی بالابرنده (δ) و خروجی زاویه‌ی حمله‌ی هواپیما (θ) می‌باشد.

جهت مشاهده‌ی سیستم مورد بررسی و روش استخراج معادله‌ی بالا، به **بخش اول: مدل‌سازی سیستم** مراجعه کنید.

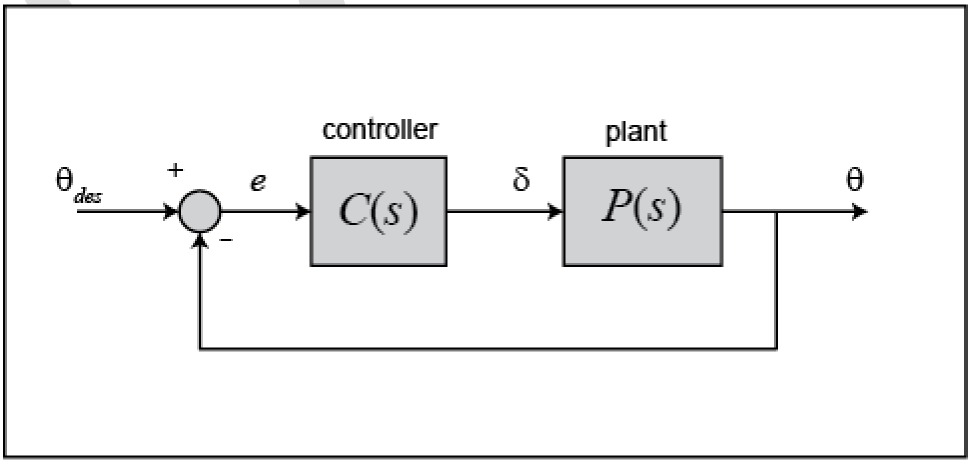
برای ورودی پله ۰/۲ رادیان، الزامات طراحی به صورت زیر است:

- فراجهش کمتر از ۱۰٪
- زمان نمو کمتر از ۲ ثانیه
- زمان نشست کمتر از ۱۰ ثانیه
- خطای حالت پایا کمتر از ۲٪

با یادآوری از **فصل دوم - بخش سوم: طراحی کنترلر PID** می‌دانیم که تابع تبدیل کنترلر PID به شکل زیر است:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \quad (۲)$$

ما ترکیبی از کنترل تناسبی K_p، انتگرالی K_i، مشتق K_d را در ساختار فیدبک واحد جهت دستیابی به پاسخ مورد نظر به کار می‌گیریم.



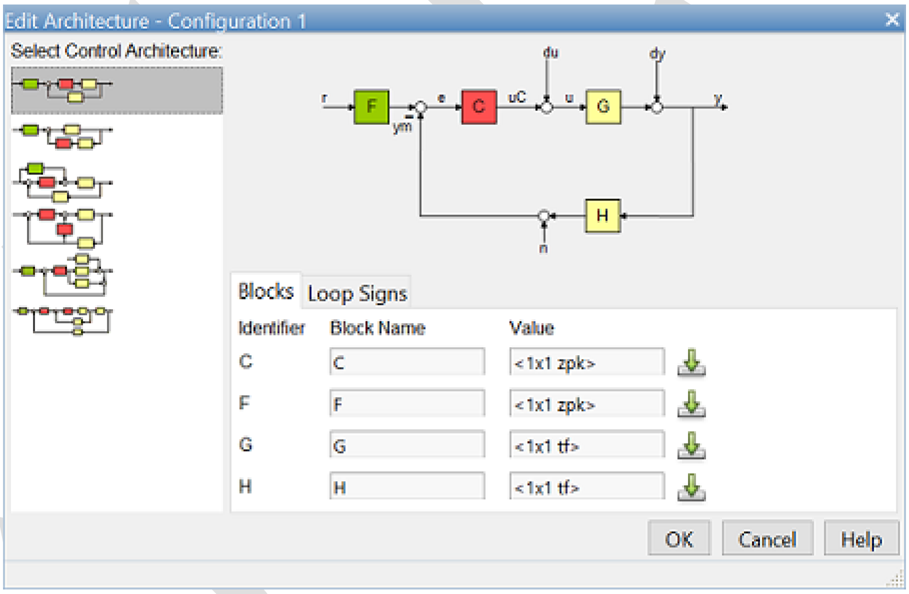
ما از امکانات تنظیم خودکار کنترلر در متلب با استفاده از Control System Designer جهت تنظیم ضرایب کنترلر PID خود بهره می‌بریم. ابتدا کد زیر را وارد کنید تا تابع تبدیل سیستم خود را تعریف کنیم:

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s);
```

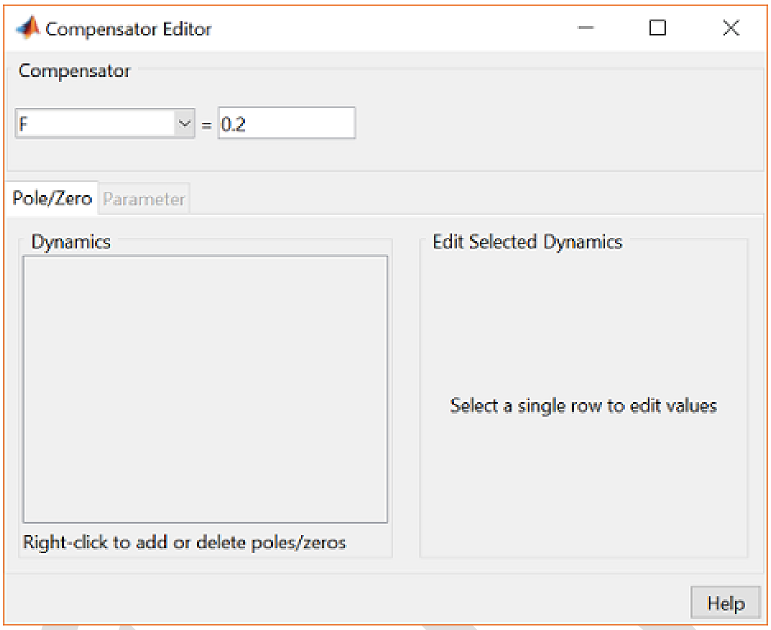
کنترل تناسبی

در ابتدا با طراحی یک کنترل تناسبی به فرم $C(s) = K_p$ شروع می‌نماییم. ابزار Control System Design را با تایپ کردن `controlSystemDesigner(P_pitch)` در قسمت پنجره‌ی دستور متلب باز می‌کنیم. پنجره‌ی مربوطه در ابتدا شامل مکان هندسی ریشه‌ها، دیاگرام بودی حلقه باز و پاسخ پله‌ی حلقه بسته برای تابع تبدیل داده شده به همراه کنترلر $C(s) = 1$ می‌باشد.

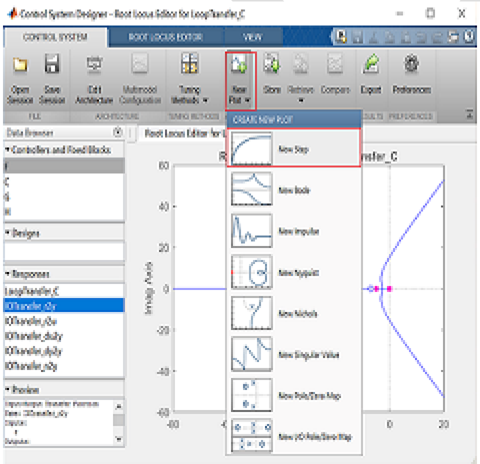
دکمه Edit Architecture در پنجره‌ی Control System Designer ساختار سیستم کنترلی را نمایش می‌دهد. ساختار پیش‌فرض با ساختار مورد نظر ما مطابقت دارد.

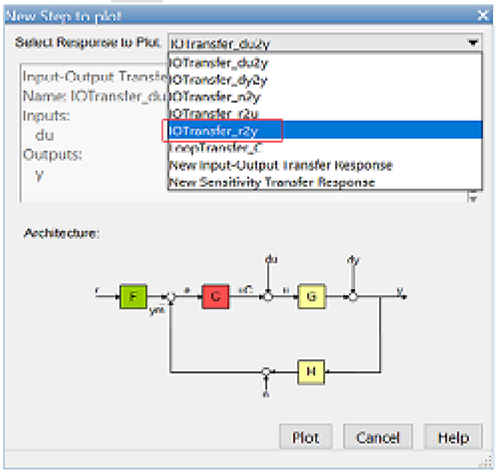


از آنجا که مقدار مرجع ما پله با اندازه‌ی ۰/۲ رادیان است، مقدار بلوک پیش‌جبران‌ساز F(s) را بر روی ۰/۲ تنظیم می‌کنیم. این کار از پنجره‌ی Compensator Editor که با راست کلیک بر روی شکل و انتخاب گزینه‌ی Edit Compensator باز می‌شود، انجام می‌شود . در بخش Compensator از منوی کشویی F را انتخاب کنید و مقدار آن را ۰/۲ قرار دهید.

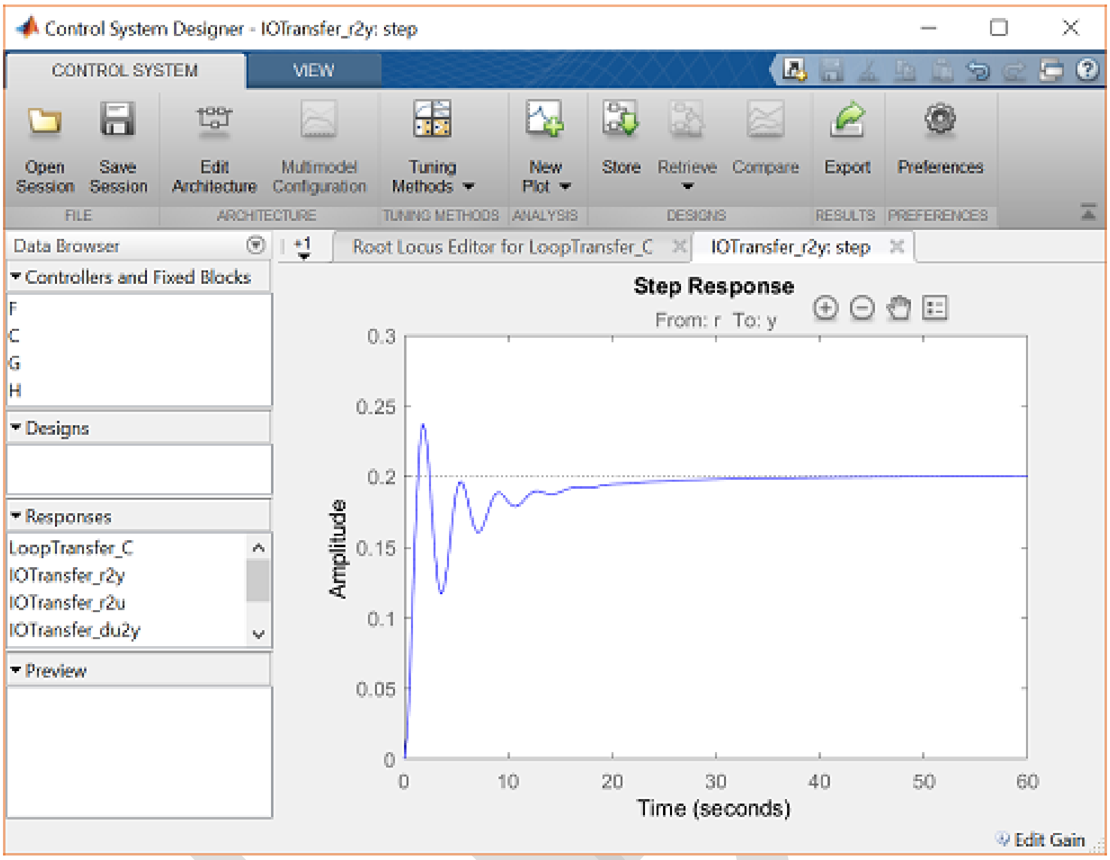


برای شروع، رفتار سیستم را با قرار دادن Kp برابر ۲ مشاهده می‌کنیم. جبران کننده‌ی C(s) همانند پیش جبران کننده‌ی F تنظیم می‌شود، منتها این بار از منوی کشویی C را انتخاب کرده و آنرا برابر ۲ تنظیم کنید. برای مشاهده‌ی عملکرد سیستم و کنترلر به برگه‌ی IOTransfer_r2y:step بروید. اگر تصادفا این برگه را بستید، می‌توانید از پنجره‌ی Control System Designer و انتخاب New step از منوی New Plot مجددا آن را باز کنید. حال در پنجره‌ی New Step to Plot گزینه‌ی Select Responses to Plot را انتخاب کرده و سپس از منوی کشویی گزینه‌ی IOTransger_r2y را انتخاب کرده و دکمه‌ی Plot را بزنید.

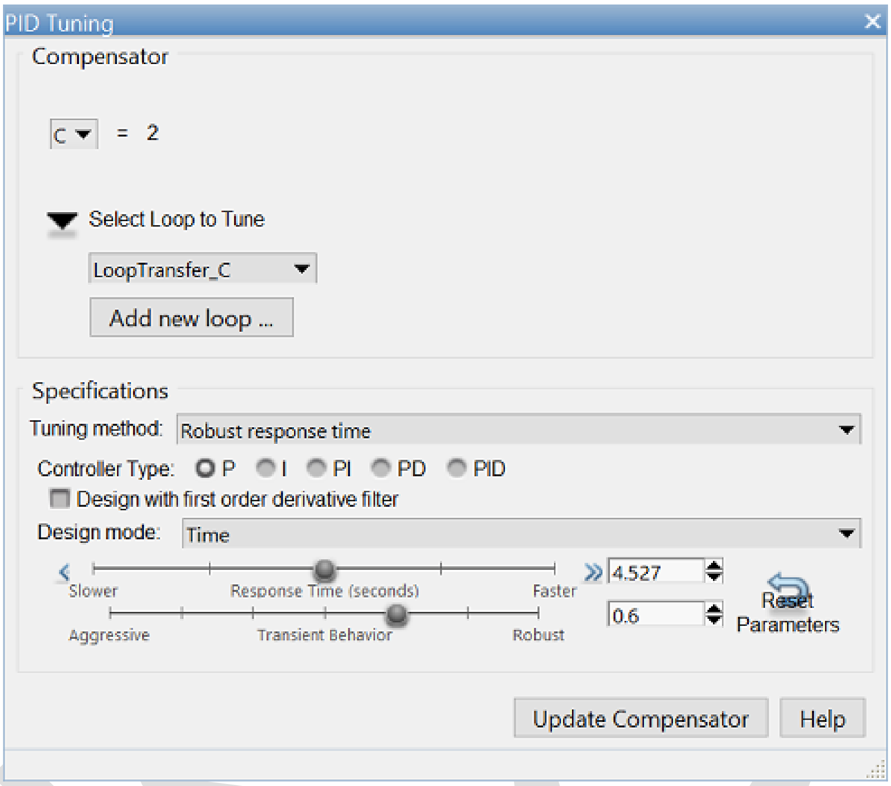




یک پنجره باز می‌شود که پاسخ سیستم را نشان می‌دهد:



با بررسی نمودار بالا متوجه می‌شویم که به جز خطای حالت ماندگار، الزامات طراحی ارضا نشده‌اند. مقدار K_p را جهت رسیدن به مقدار مورد نظر می‌توان از پنجره‌ی Compensator Editor تغییر داد اما بجای آن از Control System Designer برای تنظیم خودکار این بهره استفاده می‌کنیم. برای استفاده از این ویژگی، وارد منوی Tuning Methods در نوار ابزار متلب شده و گزینه‌ی PID Tuning در منوی Automated Tuning را انتخاب کنید. سپس Controller type را از نوع P و از بخش Select Loop to Tune گزینه‌ی `LoopTransfer_C` را انتخاب می‌کنیم (ساختار ما تنها یک حلقه دارد).

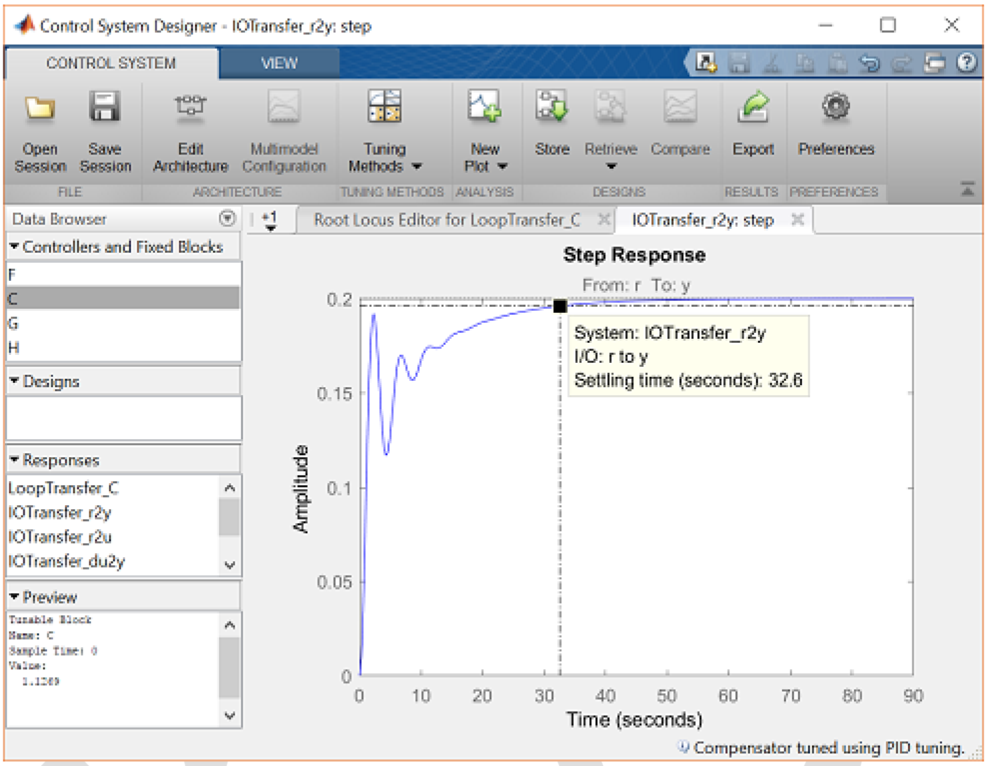


دو روش برای تنظیم بهره کنترلر وجود دارد که در منوی کشویی Tuning method آورده شده است، یکی از آنها Robust response time یا Classical design formulas می‌باشد. الگوریتم Robust response time به صورت خودکار پارامترهای PID را تنظیم می‌کند تا بین سرعت و مقاومت سیستم تعادل ایجاد کند. این الگوریتم می‌تواند کلیه پارامترهای هر نوعی از کنترلر PID را تنظیم کند. می‌توان از این الگوریتم برای تنظیم سیستم‌های پایدار، ناپایدار یا ادغام شده استفاده نمود. از طرفی الگوریتم Classical design formulas به یک سیستم پایدار یا ادغام شده احتیاج دارد و همینطور نمی‌تواند فیلتر مشتق‌گیر را تنظیم کند. اگر این گزینه را انتخاب کنید در منوی کشویی Formula تعدادی گزینه نمایش داده می‌شود. این گزینه‌ها از تکنیک‌های ابتکاری مانند زیگلر-نیکولز[۳۵] گرفته تا ترفندهای عددی که تمامی مقادیر بهره را جهت مینیمم کردن برخی پارامترهای عملکرد سیستم می‌آزماید را شامل می‌شود. برای این مثال از الگوریتم Robust response time استفاده می‌کنیم. حال در منوی کشویی Design mode می‌توانید بین Time یا Frequency انتخاب کنید. از آنجا که الزامات طراحی ما در حوزه‌ی زمان بیان شده‌اند، برای Design mode از Time استفاده می‌کنیم. چون زمان نمو مورد انتظار کمتر از ۲ ثانیه است، از مقدار ۱/۵ ثانیه برای Response Time استفاده می‌کنیم.

حال که تمامی موارد تنظیم شد دکمه‌ی Update Compensator را انتخاب می‌نماییم. الگوریتم مقدار بهره‌ی تناسبی $K_p = 1.1269$ را انتخاب می‌کند. این کنترلر، زمان نمو لازم اما زمان نشست بسیار بالایی دارد. می‌توانید با کشیدن لغزنده به سمت راست پاسخ را سریع‌تر کنید منتها با این کار مقدار فراجهش و نوسانات زیاد می‌شود. این موضوع نشان می‌دهد که تنها با کنترلر تناسبی نمی‌توان به پاسخ مطلوب دست یافت و به جملات انتگرالی و یا مشتق در کنترلر احتیاج باشد.

---

[۳۵] Ziegler-Nichols

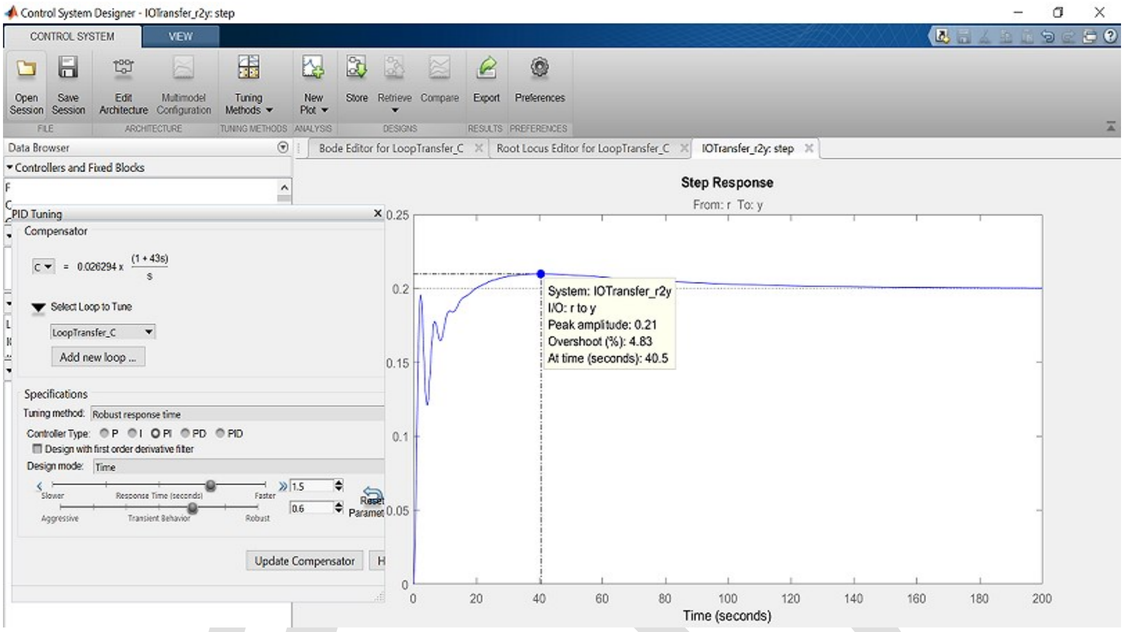


کنترل PI

با یادآوری دانسته‌های ما در **فصل دوم – بخش سوم: طراحی کنترلر PID** می‌دانیم که کنترل انتگرالی جهت کاهش خطای حالت ماندگار مورد استفاده قرار می‌گیرد. در مورد این مثال، شرط لازم برای مقدار خطای حالت ماندگار ارضا شده است. اما برای به تصویر کشیدن این مسئله، طراحی را با کنترلر PI ادامه می‌دهیم. روند طی شده در قسمت قبل را دوباره انجام می‌دهیم تا مقدار بهره‌ها به طور خودکار تنظیم شوند اما این بار مدل کنترلر را PI در نظر می‌گیریم. بقیه مراحل بدون تغییر انجام می‌شود. با کلیک بر روی Compensator Update کنترلر تولید شده به شکل زیر است:

$$C(s) = 0.026294\frac{(1+43s)}{s} \approx \frac{0.0263}{s} + 1.13 \qquad (۳)$$

تابع تبدیل جبران‌ساز دارای بهره‌های $K_i = 0.0263$ و $K_p = 1.13$ می‌باشد..



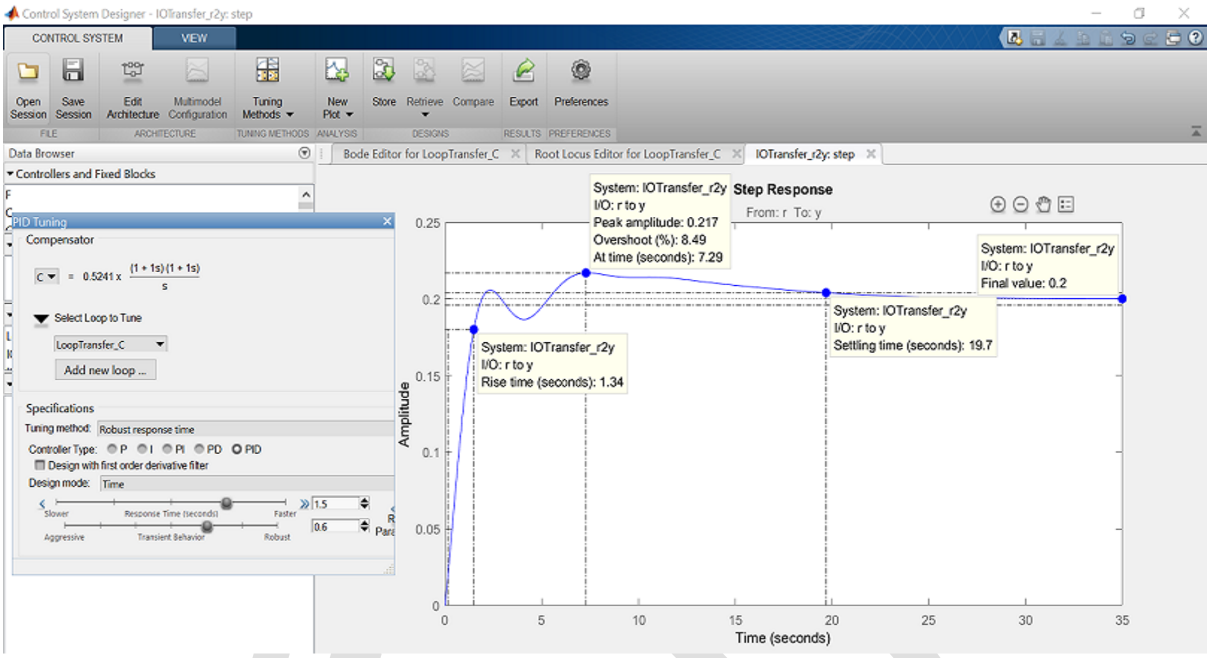
از شکل بالا مشخص است که کنترل انتگرالی به کاهش سریعتر خطای سیستم کمک کرده اما کمکی به کاهش نوسانات سیستم نکرده است. در قسمت بعد از جمله‌ی مشتق در کنترلر استفاده خواهیم نمود.

کنترل PID

مجددا با یادآوری **فصل دوم – بخش سوم: طراحی کنترلر PID**، افزودن کنترل مشتق معمولا باعث کاهش فراجهش سیستم می‌شود. با اضافه کردن این کنترل مشتق ابتدا تا جای ممکن نوسانات سیستم را کاهش داده و سپس با تنظیم دو بهره‌ی دیگر زمان نشست را کاهش می‌دهیم. فرضیه‌ی خود را با تغییر مدل کنترلر به PID و تکرار مراحل قبل می‌آزماییم. با کلیک بر روی Compensator Update کنترلر زیر تولید می‌گردد:

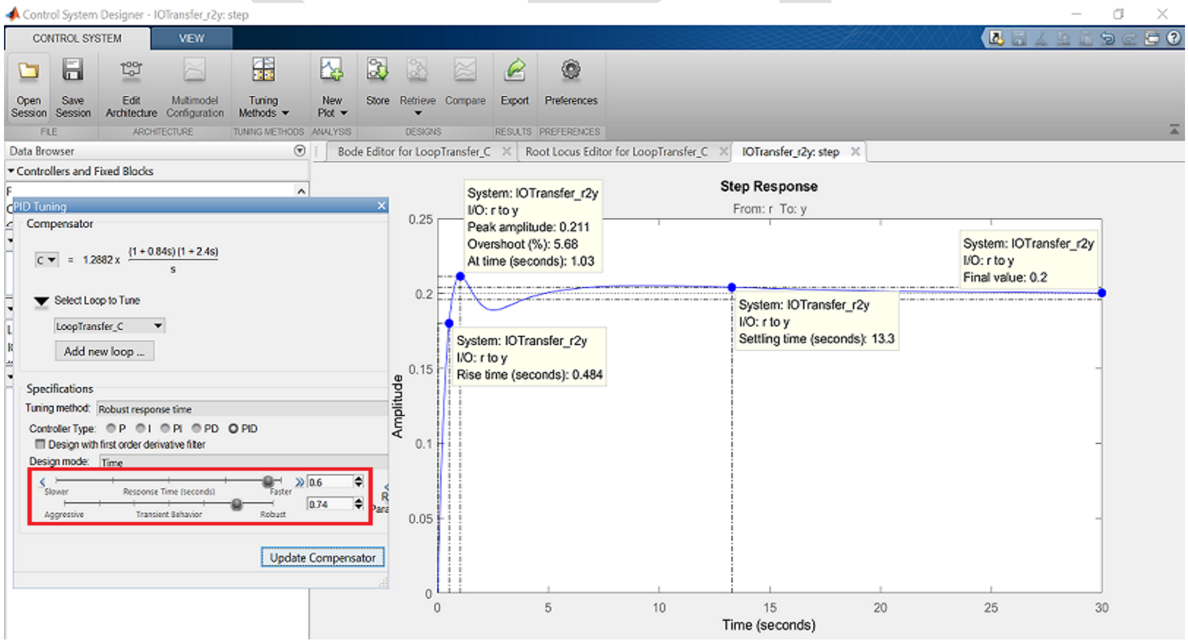
$$C(s) = 0.5241\frac{(1+1s)(1+1s)}{s} \approx \frac{0.5241}{s} + 1.0482 + 0.5241s \qquad (۴)$$

ضرایب به فرم $\mathrm{K_i} = 0.5241$ و $K_p = 1.0482$ و $K_d = 0.5241$ در می‌آید. با این کنترلر پاسخ حلقه بسته به شکل زیر می‌باشد:



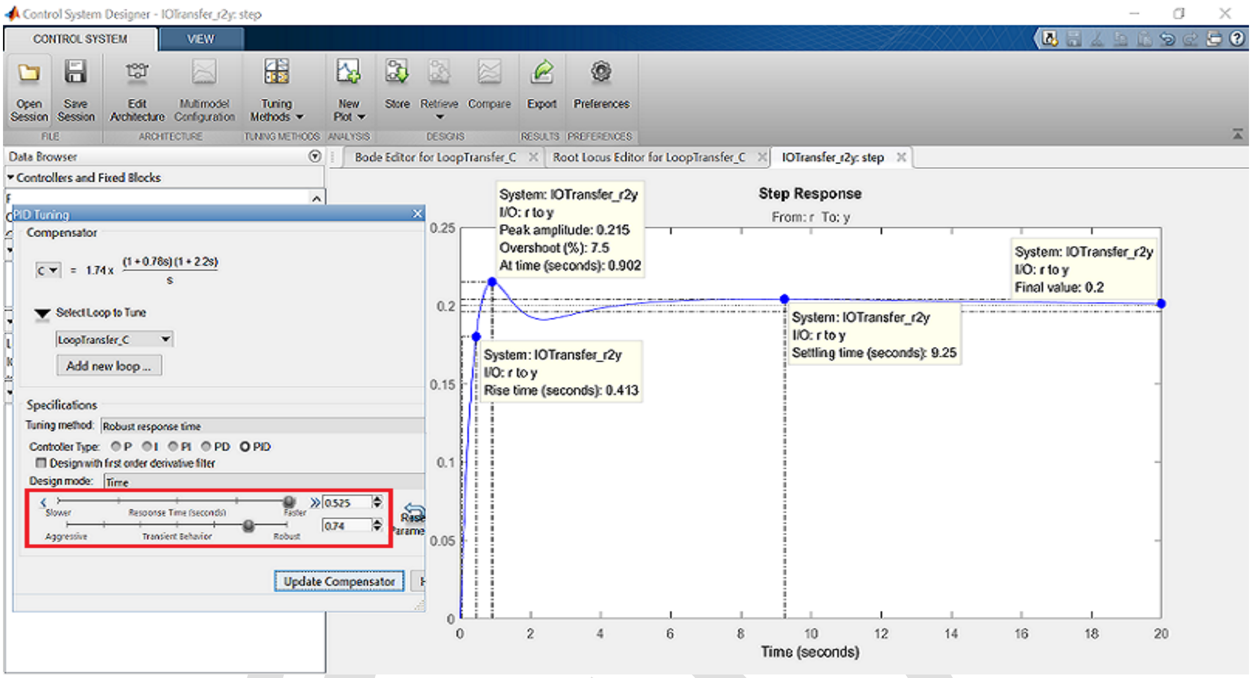
این پاسخ همه الزامات به جز زمان نشست سیستم که بجای زیر ۱۰ ثانیه، ۱۹/۷ ثانیه می‌باشد را ارضا می‌کند. با سریع‌تر کردن پاسخ (کشیدن لغزنده به سمت راست) و همچنین مقاوم‌تر کردن (کشیدن لغزنده‌ی Transient Response به سمت Robust) سیستم برای کاهش نوسانات، کنترلر به صورت زیر در می‌آید:

$$C(s) = 1.2882\frac{1 + 3.24s + 0.2016s^2}{s} \approx \frac{1.2882}{s} + 4.17 + 0.26s \qquad (۵)$$



متوجه می‌شویم که با حرکت دادن لغزنده‌ها به سمت راست، پاسخ سریع‌تر و نوسانات آن کمتر می‌شود. اما همچنان زمان نشست بیشتر از ۱۰ ثانیه می‌باشد. دوباره پاسخ سیستم را سریع‌تر می‌کنیم تا زمان نشست کمتر شود زیرا فاصله‌ی زیادی با فراجهش مطلوب داریم. کنترلر PID در نهایت به شکل زیر در می‌آید:

$$C(s) = 1.74\frac{(1 + 2.98s + 1.716s^2)}{s} \approx \frac{1.74}{s} + 5.1852 + 2.98s \qquad (۶)$$



این پاسخ همه‌ی الزامات طراحی را برآورده کرده است:

- فراجهش ٪۷/۵ < ٪۱۰
- زمان نمو ۰/۴۱۳ ثانیه < ۲ ثانیه
- زمان نشست ۹/۲۵ ثانیه < ۱۰ ثانیه
- خطای حالت ماندگار ٪۰ < ٪۲

بخش چهارم: طراحی کنترلر با مکان هندسی ریشه‌ها

دستورهای کلیدی متلب در این بخش:

```
control System Designer, tf
```

فهرست مطالب بخش

- مکان هندسی اصلی ریشه‌ها
- جبران‌ساز پیش‌فاز

با توجه به مسئله‌ی اصلی، تابع تبدیل حلقه باز برای دینامیک زاویه‌ی حمله‌ی هواپیما به شکل زیر است:

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s} \tag{۱}$$

در حالتی که ورودی زاویه‌ی جابجایی بالابرنده (δ) و خروجی زاویه‌ی حمله‌ی هواپیما (θ) می‌باشد.

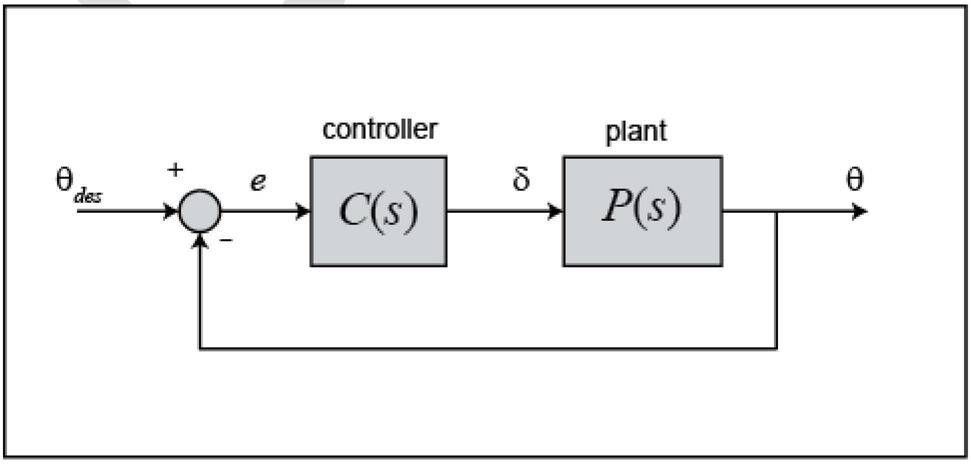
جهت مشاهده‌ی سیستم مورد بررسی و روش استخراج معادله‌ی بالا، به **بخش اول: مدل‌سازی سیستم** مراجع کنید.

برای ورودی پله ۰/۲ رادیان، الزامات طراحی به صورت زیر است:

- فراجهش کمتر از ۱۰٪
- زمان نمو کمتر از ۲ ثانیه
- زمان نشست کمتر از ۱۰ ثانیه
- خطای حالت پایا کمتر از ۲٪

با یادآوری از **فصل دوم - بخش چهارم: مقدمه‌ای بر طراحی کنترلر با مکان هندسی ریشه‌ها** می‌دانیم نمودار مکان هندسی ریشه‌ها نمایش دهنده‌ی تمامی موقعیت‌های ممکن برای قطب‌های حلقه بسته به ازای تغییر یک پارامتر (معمولا بهره‌ی تناسبی K_p) از مقدار صفر تا ∞ می‌باشد. ما با کمک نمودار مکان هندسی ریشه‌ها، موقعیت مناسب قطب‌های سیستم حلقه برای به دست آوردن پاسخ مطلوب را می‌یابیم. با یادآوری **بخش دوم: تحلیل سیستم** می‌توان دید که چگونه تغییر مکان قطب می‌تواند بر روی پاسخ سیستم نسبت به ورودی پله تاثیر بگذارد.

ما از Control System Designer برای طراحی یک جبران‌ساز که با تغییر مکان هندسی ریشه‌های سیستم، قطب‌های حلقه بسته را در موقعیت مطلوب قرار دهد استفاده می‌نماییم. مشخصا ما از مکان هندسی بهره می‌بریم تا پاسخ گذرای سیستم را به حالت مطلوب تغییر دهیم. برای طراحی از ساختار فیدبک واحد زیر استفاده می‌نماییم:

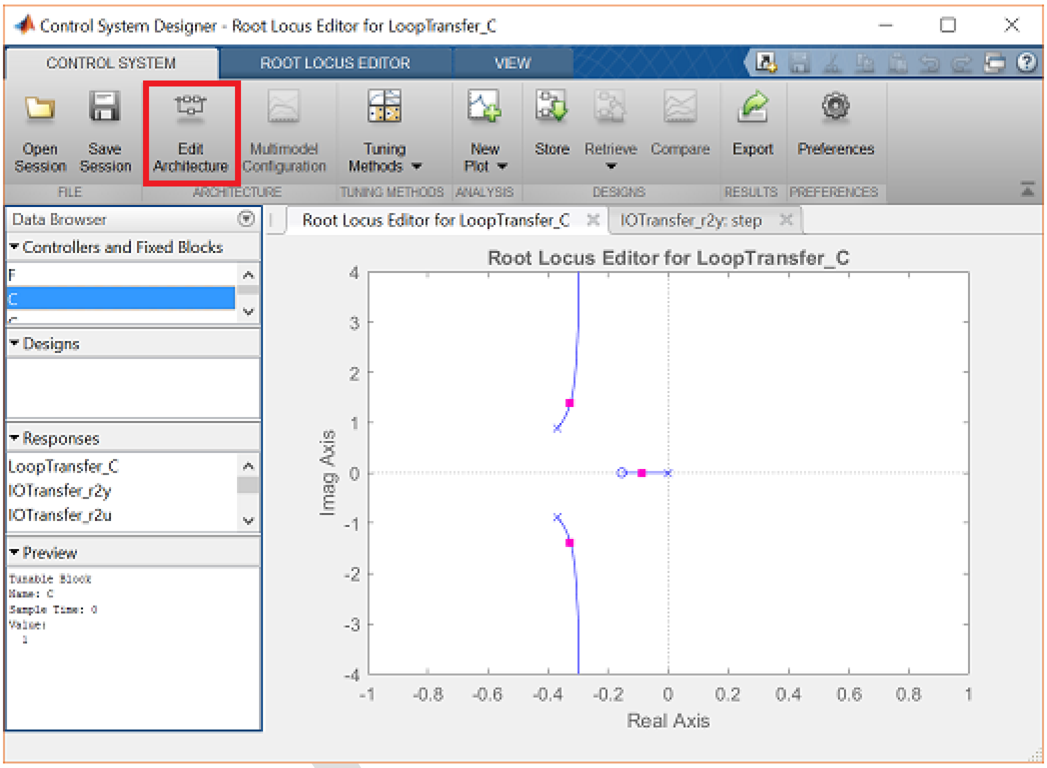


مکان هندسی اصلی ریشه‌ها

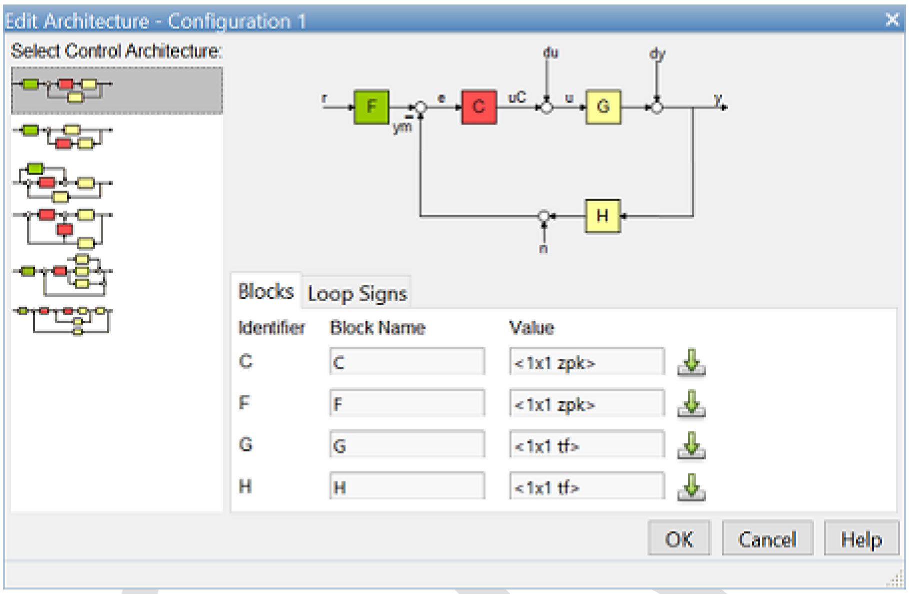
با مکان هندسی ریشه‌های سیستم به همراه یک کنترل تناسبی ساده $C(s) = K$ شروع می‌کنیم. ابتدا کد زیر را وارد کنید تا مدل سیستم تعریف گردد. جهت به دست آوردن جزییات به دست آوردن این معادله به **بخش اول: مدل‌سازی سیستم** مراجعه کنید:

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s);
```

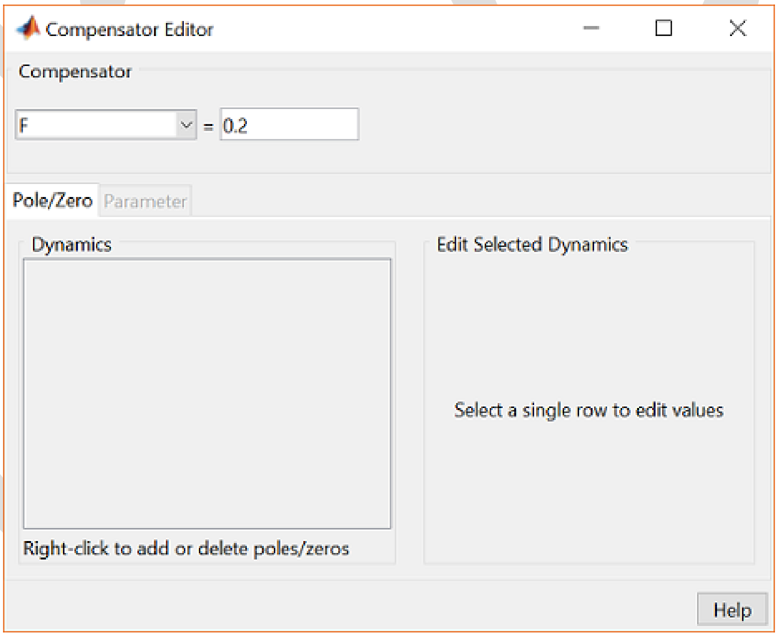
سپس عبارت `controlSystemDesigner('rlocus', P_pitch)` را در پنجره‌ی دستور تایپ کنید. پنجره‌ی Control System Designer نمودار مکان هندسی ریشه‌ها و پاسخ پله واحد سیستم به ازای کنترلر $C(s) = K$ را نشان می‌دهد:



گزینه‌ی Edit Architecture نشان‌دهنده ساختار سیستم است. ساختار پیش‌فرض مشابه با ساختار مورد استفاده‌ی ما می‌باشد:



از آنجا که مقدار مرجع ما ۰/۲ رادیان است، مقدار بلوک F(s) را بر روی ۰/۲ تنظیم می‌کنیم. این کار را از پنجره Compensator Editor انجام می‌دهیم. این پنجره با راست کلیک کردن بر روی شکل و کلیک کردن بر روی گزینه Edit Compensator انجام می‌شود. در بخش Compensator، F را انتخاب کنید و مقدار آن را ۰/۲ قرار دهید. از آنجا که اضافه کردن این پیش‌جبران‌ساز تغییری در سیستم ایجاد نمی‌کند، نمودار مکان هندسی ریشه‌ها به شکل قبل می‌ماند.

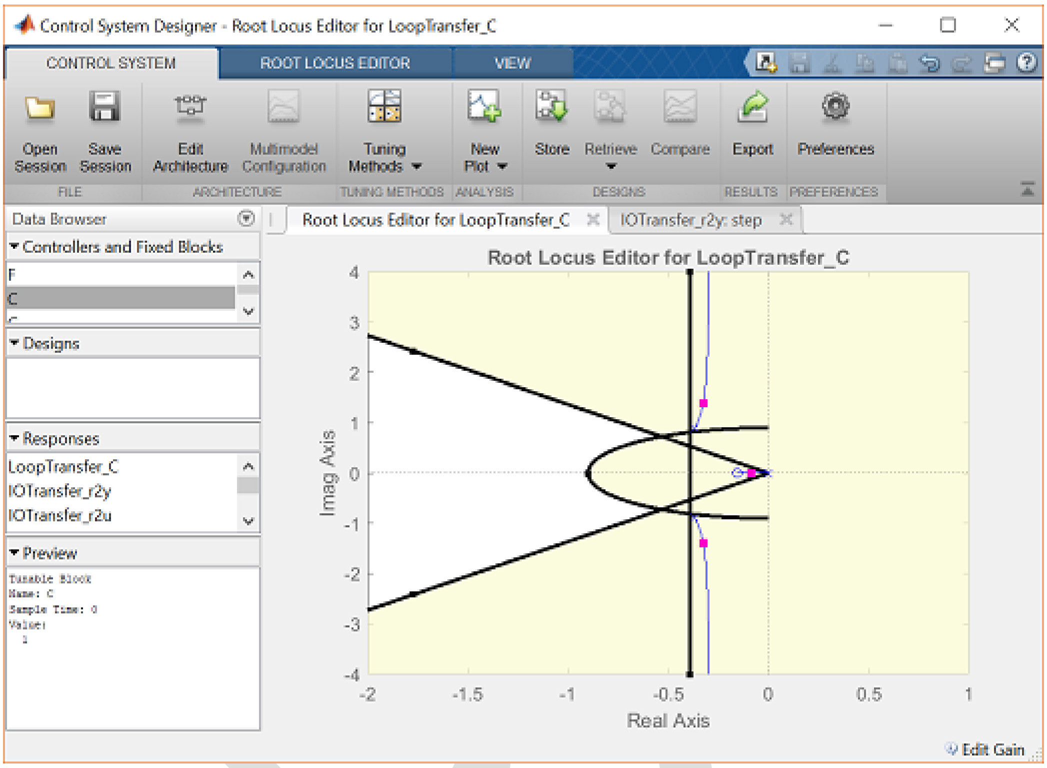


توجه داشته باشید که موقعیت ریشه‌ها بر پاسخ گذرا تاثیر گذار است بنابراین می‌توانیم قسمت‌هایی از مکان هندسی که متناسب با پاسخ مطلوب ما است را به دست بیاوریم. این مناطق با فرض سیستم مرتبه دوم کانونی می‌باشند که ما حتی با داشتن کنترل تناسبی به این فرم نمی‌توانیم دست یابیم اما به هر حال این مناطق نقطه‌ی خوبی برای شروع می‌باشند.

برای اضافه کردن الزامات طراحی به نمودار مکان هندسی ریشه‌ها بر روی شکل راست کلیک کرده و گزینه‌ی Design Requirement → New را انتخاب کنید. شما می‌توانید چندین الزام طراحی از جمله زمان نشست، ضریب میرایی، درصد فراجهش، فرکانس طبیعی، قیود عمومی منطقه را وارد کنید. شرط فراجهش و زمان نشست را به صورت مستقیم می‌توانیم وارد کنیم. زمان نمو به صورت مستقیم داخل گزینه‌ها وجود ندارد اما می‌توان از رابطه‌ی تقریبی زیر برای ارتباط آن به فرکانس طبیعی استفاده نمود:

$$\omega_n \approx \frac{1.8}{T_r} \qquad (۲)$$

لازم است که سیستم ما دارای زمان نمو کمتر از ۲ ثانیه که منجر به فرکانس طبیعی بالاتر از 0.9rad/s برای فرم کانونی مرتبه دوم و بدون میرایی می‌شود، باشد. اضافه کردن شرایط بالا، مکان هندسی سیستم را به صورت زیر در می‌آورد:



مناطق مورد نظر ما در طراحی به صورت هاشور نخورده در شکل بالا نشان داده شده‌اند. مشخصا دو خط شیبدار در بالا بیانگر مقدار فراجهش مورد نیاز سیستم است، هرچه این خطوط به قسمت منفی محور حقیقی نزدیک‌تر باشند، مقدار فراجهش کمتری مجاز است. خط عمودی در s=-0.4 نشان‌دهنده‌ی زمان نشست سیستم است. هر چه از محور موهومی دورتر باشد، مقدار زمان نشست کمتری دارد. شعاع دایره در مرکز، نمایانگر مقدار فرکانس طبیعی سیستم و به تبع آن زمان نمو می‌باشد که در اینجا شعاع دایره برابر فرکانس طبیعی 0.9 می‌باشد.

با توجه به نمودار بالا هیچکدام از ۳ شاخه‌ی مکان هندسی ریشه‌ها وارد منطقه‌ی هاشور نخورده نمی‌شود. در نتیجه نمی‌توانیم با تغییر دادن ضریب کنترل تناسبی K به منطقه‌ی مطلوب برسیم. ما به یک جبران‌ساز دینامیکی با قطب و یا صفر برای تغییر شکل مکان هندسی ریشه‌ها احتیاج داریم.

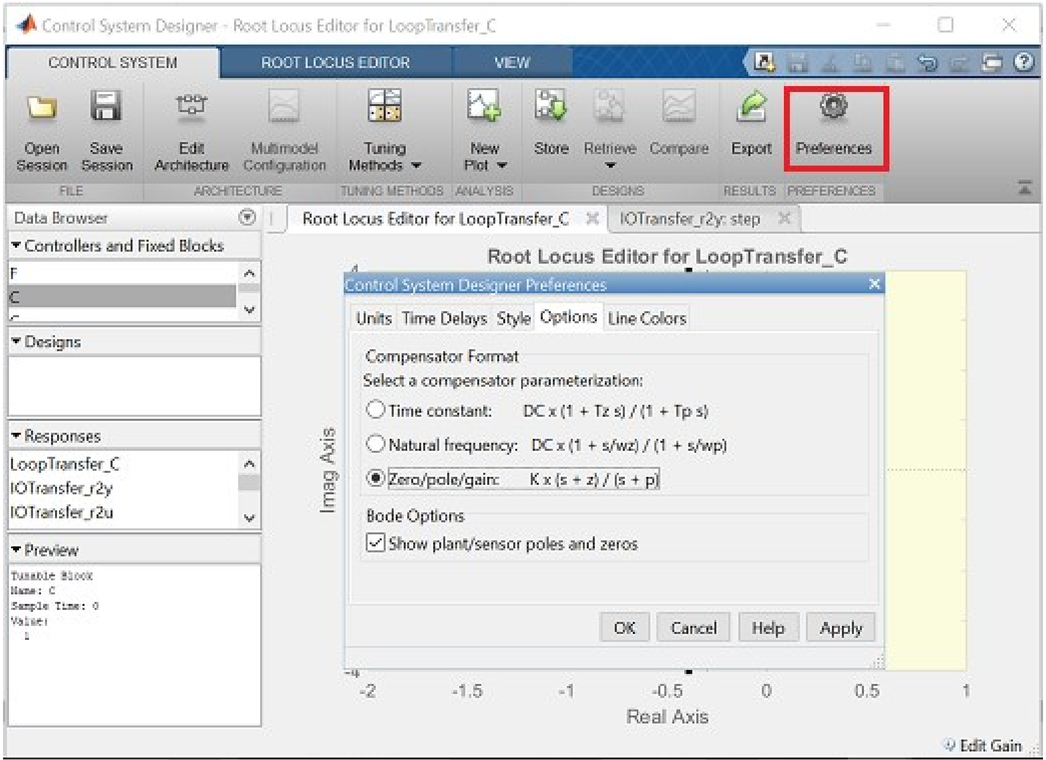
جبران‌ساز پیش‌فاز

ما احتیاج داریم که منحنی مکان هندسی را به سمت چپ سوق دهیم تا به داخل محدوده‌ی مورد نظر برسیم. یک روش برای رسیدن به این مورد، استفاده از یک جبران‌ساز پیش‌فاز است. برای اطلاعات بیشتر به **پیوست ۷: جبران‌ساز پیش‌فاز و پس‌فاز** مراجعه کنید.

تابع تبدیل یک جبران‌ساز پیش‌فاز به صورت زیر می‌باشد. به صورتی که اندازه‌ی صفر از اندازه‌ی قطب کمتر است که یعنی صفر نسبت به قطب به محور موهومی نزدیک‌تر می‌باشد:

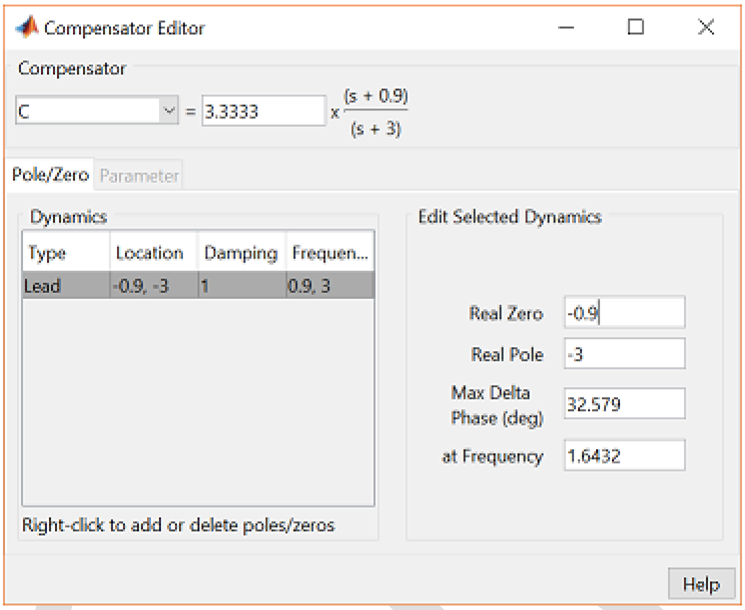
$$C(s) = K\frac{s+z}{s+p} \qquad (۳)$$

قبل از اینکه طراحی جبران‌ساز پیش‌فاز را شروع کنیم، لازم است که ساختار جبران‌ساز آمده در بالا را تشکیل دهیم. این کار با کلیک کردن بر روی Preferences در بالای پنجره‌ی Control System Designer انجام می‌شود. سپس در برگه‌ی Option گزینه‌ی Zero/pole/gain را انتخاب کنید:

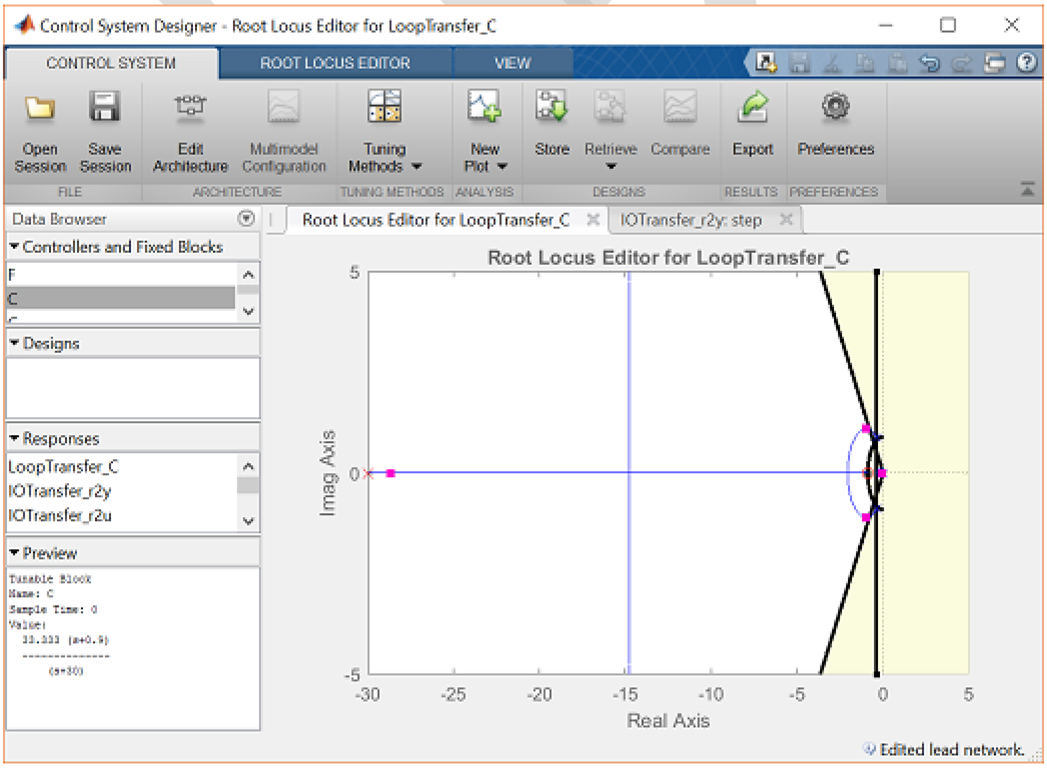


موقعیت صفر جبران‌ساز را با توجه به نیم‌دایره‌ی تعریف شده توسط زمان نمو بر روی محور حقیقی و z=-0.9 قرار می‌دهیم. با اینکار از افزایش بهره‌ی K و در کنار آن خارج نشدن شاخه‌ی مکان هندسی نزدیک به این صفر حلقه باز از منطقه‌ی مطلوب، اطمینان حاصل می‌کنیم. همچنین قطب را در سمت چپ صفر و همانگونه که در تعریف جبران‌ساز پیش‌فاز داریم قرار می‌دهیم. برای شروع قطب p=-3 را در نظر می‌گیریم.

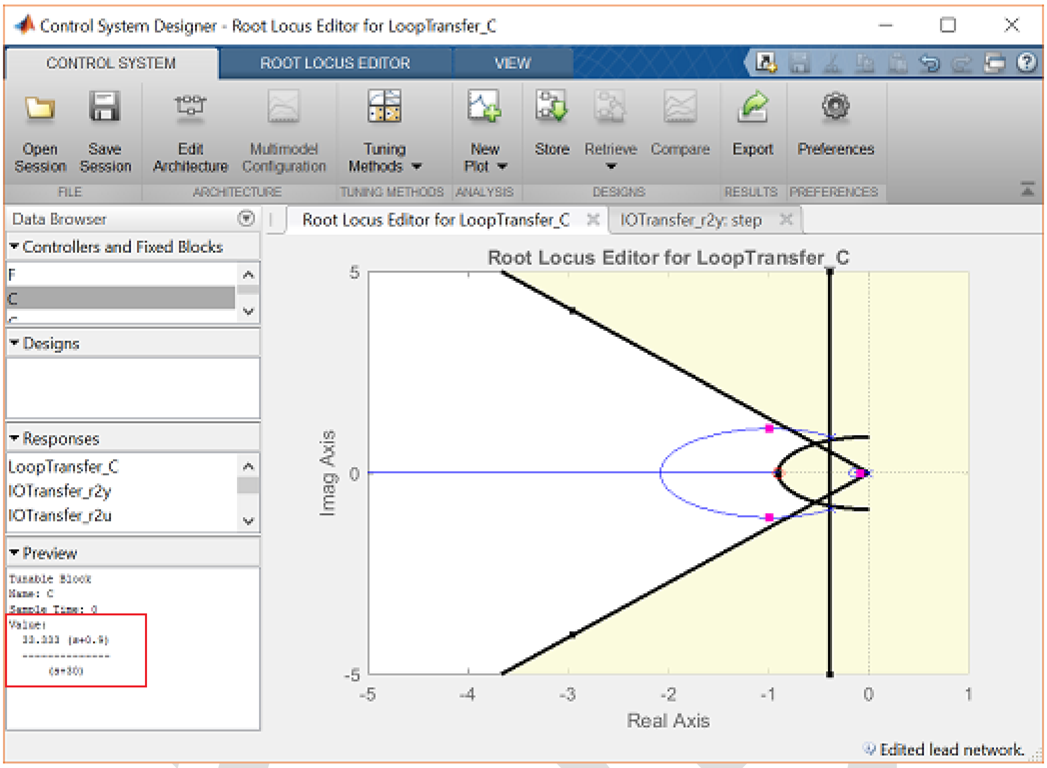
در ابزار Control System Designer می‌توان یک جبران‌ساز پیش‌فاز را در پنجره‌ی Compensator Editor با راست کلیک بر روی مکان هندسی و انتخاب Edit Compensator اضافه کرد. مشخصا با راست کلیک در بخش Dynamics و انتخاب Add Pole/Zero → Lead می‌توان موقعیت صفر و قطب حقیقی را وارد کرد:



جهت مشاهده‌ی تاثیر موقعیت قطب جبران‌ساز، می‌توانید مقادیر مختلفی را در پنجره‌ی Compensator Editor وارد کنید. هر تغییری در جبران‌ساز، تاثیرش را در مکان هندسی ریشه‌ها اعمال می‌کند. همینطور شما می‌توانید مقدار جبران‌ساز را به صورت گرافیکی و از نمودار مکان هندسی ریشه‌ها تنظیم کنید. مشخصا اگر بر روی قطب حلقه باز در نقطه 3- (نشان داده شده با x قرمز رنگ) کلیک کرده و آنرا بر روی محور حقیق جابجا کنید، تغییرات مکان هندسی را مشاهده کنید. همانطور که مشاهده می‌کنید با کشیدن قطب به سمت چپ، مکان هندسی به سمت چپ حرکت کرده و در واقع به سمت منطقه‌ی مطلوب نزدیک می‌شود. قرار دادن قطب در 30-، مکان هندسی ریشه‌ها را به صورت زیر در می‌آورد:

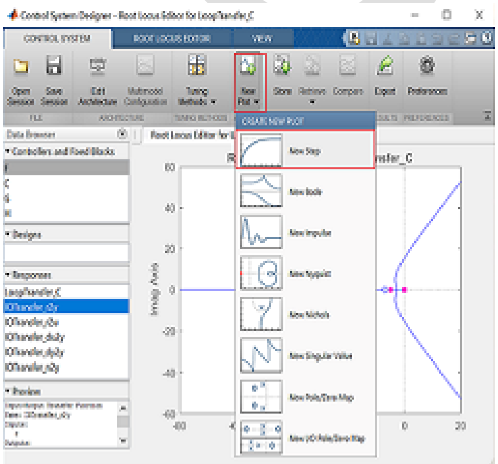


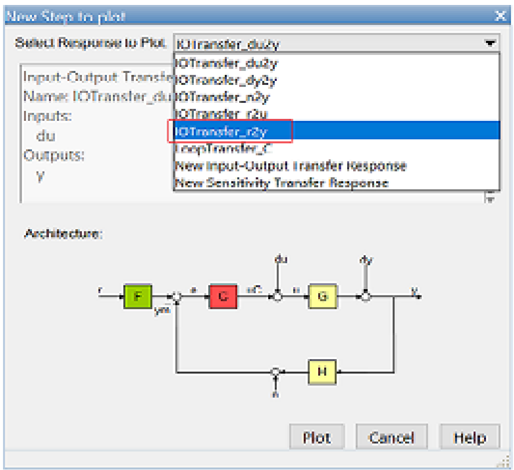
با راست کلیک بر روی نمودار و انتخاب گزینه‌ی Properties، می‌توان حدود محورها را تغییر داده و نمودار را بزرگنمایی کرد:



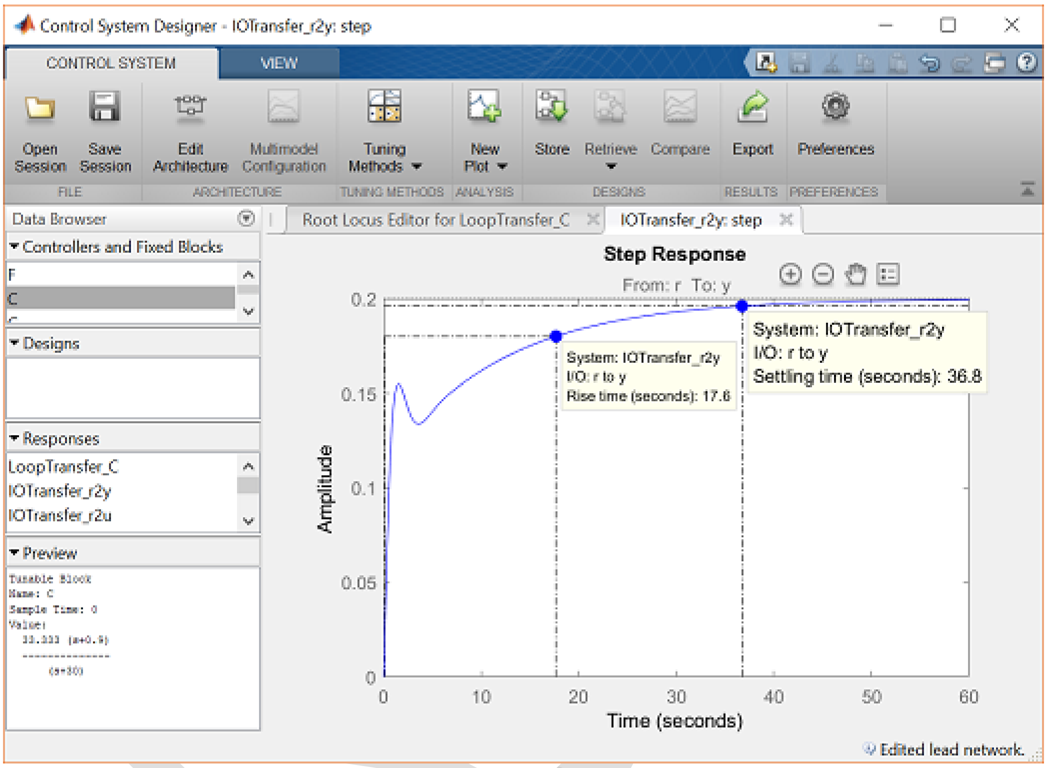
با بررسی دو منحنی بالا، سه شاخه‌ی مکان هندسی به وضوح از منطقه‌ی مطلوب می‌گذرد. شاخه‌ی چهارم نزدیک به محور حقیقی، داخل منطقه‌ی مطلوب نمی‌باشد. اگر چه که قطب حلقه بسته‌ی مربوط به آن شاخه از قطب‌های دیگر سیستم کندتر است، اثر آن به نحوی با صفر موجود در 0.1541- خنثی می‌شود. هر چه مقدار K افزایش پیدا کند، این قطب به صفر نزدیک‌تر شده و اثر کمتری خواهد گذاشت. مکان قطب‌های حلقه بسته به ازای ضریب کنونی (در شکل بالا K=33.3) به صورت مربع‌های صورتی رنگ بر روی مکان هندسی مشخص شده‌اند. دورترین قطب سمت چپ محور اثر کمتری بر روی پاسخ گذرای سیستم دارد زیرا از سایر قطب‌های سیستم سریع‌تر است.

برای مشاهده‌ی اثر صفرها و قطب‌های مرتبه بالاتر بر روی سیستم، لازم است که پاسخ سیستم حلقه بسته به ازای ورودی پله را بررسی کنیم. جهت مشاهده‌ی عملکرد سیستم به برگه‌ی IOTransfer_r2y:step وارد شوید. اگر تصادفا این برگه را بستید، میتوانید از پنجره‌ی Control System Designer و کلیک بر روی New plot و انتخاب New step مجددا آن را باز کنید. حال در پنجره‌ی New Step to Plot و در منوی کشوی Select responses to plot گزینه‌ی IOTransfer_r2y را انتخاب کرده و دکمه‌ی Plot را فشار دهید. در این صورت پاسخ خروجی y سیستم حلقه بسته به ورودی پله r در پنجره‌ی Control System Designer نمایش داده می‌شود.

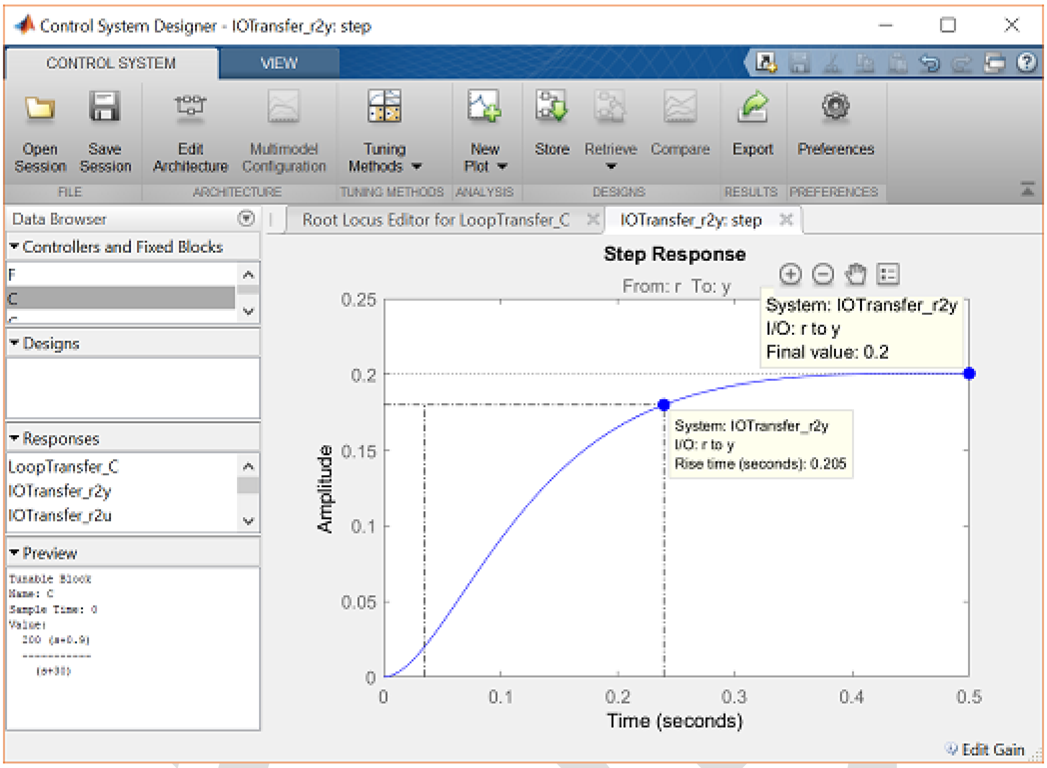




همچنین می‌توانید بعضی از مشخصات پاسخ پله را با راست کلیک بر روی شکل و وارد شدن به منوی Characteristics و انتخاب Settling time و Rise time نمایش دهید. حال نمودار رسم شده به شکل زیر است:



با توجه به شکل بالا مشخص می‌شود که با این مقدار K، هر دو مقدار زمان نشست و زمان نمو بسیار بزرگ است. حال به صورت گرافیکی، یکی از مربع‌های صورتی رنگ را کشیده و آنرا در جهت افزایش K جابجا می‌کنیم. با انجام اینکار، پاسخ پله‌ی سیستم به طور خودکار بروزرسانی می‌گردد. افزایش مقدار K به مقدار ۲۰۰ سبب نزدیک شدن دو کندترین قطب به دو صفر حلقه بسته شده و در نتیجه اثر آن قطب‌ها خنثی می‌گردد. با افزایش مقدار K، قطب کند بعدی با حرکت به سمت چپ سبب کاهش زمان نشست (افزایش σ) و زمان نمو (افزایش ω_n) سیستم می‌شود. از طرفی قرار دادن K=200 باعث می‌شود تمامی قطب‌ها بر روی محور حقیقی بمانند که این موضوع منجر به از حذف فراجهش می‌شود و همینطور در اثر حضور انتگرال‌گیر در سیستم مقدار خطای ماندگار صفر تضمین می‌گردد. در نتیجه این کنترلر همه‌ی مشخصات مطلوب را ایجاد می‌کند.



Control System Designer - IOTransfer_r2y: step
CONTROL SYSTEM
VIEW
Open Session
Save Session
Edit Architecture
Multimodel Configuration
Tuning Methods
New Plot
Store
Retrieve
Compare
Export
Preferences
FILE
ARCHITECTURE
TUNING METHODS
ANALYSIS
DESIGNS
RESULTS
PREFERENCES
Data Browser
Controllers and Fixed Blocks
F
C
Designs
Responses
LoopTransfer_C
IOTransfer_r2y
IOTransfer_r2u
Preview
Root Locus Editor for LoopTransfer_C
IOTransfer_r2y: step
Step Response
From: r To: y
System: IOTransfer_r2y
I/O: r to y
Final value: 0.2
System: IOTransfer_r2y
I/O: r to y
Rise time (seconds): 0.205
Amplitude
Time (seconds)
Edit Gain

بخش پنجم: طراحی کنترلر در حوزه‌ی فرکانس

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
tf, step, feedback, pole, margin, stepinfo
```

با توجه به مسئله‌ی اصلی، تابع تبدیل حلقه باز برای دینامیک زاویه‌ی حمله‌ی هواپیما به شکل زیر است:

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s} \tag{۱}$$

در حالتی که ورودی زاویه‌ی جابجایی بالابرنده (δ) و خروجی زاویه‌ی حمله‌ی هواپیما (θ) می‌باشد.

جهت مشاهده‌ی سیستم مورد بررسی و روش استخراج معادله‌ی بالا، به **بخش اول: مدل‌سازی سیستم** مراجع کنید.

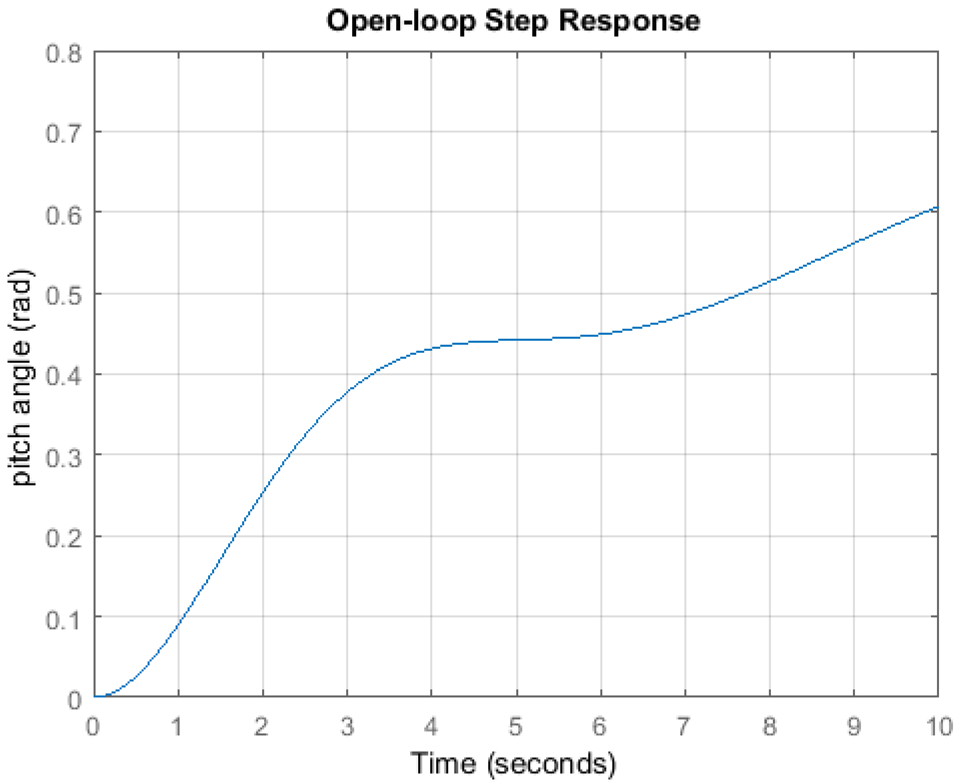
برای ورودی پله ۰/۲ رادیان، الزامات طراحی به صورت زیر است:

- فراجهش کمتر از ۱۰٪
- زمان نمو کمتر از ۲ ثانیه
- زمان نشست کمتر از ۱۰ ثانیه
- خطای حالت پایا کمتر از ۲٪

پاسخ حلقه باز

این بخش را با بررسی رفتار سیستم حلقه باز شروع می‌کنیم. یک ام‌فایل جدید باز کرده و دستور زیر را وارد کنید. توجه کنید که ورودی پله را به اندازه‌ی ۰/۲ برده تا پاسخ به ازای ورودی ۰/۲ رادیان (۱۱درجه) را مشاهده کنید.. کد زیر را اجرا کنید تا منحنی پاسخ پله را مشاهده کنید:

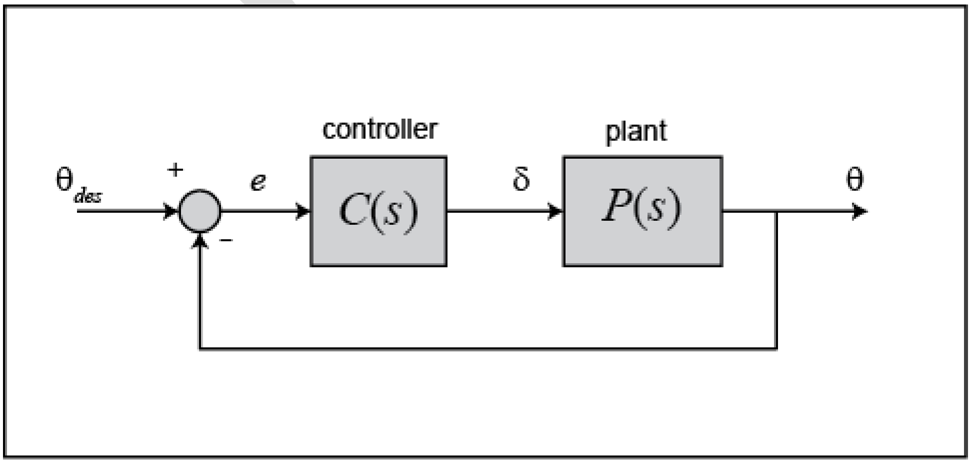
```
t = [0:0.01:10];

s = tf('s');
P_pitch = (1.151*s + 0.1774)/(s^3 + 0.739*s^2 + 0.921*s);
step(0.2*P_pitch,t);
axis([0 10 0 0.8]);
ylabel('pitch angle (rad)');
title('Open-loop Step Response');
grid
```



بررسی منحنی بالا نشان می‌دهد که پاسخ سیستم حلقه باز برای ورودی پله ناپایدار است زیرا برای ورودی پله، خروجی بی‌کران می‌شود. این به این دلیل است که تابع تبدیل دارای یک قطب در مبدا می‌باشد.

پاسخ حلقه بسته

حال بیاید حلقه سیستم را بسته و ببینیم که آیا سیستم پایدار خواهد شد. ساختار فیدبک واحد زیر را برای سیستم خود در نظر بگیرید:



وارد کردن دستور زیر در متلب، تابع تبدیل حلقه بسته سیستم را با در نظر گرفتن فیدبک واحد و کنترلر C(s)=1 تولید می‌کند:

```
sys_cl = feedback(P_pitch,1)
```

```
sys_cl =

 
            1.151 s + 0.1774
  ---------------------------------
  s^3 + 0.739 s^2 + 2.072 s + 0.1774

 
Continuous-time transfer function.
```

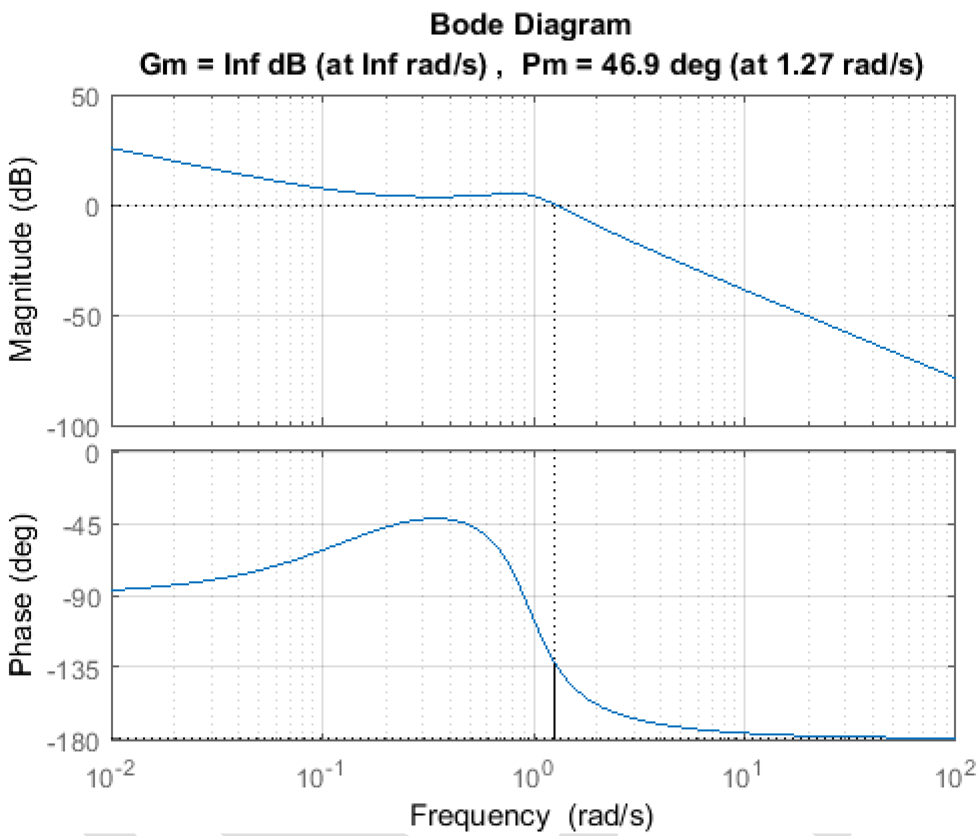
بررسی این تابع تبدیل با استفاده از دستور pole نشان می‌دهد که سیستم حلقه بسته پایدار است زیرا که کلیه قطب‌ها در سمت چپ محور موهومی قرار دارند:

```
pole(sys_cl)
```

```
ans =

  -0.3255 + 1.3816i
  -0.3255 - 1.3816i
  -0.0881 + 0.0000i
```

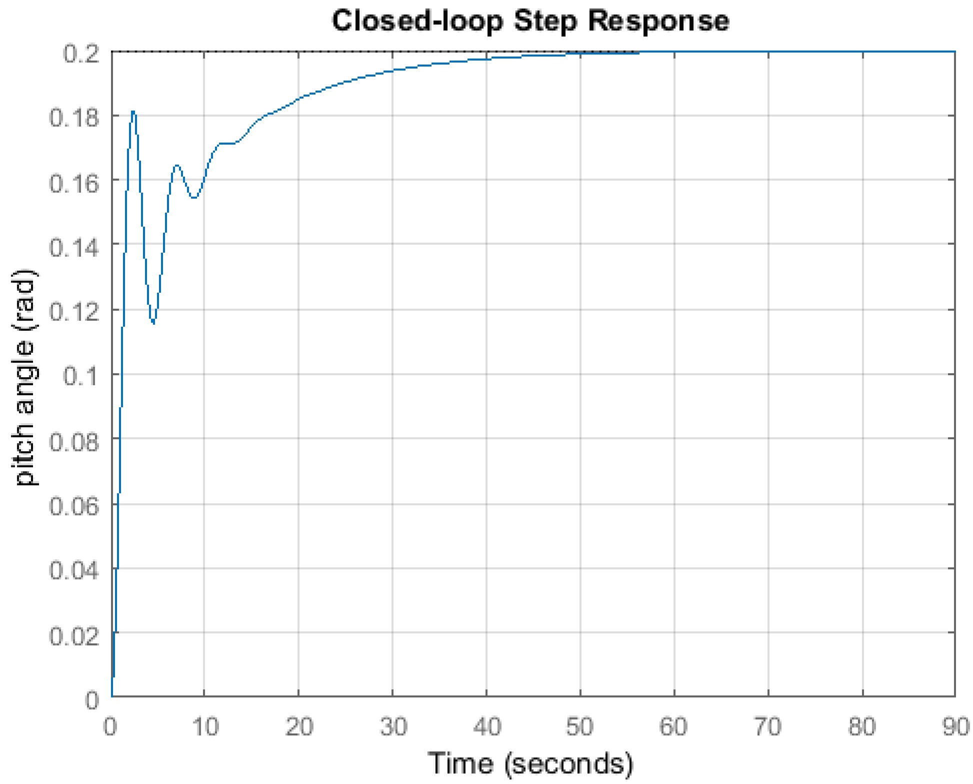
همچنین می‌توان پایداری این سیستم را از طریق بررسی پاسخ فرکانسی حلقه باز بررسی کرد. دستور margin دیاگرام بودی تابع تبدیل را با مشخص کردن حد فاز و حد بهره برای سیستم حلقه بسته، تولید می‌کند.

```
margin(P_pitch), grid
```



بررسی منحنی بالا نشان می‌دهد که سیستم حلقه بسته پایدار است زیرا که هر دو حد فاز و حد بهره مثبت هستند. مشخصا حد فاز برابر ۴۶/۹ درجه و حد بهره بینهایت است. خوب است که سیستم حلقه پایدار می‌باشد، اما آیا پاسخ مطلوب را دارد؟ اضافه کردن کد زیر پاسخ پله‌ی سیستم حلقه بسته را نشان می‌دهد:

```
sys_cl = feedback(P_pitch,1);

step(0.2*sys_cl), grid

ylabel('pitch angle (rad)');
title('Closed-loop Step Response')
```



با توجه به نمودار بالا زمان نشست بسیار بیشتر از ۱۰ ثانیه است. یک روش حل این مسئله اینست که پاسخ سیستم را سریع‌تر کنیم، اما این کار باعث ایجاد مشکل در فراجهش سیستم می‌گردد. در نتیجه باید پاسخ سیستم سریع‌تر گشته و هم زمان فراجهش سیستم نیز کاهش بیابد. می‌توانیم برای این کار از یک جبران‌ساز استفاده کنیم تا دیاگرام بودی سیستم حلقه باز را مجددا شکل دهیم. دیاگرام بودی حلقه باز نشان دهنده‌ی رفتار سیستم حلقه بسته است. نکات زیر را در دیاگرام بودی در نظر خواهیم گرفت:

- فرکانس حد بهره مستقیما با سرعت سیستم حلقه بسته رابطه دارد.
- حد بهره با فراجهش سیستم حلقه بسته رابطه‌ی عکس دارد.

در نتیجه لازم است که یک جبران‌ساز اضافه کنیم تا فرکانس حد بهره و حد فاز سیستم را افزایش دهد.

جبران‌ساز پیش‌فاز

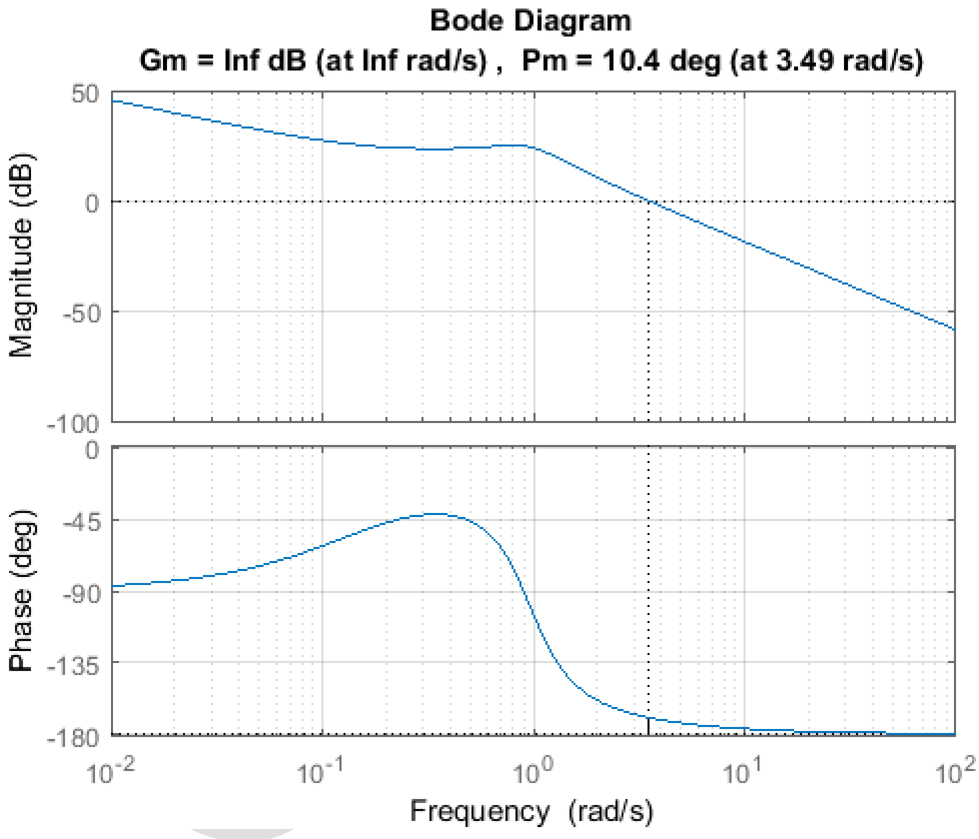
جبران‌سازی که بتواند هر دوی این اهداف را برآورده کند، جبران‌ساز پیش‌فاز است. به **پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز** مراجعه نمایید. جبران‌ساز پیش‌فاز، فاز مثبت به سیستم اضافه می‌کند. فاز اضافه شده باعث افزایش حد فاز سیستم گشته و در نتیجه میرایی سیستم بیشتر می‌شود. همچنین جبران‌ساز پیش‌فاز به طور کلی سبب افزایش اندازه‌ی پاسخ فرکانسی حلقه باز در فرکانس‌های بالا می‌شود که در نتیجه‌ی آن فرکانس گذر بهره و سرعت پاسخ سیستم بالا می‌رود. شکل کلی جبران‌ساز پیش‌فاز به صورت زیر است:

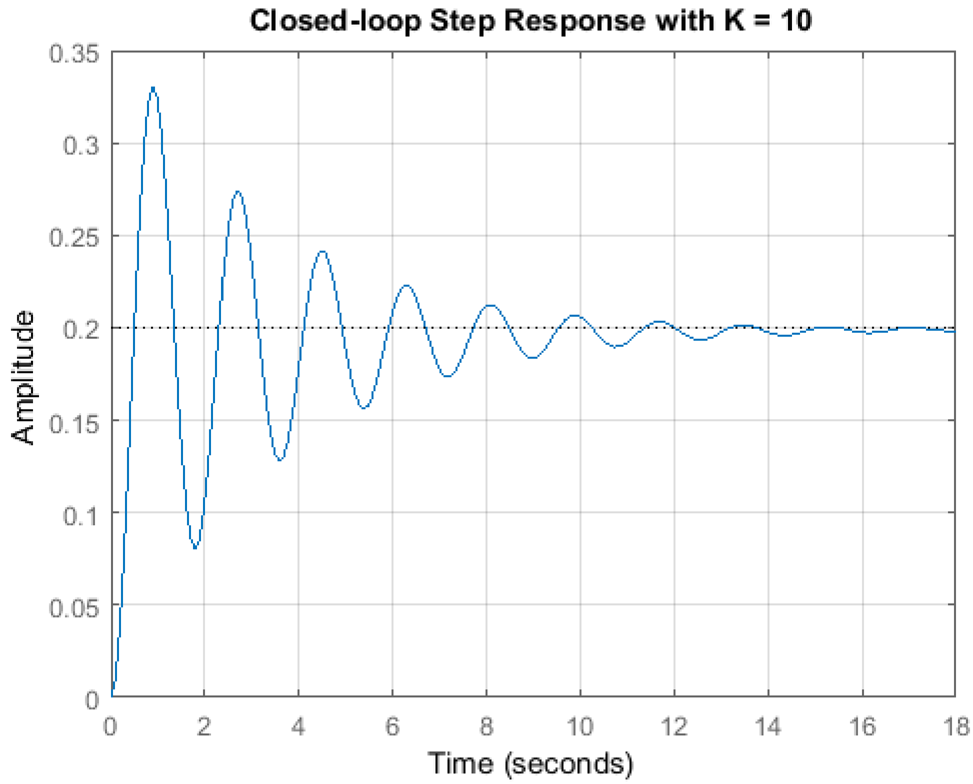
$$C(s) = K\frac{Ts+1}{\alpha Ts+1} \quad (\alpha < 1) \qquad (۲)$$

در نتیجه لازم است تا α، T و K را بیابیم. معمولا بهره‌ی K برای ارضا کردن شرط خطای حالت ماندگار می‌باشد. از آنجا که سیستم ما از نوع اول است (سیستم انتگرال‌گیر دارد) خطای حالت ماندگار با هر مقدار بهره‌ی K برای ورودی پله صفر می‌باشد. اگرچه خطای حالت ماندگار صفر است، سرعت پایین پاسخ می‌تواند وابسته به این موضوع باشد که ضریب خطای سرعت بسیار کوچک است. این کمبود می‌تواند با افزایش مقدار K به بیشتر از ۱ برطرف شود، یا به عبارت

دیگر مقداری از بهره‌ی K که منحنی اندازه را به بالا شیفت دهد. با کمی سعی و خطا ما مقدار K=10 را انتخاب می‌کنیم. با اجرای کد زیر در متلب تاثیر اضافه شدن بهره‌ی K را مشاهده می‌کنید:

```
K = 10;

margin(K*P_pitch), grid

figure;

sys_cl = feedback(K*P_pitch,1);

step(0.2*sys_cl), grid

title('Closed-loop Step Response with K = 10')
```





با بررسی دیاگرام بودی بالا، ما مقدار اندازه‌ی سیستم را در همه فرکانس‌ها بالا برده و فرکانس گذر بهره را افزایش داده‌ایم. اثر این تغییرات در منحنی پاسخ پله‌ی سیستم حلقه بسته نمایان است. متاسفانه افزایش بهره‌ی K باعث پایین آمدن حد فاز سیستم و درنتیجه بالا رفتن فراجهش سیستم شده است. همان‌گونه که قبلا گفته شد، جبران‌ساز پیش‌فاز می‌تواند به سیستم میرایی اضافه کند تا مقدار فراجهش سیستم کاهش بیابد.

طراحی جبران‌ساز را ادامه می‌دهیم، حال به پارامتر α می‌پردازیم که در واقع نسبت بین صفر و قطب می‌باشد. هر چه فاصله‌ی بین صفر و قطب بیشتر باشد، جهش بزرگتری در فاز اتفاق می‌افتد که ماکزیمم مقدار این جهش با یک جفت صفر و قطب، ۹۰ درجه است. رابطه‌ی زیر بیشترین مقدار فاز اضافه شده به سیستم را توسط یک جبران‌ساز پیش‌فاز نشان می‌دهد:

$$\sin(\phi_m) = \frac{1-\alpha}{1+\alpha} \tag{۳}$$

رابطه‌ی بین پاسخ زمانی و پاسخ فرکانسی یک سیستم استاندارد مرتبه دوم زیر میرا قابل استخراج است. یک رابطه‌ی مناسب برای تقریب ضریب میرایی کمتر از ۰/۶ یا ۰/۷ رابطه زیر است:

$$\zeta \approx \frac{PM(degrees)}{100°} \tag{۴}$$

از آنجا که سیستم ما شکل استاندارد سیستم مرتبه دو را ندارد، میتوانیم از رابطه‌ی بالا برای شروع طراحی استفاده کنیم. چون ما به فراجهش کمتر از ۱۰٪ احتیاج داریم، لازم است که ضریب میرایی بیشتر از ۰/۵۹ و در نتیجه حد بهره بالاتر از ۵۹ درجه باشد. از آنجا که حد فاز کنونی سیستم (با وجود K) حدود ۱۰/۴ درجه است، اضافه کردن یه جهش ۵۰ درجه‌ای توسط جبران‌ساز باید کافی باشد. می‌دانیم که جبران‌ساز پیش‌فاز سبب افزایش اندازه‌ی پاسخ فرکانسی سیستم می‌گردد، لازم است که بیشتر از ۵۰ درجه فاز به سیستم اضافه کنیم، چرا که فرکانس گذر بهره تا نقطه‌ی تاخیر فاز افزایش می‌یابد. پس مقدار دلخواه ۵ درجه اضافه می‌کنیم تا در نهایت ۵۵ درجه جهش در فاز داشته باشیم.

از حل معادله زیر می‌توانیم به مقدار α برسیم:

$$\alpha = \frac{1-\sin(55°)}{1+\sin(55°)} \approx 0.10 \qquad (۵)$$

از محاسبات بالا متوجه می‌شویم که مقدار α باید کمتر از ۰/۱ باشد. برای این مقدار α، میزان افزایش اندازه در محل جهش ناشی از جبران‌ساز پیش‌فاز از رابطه‌ی زیر به دست می‌آید:

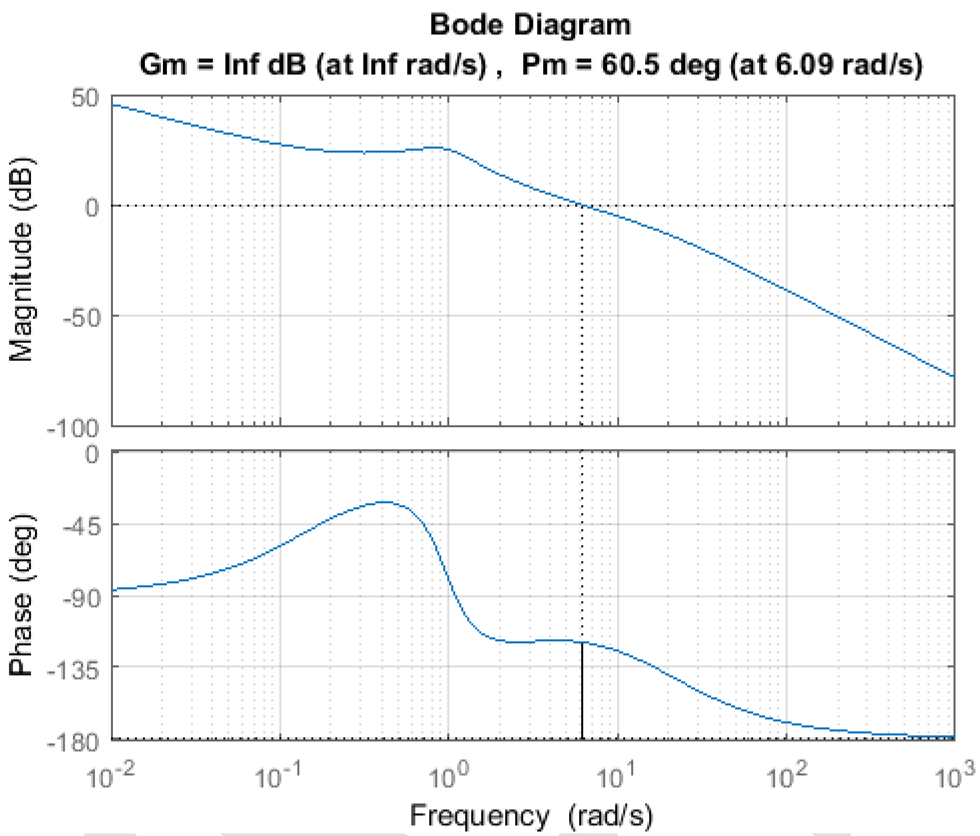
$$20\log\left(\frac{1}{\sqrt{\alpha}}\right) \approx 20\log\left(\frac{1}{\sqrt{0.10}}\right) \approx 10\ dB \qquad (۶)$$

با توجه به دیاگرام بودی نمایش داده شده در بالا، اندازه‌ی سیستم جبران نشده تقریبا در فرکانس 6.1 rad/s برابر -10 dB است. در نتیجه جبران‌ساز طراحی شده فرکانس گذر بهره را از 3.49 rad/s به تقریبا 6.1 rad/s افزایش می‌دهد. با استفاده از این داده‌ها می‌توانیم مقدار T را برای اعمال جهش فاز در فرکانس گذر بهره‌ی جدید جهت بیشینه کردن حد فاز سیستم محاسبه کنیم:

$$\omega_m = \frac{1}{T\sqrt{\alpha}} \Longrightarrow T = \frac{1}{6.1\sqrt{0.10}} \approx 0.52 \qquad (۷)$$

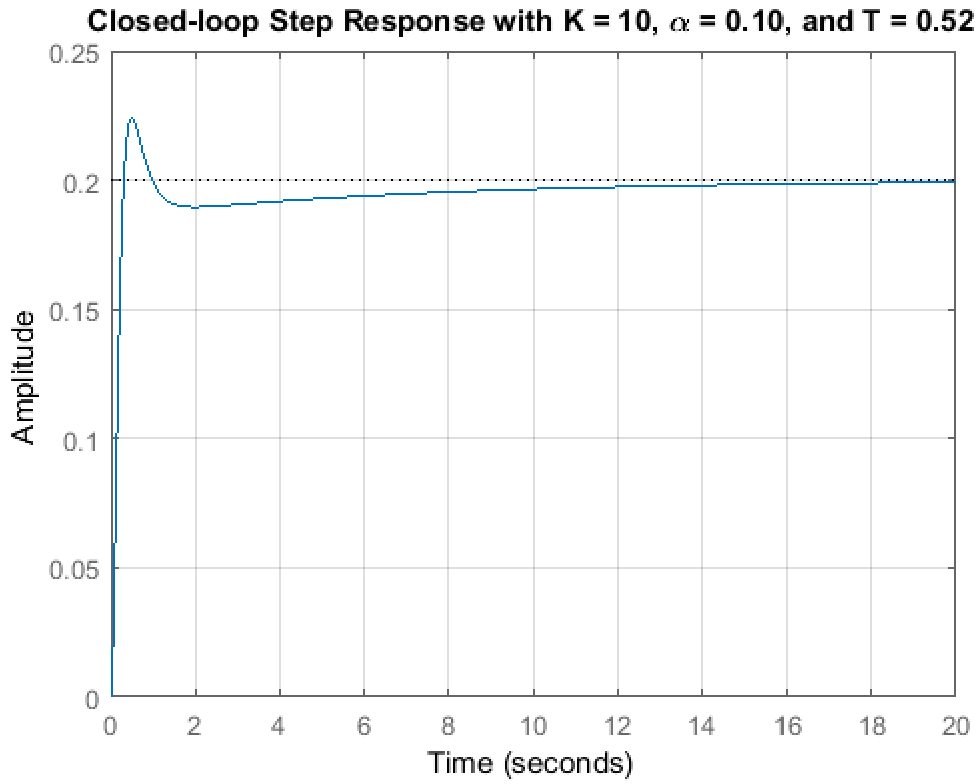
با مقادیر K=10 و $\alpha = 0.10$ و T=0.52 محاسبه شده در بالا، اولین قدم را برای جبران‌ساز پیش‌فاز برداشته‌ایم. با اضافه کردن کد زیر به ام‌فایل خود، تاثیر جبران‌ساز پیش‌فاز را بر روی پاسخ فرکانسی سیستم مشاهده می‌کنید:

```
K = 10;

alpha = 0.10;

T = 0.52;

C_lead = K*(T*s + 1) / (alpha*T*s + 1);

margin(C_lead*P_pitch), grid
```



بررسی منحنی بالا نشان می‌دهد که جبران‌ساز، باعث افزایش حد فاز و فرکانس گذر بهره که مطلوب ما می‌باشد شده است. حال لازم است به پاسخ پله حلقه بسته سیستم نگاه کنیم تا متوجه شویم که آیا به شرایط مطلوب خود رسیده‌ایم یا خیر. کد زیر را جایگزین کد پاسخ فرکانسی سیستم در ام‌فایل خود نمایید و آنرا دوباره اجرا کنید:

```
sys_cl = feedback(C_lead*P_pitch,1);

step(0.2*sys_cl), grid

title('Closed-loop Step Response with K = 10, \alpha = 0.10, and T = 0.52')
```



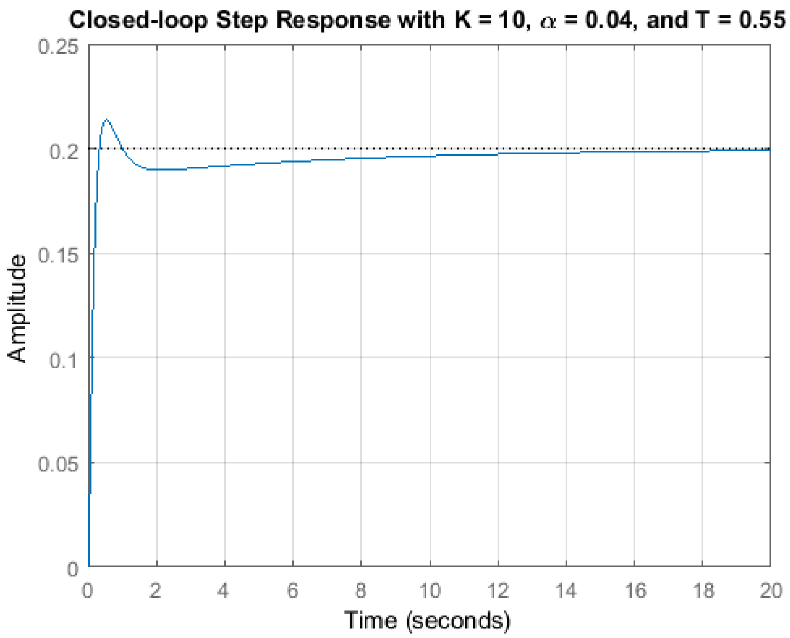
بررسی منحنی بالا نشان می‌دهد که ما به الزامات طراحی نزدیک شده‌ایم. استفاده از دستور متلب stepinfo خصوصیات دقیق پاسخ پله حلقه بسته سیستم را نشان می‌دهد:

```
stepinfo(0.2*sys_cl)
```

```
ans =

  struct with fields:


        RiseTime: 0.2074

    SettlingTime: 8.9835

     SettlingMin: 0.1801

     SettlingMax: 0.2240

       Overshoot: 11.9792

      Undershoot: 0

            Peak: 0.2240

        PeakTime: 0.4886
```

با توجه به داده‌های به دست آمده، تمامی الزامات طراحی به جز فراجهش سیستم که بالاتر از ۱۰٪ می‌باشد ارضا شده است. با تکرار روند طراحی در بالا، به مقادیر K=10 و $\alpha = 0.04$ و T=0.55 می‌رسیم. با تغییر امفایل به شکل زیر، عملکرد به دست آمده توسط این کنترلر نمایش داده می‌شود:

```
K = 10;

alpha = 0.04;

T = 0.55;

C_lead = K*(T*s + 1) / (alpha*T*s + 1);

sys_cl = feedback(C_lead*P_pitch,1);

step(0.2*sys_cl), grid

title('Closed-loop Step Response with K = 10, \alpha = 0.04, and T = 0.55')
```



بررسی نمودار بالا نشان می‌دهد که همه الزامات طراحی برآورده شده‌اند. برای نمایش خصوصیات پاسخ پله دوباره از دستور stepinfo استفاده می‌نماییم:

```
stepinfo(0.2*sys_cl)
```

```
ans =

  struct with fields:


        RiseTime: 0.2203
```

```
SettlingTime: 9.0427
 SettlingMin: 0.1805
 SettlingMax: 0.2137
   Overshoot: 6.8478
  Undershoot: 0
        Peak: 0.2137
    PeakTime: 0.5394
```

بنابراین جبران‌ساز پیش‌فاز زیر، تمامی شرایط پاسخ را برآورده می‌کند:

$$C(s) = 10\frac{0.55s + 1}{0.022s + 1} \tag{۸}$$

بخش ششم: طراحی کنترلر در فضای حالت

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
ss, ctrb, rank, lqr, step
```

در **بخش اول: مدل‌سازی سیستم**، مدل فضای حالت سیستم به صورت زیر به دست آمد:

$$\begin{bmatrix}\dot{\alpha}\\ \dot{q}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}-0.313 & 56.7 & 0\\ -0.0139 & -0.426 & 0\\ 0 & 56.7 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ q\\ \theta\end{bmatrix}+\begin{bmatrix}0.232\\ 0.0203\\ 0\end{bmatrix}[\delta] \tag{۱}$$

$$y=\begin{bmatrix}0 & 0 & 1\end{bmatrix}\begin{bmatrix}\alpha\\ q\\ \theta\end{bmatrix}+[0][\delta] \tag{۲}$$

که ورودی زاویه‌ی انحراف بالابرنده δ و خروجی زاویه‌ی حمله‌ی هواپیما θ می‌باشد. معادلات بالا با فرم کلی مدل خطی فضای حالت سازگار است:

$$\frac{d\boldsymbol{x}}{dt}=A\boldsymbol{x}+Bu \tag{۳}$$

$$y=C\boldsymbol{x}+Du \tag{۴}$$

برای ورودی پله‌ی ۰/۲ رادیان، الزامات طراحی به صورت زیر است:

- فراجهش کمتر از ۱۰٪
- زمان نمو کمتر از ۲ ثانیه
- زمان نشست کمتر از ۱۰ ثانیه
- خطای حالت ماندگار کمتر از ۲٪

در این بخش ما از تکنیک طراحی کنترلر در حوزه‌ی فضای حالت استفاده می‌کنیم. سعی داریم تا قطب‌های حلقه بسته‌ی سیستم را به کمک کنترلر مبتنی بر حالت‌های سیستم، در مکان مناسب قرار دهیم.

کنترل‌پذیری

جهت اعمال تکنیک‌های طراحی کنترلر در فضای حالت، ابتدا باید یک مشخصه‌ی مهم در طراحی کنترلر را بررسی کنیم که آن کنترل‌پذیری می‌باشد. کنترل‌پذیری مشخصه‌ای از سیستم است که بیان می‌کند آیا ما توانایی جابجایی حالت‌های سیستم را به سمت دلخواه داریم یا خیر. این توانایی در واقع قابلیت جایدهی قطب‌های حلقه بسته به هر نقطه از صحفه‌ی مختلط s می‌باشد.

برای اینکه یک سیستم کاملا کنترل‌پذیر باشد، ماتریس کنترل‌پذیری:

$$\mathcal{C}=\begin{bmatrix}B & AB & A^2B & \cdots & A^{(n-1)}B\end{bmatrix} \tag{۵}$$

باید از رنک n باشد. رنک یک ماتریس تعداد سطر (یا ستون) های مستقل خطی آن است. عدد n برابر تعداد حالت‌های سیستم می‌باشد. اضافه کردن توان‌های بیشتر ماتریس A به ماتریس کنترل‌پذیری تاثیری در رنک آن ندارد زیرا که عبارات اضافه شده در واقع ترکیب خطی عبارات قبلی است.

چون ماتریس کنترل‌پذیری ۳×۳ است، رنک ماتریس باید ۳ باشد. دستور rank در متلب رنک یک ماتریس را محاسبه می‌کند. یک m-file جدید ایجاد کرده و کد زیر را وارد در آن وارد کنید تا خروجی زیر به دست آید:

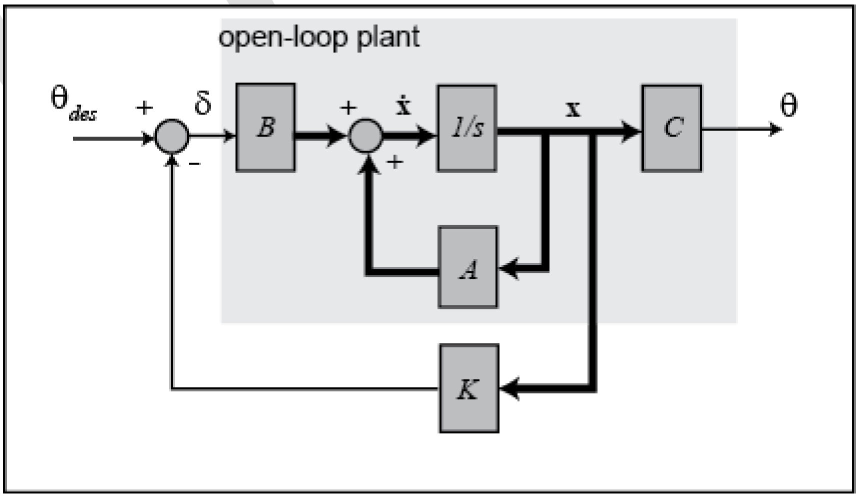
```
A = [-0.313 56.7 0; -0.0139 -0.426 0; 0 56.7 0];

B = [0.232; 0.0203; 0];

C = [0 0 1];

D = [0];


co = ctrb(A,B);


Controllability = rank(co)
```

```
Controllability =

     3
```

در نتیجه سیستم کاملا کنترل‌پذیر است زیرا ماتریس کنترل‌پذیری دارای رنک ۳ می‌باشد.

طراحی کنترلر از طریق جایدهی قطب

شماتیک یک سیستم کنترلی فیدبک تمام حالات در زیر نمایش داده شده است ($D = 0$):



که در آن:

- K = ماتریس بهره‌ی کنترلی
- $x = [\alpha \quad q \quad \theta]'$ = بردار حالت

- θ_{des} = ورودی مرجع (r)
- $(\theta_{des} - Kx) = \delta$ = ورودی کنترلی (u)
- θ = خروجی (y)

با مراجعه به معادلات فضای حالت در صفحات پیشین، مشاهده می‌کنیم که جایگذاری $\delta = (\theta_{des} - K\mathbf{x})$ به جای δ نتیجه‌ی زیر به دست می‌آید:

$$\dot{\boldsymbol{x}} = (A - BK)\boldsymbol{x} + B\theta_{des} \qquad (۶)$$

$$\theta = C\boldsymbol{x} \qquad (۷)$$

با توجه به موارد بالا، ماتریس $A - BK$ نشان‌دهنده‌ی دینامیک سیستم حلقه بسته است. ریشه‌های دترمینان ماتریس $[sI - (A - BK)]$ قطب‌های سیستم حلقه بسته می‌باشند. از آنجا که دترمینان ماتریس بالا یک چندجمله‌ای از مرتبه سوم است، سه قطب وجود دارد که می‌توانیم جایدهی کنیم. همچنین به خاطر اینکه سیستم کاملا کنترل‌پذیر است این قطب‌ها را می‌توانیم در هر کجای صفحه قرار دهیم. با یادآوری **فصل دوم – بخش ششم: مقدمه‌ای بر طراحی کنترلر در فضای حالت**، درمی‌یابیم که با استفاده از تکنیک جایدهی قطب می‌توانیم ماتریس بهره‌ی کنترلی K را به گونه‌ای به دست آوریم قطب‌های حلقه بسته را در موقعیت دلخواه قرار دهیم. توجه کنید که در فیدبک فرض می‌کنیم که تمامی متغیرهای حالت در بردار x قابل اندازه‌گیری می‌باشد هرچند که θ تنها خروجی ماست. اگر اینطور نیست باید از یک مشاهده‌گر جهت تخمین حالت‌های سیستم استفاده کنیم.

از مطالب بالا می‌دانیم که می‌توانیم قطب‌های حلقه بسته را در هر موقعیت دلخواه قرار دهیم. سوال بعدی که پیش می‌آید اینست که آنها را باید کجا قرار دهیم؟ اگر سیستم ما یک سیستم استاندارد مرتبه اول یا دوم بود می‌توانستیم موقعیت قطب‌ها را به مشخصات پاسخ پله مستقیما ارتباط داده و از این رابطه قطب‌های حلقه بسته را در موقعیت مطلوب جایدهی نماییم. این روند در سیستم مرتبه‌های بالاتر یا با وجود صفر(ها) دشوارتر خواهد بود. با یک سیستم مرتبه بالاتر، یک رویکرد اینست که قطب های مرتبه بالاتر را ۵ الی ۱۰ برابر دورتر در سمت چپ قطب‌های غالب قرار دهیم و در نتیجه تاثیر آنها در پاسخ گذرا قابل چشم پوشی خواهد بود. حل مشکل صفرها با روش جایدهی قطب کمی سخت خواهد بود. محدودیت دیگر روش جایدهی قطب اینست که توانایی در نظر گرفتن برخی عوامل مانند مقدار تلاش کنترلی را ندارد.

ریگولاسیون خطی مرتبه دوم

ما از یک تکنیک به نام رگولاتور خطی مرتبه دوم (LQR) برای به دست آوردن بهترین ماتریس بهره‌ی K و بدون انتخاب موقعیت مشخص قطب‌های حلقه بسته استفاده می‌کنیم. این تکنیک کنترلی، یک تعادل بهینه بین خطای سیستم و تلاش کنترلی با درنظر گرفتن تابع هزینه‌ی متناسب (تابع هزینه توسط طراح انتخاب شده و بیانگر وزن نسبی خطای سیستم و تلاش کنترلی می‌باشد) برقرار می‌کند. در مسائل رگولاسیون، فرض می‌شود که مقدار ورودی مرجع صفر است. پس در این صورت اندازه‌ی خطا برابر اندازه‌ی حالت است (برای جزئیات بیشتر به کتب مرجع مراجعه کنید). جهت استفاده از روش LQR باید دو پارامتر را تعریف کنیم: ماتریس وزنی هزینه حالت (Q) و ماتریس وزنی کنترل (R). برای ساده‌سازی ماتریس وزنی کنترل را برابر $R = 1$ و ماتریس هزینه‌ی حالت را برابر $Q = pC'C$ قرار می‌دهیم. استفاده از ماتریس C بیانگر اینست که در تعریف هزینه تنها از حالت‌هایی که در خروجی هستند استفاده می‌کنیم. در این مورد θ تنها حالت مورد استفاده در خروجی است. ضریب وزنی p برای تنظیم ورودی پله در نظر گرفته می‌شود. همچنین در این مورد به دلیل اینکه تنها یک ورودی داریم، R اسکالر است.

حال برای به دست آوردن ماتریس کنترلی K توسط دستور lqr در متلب آماده هستیم. ابتدا ضریب تناسب (p) را برابر ۲ قرار می‌دهیم. کد زیر را به ام‌فایل خود اضافه کرده و آنرا اجرا کنید:

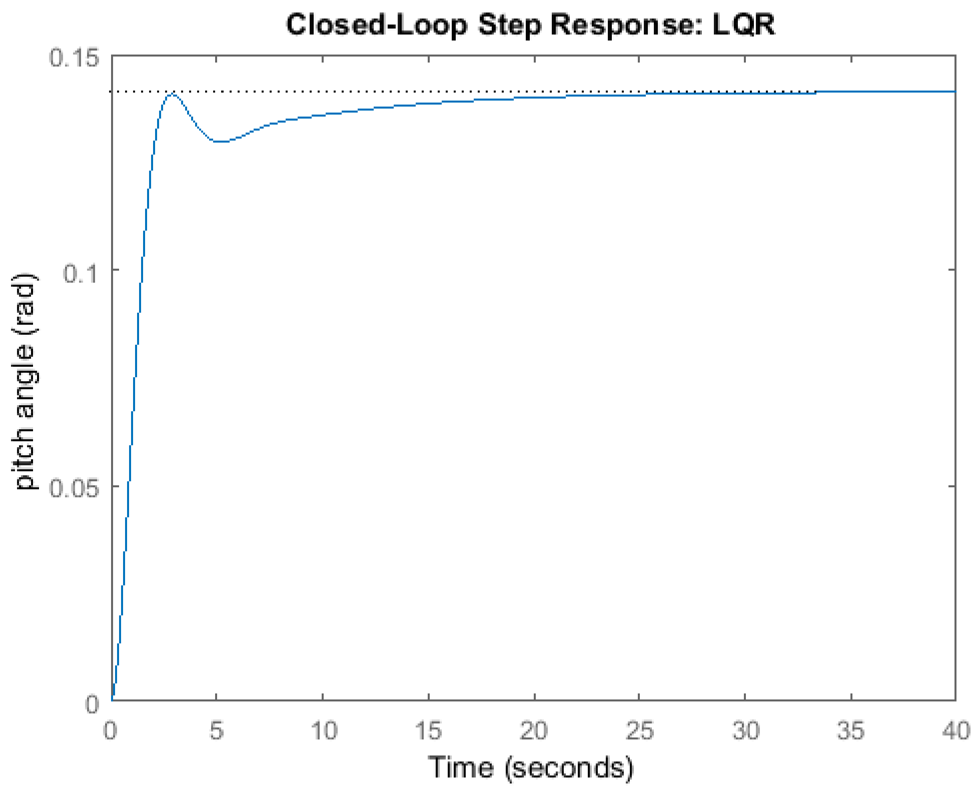
```
p = 2;

Q = p*C'*C
```

```
R = 1;

[K] = lqr(A,B,Q,R)
```

```
Q =

     0     0     0

     0     0     0

     0     0     2

K =

   -0.5034   52.8645    1.4142
```

به ساختار ماتریس‌های Q و K توجه کنید. در معادلات حالت حلقه بسته در بالا قانون کنترلی، مرجع را غیرصفر، $\delta = (\theta_{des} - K\mathbf{x})$ در نظر می‌گیرد. می‌توانیم با اضافه کردن کد زیر به ام‌فایل خود، پاسخ سیستم حلقه بسته را به دست بیاوریم. توجه کنید که به دلیل ورودی زاویه حمله‌ی پله‌ی 0.2 رادیان (۱۱درجه) پاسخ در مقدار 0.2 ضرب شده است:

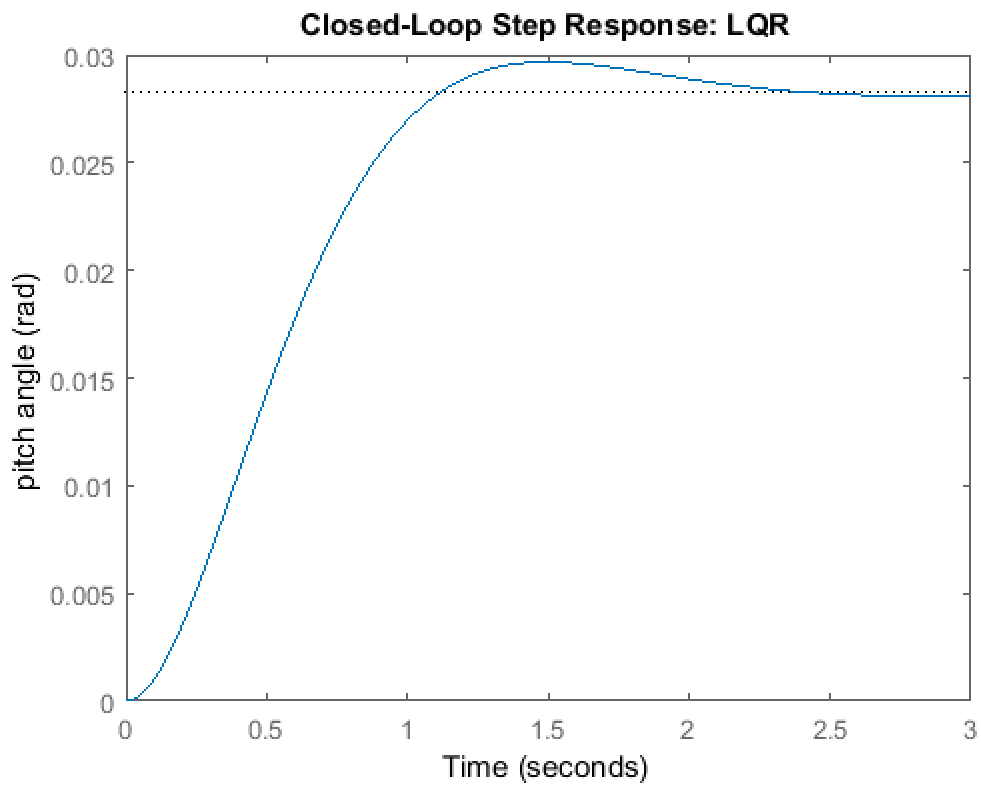
```
sys_cl = ss(A-B*K, B, C, D);

step(0.2*sys_cl)

ylabel('pitch angle (rad)');
title('Closed-Loop Step Response: LQR');
```



بررسی شکل بالا نشان می‌دهد که سیستم بسیار کند است. می‌توانیم سیستم را با درنظر گرفتن اهمیت بیشتر خطای سیستم نسبت به اهمیت تلاش کنترلی است، سریع‌تر کنیم. یعنی مقدار p را افزایش دهیم. بعد از کمی سعی و خطاء به مقدار $p = 50$ می‌رسیم. کد خود را به شکل زیر در ام‌فایل تغییر داده و آنرا اجرا کنید تا پاسخ پله به دست آید:

```
p = 50;

Q = p*C'*C;

R = 1;

[K] = lqr(A,B,Q,R)

sys_cl = ss(A-B*K, B, C, D);

step(0.2*sys_cl)

ylabel('pitch angle (rad)');
title('Closed-Loop Step Response: LQR');
```

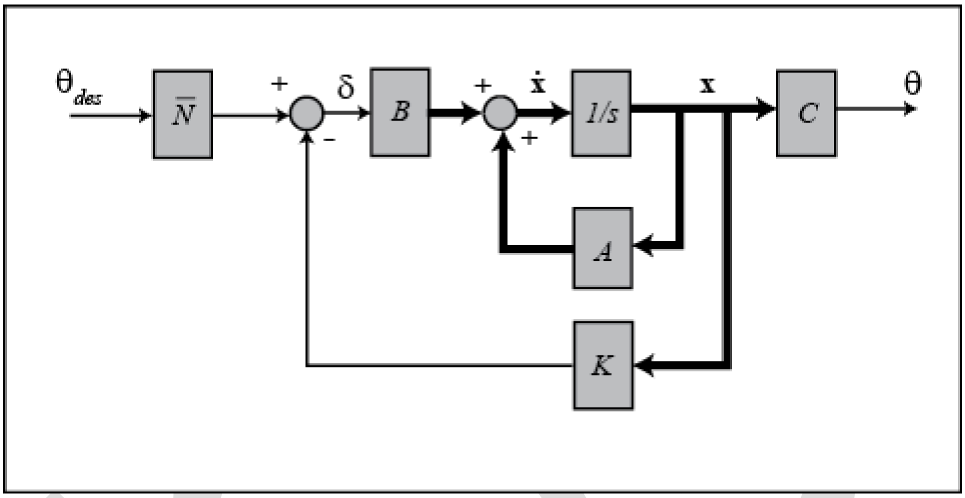
```
K =

   -0.6435  169.6950    7.0711
```



بررسی شکل بالا نشان می‌دهد که زمان نمو، زمان نشست و فراجهش سیستم رضایت بخش است اما خطای حالت ماندگار زیادی وجود دارد. یک روش حل این مشکل استفاده از یک پیش‌جبران‌ساز ($\overline{N}$) است که پاسخ سیستم را به طور کلی جابجا کند.

افزودن پیش‌جبران‌سازی

برخلاف دیگر روش‌ها، سیستم فیدبک تمام حالات، خروجی را با ورودی مرجع مقایسه نمی‌کند. در عوض، تمام حالت‌ها را ضرب در ماتریس کنترلی (Kx) کرده و با مرجع مقایسه می‌کند (شکل شماتیک بالا را مشاهده کنید). در نتیجه نباید انتظار داشته باشیم که خروجی برابر مقدار مرجع شود. برای به دست آوردن خروجی مورد نظر، باید ورودی مرجع را در ضریبی ضرب کنیم که خروجی به مقدار مرجع در حالت پایا برسد. این کار با معرفی ضریب تناسب پیش‌جبران‌سازی با نام $\overline{N}$ انجام می‌گردد. شکل شماتیک زیر سیستم فیدبک حالات با پیش‌جبران‌سازی ($\overline{N}$) را نشان می‌دهد.



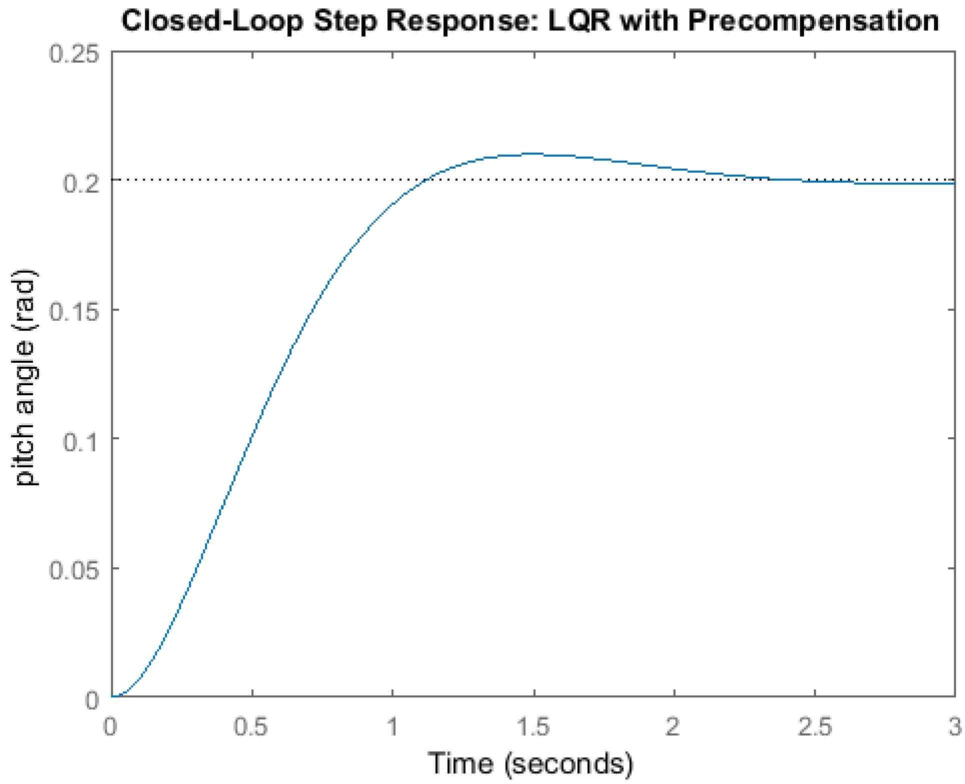
با استفاده از دستور متلب rscale.m به راحتی می‌توانیم مقدار N̄ را بیابیم. از آنجا که این دستور توسط کاربر طراحی شده است، باید این تابع را از درون سی‌دی بر روی کامپیوتر خود ذخیره کنید. بعد از اینکه تابع rscale.m را در مسیر مورد نظر ذخیره کردید، امفایل زیر را به شکل زیر تغییر داده و آنرا اجرا کنید:

```
p = 50;

Q = p*C'*C;

R = 1;

[K] = lqr(A,B,Q,R);

Nbar = rscale(A,B,C,D,K)
```

```
Nbar =

    7.0711
```

با اضافه کردن کد زیر به امفایل، پاسخ زیر را مشاهده می‌کنید:

```
sys_cl = ss(A-B*K,B*Nbar,C,D);

step(0.2*sys_cl)

ylabel('pitch angle (rad)');
title('Closed-Loop Step Response: LQR with Precompensation');
```



حال خطای حالت ماندگار از بین رفته و کلیه الزامات طراحی برآورده شده‌اند.

توجه کنید که پیش‌جبران‌ساز $\bar{N}$ با توجه به مدل سیستم به دست آمده و محل آن در خارج از حلقه‌ی فیدبک است. پس اگر خطایی در مدل (یا اغتشاش ناشناخته) وجود داشته باشد، پیش‌جبران‌ساز آن را تصحیح نمی‌کند و خطای حالت پایا وجود خواهد داشت. شما ممکن است که بیاد داشته باشید که اضافه کردن انتگرال‌گیر می‌تواند حتی در حضور اغتشاش و نامعینی در از بین بردن خطای حالت ماندگار نقش داشته باشد. نکته‌ی استفاده از انتگرال گیر اینست که خطا در ابتدا قبل اصلاح باید گسترش یابد و در نتیجه ممکن است پاسخ سیستم کند باشد. از طرف دیگر پیش‌جبران‌ساز می‌تواند خطای حالت ماندگار را با دانستن مدل سیستم پیش‌بینی کند. تکنیک مفیدی که می‌توان استفاده کرد اینست که پیش‌جبران‌ساز را با کنترل انتگرالی ترکیب کنیم تا از مزایای هر کدام بهره ببریم.

بخش هفتم: طراحی کنترلر دیجیتال

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

```
ss, c2d, rank, ctrb, dlqr, lsim, stairs
```

در این بخش مدل دیجیتال کنترل زاویه‌ی حمله‌ی هواپیما را در نظر می‌گیریم. می‌توان یک مدل نمونه‌برداری شده برای دینامیک زاویه‌ی حمله‌ی هواپیما با استفاده از مدل پیوسته تشکیل داد. در این بخش ما از تکنیک‌های فضای حالت برای طراحی کنترلر استفاده می‌کنیم.

در **بخش اول: مدل‌سازی سیستم**، مدل فضای حالت سیستم به صورت زیر به دست آمد:

$$\begin{bmatrix}\dot{\alpha}\\ \dot{q}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}-0.313 & 56.7 & 0\\ -0.0139 & -0.426 & 0\\ 0 & 56.7 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ q\\ \theta\end{bmatrix}+\begin{bmatrix}0.232\\ 0.0203\\ 0\end{bmatrix}[\delta] \qquad (۱)$$

$$y=\begin{bmatrix}0 & 0 & 1\end{bmatrix}\begin{bmatrix}\alpha\\ q\\ \theta\end{bmatrix}+[0][\delta] \qquad (۲)$$

که ورودی زاویه‌ی انحراف بالابرنده δ و خروجی زاویه‌ی حمله‌ی هواپیما θ می‌باشد.

برای ورودی پله‌ی ۰/۲ رادیان، الزامات طراحی به صورت زیر است:

- فراجهش کمتر از ۱۰٪
- زمان نمو کمتر از ۲ ثانیه
- زمان نشست کمتر از ۱۰ ثانیه
- خطای حالت ماندگار کمتر از ۲٪

فضای حالت گسسته

اولین قدم در طراحی کنترلر دیجیتال، تولید یک مدل نمونه‌برداری شده از سیستم است. متلب با استفاده از دستور c2d می‌تواند این مدل نمونه‌برداری شده را از مدل پیوسته تولید کند. دستور c2d از سه آرگومان ورودی دارد: مدل سیستم، زمان نمونه‌برداری (Ts) و نوع مدار نگه‌دارنده. در این مثال از مدار نگه‌دارنده‌ی مرتبه صفر (zoh) استفاده می‌کنیم. برای جزئیات بیشتر به **فصل دوم - بخش هفتم: مقدمه‌ای بر طراحی کنترلر دیجیتال** مراجعه کنید.

در رابطه با انتخاب زمان نمونه‌برداری، توجه کنید که فرکانس نمونه‌برداری نسبت به دینامیک سیستم به اندازه کافی سریع باشد تا رفتار کامل سیستم در خروجی مشخص باشد و بین مراحل نمونه‌برداری رفتار خاصی اتفاق نیافتد. یک مشخصه‌ی سرعت سیستم، پهنای باند حلقه بسته آن است. یک رویکرد مناسب اینست که فرکانس نمونه‌برداری حداقل ۳۰ برابر بزرگتر از فرکانس پهنای باند حلقه بسته که توسط دیاگرام بودی به دست می‌آید، باشد.

از دیاگرام بودی سیستم حلقه بسته، مقدار فرکانس پهنای باند تقریبا 2 rad/s یا 0.32 Hz به دست می‌آید. می‌توانید این موضوع را خودتان نیز بررسی کنید. برای اینکه مطمئن باشیم که از زمان نمونه‌برداری به اندازه‌ی کافی کوچک استفاده می‌کنیم، زمان نمونه‌برداری را برابر 1/100 sec/sample انتخاب می‌کنیم. حال آماده‌ایم تا از تابع c2d استفاده کنیم. کد

زیر را در ام‌فایل وارد کنید. با اجرای این کد، متلب این چهار ماتریس را که نشان‌دهنده‌ی مدل فضای حالت نمونه‌برداری شده است به شما می‌دهد:

```
A = [-0.313      56.7     0;

     -0.0139   -0.426     0;

      0  56.7     0];

B = [0.232;

     0.0203;

     0];

C = [0 0 1];

D = [0];


sys_ss = ss(A,B,C,D);


Ts = 1/100;

sys_d = c2d(sys_ss,Ts,'zoh')
```

```
sys_d =


  A =

              x1          x2          x3

   x1      0.9968      0.5649           0

   x2  -0.0001385      0.9957           0

   x3  -3.931e-05      0.5658           1


  B =

             u1

   x1   0.002374

   x2  0.0002024

   x3  5.744e-05
```

```
C =

      x1   x2   x3

  y1    0    0    1


D =

      u1

  y1    0


Sample time: 0.01 seconds

Discrete-time state-space model.
```

حال به مدل فضای حالت گسسته زیر دست پیدا کرده‌ایم:

$$\begin{bmatrix} \alpha(k+1) \\ q(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} 0.9969 & 0.05649 & 0 \\ -0.0001 & -0.9957 & 0 \\ 0 & 0.5658 & 1 \end{bmatrix} \begin{bmatrix} \alpha(k) \\ q(k) \\ \theta(k) \end{bmatrix} + \begin{bmatrix} 0.0024 \\ 0.0002 \\ 0.0001 \end{bmatrix} [\delta(k)] \tag{۳}$$

$$y(k) = [0 \quad 0 \quad 1] \begin{bmatrix} \alpha(k) \\ q(k) \\ \theta(k) \end{bmatrix} + [0][\delta(k)] \tag{۴}$$

کنترل‌پذیری

همانند حالت پیوسته، قبل از طراحی کنترلر باید از کنترل‌پذیری سیستم اطمینان حاصل کنیم. برای اینکه یک سیستم کاملا کنترل‌پذیر باشد، ماتریس کنترل پذیری:

$$\mathcal{C} = [B \quad AB \quad A^2B \quad \cdots \quad A^{(n-1)}B] \tag{۵}$$

باید از رنک n باشد. رنک یک ماتریس تعداد سطر (یا ستون) های مستقل خطی آن است. ماتریس کنترل‌پذیری گسسته نیز مشابه سیستم‌های پیوسته می‌باشد. عدد n برابر تعداد حالت‌های سیستم است. از آنجا که ماتریس کنترل‌پذیری ۳×۳ است، رنک ماتریس باید ۳ باشد. متلب رنک یک ماتریس را با دستور rank محاسبه می‌کند. در ام‌فایل خود کد زیر را اضافه کنید:

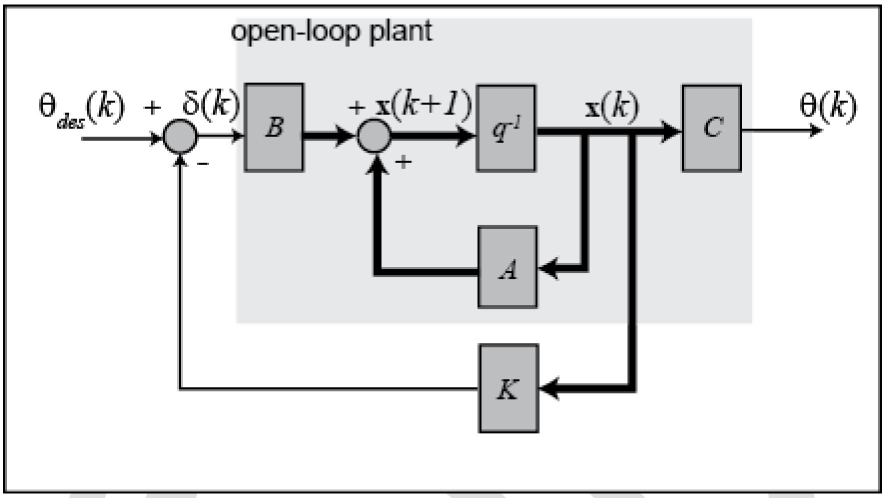
```
co = ctrb(sys_d);

Controllability = rank(co)
```

```
Controllability =

     3
```

در نتیجه سیستم کاملا کنترل‌پذیر است زیرا ماتریس کنترل‌پذیری دارای رنک ۳ می‌باشد.

طراحی کنترلر از طریق جایدهی قطب

شماتیک یک سیستم گسسته‌ی کنترلی فیدبک تمام حالات در زیر نمایش داده شده است. توجه شود که q^{-1} اپراتور تاخیر است (با نرخ زاویه‌ی حمله‌ی هواپیما q اشتباه نشود). توجه شود که فرض $\mathrm{D} = 0$ اعمال شده است.



که در آن:

- K = ماتریس بهره‌ی کنترلی
- $x = [\alpha \quad q \quad \theta]'$ = بردار حالت
- θ_{des} = ورودی مرجع (r)
- $\delta = (\theta_{des} - Kx)$ = ورودی کنترلی (u)
- θ = خروجی (y)

با مراجعه به معادلات فضای حالت در صفحات پیشین، مشاهده می‌کنیم که جایگذاری $\delta = (\theta_{des} - K\boldsymbol{x})$ به جای δ نتیجه‌ی زیر به دست می‌آید:

$$\boldsymbol{x}(k+1) = (A - BK)\boldsymbol{x}(k) + B\theta_{des}(k) \tag{۴}$$

$$\theta(k) = C\boldsymbol{x}(k) \tag{۵}$$

در حالت پیوسته طراحی کنترلر فضای حالت، از روش LQR برای پیدا کردن ماتریس K استفاده کردیم. در حالت دیجیتال، از نسخه‌ی گسسته‌ی روش LQR استفاده می‌کنیم. این تکنیک کنترلی، یک تعادل بهینه بین خطای سیستم و تلاش کنترلی با درنظر گرفتن تابع هزینه‌ی متناسب (تابع هزینه توسط طراح انتخاب شده و بیانگر وزن نسبی خطای سیستم و تلاش کنترلی می‌باشد) برقرار می‌کند. در مسائل رگولاسیون، فرض می‌شود که مقدار ورودی مرجع صفر است. پس در این صورت اندازه‌ی خطا برابر اندازه‌ی حالت است (برای جزئیات بیشتر به کتب مرجع مراجعه کنید). جهت استفاده از روش LQR باید دو پارامتر را تعریف کنیم: ماتریس وزنی هزینه حالت (Q) و ماتریس وزنی کنترل (R). برای ساده‌سازی ماتریس وزنی کنترل را برابر $\mathrm{R} = 1$ و ماتریس هزینه‌ی حالت را برابر $Q = pC'C$ قرار می‌دهیم. استفاده از ماتریس C بیانگر اینست که در تعریف هزینه تنها از حالت‌هایی که در خروجی هستند استفاده می‌کنیم. در این مورد θ تنها حالت مورد استفاده در خروجی است. ضریب وزنی p برای تنظیم ورودی پله در نظر گرفته می‌شود. همچنین در این مورد به دلیل اینکه تنها یک ورودی داریم، R اسکالر است.

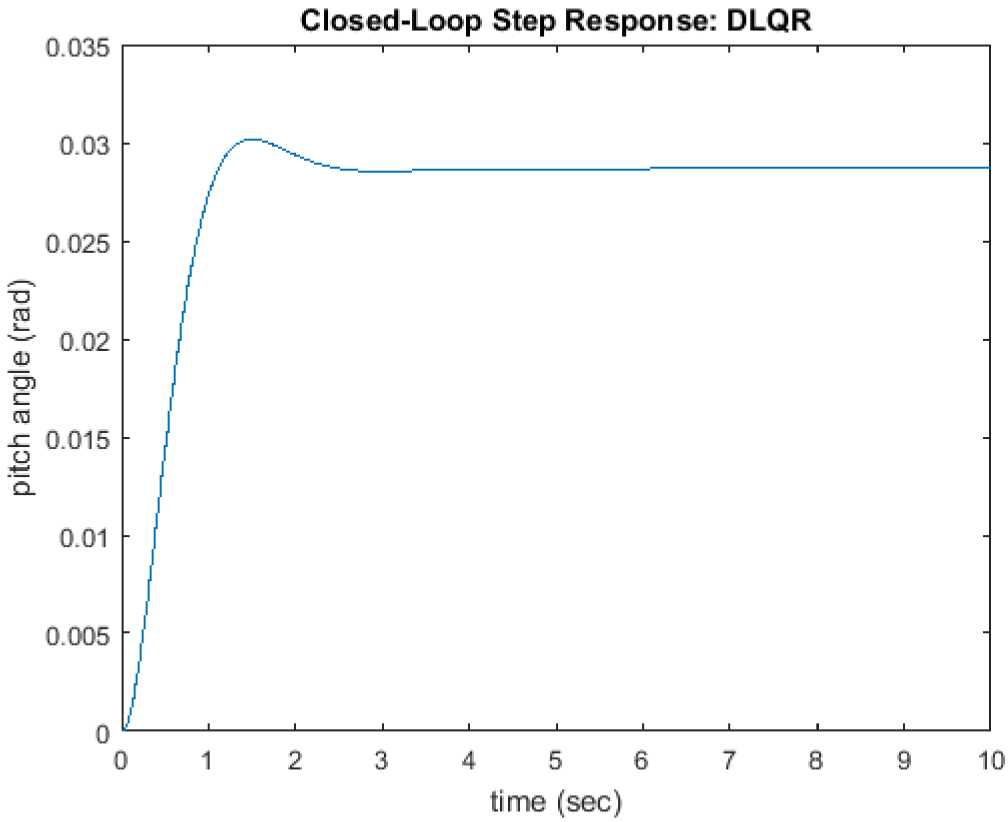
حال آماده‌ایم تا ماتریس کنترلی K را با استفاده از دستور dlqr که نسخه‌ی گسسته‌ی دستور lqr است بیابیم. ضریب وزنی p را با توجه به بخش قبل ۵۰ در نظر می‌گیریم. کد زیر را به امفایل خود اضافه کرده و آنرا اجرا کنید. توجه کنید که مقادیر ماتریس‌های فضای حالت را بر روی مقادیر قبلی که از دستور c2d به دست آمد، مجددا ذخیره کرده‌ایم:

```
A = sys_d.a;

B = sys_d.b;

C = sys_d.c;

D = sys_d.d;

p = 50;

Q = p*C'*C

R = 1;

[K] = dlqr(A,B,Q,R)
```

```
Q =

     0     0     0

     0     0     0

     0     0    50

K =

   -0.6436  168.3611    6.9555
```

به ساختار ماتریس‌های Q و K توجه کنید. در معادلات حالت حلقه بسته در بالا قانون کنترلی، مرجع را غیر صفر، $\delta(k) = \left(\theta_{des}(k) - Kx(k)\right)$ در نظر می‌گیرد. می‌توانیم با اضافه کردن کد زیر به امفایل خود، پاسخ سیستم حلقه بسته را به دست بیاوریم. پاسخ پلکانی به شکل زیر رسم می‌گردد:

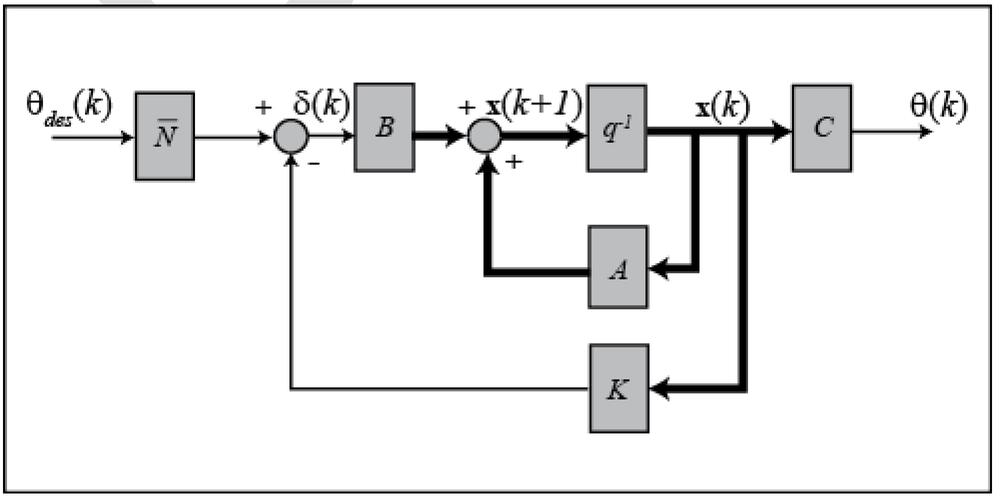
```
time = 0:0.01:10;

theta_des = 0.2*ones(size(time));

sys_cl = ss(A-B*K,B,C,D,Ts);

[y,t] = lsim(sys_cl,theta_des);

stairs(t,y)

xlabel('time (sec)');
ylabel('pitch angle (rad)');
title('Closed-Loop Step Response: DLQR');
```



بررسی شکل بالا نشان می‌دهد که زمان نمو، زمان نشست و فراجهش سیستم رضایت بخش است اما خطای حالت ماندگار زیادی وجود دارد. یک روش حل این مشکل استفاده از یک پیش‌جبران‌سازی است تا پاسخ سیستم را به طور کلی جابجا کنیم.

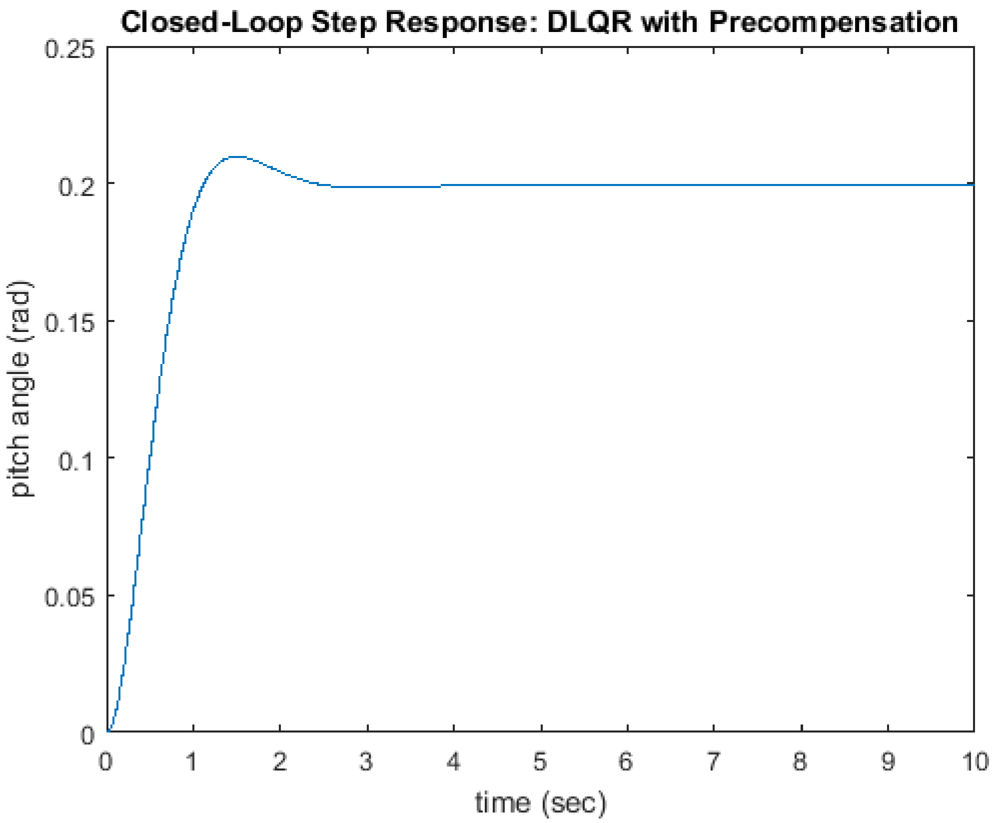
افزودن پیش‌جبران‌سازی

برخلاف دیگر روش‌ها، سیستم فیدبک تمام حالات، خروجی را با ورودی مرجع مقایسه نمی‌کند. در عوض، تمام حالت‌ها را ضرب در ماتریس کنترلی (Kx) کرده و با مرجع مقایسه می‌کند (شکل شماتیک بالا را مشاهده کنید). در نتیجه نباید انتظار داشته باشیم که خروجی برابر مقدار مرجع شود. برای به دست آوردن خروجی مورد نظر، باید ورودی مرجع را در ضریبی ضرب کنیم که خروجی به مقدار مرجع در حالت پایا برسد. این کار با معرفی ضریب تناسب پیش‌جبران‌سازی با نام $\bar{N}$ انجام می‌گردد. شکل شماتیک زیر سیستم فیدبک حالات با پیش‌جبران‌سازی ($\bar{N}$) را نشان می‌دهد.



متاسفانه نمی‌توانیم از تابع rscale.m برای پیدا کردن $\bar{N}$ استفاده کنیم زیرا که این تابع برای سیستم‌های پیوسته تعریف شده است. اما می‌توانیم مقدار مورد نظر را با سعی و خطا بیابیم. بعد از چندبار تلاش، به مقدار N=6.95 برای دستیابی به پاسخ مناسب می‌رسیم. حال کد خود را به شکل زیر تغییر داده و آنرا اجرا کنید تا پاسخ پله زیر را مشاهده کنید:

```
Nbar = 6.95;

sys_cl = ss(A-B*K,B*Nbar,C,D,Ts);

[y,t] = lsim(sys_cl,theta_des);

stairs(t,y)

xlabel('time (sec)');
ylabel('pitch angle (rad)');
title('Closed-Loop Step Response: DLQR with Precompensation');
```



از این شکل مشخص است که ضریب $\bar{N}$ خطای حالت ماندگار را از بین برده است، پس کلیه الزامات طراحی ارضا شده است.

بخش هشتم: مدل‌سازی سیمولینک

فهرست مطالب بخش

- سیستم فیزیکی و معادلات آن
- ساخت مدل فضای حالت
- تولید پاسخ حلقه باز و حلقه بسته

سیستم فیزیکی و معادلات آن

معادلات حاکم بر هواپیما شامل شش معادله‌ی پیچیده‌ی غیر خطی کوپله است. اما با فرض شرایط خاصی می‌تواند به فرم خطی و دی کوپله و به معادلات طولی و جانبی درآید. زاویه‌ی حمله‌ی هواپیما با دینامیک طولی تعریف می‌شود. در این مثال یک خلبان خودکار جهت کنترل زاویه‌ی حمله‌ی هواپیما طراحی می‌شود.

در این بخش مدل خطی‌سازی شده‌ی مدل هواپیما را به همراه کنترلر فیدبک حالات که در بخش‌های قبل طراحی گردید، شبیه‌سازی می‌نماییم. در **بخش اول: مدل‌سازی سیستم**، مدل فضای حالت سیستم با مقادیر عددی به صورت زیر درآمد:

$$\begin{bmatrix}\dot{\alpha}\\ \dot{q}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}-0.313 & 56.7 & 0\\ -0.0139 & -0.426 & 0\\ 0 & 56.7 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ q\\ \theta\end{bmatrix}+\begin{bmatrix}0.232\\ 0.0203\\ 0\end{bmatrix}[\delta] \tag{۱}$$

$$y=[0 \quad 0 \quad 1]\begin{bmatrix}\alpha\\ q\\ \theta\end{bmatrix}+[0][\delta] \tag{۲}$$

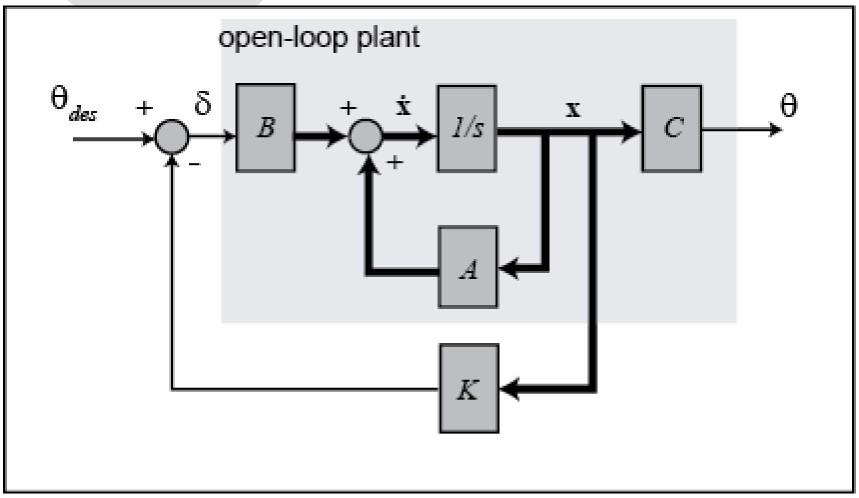
که ورودی زاویه‌ی انحراف بالابرنده δ و خروجی زاویه‌ی حمله‌ی هواپیما θ می‌باشد. معادلات بالا با فرم کلی مدل خطی فضای حالت سازگار است:

$$\frac{d\boldsymbol{x}}{dt}=A\boldsymbol{x}+Bu \tag{۳}$$

$$y=C\boldsymbol{x}+Du \tag{۴}$$

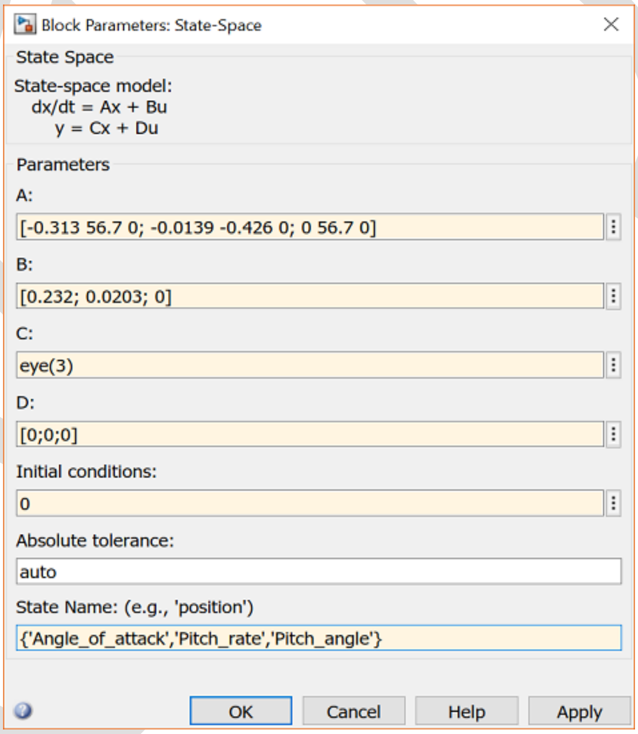
ساخت مدل فضای حالت

در این قسمت یک مدل سیمولینک از معادلات بالا می‌سازیم. یک روش ساختن یک مدل از سیستم با فیدبک حالت است که شکل زیر را نشان دهد:



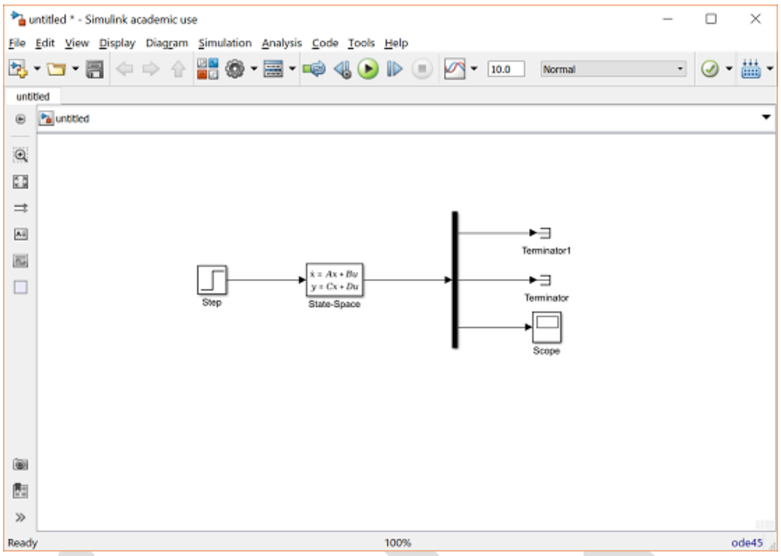
اما بجای آن از بلوک فضای حالت موجود در سیمولینک برای تشکیل سیستم حلقه باز استفاده کنیم. مراحل زیر را انجام دهید:

- سیمولینک را باز کرده و یک مدل جدید باز کنید.
- یک بلوک Step از کتابخانه‌ی Sources وارد کنید.
- برای تولید ورودی پله‌ی مناسب در t=0، دو بار بر روی بلوک Step کلیک کنید و Step time را بر برابر 0 و Final value را برابر ۰/۲ قرار دهید تا بیانگر ورودی مرجع 0.2 رادیان باشد.
- یک بلوک Demux از کتابخانه‌ی Signal Routing وارد کرده و تعداد خروجی‌های آن را برابر ۳ قرار دهید که هر خروجی برای یکی از سه متغیر حالت سیستم می‌باشد.
- یک بلوک Scope را وارد کرده و خروجی سوم بلوک Demux را به آن وصل کنید. تنها متغیر حالت سوم را رسم می‌کنیم زیرا این حالت بیانگر خروجی سیستم یا همان زاویه‌ی حمله‌ی هواپیما (θ) می‌باشد.
- دو بلوک Terminator را از کتابخانه‌ی Sinks وارد کرده و به دو خروجی دیگر بلوک Demux متصل کنید.
- یک بلوک State-Space از کتابخانه‌ی Continuous وارد کرده، ورودی آنرا به بلوک Step و خروجی آنرا به بلوک Demux متصل کنید.
- دو بار بر روی بلوک State-Space کلیک کرده و پارامترهای سیستم را به شکل زیر وارد کنید:



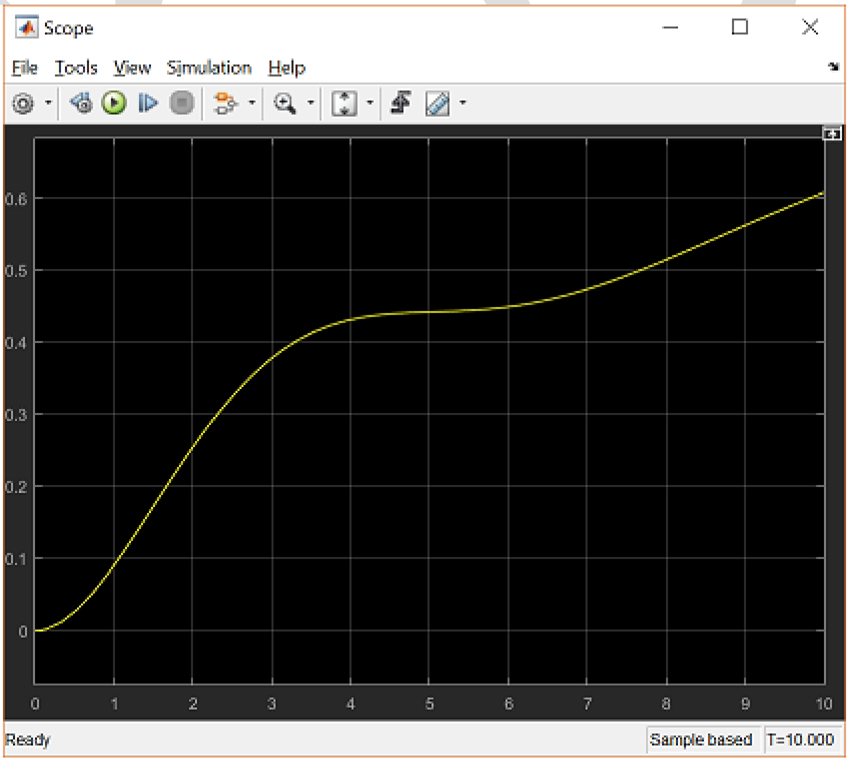
توجه شود که در بالا ماتریس C را بجای ماتریس [1 0 0]، یک ماتریس همانی ۳×۳ در نظر گرفتیم. دلیل این کار آنست که در کنترل فیدبک حالت فرض می‌کنیم تمامی متغیرهای حالت (و نه فقط خروجی سیستم) اندازه‌گیری می‌شود. اگر نمی‌توان تمامی حالات را اندازه‌گیری نمود لازم است که از مشاهده‌گر برای تخمین حالات غیر قابل اندازه‌گیری استفاده کنیم. برای جزئیات بیشتر به **فصل اول - بخش ششم: روش طراحی کنترلر فضای حالت** مراجعه کنید.

بعد از اتمام مراحل بالا، مدل سیمولینک باید به شکل زیر در بیاید:



تولید پاسخ حلقه باز و حلقه بسته

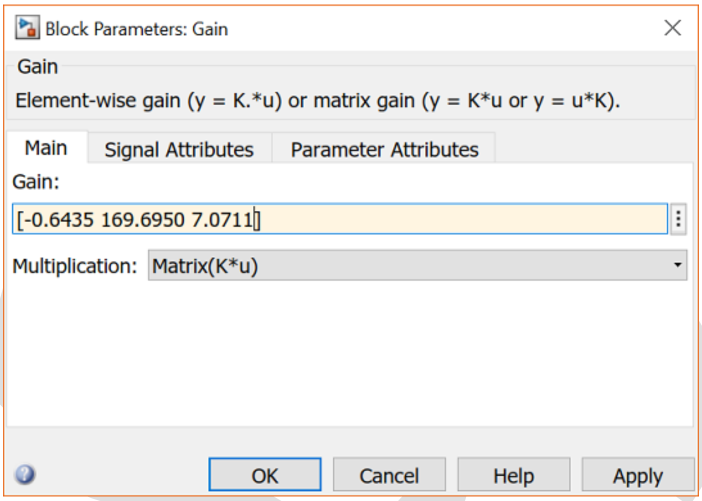
حال پاسخ پله‌ی حلقه باز را با اجرا کردن شبیه‌سازی به دست می‌آوریم. بعد از اتمام شبیه‌سازی، نمودار زیر را مشاهده می‌نمایید:



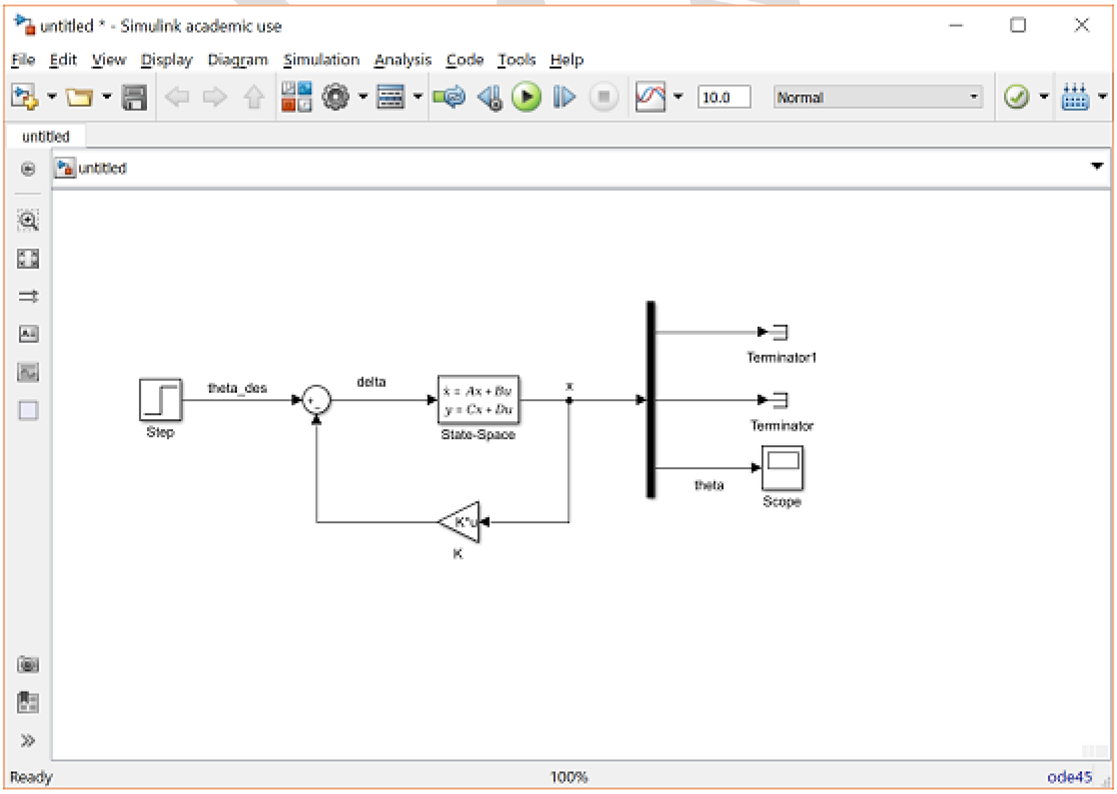
پاسخ سیستم ناپایدار و مشابه پاسخ به دست آمده در **بخش دوم: تحلیل سیستم** است. جهت مشاهده‌ی پاسخ پایدار حلقه کنترلی را بسته و از ماتریس بهره‌ی کنترلی K به دست آمده در **بخش ششم: طراحی کنترلر در فضای حالت** استفاده می‌کنیم. یادآوری می‌کنیم که این ماتریس بهره با استفاده از روش LQR به دست آمد و به صورت $K = [-0.6435\ 169.6950\ 7.0711]$ است.

جهت اضافه کردن کنترل فیدبک حالات به مدل خود، مراحل زیر را طی کنید:

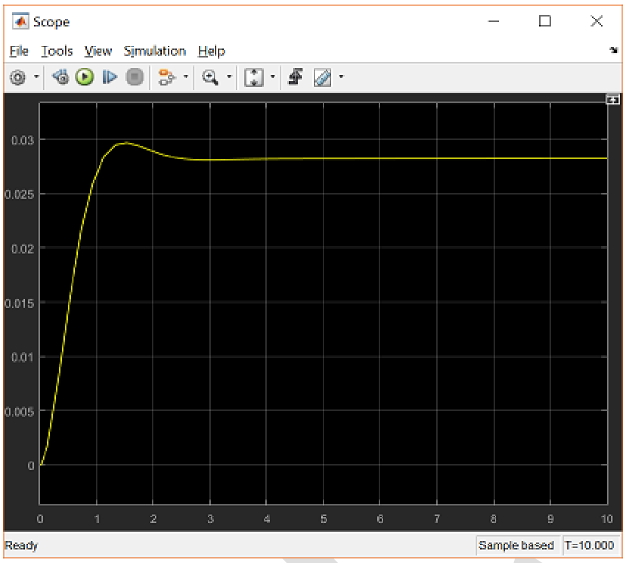
- یک بلوک Sum از کتابخانه‌ی Math Operations وارد کرده و با دوبار کلیک بر روی آن لیست علائم را به -+| در بیاورید. نماد "|" برای ایجاد فاصله بین ورودی‌های بلوک است. این بلوک را بین ورودی پله و بلوک State-Space قرار دهید.
- یک بلوک Gain از کتابخانه‌ی Math Operations وارد کرده و با زدن گزینه‌ی Ctrl-I آن را بچرخانید. سپس خروجی آن را به ورودی منفی بلوک Sum وصل کرده و ورودی آن را به خروجی بلوک State-Space متصل کنید.
- دو بار بر روی بلوک Gain کلیک کنید و اطلاعات زیر را وارد کنید. دقت نمایید که از منوی کشویی Multiplication گزینه‌ی Matrix(K*u) را انتخاب کنید.



با نام‌گذاری مناسب برای سیگنال‌ها، به مدل زیر خواهیم رسید:



سپس برنامه را اجرا کرده و پاسخ زیر را با دابل کلیک بر روی بلوک Scope مشاهده کنید:



این پاسخ شبیه پاسخ سیستم در **بخش ششم: طراحی کنترل در فضای حالت** می‌باشد.

اگر مایل به ارتقا الگوریتم کنترلی برای این سیستم می‌باشید به بخش بعد مراجعه نمایید.

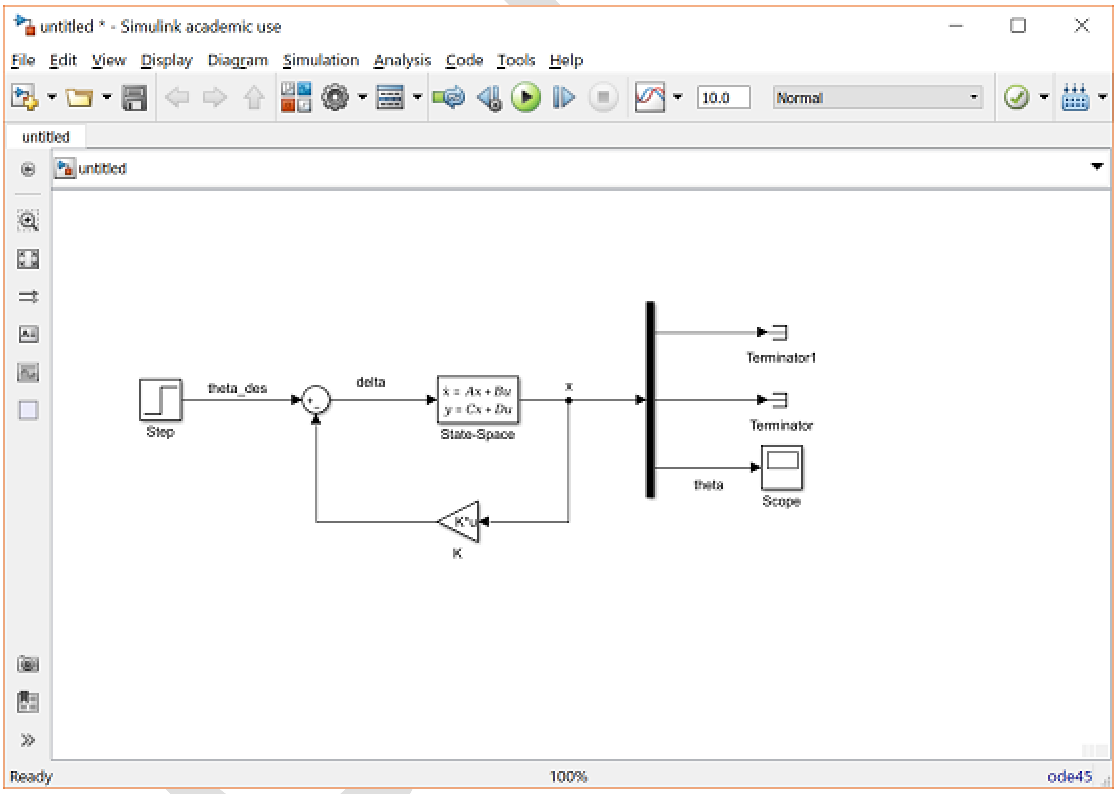
بخش نهم: طراحی کنترلر در سیمولینک

فهرست مطالب بخش

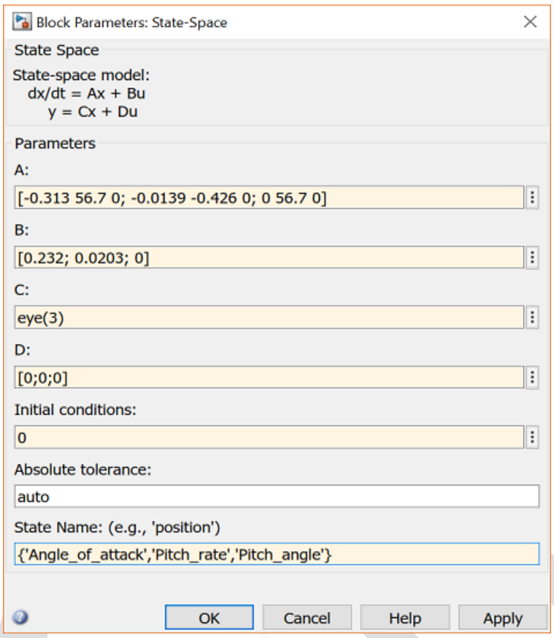
- کنترل فیدبک حالات با پیش‌جبران‌سازی
- مقاومت سیستم
- تنظیم خودکار PID با سیمولینک

کنترل فیدبک حالات با پیش‌جبران‌سازی

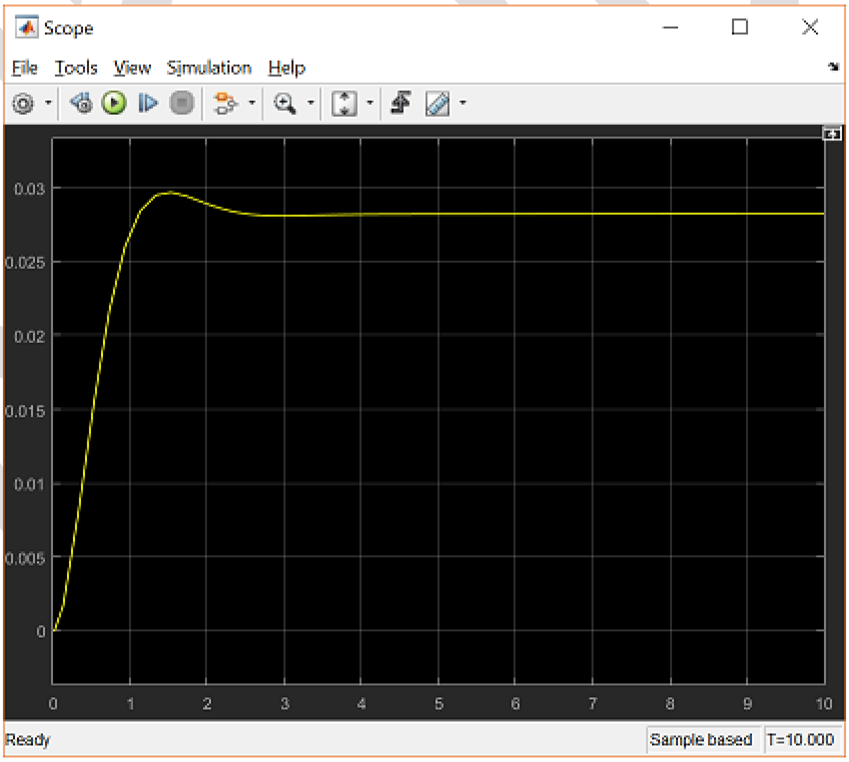
در این بخش از مدل ساخته شده در **بخش هشتم: مدل‌سازی سیمولینک** برای ارتقا و بررسی روش‌های مختلف کنترل استفاده می‌کنیم. مدل ساخته شده در بخش قبل را می‌توانید از روی سی‌دی بر روی کامپیوتر خود ذخیره نمایید. مدل کامل را در شکل زیر مشاهده می‌کنید:



در بالا بلوک State-Space با جزئیات زیر تعریف شده است و ماتریس C یک ماتریس همانی ۳×۳ می‌باشد. این تعریف به این معنی است که هر سه متغیر حالت به عنوان خروجی در نظر گرفته شده و می‌توانند در قانون کنترل فیدبک حالات استفاده شوند. اگر همه حالت‌ها قابل اندازه‌گیری نباشند باید از مشاهده‌گر استفاده کنیم.

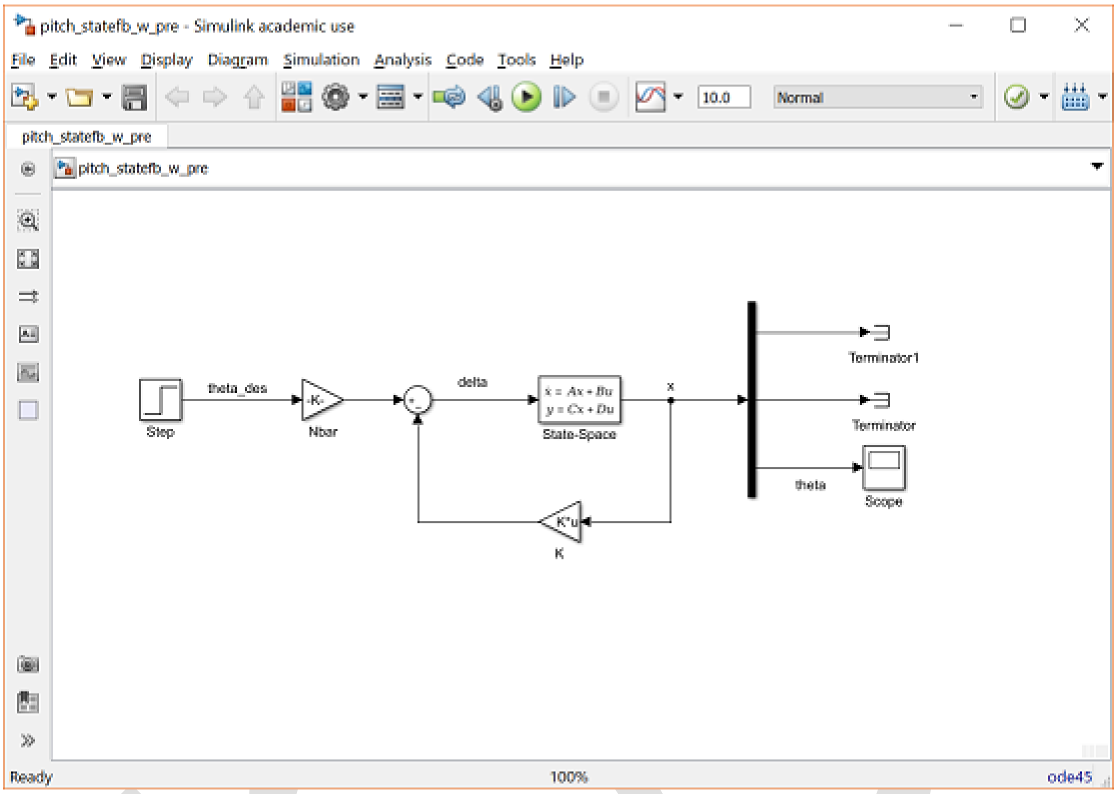


پاسخ سیستم حلقه بسته به دست آمده از شبیه‌سازی مدل بالا را در زیر مشاهده می‌کنید:

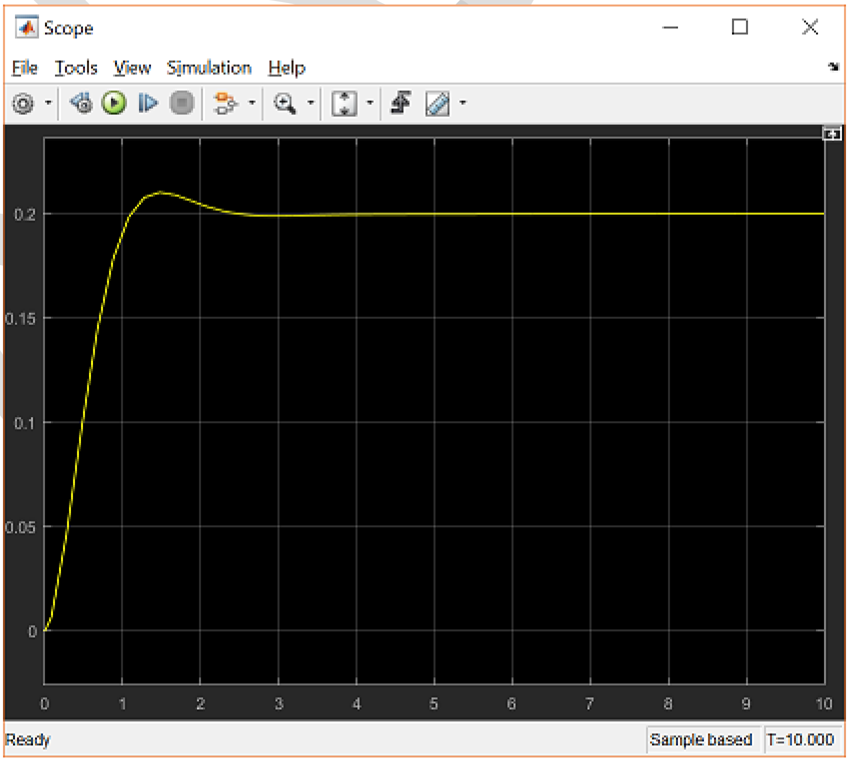


بررسی شکل بالا نشان می‌دهد که زمان نشست، زمان نمو و فراجهش ارضا شده‌اند. در حالی که خطای حالت ماندگار سیستم ارضا نشده و در ناحیه‌ی ۲٪ مقدار ۰/۲ نیست. در **بخش ششم: طراحی کنترلر در فضای حالت**، این کمبود با وارد کردن یک ضریب پیش‌جبران‌سازی به مقدار $\bar{N} = 7.0711$ برطرف شد.

این پیش‌جبران‌ساز را می‌توان با افزودن یک بلوک Gain از کتابخانه‌ی Math Operations به سیستم اضافه کرد. این بلوک را بین بلوک Step و بلوک Sum قرار دهید. دوبار بر روی این بلوک کلیک کرده و مقدار ۷/۰۷۱۱ را وارد کنید.



بعد از اجرای شبیه‌سازی، منحنی زیر را مشاهده می‌نمایید:

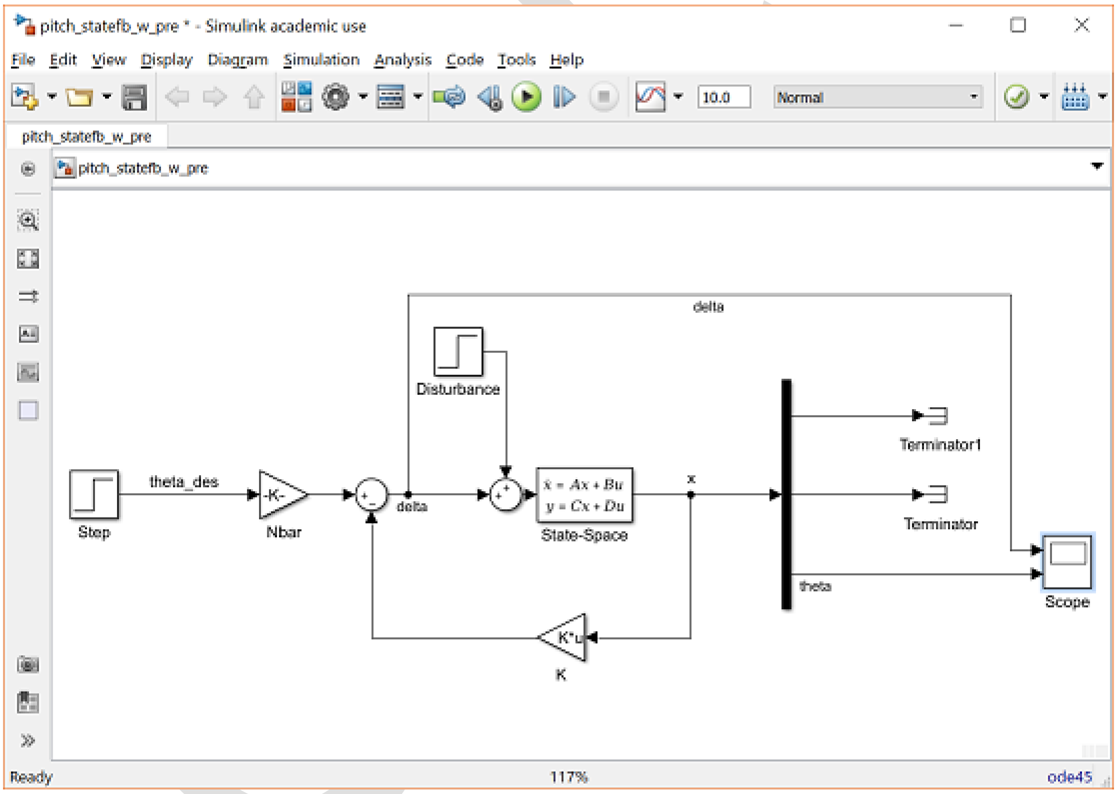


از نمودار بالا درمی‌یابیم که اضافه شدن پیش‌جبران‌ساز، خطای حالت ماندگار را به صفر سوق داده و حال تمامی الزامات طراحی برآورده شده‌اند.

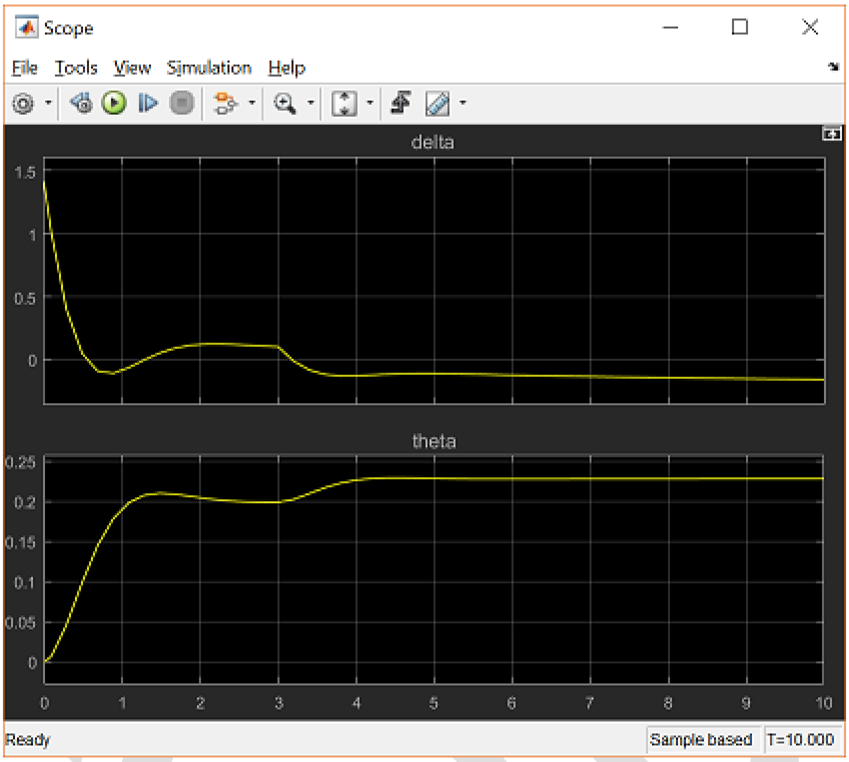
مقاومت سیستم

مشکل استفاده از پیش‌جبران‌ساز اینست که پیش‌جبران‌ساز بر اساس مدل سیستم محاسبه شده و در خارج حلقه‌ی فیدبک قرار گرفته است؛ همانطور که در مدل بالا مشاهده می‌نمایید، سیگنال خطا بعد از نقطه‌ی جمع تشکیل می‌شود. بنابراین در صورتی که خطایی در مدل سیستم یا اغتشاش نامعلومی وجود داشته باشد، پیش‌جبران‌ساز توانایی رفع آن را نداشته و خطای حالت ماندگار خواهیم داشت.

جهت بررسی این پدیده، مانند شکل زیر به مدل خود اغتشاش اضافه می‌کنیم. اغتشاش توسط یه بلوک Step با Final value برابر ۰/۲ و Step time برابر ۳ بوجود می‌آید. اغتشاش را همانطور که سیگنال کنترلی δ به سیستم وارد شده است، مدل می‌کنیم. یک بلوک Sum دیگر را وارد کرده و با لیست علائم |++ در مکان مناسب قرار دهید. همچنین بباید ورودی کنترلی سیستم را مشاهده کنیم، برای این کار بر روی بلوک Scope راست کلیک کرده، به برگه‌ی Signals & Ports رفته و تعداد درگاه‌های ورودی را برابر ۲ قرار دهید. اگر می‌خواهید دو محور داشته باشید، به منوی View در پنجره‌ی بلوک Scope رفته و چیدمان مورد نظر را انتخاب کنید. سپس یک انشعاب از سیگنال δ به ورودی جدید بلوک Scope متصل کنید. همچنین دقت کنید که سیگنال‌های وارد شده به این بلوک نام‌گذاری شده باشند. این کار با دوبار کلیک بر روی سیگنال و نوشتن نام مورد نظر صورت می‌گیرد.



اجرای شبیه‌سازی بالا به پاسخ زیر می‌انجامد:



شکل بالا نشان می‌دهد که چگونه اضافه شدن اغتشاش در زمان ۳ ثانیه باعث انحراف پاسخ از خروجی مطلوب ۰/۲ رادیان می‌شود و ضریب پیش‌جبران‌ساز قادر به حل این مشکل نمی‌باشد. ممکن است به یاد داشته باشید که انتگرال‌گیر به تصحیح نامعینی مانند اغتشاش کمک می‌کند. این روش در قسمت بعد مورد استفاده قرار می‌گیرد.

نامعینی مدل سیستم خطای دیگری است که باید در نظر داشته باشید. برای مثال اگر مدل واقعی سیستم ماتریس B برابر [0.232*1.4 0.0203*0.6 0]′ داشته باشد، کنترلر K طراحی شده در بالا منجر به ناپایداری سیستم می‌شود. این موضوع می‌تواند از طریق تغییر بلوک State-Space و تکرار شبیه‌سازی بالا یا به دست آوردن قطب‌های سیستم حلقه بسته (مقادیرویژه‌ی [A-BK]) بررسی شود. این قبیل مشکلات وقتی که از روش LQR استفاده می‌کنیم شایع است چرا که تنها عامل مهم در آن کنترلرها کمینه کردن تابع هزینه است در حالی که عوامل دیگری همچون مقاومت سیستم یا نامعینی در سیستم می‌تواند خود اهداف طراحی باشد. تکنیک‌های پیشرفته‌تر کنترل مقاوم جهت دستیابی به این اهداف وجود دارند.

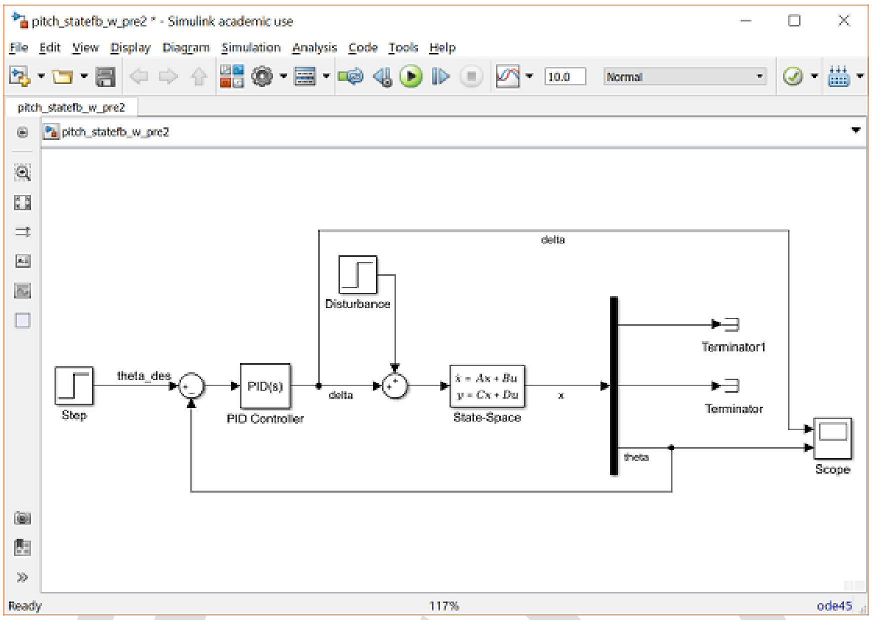
تنظیم خودکار PID با سیمولینک

همانگونه که گفته شد اضافه کردن کنترل انتگرالی به جبران‌ساز می‌تواند موجب از بین رفتن خطای ماندگار ناشی از اغتشاش یا نامعینی در سیستم گردد. می‌توانیم یک حالت جدید که بیانگر انتگرال خطا است را به بردار حالات سیستم اضافه کنیم و روش کنترل فضای حالت را دوباره پیاده نماییم. اما بجای آن از یک کنترلر PID با فرض تنها اندازه‌گیری از خروجی θ استفاده می‌کنیم. علاوه بر آن از قابلیت تنظیم خودکار PID سیمولینک استفاده می‌کنیم. توجه کنید که برای استفاده از این قابلیت باید جعبه‌ابزار Simulink Control Design را داشته باشید.

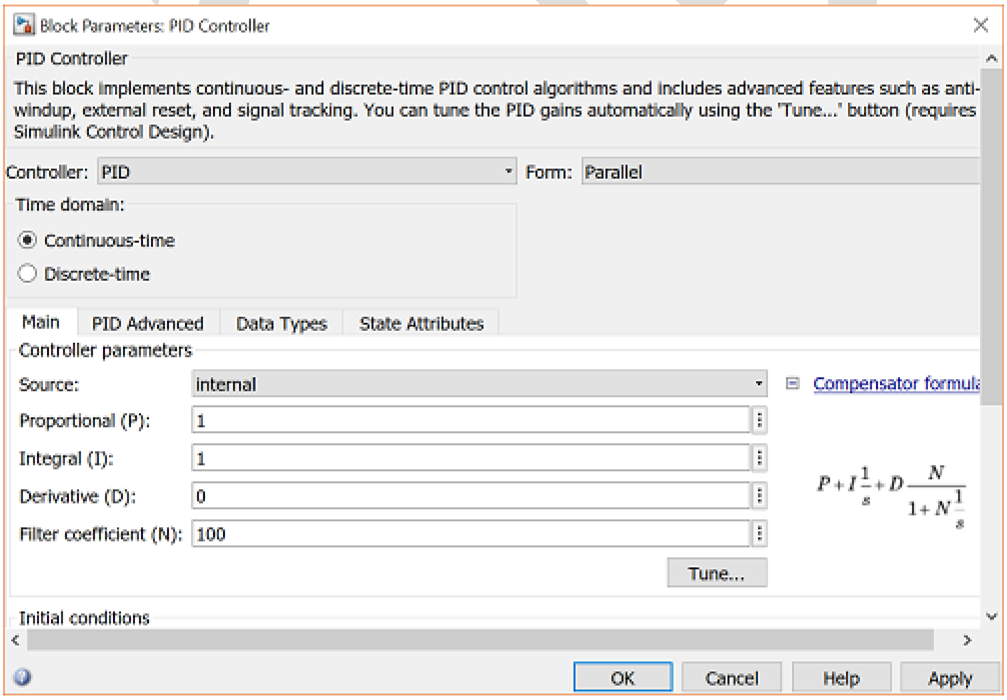
مراحل زیر را جهت طراحی کنترلر PID دنبال کنید:

- بلوک Gain برای فیدبک بهره‌ی K و پیش‌جبران‌ساز $\bar{N}$ را حذف کنید. همچنین سیگنال فیدبک از بردار حالت $\mathbf{x}$ را حذف کرده و فیدبک را از خروجی θ مجددا وصل کنید.
- بلوک PID Controller را از کتابخانه‌ی Continuous وارد کنید و آنرا دقیقا بعد از بلوک Sum فیدبک منفی قرار دهید.

نتیجه باید به شکل زیر دربیاید:



دوبار بر روی بلوک PID Controller کلیک کنید و پنجره زیر باز می‌شود:

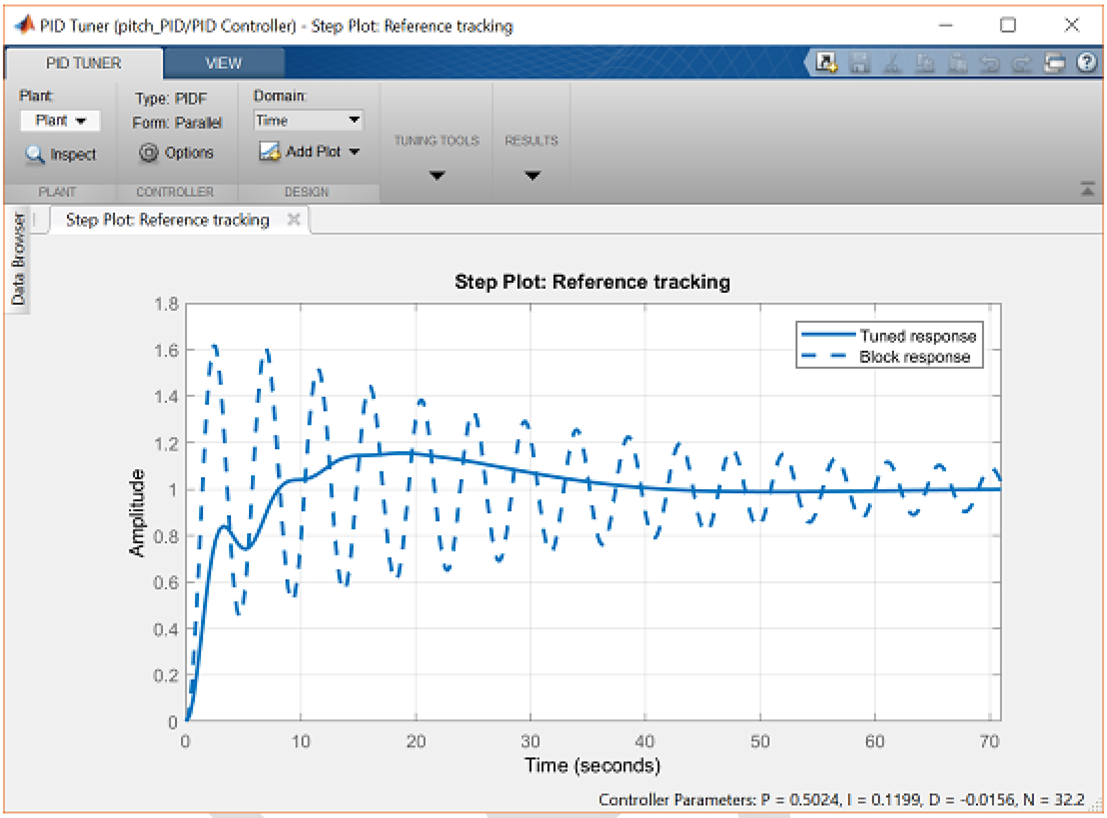


مقادیر پیش فرض بالا را در نظر می‌گیریم که شامل یک کنترلر PID به همراه یک فیلتر پایین گذر بر روی جمله‌ی مشتق می‌باشد. مشخصا ساختار کنترلر به فرمت زیر است که ضریب فیلتر (N) بیانگر ثابت زمانی فیلتر پایین گذر مرتبه اول (برابر (1/N)) می‌باشد:

$$C(s) = K_p + \frac{K_i}{s} + K_d s \frac{N}{s+N} \tag{۱}$$

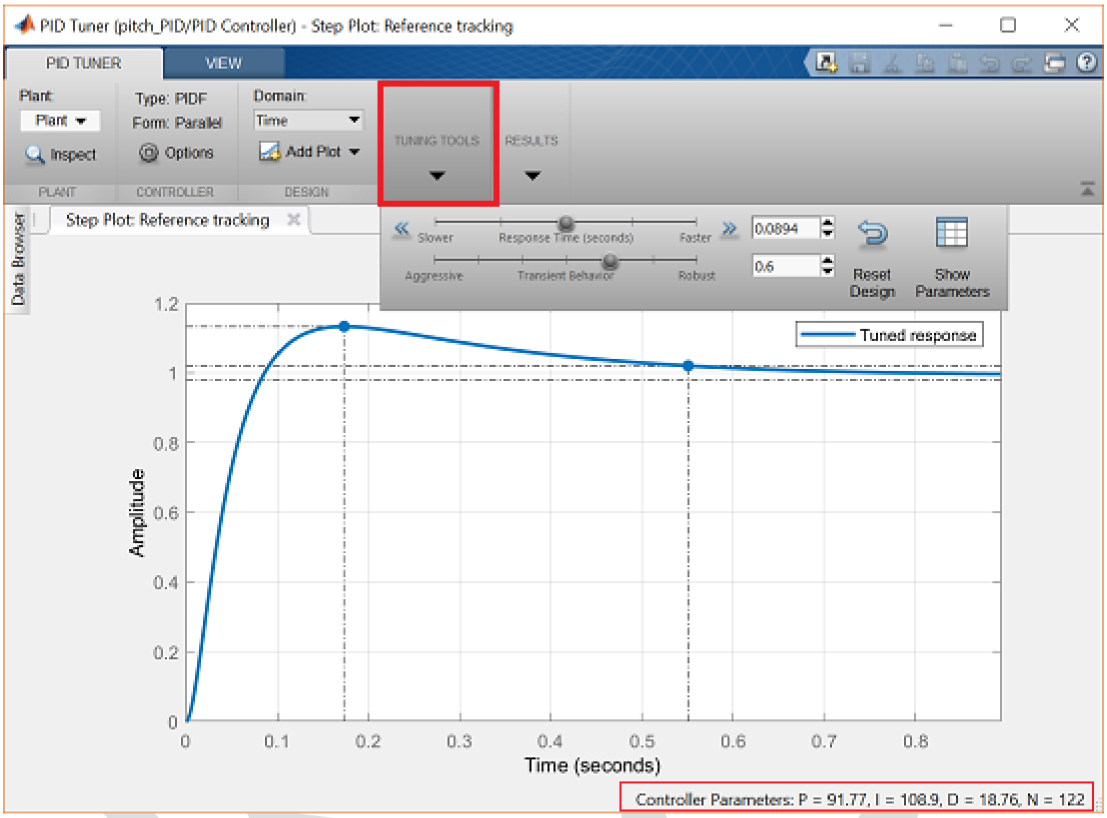
مقادیر پیش فرض آمده در بالا، به صورت دستی می‌توانند تغییر کنند. در هر حال ما از تنظیم خودکار PID استفاده می‌کنیم. این کار با زدن دکمه‌ی Tune آغاز می‌گردد. حال سیستم در حول نقطه‌ی کاری پیش‌فرض، خطی‌سازی شده و

بهره‌های PID به گونه‌ای تنظیم می‌گردد تا بین عملکرد و مقاومت سیستم تعادل برقرار کند. در مثال ما مدل از ابتدا خطی می‌باشد. شکل زیر پنجره‌ی باز شده را نمایش می‌دهد:

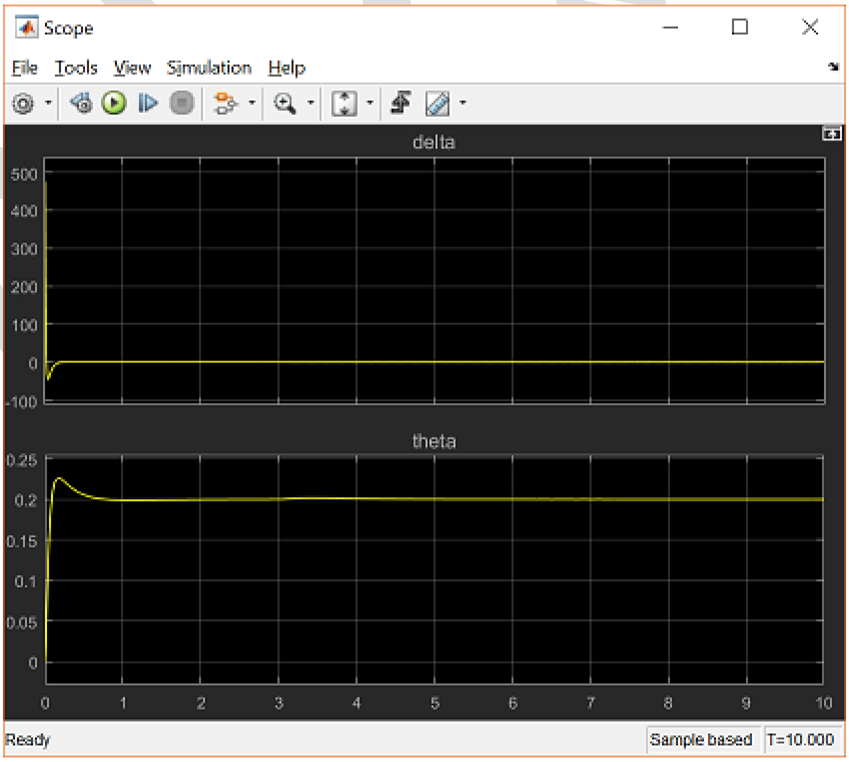


پاسخ سیستم با بهره‌های کنونی کنترلر به صورت خط چین و پاسخ سیستم با بهره‌های پیشنهاد شده برای تنظیم به صورت خط تو پر نمایش داده شده است. جهت شفاف سازی تصویر، بر روی گزینه‌ی Update Block در منوی RESULTS کلیک کنید تا بهره‌های تنظیم شده وارد مدل سیمولینک شود. حال می‌توانیم تیک گزینه‌ی Show response from block را در منوی Option برداریم زیرا کنترلر ما همان کنترلر پیشنهادی است. در نهایت با راست کلیک بر روی فضای نمودار و اضافه کردن مشخصات پاسخ پله، می‌توانیم متوجه شویم که الزامات طراحی برای پاسخ سیستم ارضا شده‌اند یا خیر. بررسی منحنی بالا نشان می‌دهد که پاسخ سیستم با کنترلر تنظیم شده بسیار کند است.

می‌توانیم سرعت پاسخ را در منوی Tuning Tools با حرکت لغزنده‌ی Response time به سمت راست افزایش دهیم. اگر محدوده‌ی لغزنده کافی نیست، می‌توانید با فشردن دکمه دو فلشه، مقیاس را تغییر دهید. به طور کلی ما قصد داریم سرعت سیستم را تا حدی بالا ببریم که به مقدار مورد نظر برسیم. پاسخ سریع‌تر عموما به بهای تلاش کنترلی بیشتر تمام می‌شود. رفتار سیستم وقتی زمان پاسخ برابر ۰/۰۸۹۴ ثانیه و بیشتر از مقدار مورد نظر ما است، نمایش داده شده است. مقدار سریع‌تر از مورد نیاز به این دلیل است که زمان نشست هدف ما حتی در حضور اغتشاش نیز ارضا شود.



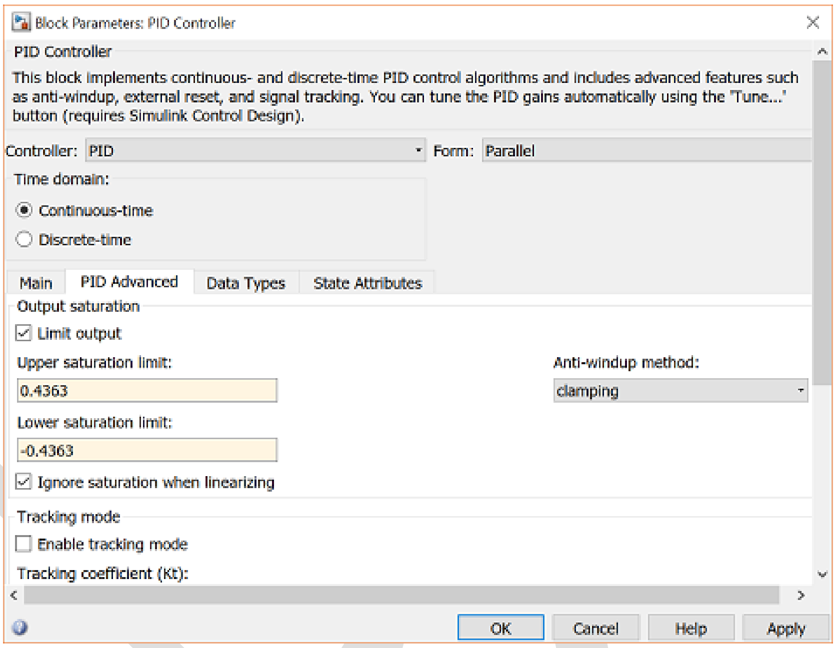
همچنین می‌توانید در سمت راست و پایین تصویر، بهره‌های کنترلر را مشاهده کنید. برای مشاهده‌ی عملکرد این کنترلر (شامل تلاش کنترلی مورد نیاز) شبیه‌سازی را اجرا کرده و دو بار بر روی بلوک Scope کلیک کنید:



با بررسی منحنی بالا، برخی موارد مشخص می‌شوند. اول، اغتشاش وارد شده در حالت پایا از بین می‌رود چرا که کنترلر PID استفاده شده دارای جمله‌ی انتگرالی است. علاوه بر آن تلاش کنترلی مورد نیاز در این کنترلر بسیار بیشتر از کنترلر

فیدبک حالات است. این احتمالا به این دلیل است که الگوریتم مورد استفاده برای تنظیم خودکار، هزینه‌ی تلاش کنترلی را در نظر نگرفته است یا اگر هم گرفته باشد وزن آن به اندازه‌ی استفاده شده در تابع هزینه‌ی روش LQR نیست.

در عمل احتمالا زاویه‌ی بالابرنده‌ی هواپیما δ چیزی حدود بین 25- درجه (0.4363- رادیان) تا 25+ درجه(0.4363 رادیان) می‌باشد. جهت اینکه مطمئن باشیم که از این محدوده خارج نمی‌شویم یک اشباع برای کنترلر قرار می‌دهیم. این کار توسط برگه‌ی PID Advanced در پنجره‌ی بلوک PID Controller انجام می‌شود؛ بدین صورت که تیک گزینه‌ی Limit output را زده و محدوده را برای Upper saturation limit و Lower saturation limit وارد می‌نماییم:

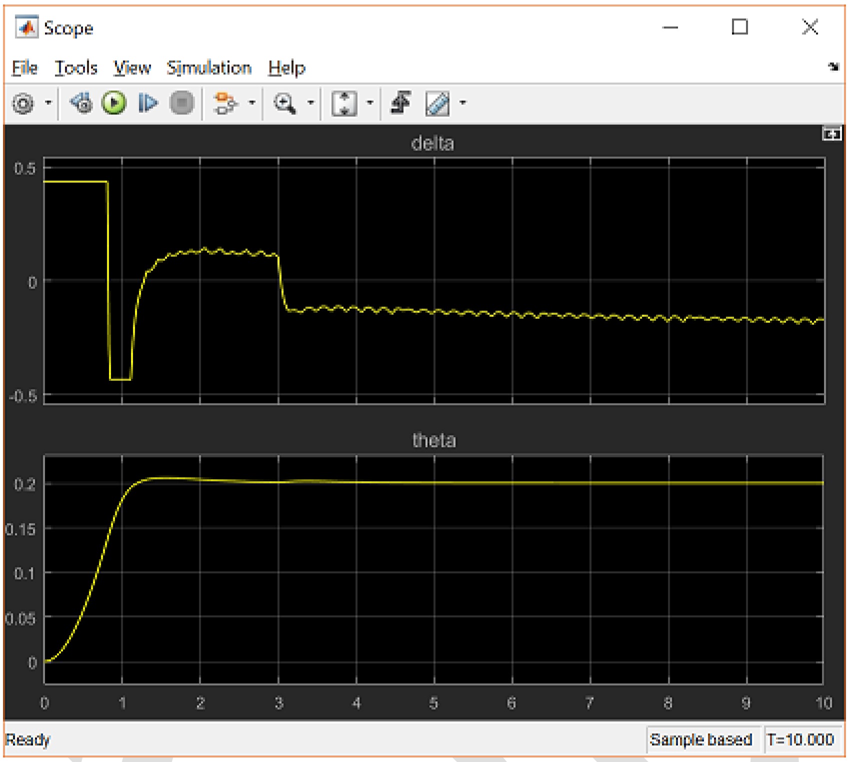


مشکلی که در هنگام اشباع عملگر با کنترلر PID بوجود می‌آید اینست که جمله‌ی انتگرالی به جمع خطاها ادامه داده و احتیاج به کنترل بیشتر و بیشتر دارد و این در حالتی است که تلاش کنترلی به حد اشباع خود رسیده است. مشکل از آنجایی شروع می‌شود که وقتی بالاخره خطا شروع به کاهش می‌کند، تلاش کنترلی همچنان اشباع می‌باشد چرا که انتگرال‌گیر مقدار بزرگی را انباشته کرده است. این کار سیستم را کند می‌کند چرا که زمان زیادی صرف می‌شود تا انتگرال‌گیر به آرامش[۳۶] برسد. بلوک PID Controller می‌تواند به حل این مشکل با استفاده از رویکرد Anti-windup کمک کند. این مورد نیز در برگه‌ی PID Advanced وجود دارد. ما برای روش Anti-windup از گزینه‌ی clamping استفاده می‌کنیم. این رویکرد هنگامی که کنترلر به اشباع می‌رسد، انتگرال‌گیر را متوقف کرده و هنگامی که عملگر دوباره به محدوده‌ی کاری بازگشت، انتگرال‌گیر را ادامه می‌دهد.

بعد از انتخاب این گزینه‌ها، شبیه‌سازی را اجرا کرده و مشاهده می‌کنیم که پاسخ، الزامات طراحی را حتی در ازای اغتشاش نیز ارضا کرده است. توجه کنید که تلاش کنترلی بین ۰/۴۳۶۳- و ۰/۴۳۶۳+ رادیان محدود شده و بر روی کاهش فراجهش سیستم نیز تاثیر گذار بوده است.

---

[۳۶] Unwind


Scope
File Tools View Simulation Help
delta
0.5
0
-0.5
theta
0.2
0.15
0.1
0.05
0
0 1 2 3 4 5 6 7 8 9 10
Ready
Sample based
T=10.000

فصل ششم: پاندول معکوس

بخش اول: مدل‌سازی سیستم

فهرست مطالب بخش

- صورت مسئله و نیازهای طراحی
- تحلیل نیرو و معادلات سیستم
- نمایش متلب

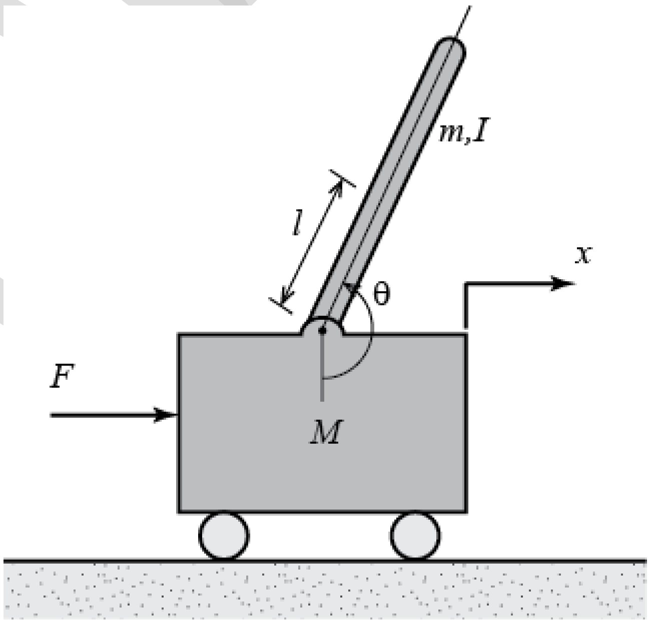
دستورهای کلیدی متلب در این بخش:

```
tf, ss, set
```

صورت مسئله و نیازهای طراحی

سیستم مورد بحث در این بخش، یک پاندول معکوس متصل به ارابه‌ی متحرک می‌باشد. سیستم پاندول معکوس یک مثال رایج در کتاب‌های کنترل و مقاله‌های تحقیقاتی می‌باشد. به علت ناپایدار بودن پاندول در صورت عدم وجود کنترل، یعنی با ثابت بودن ارابه، پاندول به راحتی می‌افتد. این سیستم بسیار پرطرفدار می‌باشد. علاوه بر آن دینامیک این سیستم غیر خطی است. هدف این سیستم کنترلی، برقرار نمودن تعادل پاندول معکوس به وسیله‌ی اعمال نیرو به ارابه می‌باشد. مثال واقعی که به صورت مستقیم با سیستم پاندول معکوس مرتبط است، سیستم کنترل راکت بوستر در هنگام تیک‌آف می‌باشد.

در این مثال، یک مسئله دو بعدی را که در آن پاندول در صفحه‌ی قائم حرکت می‌کند را در نظر بگیرید (شکل زیر). برای این سیستم، ورودی کنترل برابر نیروی F می‌باشد که ارابه را به صورت افقی حرکت داده و خروجی‌های سیستم، موقعیت زاویه‌ای پاندول θ و موقعیت افقی ارابه x می‌باشد.



برای این مثال، مقادیر زیر را در نظر بگیرید:

(M) جرم ارابه: 0.5 Kg

(m) جرم پاندول: 0.2 Kg

(b) ضریب اصطکاک ارابه: 0.1 N/m.sec

(l) فاصله از مرکز جرم پاندول: 0.3 m

(I) ممان اینرسی جرمی پاندول: 0.006 Kg.m^2

(F) نیروی وارد شده به ارابه

(x) موقعیت ارابه

(theta) زاویه پاندول از قائم

به دلیل اینکه روش‌های موجود در بخش‌های PID، مکان هندسی ریشه‌ها، پاسخ فرکانسی برای سیستم‌های تک ورودی و تک خروجی (SISO) بهتر جواب می‌دهند، تنها به موقعیت پاندول نیاز داریم. بنابراین هیچکدام از نیازهای طراحی، مربوط به موقعیت ارابه نمی‌شود. هرچند که بعد از طراحی کنترلر، تاثیر آنرا بر روی موقعیت ارابه بررسی می‌نماییم. برای این بخش‌ها کنترلی را برای بازگرداندن پاندول به موقعیت عمودی خود بعد از اعمال یک ضربه به ارابه، طراحی می‌کنیم. نیازهای طراحی به طور مشخص، بازگرداندن پاندول به موقعیت عمودی خود در زمان ۵ ثانیه می‌باشد و البته به شرطی که پاندول هیچگاه بیش از ۰/۰۵ رادیان از موقعیت عمودی خود در اثر ضربه‌ای با بزرگی 1 N.sec منحرف نشود. پاندول در موقعیت اولیه در حالت تعادل عمودی $\theta = \pi$ قرار دارد.

به طور خلاصه، نیازهای طراحی برای این سیستم به شرح زیر است:

- زمان نشست برای θ کمتر از ۵ ثانیه
- زاویه پاندول θ به قائم کمتر از ۰/۰۵ رادیان

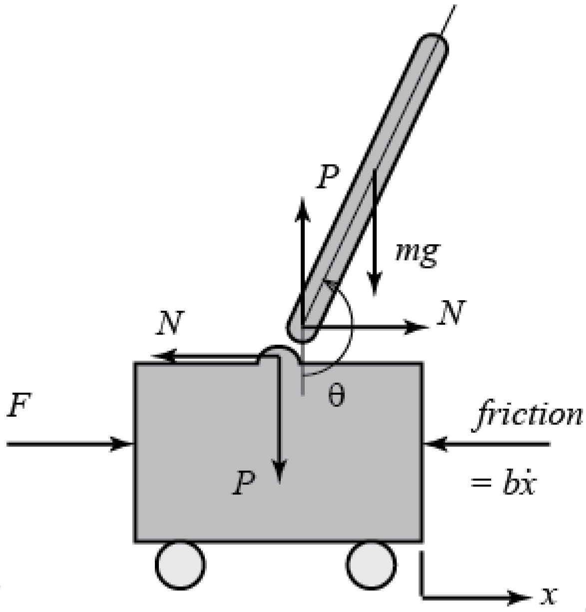
با استفاده از روش‌های طراحی فضای حالت، طرح مسئله‌ی چند خروجی ساده‌تر می‌باشد. برای این مثال، سیستم پاندول معکوس، تک ورودی و چند خروجی (SIMO) می‌باشد. بنابراین برای بخش فضای حالت، مثال پاندول معکوس را برای کنترل زاویه پاندول و همچنین موقعیت ارابه انجام می‌دهیم. برای چالش برانگیز بودن کنترل در این قسمت، ورودی پله ۰/۲ متر برای موقعیت ارابه را نیز در نظر می‌گیریم. تحت این شرایط، زمان نشست برای موقعیت ارابه کمتر از ۵ ثانیه و زمان نمو کمتر از ۰/۵ ثانیه مد نظر می‌باشد. همچنین مطلوبست زمان نشست موقعیت عمودی پاندول کمتر از ۵ ثانیه و عدم انحراف زاویه پاندول بیش از ۲۰ درجه (۰/۳۵ رادیان) از موقعیت عمودی خود.

به طور خلاصه، نیازهای طراحی برای مثال پاندول معکوس در فضای حالت عبارتست از:

- زمان نشست برای x و θ کمتر از ۵ ثانیه
- زمان نمو برای x کمتر از ۰/۵ ثانیه
- زاویه پاندول θ هیچگاه بیش از ۲۰ درجه (۰/۳۵ رادیان) از موقعیت عمودی منحرف نشود
- خطای حالت ماندگار کمتر از ۲٪ برای x و θ

تحلیل نیرو و معادلات سیستم

در شکل زیر دیاگرام آزاد هر دو المان سیستم پاندول معکوس رسم شده است:



برآیند نیروهای افقی وارد به ارابه در دیاگرام جسم آزاد، معادله زیر را به دست می‌دهد:

$$M\ddot{x} + b\dot{x} + N = F \qquad (۱)$$

همچنین می‌توان برآیند نیروهای عمودی وارد بر ارابه را محاسبه کرد اما اطلاعات مفیدی به دست نخواهد آمد.

برآیند نیروهای افقی پاندول در دیاگرام جسم آزاد، معادله‌ی زیر را برای نیروی عکس العمل N محاسبه می‌کند:

$$N = m\ddot{x} + ml\ddot{\theta}cos\theta - ml\dot{\theta}^2 sin\theta \qquad (۲)$$

اگر این معادله را در معادله‌ی اول جاگذاری نماییم، یکی از دو معادله‌ی حاکم بر سیستم به دست می‌آید:

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}cos\theta - ml\dot{\theta}^2 sin\theta = F \qquad (۳)$$

برای به دست آوردن معادله‌ی دوم حرکت سیستم، برآیند نیروهای عمودی پاندول را به دست می‌آوریم. حل سیستم در راستای این محور، محاسبات را بسیار ساده می‌نماید. در این صورت معادله زیر به دست می‌آید:

$$Psin\theta + Ncos\theta - mgsin\theta = ml\ddot{\theta} + m\ddot{x}cos\theta \qquad (۴)$$

برای حذف کردن P و N در معادله‌ی بالا، برآیند گشتاورها حول مرکز دوران پاندول را به دست می‌آوریم:

$$-Plsin\theta - Nlcos\theta = I\ddot{\theta} \qquad (۵)$$

با ترکیب دو معادله‌ی اخیر، معادله‌ی حاکم دوم به دست می‌آید:

$$(I+ml^2)\ddot{\theta} + mglsin\theta = -ml\ddot{x}cos\theta \qquad (۶)$$

به علت اینکه روش‌های تحلیل و کنترل که در بخش‌های آینده بر روی این مثال پیاده می‌شوند تنها بر روی سیستم‌های خطی اعمال می‌شوند، لازم است تا این مجموعه از معادلات را خطی‌سازی نماییم. این خطی‌سازی باید حول نقطه‌ی تعادل عمودی $\theta = \pi$ انجام گردد و فرض بر این است که سیستم در محدوده‌ی کوچکی حول این نقطه نوسان می‌کند. این فرضیات بسیار معقول می‌باشند زیرا می‌خواهیم پاندول تحت کنترل، بیش از ۲۰ درجه از موقعیت عمودی خود منحرف نشود. متغیر ϕ که بیانگر میزان انحراف موقعیت پاندول از حالت تعادل می‌باشد را به صورت $\theta = \pi + \phi$ تعریف می‌نماییم. دوباره با فرض انحرافات کوچک (ϕ) از نقطه‌ی تعادل، می‌توان از تقریب‌های زیر در معادلات غیر خطی سیستم استفاده نمود:

$$cos\theta = \cos(\pi + \phi) \approx -1 \qquad (۷)$$

$$sin\theta = \sin(\pi + \phi) \approx -\phi \tag{۸}$$

$$\dot{\theta}^2 = \dot{\phi}^2 \approx 0 \tag{۹}$$

بعد از جاگذاری تقریب‌های بالا در معادلات غیر خطی سیستم، به دو معادله‌ی خطی شده‌ی حرکت خواهیم رسید. متغیر u به جای ورودی F قرار داده شده است:

$$(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x} \tag{۱۰}$$

$$(M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} = u \tag{۱۱}$$

۱. تابع تبدیل

برای به دست آوردن توابع تبدیل معادلات سیستم خطی شده، ابتدا باید با فرض صفر بودن شرایط اولیه، از معادلات سیستم تبدیل لاپلاس گرفت. نتیجه‌ی این تبدیل لاپلاس در زیر مشاهده می‌شود:

$$(I + ml^2)\boldsymbol{\Phi}(\mathrm{s})\mathrm{s}^2 - mgl\boldsymbol{\Phi}(\mathrm{s}) = \mathrm{mlX}(\mathrm{s})\mathrm{s}^2 \tag{۱۲}$$

$$(M + m)X(s)s^2 + bX(s)s - ml\boldsymbol{\Phi}(s)s^2 = U(s) \tag{۱۳}$$

یادآوری می‌شود که نمایش تابع تبدیل، رابطه‌ی بین تک ورودی و تک خروجی را در یک زمان نشان می‌دهد. برای به دست آوردن تابع تبدیل اولیه برای خروجی $\Phi(s)$ و ورودی $U(s)$، باید $X(s)$ را از معادلات بالا حذف کنیم. معادله اول را برای $X(s)$ حل می‌کنیم:

$$X(s) = \left[\frac{I + ml^2}{ml} - \frac{g}{s^2}\right]\boldsymbol{\Phi}(s) \tag{۱۴}$$

سپس معادله‌ی به دست آمده را در رابطه‌ی دوم قرار می‌دهیم:

$$(M + m)\left[\frac{I + ml^2}{ml} - \frac{g}{s^2}\right]\boldsymbol{\Phi}(s)s^2 + b\left[\frac{I + ml^2}{ml} - \frac{g}{s^2}\right]\boldsymbol{\Phi}(s)s - ml\boldsymbol{\Phi}(s)s^2 = U(s) \tag{۱۵}$$

با مرتب‌سازی این معادله، تابع تبدیل به دست می‌آید:

$$\frac{\boldsymbol{\Phi}(s)}{U(s)} = \frac{\frac{ml}{q}s^2}{s^4 + \frac{b(I + ml^2)}{q}s^3 - \frac{(M + m)mgl}{q}s^2 - \frac{bmgl}{q}s} \tag{۱۶}$$

که q برابر است با:

$$q = [(M + m)(I + ml^2) - (ml)^2] \tag{۱۷}$$

با مشاهده‌ی تابع تبدیل بالا، می‌توان فهمید که یک قطب و یک صفر در مبدا وجود دارد. با خنثی شدن این صفر و قطب، تابع تبدیل به شکل زیر در می‌آید:

$$P_{pend}(s) = \frac{\boldsymbol{\Phi}(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I + ml^2)}{q}s^2 - \frac{(M + m)mgl}{q}s - \frac{bmgl}{q}} \quad \left[\frac{rad}{N}\right] \tag{۱۸}$$

در قدم دوم، تابع تبدیل برای موقعیت ارابه $X(s)$ را به عنوان خروجی به روش مشابه به دست می‌آوریم:

$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I + ml^2)s^2 - gml}{q}}{s^4 + \frac{b(I + ml^2)}{q}s^3 - \frac{(M + m)mgl}{q}s^2 - \frac{bmgl}{q}s} \quad \left[\frac{m}{N}\right] \tag{۱۹}$$

۲. فضای حالت

می‌توان با تبدیل معادلات حرکت خطی شده در بالا، آنها را به مجموعه‌ای از معادلات دیفرانسیل مرتبه اول، به فرم فضای حالت نمایش داد. به علت خطی بودن معادلات، می‌توان آنها را در فرم ماتریسی استاندارد به شکل زیر نمایش داد:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\phi}\\ \ddot{\phi}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix}+\begin{bmatrix}0\\ \frac{I+ml^2}{I(M+m)+Mml^2}\\ 0\\ \frac{ml}{I(M+m)+Mml^2}\end{bmatrix}u \qquad (۲۰)$$

$$y=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix}+\begin{bmatrix}0\\ 0\end{bmatrix}u \qquad (۲۱)$$

ماتریس C دارای دو ردیف می‌باشد زیرا موقعیت ارابه و پاندول، بخشی از خروجی بوده و موقعیت ارابه، المان اول خروجی y و انحراف پاندول از موقعیت تعادل خود، المان دوم y می‌باشد.

نمایش متلب

۱. تابع تبدیل

می‌توانیم توابع تبدیلی که در بالا به دست آوردیم را با استفاده از دستورات زیر در متلب وارد نمود. همچنین می‌توان خروجی‌ها را نام‌گذاری نمود تا اختلاف بین موقعیت ارابه و پاندول را به دست آورد. با اجرای این کد، خروجی‌های زیر به نمایش در می‌آید:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;

q = (M+m)*(I+m*l^2)-(m*l)^2;

s = tf('s');

P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);

P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);

sys_tf = [P_cart ; P_pend];

inputs = {'u'};
outputs = {'x'; 'phi'};

set(sys_tf,'InputName',inputs)
set(sys_tf,'OutputName',outputs)
```

```
sys_tf
```

```
sys_tf =


  From input "u" to output...

                         4.182e-06 s^2 - 0.0001025

   x:  ----------------------------------------------------------

       2.3e-06 s^4 + 4.182e-07 s^3 - 7.172e-05 s^2 - 1.025e-05 s



                                1.045e-05 s

   phi:  -----------------------------------------------------

         2.3e-06 s^3 + 4.182e-07 s^2 - 7.172e-05 s - 1.025e-05


Continuous-time transfer function.
```

۲. فضای حالت

همچنین می‌توانیم این سیستم را با معادلات فضای حالت نمایش دهیم. دستورات متلب که در زیر آمده است، یک مدل فضای حالت برای پاندول معکوس ایجاد کرده و خروجی آن بعد از اجرای کد به شکل زیر به دست می‌آید. در اینجا هم می‌توان برای ورودی‌ها، خروجی‌ها و حالت‌ها، نام‌هایی انتخاب کرد تا مدل قابل فهم‌تر باشد.

```
M = .5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;



p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices

A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
```

```
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs)
```

```
sys_ss =

  A =
                 x    x_dot      phi  phi_dot
   x             0        1        0        0
   x_dot         0  -0.1818    2.673        0
   phi           0        0        0        1
   phi_dot       0  -0.4545    31.18        0

  B =
               u
   x           0
   x_dot   1.818
   phi         0
   phi_dot 4.545

  C =
             x    x_dot      phi  phi_dot
   x         1        0        0        0
   phi       0        0        1        0

  D =
       u
```

```
   x    0

   phi  0


Continuous-time state-space model.
```

برای تبدیل مدل فضای حالت بالا به فرم تابع تبدیل، می‌توان از دستور tf استفاده نمود که در ادامه آورده شده است. همچنین می‌توان تابع تبدیل را با دستور ss به فرم فضای حالت تبدیل نمود:

```
sys_tf = tf(sys_ss)
```

```
sys_tf =


  From input "u" to output...

          1.818 s^2 + 1.615e-15 s - 44.55

   x:  --------------------------------------

       s^4 + 0.1818 s^3 - 31.18 s^2 - 4.455 s


                4.545 s - 1.277e-16

   phi:  ----------------------------------

         s^3 + 0.1818 s^2 - 31.18 s - 4.455


Continuous-time transfer function.
```

با بررسی خروجی‌های نشان داده شده، بعضی از جملات با ضرایب بسیار کوچک ایجاد شده‌اند. این جملات در واقع صفر بوده و به علت خطای رند کردن عددی که در الگوریتم‌های تبدیل متلب وجود دارد بوجود آمده‌اند. اگر این ضرایب را برابر صفر قرار دهید، مدل تابع تبدیل بالا با تابع تبدیل به دست آمده در بخش قبل یکسان خواهد شد.

بخش دوم: تحلیل سیستم

فهرست مطالب بخش

- پاسخ ضربه‌ی حلقه باز
- پاسخ پله‌ی حلقه باز

دستورهای کلیدی متلب در این بخش:

```
tf , ss , zpkdata , impulse , lsim
```

با توجه به مسئله‌ی مطرح شده، تابع تبدیل حلقه باز برای سیستم پاندول معکوس به شکل زیر به دست آمد:

$$P_{pend}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \quad \left[\frac{rad}{N}\right] \tag{۱}$$

$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \quad \left[\frac{m}{N}\right] \tag{۲}$$

که q برابر است با:

$$q = [(M+m)(I+ml^2) - (ml)^2] \tag{۳}$$

یادآوری می‌شود که دو تابع تبدیل به دست آمده در بالا فقط برای مقادیر کوچک زاویه ϕ صادق هستند که ϕ جابجایی زاویه‌ای پاندول از حالت عمودی می‌باشد. همچنین زاویه مطلق پاندول θ برابر با $\pi + \phi$ می‌باشد.

با فرض پاسخ پاندول به ضربه‌ی 1 Nsec وارد شده به ارابه، نیازهای طراحی برای پاندول عبارتند از:

- زمان نشست برای θ کمتر از ۵ ثانیه
- زاویه پاندول θ هیچگاه بیشتر از ۰/۰۵ رادیان از حالت عمودی نشود

علاوه بر آن، خواسته‌های حالت سیستم برای ورودی پله ۰/۲ برای موقعیت ارابه عبارتند از:

- زمان نشست برای x و θ کمتر از ۵ ثانیه
- زمان نمو برای x کمتر از ۰/۵ ثانیه
- زاویه پاندول θ هیچگاه بیشتر از ۲۰ درجه (۰/۳۵ رادیان) از حالت عمودی نشود

پاسخ ضربه‌ی حلقه باز

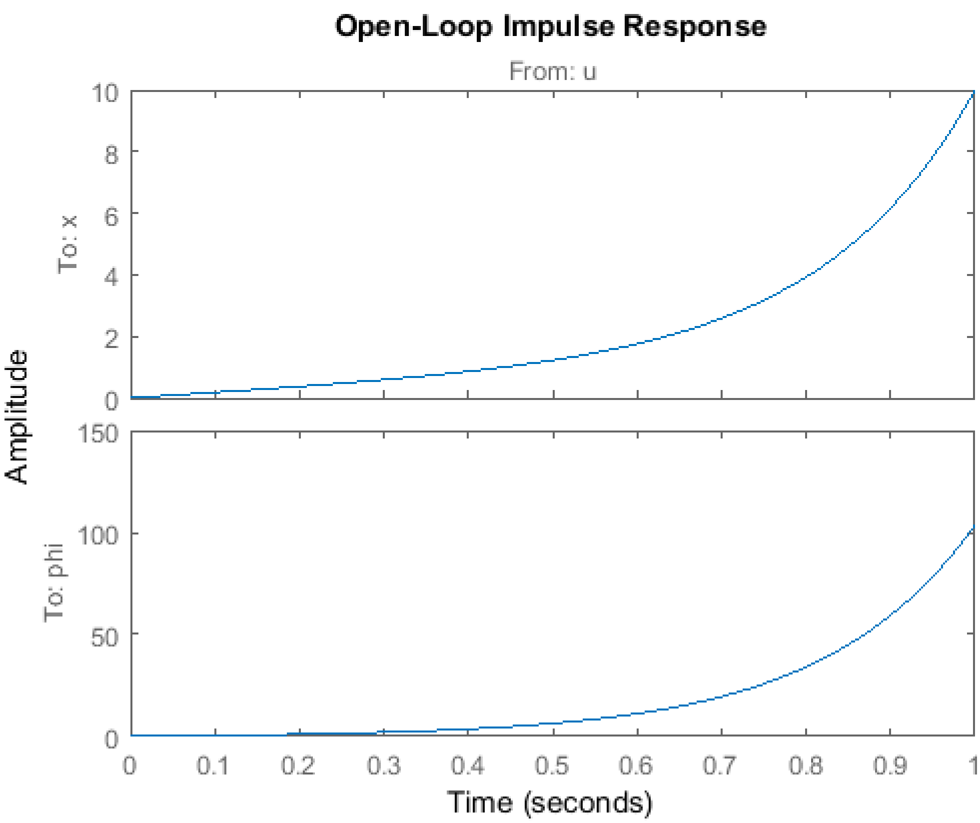
قدم اول مشاهده‌ی پاسخ حلقه باز سیستم پاندول معکوس می‌باشد. یک ام‌فایل جدید ساخته و دستورات زیر را در آن وارد کنید تا مدل سیستم تعریف شود:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;
```

```
g = 9.8;

l = 0.3;

q = (M+m)*(I+m*l^2)-(m*l)^2;

s = tf('s');

P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);

P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);

sys_tf = [P_cart ; P_pend];

inputs = {'u'};
outputs = {'x'; 'phi'};

set(sys_tf,'InputName',inputs)
set(sys_tf,'OutputName',outputs)
```

حال می‌توانیم پاسخ ضربه‌ی حلقه باز سیستم را بررسی کنیم. یعنی پاسخ سیستم را در صورت اعمال نیروی ضربه‌ای وارد بر ارابه با دستور impulse مشاهده کنیم. دستور زیر را در انتهای امفایل ساخته شده وارد کرده و آنرا در متلب وارد کنید تا نمودار زیر به دست آید:

```
t=0:0.01:1;

impulse(sys_tf,t);

title('Open-Loop Impulse Response')
```



همانطور که از نمودار می‌توان مشاهده نمود، پاسخ سیستم اصلا رضایت‌بخش نیست. در واقع سیستم حلقه باز پایدار نمی‌باشد. هرچند دیده می‌شود که موقعیت پاندول از ۱۰۰ رادیان (۱۵ دور) رد شده است، اما مدل ما تنها برای مقادیر کوچک ϕ معتبر است. همچنین مشاهده می‌کنید که موقعیت ارابه به اندازه‌ی بینهایت به راست رسیده است، هرچند که برای ورودی نیروی ضربه‌ای، شرطی برای موقعیت ارابه وجود ندارد.

همچنین می‌توان از قطب‌های سیستم پاسخ زمانی آنرا پیش‌بینی کرد. از آنجایی که سیستم ما دارای دو خروجی و یک ورودی می‌باشد، آنرا با دو تابع تبدیل تعریف نمودیم. به طور کلی، تمامی توابع تبدیل از هر ورودی به هر خروجی برای یک سیستم چند ورودی و چند خروجی (MIMO)، قطب‌های یکسان (اما صفرهای متفاوت) دارند مگر اینکه خنثی سازی صفر و قطب وجود داشته باشد. می‌توانیم با دستور zpkdata صفرها و قطب‌های سیستم را بررسی کنیم. پارامتر 'v' در کد زیر، قطب‌ها و صفرها را به صورت بردار ستونی باز می‌گرداند.

```
[zeros poles] = zpkdata(P_pend,'v')
```

```
zeros =

     0

poles =

    5.5651

   -5.6041

   -0.1428
```

بر همین منوال، صفرها و قطب‌های سیستم برای خروجی موقعیت ارابه به شکل زیر به دست می‌آید:

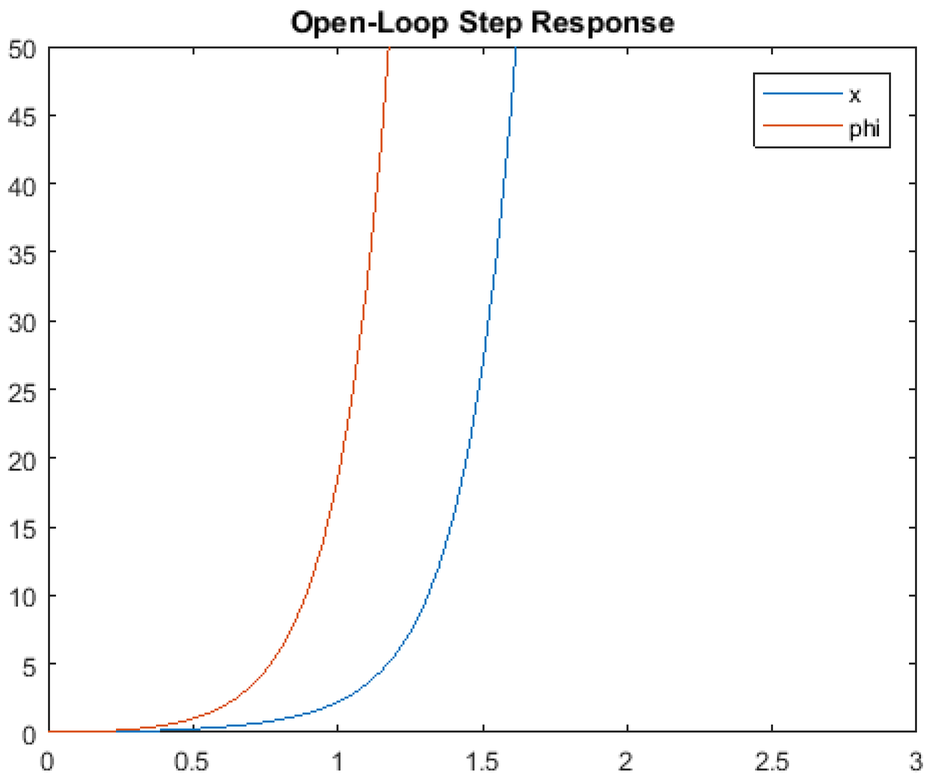
```
[zeros poles] = zpkdata(P_cart,'v')
```

```
zeros =

    4.9497

   -4.9497

poles =

         0

    5.5651

   -5.6041

   -0.1428
```

پاسخ پله‌ی حلقه باز

چون سیستم ما دارای یک قطب با قسمت حقیقی مثبت می‌باشد، پاسخ آن به ورودی پله نیز بی‌کران می‌شود. برای صحت این موضوع، از دستور lsim که پاسخ مدل‌های LTI به ورودی دلخواه را شبیه‌سازی می‌کند استفاده می‌کنیم. در این حالت، یک ورودی پله ۱ نیوتنی استفاده می‌شود. با اضافه کردن کد زیر در انتهای ام‌فایل و اجرای آن، نمودار زیر ایجاد می‌شود:

```
t = 0:0.05:10;

u = ones(size(t));

[y,t] = lsim(sys_tf,u,t);

plot(t,y)

title('Open-Loop Step Response')
axis([0 3 0 50])
legend('x','phi')
```



همچنین می‌توان بعضی از ویژگی‌های مهم پاسخ را با استفاده از دستور lsiminfo به دست آورد:

```
step_info = lsiminfo(y,t);

cart_info = step_info(1)

pend_info = step_info(2)
```

```
cart_info =

  struct with fields:
```

```
    SettlingTime: 9.9959
             Min: 0
         MinTime: 0
             Max: 8.7918e+21
         MaxTime: 10
pend_info =
  struct with fields:

    SettlingTime: 9.9959
             Min: 0
         MinTime: 0
             Max: 1.0520e+23
         MaxTime: 10
```

نتایج بالا، تردید ما را از ناپایدار بودن پاسخ سیستم به ورودی پله به یقین می‌رساند.

از تحلیل انجام شده مشخص است که برای بهبود پاسخ سیستم، باید کنترلی برای سیستم طراحی گردد. در این فصل از چهار کنترلر PID، مکان هندسی ریشه‌ها، پاسخ فرکانسی و فضای حالت استفاده می‌شود.

نکته: جواب‌های به دست آمده در بخش‌های PID، مکان هندسی ریشه‌ها و پاسخ فرکانسی ممکن است کنترلر عملی را نتیجه ندهند. همانطور که گفته شد، وقتی پاندول معکوس را به عنوان یک سیستم تک ورودی تک خروجی در نظر می‌گیریم، از موقعیت ارابه x چشم پوشی می‌کنیم. در هر یک از این کنترلرها، در صورت امکان، موقعیت ارابه را در حضور پیاده‌سازی کنترلر بررسی می‌کنیم.

بخش سوم: طراحی کنترلر PID

فهرست مطالب بخش

- ساختار سیستم
- کنترل PID
- موقعیت ارابه چگونه تغییر می‌کند؟

دستورهای کلیدی متلب در این بخش:

```
tf , impulse , feedback , pid
```

در این فصل به طراحی کنترلر PID برای سیستم پاندول معکوس می‌پردازیم. در روند طراحی، سیستم را به صورت تک ورودی تک خروجی که با تابع تبدیل زیر تعریف شده است در نظر می‌گیریم. برای طراحی، زاویه‌ی پاندول را بدون در نظر گرفتن موقعیت ارابه کنترل می‌کنیم.

$$P_{pend}(s) = \frac{\mathbf{\Phi}(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \quad \left[\frac{rad}{N}\right] \qquad (۱)$$

که:

$$q = [(M+m)(I+ml^2) - (ml)^2] \qquad (۲)$$

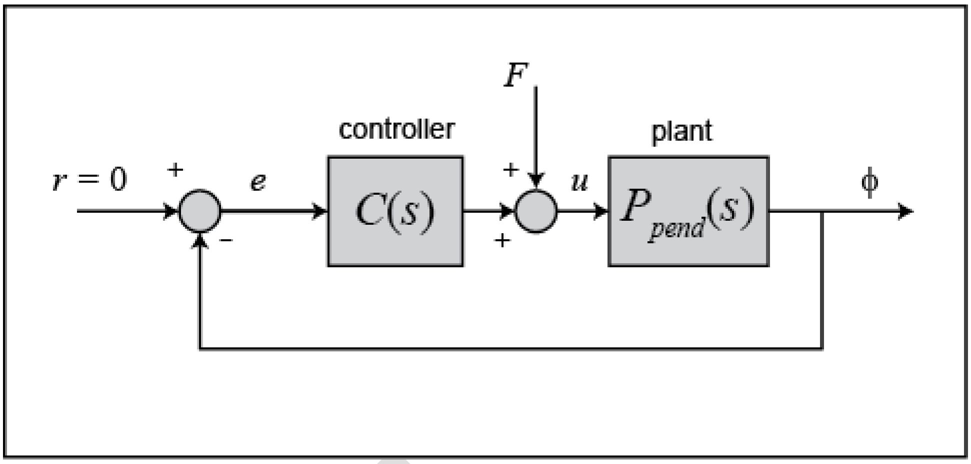
هدف کنترلر، نگه داشتن پاندول در موقعیت عمودی در هنگام اعمال ضربه 1 Nsec به ارابه می‌باشد. تحت شرایط ذکر شده، نیازهای طراحی عبارتند از:

- زمان نشست کمتر از ۵ ثانیه
- پاندول هیچگاه بیش از ۰/۰۵ رادیان از موقعیت عمودی منحرف نشود

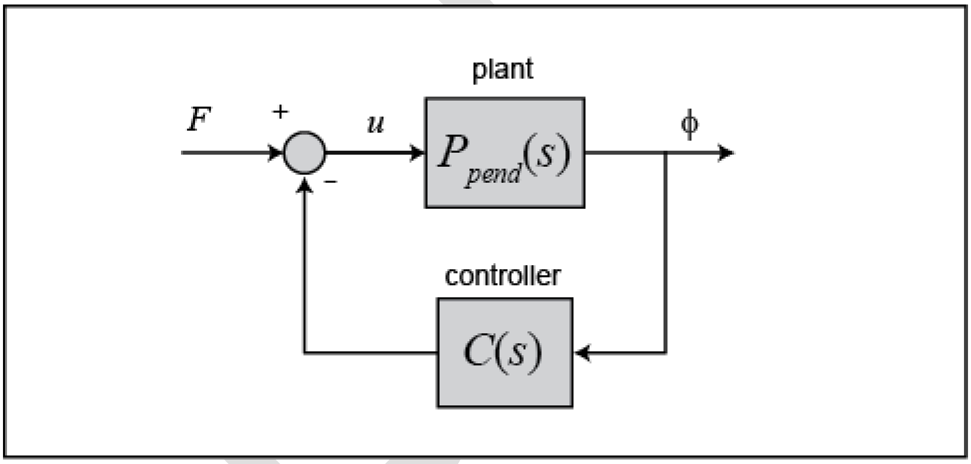
ساختار سیستم

ساختار کنترلر برای این مسئله تا حدی متفاوت با مسائل کنترل استاندارد است. به علت اینکه ما موقعیت پاندول را کنترل می‌کنیم (یعنی بعد از اعمال اغتشاش، پاندول به موقعیت اولیه خود بازگردد)، سیگنال مرجع صفر می‌باشد. اینگونه از مسائل اغلب با نام مسائل رگولاسیون[۳۷] یاد می‌شوند. نیروی خارجی وارد شده به ارابه را می‌توان به صورت یک اغتشاش ضربه در نظر گرفت. شماتیک این مسئله در شکل زیر نمایش داده شده است:

[۳۷] Regulation



با تغییر چیدمان سیستم به صورت زیر، طراحی و تحلیل ساده‌تر می‌باشد:



تابع تبدیل $T(s)$ برای سیستم حلقه بسته از ورودی نیروی F به خروجی زاویه پاندول ϕ برابر است با:

$$T(s)=\frac{\mathbf{\Phi}(s)}{F(s)}=\frac{P_{pend}(s)}{1+C(s)P_{Pend}(s)} \quad (۳)$$

پیش از شروع طراحی کنترلر PID، باید سیستم خود را در متلب وارد نماییم. یک ام‌فایل جدید ساخته و دستورات زیر را برای ساخت مدل سیستم اجرا کنید:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;

q = (M+m)*(I+m*l^2)-(m*l)^2;

s = tf('s');
```

```
P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);
```

در قدم بعد کنترلر PID را طراحی می‌نماییم.

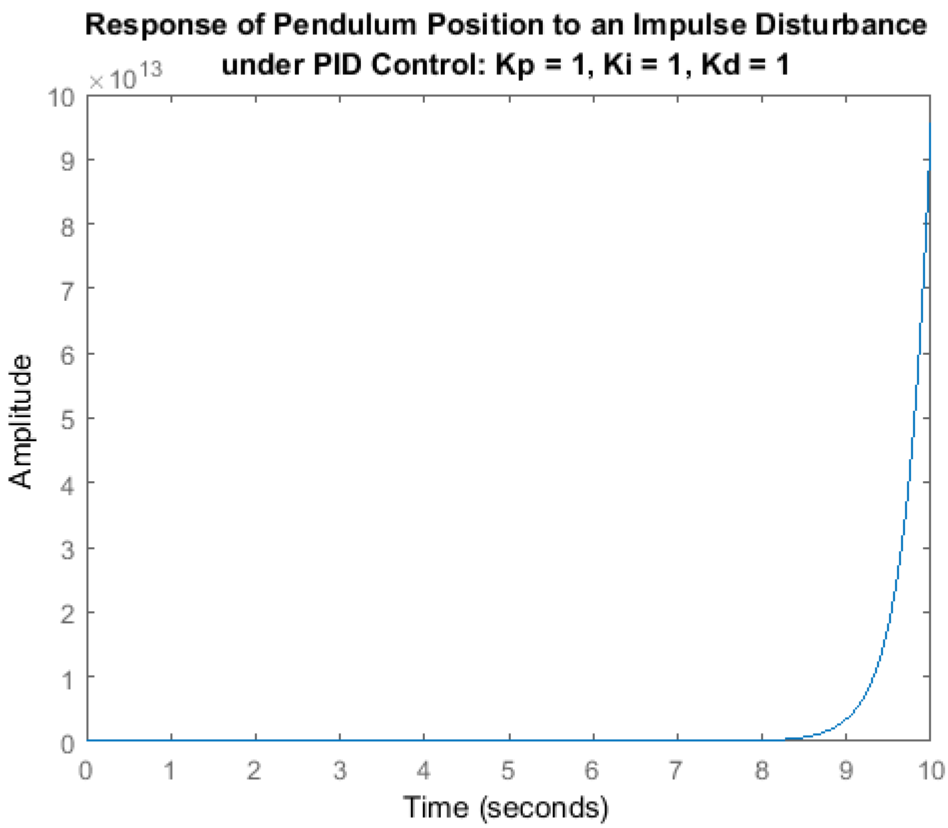
کنترل PID

تابع تبدیل حلقه بسته را می‌توان با وارد کردن دستورات زیر در ادامه‌ی ام‌فایل خود به دست آورد (چه در حالت تابع تبدیل چه در فرم فضای حالت). کنترلر خود را با دستور pid در متلب تعریف می‌نماییم. از دستور feedback برای به دست آوردن تابع تبدیل حلقه بسته $T(s)$ که در آن F نیروی اغتشاش به عنوان ورودی و ϕ زاویه پاندول از حالت عمودی به عنوان خروجی می‌باشد استفاده می‌کنیم.

```
Kp = 1;

Ki = 1;

Kd = 1;

C = pid(Kp,Ki,Kd);

T = feedback(P_pend,C);
```

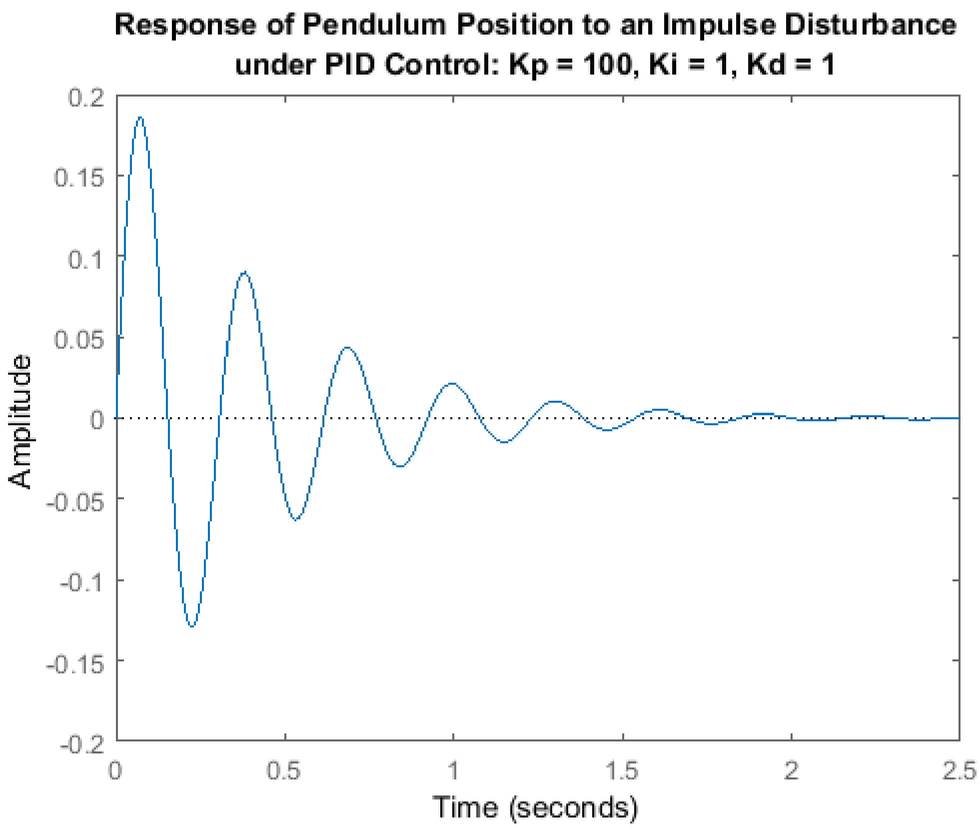
حال می‌توانیم کنترلر خود را تنظیم کنیم. ابتدا پاسخ سیستم حلقه بسته به اغتشاش ضربه را برای تنظیم ضرایب اولیه به دست می‌آوریم. کد زیر را به انتهای ام‌فایل خود اضافه کرده و آنرا اجرا کنید. در اینصورت نمودار پاسخ سیستم به شکل زیر به دست می‌آید:

```
t=0:0.01:10;

impulse(T,t)

title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 1, Ki = 1, Kd = 1'});
```



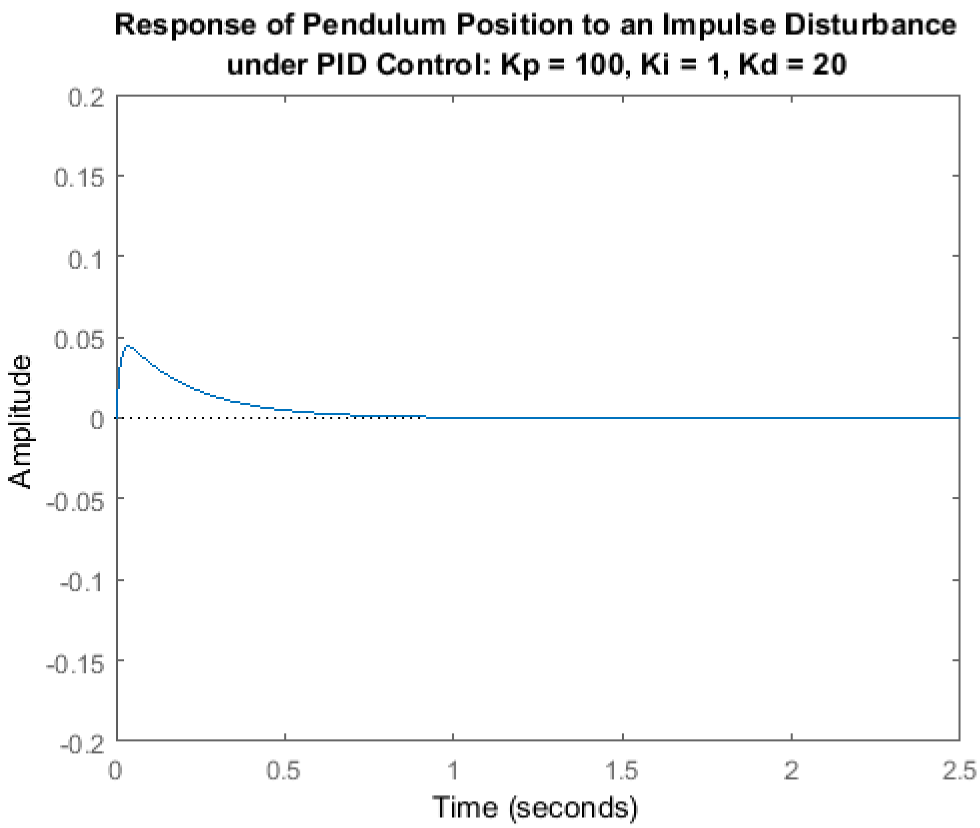
پاسخ به دست آمده ناپایدار است. با اضافه کردن ضریب تناسبی، پاسخ را اصلاح می‌نماییم. متغیر K_p را افزایش دهید تا تاثیر آنرا بر روی پاسخ مشاهده کنید. اگر امفایل را به گونه‌ای تغییر دهید که $K_p = 100$ باشد و آنرا اجرا کنید، نمودار پاسخ زیر به دست می‌آید:

```
Kp = 100;

Ki = 1;

Kd = 1;

C = pid(Kp,Ki,Kd);

T = feedback(P_pend,C);

t=0:0.01:10;

impulse(T,t)

axis([0, 2.5, -0.2, 0.2]);

title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 1'});
```



بر روی نمودار راست کلیک کرده و از منوی نمایش داده شده گزینه Characteristics را انتخاب کنید تا مشخصات مهم پاسخ را مشخص کنید. مشخصات مهم پاسخ عبارتند از زمان نشست برابر ۱/۶۴ ثانیه که کمتر از خواسته مسئله یعنی ۵ ثانیه است. چون خطای حالت ماندگار با سرعت زیادی به صفر نزدیک می‌شود، تغییری در کنترل انتگرالی نیاز نمی‌باشد. می‌توانید ضریب K_i را برابر صفر قرار دهید تا تاثیر آنرا بر روی خطای حالت ماندگار مشاهده کنید. ماکزیمم مقدار پاسخ (قله) بیشتر از مقدار مشخص شده یعنی ۰/۰۵ رادیان است. از قبل می‌دانیم که با افزایش کنترل مشتقی، مقدار فراجهش کاهش می‌یابد. بعد از چند مرتبه سعی و خطا متوجه می‌شویم که بهره مشتق $K_d = 20$ پاسخ مناسبی را به دست می‌دهد. امفایل خود را به شکل زیر تغییر دهید:

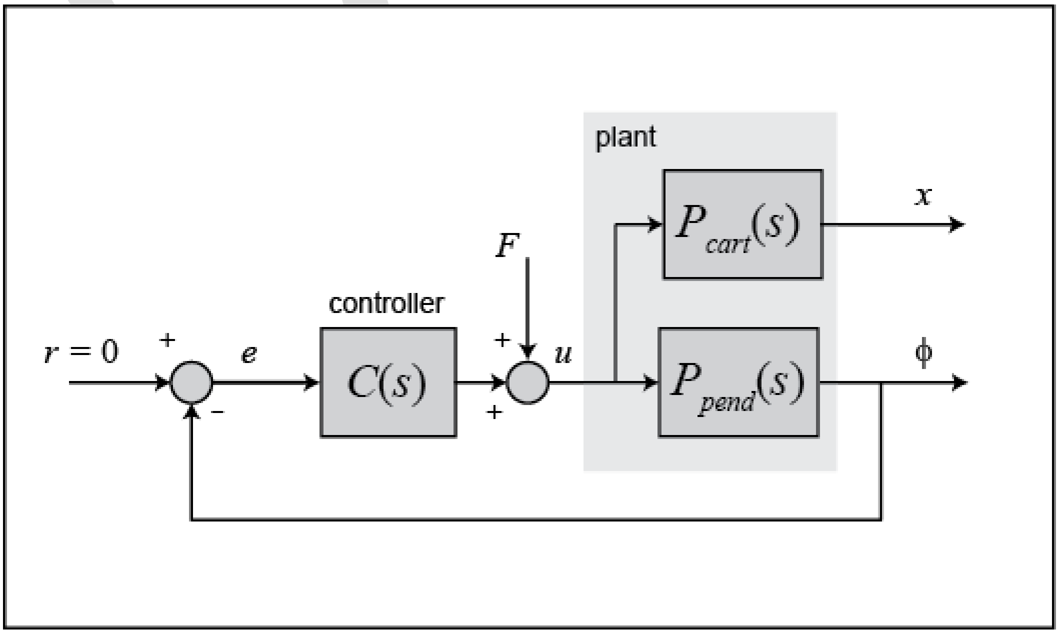
```
Kp = 100;

Ki = 1;

Kd = 20;

C = pid(Kp,Ki,Kd);

T = feedback(P_pend,C);

t=0:0.01:10;

impulse(T,t)

axis([0, 2.5, -0.2, 0.2]);

title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});
```



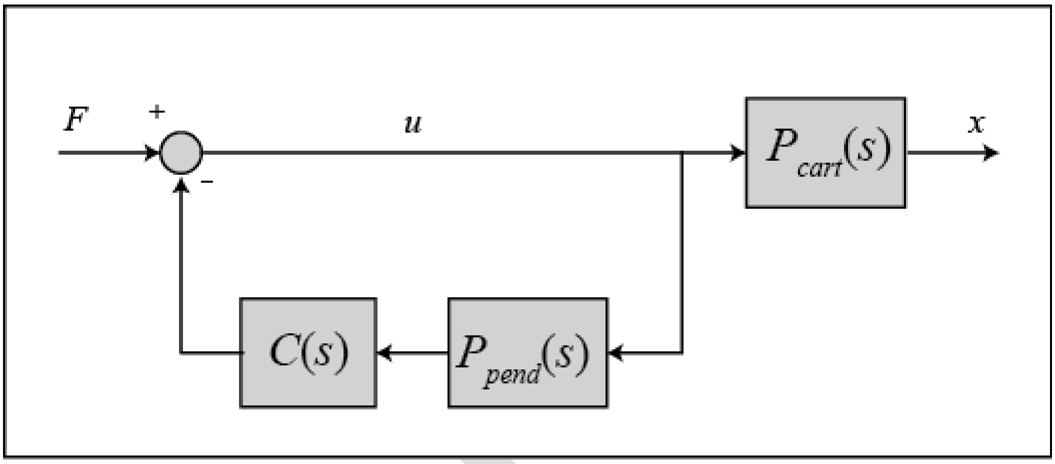
همانطور که مشاهده می‌کنید، مقدار فراجهش به مقداری کاهش یافته است که پاندول بیش از ۰/۰۵ رادیان از موقعیت عمودی حرکت نمی‌کند. چون تمامی نیازهای طراحی برآورده شده است پس نیازی به اصلاحات بیشتر نمی‌باشد.

موقعیت ارابه چگونه تغییر می کند؟

در ابتدای این بخش، دیاگرام بلوکی سیستم پاندول معکوس نمایش داده شده است. دیاگرام رسم شده کامل نمی‌باشد. بلوکی که بیانگر پاسخ موقعیت ارابه x می‌باشد در این دیاگرام آورده نشده است زیرا متغیر آن کنترل نمی‌شود. هرچند که برای ما جالب است بدانیم که موقعیت ارابه در هنگام کنترل زاویه پاندول به چه صورتی حرکت می‌کند. برای اینکار باید بلوک دیاگرام کل سیستم را به شکل زیر در نظر بگیریم:



با تغییر چیدمان داریم:



در دیاگرام بالا، $C(s)$ کنترلری است که برای حفظ موقعیت پاندول طراحی شده است. تابع تبدیل حلقه بسته‌ی $T_2(s)$ از ورودی نیروی وارد شده به ارابه به خروجی موقعیت ارابه تعریف شده است، در نتیجه داریم:

$$T_2(s) = \frac{X(s)}{F(s)} = \frac{P_{cart}(s)}{1 + P_{pend}(s)C(s)} \tag{۴}$$

با مراجعه به قسمت مدل‌سازی پاندول معکوس، تابع تبدیل $P_{cart}(s)$ عبارتست از:

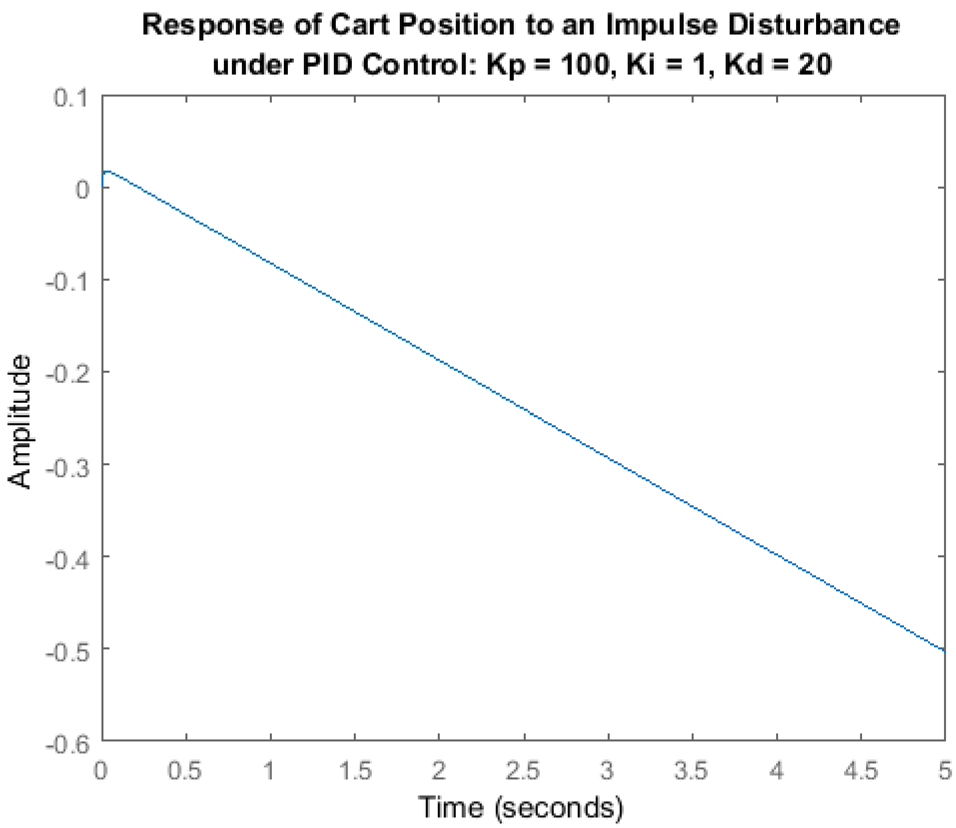
$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \quad \left[\frac{m}{N}\right] \tag{۵}$$

که:

$$q = [(M+m)(I+ml^2) - (ml)^2] \tag{۶}$$

با اضافه کردن دستورات زیر به امفایل خود، پاسخ موقعیت ارابه به همان اغتشاش ضربه به دست می‌آید:

```
P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);

T2 = feedback(1,P_pend*C)*P_cart;

t = 0:0.01:5;

impulse(T2, t);

title({'Response of Cart Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});
```



همانطور که مشاهده می‌کنید، ارابه با سرعت ثابت در جهت منفی حرکت می‌کند. در نتیجه با وجود اینکه کنترلر PID، زاویه پاندول را پایدار می‌سازد اما برای یک سیستم واقعی قابل پیاده سازی نمی‌باشد.

بخش چهارم: طراحی کنترلر با مکان هندسی ریشه‌ها

فهرست مطالب بخش

دستورهای کلیدی متلب در این بخش:

`tf , rlocus , pole , zero , zpk , feedback , impulse`

در این صفحه به طراحی یک کنترل برای سیستم پاندول معکوس با روش مکان هندسی ریشه‌ها می‌پردازیم. در روند طراحی، سیستم را تک ورودی تک خروجی در نظر گرفته که با تابع تبدیل زیر تعریف می‌شود. همانطور که گفته شد، هدف ما کنترل زاویه پاندول بدون در نظر گرفتن موقعیت ارابه می‌باشد.

$$P_{pend}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \left[\frac{rad}{N}\right] \qquad (۱)$$

که:

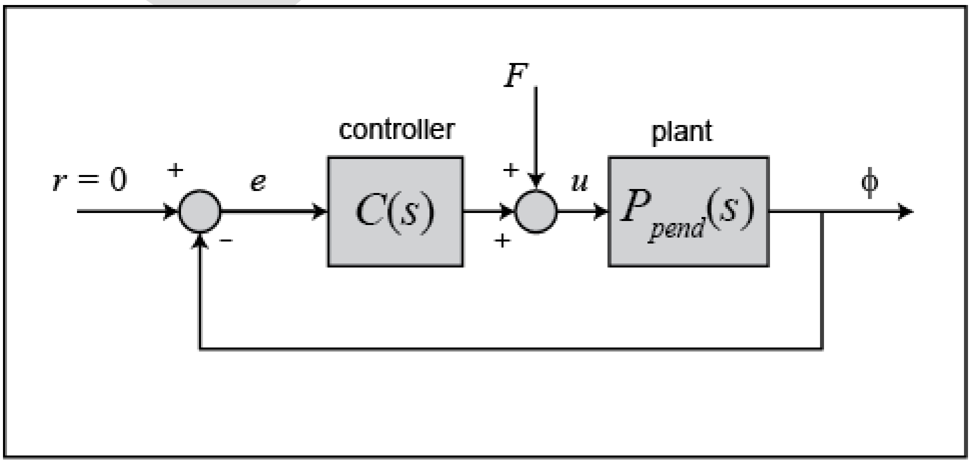
$$q = [(M+m)(I+ml^2) - (ml)^2] \qquad (۲)$$

کنترلر باید بتواند تحت ضربه‌ی وارد شده 1 Nsec به ارابه، موقعیت پاندول را عمودی نگه دارد. نیازهای طراحی عبارتند از:

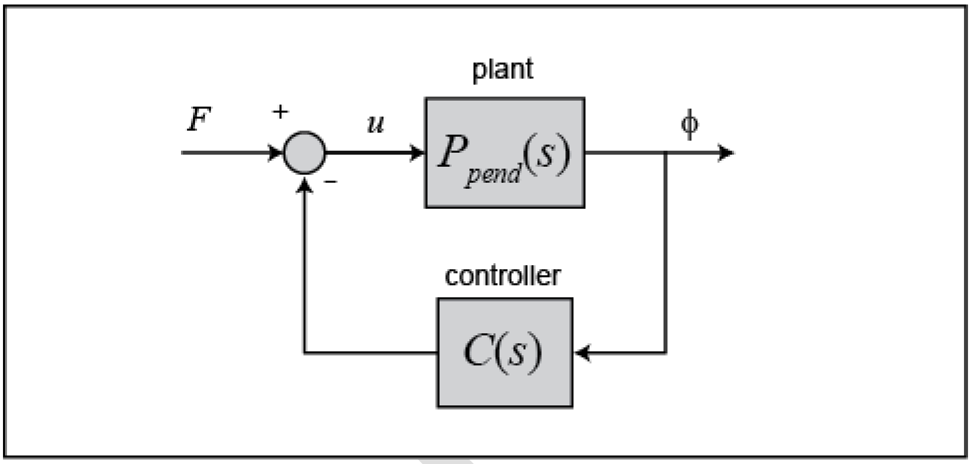
- زمان نشست کمتر از ۵ ثانیه
- پاندول نباید هیچگاه بیش از ۰/۰۵ رادیان از موقعیت عمودی منحرف شود

ساختار سیستم

ساختار کنترلر برای این مسئله تا حدی متفاوت با مسائل کنترل استاندارد است. به علت اینکه ما موقعیت پاندول را کنترل می‌کنیم (یعنی بعد از اعمال اغتشاش، پاندول به موقعیت اولیه خود بازگردد)، سیگنال مرجع صفر می‌باشد. اینگونه از مسائل اغلب به نام مسائل رگولاسیون یاد می‌شوند. نیروی خارجی وارد شده به ارابه را می‌توان به صورت یک اغتشاش ضربه در نظر گرفت. شماتیک این مسئله در شکل زیر نمایش داده شده است:



با تغییر چیدمان سیستم به صورت زیر، طراحی و تحلیل ساده‌تر می‌باشد:



تابع تبدیل $T(s)$ برای سیستم حلقه بسته از ورودی نیروی F به خروجی زاویه‌ی پاندول ϕ برابر است با:

$$T(s) = \frac{\Phi(s)}{F(s)} = \frac{P_{pend}(s)}{1 + C(s)P_{Pend}(s)} \tag{۳}$$

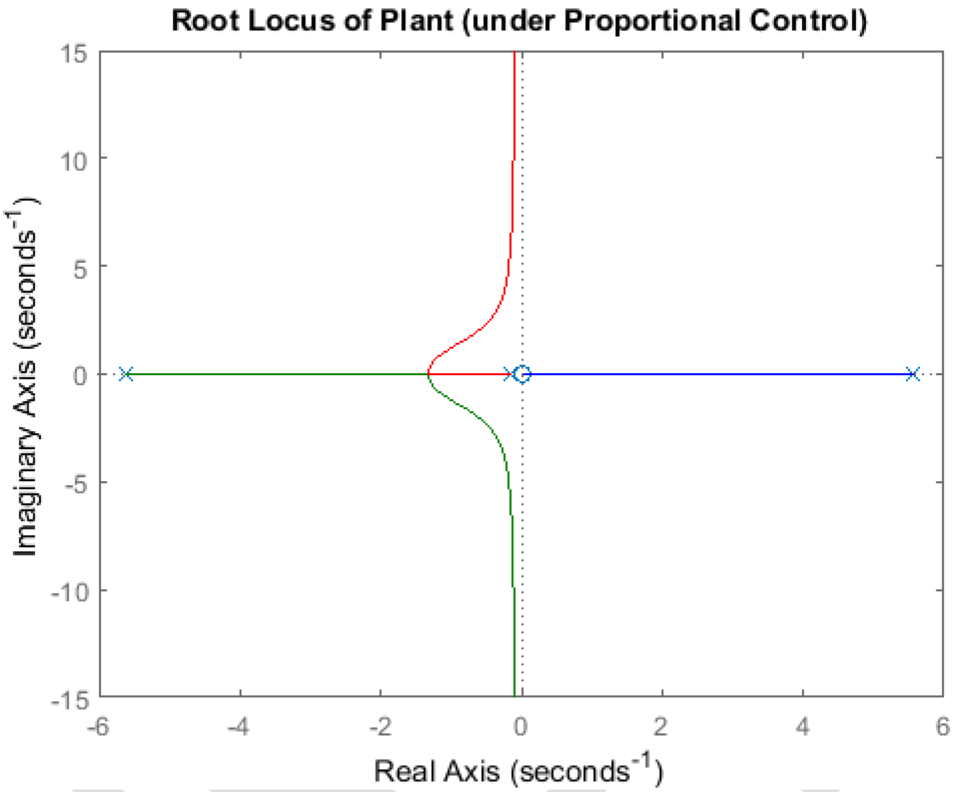
پیش از شروع طراحی کنترلر ، باید سیستم خود را در متلب وارد نماییم. یک ام‌فایل جدید ساخته و دستورات زیر را برای ساخت مدل سیستم اجرا کنید:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;

q = (M+m)*(I+m*l^2)-(m*l)^2;

s = tf('s');
P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);
```

طراحی مکان هندسی ریشه‌ها

اکنون می‌توانیم برای سیستم بالا، کنترلری را با روش مکان هندسی ریشه‌ها طراحی نماییم. برای ایجاد نمودار مکان هندسی ریشه‌ها در متلب می‌توان از دستور rlocus استفاده کرد. با اضافه کردن دستورات زیر در ام‌فایل خود و اجرای آن، نمودار مکان هندسی ریشه‌ها به دست می‌آید. این نمودار تمامی موقعیت‌های ممکن برای قطب‌های حلقه بسته را با تغییر یک بهره کنترل تناسبی K از صفر تا بینهایت را نشان می‌دهد. مکان هندسی ریشه‌ها در صورتی که بهره K در مسیر مستقیم یا فیدبک باشد یکسان می‌باشد.

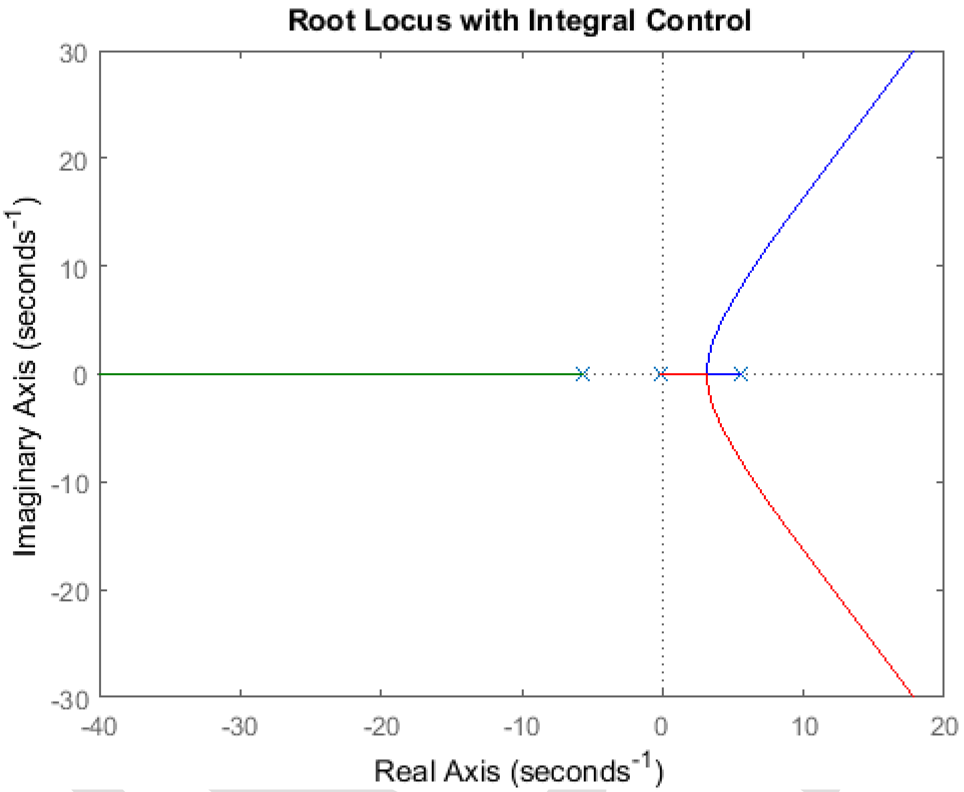
```
rlocus(P_pend)

title('Root Locus of Plant (under Proportional Control)')
```



همانطور که مشاهده می‌کنید، یکی از شاخه‌های مکان هندسی ریشه‌ها به طور کامل در سمت راست محور موهومی قرار دارد. این به معنی آنست که تحت هیچ بهره‌ی K، قطب سیستم حلقه بسته در نیم صفحه راست قرار نخواهد گرفت و سیستم ناپایدار خواهد بود.

برای حل این مشکل، باید یک قطب به مبدا (انتگرال‌گیر) اضافه کنیم تا با صفر سیستم که در مبدا قرار دارد خنثی شود. با اینکار دو قطب حلقه بسته در سمت راست محور موهومی خواهیم داشت. در طراحی بعدی خود می‌توانیم کنترلر را به گونه‌ای تغییر دهیم تا این قطب‌ها در نیم صفحه‌ی چپ قرار بگیرند و سیستم حلقه بسته پایدار گردد. امفایل خود را به شکل زیر اصلاح کرده و آنرا دوباره اجرا کنید تا نمودار مکان هندسی ریشه‌ها به دست آید:

```
C = 1/s;

rlocus(C*P_pend)

title('Root Locus with Integral Control')
```



همچنین موقعیت صفرها و قطب‌های حلقه باز را نیز بررسی می‌کنیم تا بفهمیم چگونه شاخه‌های مکان هندسی را به سمت چپ متمایل نماییم. کدهای زیر را در پنجره دستور متلب وارد کنید تا خروجی زیر به دست آید:

```
zeros = zero(C*P_pend)

poles = pole(C*P_pend)
```

```
zeros =

     0

poles =

         0

    5.5651

   -5.6041

   -0.1428
```

همانطور که مشاهده می‌کنید، سیستم ما دارای چهار قطب و یک صفر می‌باشد. این بدین معناست که مکان هندسی ریشه‌ها دارای سه مجانب می‌باشد: یکی در راستای محور حقیقی و در جهت منفی و دو مجانب دیگر با زاویه ۱۲۰ درجه از مجانب اول.

این حالت از شاخه‌های مکان هندسی ریشه‌ها رضایت‌بخش نمی‌باشد زیرا همچنان شاخه‌های آن کاملا در سمت راست محور موهومی قرار دارند. به طور کلی می‌توانیم با اضافه کردن صفر به سیستم، شاخه‌های مکان هندسی را به سمت

چپ متمایل کنیم. با اضافه شدن یک صفر به کنترلر، تعداد مجانب‌ها از سه به دو کاهش می‌یابد. این دو مجانب موازی با محور موهومی بوده و در محل برخورد آنها با محور حقیقی توسط رابطه زیر به دست می‌آید:

$$s = \frac{\Sigma poles - \Sigma zeros}{\#poles - \#zeros} \tag{۴}$$

بنابراین برای سیستم ما با فرض یک صفر مینیمم فاز (منفی)، محل تقاطع عبارتست از:

$$s = \frac{(-5.6041 + 5.5651 - 0.1428 + 0) - (0 - z)}{4 - 2} = \frac{-0.1818 + z}{2} \tag{۵}$$

با نتیجه به دست آمده، با استفاده از صفر بسیار کوچک، نهایتا می‌توانیم مجانب‌ها را تقریبا به 0.1- برسانیم. به یاد داریم که زمان نشست ۲٪ را از رابطه‌ی زیر به طور تقریبی به دست می‌آوریم:

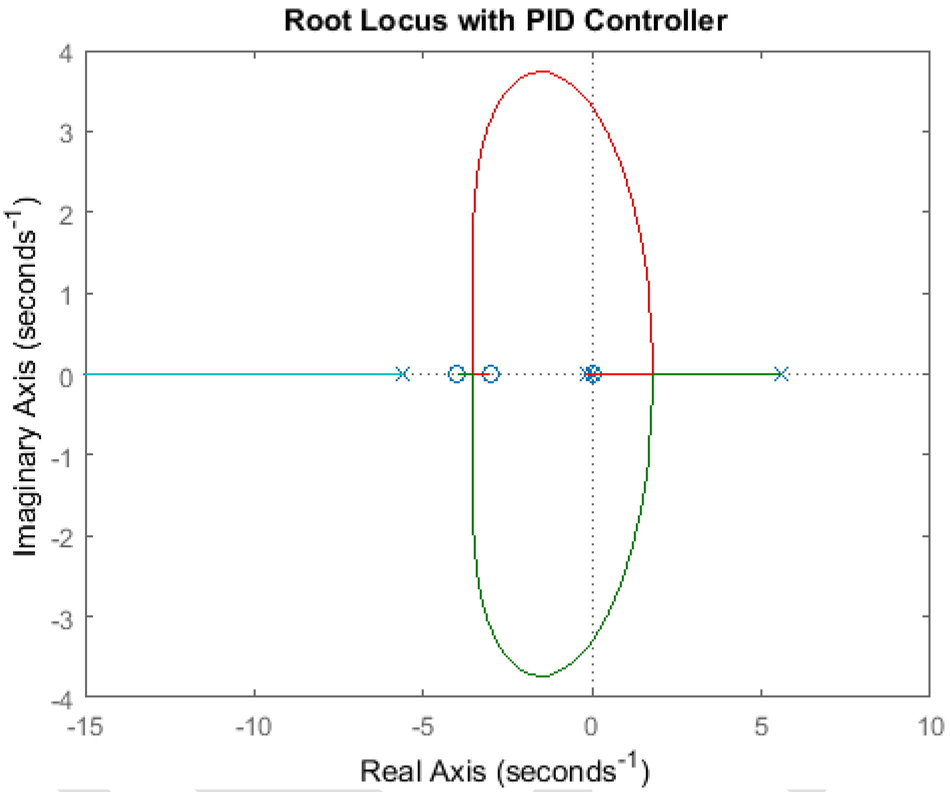
$$T_s = \frac{4}{\sigma} \tag{۶}$$

در نتیجه قطب‌های غالب حلقه بسته با قسمت حقیقی که به 0.1- میل می‌کنند برای ارضای شرط زمان نشست ۵٪ کافی نمی‌باشند.

کنترل PID

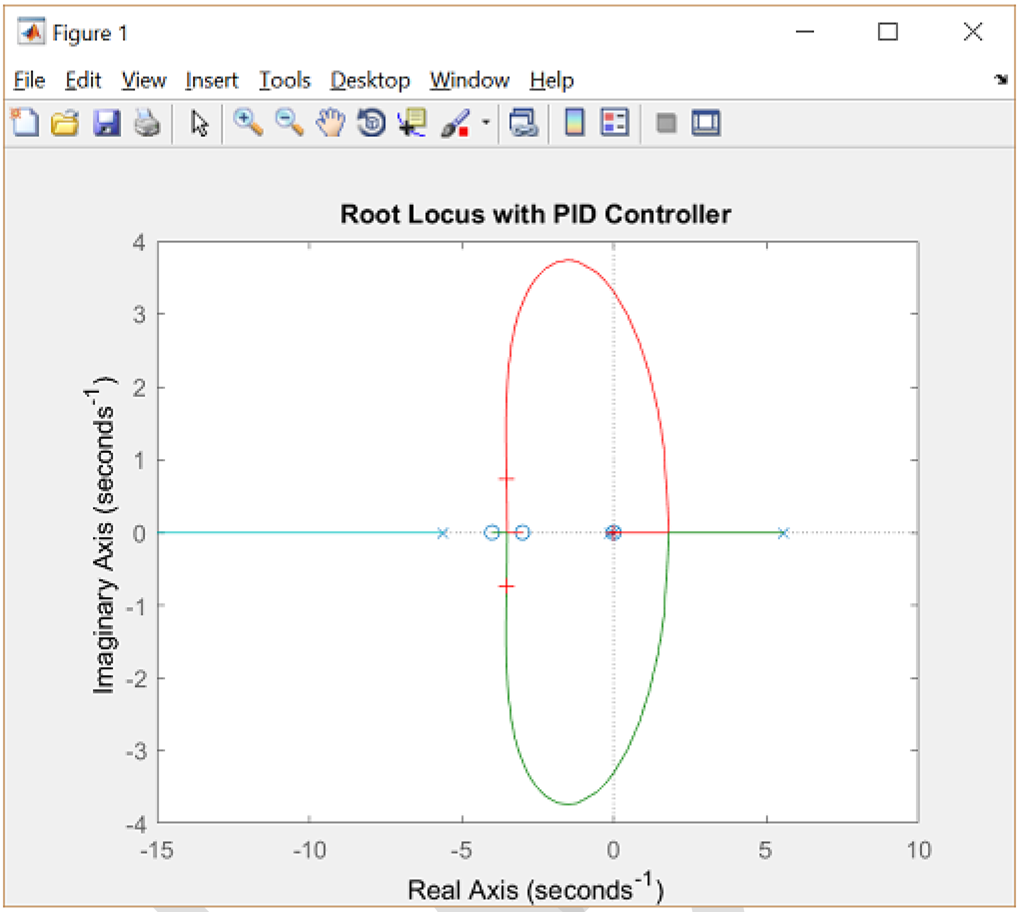
در قسمت قبل نشان دادیم که با اضافه کردن صفر به کنترلر انتگرالی خود، می‌توانیم شاخه‌های مکان هندسی ریشه‌ها را به سمت چپ صفحه مختلط متمایل کنیم اما قادر به حرکت دادن شاخه‌های غالب به اندازه کافی به سمت چپ نمی‌باشیم. یکی از راه‌های ممکن، اضافه کردن یک صفر دیگر می‌باشد. اگر دو صفر در قسمت منفی محور موهومی و بین دو قطب سیستم قرار دهیم، آنگاه دو شاخه مکان هندسی ریشه‌ها به سمت چپ متمایل شده و به این دو صفر می‌رسند. اینکار را برای کنترلری با یک انتگرال‌گیر و صفرهای موجود در 3- و 4- انجام می‌دهیم. لازم به ذکر است که این کنترلر در واقع همان کنترلر PID است. برای ساخت این کنترلر در متلب می‌توانیم از دستور zpk استفاده کنیم که این دستور با استفاده از صفر و قطب و بهره داده شده، یک مدل ایجاد می‌نماید. امفایل خود را به شکل زیر تغییر دهید و آنرا دوباره اجرا کنید تا نمودار مکان هندسی ریشه‌ها به دست آید:

```
z = [-3 -4];

p = 0;

k = 1;

C = zpk(z,p,k);

rlocus(C*P_pend)

title('Root Locus with PID Controller')
```



با بررسی متوجه می‌شویم که آیا نیازهای طراحی برآورده شده‌اند یا خیر. خصوصا از آنجایی که زمان نشست مطلوب سیستم باید کمتر از ۵ ثانیه باشد، قسمت‌های حقیقی قطب‌های غالب حلقه بسته باید کمتر از 0.8-=4/5- باشند. به عبارت دیگر قطب‌های غالب حلقه بسته باید در سمت چپ محور موهومی و برابر s=-0.8 باشند. با بررسی نمودار بالا متوجه می‌شویم این حالت امکان‌پذیر می‌باشد. همچنین از آنجایی که پاندول نباید بیش از ۰/۰۵ رادیان از موقعیت عمودی خود حرکت نماید، باید از میرایی کافی سیستم نیز اطمینان حاصل نماییم. قرار دادن قطب‌های غالب حلقه بسته در نزدیکی محور حقیقی سبب افزایش میرایی سیستم می‌شود (β کوچک).

برای به دست آوردن بهره متناسب با نقطه‌ی انتخاب شده بر روی مکان هندسی ریشه‌ها از دستور rlocfind استفاده می‌نماییم. دستور [k,poles] = rlocfind(C*P_pend) را در پنجره دستور متلب وارد کنید. سپس از نمودار انتخاب شده نقطه‌ای را در سمت چپ حلقه و نزدیک به محور حقیقی، مشابه نقطه‌ای که در شکل زیر با علامت + مشخص شده است، انتخاب کنید. با انتخاب این قطب‌ها، از سرعت نشست سیستم اطمینان حاصل کرده و همچنین امیدواریم تا میرایی سیستم به اندازه کافی باشد.



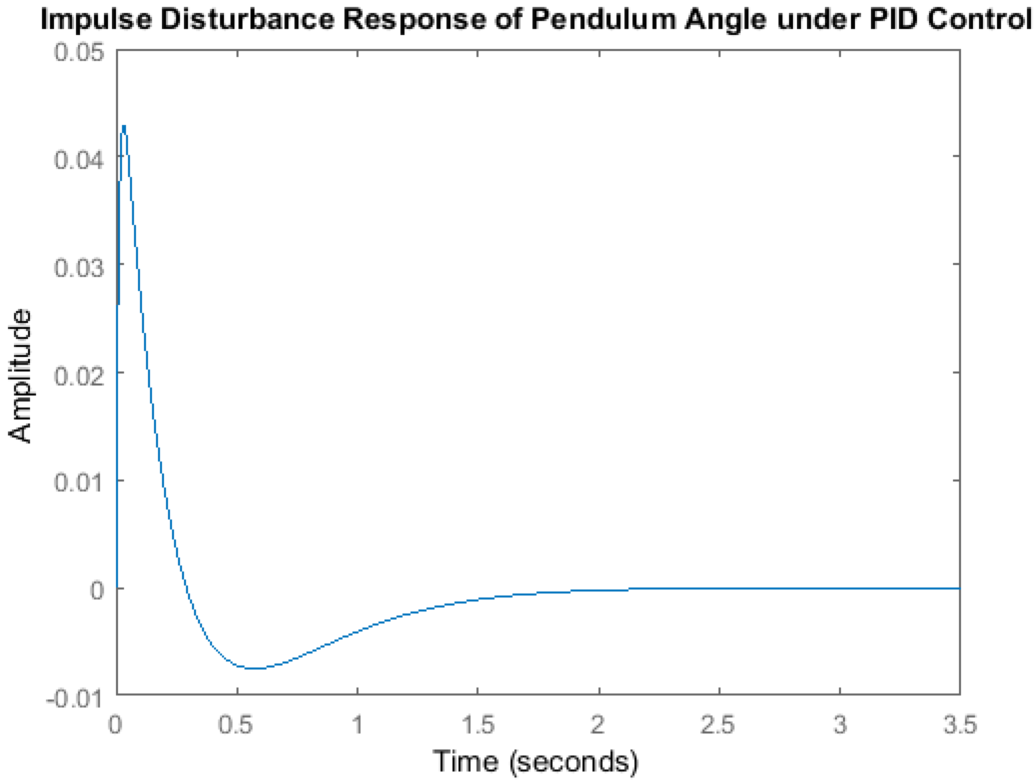
بعد از انتخاب نقطه، خروجی‌های زیر در متلب به نمایش در می‌آید:

```
Select a point in the graphics window


selected_point =

-3.5367 + 0.7081i


k =

20.2396


poles =

0
-85.1333
-3.5232 + 0.7086i
-3.5232 - 0.7086i
```

ممکن است مقادیر نشان داده شده در متلب، با مقادیری که در اینجا آورده شده است یکسان نباشد اما باید حداقل در یک مقیاس باشند.

حال می‌توانیم پاسخ ضربه‌ی سیستم حلقه بسته را با بهره‌ی K تقریبا برابر ۲۰ بررسی کرده تا ببینیم آیا نیازهای طراحی برآورده شده‌اند یا خیر. دستورات زیر را به ام‌فایل خود اضافه کرده و آنرا دوباره اجرا کنید تا پاسخ ضربه‌ی سیستم مانند زیر ایجاد شود:

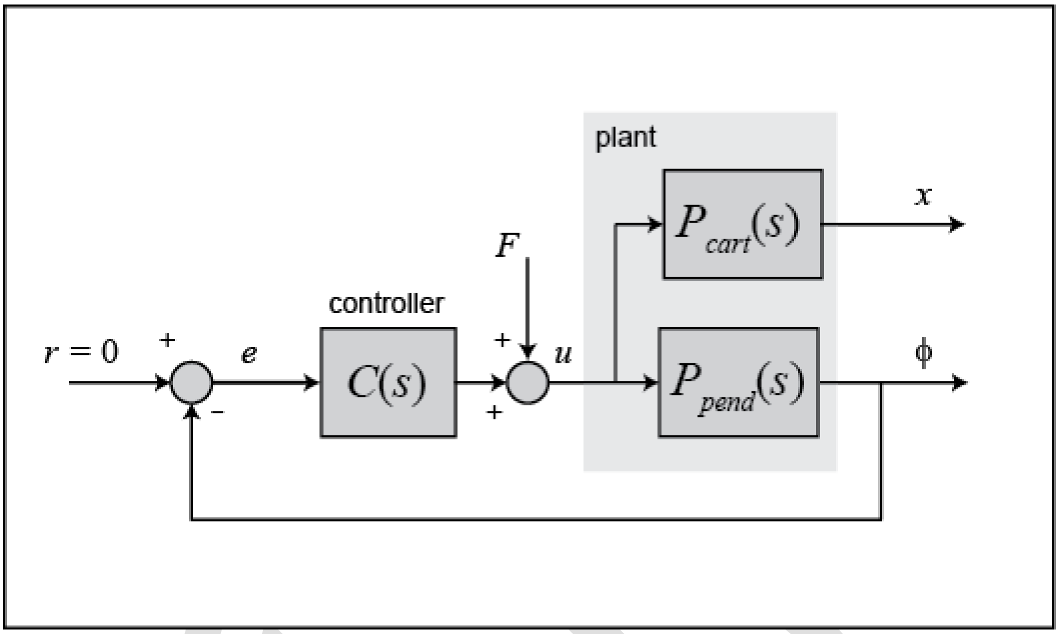
```
K = 20;

T = feedback(P_pend,K*C);

impulse(T)

title('Impulse Disturbance Response of Pendulum Angle under PID Control');
```



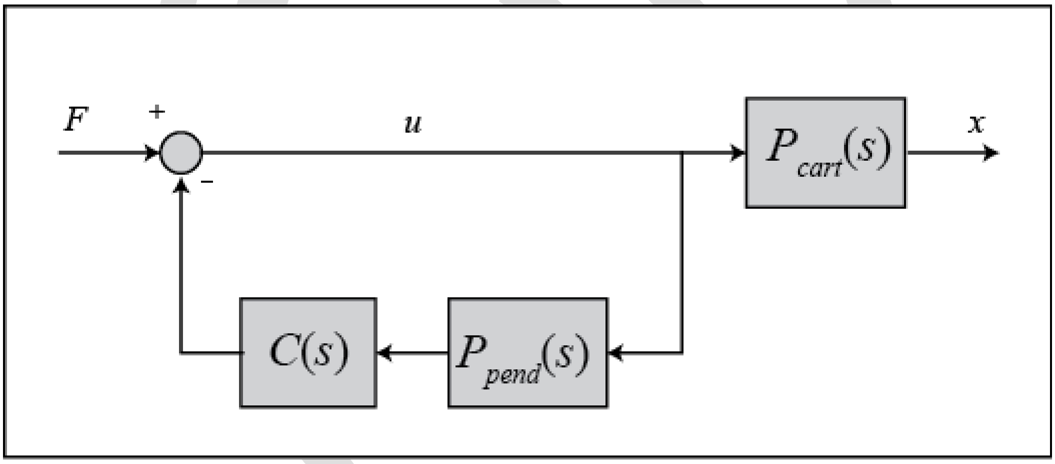
با بررسی نمودار بالا مشخص می‌شود که تمامی نیازهای طراحی برآورده شده‌اند.

موقعیت ارابه چگونه تغییر می کند؟

در ابتدای این فصل، دیاگرام بلوکی سیستم پاندول معکوس نمایش داده شده است. دیاگرام رسم شده کامل نمی‌باشد. بلوکی که بیانگر پاسخ موقعیت ارابه x می‌باشد در این دیاگرام آورده نشده است زیرا متغیر آن کنترل نمی‌شود. هرچند که برای ما جالب است بدانیم که موقعیت ارابه در هنگام کنترل زاویه‌ی پاندول به چه صورتی حرکت می کند. برای اینکار باید بلوک دیاگرام کل سیستم را به شکل زیر در نظر بگیریم:



با تغییر چیدمان داریم:



در دیاگرام بالا، $C(s)$ کنترلری است که برای حفظ موقعیت پاندول طراحی شده است. تابع تبدیل حلقه بسته‌ی $T_2(s)$ از ورودی نیروی وارد شده به ارابه به خروجی موقعیت ارابه تعریف شده است، در نتیجه داریم:

$$T_2(s) = \frac{X(s)}{F(s)} = \frac{P_{cart}(s)}{1 + P_{pend}(s)C(s)} \tag{۴}$$

با مراجعه به قسمت مدل‌سازی پاندول معکوس، تابع تبدیل $P_{cart}(s)$ عبارتست از:

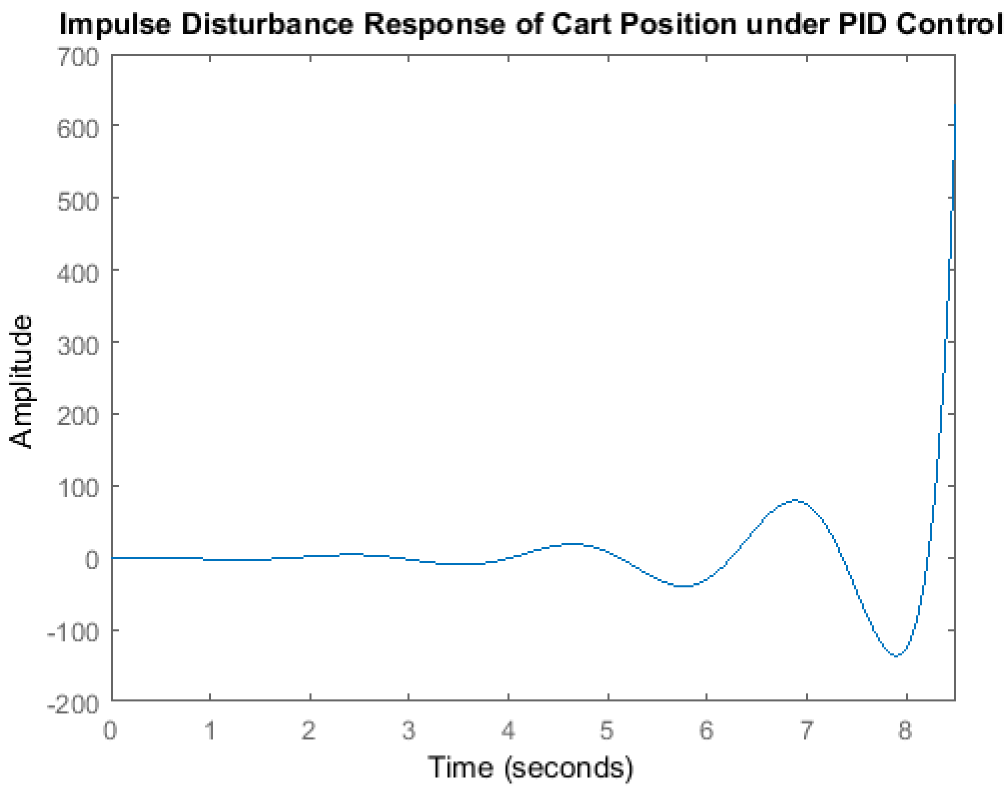
$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \quad \left[\frac{m}{N}\right] \tag{۵}$$

که:

$$q = [(M+m)(I+ml^2) - (ml)^2] \tag{۶}$$

با اضافه کردن دستورات زیر (با فرض معین بودن $P_{pend}(s)$ و $C(s)$ در متلب) پاسخ موقعیت ارابه به اغتشاش ضربه به دست می‌آید:

```
P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);

T2 = feedback(1,P_pend*C)*P_cart;

t = 0:0.01:8.5;

impulse(T2, t);

title('Impulse Disturbance Response of Cart Position under PID Control');
```



همانطور که مشاهده می‌کنید، موقعیت ارابه در اثر وارد شدن اغتشاش ضربه، ناپایدار می‌گردد. بنابراین اگرچه کنترلر PID، زاویه پاندول را پایدار می‌کند اما این طراحی در سیستم فیزیکی قابل پیاده سازی نمی‌باشد.

بخش پنجم: طراحی کنترلر در حوزه‌ی فرکانسی

فهرست مطالب بخش

- ساختار سیستم
- پاسخ حلقه بسته بدون جبران‌سازی
- پاسخ حلقه بسته با جبران‌سازی
- موقعیت ارابه چگونه تغییر می‌کند؟

دستورهای کلیدی متلب در این بخش:

```
tf , zpkdata , controlSystemDesigner , feedback , impulse
```

در این فصل به طراحی کنترلر برای سیستم پاندول معکوس با استفاده از روش‌های طراحی پاسخ فرکانسی می‌پردازیم. در روند طراحی سیستم را تک ورودی تک خروجی با تابع تبدیل زیر در نظر می‌گیریم. همانطور که گفته شد، هدف ما کنترل زاویه‌ی پاندول بدون در نظر گرفتن موقعیت ارابه می‌باشد.

$$P_{pend}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \left[\frac{rad}{N}\right] \qquad (۱)$$

که:

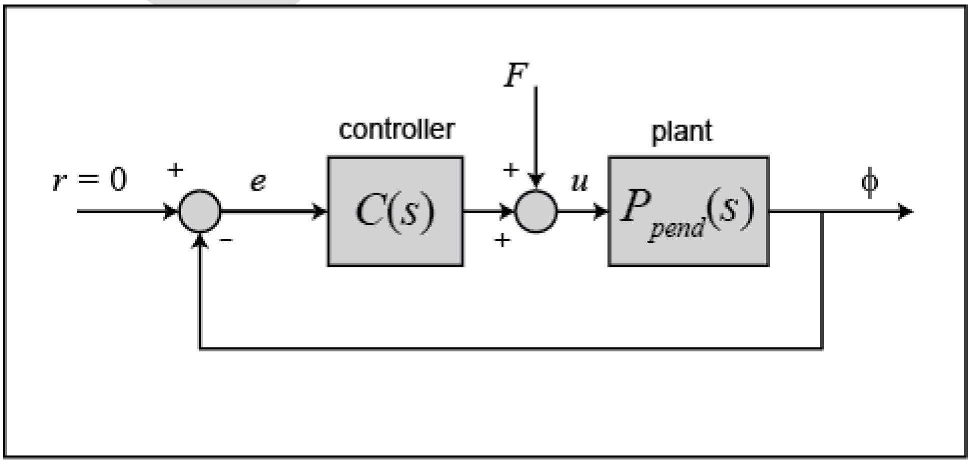
$$q = [(M+m)(I+ml^2) - (ml)^2] \qquad (۲)$$

کنترلر باید بتواند تحت ضربه‌ی وارد شده 1 Nsec به ارابه، موقعیت پاندول را عمودی نگه دارد. نیازهای طراحی عبارتند از:

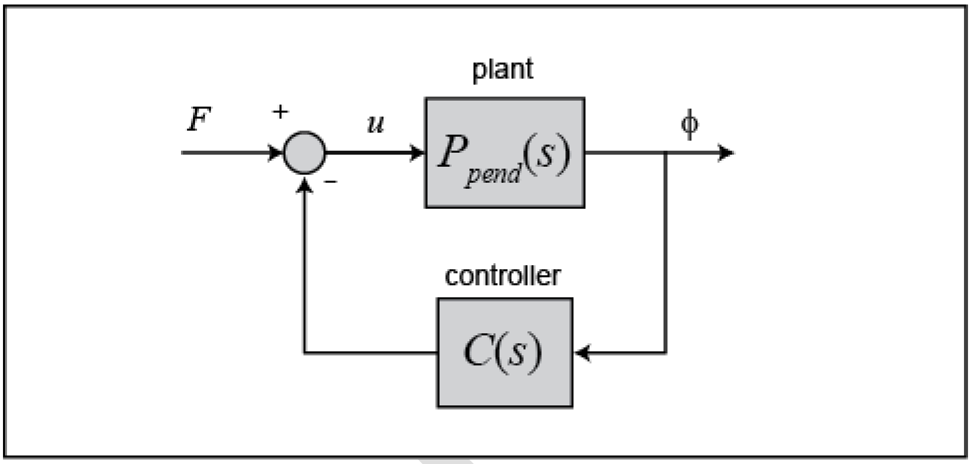
- زمان نشست کمتر از ۵ ثانیه
- پاندول نباید هیچگاه بیش از ۰/۰۵ رادیان از موقعیت عمودی منحرف شود

ساختار سیستم

ساختار کنترلر برای این مسئله تا حدی متفاوت با مسائل کنترل استاندارد است. به علت اینکه ما موقعیت پاندول را کنترل می‌کنیم (یعنی بعد از اعمال اغتشاش، پاندول به موقعیت اولیه خود بازگردد)، سیگنال مرجع صفر می‌باشد. اینگونه از مسائل اغلب به نام مسائل رگولاسیون یاد می‌شوند. نیروی خارجی وارد شده به ارابه را می‌توان به صورت یک اغتشاش ضربه در نظر گرفت. شماتیک این مسئله در شکل زیر نمایش داده شده است:



با تغییر چیدمان سیستم به صورت زیر، طراحی و تحلیل ساده‌تر می‌باشد:



تابع تبدیل $T(s)$ برای سیستم حلقه بسته از ورودی نیروی F به خروجی زاویه‌ی پاندول ϕ برابر است با:

$$T(s) = \frac{\Phi(s)}{F(s)} = \frac{P_{pend}(s)}{1 + C(s)P_{pend}(s)} \quad (۳)$$

پیش از شروع طراحی کنترلر ، باید سیستم خود را در متلب وارد نماییم. یک ام‌فایل جدید ساخته و دستورات زیر را برای ساخت مدل سیستم اجرا کنید:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;

q = (M+m)*(I+m*l^2)-(m*l)^2;

s = tf('s');
P_pend = (m*l*s/q)/(s^3 + (b*(I + m*l^2))*s^2/q - ((M + m)*m*g*l)*s/q - b*m*g*l/q);
```

همانطور که ذکر شد، سیستم بدون کنترل، ناپایدار است. برای نشان دادن این موضوع، می‌توانیم از دستور zpkdata استفاده کرد. در اینجا دستور zpkdata به ما صفرها و قطب‌های تابع تبدیل را می‌دهد. استفاده از پارامتر 'v' در این دستور، باعث بازگرداندن صفرها و قطب‌ها به فرم برداری می‌شود که تنها برای مدل‌های SISO قابل استفاده است. وارد کردن دستورات زیر در متلب خروجی‌های زیر را به ما نشان می‌دهد:

```
[zeros poles] = zpkdata(P_pend,'v')
```

```
zeros =

     0

poles =
```

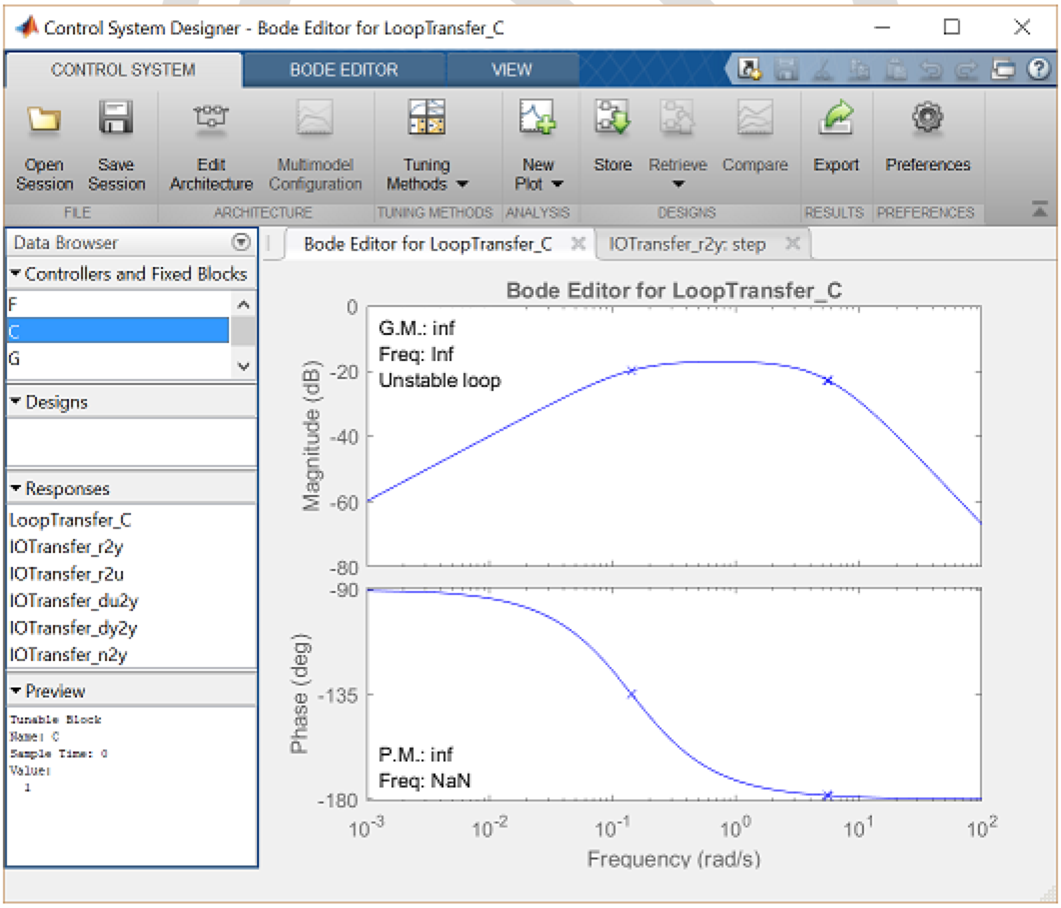
```
 5.5651

-5.6041

-0.1428
```

پاسخ حلقه بسته بدون جبران‌سازی

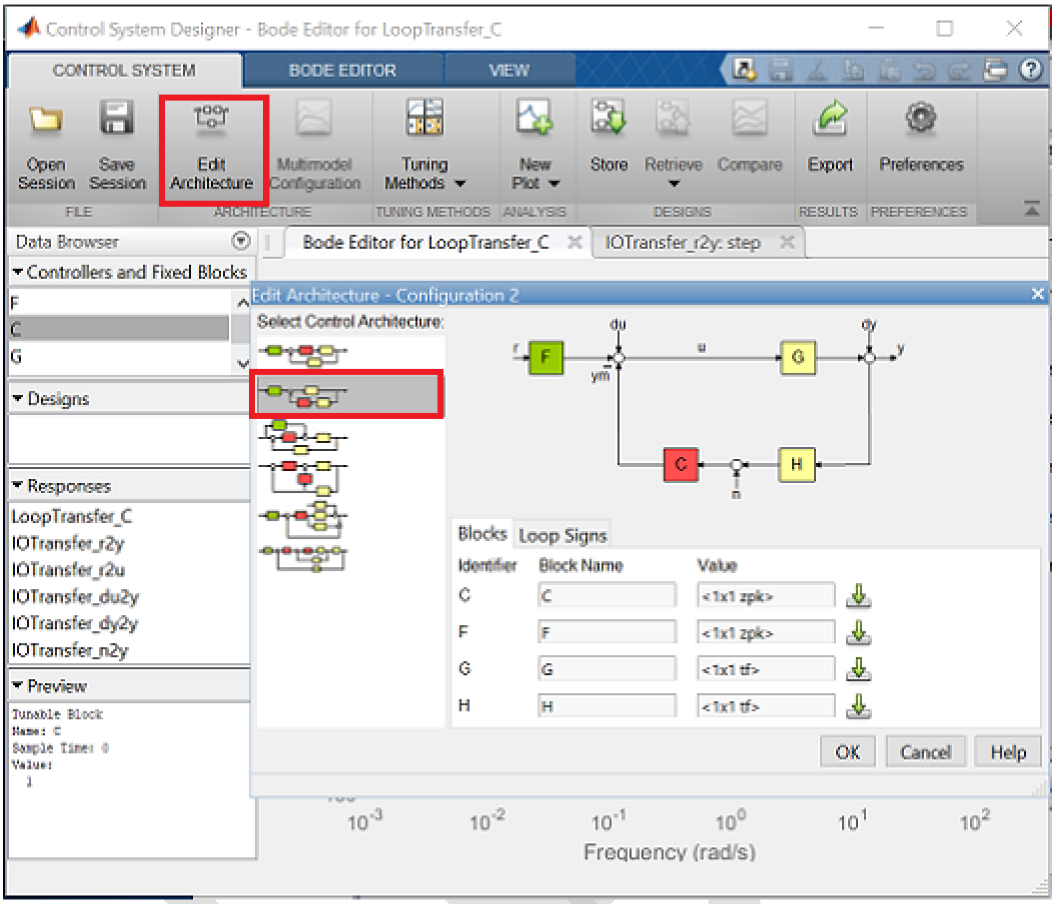
پیش از شروع طراحی کنترلر، پاسخ سیستم حلقه بسته را بدون جبران‌سازی بررسی می‌نماییم. در این مثال از Control System Designer برای بررسی نمودارهای تحلیل مختلف به جای استفاده از دستورهای جداگانه bode، nyquist و impulse استفاده می‌نماییم. Control System Designer یک ابزار تعاملی با رابط کاربری گرافیکی (GUI) بوده که از طریق دستور controlSystemDesigner به شکل زیر اجرا می‌شود. همچنین می‌توان از برگه‌ی APPS در نوار ابزار متلب و کلیک بر روی آیکون Control System Designer and Analysis این ابزار را اجرا نمود:

```
controlSystemDesigner('bode',P_pend)
```

پارامتر اضافی 'bode' در این دستور سبب باز شدن نمودار بودی و پاسخ پله سیستم حلقه بسته سیستم $P_{pend}(s)$ در هنگام باز شدن پنجره Control System Designer می‌شود.



سپس می‌توان معماری سیستم را به گونه‌ای تغییر داد که کنترلر $C(s)$ در مسیر فیدبک سیستم قرار گرفته باشد. برای تغییر معماری سیستم از پنجره Control System Designer، گزینه Edit Architecture را انتخاب کنید. سپس در پنجره‌ی باز شده، معماری دوم سیستم را که در شکل زیر نشان داده شده است انتخاب کنید:

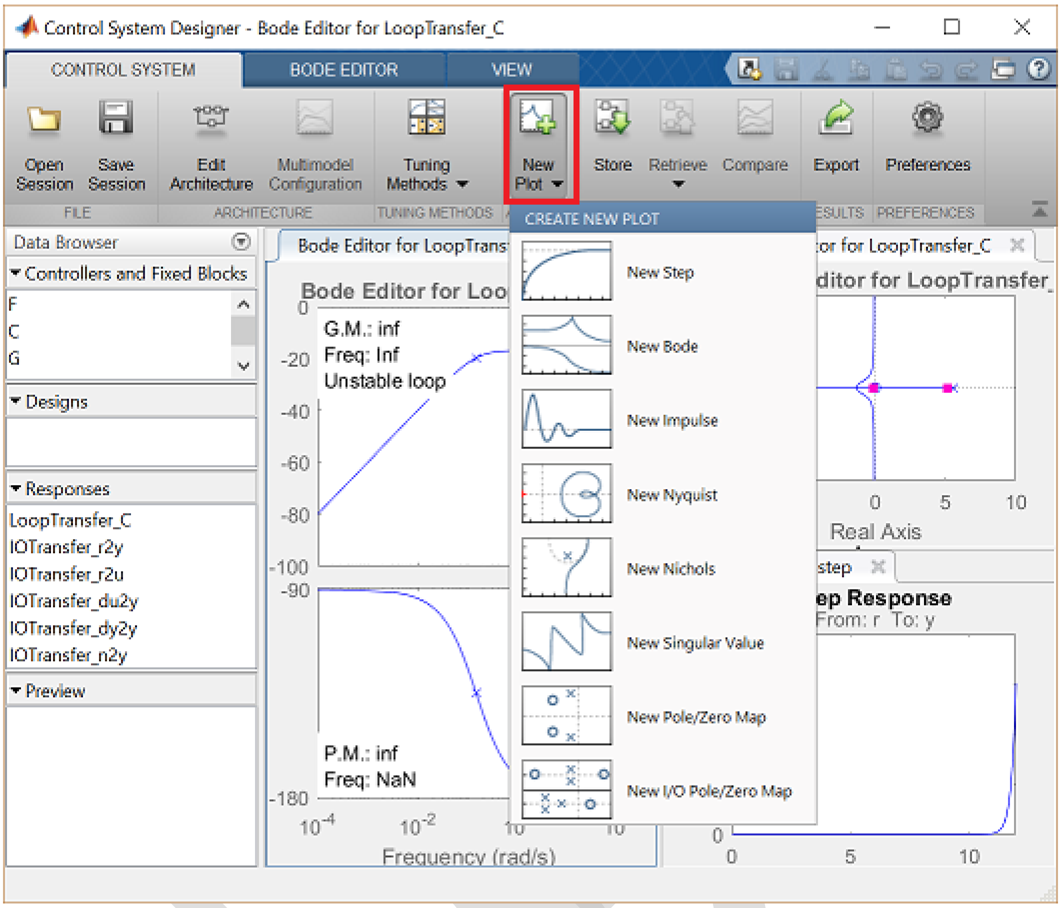


در قدم بعد می‌توانیم نمودارهای تحلیلی سیستم را بررسی نماییم. یادآوری می‌کنیم که می‌توان از روی پاسخ فرکانسی حلقه باز، پایداری سیستم حلقه بسته را ارزیابی نمود. در این مثال، تابع تبدیل حلقه باز $L(s)$ عبارتست از:

$$L(s) = P_{pend}(s)C(s) \quad (۴)$$

با وجود اینکه کنترلر در مسیر فیدبک سیستم وجود دارد، اما در معادله‌ی بالا ظاهر می‌شود.

در سایر مثال‌ها از دستور رسم دیاگرام بودی برای ترسیم پاسخ فرکانسی سیستم حلقه باز استفاده نمودیم. اما در این مثال، برای حالت بدون جبران‌سازی، به جای اینکه از دستور bode(P_Pend) استفاده کنیم، از ابزار Control System Designer استفاده خواهیم کرد. دیاگرام بودی حلقه باز در حال حاضر رسم شده است، اما در صورتی که نمایش داده نشده باشد یا اینکه مایل به رسم نمودار دیگری باشیم، از دکمه New Plot، نمودار جدیدی را رسم می‌کنیم.

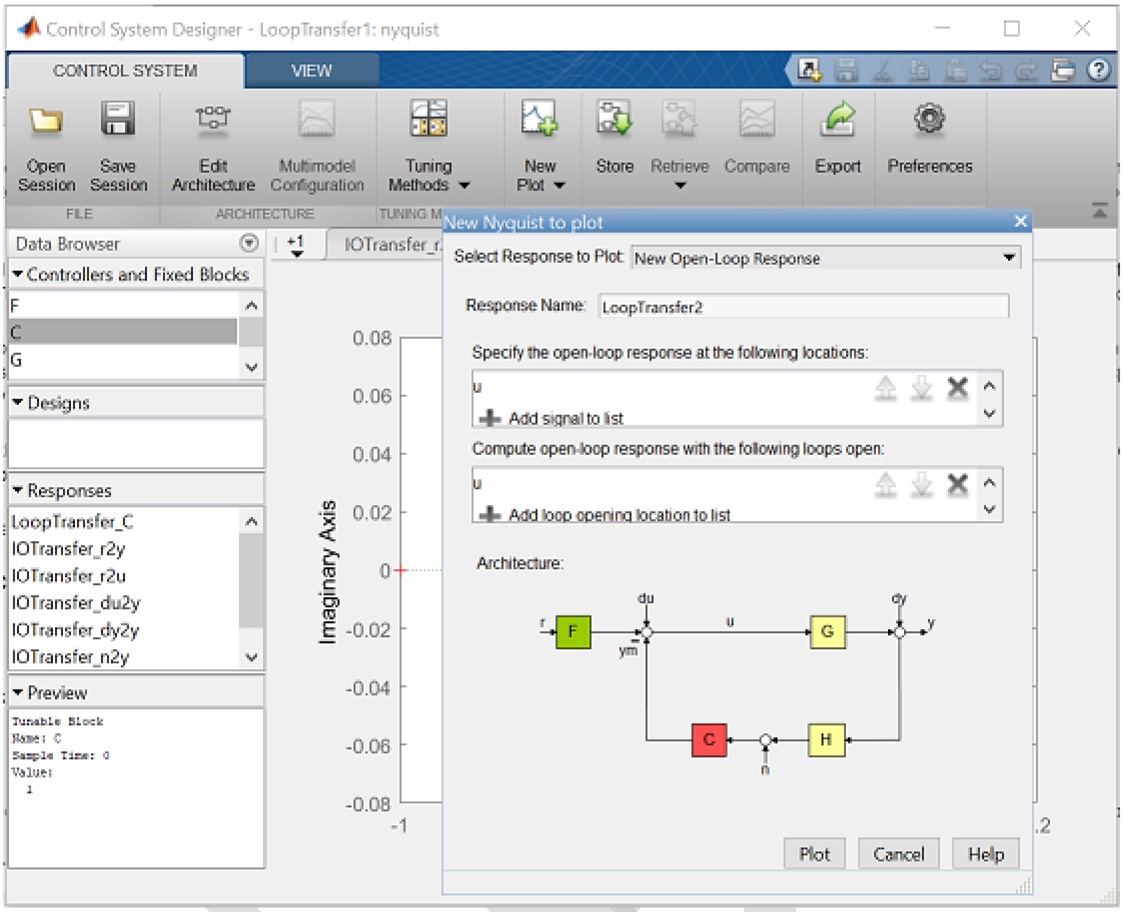


با بررسی دیاگرام بودی متوجه می‌شویم که برای تمام فرکانس‌ها اندازه کمتر از 0 dB و فاز بزرگتر از 180- درجه می‌باشد. برای یک سیستم مینیمم فاز، این عبارت به معنی پایدار بودن سیستم حلقه بسته با حد بهره و حد فاز بینهایت می‌باشد. اما از آنجایی که سیستم ما دارای یک قطب در سمت راست محور موهومی می‌باشد، سیستم ما نامینیمم فاز می‌باشد و سیستم حلقه بسته عملا ناپایدار است. برای اثبات این موضوع از چند نمودار تحلیل دیگر استفاده می‌کنیم.

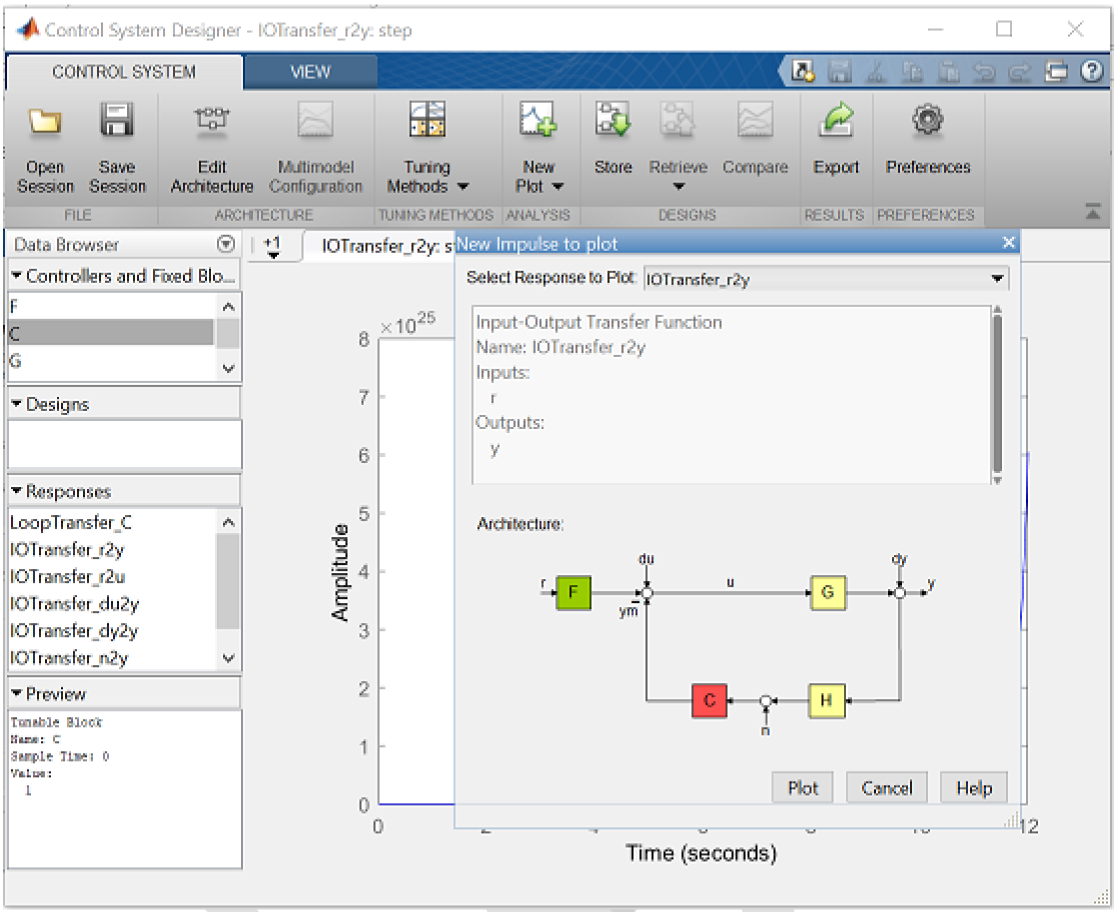
به طور کلی در هنگامی که با سیستم نامینیمم فاز کار می‌کنیم، ترجیحا از نمودار نایکوئست تابع تبدیل حلقه باز، پایداری نسبی را تحلیل می‌نماییم. همچنین در تحلیل سیستم‌های مرتبه بالاتر، نمودار نایکوئست ارجحیت دارد. زیرا دیاگرام بودی فرکانس‌هایی با فاصله ۳۶۰ را به عنوان فرکانس‌های جداگانه نشان می‌دهد در صورتی که در واقع آنها یکی هستند. به علت اینکه نمودار نایکوئست نمودار قطبی می‌باشد، این ابهام در آن وجود ندارد.

برای ایجاد نمودارهای دیگر جهت درک بهتر عملکرد سیستم حلقه بسته، بر روی New Plot کلیک کنید. ابزار Control System Designer این اجازه را به کاربر می‌دهد که تا ۸ نمودار را به طور همزمان مشاهده نماید. این نمودارها می‌توانند با تنظیمات مختلفی نمایش داده شوند، مثلا سیستم حلقه باز و حلقه بسته. ما نمودار نایکوئست سیستم حلقه باز و پاسخ ضربه سیستم حلقه بسته را که در زیر توضیح داده شده است ایجاد می‌نماییم.

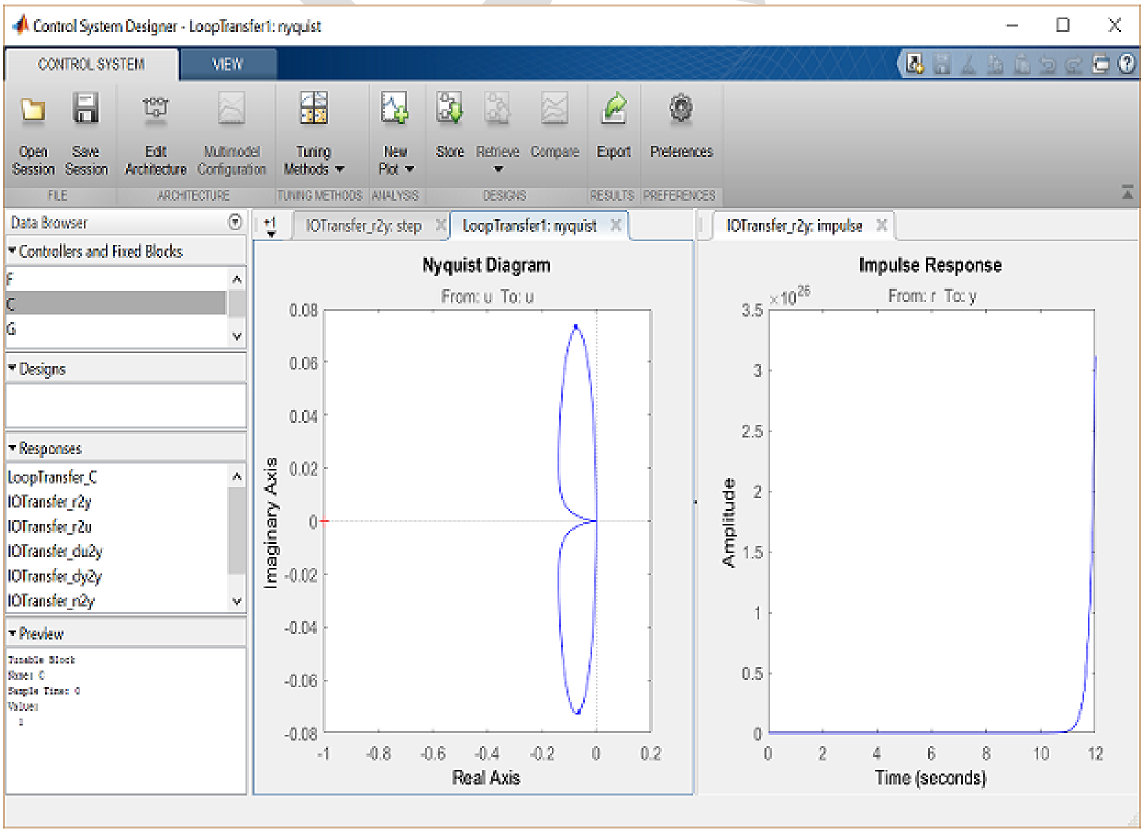
۱. در منوی New Plot، گزینه New Nyquist را انتخاب کنید. پنجره جدیدی با نام New Nyquist Plot نمایش داده می‌شود. از منوی کشوی Select Response to Plot، گزینه New Open-Loop Response را انتخاب کنید. در فیلدهای Specify the open-loop response at the following locations و Specify the open-loop response with the following loops open، سیگنال u را انتخاب کنید. سپس بر روی دکمه Plot کلیک کنید. همچنین با انتخاب LoopTransfer_C از منوی کشوی Select Response to Plot، نمودار نایکوئست مورد نظر به دست می‌آید.



۲. در منوی New Plot، بر روی New Impulse کلیک کنید. پنجره جدیدی با نام New Impulse to Plot نمایان می گردد. در منوی کشویی Select Response to Plot، گزینه IOTransfer_r2y را انتخاب کنید. سپس بر روی دکمه Plot کلیک کنید.



سپس مانند شکل زیر نمودارها نمایش داده می‌شوند:



با بررسی پاسخ ضربه متوجه می‌شویم که سیستم حلقه بسته ناپایدار است. همچنین می‌توان از نمودار حلقه باز نایکوئست و استفاده از معیار پایداری نایکوئست برای به دست آوردن این نتیجه استفاده کرد:

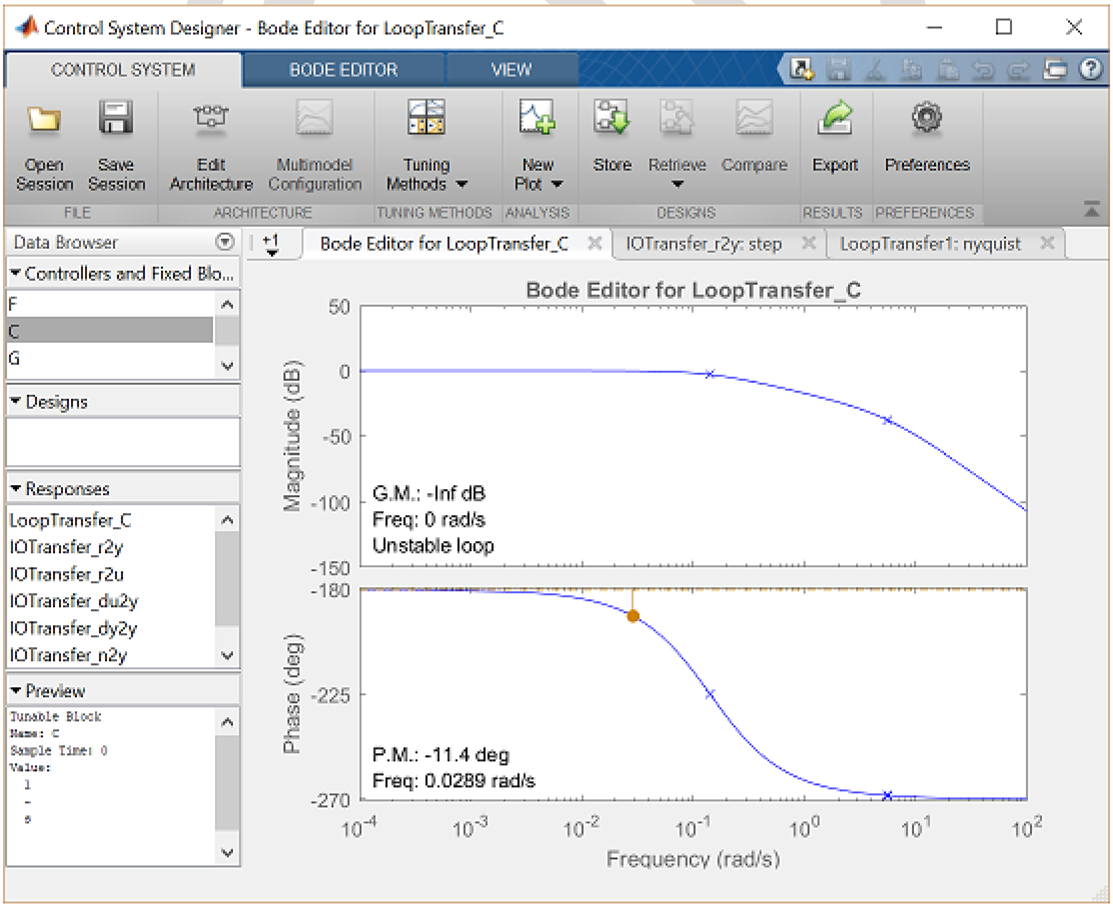
$$Z = P + N \qquad (۵)$$

که Z تعداد قطب‌های حلقه بسته در سمت راست محور موهومی، P تعداد قطب‌های حلقه باز در سمت راست محور موهومی و N تعداد دوران ساعتگرد منحنی حلقه باز نایکوئست حول نقطه‌ی 1- می‌باشد.

در بحث‌های انجام شده می‌دانیم که سیستم حلقه باز دارای یک قطب در نیم صفحه راست می‌باشد $(P = 1)$ و با بررسی نمودار نایکوئست بالا متوجه می‌شویم چرخشی حول نقطه 1- صورت نگرفته است $(N = 0)$. در نتیجه داریم $Z = 0 + 1 = 1$ و سیستم حلقه بسته دارای یک قطب در نیم صفحه راست دارد که بیانگر ناپایداری سیستم می‌باشد.

پاسخ حلقه بسته با جبران‌سازی

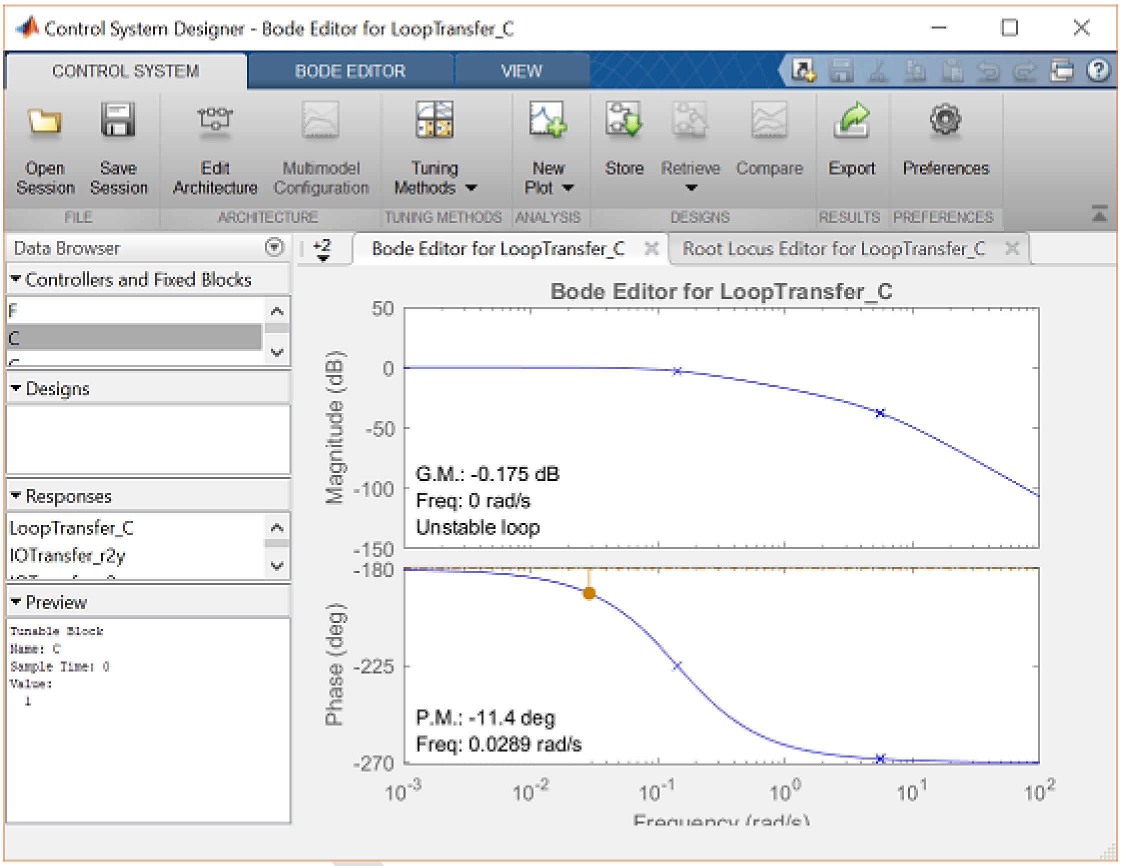
چون سیستم حلقه بسته بدون جبران‌ساز، ناپایدار است، لازم است از کنترلر استفاده کنیم تا آنرا پایدار کرده و نیازهای طراحی را برآورده سازیم. قدم اول اضافه کردن انتگرال‌گیر است تا صفر موجود در مبدا را خنثی کند. برای اضافه کردن انتگرال‌گیر، بر روی دیاگرام بودی راست کلیک کرده و از منوی نمایان شده گزینه Add Pole/Zero > Integrator را انتخاب کنید. دیاگرام بودی ایجاد شده در شکل زیر آورده شده است که در فرکانس‌های پایین رفتار مورد انتظار ما را دارد:



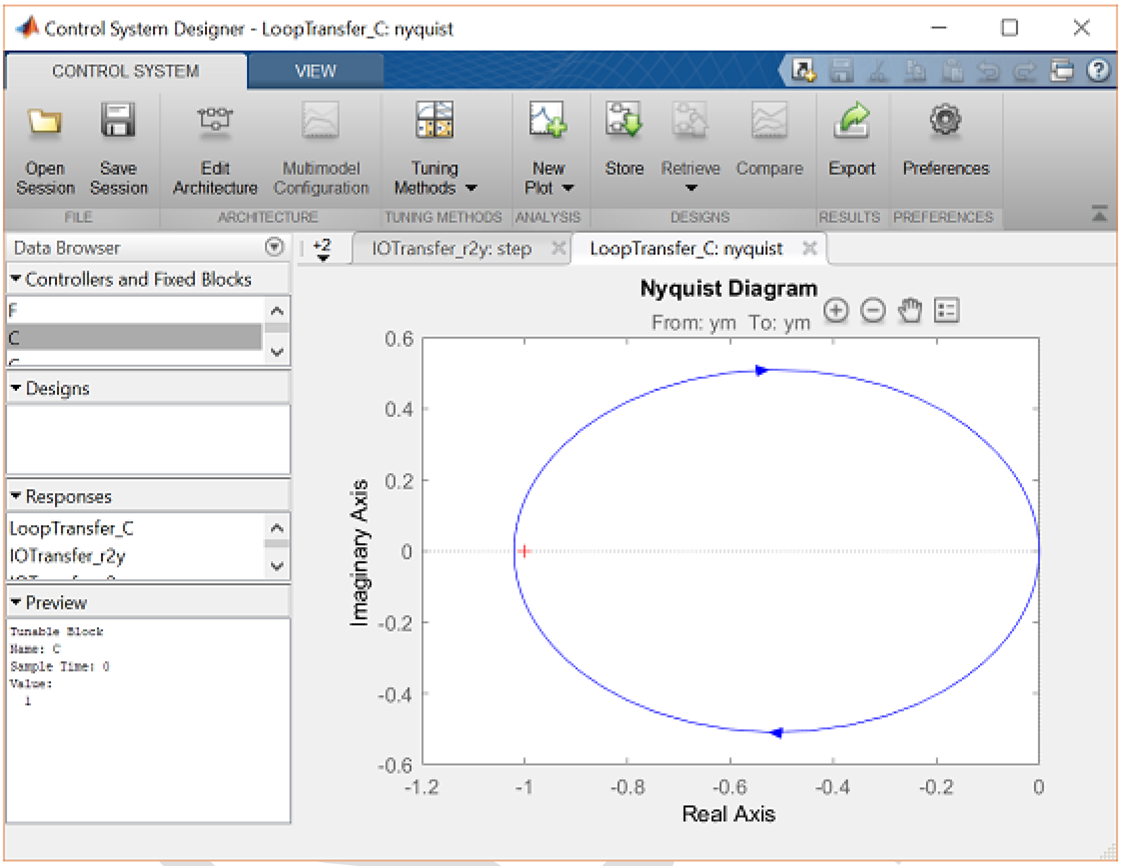
اگر با دقت به تابع تبدیل نگاه کنید متوجه خنثی شدن صفر و قطب در مبدا خواهید شد. اگرچه این خنثی‌سازی به طور اتوماتیک توسط ابزار Control System Designer تشخیص داده نمی‌شود و سبب خطای عددی در نمودارهای حاصله خواهد شد. بنابراین لازم است تا از ابزار Control System Designer خارج شده و آنرا دوباره با انتگرال‌گیر اضافه شده باز کنید که اینکار با دستور زیر انجام می‌گردد:

```
controlSystemDesigner('bode',minreal(P_pend*(1/s)))
```

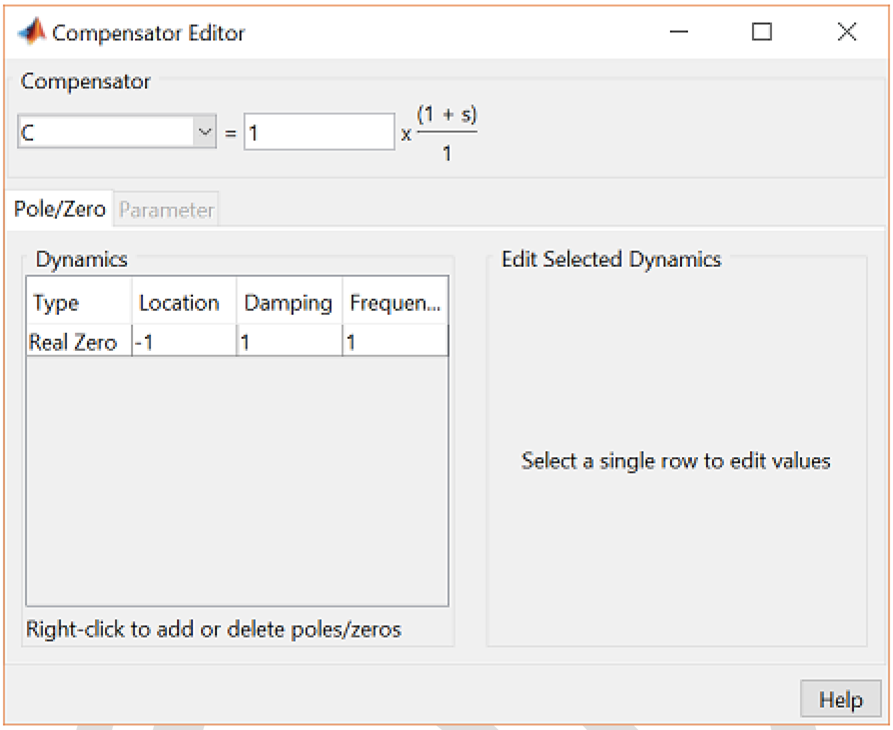
چون انتگرال‌گیر در مسیر مستقیم به سیستم متصل شده است در صورتی که کنترلر ما در مسیر فیدبک قرار دارد، از ابزار Control System Designer برای تحلیل پاسخ حلقه بسته استفاده نمی‌نماییم. اما برای تابع تبدیل حلقه باز تفاوتی نمی‌کند که کنترلر در مسیر مستقیم یا فیدبک قرار دارد بنابراین همچنان می‌توانیم از نمودارهای سیستم حلقه باز برای تحلیل و طراحی استفاده نماییم. دیاگرام بودی حاصل در زیر نمایش داده شده است:



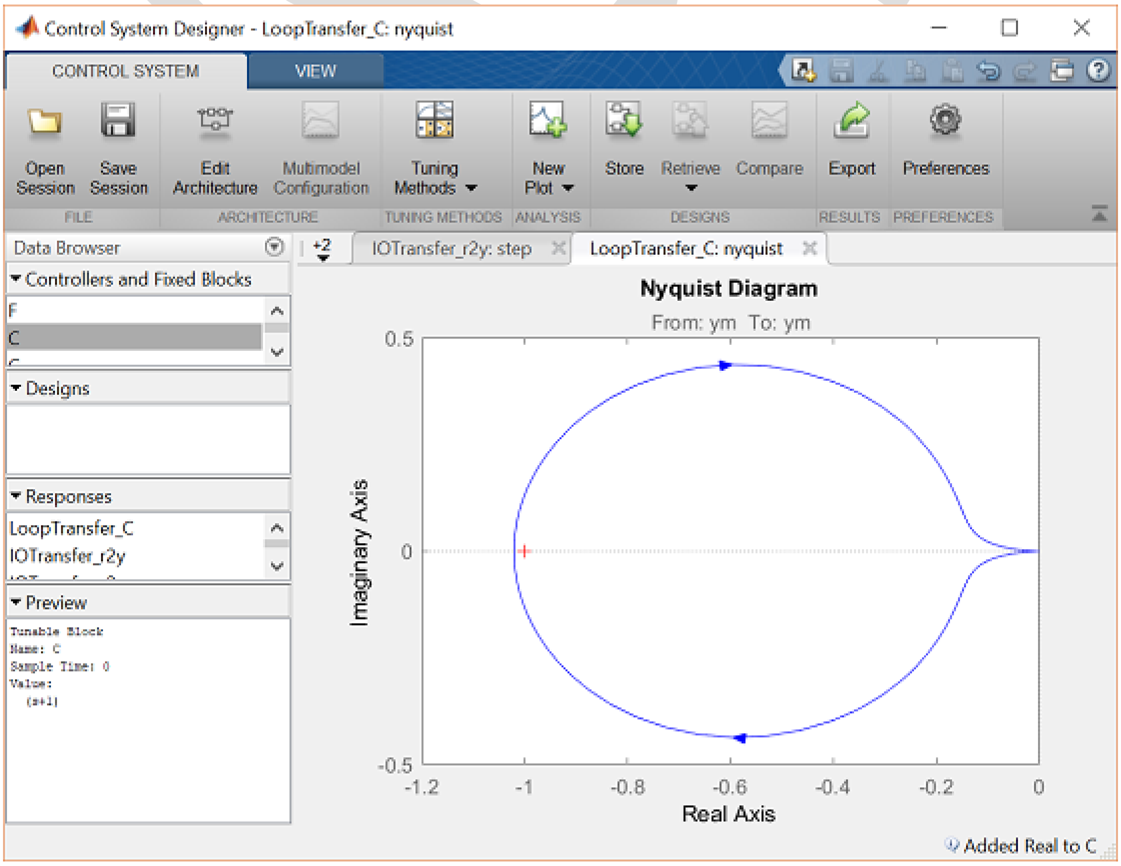
حتی با وجود اضافه شدن انتگرال‌گیر، همچنان سیستم حلقه بسته ناپایدار است. برای درک بهتر ناپایداری آن (و روش حل آن) از نمودار نایکوئست کمک می‌گیریم. مشابه آموزش‌های قبل نمودار نایکوئست را ایجاد می‌کنیم که در شکل زیر رسم شده است:



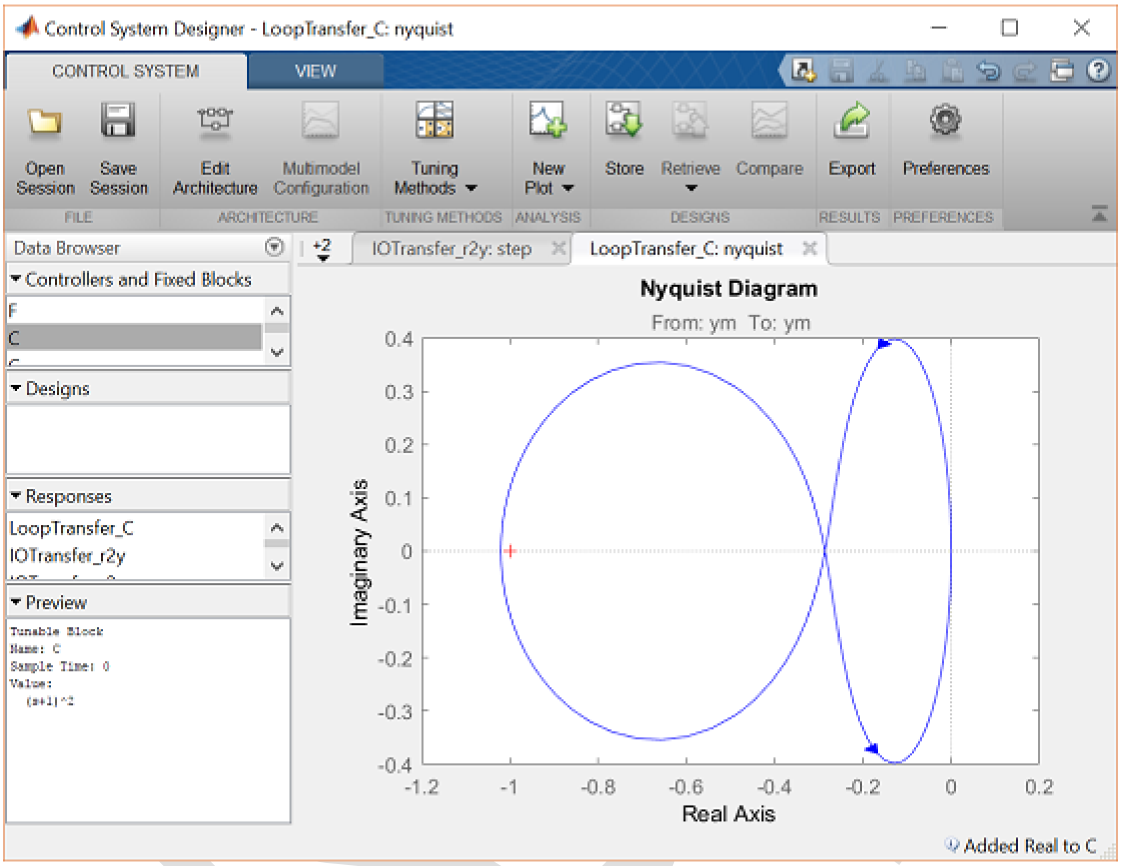
با مشاهده این نمودار متوجه چرخش نمودار نایکوئست حلقه باز به دور نقطه 1- در جهت ساعتگرد می‌شویم. این بدین معناست که سیستم حلقه بسته اکنون دارای دو قطب در نیم صفحه راست می‌باشد ($Z = P + N = 1 + 1 = 2$). در نتیجه سیستم حلقه بسته همچنان ناپایدار است. برای دوران پادساعتگرد نیاز به اضافه کردن فاز داریم. برای اضافه کردن فاز به سیستم، صفری را به آن اضافه می‌کنیم. برای شروع، صفر را در نقطه 1- قرار می‌دهیم و نمودارهای حاصل را بررسی می‌کنیم. برای اینکار می‌توان از روش گرافیکی مشابه کاری که برای اضافه کردن قطب انجام شد انجام دهیم اما به جای آن از پنجره Compensator Editor در ابزار Control System Designer که با کلیک راست بر روی نمودار نایکوئست و انتخاب Edit Compensator نمایش داده می‌شود، استفاده خواهیم کرد. سپس بر روی قسمت Dynamics پنجره Compensator Editor کلیک راست کرده و گزینه Add Pole/Zero > Real Zero را انتخاب کنید. به طور پیش‌فرض موقعیت صفر بر روی 1- است. در نهایت پنجره نمایش داده شده به شکل زیر خواهد بود:



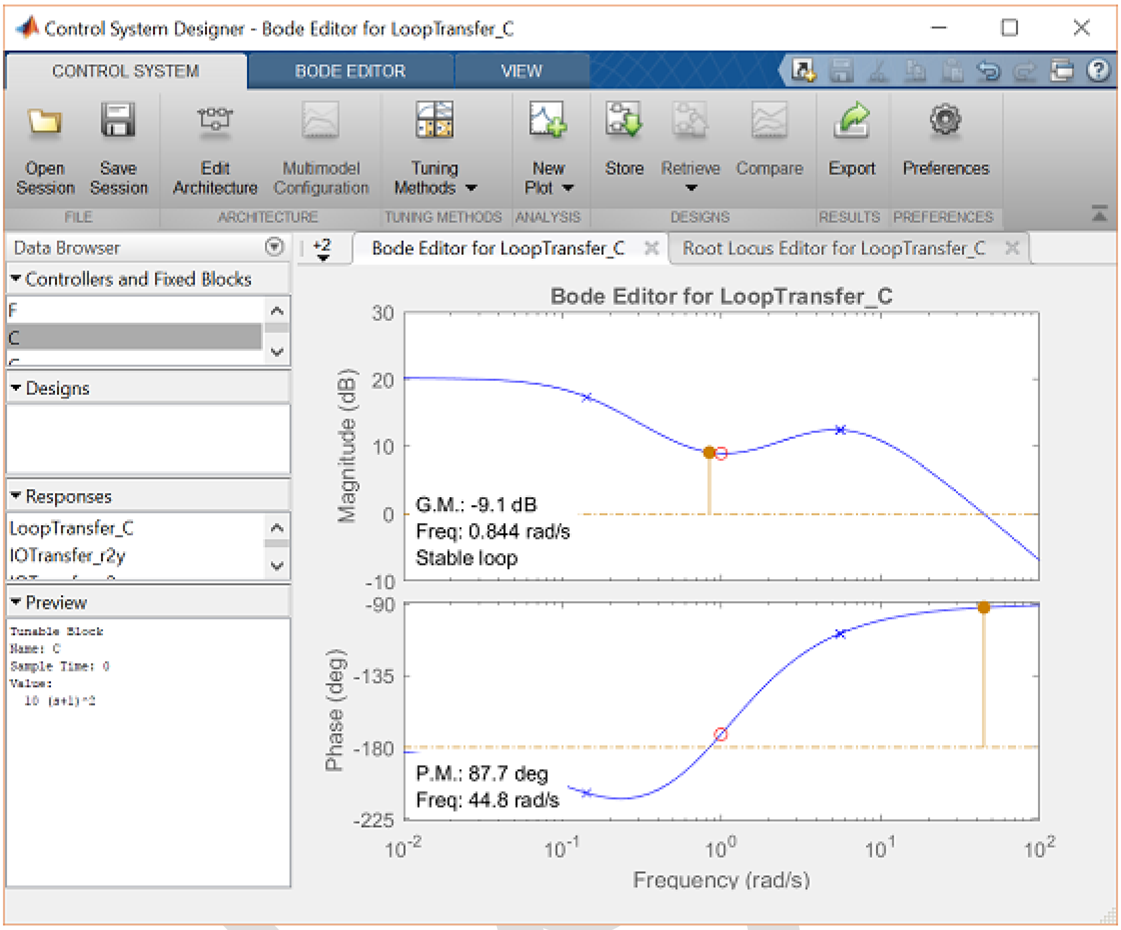
اضافه شدن صفر به طور خودکار نمودارهای بودی و نایکوئست را تغییر می‌دهد. نمودار نایکوئست به دست آمده در شکل زیر آورده شده است:



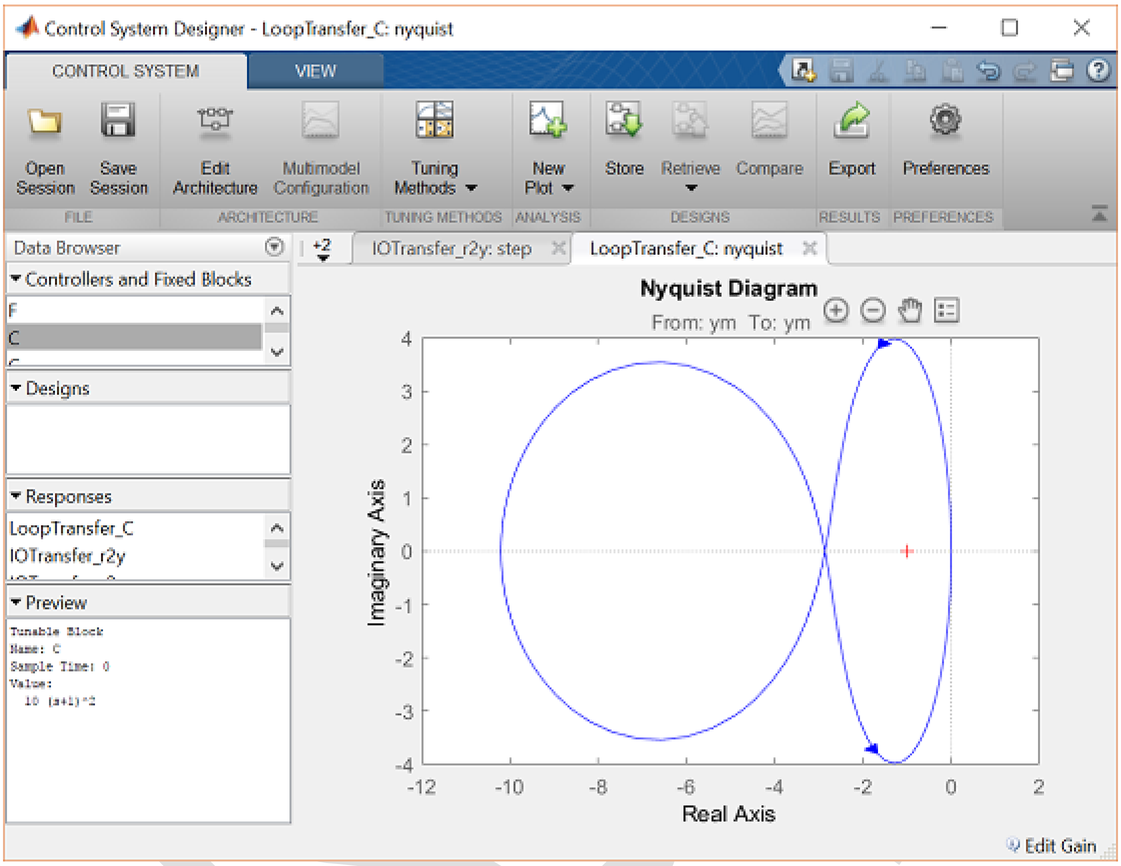
همانطور که مشاهده می‌کنید، این تغییر، فاز سیستم را به مقدار کافی نرسانده است. دوران حول ‎-1‎ همچنان ساعتگرد می‌باشد. با اضافه کردن صفر دوم در ‎-1‎ مانند قبل، نمودار نایکوئست به شکل زیر می‌شود:



باز هم همچنان یک دور ساعتگرد حول 1- وجود دارد. اما با اضافه کردن بهره می‌توانیم اندازه نقاط منحنی نایکوئست را تغییر دهیم و در نتیجه شعاع دایره پادساعتگرد را به قدری افزایش دهیم تا نقطه 1- را در بر گیرد. با این حساب داریم $N=-1$ چون دوران پادساعتگرد منفی حساب می‌شود. برای اینکار به صورت دستی مقدار بهره‌ی جدید را در پنجره‌ی Compensator Editor در ابزار Control System Designer وارد می‌کنیم. راه دیگر تغییر دادن بهره به صورت گرافیکی بر روی دیاگرام بودی می‌باشد. برای تغییر بهره در دیاگرام بودی، به پنجره دیاگرام بودی مراجعه کرده و بر روی منحنی اندازه کلیک کنید و آنرا تا جایی که دایره پادساعتگرد منحنی نایکوئست به 1- برسد بالا بکشید. تقریبا مقدار بهره ۳/۸ این ویژگی را ایجاد می‌کند. همانطورکه در قسمت Preview پنجره Control System Designer مشخص است، ما بهره را تا تقریبا تا اندازه ۱۰ بزرگ می‌کنیم. دیاگرام بودی حاصل به شکل زیر است:



دیاگرام نایکوئست مربوطه به شکل زیر است:

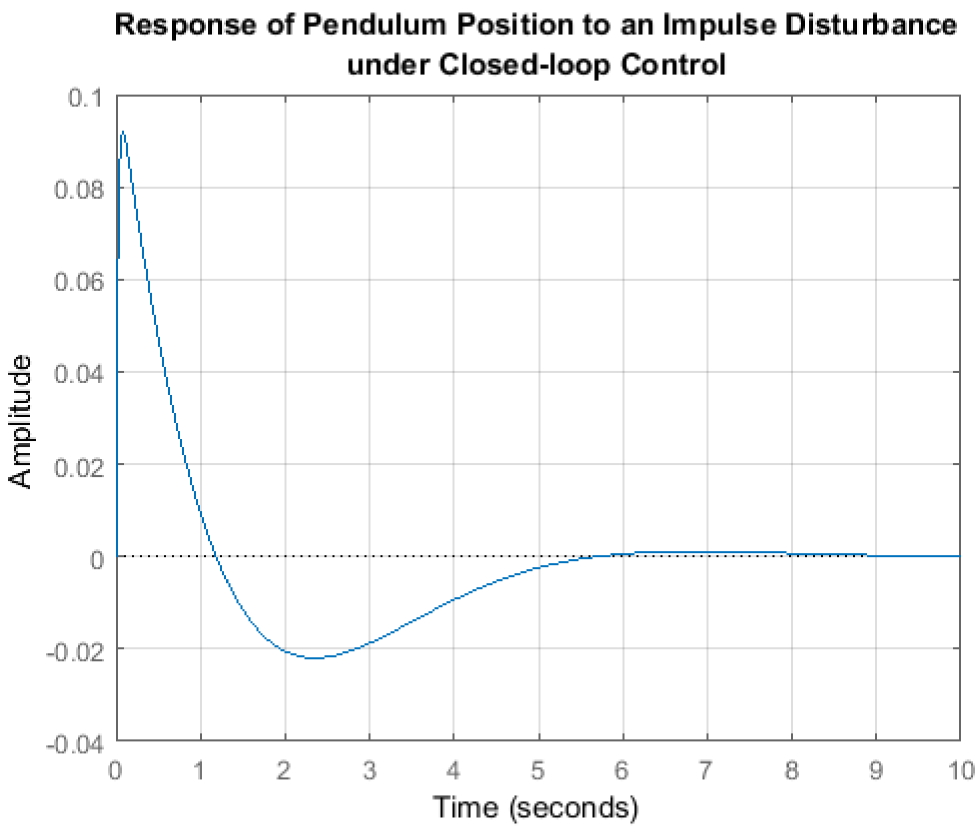


از بحث‌هایی که در بخش قبل انجام شد می‌دانیم که $P = 1$ و حال نیز $N = -1$ می‌باشد در نتیجه داریم $Z = -1 + 1 = 0$ که تعداد قطب‌های حلقه بسته در نیم صفحه راست می‌باشد و نشان دهنده پایدار بودن سیستم حلقه بسته می‌باشد. می‌توان با به دست آوردن پاسخ ضربه‌ی واحد سیستم به نیروی اغتشاش، پایداری سیستم و ارضا شدن نیازهای طراحی را بررسی نمود. چون انتگرال‌گیر با سیستم ادغام شده است، از ابزار Control System Designer خارج شده و پاسخ ضربه‌ی حلقه بسته را با دستور متلب به دست می‌آوریم. تا به حال کنترلری که طراحی شده است به شکل زیر می‌باشد:

$$C(s) = 10\frac{(s+1)^2}{s} \tag{۶}$$

با اضافه کردن کد زیر به امفایل خود، تابع تبدیلی از ورودی $F(s)$ به خروجی $\Phi(s)$ به دست می‌آید. با اجرای امفایل، نمودار پاسخ سیستم به ضربه به دست می‌آید.

```
K = 10;

C = K*(s+1)^2/s;

T = feedback(P_pend,C);

t = 0:0.01:10;

impulse(T,t), grid

title({'Response of Pendulum Position to an Impulse Disturbance';'under Closed-loop Control'});
```

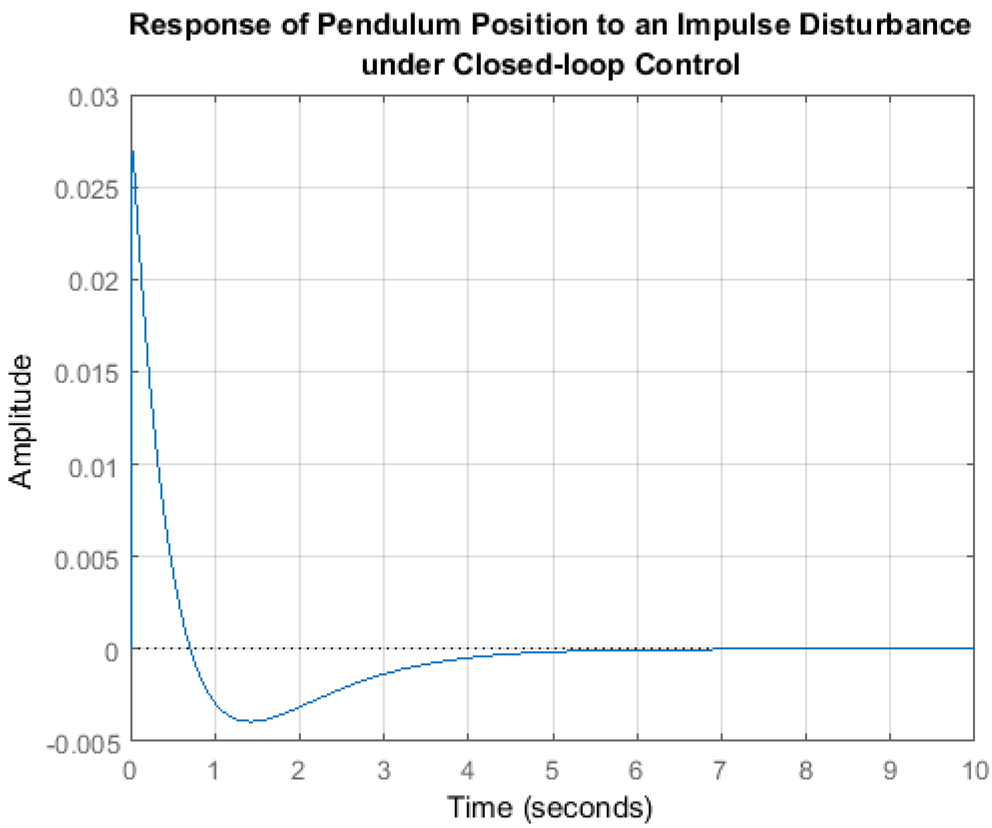


با بررسی نمودار بالا، واضح است که پاسخ سیستم پایدار می‌باشد. هرچند که فراجهش موقعیت پاندول از حد ۰/۰۵ رادیان عبور می‌کند و زمان نشست سیستم تا حدی بیشتر از ۵ ثانیه می‌باشد. در قدم بعد بر روی بهبود پاسخ سیستم می‌پردازیم. می‌توانیم از ابزار Control System Designer برای مشاهده تاثیر بهره کنترلر و موقعیت صفرها بر روی منحنی پاسخ فرکانسی سیستم استفاده نماییم. در این صورت با اضافه شدن بهره کنترلر، حاشیه فاز سیستم افزایش یافته که سبب کاهش فراجهش پاسخ می‌گردد. علاوه بر آن، سعی و خطا نشان می‌دهد که جابجا کردن یکی از صفرها به سمت چپ صفحه‌ی مختلط (منفی‌تر شدن) پاسخ سیستم را سریع‌تر می‌سازد. این تغییر، فراجهش را افزایش می‌دهد اما می‌شود با افزایش بهره آنرا جبران نمود. با سعی و خطا، کنترلری که خواسته‌های مسئله را ارضا کند به شکل زیر می‌باشد:

$$C(s) = 35\frac{(s+1)(s+2)}{s} \tag{۷}$$

امفایل خود را به شکل زیر تصحیح و دوباره اجرا کنید تا نمودار پاسخ ضربه به دست آید:

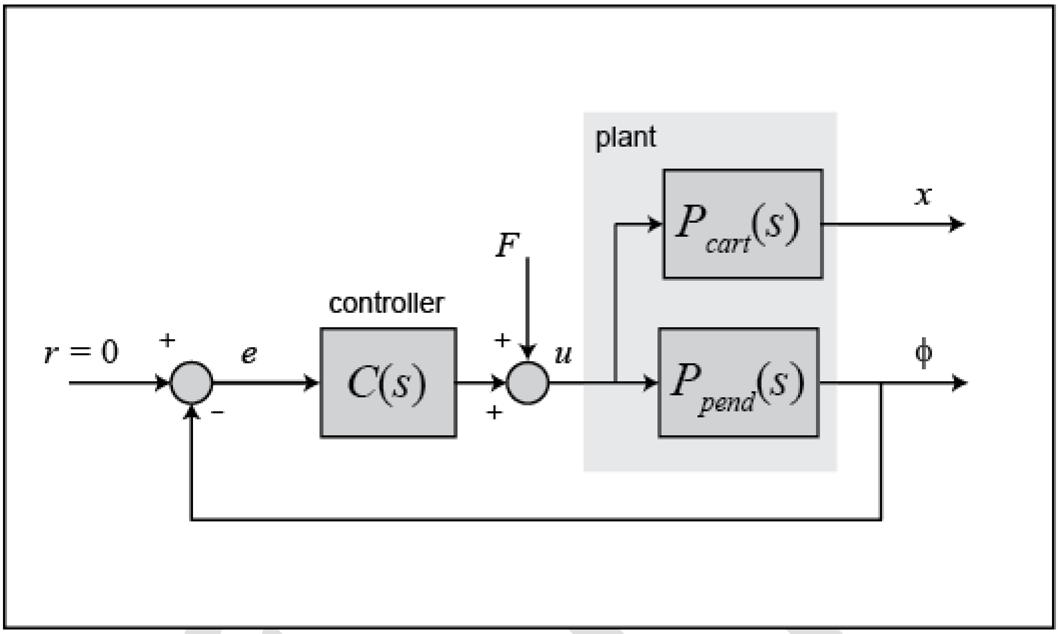
```
K = 35;

C = K*(s+1)*(s+2)/s;

T = feedback(P_pend,C);

t = 0:0.01:10;

impulse(T, t), grid

title({'Response of Pendulum Position to an Impulse Disturbance';'under Closed-loop Control'});
```



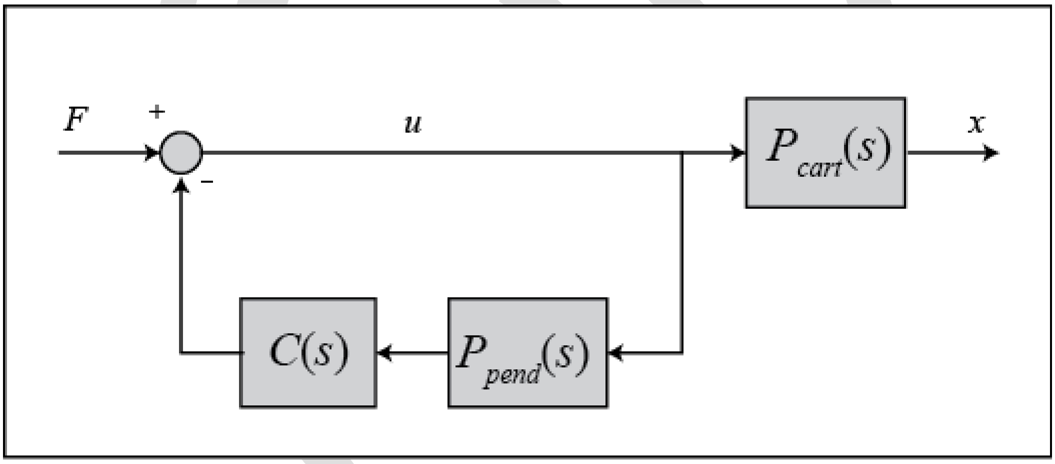
پاسخ به دست آمده خواسته‌های طراحی را ارضا می‌کند. پارامترها را تغییر دهید و تاثیر آنها را مشاهده کنید. لازم به ذکر است که می‌توان در ابزار Control System Designer پاسخ سیستم به اغتشاش ضربه را به دست آورد. ادغام انتگرال‌گیر با کنترلر (به جای ادغام آن با سیستم) سبب بروز خطاهای عددی که در بالا مشاهده کردیم می‌شود. اما اینکار تاثیر محسوسی بر روی پاسخ ضربه‌ی محاسبه شده توسط متلب ندارد.

موقعیت ارابه چگونه تغییر می کند؟

در ابتدای این بخش، دیاگرام بلوکی سیستم پاندول معکوس نمایش داده شده است. دیاگرام رسم شده کامل نمی‌باشد. بلوکی که بیانگر پاسخ موقعیت ارابه x می‌باشد در این دیاگرام آورده نشده است زیرا متغیر آن کنترل نمی‌شود. هرچند که برای ما جالب است بدانیم که موقعیت ارابه در هنگام کنترل زاویه پاندول به چه صورتی تغییر می‌کند. برای اینکار باید بلوک دیاگرام کل سیستم را به شکل زیر در نظر بگیریم:



با تغییر چیدمان داریم:



در دیاگرام بالا، $C(s)$ کنترلری است که برای حفظ موقعیت پاندول طراحی شده است. تابع تبدیل حلقه بسته‌ی $T_2(s)$ از ورودی نیروی وارد شده به ارابه به خروجی موقعیت ارابه تعریف شده است، در نتیجه داریم:

$$T_2(s)=\frac{X(s)}{F(s)}=\frac{P_{cart}(s)}{1+P_{pend}(s)C(s)} \tag{۴}$$

با مراجعه به قسمت مدل‌سازی پاندول معکوس، تابع تبدیل $P_{cart}(s)$ عبارتست از:

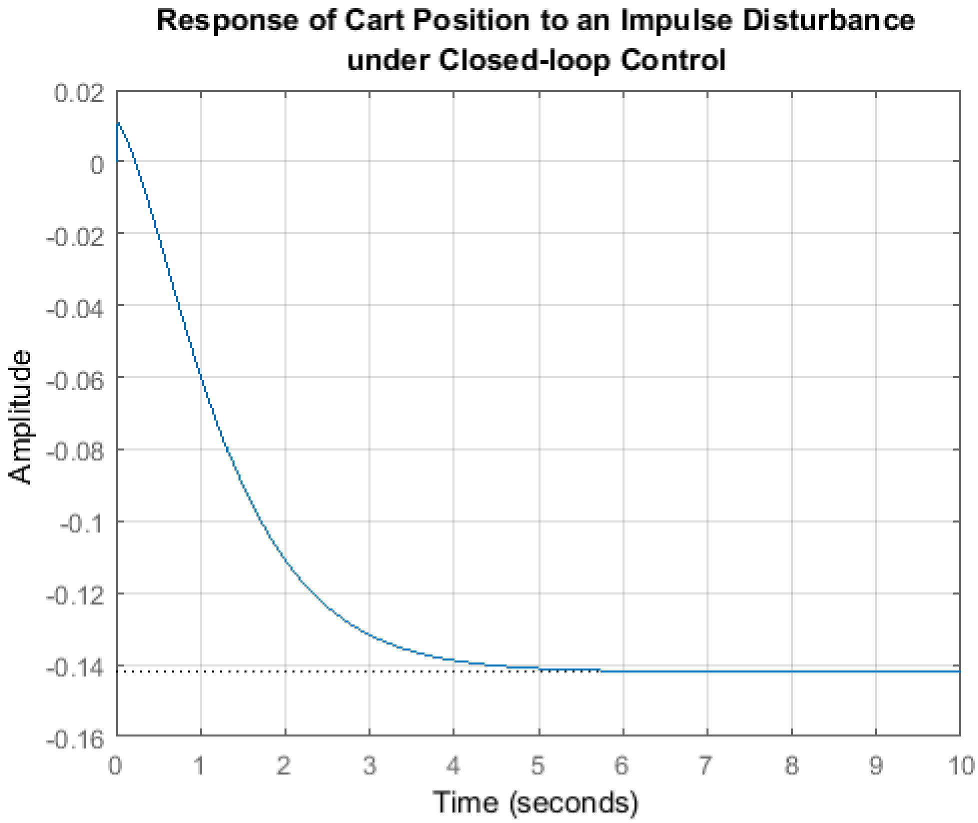
$$P_{cart}(s)=\frac{X(s)}{U(s)}=\frac{\frac{(I+ml^2)s^2-gml}{q}}{s^4+\frac{b(I+ml^2)}{q}s^3-\frac{(M+m)mgl}{q}s^2-\frac{bmgl}{q}s}\quad\left[\frac{m}{N}\right] \tag{۵}$$

که:

$$q=[(M+m)(I+ml^2)-(ml)^2] \tag{۶}$$

با اضافه کردن دستورات زیر (با فرض از پیش تعریف شده بودن $P_{pend}(s)$ و $C(s)$ در متلب) پاسخ موقعیت ارابه به اغتشاش ضربه به دست می‌آید:

```
P_cart = (((I+m*l^2)/q)*s^2 - (m*g*l/q))/(s^4 + (b*(I + m*l^2))*s^3/q - ((M + m)*m*g*l)*s^2/q - b*m*g*l*s/q);

T2 = feedback(1,P_pend*C)*P_cart;

T2 = minreal(T2);

t = 0:0.01:10;

impulse(T2, t), grid

title({'Response of Cart Position to an Impulse Disturbance';'under Closed-loop Control'});
```



دستور minreal به طور مفیدی تمامی قطب و صفرهای مشترک را در تابع تبدیل حلقه بسته خنثی می‌کند. این کار سبب عملکرد عددی بهتری در تابع impulse می‌شود. همانطور که مشاهده می‌کنید ارابه به در جهت منفی حرکت کرده و در - 0.14 متر پایدار می‌شود. این روش را به خوبی می‌توان توسط یک کنترلر واقعی با فرض فضای کافی برای حرکت ارابه پیاده‌سازی نمود. به یاد داشته باشید که کنترل ارابه، اتفاق بوده است و ما کنترل خود را بر اساس پایداری موقعیت ارابه طراحی نکرده‌ایم، در واقع چیزی که به دست آمد از بخت خوب ماست!

بخش ششم: طراحی کنترلر در فضای حالت

فهرست مطالب بخش

- قطب‌های حلقه باز
- رگولاسیون درجه دوم خطی (LQR[38])
- اضافه کردن پیش جبران‌سازی
- کنترل مشاهده‌گر-محور[۳۹]

دستورهای کلیدی متلب در این بخش:

```
ss , eig , lsim , lqr , ctrb , plotyy , obsv , place
```

معادلات دینامیکی سیستم پاندول معکوس در فرم فضای حالت برای مسئله‌ی اصلی عبارتست از:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\phi}\\ \ddot{\phi}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & -0.1818 & 2.6727 & 0\\ 0 & 0 & 0 & 1\\ 0 & -0.4545 & 31.1818 & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix}+\begin{bmatrix}0\\ 1.8182\\ 0\\ 4.5455\end{bmatrix}u \qquad (۱)$$

$$y=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix}+\begin{bmatrix}0\\ 0\end{bmatrix}u \qquad (۲)$$

برای به دست آوردن این معادلات به **بخش اول: مدل‌سازی سیستم** مراجعه کنید. خروجی‌های این مسئله عبارتند از جابجایی ارابه (x بر حسب متر) و زاویه‌ی پاندول (ϕ بر حسب رادیان) که ϕ بیانگر میزان انحراف پاندول از موقعیت عمودی خود در بالا می‌باشد یعنی $\theta = \pi + \phi$.

نیازهای طراحی با دستور پله ۰/۲ متر برای موقعیت ارابه x به شرح زیر است:

- زمان نشست برای x و θ کمتر از ۵ ثانیه
- زمان نمو برای x کمتر از ۰/۵ ثانیه
- زاویه پاندول θ هیچگاه بیشتر از ۲۰ درجه (۰/۳۵ رادیان) از حالت عمودی نشود
- خطای حالت ماندگار کمتر از ۲٪ برای x و θ

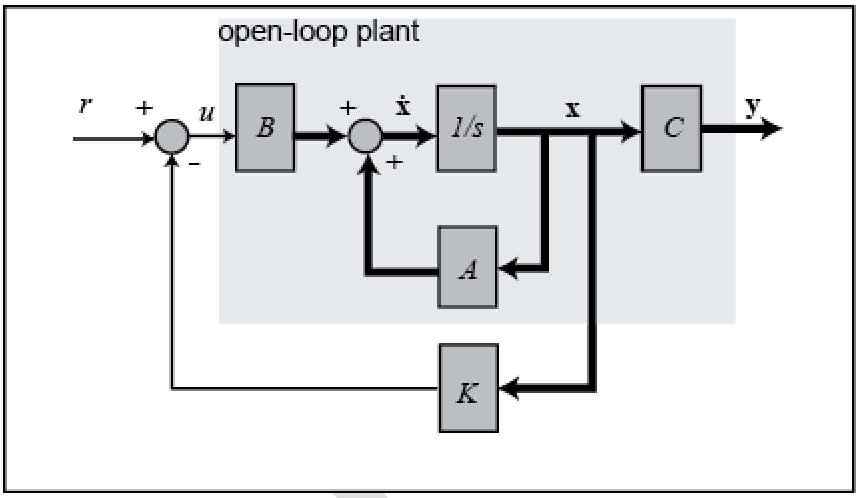
همانطور که ممکن است متوجه شده باشید، نیازهای طراحی در این فصل متفاوت با فصل‌های پیشین می‌باشد. در فصل‌های دیگر هدف از طراحی ثابت بودن موقعیت عمودی پاندول در هنگام اعمال اغتشاش ضربه به ارابه بود و تلاشی برای کنترل موقعیت ارابه نمی‌شد. در این فصل سعی داریم تا موقعیت عمودی پاندول را حفظ کرده و همزمان ارابه را به ۰/۲ متر به سمت راست جابجا کنیم. روش فضای حالت برای کنترل سیستم‌های با چند خروجی مانند این مثال بسیار مناسب می‌باشد.

برای حل این مسئله می‌توان از فیدبک تمام حالات استفاده نمود. شماتیک اینگونه از مسائل کنترل در شکل زیر نشان داده شده است که در آن K ماتریس بهره‌های کنترل می‌باشد. لازم به ذکر است که در اینجا به جای گرفتن فیدبک از خروجی سیستم، از تمامی حالت‌های سیستم فیدبک می‌گیریم.

---

[38] Linear Quadratic Regulation

[۳۹] Observer-based



قطب‌های حلقه باز

در این مسئله، r بیانگر ورودی پله موقعیت ارابه می‌باشد. چهار حالتِ سیستم بیانگر موقعیت و سرعت ارابه و زاویه و سرعت زاویه‌ای پاندول می‌باشند. خروجی y شامل هر دو موقعیت ارابه و زاویه پاندول است. می‌خواهیم کنترلی طراحی کنیم تا با ورودی مرجع پله به سیستم، پاندول جابجا شود اما در نهایت به صفر (موقعیت عمودی) بازگردد و ارابه به موقعیت جدید خود حرکت کند. برای مشاهده پاسخ حلقه باز سیستم به **بخش دوم: تحلیل سیستم** مراجعه کنید.

اولین قدم در طراحی یک کنترلر فیدبک تمام حالات، مشخص کردن قطب‌های حلقه باز سیستم می‌باشد. خطوط کد زیر را در یک ام‌فایل وارد کنید. بعد از اجرای آن، در خروجی لیستی از قطب‌های حلقه باز (مقادیر ویژه ماتریس A) نمایش داده می‌شود:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;



p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices

A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
```

```
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);

poles = eig(A)
```

```
poles =

         0

   -0.1428

   -5.6041

    5.5651
```

مشاهده می‌شود که یکی از قطب‌ها در نیم‌صفحه سمت راست یعنی در ۵/۵۶۵۱ قرار دارد. بدین ترتیب شکی نداریم که سیستم حلقه باز ناپایدار است.

رگولاسیون درجه دوم خطی (LQR)

قدم بعد در روند طراحی، به دست آوردن بردار بهره‌های کنترل فیدبک حالات یا K با فرض دسترسی به تمام چهار متغیر حالت (قابل اندازه‌گیری بودن تمام حالات) می‌باشد. اینکار را می‌توان به روش‌های گوناگونی انجام داد. اگر موقعیت مطلوب قطب‌های حلقه بسته را می‌دانید می‌توانید از دستور place یا acker در متلب استفاده کنید. راه دیگر استفاده از دستور lqr می‌باشد که بهره‌ی کنترلر بهینه را با فرض خطی بودن سیستم، تابع هزینه‌ی درجه دوم و ورودی مرجع صفر به ما می‌دهد (برای اطلاعات بیشتر به کتب مرجع رجوع کنید).

پیش از طراحی کنترلر، کنترل‌پذیری سیستم را بررسی می‌نماییم. کنترل‌پذیر بودن سیستم به معنی این است که می‌توانیم در هر زمان محدود (تا جایی که قیدهای فیزیکی سیستم اجازه می‌دهند) حالت‌های سیستم را تغییر دهیم. برای یک سیستم با کنترل‌پذیری تمام حالات، باید ماتریس کنترل‌پذیری از رنک n باشد که n همان تعداد ردیف‌های (یا ستونها) مستقل خطی یک ماتریس می‌باشد. ماتریس کنترل‌پذیری یک سیستم به فرم نشان داده در زیر است. عدد n بیانگر تعداد متغیرهای حالت سیستم می‌باشد. اضافه کردن جملاتی از A با توان بالاتر به ماتریس کنترل‌پذیری، رنک این ماتریس را افزایش نخواهد داد زیرا جملات اضافه شده تنها ترکیبی خطی از سایر جملات پیشین می‌باشند.

$$\mathcal{C} = [B \quad AB \quad A^2B \quad \cdots \quad A^{n-1}B] \qquad (۳)$$

چون ماتریس کنترل‌پذیری در اینجا 4×4 می‌باشد، رنک ماتریس باید ۴ باشد. از دستور ctrb متلب برای تشکیل ماتریس کنترل‌پذیری و از دستور rank برای تعیین رنک آن استفاده خواهیم کرد. به امفایل خود دستورات زیر را اضافه کنید و با اجرای آن، خروجی زیر را دریافت نمایید:

```
co = ctrb(sys_ss);

controllability = rank(co)
```

```
controllability =

     4
```

بنابراین سیستم ما کنترل‌پذیر بوده و در نتیجه می‌توانیم کنترلری را طراحی کنیم تا خواسته‌های مورد نظر را ارضا کند. از روش رگولاسیون درجه دوم خطی برای تعیین ماتریس بهره کنترل فیدبک حالات K استفاده خواهیم کرد. دستور lqr در متلب به کاربر اجازه‌ی تعیین دو پارامتر R و Q را می‌دهد که به ترتیب اهمیت نسبی سیگنال کنترلی (u) و خطای سیستم

(انحراف از صفر) در تابع هزینه‌ای که قصد بهینه‌سازی آنرا داریم را مشخص می‌کنند. در ساده‌ترین حالت $R = 1$ و $Q = C'C$ را در نظر می‌گیریم. تابع هزینه متناسب با این R و Q، اهمیت یکسانی را برای کنترل و متغیرهای حالت (که خروجی‌ها هستند) تعیین می‌کند. روش lqr قابلیت کنترل هر دو خروجی را به ما می‌دهد. در این مثال، کنترل هر دو خروجی بسیار ساده است. کنترلر را می‌توان با تغییر المان‌های غیرصفر ماتریس Q به گونه‌ای تنظیم نمود تا پاسخ مطلوب حاصل گردد. برای مشاهده‌ی ساختار Q، کد زیر را در پنجره دستور متلب وارد کنید تا خروجی آن به شکل زیر نمایش داده شود:

```
Q = C'*C
```

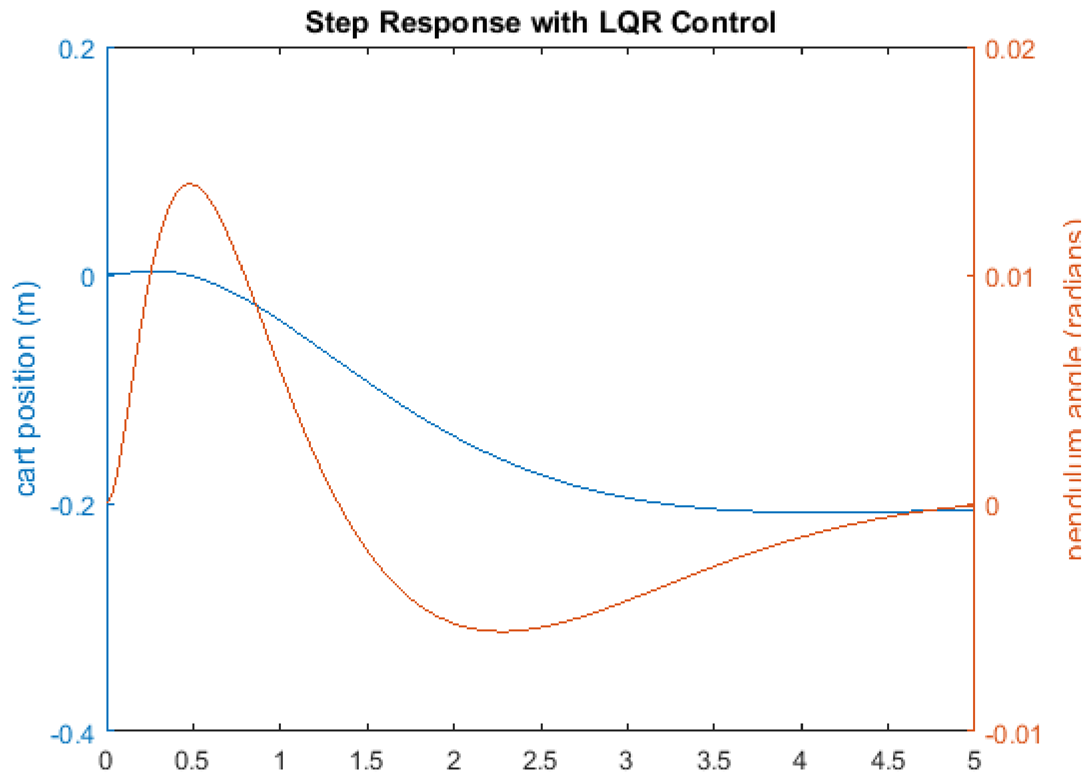
```
Q =

       1       0       0       0

       0       0       0       0

       0       0       1       0

       0       0       0       0
```

المان موجود در (۱،۱) ماتریس Q بیانگر وزن موقعیت ارابه و المان موجود در (۳،۳) بیانگر وزن زاویه پاندول می‌باشد. وزن ورودی R یک می‌باشد. در نهایت چیزی که تعیین کننده است مقادیر نسبی Q و R می‌باشد و نه مقادیر مطلق آنها. حال که تفسیر ماتریس Q را می‌دانیم می‌توانیم با آزمون، ماتریس K که کنترلر خوبی را ایجاد می‌کند به دست آوریم. پس در قدم بعد ماتریس K را انتخاب کرده و هر دو پاسخ (موقعیت ارابه و زاویه پاندول) را در یک نمودار رسم می‌کنیم تا بتوانیم تغییرات در کنترل و پاسخ‌ها را به طور همزمان مشاهده کنیم. دستورات زیر را در انتهای ام‌فایل خود اضافه کرده و آنرا اجرا کنید تا مقدار K و نمودار پاسخ سیستم به دست آید:

```
Q = C'*C;

R = 1;

K = lqr(A,B,Q,R)



Ac = [(A-B*K)];

Bc = [B];

Cc = [C];

Dc = [D];



states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
```

```
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with LQR Control')
```

```
K =

   -1.0000   -1.6567   18.6854    3.4594
```

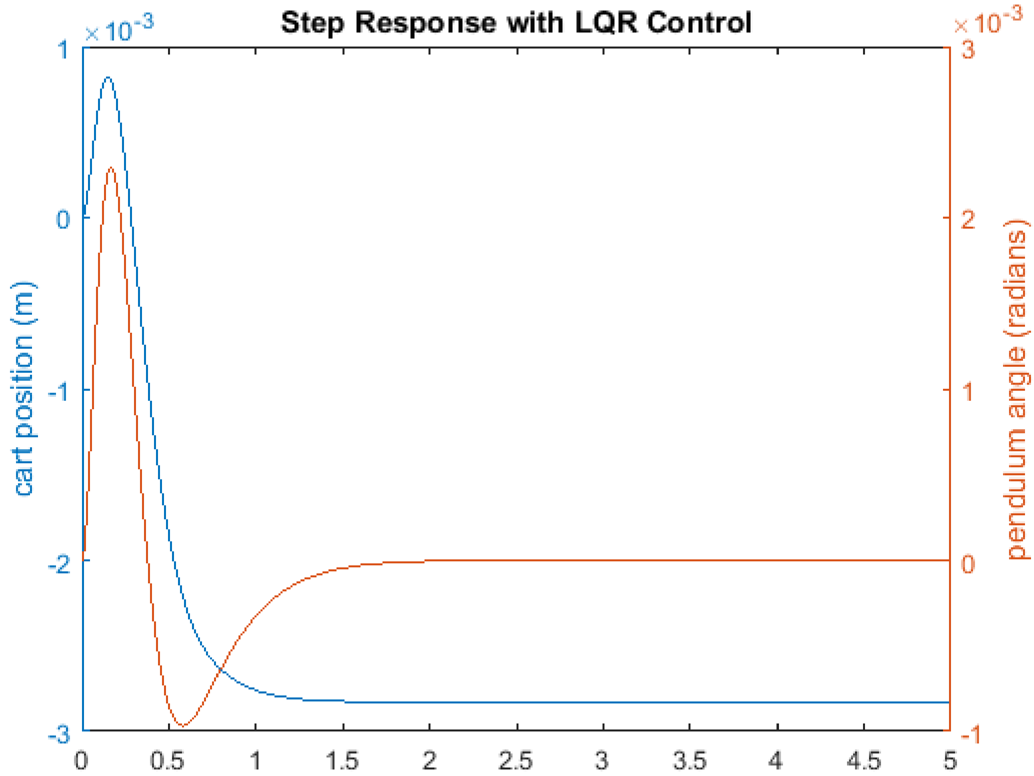


خط قرمز بیانگر زاویه پاندول به رادیان و خط آبی بیانگر موقعیت ارابه به متر می‌باشد. همانطور که مشاهده می‌کنید، این نمودار رضایت‌بخش نمی‌باشد. فراجهش پاندول و ارابه مناسب می‌باشد اما زمان نشست آنها باید بهبود و زمان نمو ارابه نیز کاهش یابد. ممکن است متوجه شده باشید که موقعیت ارابه در نهایت اصلا به موقعیتی که باید باشد نزدیک نمی‌باشد و در جهت عکس حرکت کرده است. در قسمت بعد این مشکل را حل می‌کنیم و اکنون تنها بر روی زمان نشست و نمو تمرکز می‌کنیم. به امفایل خود بازگشته و ماتریس Q را تغییر دهید تا پاسخ بهتری را به دست آورید. درمی‌یابیم که با افزایش مقادیر المان‌های (۱،۱) و (۳،۳) زمان نشست و نمو کاهش یافته و زاویه حرکت پاندول نیز کمتر می‌شود. به بیان دیگر با اینکار شما وزن بیشتری را بر روی خطا نسبت به سیگنال کنترلی در تابع هزینه قرار می‌دهید. در امفایل خود المان (۱،۱) را برابر ۵۰۰۰ و المان (۳،۳) را برابر ۱۰۰ قرار دهید تا K و پاسخ زیر به دست آید:

```
Q = C'*C;

Q(1,1) = 5000;

Q(3,3) = 100

R = 1;

K = lqr(A,B,Q,R)
```

```
Ac = [(A-B*K)];

Bc = [B];

Cc = [C];

Dc = [D];



states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with LQR Control')
```

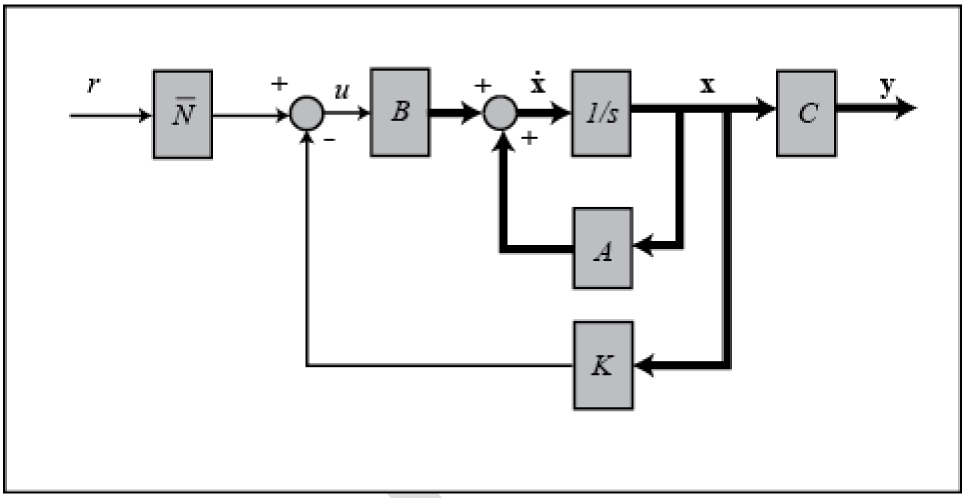
```
Q =

        5000            0            0            0

           0            0            0            0

           0            0          100            0

           0            0            0            0

K =

  -70.7107  -37.8345  105.5298   20.9238
```



شاید متوجه شده باشید که با افزایش بیشتر المان‌های Q می‌توانید پاسخ را هرچه بیشتر بهبود بخشید. دلیلی که ما این وزن‌ها را انتخاب کردیم این است که خواسته‌های پاسخ گذرا را برآورده می‌کنند. افزایش بیشتر اندازه Q سبب کاهش خطای ردیابی شده اما نیروی کنترلی u را نیز افزایش می‌دهد. افزایش سیگنال کنترلی سبب هزینه بیشتر (انرژی بیشتر، عملگر قوی‌تر و ...) خواهد شد.

اضافه کردن پیش جبران‌سازی

کنترلی که تا به حال طراحی کردیم خواسته‌های گذرا را برآورده می‌سازد اما همچنان خطای حالت ماندگار حل نشده باقی مانده است. برخلاف روش‌های طراحی دیگر که از خروجی فیدبک گفته و آنرا با ورودی مرجع مقایسه کرده و خطا را تشکیل می‌دهیم، در کنترلر فیدبک تمام حالات، از تمامی متغیرهای حالت فیدبک می‌گیریم. در این صورت باید مقدار حالت ماندگار هر حالت را به دست آوریم و آنرا در بهره‌ی موجود در K ضرب کنیم و از این مقدار جدید به عنوان ورودی مرجع استفاده کنیم. می‌توان از بهره‌ی ثابت $\bar{N}$ در مسیر ورودی مرجع استفاده نمود. در شماتیک زیر رابطه‌ی این مقادیر را مشاهده می‌کنید:



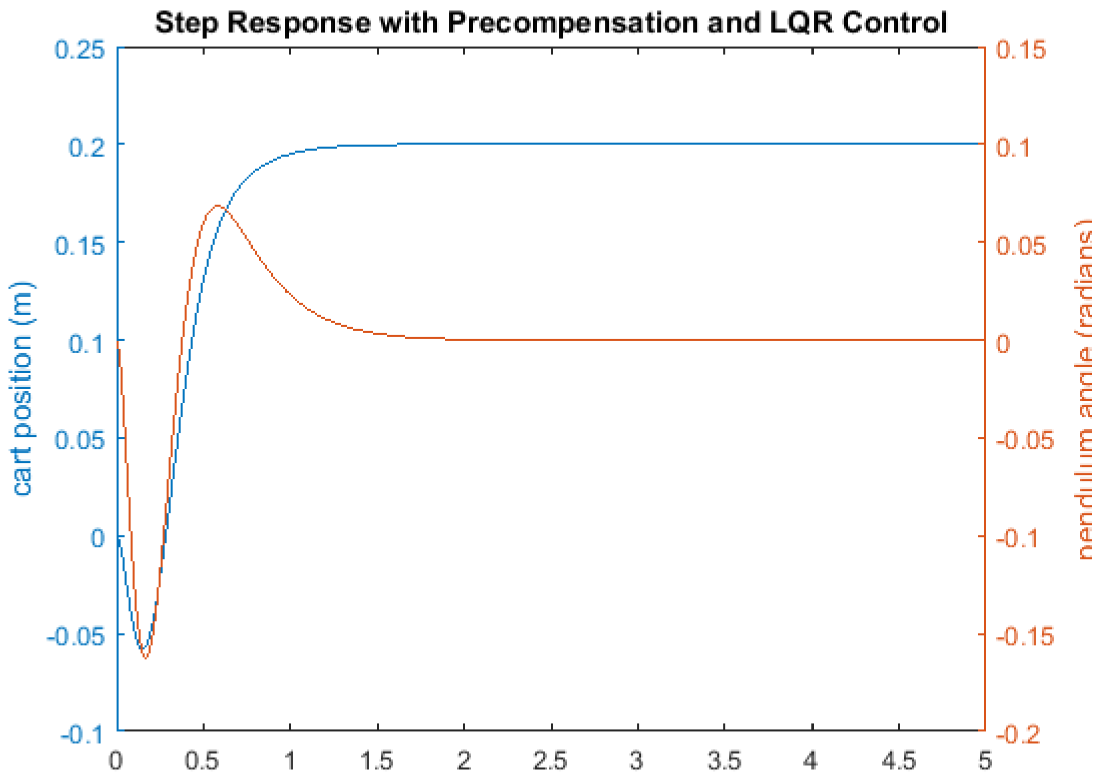
می‌توانیم فاکتور $\bar{N}$ را با استفاده از تابع تعریف شده rscale.m به دست آوریم (این تابع در سی‌دی موجود می‌باشد). ماتریس C به گونه‌ای اصلاح شده است که ورودی مرجع تنها موقعیت ارابه را دستور دهد.

```
Cn = [1 0 0 0];

sys_ss = ss(A,B,Cn,0);

Nbar = rscale(sys_ss,K)
```

```
Nbar =

  -70.7107
```

تابع rscale.m تابع استاندارد متلب نمی‌باشد و باید فایل آن را در شاخه کاری متلب قرار داده و از آن استفاده کنید. برای اطلاعات بیشتر به فصل **پیوست‌ها: rscale** مراجعه کنید. حال با استفاده از دستورات بالا و همچنین اضافه کردن دستورات زیر در ام‌فایل و اجرای آن، پاسخ پله‌ی سیستم به دست خواهد آمد:

```
sys_cl                                                              =
ss(Ac,Bc*Nbar,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with Precompensation and LQR Control')
```

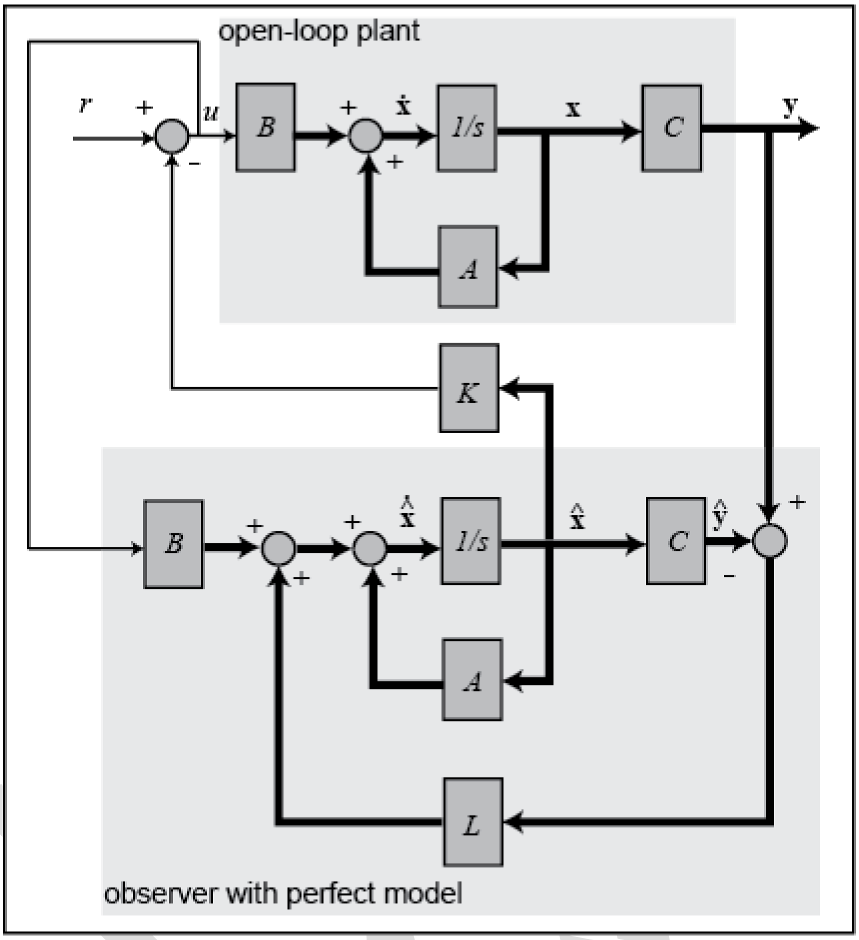


حال خطای حالت ماندگار به حد مطلوب رسیده و زمان نشست و نمو نیز قابل قبول بوده و فراجهش پاندول نیز در محدوده‌ی طراحی قرار دارد.

لازم به ذکر است که پیش جبران‌ساز $\bar{N}$ بر اساس مدل سیستم محاسبه شده و خارج از حلقه‌ی فیدبک قرار داده شده است. بنابراین اگر خطایی (یا اغتشاش نامعلومی) در مدل وجود داشته باشد، پیش جبران‌ساز قادر به اصلاح آن نبوده و خطای حالت ماندگار وجود خواهد داشت. می‌دانیم که اضافه کردن کنترل انتگرالی سبب حذف خطای حالت ماندگار حتی با وجود عدم قطعیت در مدل یا اغتشاش پله خواهد شد. مشکل استفاده از کنترل انتگرالی این است که ابتدا باید خطا تولید شده و سپس کنترلر آنرا اصلاح نماید که در نتیجه کاهش سرعت پاسخ سیستم را در پی دارد. از طرفی پیش جبران‌ساز به علت آگاهی از مدل سیستم، قابلیت پیش‌بینی خطای حالت ماندگار را دارد. یک راه حل مفید، ترکیب پیش جبران‌ساز و کنترل انتگرالی می‌باشد که مزایای هر دو روش را داراست.

کنترل مشاهده‌گر-محور

پاسخ به دست آمده در بالا خوب است، اما این نتیجه با فرض فیدبک تمام حالات به دست آمد که لزوما این فرض برقرار نمی‌باشد. در مواردی که تمامی متغیرهای حالت قابل اندازه‌گیری نمی‌باشند باید یک تخمین‌گر حالت طراحی شود. شماتیک کنترل فیدبک حالت با تخمین‌گر تمام حالات و بدون پیش جبران‌ساز $\bar{N}$ در شکل زیر نمایش داده شده است:



پیش از طراحی تخمین‌گر، باید از مشاهده‌پذیری سیستم خود اطمینان حاصل کنیم. مشاهده‌پذیری مشخص می‌کند که آیا می‌توان بر اساس خروجی‌های اندازه‌گیری شده، حالت‌های سیستم را تخمین زد یا خیر. مشابه روشی که برای کنترل‌پذیری انجام می‌شد، سیستمی مشاهده‌پذیر است که ماتریس مشاهده‌پذیری آن رنک کامل باشد. ماتریس مشاهده‌پذیری به شکل زیر تعریف می‌گردد:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{۴}$$

می‌توانیم از دستور obsv برای تشکیل ماتریس مشاهده‌پذیری و دستور rank برای محاسبه‌ی رنک آن استفاده نمود:

```
ob = obsv(sys_ss);

observability = rank(ob)
```

```
observability =

     4
```

از آنجایی که ماتریس مشاهده‌پذیری ۸x۴ بوده و دارای رنک ۴ می‌باشد در نتیجه این ماتریس رنک کامل بوده و سیستم ما مشاهده‌پذیر است. در این مثال ماتریس مشاهده‌پذیری مربعی نمی‌باشد زیرا سیستم ما دارای دو خروجی می‌باشد. باید به نکته اشاره کنیم که اگر تنها بتوانیم زاویه پاندول را اندازه‌گیری کنیم نمی‌توانیم تمامی حالت‌های سیستم را تخمین

بزنیم. برای بررسی این موضوع اگر از دستور obsv(A,C(2,:)) استفاده کنیم، ماتریس مشاهده‌پذیری دارای رنک کامل نمی‌باشد.

چون می‌دانیم که می‌توانیم حالت سیستم را تخمین بزنیم، روند طراحی تخمین‌گر حالت را توضیح می‌دهیم. با توجه به دیاگرام بالا، دینامیک تخمین حالات با معادله‌ی زیر تعریف می‌شود.

$$\dot{\hat{\boldsymbol{x}}} = A\hat{\boldsymbol{x}} + Bu + L(y - \hat{y}) \tag{۵}$$

ذات این معادله مشابه کنترل حلقه بسته می‌باشد که جمله‌ی آخر، اصلاحی بر اساس فیدبک انجام می‌دهد. یعنی جمله آخر تخمین حالت را بر اساس اختلاف بین خروجی واقعی $\boldsymbol{y}$ و خروجی تخمین زده شده $\hat{\boldsymbol{y}}$ اصلاح می‌کند. حال به دینامیک خطای تخمین حالت نگاه می‌اندازیم:

$$\dot{e} = \dot{x} - \dot{\hat{x}} = (Ax + Bu) - \left(A\hat{x} + Bu + L(Cx - C\hat{x})\right) \tag{۶}$$

پس دینامیک خطای تخمین حالت برابر است با:

$$\dot{e} = (A - LC)e \tag{۷}$$

اگر ماتریس $A - LC$ پایدار باشد (دارای مقادیر ویژه منفی)، خطا به صفر میل می‌کند ($\hat{\boldsymbol{x}}$ به $\boldsymbol{x}$ میل می‌کند). اگر از جنبه‌ی کنترلی نگاه کنیم، سرعت همگرایی به قطب‌های تخمین‌گر (مقادیر ویژه $A - LC$) بستگی دارد. به دلیل اینکه قصد استفاده از تخمین حالت به عنوان ورودی کنترلر را داریم، می‌خواهیم تخمین حالت سریع‌تر از کل سیستم حلقه بسته همگرا شود. یعنی ما می‌خواهیم قطب‌های مشاهده‌گر سریع‌تر از قطب‌های کنترلر باشد. یک روش رایج، تعیین قطب‌های تخمین‌گر ۴ تا ۱۰ برابر سریع‌تر از قطب‌های کنترلر می‌باشد. اگر نویز در اندازه‌گیری یا خطایی در سنسورهای اندازه‌گیری وجود داشته باشد، بسیار سریع بودن قطب‌های تخمین‌گر نیز مشکل ایجاد می‌کند.

بر این اساس ابتدا باید قطب‌های کنترلر را به دست آوریم. برای اینکار کد زیر را به انتهای ام‌فایل خود اضافه کنیم. اگر از ماتریس Q بروزرسانی شده استفاده کرده باشید، قطب‌های زیر در پنجره دستور متلب نمایش داده می‌شود:

```
poles = eig(Ac)
```

```
poles =

  -8.4910 + 7.9283i

  -8.4910 - 7.9283i

  -4.7592 + 0.8309i

  -4.7592 - 0.8309i
```

کندترین قطب‌ها دارای قسمت حقیقی برابر 4.7592- می‌باشند در نتیجه قطب‌های تخمین‌گر را در 40- قرار می‌دهیم. چون دینامیک تخمین‌گر حلقه بسته توسط ماتریس $(A - LC)$ که فرمی مشابه ماتریس دینامیک سیستم فیدبک حالت $(A - BK)$ دارد تعریف می‌شود از همان دستورها که برای به دست آوردن بهره‌ی فیدبک حالت K استفاده شده برای پیدا کردن بهره‌ی تخمین‌گر L استفاده می‌نماییم. به این دلیل که ترانهاده‌ی ماتریس $A - LC$ تغییری در مقادیر ویژه نداده و برابر $A' - C'L'$ می‌باشد که دقیقا مشابه فرم $A - BK$ است می‌توانیم از دستورات acker یا place استفاده نماییم. یادآوری می‌کنیم که دستور place نمی‌تواند قطب‌های با فاصله بزرگتر از ۱ را جایدهی کند در نتیجه قطب‌های مشاهده‌گر را به شکل زیر قرار می‌دهیم. دستورات زیر را به ام‌فایل خود اضافه کرده تا ماتریس L به شکل زیر در خروجی نمایش داده شود:

```
P = [-40 -41 -42 -43];
```

```
L = place(A',C',P)'
```

```
L =

   1.0e+03 *

    0.0826   -0.0010

    1.6992   -0.0402

   -0.0014    0.0832

   -0.0762    1.7604
```

از هر دو خروجی (زاویه پاندول و موقعیت ارابه) برای طراحی مشاهده‌گر استفاده نمودیم.

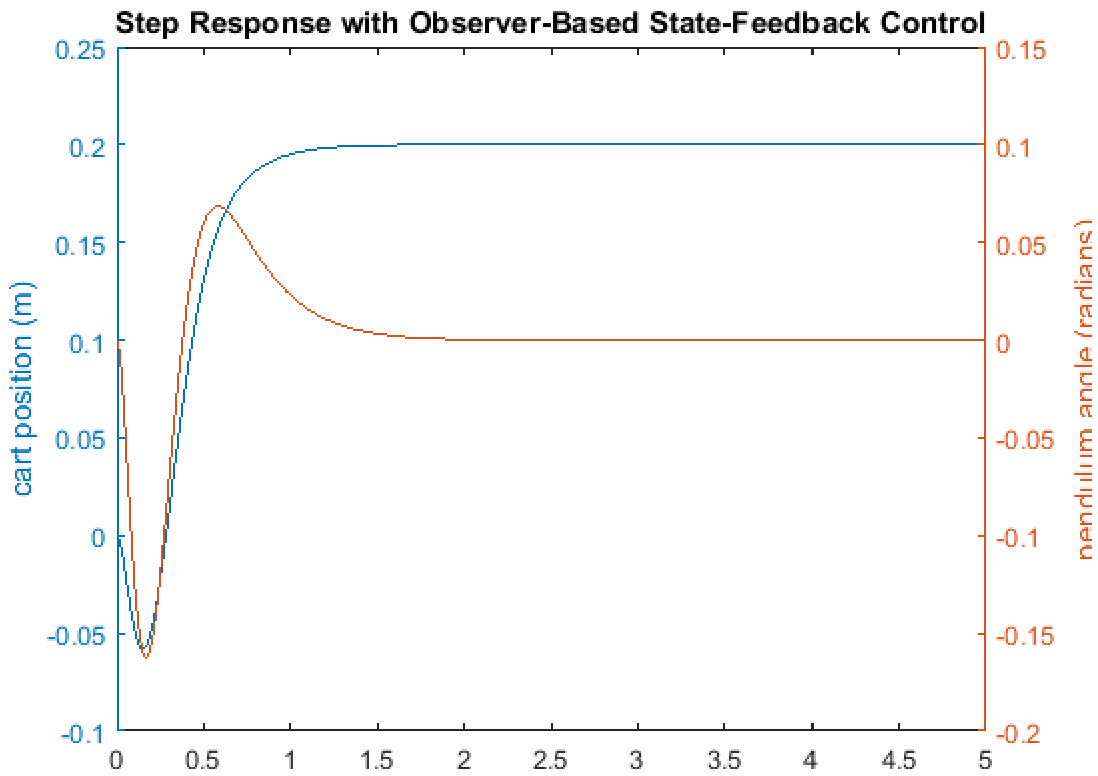
اکنون کنترلر فیدبک حالت پیشین را با تخمین‌گر حالت ترکیب کرده تا جبران‌ساز کامل به دست آید. سیستم حلقه بسته‌ی حاصل توسط معادلات ماتریسی زیر تعریف می‌شود:

$$\begin{bmatrix} \dot{x} \\ \dot{e} \end{bmatrix} = \begin{bmatrix} A-BK & BK \\ 0 & A-LC \end{bmatrix} \begin{bmatrix} x \\ e \end{bmatrix} + \begin{bmatrix} B\overline{N} \\ 0 \end{bmatrix} r \tag{۸}$$

$$y = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x \\ e \end{bmatrix} + [0]r \tag{۹}$$

سیستم حلقه بسته‌ی تعریف شده در بالا را می‌توان با اضافه کردن دستورات زیر به انتهای ام‌فایل خود در متلب پیاده‌سازی نمود. بعد از اجرای ام‌فایل، پاسخ پله به شکل زیر به دست می‌آید:

```
Ace = [(A-B*K) (B*K);

       zeros(size(A)) (A-L*C)];

Bce = [B*Nbar;

       zeros(size(B))];

Cce = [Cc zeros(size(Cc))];

Dce = [0;0];



states = {'x' 'x_dot' 'phi' 'phi_dot' 'e1' 'e2' 'e3' 'e4'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_est_cl                                                          =
ss(Ace,Bce,Cce,Dce,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r = 0.2*ones(size(t));
[y,t,x]=lsim(sys_est_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with Observer-Based State-Feedback Control')
```



این پاسخ تقریبا مشابه پاسخ به دست آمده با فرض دسترسی کامل به تمامی متغیرهای حالت می‌باشد. دلیل این امر این است که قطب‌های مشاهده‌گر سریع بوده و همچنین مدل فرض شده برای مشاهده‌گر مشابه مدل سیستم واقعی (با شرایط اولیه یکسان) می‌باشد. در نتیجه تمامی نیازهای طراحی ارضا شده و حداقل تلاش کنترلی صرف شده است و نیازی به اقدامات بیشتر نمی‌باشد.

این مثال نشان می‌دهد که برای کنترل سیستم‌های چند ورودی چند خروجی، استفاده از روش فضای حالت ساده‌تر از سایر روش‌ها که پیش از این گفته شد جواب می‌دهد.

بخش هفتم: طراحی کنترلر دیجیتال

فهرست مطالب بخش

- فضای حالت گسسته
- کنترل‌پذیری و مشاهده‌پذیری
- طراحی کنترل با استفاده از جایدهی قطب
- طراحی پیش جبران‌ساز
- طراحی مشاهده‌گر

دستورهای کلیدی متلب در این بخش:

```
ss , c2d , ctrb , obsv , dlqr , lsim , plotyy , eig , place
```

در نسخه کنترل دیجیتال سیستم پاندول معکوس، از روش فضای حالت برای طراحی کنترلر دیجیتال استفاده می‌نماییم. اگر به **بخش دوم: مدل‌سازی سیستم** مراجعه کنید، معادلات فضای حالت خطی شده به شکل زیر استخراج شده‌اند:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \dfrac{-(I+ml^2)b}{I(M+m)+Mml^2} & \dfrac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{-mlb}{I(M+m)+Mml^2} & \dfrac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \dfrac{ml}{I(M+m)+Mml^2} \end{bmatrix} u \qquad (۱)$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \qquad (۲)$$

که پارامترهای تعریف شده عبارتند از:

(M) جرم ارابه: 0.5 Kg

(m) جرم پاندول: 0.2 Kg

(b) ضریب اصطکاک ارابه: 0.1 N/m.sec

(l) فاصله از مرکز جرم پاندول: 0.3 m

(I) ممان اینرسی جرمی پاندول: 0.006 Kg.m^2

(F) نیروی وارد شده به ارابه

(x) موقعیت ارابه

(theta) زاویه پاندول از قائم

برای این مسئله، خروجی برابر موقعیت ارابه (x بر حسب متر) و زاویه پاندول (ϕ بر حسب رادیان) بوده که ϕ بیانگر انحراف موقعیت پاندول از حالت تعادل می‌باشد و داریم $\theta = \pi + \phi$.

نیازهای طراحی برای ورودی پله ۰/۲ متر برای موقعیت ارابه x به شرح زیر است:

- زمان نشست برای x و θ کمتر از ۵ ثانیه
- زمان نمو برای x کمتر از ۰/۵ ثانیه

- زاویه پاندول θ هیچگاه بیشتر از ۲۰ درجه (۰/۳۴ رادیان) از حالت عمودی منحرف نشود
- خطای حالت ماندگار برای x و θ کمتر از ۲٪

فضای حالت گسسته

قدم اول در طراحی کنترلر دیجیتال، تبدیل معادلات فضای حالت پیوسته به فرم گسسته می‌باشد. اینکار را با دستور c2d متلب انجام خواهیم داد. برای استفاده از این دستور، نیاز به تعیین سه آرگومان داریم: مدل سیستم پیوسته، زمان نمونه‌برداری (Ts بر حسب sec/sample) و روش 'method'. در حال حاضر با آرگومان اول یعنی ساختار ماتریس‌های $A.B.C$ و D در فرم فضای حالت آشنایی دارید.

برای انتخاب زمان نمونه‌برداری، باید در نظر داشت که فرکانس نمونه‌برداری نسبت به دینامیک سیستم سریع باشد. یکی از مشخصه‌های سرعت سیستم، پهنای باند حلقه بسته‌ی آن است. یکی از روش‌ها انتخاب فرکانس نمونه‌برداری به اندازه حداقل ۳۰ برابر بزرگتر از فرکانس پهنای باند حلقه بسته است که در دیاگرام بودی تعیین می‌شود.

فرض کنیم فرکانس پهنای باند حلقه بسته برای ارابه و پاندول حدود 1 rad/sec باشد. زمان نمونه‌برداری را برابر 1/100 sec/sample در نظر می گیریم. همچنین از روش گسسته‌سازی نگه‌داشتن مرتبه صفر ('zoh') استفاده خواهیم کرد. برای اطلاعات بیشتر به **فصل دوم - بخش هفتم: مقدمه‌ای بر طراحی کنترلر دیجیتال** مراجعه کنید. حال برای استفاده از تابع c2d آماده هستیم. دستورات زیر را در یک ام‌فایل وارد کنید. با اجرای این ام‌فایل، پنجره دستور متلب چهار ماتریس زیر را که بیانگر مدل فضای حالت گسسته می‌باشند نشان می‌دهد:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;



p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices

A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);

Ts = 1/100;
```

```
sys_d = c2d(sys_ss,Ts,'zoh')
```

```
sys_d =


  A =

                     x        x_dot          phi      phi_dot

   x                 1     0.009991    0.0001336    4.453e-07

   x_dot             0       0.9982      0.02672    0.0001336

   phi               0   -2.272e-05        1.002      0.01001

   phi_dot           0    -0.004544       0.3119        1.002


  B =

                    u

   x        9.086e-05

   x_dot      0.01817

   phi      0.0002272

   phi_dot    0.04544


  C =

              x     x_dot       phi   phi_dot

   x          1         0         0         0

   phi        0         0         1         0


  D =

         u

   x     0

   phi   0


Sample time: 0.01 seconds
```

```
Discrete-time state-space model.
```

حال مدل فضای حالت گسسته به فرم زیر است:

$$\begin{bmatrix} x(k+1) \\ \dot{x}(k+1) \\ \phi(k+1) \\ \dot{\phi}(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.01 & 0.0001 & 0 \\ 0 & 0.9982 & 0.0267 & 0.0001 \\ 0 & 0 & 1.0016 & 0.01 \\ 0 & -0.0045 & 0.3119 & 1.0016 \end{bmatrix} \begin{bmatrix} x(k) \\ \dot{x(k)} \\ \phi(k) \\ \dot{\phi(k)} \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.0182 \\ 0.0002 \\ 0.0454 \end{bmatrix} u \tag{۳}$$

$$y(k) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ \dot{x(k)} \\ \phi(k) \\ \dot{\phi(k)} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u(k) \tag{۴}$$

کنترل‌پذیری و مشاهده‌پذیری

قدم بعدی بررسی کنترل‌پذیری و مشاهده‌پذیری سیستم می‌باشد. برای اینکه یک سیستم کنترل‌پذیری کامل حالت را داشته باشد ماتریس کنترل‌پذیری یعنی ماتریس:

$$\mathcal{C} = [B \quad AB \quad A^2B \quad \cdots \quad A^{n-1}B] \tag{۵}$$

باید دارای رنک n باشد. رنک یک ماتریس تعداد سطرهای (یا ستون‌ها) مستقل خطی آن ماتریس می‌باشد. بر همین منوال، برای اینکه یک سیستم مشاهده‌پذیری کامل حالت را داشته باشد، ماتریس مشاهده‌پذیری یعنی

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{۶}$$

باید دارای رنک n باشد. بررسی کنترل‌پذیری و مشاهده‌پذیری برای کنترل در زمان پیوسته مشابه یکدیگر می‌باشند اما برای مدل فضای حالت گسسته اینطور نیست.

از آنجایی که تعداد متغیرهای حالت سیستم ما چهار عدد است، باید رنک هر دو ماتریس برابر ۴ باشد. دستور rank رنک هر ماتریس را محاسبه می‌کند. دستورات زیر را به امفایل خود اضافه کردن و آنرا در متلب اجرا کنید:

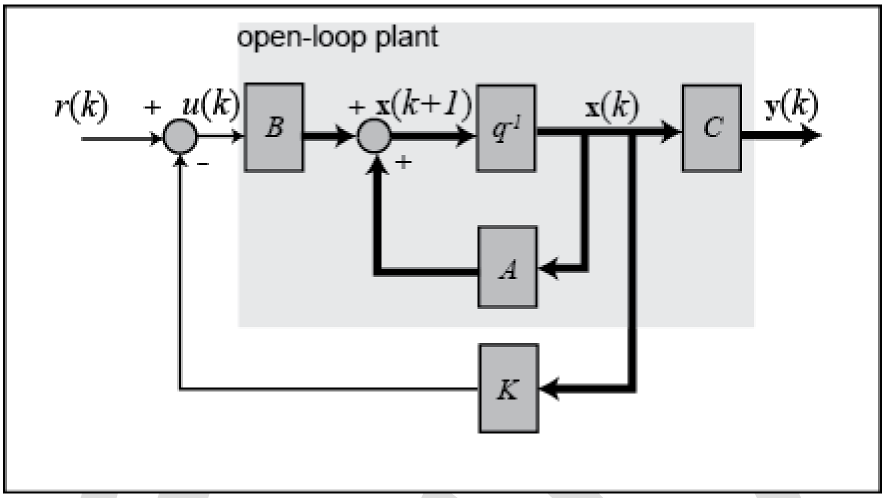
```
co = ctrb(sys_d);

ob = obsv(sys_d);



controllability = rank(co)

observability = rank(ob)
```

```
controllability =

     4

observability =

     4
```

بدین صورت ثابت می‌شود که سیستم گسسته کنترل‌پذیر و مشاهده‌پذیر می‌باشد.

طراحی کنترل با استفاده از جایدهی قطب

شماتیک سیستم کنترل فیدبک تمام حالت به شکل زیر است:



با فرض اینکه هر چهار متغیر حالت سیستم قابل اندازه‌گیری می‌باشند ماتریس بهره‌ی کنترل K را طراحی می‌کنیم. اگر به **بخش ششم: طراحی کنترلر در فضای حالت** مراجعه کنید، از روش رگولاسیون درجه دوم خطی (LQR) برای پیدا کردن ماتریس بهره‌ی کنترل K استفاده شد. در حالت دیجیتال از همان خطاها و تلاش کنترلی استفاده می‌کنیم. برای جزییات بیشتر به کتب مرجع مراجعه کنید. برای استفاده از روش LQR باید دو پارامتر را مشخص کنیم، یکی ماتریس شاخص عملکرد R و دیگری ماتریس هزینه‌ی حالت Q. برای سادگی در شروع ماتریس شاخص عملکرد R را برابر ۱ و ماتریس هزینه حالت Q را برابر $C'C$ در نظر می‌گیریم. سپس وزن‌های نسبی این دو ماتریس را با سعی و خطا تنظیم می‌نماییم. ماتریس هزینه‌ی حالت Q ساختار زیر را دارد:

$$Q = C'C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad (۷)$$

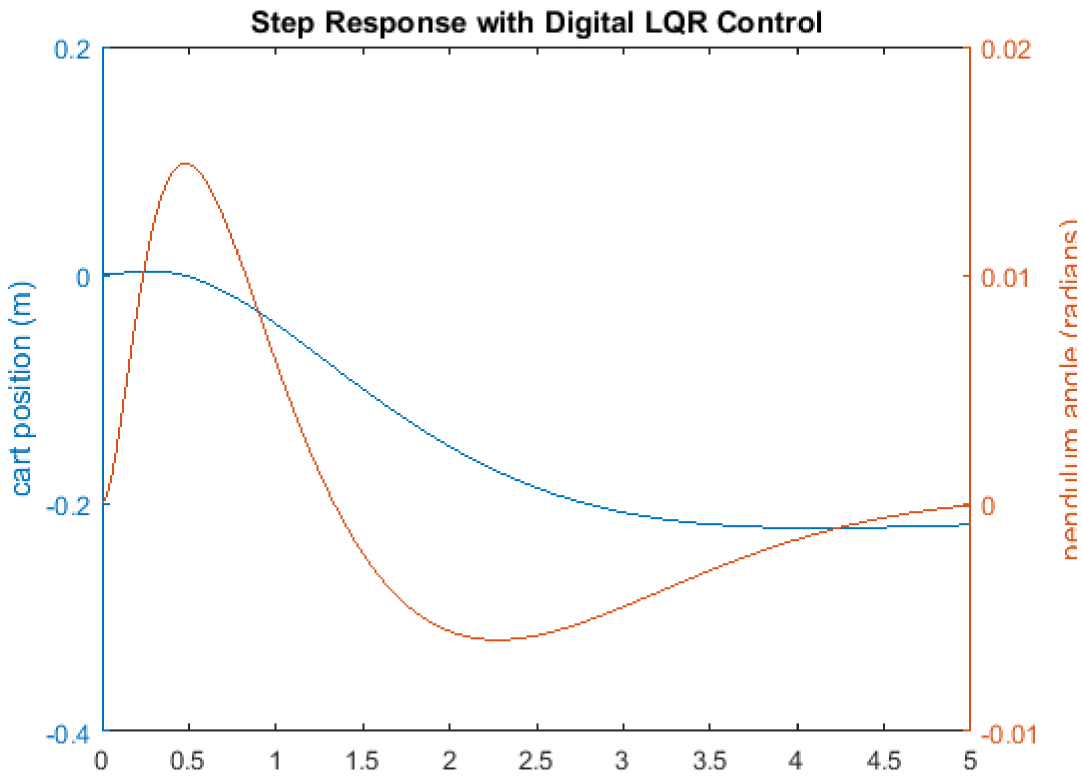
المانی که در موقعیت (۱،۱) ماتریس Q قرار دارد وزن موقعیت ارابه و المانی که در موقعیت (۳،۳) قرار دارد بیانگر وزن زاویه‌ی پاندول می‌باشد.

حال برای محاسبه‌ی ماتریس بهره‌ی کنترل K و مشاهده‌ی پاسخ حلقه بسته‌ی سیستم آماده هستیم. چون هدف ما طراحی کنترلر دیجیتال است از تابع متلب dlqr استفاده می‌کنیم. دستور زیر را به امفایل خود اضافه کرده و آنرا اجرا کنید. لازم به ذکر است که در کد زیر مقادیر ماتریس‌های فضای حالت A، B، C و D با معادل‌های گسسته آنها که در قبل از دستور c2d به دست آمد جایگزین شده است:

```
A = sys_d.a;

B = sys_d.b;

C = sys_d.c;

D = sys_d.d;

Q = C'*C

R = 1;
```

```
[K] = dlqr(A,B,Q,R)



Ac = [(A-B*K)];

Bc = [B];

Cc = [C];

Dc = [D];



states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl                                                              =
ss(Ac,Bc,Cc,Dc,Ts,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with Digital LQR Control')
```

```
Q =

     1     0     0     0

     0     0     0     0

     0     0     1     0

     0     0     0     0

K =

   -0.9384   -1.5656   18.0351    3.3368
```

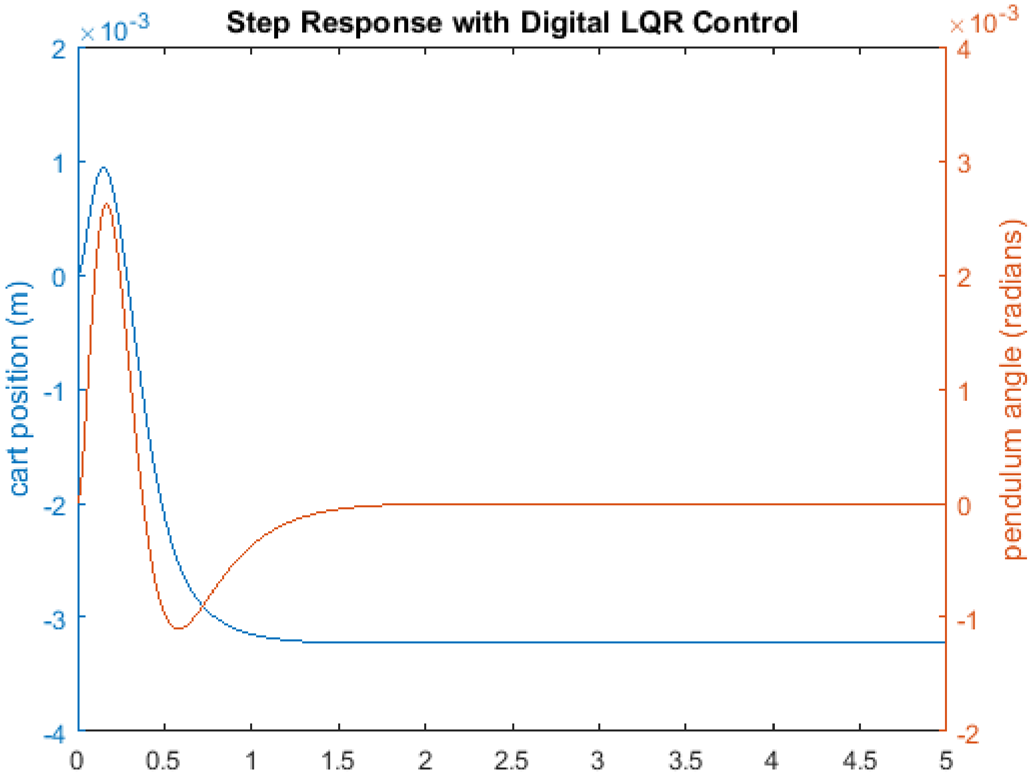


منحنی قرمز رنگ معرف زاویه‌ی پاندول به رادیان و منحنی آبی رنگ معرف موقعیت ارابه به متر می‌باشد. همانطور که مشاهده می‌کنید این منحنی رضایت‌بخش نمی‌باشد. فراجهش پاندول و ارابه مناسب می‌باشد اما باید زمان نشست آنها بهبود و زمان نمو ارابه کاهش یابد. همچنین ممکن است متوجه شده باشید که موقعیت نهایی ارابه حتی در نزدیکی موقعیت مطلوب نیست و در واقع در جهت خلاف است. به این خطا در بخش بعد پرداخته و در این بخش بر روی زمان نشست و نمو تمرکز می‌کنیم. به امفایل خود بازگشته و ماتریس Q را تغییر دهید تا پاسخ بهتری به دست آورید. متوجه خواهیم شد که با افزایش المان (۱،۱) و (۳،۳) زمان نشست و نمو کاهش می‌یابد و زاویه حرکت پاندول کاسته می‌شود. به بیان دیگر با اینکار شما در تابع هزینه وزن بیشتری را بر روی خطا نسبت به سیگنال کنترلی قرار می‌دهید. در امفایل خود المان (۱،۱) را برابر ۵۰۰۰ و المان (۳،۳) را برابر ۱۰۰ قرار دهید تا K و پاسخ زیر به دست آید.

```
A = sys_d.a;

B = sys_d.b;

C = sys_d.c;

D = sys_d.d;

Q = C'*C;

Q(1,1) = 5000;

Q(3,3) = 100

R = 1;

[K] = dlqr(A,B,Q,R)
```

```
Ac = [(A-B*K)];

Bc = [B];

Cc = [C];

Dc = [D];



states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_cl                                                                                    =
ss(Ac,Bc,Cc,Dc,Ts,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with Digital LQR Control')
```

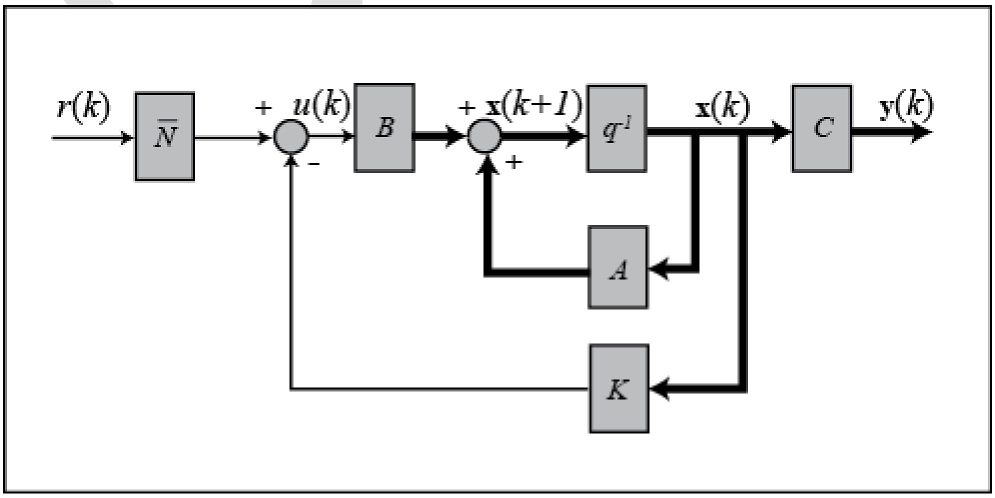
```
Q =

        5000           0           0           0

           0           0           0           0

           0           0         100           0

           0           0           0           0

K =

  -61.9933  -33.5040   95.0597   18.8300
```



از این نمودار متوجه می‌شویم تمامی نیازهای طراحی به جز خطای حالت ماندگار موقعیت ارابه x ارضا شده‌اند. این مشکل را به راحتی با تعریف ضریب مقیاس پیشخور $\bar{N}$ مرتفع می‌کنیم.

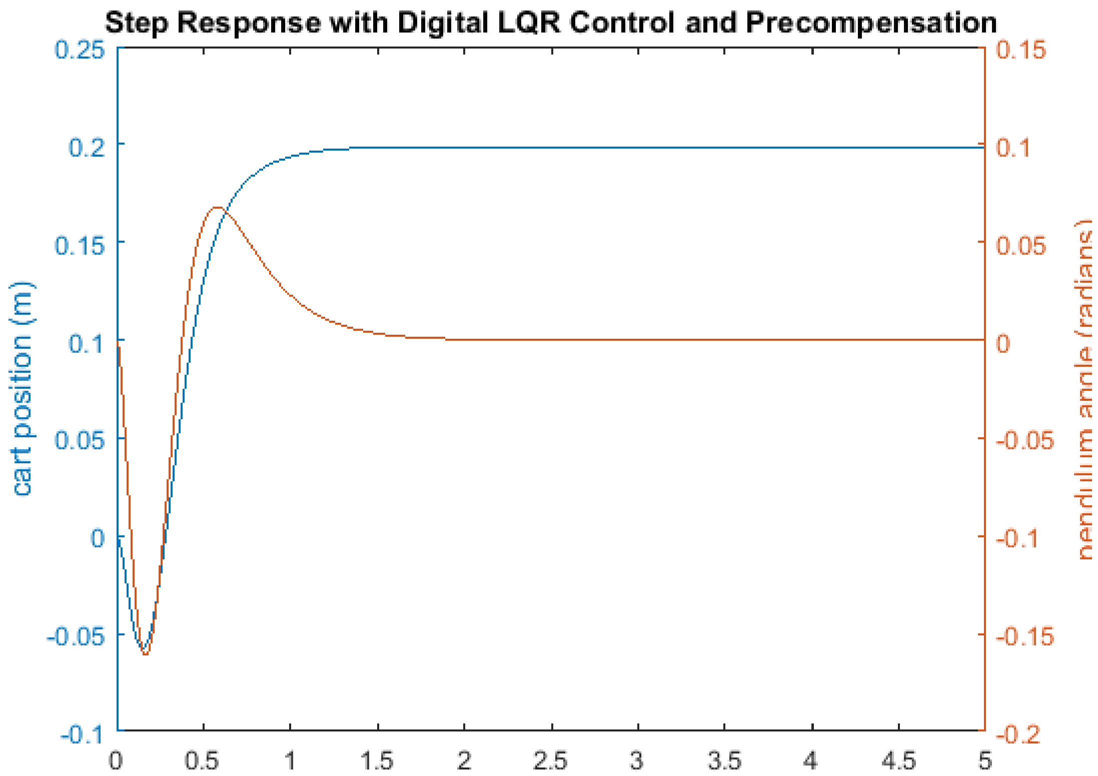
طراحی پیش جبران‌ساز

برخلاف سایر روش‌های طراحی، سیستم فیدبک تمام حالت خروجی را مستقیما با ورودی مرجع مقایسه نمی‌کند، بلکه حاصل ضرب بردار حالت در ماتریس کنترل (Kx) را با ورودی مرجع مقایسه می‌کند (شماتیک بخش قبل را مشاهده کنید). بنابراین نباید توقع داشت که خروجی به ورودی مرجع همگرا شود. برای به دست آوردن خروجی مطلوب، باید ورودی مرجع را بزرگنمایی کنیم تا خروجی با ورودی مرجع برابر شود. برای اینکار از ضریب پیشخور مقیاس $\bar{N}$ استفاده می‌کنیم. شماتیک کلی سیستم فیدبک تمام حالت با ضریب مقیاس به شکل زیر است:



متاسفانه در اینجا نمی‌توانیم از تابع rscale برای به دست آوردن $\bar{N}$ استفاده کنیم زیرا این تابع برای سیستم‌های تک خروجی پیوسته در زمان تعریف شده است. اما می‌توانیم ضریب مقیاس را با سعی و خطا پیدا کنیم. بعد از چندبار تلاش، مقدار $\bar{N}$ برابر 61.55- پاسخ رضایت‌بخشی را به دست می‌دهد. کد زیر را به امفایل خود اضافه کنید و با اجرای آن پاسخ سیستم را به دست آورید:

```
Nbar = -61.55;

sys_cl                                                                    =
ss(Ac,Bc*Nbar,Cc,Dc,Ts,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with Digital LQR Control and Precompensation')
```



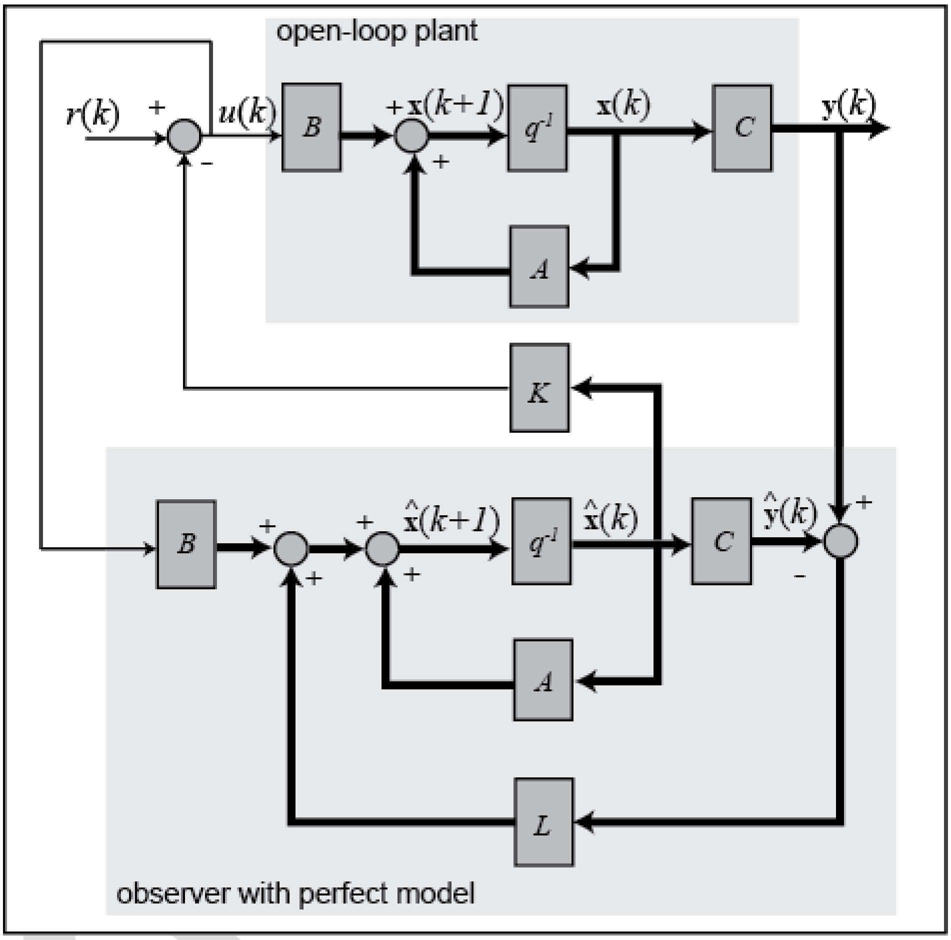
متوجه می‌شویم که خطای حالت ماندگار موقعیت ارابه از بین رفته است. اکنون سیستمی را طراحی کردیم که تمامی نیازهای طراحی را ارضا می‌کند. لازم به ذکر است که ضریب مقیاس $\bar{N}$ بر اساس مدل سیستم طراحی شده است. اگر خطایی در مدل یا اغتشاش نامعلومی به سیستم وارد شود، خطای حالت ماندگار دیگر به صفر نمی‌رسد.

طراحی مشاهده‌گر

پاسخ به دست آمده در بالا تمامی نیازهای طراحی را برآورده می‌کند هرچند که این نتیجه با فرض قابل اندازه‌گیری بودن تمامی متغیرهای حالت سیستم به دست آمده است. این فرض ممکن است برای تمامی سیستم‌ها صادق نباشد. در این قسمت روشی را برای تخمین حالت‌های سیستم بر اساس اندازه‌گیری خروجی و مدل سیستم معرفی می‌کنیم. ابزاری که حالت سیستم را تخمین می‌زند تخمین‌گر نامیده می‌شود. پس در این بخش یک مشاهده‌گر حالت درجه کامل برای تخمین

تمامی متغیرهای حالت سیستم (شامل آنهایی که قابل اندازه‌گیری هستند) طراحی می‌شود. برای توضیحات بیشتر در خصوص طرز کار مشاهده‌گر، به کتب مرجع رجوع کنید).

شماتیک کلی سیستم فیدبک حالت مشاهده‌گر-محور در شکل زیر نشان داده شده است:



طراحی مشاهده‌گر چیزی جز پیدا کردن ماتریس بهره‌ی L مشاهده‌گر نمی‌باشد. برای اینکار ابتدا باید قطب‌های سیستم حلقه بسته‌ی بدون مشاهده‌گر (مقادیر ویژه $A-BK$) را به دست آوریم. این مقادیر را با استفاده از دستور متلب eig به شکل زیر حساب می‌کنیم:

```
poles = eig(A-B*K)
```

```
poles =

   0.9157 + 0.0728i

   0.9157 - 0.0728i

   0.9535 + 0.0079i

   0.9535 - 0.0079i
```

از آنجایی که مشاهده‌گر سعی بر تخمین مقادیر متغیرهای حالت که خود تغییر می‌کنند را دارد، مطلوبست دینامیک مشاهده‌گر سریعتر از دینامیک سیستم حلقه بسته‌ی بدون مشاهده‌گر باشد. یک روش رایج انتخاب قطب‌های تخمین‌گر (مقادیر ویژه $A-LC$) ۴ تا ۱۰ برابر سریع‌تر از کندترین قطب کنترلر (مقادیر ویژه $A-BK$) می‌باشد. انتخاب قطب‌های

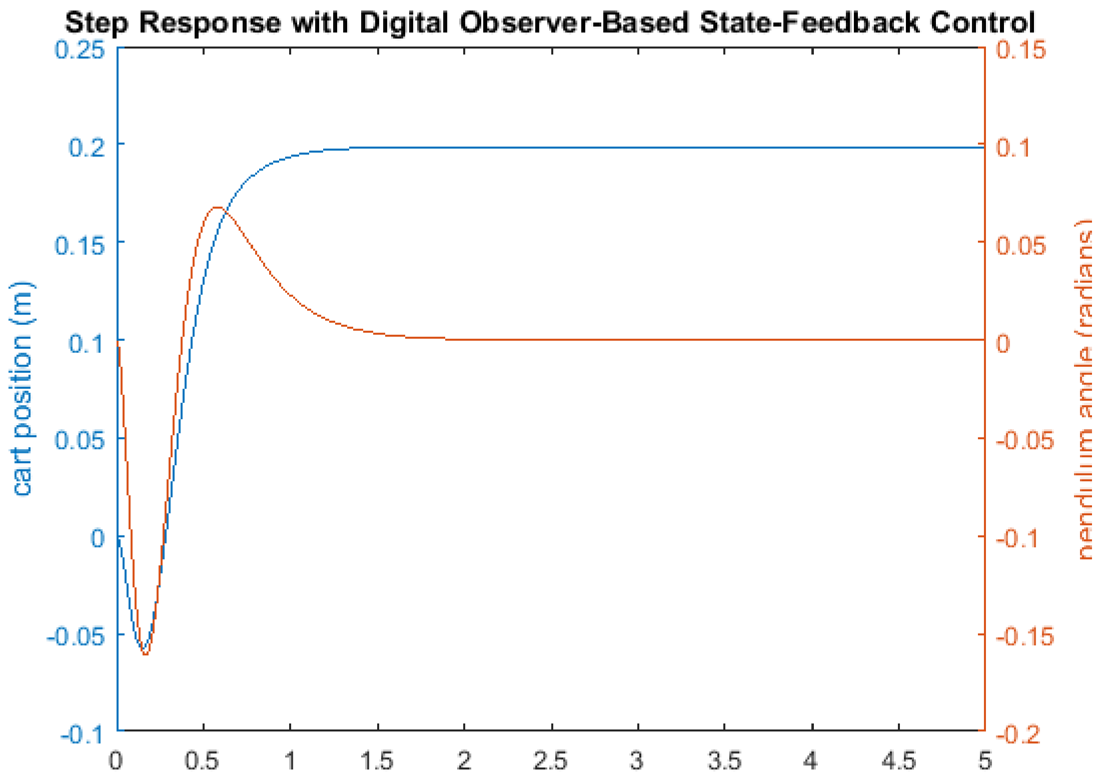
خیلی سریع برای تخمین‌گر در صورت وجود نویز یا خطا در سنسور اندازه‌گیری می‌تواند مشکل آفرین باشد. بر اساس قطب‌های به دست آمده در بالا، قطب‌های مشاهده‌گر را در [0.23- 0.22- 0.21- 0.2-] انتخاب می‌کنیم. این قطب‌ها را می‌توان در صورت نیاز بعدا اصلاح نمود. از دستور place متلب برای پیدا کردن ماتریس L استفاده می‌نماییم. کد زیر را به امفایل اضافه کرده و آنرا اجرا کنید تا ماتریس بهره‌ی مشاهده‌گر به دست آید:

```
P = [-0.2 -0.21 -0.22 -0.23];

L = place(A',C',P)'
```

```
L =

    2.4308   -0.0104

  147.6324   -1.2418

   -0.0131    2.4305

   -1.8079  147.9057
```

حال پاسخ کلی سیستم به همراه مشاهده‌گر را به دست می‌آوریم. دستورات زیر را به امفایل خود اضافه کرده و آنرا اجرا کنید تا پاسخ محاسبه شود:

```
Ace = [(A-B*K) (B*K);

       zeros(size(A)) (A-L*C)];

Bce = [B*Nbar;

       zeros(size(B))];

Cce = [Cc zeros(size(Cc))];

Dce = [0;0];



states = {'x' 'x_dot' 'phi' 'phi_dot' 'e1' 'e2' 'e3' 'e4'};
inputs = {'r'};
outputs = {'x'; 'phi'};

sys_est_cl                                                              =
ss(Ace,Bce,Cce,Dce,Ts,'statename',states,'inputname',inputs,'outputname',outputs);

t = 0:0.01:5;
r = 0.2*ones(size(t));
[y,t,x]=lsim(sys_est_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with Digital Observer-Based State-Feedback Control')
```



این پاسخ تقریبا مشابه پاسخ به دست آمده در حالتی که فرض بر دسترسی کامل به تمامی متغیرهای حالت داشتیم است. دلیل آن اینست که قطب‌های مشاهده‌گر بسیار سریع بوده و همچنین مدل فرض شده برای مشاهده‌گر مشابه مدل سیستم واقعی (با شرایط اولیه یکسان) است. بنابراین تمامی نیازهای طراحی با حداقل تلاش کنترلی برآورده شده است. در نتیجه اقدامات بیشتری نیاز نمی‌باشد.

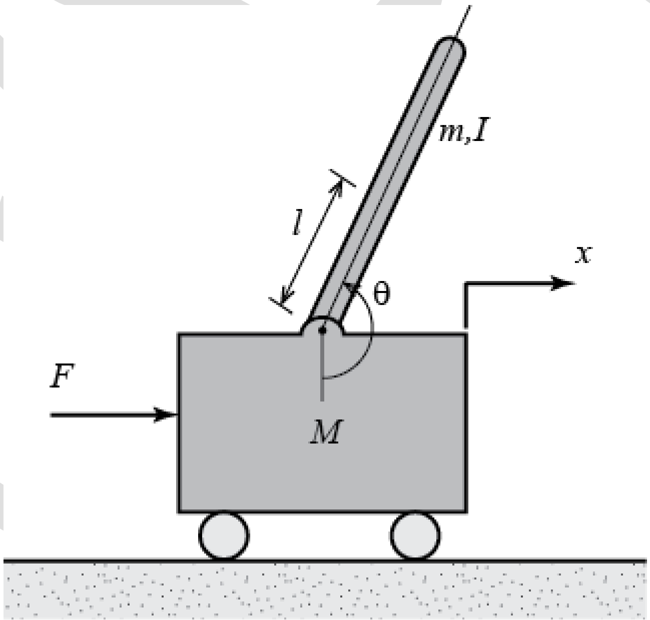
بخش هشتم: مدل‌سازی سیمولینک

فهرست مطالب بخش

در این فصل به روش ساخت مدل سیستم پاندول معکوس برای شبیه‌سازی در سیمولینک و افزونه‌های آن اشاره می‌کنیم. همانطور که نشان خواهیم داد مزیت بزرگ شبیه‌سازی این است که می‌تواند پاسخ‌های عددی برای معادلات غیر خطی به دست آورد که با استفاده از راه‌حل‌های با فرم بسته قادر به اینکار نمی‌باشیم. سپس می‌توان شبیه‌سازی غیر خطی را برای آزمون اعتبار مدل خطی شده استفاده کرد. همچنین می‌توان از مدل شبیه‌سازی برای ارزیابی عملکرد کنترل طراحی شده بر اساس مدل خطی استفاده نمود.

سیستم فیزیکی و معادلات آن

در این مثال یک سیستم پاندول معکوس به همراه ارابه دو بعدی را در نظر می‌گیریم که در آن پاندول به حرکت در صفحه قائم مقید شده است. برای این سیستم ورودی کنترلی، نیروی F است که ارابه را در راستای افقی حرکت می‌دهد و خروجی‌های سیستم موقعیت زاویه‌ای پاندول θ و موقعیت افقی ارابه x می‌باشند.



برای این مثال مقادیر زیر را در نظر می‌گیریم:

(M) جرم ارابه: 0.5 Kg

(m) جرم پاندول: 0.2 Kg

(b) ضریب اصطکاک ارابه: 0.1 N/m.sec

(l) فاصله از مرکز جرم پاندول: 0.3 m

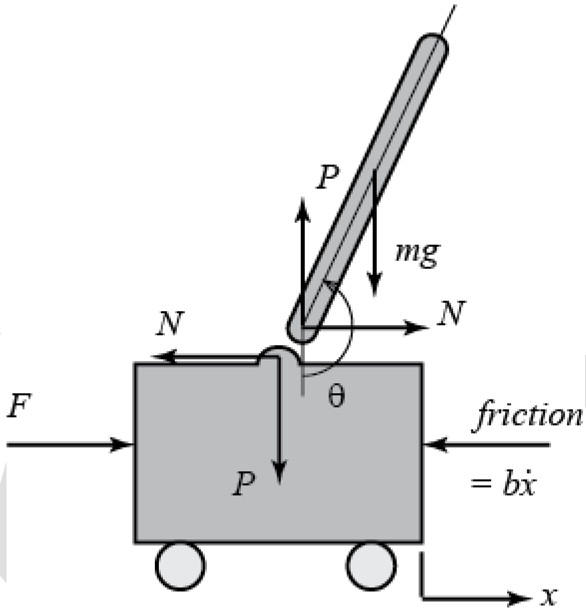
(I) ممان اینرسی جرمی پاندول: 0.006 Kg.m^2

(F) نیروی وارد شده به ارابه

(x) موقعیت ارابه

(theta) زاویه پاندول از قائم

در شکل زیر دیاگرام جسم آزاد سیستم رسم شده است:



مدل‌سازی این سیستم در سیمولینک به دلیل وجود قیدهای فیزیکی (مفصل پین شده) بین ارابه و پاندول که سبب کاهش درجه آزادی سیستم می‌شود دشوار است. هم ارابه و هم پاندول دارای یک درجه آزادی می‌باشند (به ترتیب x و θ). ما با استفاده از قانون دوم نیوتن ($F = ma$) معادلات دیفرانسیل مربوط به این درجات آزادی را به شکل زیر به دست می‌آوریم:

$$\ddot{x} = \frac{1}{M}\sum_{cart} F_x = \frac{1}{M}(F - N - b\dot{x}) \tag{۱}$$

$$\ddot{\theta} = \frac{1}{I}\sum_{pend} \tau = \frac{1}{I}(-Nlcos\theta - Plsin\theta) \tag{۲}$$

برای مدل‌سازی کامل دینامیک سیستم لازم است که نیروهای عکس‌العمل N و P بین ارابه و پاندول در نظر گرفته شوند. گنجاندن این نیروها در معادلات، علاوه بر مدل‌سازی دینامیک دورانی پاندول، نیاز به مدل‌سازی مولفه‌های ناشی از جابجایی مرکز جرم پاندول در راستای x و y را دارد. در **بخش دوم: مدل‌سازی سیستم**، نیروهای عکس‌العمل N و P به طور جبری به دست آورده شده‌اند.

به طور کلی می‌خواهیم از قدرت مدل‌سازی سیمولینک برای انجام محاسبات جبری مسئله استفاده کنیم. بنابراین معادله‌ی مولفه‌های x و y پاندول را به شکل زیر مدل‌سازی می‌کنیم:

$$m\ddot{x}_p = \sum_{pend} F_x = N \tag{۳}$$

$$\Rightarrow N = m\ddot{x}_p \tag{۴}$$

$$m\ddot{y}_p = \sum_{pend} F_y = P - mg \tag{۵}$$

$$\Rightarrow P = m(\ddot{y}_p + g) \qquad (۶)$$

هرچند که موقعیت در مختصات x_p و y_p توابع دقیق از θ می‌باشند. بنابراین می‌توانیم مشتقات آنها را بر حسب مشتقات θ نمایش دهیم. ابتدا برای معادلات مولفه x به معادله زیر دست می‌یابیم:

$$x_p = x + lsin\theta \qquad (۷)$$

$$\dot{x}_p = \dot{x} + l\dot{\theta}cos\theta \qquad (۸)$$

$$\ddot{x}_p = \ddot{x} - l\dot{\theta}^2 sin\theta + l\ddot{\theta}cos\theta \qquad (۹)$$

سپس برای معادلات مولفه‌ی y داریم:

$$y_p = -lcos\theta \qquad (۱۰)$$

$$\dot{y}_p = l\dot{\theta}sin\theta \qquad (۱۱)$$

$$\ddot{y}_p = l\dot{\theta}^2 cos\theta + l\ddot{\theta}sin\theta \qquad (۱۲)$$

سپس می‌توان این معادلات را بر حسب N و P نمایش داد:

$$N = m(\ddot{x} - l\dot{\theta}^2 sin\theta + l\ddot{\theta}cos\theta) \qquad (۱۳)$$

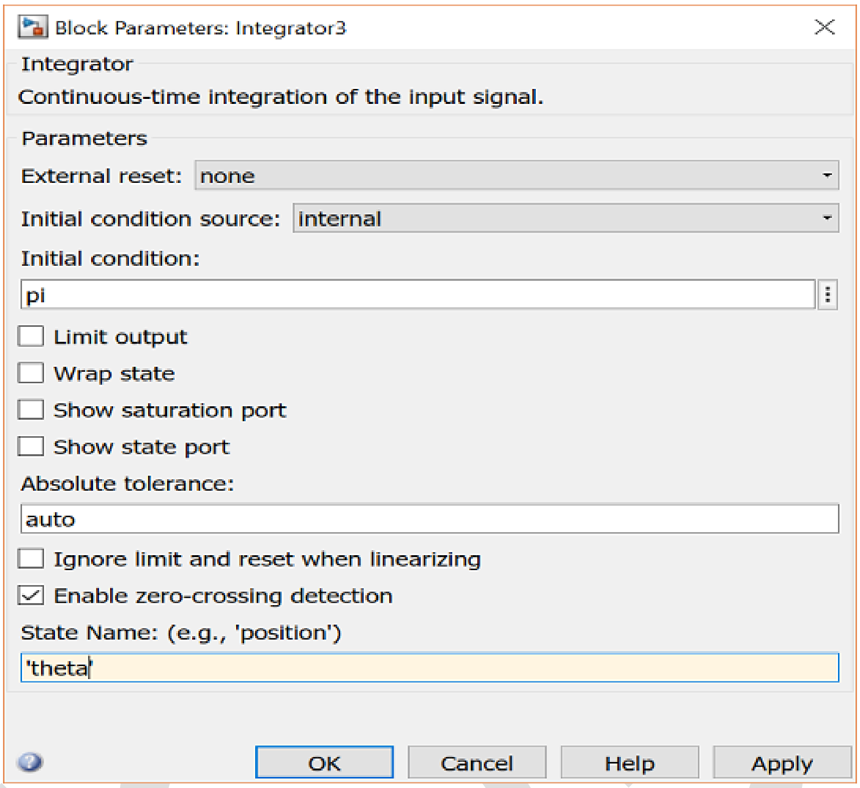
$$P = m(l\dot{\theta}^2 cos\theta + l\ddot{\theta}sin\theta + g) \qquad (۱۴)$$

حال می‌توانیم این معادلات را در سیمولینک نمایش دهیم. به دلیل اینکه سیمولینک قادر به کارکردن مستقیم با معادلات غیر خطی می‌باشد، خطی‌سازی این معادلات (مانند کاری که در **بخش دوم: مدل‌سازی سیستم** انجام شد) نیاز نمی‌باشد.

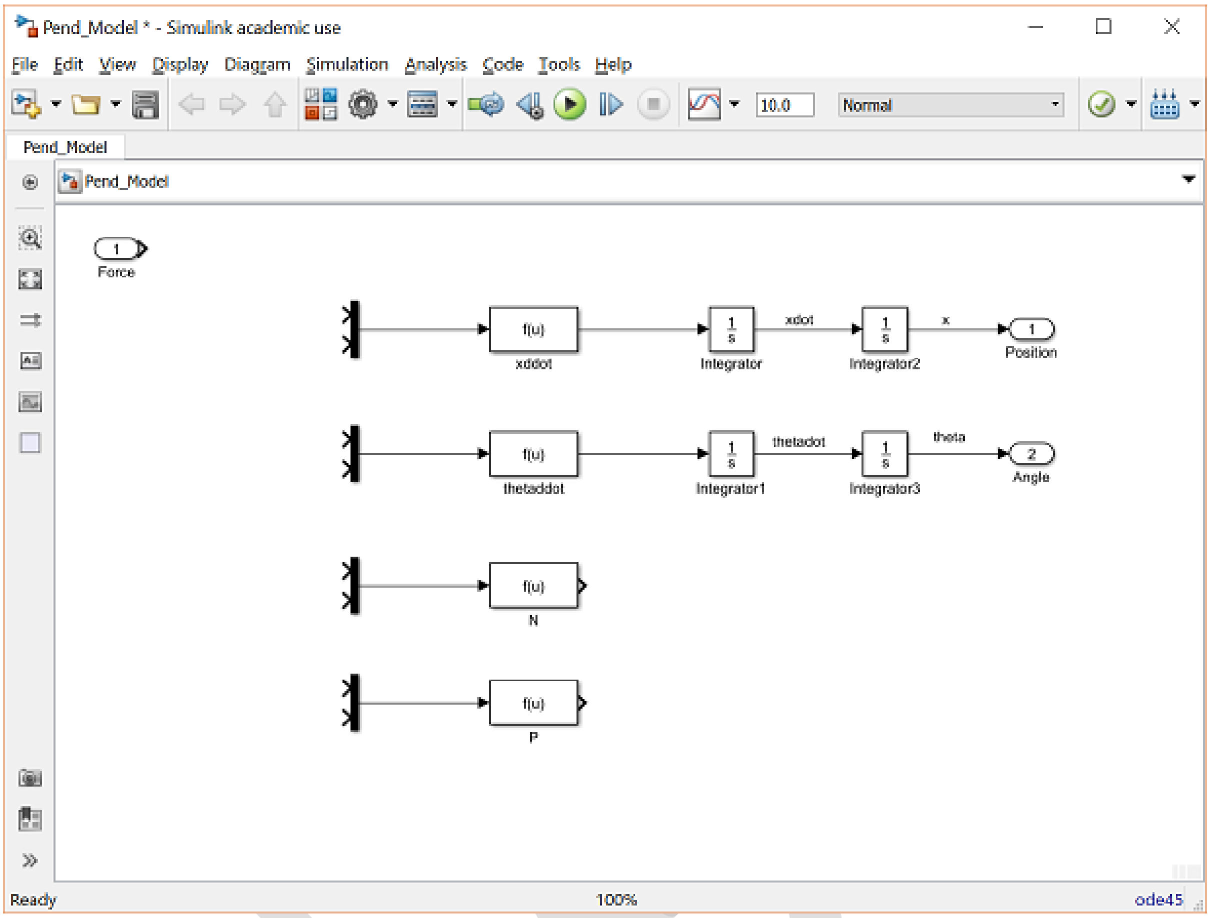
ساخت مدل غیر خطی در سیمولینک

با استفاده از معادلات به دست آمده در بالا می‌توانیم مدل پاندول معکوس را به شرح زیر در سیمولینک ایجاد کنیم.

- با تایپ دستور Simulink در متلب شروع کرده تا محیط سیمولینک باز شود. با انتخاب New > Simulink > Blank Model یا فشردن کلید Ctrl+N یک مدل جدید سیمولینک باز کنید.
- چهار بلوک Fcn از کتابخانه User-Defined Functions در مدل قرار دهید. با استفاده از این بلوک‌ها معادلات $\dot{x}_d$، $\dot{\theta}$، P و N را تشکیل می‌دهیم.
- نام هر یک از بلوک‌های Fcn را متناسب با تابع آن تغییر دهید.
- چهار بلوک انتگرال‌گیر از کتابخانه Continuous در مدل قرار دهید. خروجی هر یک از این بلوک‌های انتگرال‌گیر یکی از متغیرهای حالت سیستم یعنی x، $\dot{x}$، θ و $\dot{\theta}$ می‌باشند.
- بر روی هر یک از بلوک‌های انتگرال‌گیر دابل کلیک کرده و State Name: را متناسب با متغیر حالت اضافه کنید. برای مثال شکل زیر را نگاه کنید. همچنین فیلد Initial Condition: را برای θ (زاویه پاندول) به "pi" که نشان‌دهنده موقعیت اولیه عمودی بالا می‌باشد تغییر دهید.



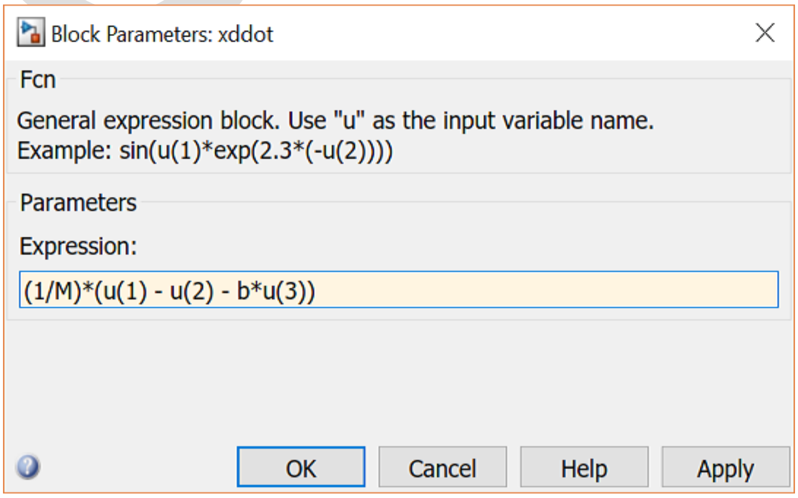
- چهار بلوک Multiplexer (Mux) از کتابخانه Signal Routing قرار دهید که هرکدام برای یک بلوک Fcn می‌باشد.
- دو بلوک Out1 و یک بلوک In1 از کتابخانه‌های Sinks و Sources به مدل اضافه کنید. سپس بر روی لیبل هر کدام از آنها دابل کلیک کرده و نام آنرا تغییر دهید. دو خروجی ”Position“ موقعیت ارابه و ”Angle“ زاویه پاندول بوده و یک ورودی ”Force“ نیروی وارد به ارابه است.
- خروجی هر بلوک Mux را به ورودی بلوک Fcn مربوطه متصل کنید.
- خروجی بلوک‌های Fcn مربوط به $\dot{x}_d$ و $\dot{\theta}$ را به دو انتگرال‌گیر متوالی وصل کنید تا موقعیت ارابه و زاویه پاندول ایجاد شود. مدل شما باید به شکل زیر درآمده باشد.



حال چهار معادله‌ی (۱)، (۲)، (۱۳) و (۱۴) را درون هر یک از بلوک‌های Fcn وارد می‌کنیم. از معادله (۱) شروع می‌کنیم که در پایین دوباره آورده شده است:

$$\ddot{x} = \frac{1}{M}\sum_{cart} F_x = \frac{1}{M}(F - N - b\dot{x}) \qquad (۱۵)$$

- این معادله نیازمند سه ورودی می‌باشد: $u(1) = F$، $u(2) = N$ و $u(3) = \dot{x}$. بر روی بلوک Mux مربوطه دابل کلیک کرده و Number of inputs: را برابر "3" قرار دهید.
- این سه ورودی را به ترتیب گفته شده در قدم قبل، به این بلوک Mux متصل کنید.
- بر روی بلوک Fcn اول دابل کلیک کرده و معادله xddot را مانند زیر وارد کنید.



حال معادله (۲) را که در زیر دوباره آورده شده است وارد می‌کنیم:

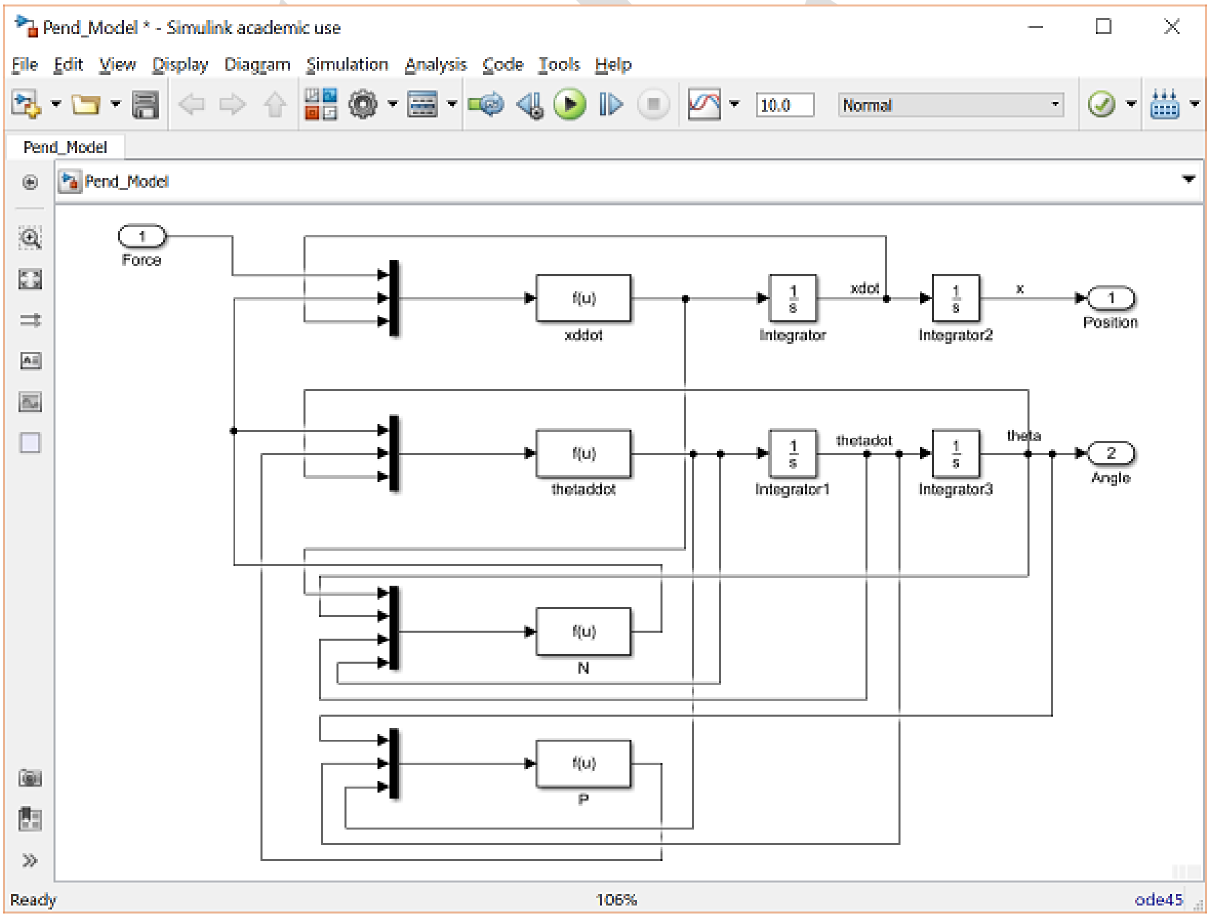
$$\ddot{\theta} = \frac{1}{I}\sum_{pend}\tau = \frac{1}{I}(-Nlcos\theta - Plsin\theta) \qquad (۱۶)$$

- این معادله نیازمند ۳ ورودی می‌باشد: $u(1) = N. u(2) = P$ و $u(3) = \theta$
- معادله بالا را در بلوک Fcn وارد کرده و تعداد ورودی‌های بلوک Mux را تغییر داده و هرکدام را با ترتیب صحیح به سیگنال مربوطه متصل کنید.
- این فرآیند را برای معادلات (۱۳) و (۱۴) که در زیر دوباره آورده شده است تکرار کنید:

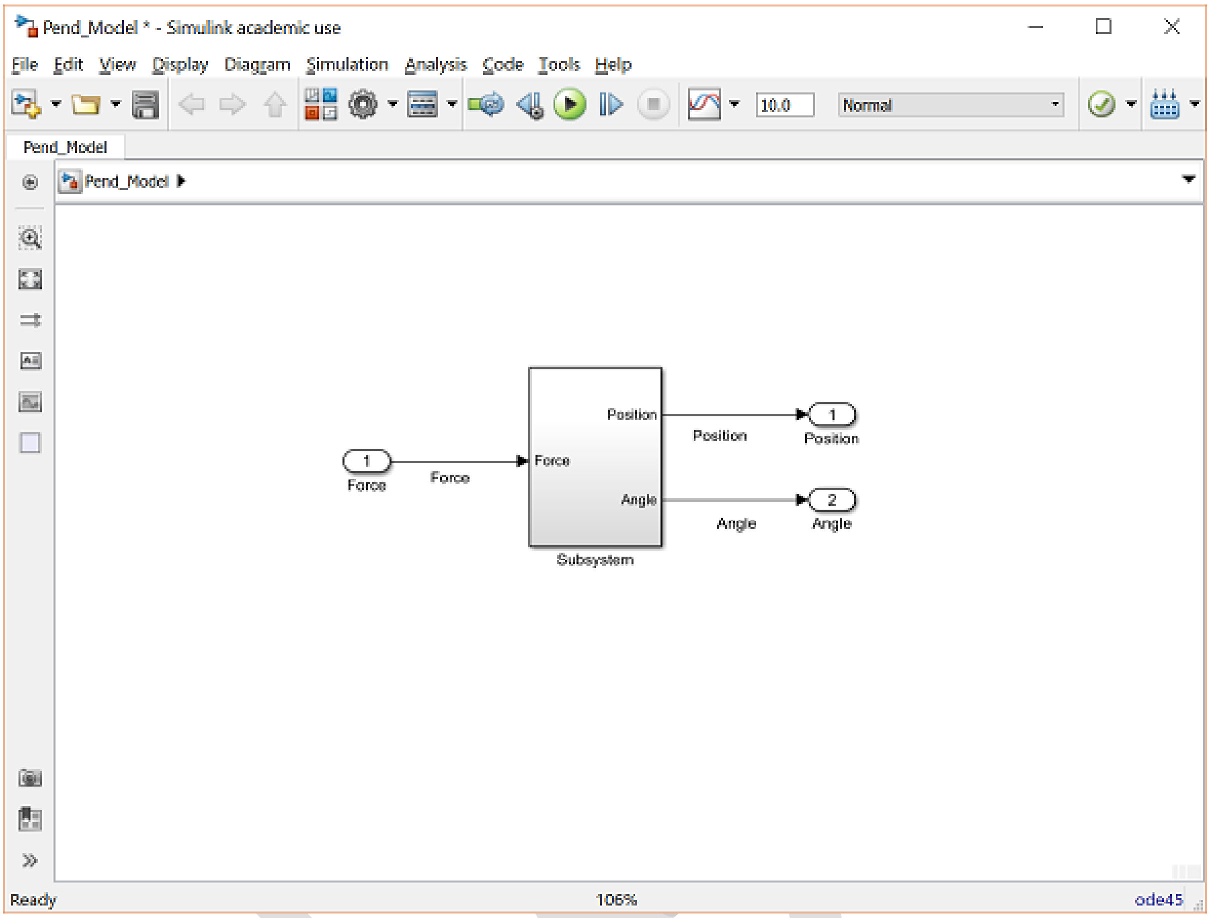
$$N = m(\ddot{x} - l\dot{\theta}^2 sin\theta + l\ddot{\theta} cos\theta) \qquad (۱۷)$$

$$P = m(l\dot{\theta}^2 cos\theta + l\ddot{\theta} sin\theta + g) \qquad (۱۸)$$

وقتی که تمامی این مراحل به اتمام رسد مدل شما به شکل زیر می‌باشد:



برای اینکه تمامی این اجزا را به عنوان یک بلوک زیرسیستم ذخیره کنیم، ابتدا تمامی بلوک‌ها را انتخاب کرده (کلید ترکیبی Ctrl+A) و سپس با راست کلیک بر روی ناحیه انتخاب شده، از منوی باز شده بر روی Create Subsystem from Selection کلیک کنید. مدل شما به شکل زیر در می‌آید. این مدل را می‌توانید از اینجا دریافت کنید.

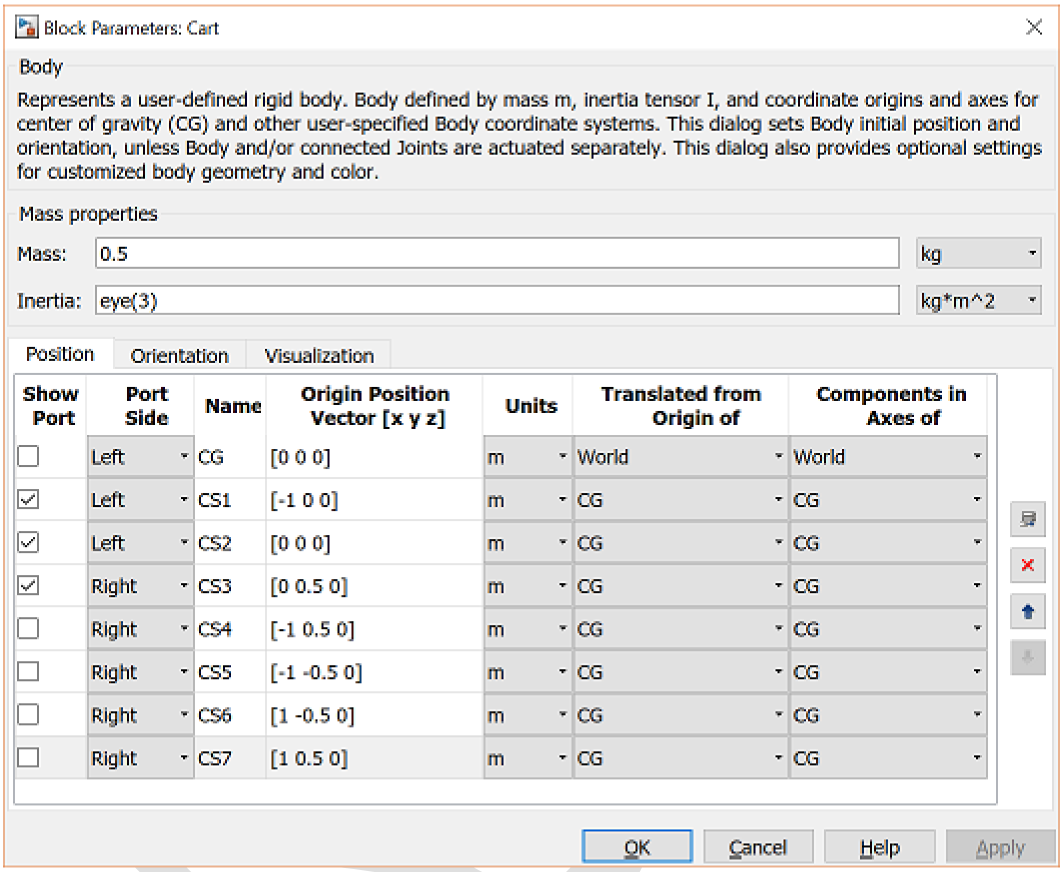


ساخت مدل غیر خطی در Simscape

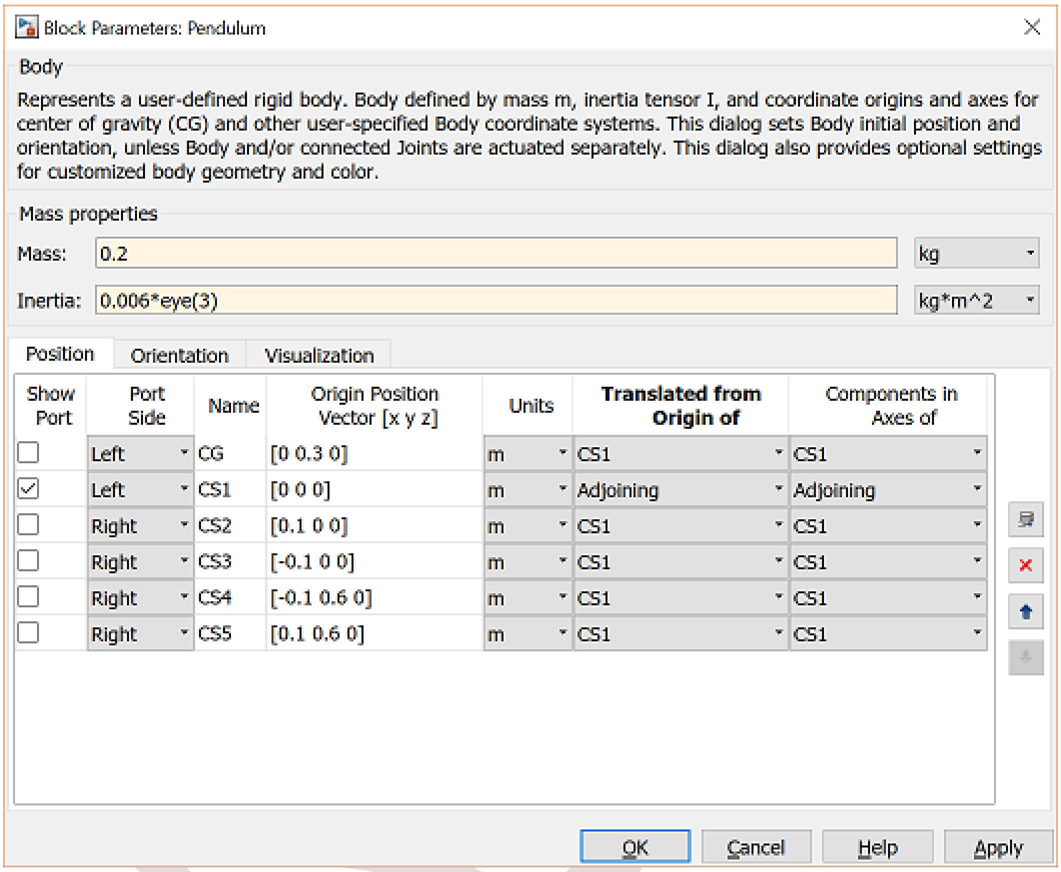
در این بخش، روش دیگری را برای مدل‌سازی سیستم پاندول معکوس با استفاده از بلوک‌های مدل‌سازی فیزیکی در افزونه Simscape سیمولینک تشریح می‌کنیم. بلوک‌های موجود در کتابخانه Simscape بیانگر اجزای فیزیکی واقعی می‌باشند، بنابراین می‌توان مدل‌های دینامیکی چند جزئی پیچیده را بدون نیاز به به دست آوردن معادلات ریاضی از قوانین فیزیکی (مانند کاری که در قسمت قبل با استفاده از قوانین نیوتن انجام شد) تولید کرد.

یک مدل سیمولینک جدید باز کرده و مراحل زیر را برای ساخت مدل پاندول معکوس در Simscape طی کنید. برای اینکه طراحی را بر مبنای یک دستگاه مختصات انجام دهیم، دستگاه مختصاتی را فرض می‌کنیم در جهت حرکت ارابه در راستای x (مثبت به سمت راست) و جهت مثبت راستای y به سمت بالا می‌باشد. در نتیجه با توجه به چیدمان استاندارد، جهت مثبت راستای z، خارج از صفحه‌ی حرکت به دست می‌آید.

- یک بلوک Body از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Bodies برای ارابه انتخاب کنید. با استفاده از پارامترهای سیستم که در بالا داده شده است، بر روی این بلوک دابل کلیک کرده و :Mass را برابر "0.5" بر حسب کیلوگرم قرار دهید. بلوک Body به طور پیش‌فرض دارای دو Port یا درگاه می‌باشد. چون نیاز به درگاهی داریم که محل اتصال پاندول و محل اعمال نیروی خارجی و نیروی اصطکاک را مشخص کنیم درگاه سومی را باید اضافه نمود. برای اینکار با استفاده از دکمه‌ای که در سمت راست برگه‌ی Position قرار دارد استفاده می‌کنیم. از آنجایی که ارابه فقط در یک بعد حرکت می‌کند، دو نیروی ذکر شده باید در این راستا قرار بگیرند (راستای x). چون می‌خواهیم ارابه را به شکل یک جرم نقطه‌ای (متمرکز) مدل‌سازی کنیم که برای آن تنها انتقال معنا دارد، نیازی به تغییر پارامتر دیگر ندارید. اما به دلیل اینکه می‌خواهیم از Simscape برای به تصویر کشیدن حرکت سیستم استفاده کنیم، باید درگاه‌های دیگری که چهار گوشه‌ی ارابه (تنها دو بعدی) را نسبت به مرکز گرانش آن تعریف می‌کنند اضافه کنیم. در شکل زیر تعریف Body ارابه را مشاهده می‌کنید:

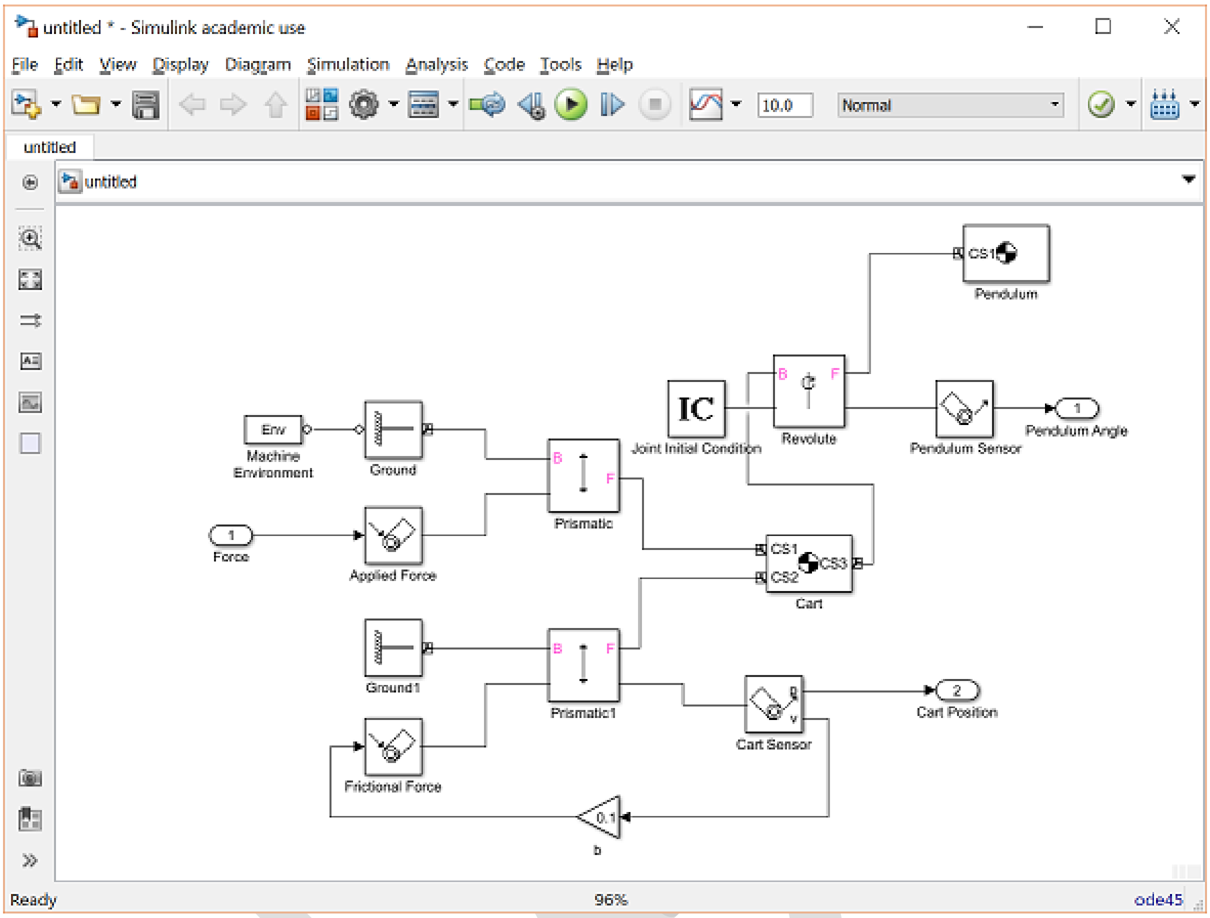


- بلوک Body دوم را که بیانگر پاندول می‌باشد در مدل قرار دهید. بر روی آن دابل کلیک کرده و :Mass را برابر "0.2" بر حسب کیلوگرم قرار دهید. چون پاندول تنها حول محور z دوران می‌کند، تنها اینرسی مربوط به این راستا باید تعریف شود. برای سادگی :Inertia را برابر "0.006*eye(3)" بر حسب kg*m^2 قرار دهید. چون پاندول را به عنوان یک جسم صلب تعریف می‌کنیم که علاوه بر جرم، دارای ابعاد نیز می‌باشد، این جسم می‌تواند دوران کند و همچنین تعریف موقعیت اتصال پاندول به ارابه و موقعیت مرکز گرانش آن بسیار مهم است. نقطه‌ی اتصال CS1 باید در موقعیت [0 0 0] با انتقال از مبدا Adjoining و مرکز گرانش CG باید ۰/۳ متر دورتر از محل اتصال CS1 (که در قبل تعریف شد) باشد. همچنین چهار گوشه‌ی پاندول را نیز تعریف می‌کنیم. در بخش Show Port تیک درگاهی که محل اتصال را تعریف می‌کند را قرار دهید. در برگه‌ی Visualization می‌توانید رنگ پاندول را عوض کنید تا با ارابه متفاوت باشد.



- در قدم بعد یک بلوک Revolute از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Joints برای تعریف مفصل بین ارابه و پاندول بردارید. به طور پیش‌فرض، این مفصل، دوران حول محور z را تعریف می‌کند که در این مثال درست می‌باشد. بلوک Body مربوط به ارابه را به درگاه base (B) مفصل و بلوک Body مربوط به پاندول را به درگاه follower (F) مفصل متصل کنید. بر روی بلوک Revolute دابل کلیک کرده و Number of sensor / actuator port: را برابر "2" قرار دهید.
- سپس یک بلوک Joint Initial Condition و یک بلوک Joint Sensor را از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Sensors & Actuators به مدل اضافه کرده و آنها را به بلوک Revolute متصل کنید. بر روی بلوک Joint Initial Condition دابل کلیک کرده و تیک گزینه‌ی Enable را بزنید. می‌توانیم برای شرایط اولیه‌ی موقعیت و سرعت مفصل از مقادیر پیش‌فرض استفاده کنیم. بر اساس تعریف Body پاندول در بالا، مقدار اولیه‌ی موقعیت 0، بیانگر عمود و بالا بودن پاندول می‌باشد. این تعریف سازگار با تعریفی که از θ داشتیم نیست اما سبب سازگاری نتایج با نتایج به دست آمده در مدل خطی بخش قبل می‌باشد. سپس بر روی بلوک Joint Sensor دابل کلیک کرده و واحد Angle را به rad تبدیل کنید. تنها اندازه‌گیری که برای این مفصل لازم است، موقعیت زاویه‌ای می‌باشد پس تنها تیک این گزینه را بزنید.
- دو بلوک Prismatic (کشویی) از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Joints برای تعریف درجه‌ی آزادی انتقالی ارابه و همچنین نیروهای وارده به آن به مدل اضافه کنید. چون ارابه در واقع یک جرم نقطه‌ای می‌باشد تنها نیاز به یک بلوک Prismatic است اما با استفاده از دو بلوک، می‌توانیم نیروها را در نقاط مختلف وارد کنیم. بر روی هر بلوک Prismatic دابل کلیک کرده و Axis of Action را به "[1 0 0]" تغییر دهید تا دو نیرو در جهت x تعریف شوند. سپس درگاه follower (F) هر بلوک را به درگاه‌های نیروی اعمال شده (CS1) و نیروی اصطکاک (CS2) بلوک Body ارابه متصل کنید.
- سپس دو بلوک Ground (زمین) را از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Bodies برای تعریف زمین حرکت ارابه به مدل اضافه کنید. خروجی هر یک از بلوک‌های Ground را به درگاه base (B) هر بلوک Prismatic متصل کنید.

- بر یکی از بلوک‌های Ground دابل کلیک کرده و تیک گزینه‌ی Show Machine Environment port را بزنید. سپس از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Bodies یک بلوک Machine Environment به مدل اضافه کنید و آنرا به بلوک Ground که درگاه آنرا اضافه کردید متصل کنید. بلوک Machine Environment به ما قابلیت تعریف نیروی گرانش را در شبیه‌سازی خود می‌دهد. در این مثال، مقادیر پیش‌فرض یعنی جهت (منفی راستای y) و اندازه ("9.81") بر حسب m/s^2 صحیح می‌باشد. این بلوک همچنین به ما اجازه‌ی تعریف پارامترهایی را برای نمایش و حل عددی شبیه‌سازی می‌دهد. مقادیر پیش‌فرض برای این مثال مناسب می‌باشند.
- سپس دو بلوک Joint Actuator و یک بلوک Joint Sensor از کتابخانه‌ی Simscape/Multibody/First Generation(1G)/Sensors & Actuators در مدل قرار دهید. از بلوک Joint Actuator برای ایجاد نیروی خارجی و نیروی اصطکاک استفاده کرده و از بلوک Joint Sensor برای خواندن حرکت ارابه استفاده می‌کنیم. لازم است متذکر شویم که یک بلوک Translational Friction نیز برای اینکار وجود دارد اما به علت اینکه ما از یک مدل ویسکوز ساده برای اصطکاک استفاده می‌کنیم محاسبات مربوط به نیروی اصطکاک را خودمان انجام خواهیم داد. بر روی یکی از بلوک‌های Prismatic دابل کلیک کرده و Nu,ber of sensor / actuator ports: را برابر "1" (برای عملگر نیرو) قرار دهید. برای بلوک Prismatic دیگر مقدار Number of sensor / actuator ports: را برابر "2" (یکی برای عملگر نیرو و دیگری برای سنسور ارابه) قرار می‌دهیم. سپس طبق تعریف گفته شده، بلوک‌های Joint Sensor و Joint Actutator را به این بلوک‌ها متصل می‌نماییم. مقادیر پیش‌فرض برای بلوک‌های Joint Actuator برای مثال ما کافی می‌باشند اما برای بلوک Joint Sensor باید آنرا با خروجی موقعیت و سرعت تبدیل کنیم زیرا برای محاسبه‌ی نیروی اصطکاک به سرعت نیاز داریم. بر روی بلوک Joint Sensor دابل کلیک کرده و تیک گزینه‌ی Velocity و همچنین Position را بزنید. واحدهای متریک پیش‌فرض نیاز به تغییر ندارند. همچنین تیک گزینه‌ی Output selected parameters as one signal را بردارید.
- یک بلوک Gain (بهره) از کتابخانه‌ی Simulink/Math Operations که بیانگر ضریب اصطکاک ویسکوز می‌باشد به مدل اضافه کنید. بهره را با توجه به پارامترهای سیستم که در بالا ذکر شد برابر "0.1" قرار دهید و ورودی آنرا به خروجی سرعت ارابه‌ی بلوک Joint Sensor متصل کنید. خروجی آنرا به بلوک Joint Actuator مربوط به نیروی اصطکاک متصل کنید.
- در قدم بعد دو بلوک Out1 و یک بلوک In1 از کتابخانه‌ی Simulink/Ports & Subsystems اضافه کنید. بلوک‌های Out1 را به خروجی‌های باقی‌مانده‌ی بلوک Joint Sensor متصل کنید. بلوک In1 را به دیگر ورودی بلوک Joint Actuator وصل کنید.
- در نهایت اجزا را مانند شکل زیر نام‌گذاری و متصل نمایید. برای چرخش بلوک‌ها مانند وارونه کردن آنها از طریق راست کلیک بر روی هر بلوک و انتخاب Rotate Block از منوی Rotate & Flip اقدام کنید.

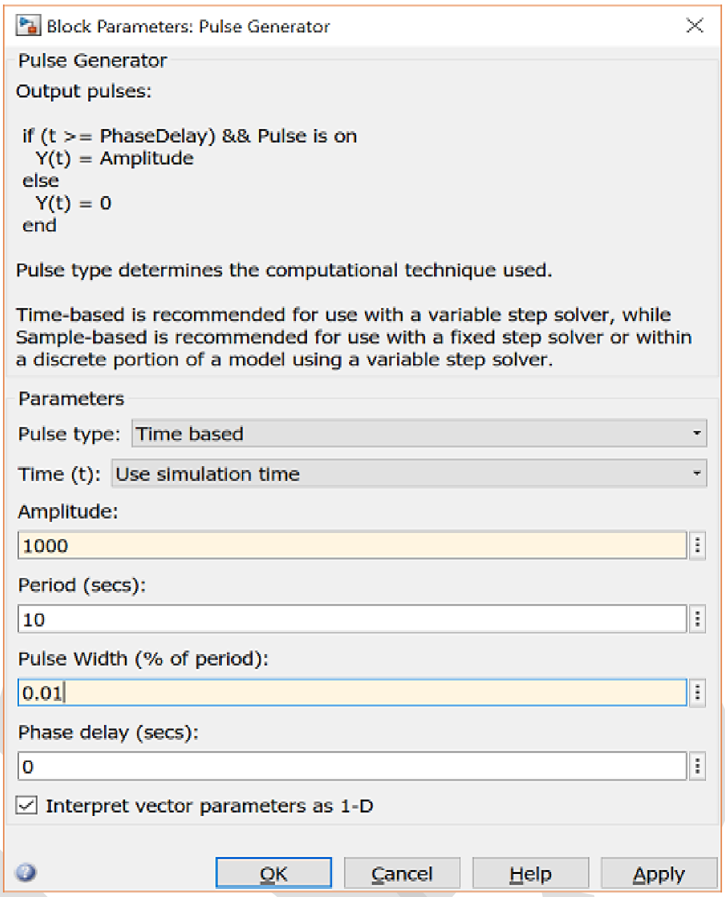


همچنین می‌توانید این مدل را مانند قسمت قبل در قالب یک Subsystem ذخیره نمایید. برای تغییر رنگ Subsystem بر روی بلوک آن راست کلیک کرده و از منوی نمایش داده شده گزینه‌ی Background Color را انتخاب کنید. فایل مدل کامل را می‌توانید از اینجا دریافت کنید. برای اجرای این مدل نیاز به افزونه‌ی Simscape سیمولینک دارید. در **بخش نهم: طراحی کنترلر در سیمولینک** از این مدل استفاده خواهیم نمود.

تولید پاسخ حلقه باز

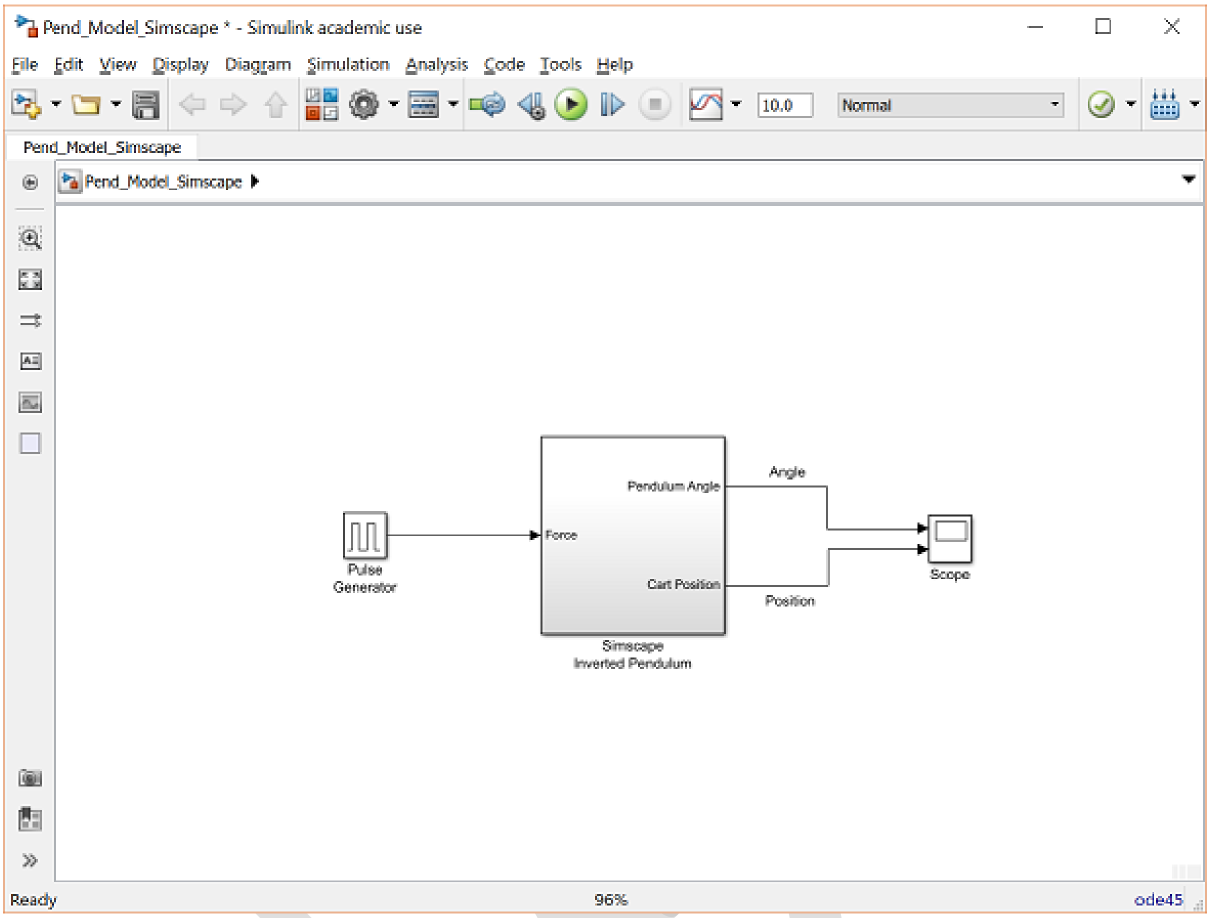
در این قسمت پاسخ سیستم پاندول معکوس به نیروی ضربه‌ای اعمال شده بر ارابه را به دست می‌آوریم. این شبیه‌سازی نیاز به یک ورودی ضربه دارد. از آنجایی که چنین بلوکی در کتابخانه‌ی سیمولینک وجود ندارد از بلوک Pulse Generator برای تخمین یک ورودی ضربه واحد استفاده می‌نماییم. برای اینکار می‌توانیم از هر یک از دو مدل‌های قسمت قبل استفاده کنیم اما ما مدل Simscape را انتخاب می‌نماییم زیرا به ما قابلیت نمایش حرکات سیستم پاندول معکوس را می‌دهد. قدم‌های زیر را دنبال کنید:

- مدل Simscape سیستم پاندول معکوس ساخته شده در قسمت قبل را باز کنید.
- یک بلوک Pulse Generator از کتابخانه‌ی Simulink/Sources بردارید. بر روی آن دابل کلیک کرده و پارامترهای آنرا به شکل زیر تغییر دهید. خصوصا پارامتر Period: را به "10" تغییر دهید زیرا باید برای تولید تنها یک پالس، شبیه‌سازی را برای ۱۰ ثانیه اجرا کنیم. همچنین Amplitude را به "1000" و Pulse Width (% of period) را به "0.01" تغییر دهید. تمام این تنظیمات با یکدیگر پالسی را تشکیل می‌دهند که تقریبا ضربه‌ی واحد که به معنی ورودی بسیار بزرگ در مدت کوتاهی از زمان را تخمین می‌زند. مساحت زیر منحنی این پالس برابر ۱ می‌باشد.



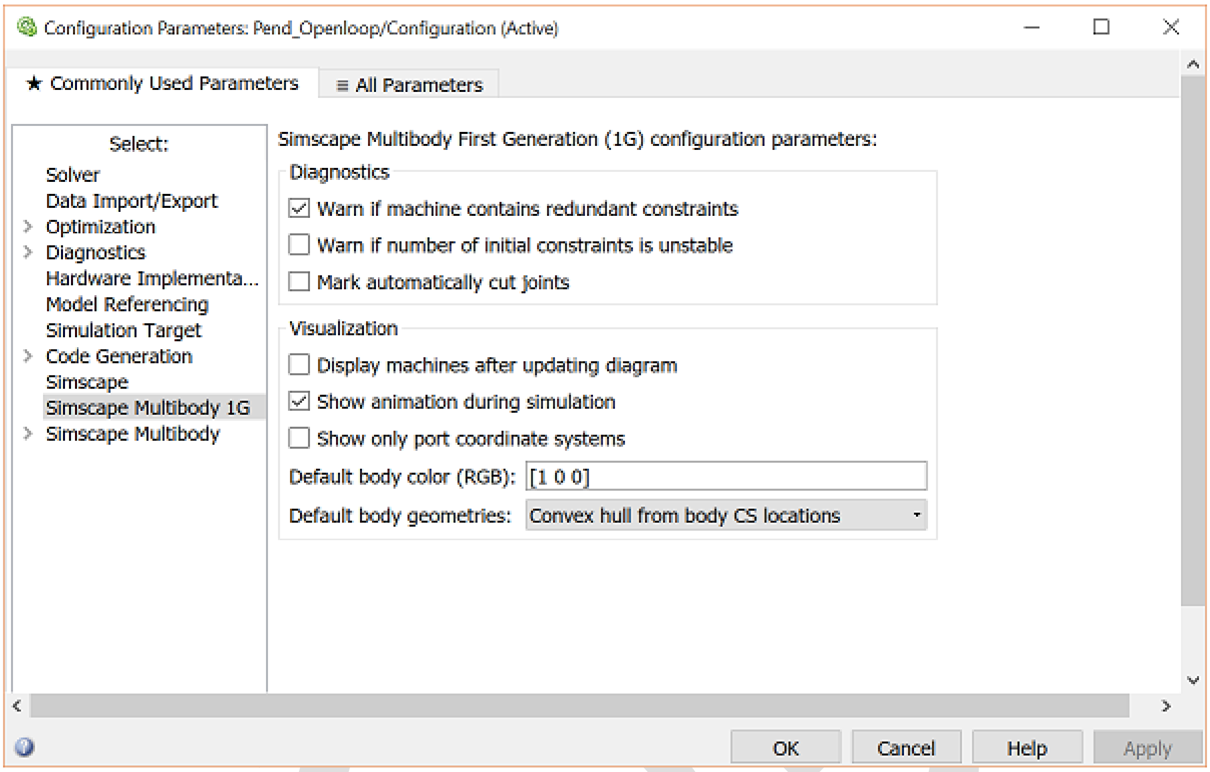
- یک بلوک Scope از کتابخانه‌ی Simulink/Sinks بردارید.
- برای اینکه در یک Scope، دو ورودی را نمایش دهید، بر روی بلوک Scope راست کلیک کرده و گزینه‌ی Signals & Ports را انتخاب کرده و Number of Input Ports را برابر "2" قرار دهید.

بلوک‌ها را به شکل زیر متصل و نام‌گذاری نمایید:

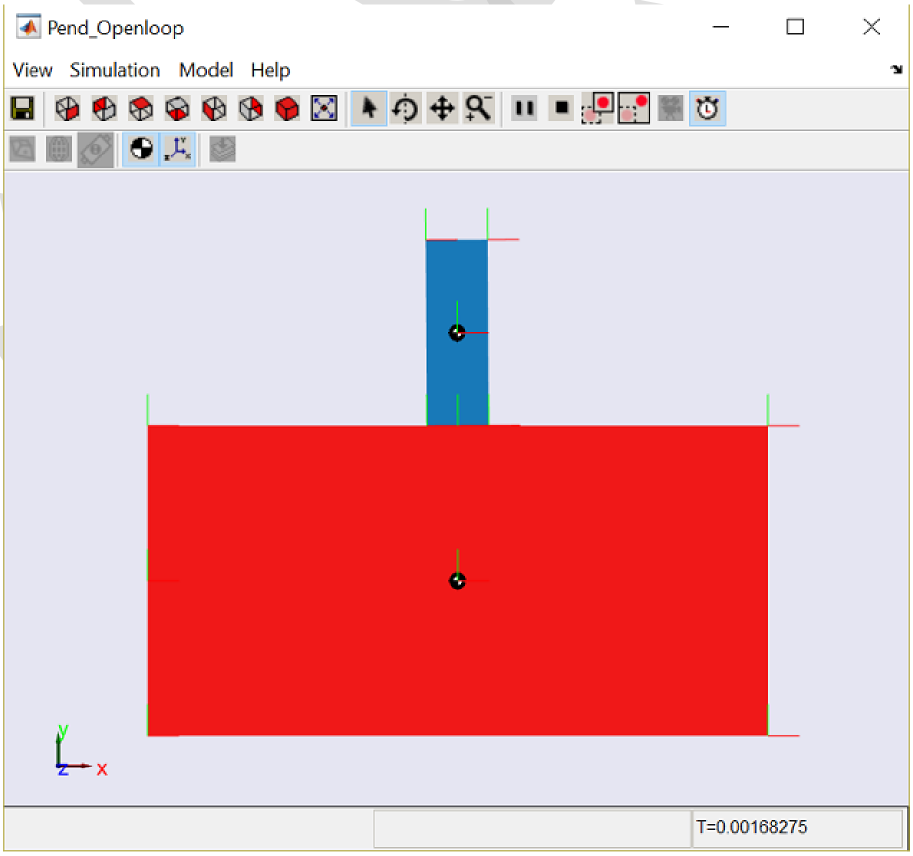


این سیستم را با نام Pend_Openloop.slx ذخیره کرده یا از داخل سی‌دی کپی کنید.

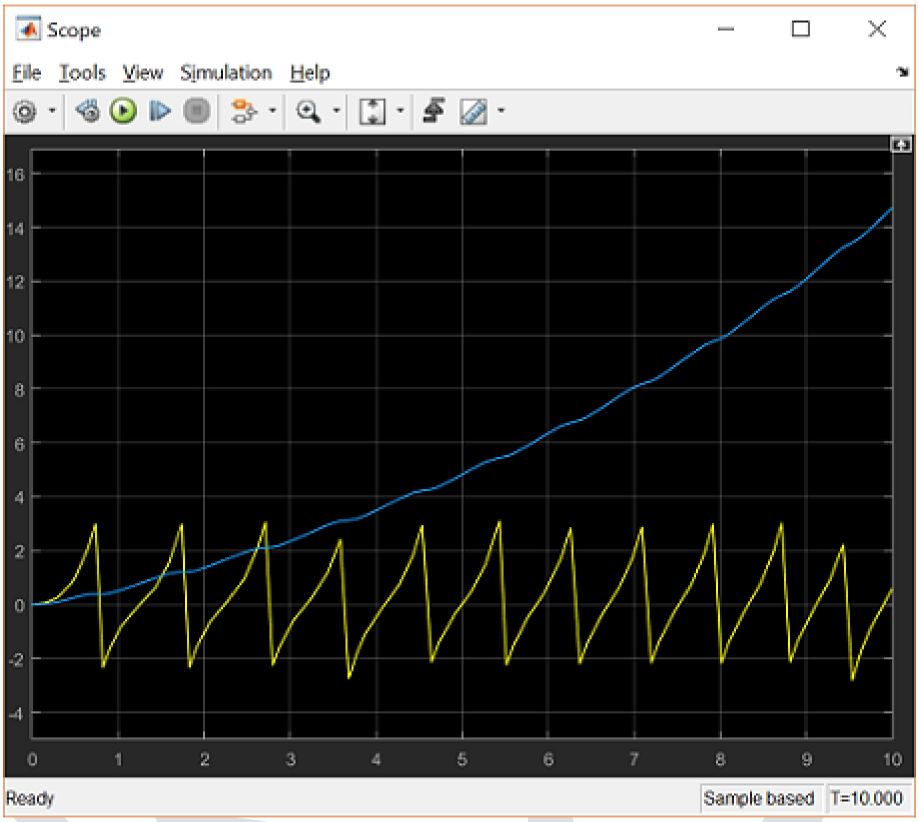
پیش از شروع شبیه‌سازی، باید قابلیت نمایش سیستم پاندول معکوس را فعال کنیم. از منوی بالای پنجره‌ی مدل، به قسمت Simulation > Model Configuration Parameters بروید. سپس در پنجره‌ی باز شده، در سمت چپ پنجره به قسمت Simscape Multibody 1G بروید. در این قسمت تیک Show animation during simulation را که در شکل زیر نمایش داده شده است بزنید.



حال شبیه‌سازی را اجرا کنید (انتخاب گزینه Run از منوی Simulation یا فشردن دکمه Ctrl+T). در حین اجرای شبیه‌سازی، انیمیشن پاندول معکوس، نتایج به دست آمده از سیستم را نمایش می‌دهد.



سپس با باز کردن Scope، خروجی زاویه‌ی پاندول و موقعیت ارابه را مشاهده می‌نمایید:



متوجه می‌شویم که پاندول به طور مکرر چرخش‌های کاملی را انجام می‌دهد که در بازه $\pm\pi$ رادیان نمایش داده می‌شود. علاوه بر آن موقعیت ارابه به بینهایت میل کرده و بی کران می‌شود اما تحت چرخش‌های پاندول نوسانانی را دارد. نتایج به دست آمده تا حدی متفاوت از نتایج به دست آمده در فصل پاندول معکوس: تحلیل سیستم می‌باشد. دلیل این تفاوت این است که در این شبیه‌سازی از مدل کاملا غیر خطی استفاده شده است اما در تحلیل پیشین از تقریب خطی مدل پاندول معکوس استفاده کردیم. برای اینکه نتایج به دست آمده از شبیه‌سازی قابل مقایسه با نتایج سابق باشند، از مدل شبیه‌سازی خود، مدلی خطی را استخراج خواهیم کرد.

استخراج مدل خطی از شبیه‌سازی

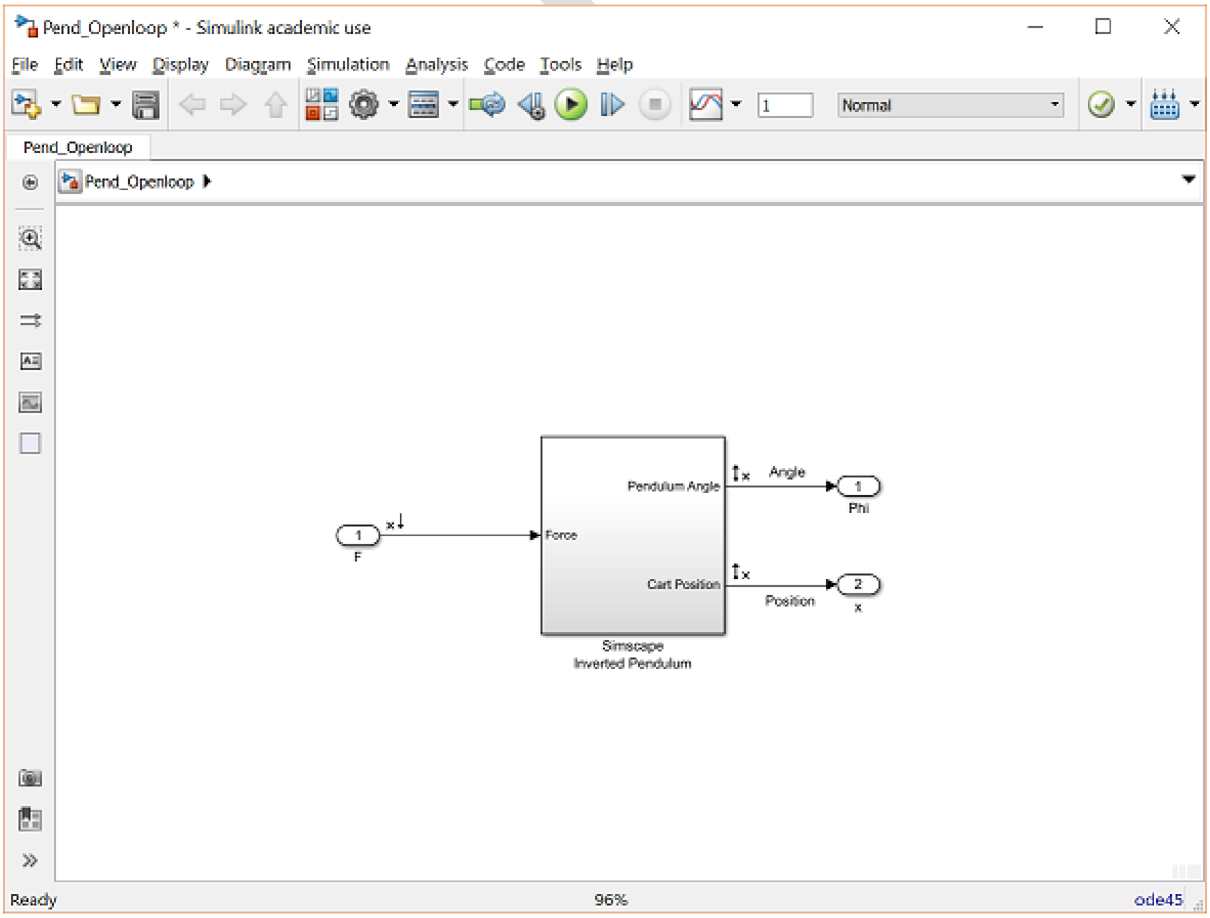
علاوه بر مقایسه‌ی مدل شبیه‌سازی با نتایج سابق، ممکن است برای تحلیل و طراحی به دست آوردن یک مدل خطی مدنظر باشد. بسیاری از تکنیک‌های تحلیل که به صورت رایج برای تحلیل سیستم‌های دینامیکی انجام شده و طراحی‌هایی که از طریق آنها صورت می‌گیرند تنها بر روی مدل‌های خطی قابل پیاده‌سازی می‌باشند. بنابراین ممکن است هدف ما به دست آوردن یک تقریب مدل خطی از مدل شبیه‌سازی غیر خطی باشد. اینکار را با استفاده از سیمولینک انجام خواهیم داد:

- برای شروع، یکی از مدل‌های ساخته شده در فصل‌های قبل (Pend_Model.slx یا Pend_Model_Simscape.slx) را باز کنید.
- اگر با استفاده از متغیرها مدل شبیه‌سازی خود را تعریف کرده باشید لازم است تا ثابت‌های فیزیکی را پیش از فرآیند خطی‌سازی در فضای کاری متلب تعریف کنید. برای اینکار دستورات زیر را در پنجره دستور متلب وارد کنید:

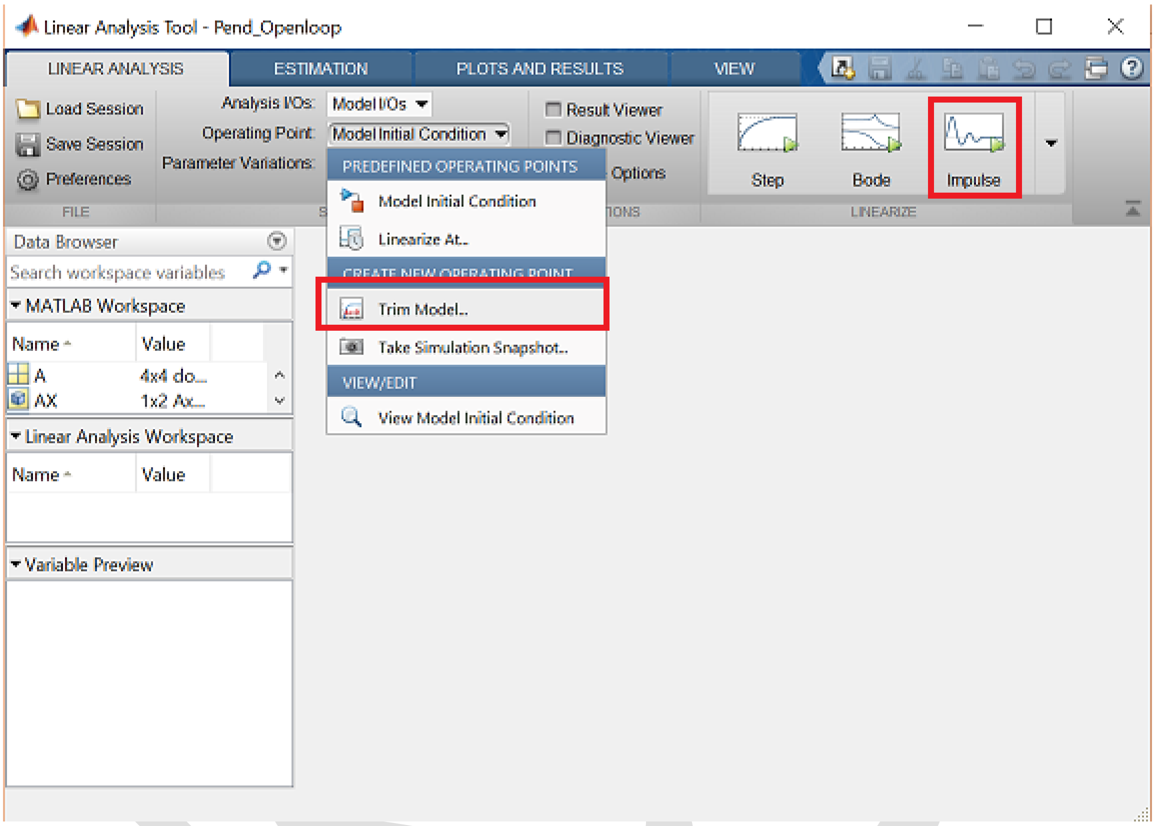
```
• m = 0.2;

• b = 0.1;

• I = 0.006;
```

```
•  g = 9.8;

•  l = 0.3;
```

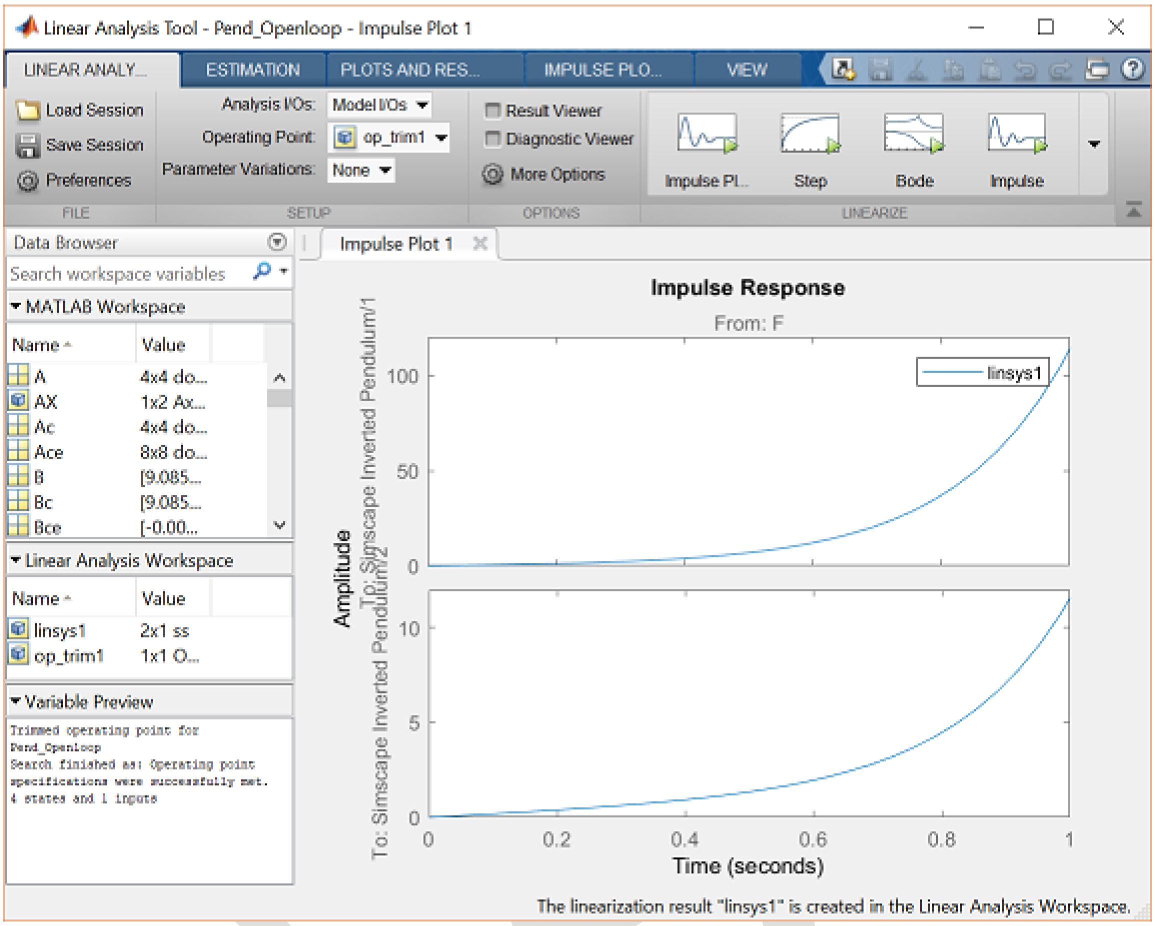
- سپس از منوی بالای مدل به قسمت Analysis > Control Design > Linear Analysis بروید. بدین صورت پنجره‌ی Linear Analysis Tool باز خواهد شد.
- برای انجام خطی‌سازی، ابتدا باید ورودی و خروجی‌های مدل و نقطه‌ی کاری که خطی‌سازی حول آن انجام می‌شود را مشخص کنیم. ابتدا بر روی سیگنال ورودی نیرو در مدل سیمولینک راست کلیک کنید. سپس از منوی حاصل گزینه‌ی Linear Analysis Points > Open-loop Input را انتخاب کنید. به روش مشابه بر روی هر یک از دو سیگنال خروجی مدل (زاویه پاندول و موقعیت ارابه) راست کلیک کرده و از منوی حاصل گزینه‌ی Linear Analysis Points > Open-loop Output را انتخاب نمایید. حال باید سیگنال‌های ورودی و خروجی با یک علامت فلش بر روی آنها مانند شکل زیر مشخص شده باشند:



- حال باید نقطه‌ی کاری سیستم که حول آن خطی‌سازی انجام می‌شود را مشخص کنیم. از منوی Operating Point: گزینه Trim Model را مانند شکل زیر انتخاب کنید. با اینکار پنجره‌ی Trim the model باز خواهد شد. در این پنجره، بر روی دکمه Start trimming که با علامت مثلث سبز رنگ مشخص شده است کلیک کنید. با اینکار نقطه‌ی کاری op_trim1 ایجاد می‌شود.
- چون می‌خواهیم پاسخ ضربه این سیستم را به دست آوریم، به برگه‌ی LINEAR ANALYSIS بازگشته و impulse را مانند شکل زیر انتخاب کنید:



- در نهایت op_trim1 را از منوی کشویی Operating Point: انتخاب کرده و بر روی دکمه‌ی Impulse که با مثلث سبز رنگ مشخص شده است کلیک کنید. با اینکار به طور خودکار منحنی پاسخ ضربه و مدل خطی linsys1 ایجاد خواهد شد.
- برای مقایسه‌ی این نتایج با نتایج به دست آمده از **بخش دوم: تحلیل سیستم**، لازم است تا مقیاس محور x را تغییر دهید. برای اینکار راست کلیک کرده و از منوی حاصل، گزینه‌ی Properties را انتخاب کنید. پنجره‌ای مانند شکل زیر باز خواهد شد که نمودار بالا پاسخ زاویه‌ی پاندول و منوی پایین پاسخ موقعیت ارابه است:



این نمودارها بسیار مشابه نمودارهای به دست آمده در **بخش دوم: تحلیل سیستم** هستند.

همچنین می‌توان مدل خطی شده را برای تحلیل و طراحی بیشتر، به درون فضای کاری متلب صادر کرد. برای اینکار در قسمت Linear Analysis Workspace بر روی شئ linsys1 راست کلیک کرده و آنرا کپی کنید. سپس بر روی فضای کاری متلب راست کلیک کرده و شئ کپی شده را paste کنید.

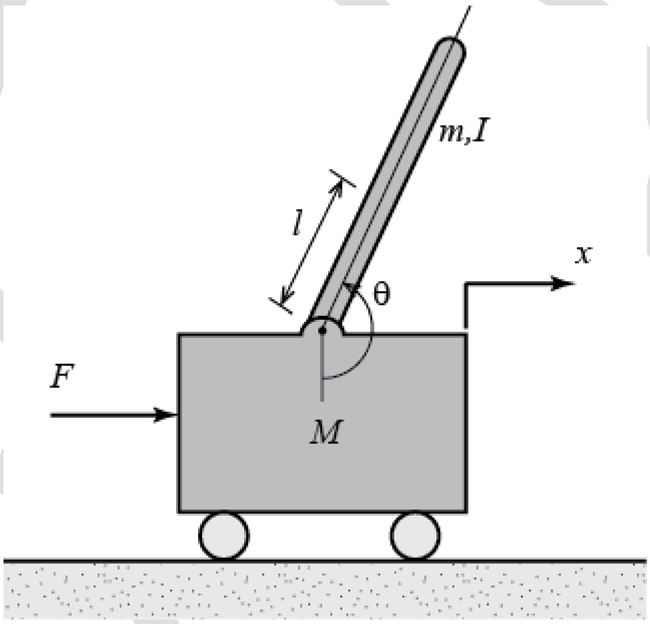
بخش نهم: طراحی کنترلر در سیمولینک

فهرست مطالب بخش

در **بخش هشتم: مدل‌سازی سیمولینک**، دو مدل متفاوت برای شبیه‌سازی به دست آوردیم. حال با استفاده از این مدل‌ها در سیمولینک به طراحی کنترلر از روش‌های مختلف و شبیه‌سازی نتایج حلقه بسته‌ی آنها می‌پردازیم.

صورت مسئله و نیازهای طراحی

در این مسئله، ارابه به همراه پاندول معکوس متصل به آن که در شکل زیر نشان داده شده است، تحت تاثیر نیروی ضربه‌ای F قرار می‌گیرد:



برای این مسئله، فرض می‌کنیم:

(M) جرم ارابه: 0.5 Kg

(m) جرم پاندول: 0.2 Kg

(b) ضریب اصطکاک ارابه: 0.1 N/m.sec

(l) فاصله از مرکز جرم پاندول: 0.3 m

(I) ممان اینرسی جرمی پاندول: 0.006 Kg.m^2

(F) نیروی وارد شده به ارابه

(x) موقعیت ارابه

(theta) زاویه پاندول از قائم

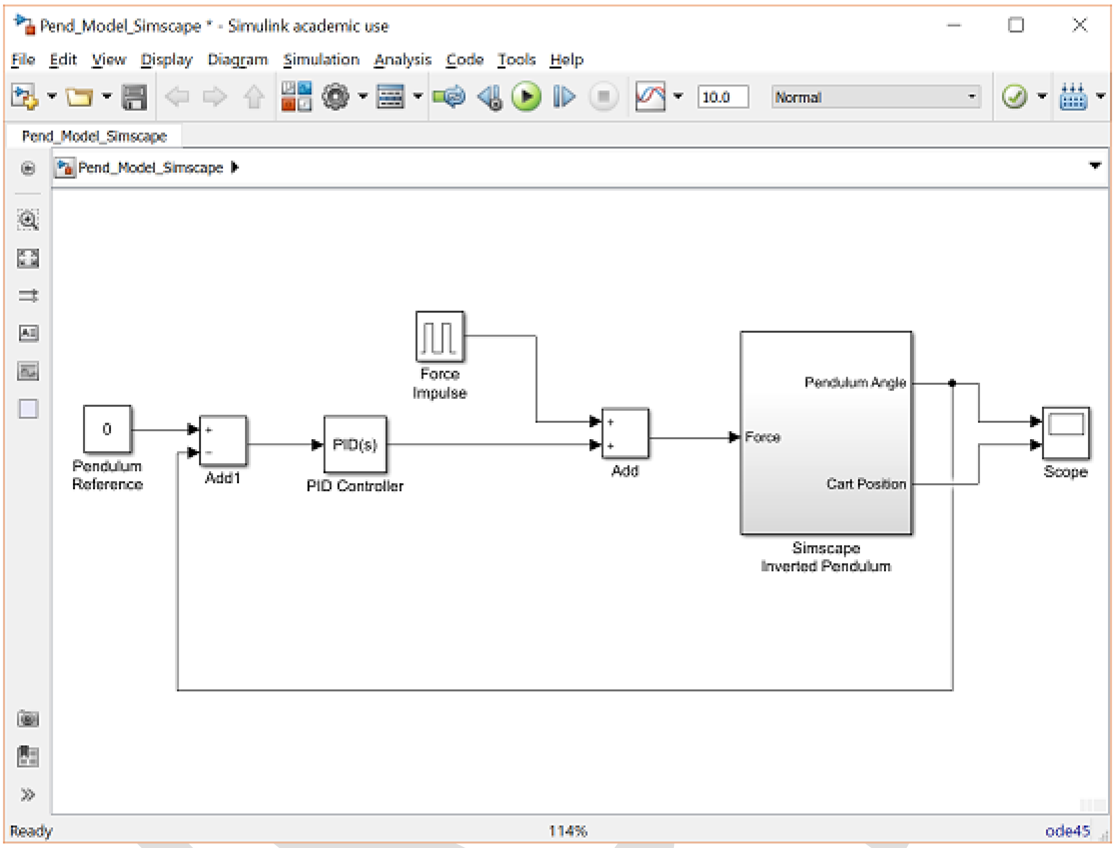
در روند طراحی یک کنترلر PID را توسعه داده و آنرا برای سیستم تک ورودی تک خروجی اعمال می‌کنیم. این کنترلر سعی بر حفظ حالت عمودی پاندول در هنگام اعمال نیروی ضربه‌ای 1 Nsec دارد. موقعیت ارابه را در نظر نمی‌گیریم. تحت این شرایط، نیازهای طراحی عبارتست از:

- زمان نشست کمتر از ۵ ثانیه
- پاندول نباید هیچگاه بیشتر از ۰/۰۵ رادیان از موقعیت عمودی حرکت کند

پیاده سازی کنترل PID برای مدل غیر خطی

در **بخش سوم: طراحی کنترلر PID**، یک کنترلر PID با بهره‌های تناسبی، انتگرالی و مشتق به ترتیب برابر ۱۰۰، ۱ و ۲۰ طراحی گردید. برای به کارگیری این سیستم حلقه بسته، باید از یکی از مدل‌های ساخته شده در **بخش هشتم: مدل‌سازی سیمولینک** استفاده نماییم. با دنبال کردن مراحل زیر، یک مدل حلقه بسته با ورودی مرجع برای زاویه پاندول و نیروی اغتشاش وارده شده به ارابه خواهیم ساخت:

- برای شروع یکی از مدل‌های ساخته شده در فصل قبل را باز کنید. می‌توانید مدل‌های Pend_Model.slx یا Pend_Model_Simscape.slx را از اینجا دریافت کنید. ما در این قسمت از مدل Simscape برای نمایش انیمیشن استفاده خواهیم کرد.
- دو بلوک Add را از کتابخانه‌ی Simulink/Math Operations وارد کنید.
- در بلوک‌های Add گزینه‌ی List of signs: را به شکل “+-” تغییر دهید.
- یک بلوک Constant را از کتابخانه‌ی Simulink/Sources وارد کنید. مقدار آنرا به 0 تغییر دهید. این مقدار بیانگر ورودی مرجع مربوط به موقعیت عمودی پاندول می‌باشد. لازم به ذکر است که در مدل غیر Simscape (و بقیه قسمت‌های این مثال) برای تعریف زاویه‌ی پاندول در حالت عمودی از مقدار π استفاده کنید.
- یک بلوک PID Controller را از کتابخانه‌ی Simulink/Continuous وارد کنید.
- بلوک PID را با دابل کلیک بر روی آن ویرایش کنید. بهره‌ی تناسبی (P) Proportional را برابر ”100“، بهره‌ی انتگرالی (I) Integral را برابر ”1“ و بهره‌ی مشتق (D) Derivative را برابر ”20“ قرار دهید.
- حال بلوک‌ها را به شکل زیر متصل کنید:



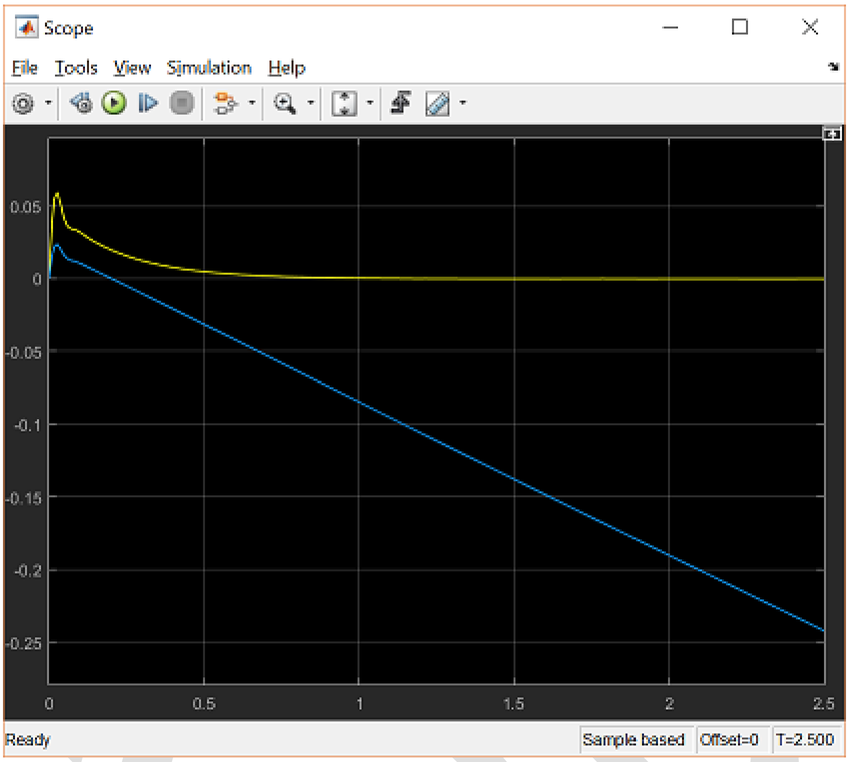
این مدل را می‌توانید از درون سی‌دی دریافت کنید.

پاسخ حلقه بسته غیر خطی

حال می‌توانیم سیستم حلقه بسته را شبیه‌سازی کنیم. دقت کنید که پارامترهای فیزیکی به شکل زیر تعریف شده باشند:

```
M = 0.5;

m = 0.2;

b = 0.1;

I = 0.006;

g = 9.8;

l = 0.3;
```

حال شبیه‌سازی را اجرا کنید (انتخاب Run از منوی Simulation یا فشردن دکمه Ctrl+T). در حین اجرای شبیه‌سازی، انیمیشن پاندول معکوس حرکت حاصل سیستم را نمایش می‌دهد. یادآوری می‌کنیم که باید تیک گزینه Show animation during simulation در منوی Simulation > Model Configuration Parameters زده شده باشد. بعد از اتمام شبیه‌سازی، پاسخ زیر به دست می‌آید:



این پاسخ تقریبا مشابه پاسخ حلقه بسته‌ی به دست آمده در متلب می‌باشد (برای مثال در **بخش سوم: طراحی کنترلر PID**). متذکر می‌شویم به دلیل اینکه انحراف زاویه از نقطه‌ی کاری (تقریبا ۰/۰۵ رادیان) بسیار کم می‌باشد، کنترلر PID سیستم غیر خطی را به خوبی کنترل می‌کند.

پیوست

پیوست اول: لیست دستورات متلب

در جدول زیر لیستی از دستورات مورد استفاده در این کتاب آورده شده است. می‌توان از دستور help در متلب برای نحوه‌ی استفاده از هر یک از دستورات بهره برد.

در این کتاب از دستورات و توابع متلب، سیمولینک و چند تابع که به صورت جداگانه نوشته شده بود استفاده می‌نماییم. توابعی که از توابع استاندارد متلب نمی‌باشند را می‌توانید در سی‌دی ارائه شده همراه کتاب پیدا کنید.

توضیحات	دستور
مقدار مطلق	abs
محاسبه‌ی ماتریس K برای جایدهی قطب‌های A-BK (دستور place را نیز مشاهده نمایید)	acker
تنظیم مقیاس نمودار کنونی (دستور plot و figure را نیز مشاهده نمایید)	axis
رسم دیاگرام بودی (دستور logspace، margin و nyquist1 را نیز مشاهده نمایید)	bode
تبدیل سیستم پیوسته به سیستم گسسته	c2d
پاک کردن شکل	clf
کانولوشن (برای ضرب چندجمله‌ای‌ها در یکدیگر) (دستور deconv را نیز مشاهده نمایید)	conv
ماتریس کنترل‌پذیری (دستور obsv را نیز مشاهده نمایید)	ctrb
دکانولوشن و تقسیم چندجمله‌ای‌ها (دستور conv را نیز مشاهده نمایید)	deconv
به دست آوردن دترمینان یک ماتریس	det
طراحی رگولاتور خطی درجه دوم برای سیستم‌های گسسته (دستور lqr را نیز مشاهده نمایید)	dlqr
محاسبه‌ی مقادیر ویژه‌ی یک ماتریس	eig
تلرانس عددی متلب	eps
اتصال سیستم خطی در حلقه‌ی فیدبک	feedback
ساخت یک شکل جدید یا بازسازی شکل کنونی (دستور subplot و axis را نیز مشاهده نمایید)	figure
حلقه‌ی For	for
فرمت نمایش اعداد (اعشاری، عدد علمی و ...)	format
ساخت امفایل تابع	function
رسم خطوط شطرنجی بر روی نمودار کنونی	grid
اضافه کردن متن به نمودار کنونی (دستور text را نیز مشاهده نمایید)	gtext
راهنمایی در متلب	help
نگه‌داشتن نمودار کنونی (دستور figure را نیز مشاهده نمایید)	hold
جمله‌ی شرطی	if
به دست آوردن قسمت موهومی یک عدد مختلط (دستور real را نیز مشاهده نمایید)	imag
پاسخ ضربه‌ی سیستم خطی (دستور step و lsim را نیز مشاهده نمایید)	impulse
پیام برای دریافت ورودی کاربر	input
به دست آوردن معکوس یک ماتریس	inv
راهنمای نمودار	legend
اندازه‌ی یک بردار (دستور size را نیز مشاهده نمایید)	length
به دست آوردن یک بردار با فواصل خطی	linspace
ایجاد نمودار نایکوئست بر مقیاس لگاریتمی (دستور nyquist1 را نیز مشاهده نمایید)	lnyquist
لگاریتم طبیعی، مشابه دستور log10	log
رسم نمودار بر حسب مختصات لگاریتمی، مشابه دستور semilogx/semilogy	loglog
به دست آوردن یک بردار با فواصل لگاریتمی	logspace

طراحی رگولاتور خطی درجه دوم برای سیستم‌های پیوسته (دستور dlqr را نیز مشاهده نمایید)	lqr
شبیه‌سازی یک سیستم خطی (دستور step و impulse را نیز مشاهده نمایید)	lsim
به دست آوردن حد بهره، حد فاز و فرکانس‌های گذر (دستور bode را نیز مشاهده نمایید)	margin
ایجاد مینیمال حقیقی[۴۰] یک سیستم (خنثی‌سازی صفر و قطب اجباری)	minreal
نرم یک بردار	norm
رسم نمودار نایکوئست (دستور lnyquist را نیز مشاهده نمایید). این دستور یک جایگزین برای دستور استاندارد متلب nyquist می‌باشد که نمودارهای دقیق‌تری را نتیجه می‌دهد.	nyquist1
ماتریس مشاهده‌پذیری (دستور ctrb را نیز مشاهده نمایید)	obsv
ایجاد یک بردار یا ماتریس با درایه‌های یک (دستور zeros را نیز مشاهده نمایید)	ones
محاسبه‌ی ماتریس K برای جایدهی قطب‌های A-BK (دستور acker را نیز مشاهده نمایید)	place
رسم یک نمودار (دستور figure، axis و subplot را نیز مشاهده نمایید)	plot
به دست آوردن چندجمله‌ای مشخصه	poly
ارزیابی چندجمله‌ای	polyval
چاپ نمودار کنونی (بر روی یک پرینتر یا فایل)	print
نمودار صفر-قطب برای سیستم‌های خطی	pzmap
به دست آوردن تعداد ردیف یا ستون‌های خطی مستقل در یک ماتریس	rank
به دست آوردن قسمت حقیقی یک عدد مختلط (دستور imag را نیز مشاهده نمایید)	real
به دست آوردن مقدار بهره (k) و قطب‌های متناظر با نقاط انتخاب شده بر روی نمودار مکان هندسی ریشه‌ها	rlocfind
رسم نمودار مکان هندسی ریشه‌ها	rlocus
به دست آوردن ریشه‌های یک چندجمله‌ای	roots
به دست آوردن ضریب تناسب برای یک سیستم فیدبک تمام حالات	rscale
رسم خطوط ضریب میرایی ثابت (zeta) و فرکانس طبیعی ثابت (Wn) (دستور sigrid و zgrid را نیز مشاهده نمایید)	sgrid
به دست آوردن ابعاد یک بردار یا ماتریس (دستور length را نیز مشاهده نمایید)	size
جذر (ریشه‌ی دوم)	sqrt
ساخت مدل فضای حالت و تبدیل مدل LTI به فضای حالت (دستور tf را نیز مشاهده نمایید)	ss
دسترسی به اطلاعات فضای حالت (دستور tfdata را نیز مشاهده نمایید)	ssdata
نمودار پله‌ای برای پاسخ گسسته	stairs
رسم پاسخ پله (دستور impulse و lsim را نیز مشاهده نمایید)	step
تقسیم پنجره‌ی نمودار به چند قسمت (دستور plot و figure را نیز مشاهده نمایید)	subplot
اضافه کردن متن به نمودار کنونی (دستور title، xlabel، ylabel و gtext را نیز مشاهده نمایید)	text
ساخت تابع تبدیل یا تبدیل به فرم تابع تبدیل (دستور ss را نیز مشاهده نمایید)	tf
دسترسی به اطلاعات تابع تبدیل (دستور ssdata را نیز مشاهده نمایید)	tfdata
اضافه کردن عنوان به نمودار کنونی	title
به دست آوردن فرکانس پهنای باند با داشتن ضریب میرایی و زمان نشست یا نمو	wbw
اضافه کردن عنوان برای محور افقی/عمودی به نمودار کنونی (دستور title، text و gtext را نیز مشاهده نمایید)	xlabel/ylabel
ایجاد کردن یک بردار یا ماتریس با درایه‌های صفر	zeros
رسم خطوط ضریب میرایی ثابت (zeta) و فرکانس طبیعی ثابت (Wn) (دستور sgrid و sigrid را نیز مشاهده نمایید)	zgrid

[۴۰] Minimal Realization

پیوست دوم: تبدیل فرم نمایش سیستم

فهرست مطالب بخش

- تبدیل متغیر سیستم
- فضای حالت به تابع تبدیل
- صفرها در بینهایت
- تابع تبدیل به فضای حالت
- فضای حالت به صفر/قطب و تابع تبدیل به صفر/قطب
- صفر/قطب به فضای حالت و صفر/قطب به تابع تبدیل

یک سیستم دینامیکی اغلب با یکی از سه راه زیر تعریف می‌گردد:

۱. به وسیله‌ی مجموعه‌ای از معادلات فضای حالت و ماتریس‌های مربوط به آن

۲. به وسیله‌ی تابع تبدیل و استفاده از متغیر نمادین s و چندجمله‌ای‌های صورت و مخرج

۳. به وسیله‌ی لیستی از قطب‌ها و صفرها و بهره‌ی متناظر با آن

بعضی مواقع بهتر است تا نمایش سیستم را بین سه روش گفته شده، تبدیل نمود. متلب می‌تواند این تبدیلات را به سرعت و سادگی انجام دهد.

تبدیل متغیر سیستم

فرض کنید تابع تبدیلی به فرم زیر داریم:

$$G(s) = \frac{2s+1}{4s^2+3s+2} \tag{۱}$$

این معادله را به شکل زیر می‌توان به فرم تابع تبدیل نمایش داد:

```
s = tf('s');
G = (2*s+1)/(4*s^2+3*s+2)
```

```
G =


      2 s + 1

  ---------------

  4 s^2 + 3 s + 2



Continuous-time transfer function.
```

یا به طور مشابه با مشخص نمودن ضرایب چندجمله‌ای صورت و مخرج به شکل برداری، داریم:

```
num = [2 1];
```

```
den = [4 3 2];

G = tf(num,den)
```

```
G =


      2 s + 1

  ---------------

  4 s^2 + 3 s + 2


Continuous-time transfer function.
```

می‌توان از متغیر سیستم G برای به دست آوردن مدل فضای حالت با کمک از دستور زیر استفاده نمود:

```
[A,B,C,D] = ssdata(G)
```

```
A =

   -0.7500   -0.5000

    1.0000         0

B =

     1

     0

C =

    0.5000    0.2500

D =

     0
```

این نمایش فضای حالت را می‌تواند در یک متغیر سیستم دیگر، مثلا با نام H، ذخیره نمود:

```
H = ss(A,B,C,D)
```

```
H =


  A =
```

```
          x1      x2
   x1   -0.75    -0.5
   x2       1       0

  B =
       u1
   x1   1
   x2   0

  C =
         x1     x2
   y1    0.5   0.25

  D =
       u1
   y1   0

Continuous-time state-space model.
```

برای استخراج مدل صفر-قطب-بهره با استفاده از این متغیر سیستم، دستور زیر را اجرا می‌نماییم:

```
[z,p,k] = zpkdata(H,'v')
```

```
z =
   -0.5000
p =
  -0.3750 + 0.5995i
  -0.3750 - 0.5995i
k =
    0.5000
```

پارامتر 'v' در دستور بالا، سبب می‌شود تا مقادیر صفر و قطب به صورت برداری برگردانده شوند که برای سیستم‌های SISO مفید می‌باشد. حال می‌توانیم یک متغیر سیستم دیگر با نام K را به نمایش zpk اختصاص دهیم:

```
K = zpk(z,p,k)
```

```
K =


      0.5 (s+0.5)

  -------------------

  (s^2 + 0.75s + 0.5)



Continuous-time zero/pole/gain model.
```

در نهایت با استفاده از این متغیر سیستم می‌توانیم فرم تابع تبدیل سیستم را به دست آوریم:

```
[num,den] = tfdata(K,'v')
```

```
num =

         0    0.5000    0.2500

den =

    1.0000    0.7500    0.5000
```

با استفاده از این دستور مشاهده می‌نماییم که فرم تابع تبدیل به دست آمده مشابه‌ی تابع تبدیل اولیه قبل از انجام تبدیلات فوق می‌باشد (هرچند که ضرایب صورت و مخرج در ۴ ضرب شده است).

فضای حالت به تابع تبدیل

علاوه بر استفاده از متغیر سیستم برای تبدیل فرم‌های نمایش سیستم به یکدیگر، می‌توان مستقیما این تبدیل را نیز انجام داد.

فرض کنید مجموعه‌ای از معادلات فضای حالت داریم که می‌خواهیم معادل تابع تبدیل آنها را به دست آوریم. برای اینکار از دستور `ss2tf` استفاده می‌نماییم:

```
s = tf('s');

G = (2*s+1)/(4*s^2+3*s+2);

[A,B,C,D] = ssdata(G);

[num,den] = ss2tf(A,B,C,D)
```

```
num =
```

```
         0    0.5000    0.2500

den =

    1.0000    0.7500    0.5000
```

برای مثال فرض کنید مجموعه‌ی معادلات حالت به شکل زیر است:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{k}{m} & -\frac{b}{m}\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m}\end{bmatrix}F(t) \qquad (۲)$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\end{bmatrix} \qquad (۳)$$

پارامترهای سیستم برابر است با:

- m = 100 kg
- b = 50 Ns/m
- u = 500 N

اگر بخواهیم فرم نمایش این سیستم را به فرم تابع تبدیل تغییر دهیم، ام‌فایل زیر را اجرا می‌نماییم:

```
A = [0   1

     0   -0.05];



B = [0;

     0.001];



C = [0   1];



D = 0;



[num,den]=ss2tf(A,B,C,D)
```

```
num =

   1.0e-03 *

         0    1.0000         0

den =

    1.0000    0.0500         0
```

چند نکته در رابطه با استفاده از دستور `ss2tf`:

- صورت کسر (num) به تعداد خروجی‌های سیستم، دارای ردیف می‌باشد (یا به اندازه‌ی تعداد ردیف‌های ماتریس C)
- صورت و مخرج به شکل توان‌های نزولی s نمایش داده می‌شوند.
- باید صورت و مخرج را بررسی نمود تا در صفرهای موجود در بینهایت، تابع تبدیل خطا نداشته باشد. در بخش بعد این مورد توضیح داده شده است.

صفرهای در بینهایت

آخرین نکته‌ی گفته شده در بالا نیاز به توضیحات تکمیلی دارد. هنگامی می‌گوییم یک سیستم دارای صفرهایی در بینهایت است که اگر مقدار تابع تبدیل وقتی $s \to \infty$ برابر با صفر باشد. این اتفاق زمانی می‌افتد که تعداد قطب‌های سیستم بیشتر از تعداد صفرهای آن باشد. این مورد در نمودار مکان هندسی ریشه‌ها به صورت مجانب‌هایی که به بینهایت می‌روند نمود پیدا می‌کند (تعداد مجانب‌ها برابر است با تعداد صفرهای در بینهایت). بعضی مواقع متلب این صفرهای در بینهایت را به شکل عددهای کراندار بسیار بزرگ نشان می‌دهد. هنگامی که این اتفاق می‌افتد، بعضی از ضرایب موجود در صورت که باید صفر باشند، به صورت عددهای بسیار کوچکی نشان داده می‌شوند. هرچند که این مسئله شاید خیلی مهم نباشد اما در هنگام استفاده از تابع تبدیل مشکلاتی را به وجود می‌آورد. در نتیجه باید همواره تابع تبدیل را بررسی نمود و اگر عددی با مقدار 0.0000 پدیدار گردید باید آنرا به صورت دستی به مقدار مطلق 0 تبدیل نمود.

به یک مثال مرتبط با این موضوع توجه نمایید:

```
A = [0    1      0       0

     0 -0.1818 2.6727 0

     0    0      0       1

     0 -4.5454 31.1818 0];



B = [0

     1.8182

     0

     4.5455];



C = [1  0  0  0

    0  0  1  0];



D = [0

     0];
```

```
[num,den]=ss2tf(A,B,C,D)
```

```
num =

         0         0    1.8182    0.0000  -44.5460

         0         0    4.5455   -7.4381    0.0000

den =

    1.0000    0.1818  -31.1818    6.4796         0
```

اگر به بردار صورت توجه کنید، درایه‌های اول و دوم هر ردیف 0 می‌باشد، اما درایه‌ی چهارم ردیف اول و درایه‌ی آخر ردیف دوم برابر 0.000 است. اگر با دقت بیشتری به این درایه‌ها دقت کنیم متوجه می‌شویم که آنها در واقع صفر نمی‌باشند و یک عدد بسیار کوچک هستند. برای دیدن مقادیر دقیق آنها دستور num(1,4) یا num(2,5) را در محیط متلب اجرا کنید. در اینصورت به ترتیب مقادیر 3.2298e-15 و 3.3032e-15- نمایش داده می‌شود. این مشکل عددی را می‌توان با اضافه نمودن دو دستور زیر بعد از دستور ss2tf حل نمود:

```
num(1,4) = 0;

num(2,5) = 0;

num
```

```
num =

         0         0    1.8182         0  -44.5460

         0         0    4.5455   -7.4381         0
```

حال تمامی اعداد کوچک با صفر جایگزین شدند. همواره تابع تبدیل خود را بررسی و قبل از شروع روند طراحی، آنرا تحلیل نمایید.

تابع تبدیل به فضای حالت

معکوس دستور ss2tf، دستور tf2ss می‌باشد که یک سیستم به فرم تابع تبدیل را به فرم فضای حالت تبدیل می‌نماید. از این دستور به شکل زیر استفاده می‌گردد:

```
[A,B,C,D] = tf2ss(num,den)
```

```
A =

   -0.1818   31.1818   -6.4796         0

    1.0000         0         0         0

         0    1.0000         0         0

         0         0    1.0000         0
```

```
B =

     1
     0
     0
     0

C =

         0    1.8182         0  -44.5460
         0    4.5455   -7.4381         0

D =

     0
     0
```

یک نکته‌ی مهم در اینجا این است که اگرچه برای نمایش سیستم یک تابع تبدیل منحصر به فرد وجود دارد اما می‌توان چندین معادلات فضای حالت برای نمایش همان سیستم به دست آورد. دستور `tf2ss` ماتریس‌های فضای حالت را به فرم کنترل کانونی[41] باز می‌گرداند. در نتیجه اگر یک مجموعه از معادلات فضای حالت داشته باشید و آنرا به تابع تبدیل تغییر دهید، با تبدیل دوباره‌ی تابع تبدیل به فرم فضای حالت، معادلات فضای حالت متفاوت می‌باشند مگر اینکه معادلات اولیه‌ی فضای حالت به فرم کنترل کانونی باشد.

برای مثال صورت و مخرج ساخته شده در بالا را دوباره به فضای حالت تبدیل می‌نماییم:

```
[A,B,C,D] = tf2ss(num,den)
```

```
A =

   -0.1818   31.1818   -6.4796         0
    1.0000         0         0         0
         0    1.0000         0         0
         0         0    1.0000         0

B =

     1
     0
     0
```

[41] Control Canonical Form

```
     0

C =

         0    1.8182         0  -44.5460

         0    4.5455   -7.4381         0

D =

     0

     0
```

واضح است که این مجموعه از ماتریس‌ها، همان ماتریس‌های اولیه نمی‌باشند اما رفتار ورودی و خروجی سیستم مشابه سیستم سابق است. بینهایت راه برای نمایش یک تابع تبدیل به فرم فضای حالت وجود دارد و متلب فرم کنترل کانونی را انتخاب می‌کند. در هر نمایش فضای حالت، متغیرهای حالت سیستم دارای معانی یکسان نمی‌باشند.

فضای حالت به صفر-قطب و تابع تبدیل به صفر-قطب

برای نمایش یک سیستم دینامیکی روش دیگری به نام مدل قطب-صفر-بهره وجود دارد. این مدل اساسا همان مدل تابع تبدیل است اما چندجمله‌ای‌های صورت و مخرج به صورت ریشه‌های آنها فاکتورگیری می‌شوند. شکل کلی این فرم به صورت زیر است:

$$G(s) = k\frac{(s-z_1)(s-z_2)\cdots(s-z_m)}{(s-p_1)(s-p_2)\cdots(s-p_n)} \quad (۴)$$

به یاد داشته باشید که برای نمایش صحیح تابع تبدیل، تعداد قطب‌ها (n) باید بزرگتر یا مساوی تعداد صفرها (m) باشد. متلب می‌تواند این تبدیل را چه از فضای حالت و چه از تابع تبدیل به نمایش قطب-صفر-بهره انجام دهد.

اگر مدل تابع تبدیل داشته باشیم:

```
[z,p,k] = tf2zp(num,den)
```

```
z =

    4.9498         0

   -4.9498    1.6364

p =

         0

   -5.7753

    5.3851

    0.2083

k =

    1.8182
```

```
    4.5455
```

اگر مدل فضای حالت داشته باشیم:

```
[z,p,k] = ss2zp(A,B,C,D)
```

```
z =

    4.9498         0

   -4.9498    1.6364

p =

         0

   -5.7753

    5.3851

    0.2083

k =

    1.8182

    4.5455
```

هر دوی این دستورات باید سه متغیر را برگرداند: z، p و k. متغیر z تمامی صفرها را به صورت ستونی باز می‌گرداند. به ازای هر ردیف از صورت تابع تبدیل یا هر خروجی سیستم (y)، یک ستون در متغیر z وجود دارد. متغیر p تمامی قطب‌ها را به صورت ستونی باز می‌گرداند. متغیر k نیز ستونی از مقادیر بهره را باز می‌گرداند. این متغیر به تعداد خروجی‌های سیستم، دارای ردیف می‌باشد. برای مثال با استفاده از مدل فضای حالت و تابع تبدیل تعریف شده در بالا، یکی از دو اِم‌فایل زیر را اجرا نمایید:

```
num = [1.8182    0       -44.5460;

           4.5455   -7.4373   0];



den = [1 0.1818 -31.1818 6.4786 0];



[z,p,k] = tf2zp(num,den)
```

```
z =

    4.9498         0

   -4.9498    1.6362
```

```
p =

         0

   -5.7753

    5.3851

    0.2083

k =

    1.8182

    4.5455
```

یا:

```
A = [0      1            0                0

         0     -0.1818    2.6727          0

         0      0          0              1

     0     -4.545     31.1818         0];


B = [0

     1.8182

     0

     4.5455];


C = [1  0  0  0

     0  0  1  0];


D = [0

     0];


[z,p,k] = ss2zp(A,B,C,D)
```

```
z =
```

```
  -4.9498         0
   4.9498    1.6362
p =
        0
  -5.7753
   0.2083
   5.3851
k =
   1.8182
   4.5455
```

به دلیل اینکه دو ستون از صفر وجود دارد، ماتریس k دارای دو ردیف می‌باشد (هر ردیف برای یک ستون z).

صفر-قطب به فضای حالت و صفر-قطب به تابع تبدیل

همانطور که ممکن است حدس زده باشید، اگر سیستمی داشته باشیم که به روش صفر-قطب نمایش داده شده باشد، می‌توانیم آن سیستم را به فرم فضای حالت یا تابع تبدیل نیز نمایش دهیم. برای به دست آوردن مدل فضای حالت، دستور زیر را وارد نمایید:

```
[A,B,C,D] = zp2ss(z,p,k)
```

```
A =
  -0.1818   31.1818   -6.4786         0
   1.0000         0         0         0
        0    1.0000         0         0
        0         0    1.0000         0
B =
     1
     0
     0
     0
C =
        0    1.8182         0  -44.5460
```

```
         0    4.5455   -7.4373         0

D =

     0

     0
```

دوباره باید متذکر شویم که بیش از یک مجموعه از ماتریس‌های فضای حالت برای تعریف یک سیستم وجود دارد. ماتریس‌های فضای حالتی که با استفاده از این دستور به دست می‌آیند به فرم کنترل کانونی می‌باشند. یعنی با دستور `tf2ss` نیز همین ماتریس‌ها به دست خواهد آمد.

برای اینکه مدل قطب-صفر را به فرم مدل تابع تبدیل نمایش دهیم، دستور زیر را وارد کنید:

```
[num,den] = zp2tf(z,p,k)
```

```
num =

         0         0    1.8182         0  -44.5460

         0         0    4.5455   -7.4373         0

den =

    1.0000    0.1818  -31.1818    6.4786         0
```

تابع تبدیل به دست آمده همان تابع تبدیلی می‌باشد که در ابتدا آنرا در نظر گرفته بودیم.

پیوست سوم: شناسایی سیستم

فهرست مطالب بخش

- تخمین مرتبه‌ی سیستم
- شناسایی سیستم با استفاده از پاسخ پله
- شناسایی سیستم با استفاده از دیاگرام بودی
- شناسایی پارامترهای سیستم

پارامترهای سیستم از جمله ضریب میرایی، فرکانس طبیعی و بهره‌ی DC را می‌توان از پاسخ پله یا دیاگرام بودی به دست آورد.

تخمین مرتبه‌ی سیستم

مرتبه‌ی سیستم و درجه‌ی نسبی آن را می‌توان با استفاده از پاسخ پله یا دیاگرام بودی سیستم تخمین زد. به تفاوت بین مرتبه‌ی مخرج و مرتبه‌ی صورت یک تابع تبدیل سیستم، درجه‌ی نسبی سیستم گفته می‌شود. درجه‌ی نسبی سیستم کمترین مرتبه‌ای است که سیستم می‌تواند داشته باشد.

پاسخ پله

اگر پاسخ یک سیستم به ورودی پله‌ی غیر صفر، در لحظه‌ی t=0 دارای شیب صفر باشد، آنگاه سیستم باید از درجه‌ی دوم یا بیشتر باشد زیرا درجه‌ی نسبی سیستم برابر ۲ یا بیشتر است.

اگر در پاسخ پله‌ی سیستم نوساناتی وجود داشته باشد، آنگاه سیستم باید یک سیستم زیر میرای مرتبه‌ی دوم یا بیشتر بوده و دارای درجه‌ی نسبی ۲ یا بیشتر می‌باشد.

دیاگرام بودی

نمودار فاز نشان‌دهنده‌ی خوبی برای مرتبه‌ی سیستم است. اگر فاز سیستم به کمتر از 90- درجه برسد آنگاه سیستم باید درجه دوم یا بیشتر باشد. درجه‌ی نسبی سیستم حداقل باید به بزرگی تعداد مضرب‌های 90- درجه که منحنی فاز به صورت مجانبی در پایین‌ترین به آن رسیده است، باشد.

شناسایی سیستم با استفاده از پاسخ پله

بهره‌ی DC

بهره‌ی DC یا K، نسبت پاسخ حالت ماندگار ورودی پله به بزرگی آن ورودی پله می‌باشد.

ضریب میرایی

برای یک سیستم زیر میرای درجه دوم، ضریب میرایی را می‌توان از رابطه‌ی درصد فراجهش به صورت زیر به دست آورد:

$$\zeta = \frac{-\ln(M_p)}{\sqrt{\pi^2 + \ln^2(M_p)}} \tag{۱}$$

در این رابطه M_p ماکزیمم درصد فراجهش بوده که مقدار آنرا می‌توان از نمودار پاسخ پله تخمین زد.

فرکانس طبیعی

فرکانس طبیعی (ω_n) یک سیستم زیر میرای درجه‌ی دوم را می‌توان از فرکانس طبیعی میرا (ω_d) به دست آورد که فرکانس طبیعی میرا را می‌توان از نمودار پاسخ پله و ضریب میرایی را از رابطه‌ی بالا به دست آورد:

$$\omega_n = \frac{\omega_d}{\sqrt{1-\zeta^2}} \tag{۲}$$

که در آن:

$$\omega_d = \frac{2\pi}{\Delta t} \tag{۳}$$

و Δt بازه‌ی زمانی بین دو قله‌ی متوالی در نمودار پاسخ پله می‌باشد.

شناسایی سیستم با استفاده از دیاگرام بودی

بهره‌ی DC

بهره‌ی DC یک سیستم را می‌توان از اندازه‌ی دیاگرام بودی در $s = 0$ محاسبه نمود:

$$K = 10^{\frac{M(0)}{20}} \tag{۴}$$

که $M(0)$ اندازه‌ی دیاگرام بودی در $j\omega = 0$ می‌باشد.

فرکانس طبیعی

ضریب میرایی یک را می‌توان با استفاده از بهره‌ی DC و اندازه‌ی دیاگرام بودی در فاز 90- درجه به دست آورد:

$$\zeta = \frac{\mathrm{K}}{2 \times 10^{\frac{\mathrm{M}(-90°)}{20}}} \tag{۵}$$

شناسایی پارامترهای سیستم

اگر نوع سیستم مشخص باشد، آنگاه پارامترهای فیزیک را می‌توان از معیارهای دینامیکی بالا تعیین نمود.

فرم کلی تابع تبدیل یک سیستم مرتبه اول به شکل زیر است:

$$G(s) = \frac{b}{s+a} = \frac{K}{\tau s + 1} \tag{۶}$$

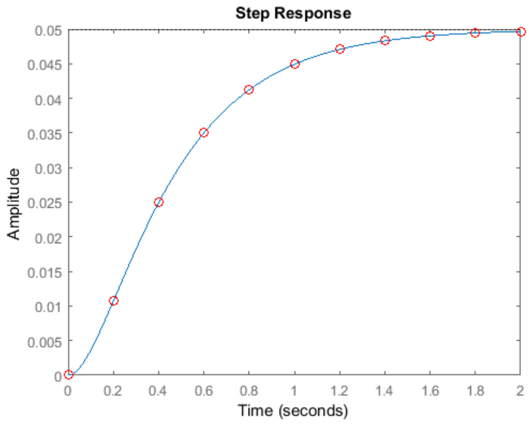
فرم کلی تابع تبدیل یک سیستم مرتبه دوم به شکل زیر است:

$$G(s) = \frac{a}{s^2 + bs + c} = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{۷}$$

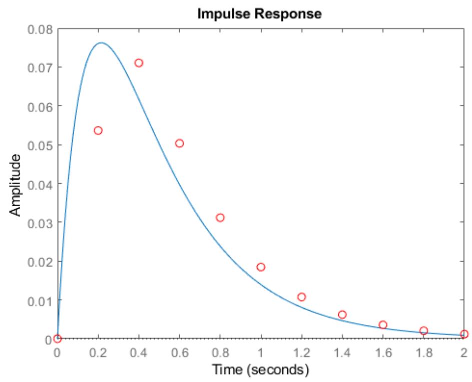
پیوست چهارم: تاخیر ناشی از نگهدارنده

یکی از تبعات مهم استفاده از سیستم کنترل دیجیتالی، تاخیر ناشی از نگهدارنده می‌باشد. برای شروع دستورات زیر را در یک اِم‌فایل وارده کرده و آنرا اجرا کنید:

```
s = tf('s');
sys = 1/(s^2+10*s+20);

Ts=0.2;
sys_d = c2d(sys,Ts);

step(sys,2)             %plots continuous output response
hold on
[x,t]=step(sys_d,2);
plot(t,x,'ro')          %plots discrete output response
hold off
```

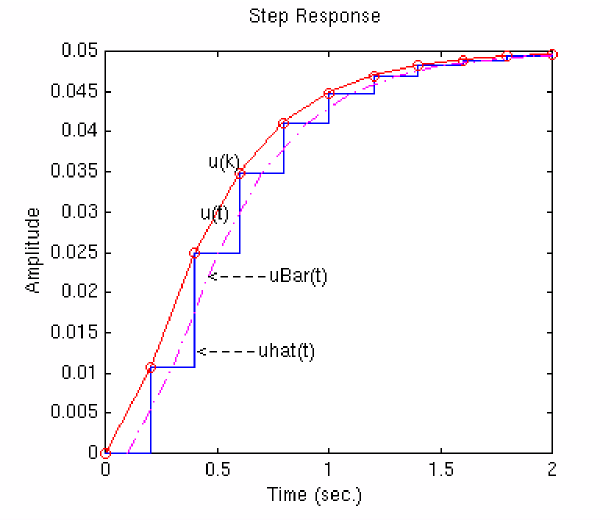


با توجه به این نمودار مشاهده می‌نماییم که در هر نمونه‌ی زمانی، پاسخ گسسته کاملا منطبق با پاسخ پیوسته می‌باشد. این امر به دلیل اینکه ورودی داده شده تابع پله‌ی تکه‌ای ثابت بوده است صادق می‌باشد. اگر ورودی تابع متغیر با زمان پیوسته باشد آنگاه این تطابق دیگر صورت نخواهد گرفت. برای تغییر ورودی، در کد بالا، ورودی step را به ورودی impulse تبدیل نمایید. با اجرای دوباره‌ی این ام‌فایل، نمودار زیر به دست خواهد آمد:



از این نمودار می‌توان مشاهده نمود که پاسخ گسسته با پاسخ پیوسته مطابق نمی‌باشد. پاسخ گسسته دارای تاخیر مشخصی نسبت به پاسخ پیوسته می‌باشد.

حتی اگر در زمان‌های نمونه‌برداری، پاسخ گسسته با پاسخ پیوسته مانند شکل زیر مطابق باشد:



سیگنال میانگین uBar(t) نسبت به سیگنال پیوسته‌ی u(t) به اندازه‌ی Ts/2 تاخیر دارد. اگرچه با کاهش زمان نمونه‌برداری (Ts بر حسب sec/sample) این تاخیر کمتر شده و سیگنال uBar(t) به سیگنال پیوسته‌ی u(t) نزدیکتر خواهد شد.

پیوست پنجم: خطای حالت ماندگار دیجیتال

فهرست مطالب بخش

- به دست آوردن خطای حالت ماندگار به ورودی پله‌ی واحد
- به دست آوردن خطای حالت ماندگار به ورودی ضربه

در طراحی سیستم‌های پیوسته، اغلب از قضیه مقدار نهایی برای به دست آوردن خطای حالت ماندگار سیستم استفاده می‌نماییم:

$$\lim_{t\to\infty} x(t) = x_{ss} = \lim_{s\to 0} sX(s) \tag{۱}$$

لازم به ذکر است که این قضیه تنها در صورتی که قطب‌های $sX(s)$ دارای قسمت حقیقی منفی باشند صادق است. برای سیستم‌های گسسته نیز قضیه‌ی مقدار نهایی وجود دارد. قضیه‌ی مقدار نهایی برای سیستم‌های گسسته به شکل زیر تعریف می‌گردد:

$$\lim_{k\to\infty} x(k) = x_{ss} = \lim_{z\to 1}(1-z^{-1})X(z) \tag{۲}$$

که در این صورت لازم است تا تمام قطب‌های $(1-z^{-1})X(z)$ در داخل دایره‌ی واحد قرار گرفته باشند.

برای مثال تابع تبدیل گسسته‌ی زیر را در نظر بگیرید:

$$\frac{X(z)}{U(z)} = \frac{z+0.5}{z^2-0.6z+0.3} \tag{۳}$$

ابتدا قطب‌های این تابع تبدیل را به دست می‌آوریم تا از قرار داشتن آنها در داخل دایره‌ی واحد اطمینان حاصل کنیم. این قطب‌ها را می‌توان به روش دستی و یا با دستور متلب زیر به دست آورد:

```
z = tf('z',-1);
sys_d = (z + 0.5)/(z^2 - 0.6*z + 0.3);
[p,z] = pzmap(sys_d)
```

```
p =

   0.3000 + 0.4583i

   0.3000 - 0.4583i

z =

  -0.5000
```

چون هر دو قطب این سیستم در داخل دایره‌ی واحد قرار دارد مجاز به استفاده از قضیه‌ی مقدار نهایی می‌باشیم.

به دست آوردن خطای حالت ماندگار به ورودی پله‌ی واحد

ورودی $U(z)$ را برابر ورودی پله‌ی واحد به شکل زیر قرار می‌دهیم:

$$U(z) = \frac{1}{(1-z^{-1})} \tag{۴}$$

در این صورت خروجی $X(z)$ برابر است با:

$$X(z) = \frac{(z+0.5)}{(z^2-0.6z+0.3)} \frac{1}{(1-z^{-1})} \quad (۵)$$

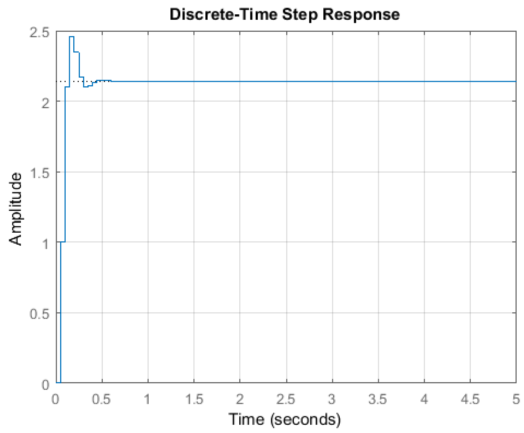
با استفاده از قضیه‌ی مقدار نهایی داریم:

$$x_{ss} = \lim_{z\to 1}(1-z^{-1}) \frac{(z+0.5)}{(z^2-0.6z+0.3)} \frac{1}{(1-z^{-1})} = 2.14 \quad (۶)$$

بنابراین خروجی حالت ماندگار سیستم گسسته‌ی بالا به ورودی پله‌ی واحد برابر 2.14 می‌باشد. در نتیجه این مقدار معادل خطای حالت ماندگار ۱۱۴٪ می‌باشد.

برای بررسی صحت این مقدار، پاسخ پله‌ی سیستم را رسم می‌نماییم. یک ام‌فایل جدید ایجاد کرده و دستورات زیر را در آن وارد کنید. با اجرای این ام‌فایل، پاسخ پله‌ی سیستم مشابه شکل زیر رسم می‌گردد:

```
Ts = .05;

z = tf('z',Ts);
sys_d = (z + 0.5)/(z^2 - 0.6*z + 0.3);

step(sys_d,5);
grid
title('Discrete-Time Step Response')
```



همانطور که انتظار می‌رفت مقدار خطای حالت ماندگار 2.14 می‌باشد.

به دست آوردن خطای حالت ماندگار به ورودی ضربه

حال ورودی $U(z)$ را از ورودی پله به ورودی ضربه تغییر می‌دهیم:

$$U(z) = 1 \quad (۷)$$

با اعمال قضیه‌ی مقدار نهایی، نتیجه‌ی زیر به دست می‌آید:

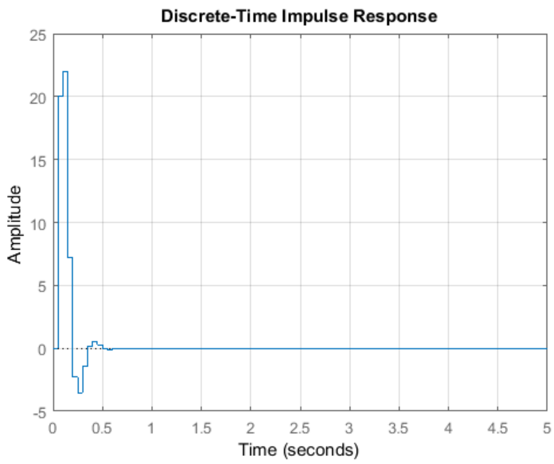
$$x_{ss} = \lim_{z\to 1}(1-z^{-1}) \frac{(z+0.5)}{(z^2-0.6z+0.3)} (1) = 0 \quad (۸)$$

بنابراین خروجی حالت ماندگار سیستم بالا به ورودی ضربه برابر صفر می‌باشد.

دستور step را در ام‌فایل بالا به دستور impulse تغییر دهید و آنرا دوباره اجرا نمایید تا پاسخ زیر به دست آید:

```
Ts = .05;
```

```
z = tf('z',Ts);
sys_d = (z + 0.5)/(z^2 - 0.6*z + 0.3);

impulse(sys_d,5);
grid
title('Discrete-Time Impulse Response')
```



از این نمودار همانطور که پیش‌بینی می‌شد، خروجی حالت ماندگار به ورودی ضربه‌ی واحد برابر صفر به دست آمد.

پیوست ششم: موقعیت قطب‌های گسسته و پاسخ گذرا

فهرست مطالب بخش

- میرایی کم
- میرایی متوسط
- میرایی زیاد

در این بخش به جزییات بیشتری از موقعیت قطب‌های تابع تبدیل گسسته و ارتباط آن با پاسخ زمانی سیستم می‌پردازیم.

میرایی کم

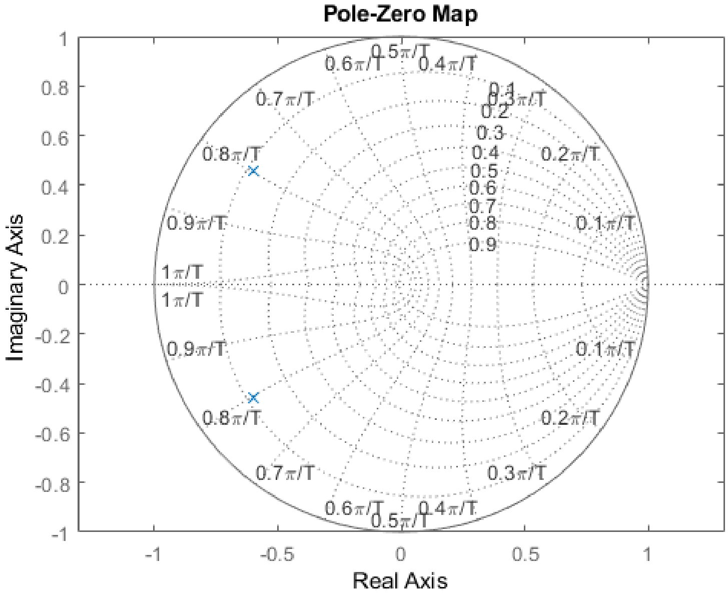
ابتدا تابع تبدیل گسسته‌ی زیر را با $\zeta = 0.1$ و $\omega_n = 0.8\pi/T$ در نظر بگیرید:

$$\frac{Y(z)}{F(z)} = \frac{1}{z^2 + 1.2z + 0.57} \qquad (۱)$$

دستورات زیر قطب‌های این تابع تبدیل را به دست آورده و رسم می‌نماید. این دستورات را در یک ام‌فایل جدید وارد کرده و آنرا اجرا کنید. بعد از اجرای آن نمودار قطب-صفر نمایش داده می‌شود:

```
T = .05;

z = tf('z',T);
sys = 1/(z^2+1.2*z+0.57);
[poles,zeros] = pzmap(sys)
pzmap(sys)

axis equal
zgrid
```

```
poles =

  -0.6000 + 0.4583i

  -0.6000 - 0.4583i

zeros =

  0×1 empty double column vector
```



با توجه به این نمودار، قطب‌ها به گونه‌ای قرار گرفته‌اند که دارای فرکانس طبیعی $0.8\pi/T$ (T بازه‌ی نمونه‌برداری بر حسب sample/sec) و ضریب میرایی حدودا 0.1 می‌باشند. اگر فرض کنیم که بازه‌ی نمونه‌برداری برابر 1/20 sec/sample باشد، با استفاده از سه معادله‌ی زیر می‌توان زمان نمو، زمان نشست و ماکزیمم درصد فراجهش را به دست آورد:

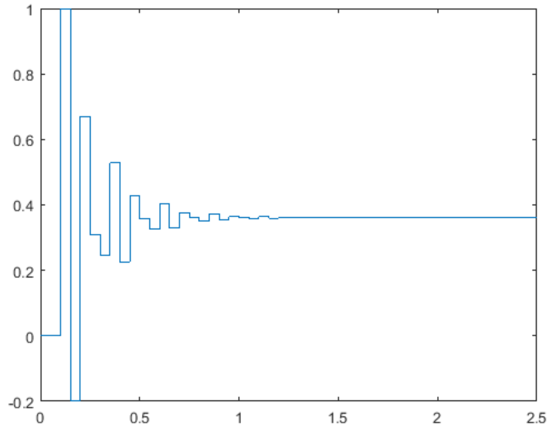
$$T_s = \frac{4}{\zeta\omega_n} \tag{۲}$$

$$T_r = \frac{1.8}{\omega_n} \tag{۳}$$

$$M_p = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} \tag{۴}$$

مقدار زمان نمو برابر 0.035 ثانیه، زمان نشست برابر 0.8 ثانیه و ماکزیمم درصد فراجهش برابر ۷۰٪ می‌باشد. برای استفاده از این معادلات باید فرکانس طبیعی (ω_n) را از واحد rad/sample به rad/sec تبدیل نمود. همچنین این روابط برای سیستم مرتبه دوم زیر میرا و بدون صفر صادق می‌باشد. برای صحت این نتایج، پاسخ پله‌ی سیستم را به دست می‌آوریم. دستورات زیر را به ام‌فایل بالا اضافه نموده و آنرا دوباره اجرا نمایید تا پاسخ پله به دست آید:

```
[x,t] = step(sys,2.5);

stairs(t,x)
```



از نمودار می‌توان دریافت که زمان نمو، زمان نشست و فراجهش سیستم تقریبا برابر مقدار پیش‌بینی شده است.

میرایی متوسط

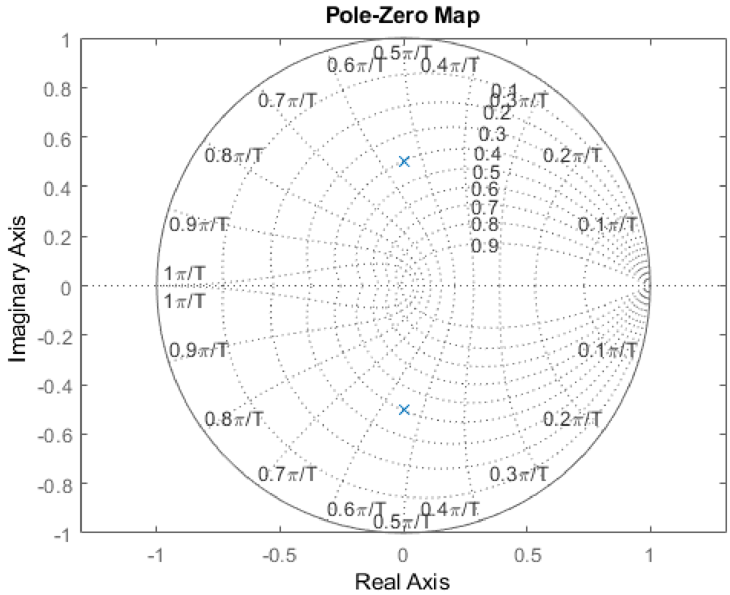
حال تابع تبدیل گسسته‌ی زیر را با $\zeta = 0.4$ و $\omega_n = (11\pi)/(20T)$ در نظر بگیرید:

$$\frac{Y(z)}{F(z)} = \frac{1}{z^2 + 0.25} \qquad (۵)$$

مراحل قسمت قبل را دوباره تکرار می‌نماییم. یک ام‌فایل جدید باز نموده و دستورات زیر را در آن وارد و اجرا کنید تا نمودار قطب‌ها نمایش داده شود:

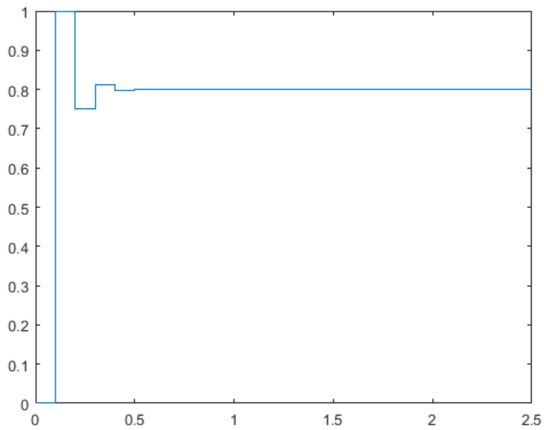
```
T = .05;

z = tf('z',T);
sys = 1/(z^2 + 0.25);
[poles,zeros] = pzmap(sys)
pzmap(sys)

axis equal
zgrid
```

```
poles =

   0.0000 + 0.5000i

   0.0000 - 0.5000i

zeros =

  0×1 empty double column vector
```



از نمودار قطب‌ها متوجه می‌شویم که قطب ها در فرکانس طبیعی $(11\pi)/(20T)$ و ضریب میرایی حدودا 0.4 قرار گرفته‌اند. بازه‌ی نمونه‌برداری را برابر 1/20 ثانیه (مانند قبل) در نظر گرفته و با استفاده از سه معادله‌ی بالا مشخصات پاسخ پله را به دست می‌آوریم. زمان نمو برابر 0.05 ثانیه، زمان نشست برابر 0.3 ثانیه و ماکزیمم درصد فراجهش برابر ۲۵٪ به دست می‌آید. برای بررسی بیشتر پاسخ پله را رسم می‌نماییم. دستورات زیر را به امفایل بالا اضافه کرده و آنرا اجرا کنید تا پاسخ پله رسم شود:

```
[x,t] = step(sys,2.5);

stairs(t,x)
```



دوباره مشاهده می‌شود که پاسخ پله دارای زمان نمو، زمان نشست و فراجهش تقریبا مورد انتظار می‌باشد.

میرایی زیاد

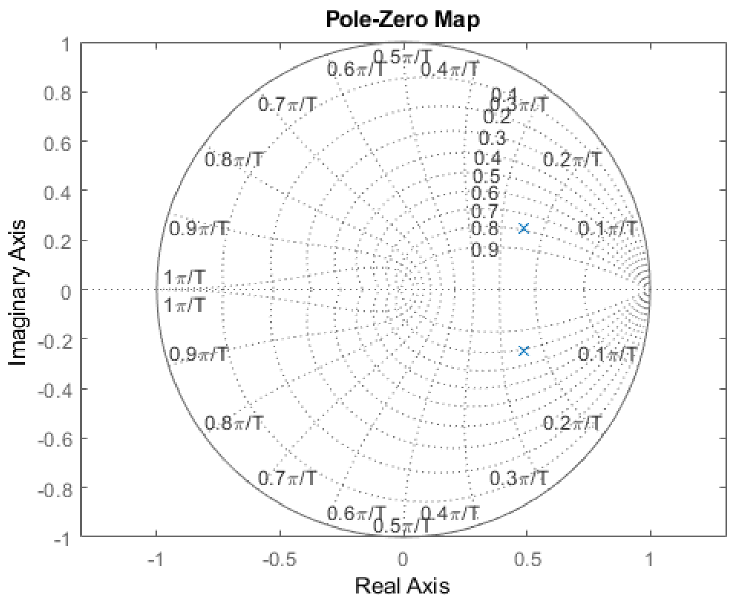
برای مثال آخر، تابع تبدیل گسسته‌ی زیر را با $\zeta = 0.8$ و $\omega_n = \pi/(4T)$ در نظر بگیرید:

$$\frac{Y(z)}{F(z)} = \frac{1}{z^2 - 0.98z + 0.3} \tag{۶}$$

مشابه قبل، دستورات زیر را در یک ام‌فایل جدید وارد کرده و آنرا اجرا کنید تا نمودار قطب-صفر نمایش داده شود:

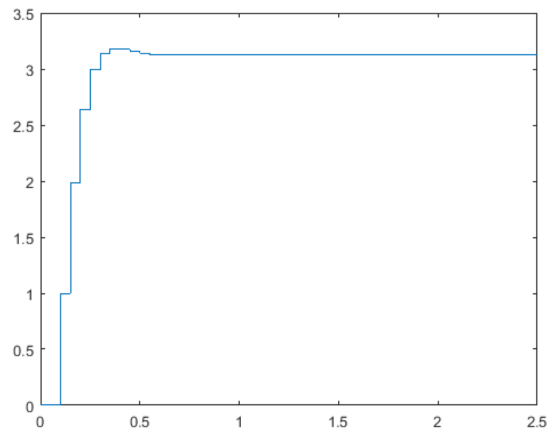
```
T = .05;

z = tf('z',T);
sys = 1/(z^2 - 0.98*z + 0.3);
[poles,zeros] = pzmap(sys)
pzmap(sys)

axis([-1 1 -1 1])
zgrid
```

```
poles =

   0.4900 + 0.2447i

   0.4900 - 0.2447i

zeros =

  0×1 empty double column vector
```



از این نمودار می‌توان مشاهده کرد که قطب‌ها در فرکانس طبیعی $\pi/(4T)$ و ضریب میرایی تقریبا 0.8 قرار گرفته‌اند. با فرض زمان نمونه بردار 1/20 ثانیه، این سیستم دارای زمان نمو 0.1 ثانیه، زمان نشست 0.3 ثانیه و فراجهش ۱٪ می‌باشد. با اضافه کردن دستورات زیر به ام‌فایل بالا و مشاهده‌ی نمودار پاسخ پله، از نتایج به دست آمده اطمینان حاصل می‌نماییم:

```
[x,t] = step(sys,2.5);

stairs(t,x)
```



نمودار رسم شده تقریبا دارای خصوصیات محاسبه شده است.

با استفاده از این سه مثال ثابت کردیم که می‌توان با استفاده از موقعیت قطب‌ها، تخمینی از پاسخ گذرای سیستم داشته باشیم. این تحلیل خصوصا برای طراحی به کمک مکان هندسی ریشه‌ها که هدف ما جایدهی قطب‌های حلقه بسته برای دستیابی به پاسخ مطلوب است، مناسب می‌باشد.

پیوست هفتم: طراحی جبران‌سازهای پیش‌فاز و پس‌فاز

فهرست مطالب بخش

از جبران‌سازهای پیش‌فاز و پس‌فاز به طور گسترده در زمینه‌ی کنترل استفاده می‌گردد. جبران‌ساز پیش‌فاز می‌تواند پایداری را افزایش داده و پاسخ سیستم را سرعت دهد؛ در حالیکه جبران‌ساز پس‌فاز می‌تواند خطای حالت ماندگار را کاهش دهد (اما حذف نمی‌کند). بسته به تاثیر مورد نیاز، یک یا چند جبران‌ساز پیش‌فاز و پس‌فاز به صورت‌های مختلف استفاده می‌گردد.

جبران‌سازی‌های پیش‌فاز، پس‌فاز و پیش-پس‌فاز معمولا برای سیستم‌های به فرم تابع تبدیل طراحی می‌گردند. در **پیوست دوم: تبدیل فرم نمایش سیستم** به تبدیل سیستم‌ها به یکدیگر اشاره شده است.

جبران‌ساز پیش‌فاز به کمک مکان هندسی ریشه‌ها

جبران‌ساز پیش‌فاز مرتبه اول C(s) را می‌توان با استفاده از مکان هندسی ریشه‌ها طراحی نمود. یک جبران‌ساز پیش‌فاز به فرم مکان هندسی ریشه‌ها به صورت زیر تعریف می‌شود:

$$C(s) = K_c \frac{(s - z_0)}{(s - p_0)} \qquad (۱)$$

در جبران‌ساز پیش‌فاز بالا، باید اندازه‌ی z_0 کوچکتر از اندازه‌ی p_0 باشد. جبران‌ساز پیش‌فاز شاخه‌های مکان هندسی ریشه‌ها را به سمت چپ صفحه‌ی مختلط متمایل می‌کند. به همین دلیل سبب بهبود پایداری سیستم و افزایش سرعت پاسخ سیستم می‌گردد.

برای انجام این کار، به یاد داریم که در به دست آوردن مجانب‌های مکان هندسی ریشه‌ها که در بینهایت به صفر میل می‌کرد از معادله‌ی زیر برای تعیین نقطه‌ی تماس مجانب با محور حقیقی استفاده می‌گردید:

$$a = \frac{\Sigma(poles) - \Sigma(zeros)}{(\#poles) - (\#zeros)} \qquad (۲)$$

زمانی که جبران‌ساز پیش‌فاز به سیستم اضافه گردد، محل این تقاطع عدد منفی بزرگتری خواهد شد. اختلاف تعداد صفرها و قطب‌ها ثابت می‌ماند (زیرا یک قطب و صفر اضافه گردیده است) اما قطب اضافه شده از صفر اضافه شده، منفی‌تر (از لحاظ اندازه بزرگتر) است. بنابراین نتیجه‌ی این جبران‌ساز پیش‌فاز این است که محل تقاطع مجانب‌ها را به سمت چپ صفحه‌ی مختلط حرکت می‌دهد و تمام مکان هندسی ریشه‌ها به سمت چپ نیز حرکت می‌کند. به همین دلیل ناحیه‌ی پایداری بزرگتر شده و سرعت پاسخ سیستم افزایش می‌یابد.

در متلب برای به کارگیری جبران‌ساز پیش‌فاز به فرم مکان هندسی ریشه‌ها، از دستورات زیر استفاده می‌نماییم (مقدار Kc، z و p از قبل تعریف شده است):

```
s = tf('s');

C_lead = Kc*(s-z)/(s-p);
```

می‌توانیم با دستور زیر جبران‌ساز C(s) را با سیستم P(s) ادغام نماییم:

```
sys_ol = C_lead*P;
```

جبران‌ساز پیش‌فاز به کمک پاسخ فرکانسی

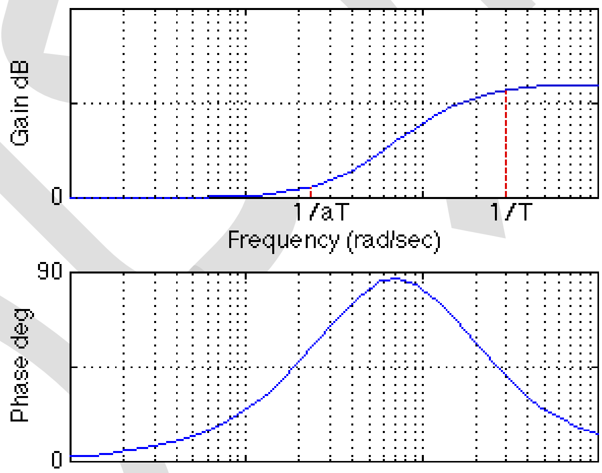
جبران‌ساز پیش‌فاز مرتبه اول را می‌توان با استفاده از روش پاسخ فرکانسی نیز طراحی نمود. جبران‌ساز پیش‌فاز به فرم پاسخ فرکانسی به صورت زیر نشان داده می‌شود:

$$C(s) = \frac{1 + aT_s}{1 + T_s} \quad [a > 1] \tag{۳}$$

این فرم نمایش مشابه‌ی فرم مکان هندسی ریشه‌ها وقتی p=1/T، z=1/aT و Kc=a است می‌باشد که در زیر دوباره آورده شده است:

$$C(s) = K_c \frac{(s - z_0)}{(s - p_0)} \tag{۴}$$

در طراحی به کمک پاسخ فرکانسی، جبران‌ساز پیش‌فاز، مقدار فاز مثبتی را در بازه‌ی فرکانسی 1/aT تا 1/T به سیستم اضافه می‌نماید. دیاگرام بودی جبران‌ساز پیش‌فاز C(s) به شکل زیر رسم می‌گردد:



دو فرکانس گوشه در فرکانس‌های 1/aT و 1/T می‌باشند؛ فاز اضافه شده به سیستم مابین این دو فرکانس اعمال می‌شود. بسته به مقدار a، مقدار فاز مثبت می‌تواند تا ۹۰ درجه باشد. اگر به بیش از ۹۰ درجه فاز نیاز باشد، از دو جبران‌ساز پیش‌فاز به صورت سری استفاده می‌شود. مقدار حداکثر فاز در مرکز فرکانس‌ها اضافه شده و از رابطه‌ی زیر محاسبه می‌گردد:

$$\omega_m = \frac{1}{T\sqrt{a}} \tag{۵}$$

رابطه‌ی که مقدار فرکانس ماکزیمم را حساب می‌کند به صورت زیر تعریف می‌گردد:

$$\sin\phi = \frac{a - 1}{a + 1} \tag{۶}$$

فرکانس اضافه شده سبب افزایش حد فاز گشته و در نتیجه پایداری سیستم را افزایش می‌دهد. برای طراحی این نوع جبران‌ساز، با توجه به مقدار فاز مورد نیاز برای رسیدن به حد فاز مطلوب، مقدار a را به دست آورده و با توجه به فرکانس گذر بهره‌ی جدید که فاز در آن نقطه اضافه می‌گردد، مقدار T را تعیین می‌نماییم.

تاثیر دیگر جبران‌ساز پیش‌فاز را می‌توان در نمودار اندازه مشاهده نمود. جبران‌ساز پیش‌فاز، بهره‌ی سیستم را در فرکانس‌های بالا افزایش می‌دهد (مقدار این بهره برابر a می‌باشد). این امر سبب افزایش فرکانس گذر می‌شود که عامل کاهش زمان نمو و نشست سیستم می‌گردد (اما ممکن است نویز فرکانس بالا را تقویت نماید).

در متلب برای پیاده‌سازی جبران‌ساز پیش‌فاز به فرم پاسخ فرکانسی، از دستور زیر استفاده می‌شود (مقادیر a و T از قبل تعیین شده‌اند):

```
s = tf('s');

C_lead = (1+a*T*s)/(1+T*s);
```

سپس می‌توان این جبران‌ساز را با دستور زیر با سیستم ادغام نمود:

```
sys_ol = C_lead*P;
```

جبران‌ساز پس‌فاز به کمک مکان هندسی ریشه‌ها

جبران‌ساز پس‌فاز مرتبه اول C(s) را می‌توان با استفاده از مکان هندسی ریشه‌ها طراحی نمود. یک جبران‌ساز پیش‌فاز به فرم مکان هندسی ریشه‌ها به صورت زیر تعریف می‌شود:

$$C(s) = \frac{(s - z_0)}{(s - p_0)} \qquad (۷)$$

این فرم مشابه فرم جبران‌ساز پیش‌فاز می‌باشد با این تفاوت که در اینجا اندازه‌ی z_0 بزرگتر از اندازه‌ی p_0 می‌باشد (و از بهره‌ی Kc صرف نظر شده است). جبران‌ساز پس‌فاز، شاخه‌های مکان هندسی ریشه‌ها را به سمت راست صفحه‌ی مختلط متمایل می‌کند. به همین دلیل صفر و قطب جبران‌ساز پس‌فاز اغلب نزدیک به هم انتخاب می‌شوند (معمولا در نزدیکی مبدا) تا تغییر زیادی بر روی پاسخ گذرا یا پایداری سیستم ایجاد نکند.

در اینجا دوباره با استفاده از رابطه‌ی محل تقاطع مجانب‌های مکان هندسی ریشه‌ها با محور حقیقی، حرکت مکان هندسی ریشه‌ها به سمت راست صفحه‌ی مختلط را بررسی می‌نماییم:

$$a = \frac{\Sigma(poles) - \Sigma(zeros)}{(\#poles) - (\#zeros)} \qquad (۸)$$

زمانی که یک جبران‌ساز پس‌فاز به سیستم اضافه می‌گردد، مقدار محل تقاطع، عدد منفی کوچکتری می‌شود. اختلاف تعداد قطب‌ها و صفرها ثابت است (یک صفر و قطب اضافه شده است) اما صفر اضافه شده از قطب اضافه شده منفی‌تر (از لحاظ اندازه بزرگتر) است. در نتیجه تاثیر جبران‌ساز پس‌فاز، حرکت محل تقاطع مجانب‌ها به سمت راست صفحه‌ی مختلط و حرکت تمام مکان هندسی ریشه‌ها به سمت راست می‌باشد.

در پیش از این هم ذکر شد که اغلب جبران‌ساز پس‌فاز به گونه‌ای طراحی می‌شود تا حداقل تغییر را در پاسخ گذرای سیستم ایجاد نماید زیرا این تغییر مضر می‌باشد. اگر جبران‌ساز پس‌فاز تاثیر چشمگیری را در پاسخ گذرا ایجاد نکند پس چه فایده‌ای دارد؟ جواب این سوال این است که جبران‌ساز پس‌فاز می‌تواند پاسخ حالت ماندگار سیستم را بهبود دهد. این بهبود به این صورت ایجاد می‌شود که جبران‌ساز پس‌فاز در فرکانس‌های بالا دارای بهره‌ی واحد و در فرکانس‌های پایین دارای بهره‌ی z_0/p_0 (که بزرگتر از ۱ است) می‌باشد. مقدار z_0/p_0 در ثابت‌های موقعیت، سرعت یا شتاب (Kp، Kv و Ka) ضرب شده و در نتیجه خطای حالت ماندگار به همان نسبت کاهش می‌یابد.

برای پیاده‌سازی جبران‌ساز پس‌فاز C(s)در متلب از دستور زیر استفاده می‌نماییم:

```
s = tf('s');
```

```
        C_lag = (s-z)/(s-p);
```

همچنین با دستور زیر، جبران‌ساز را با سیستم ادغام می‌نماییم:

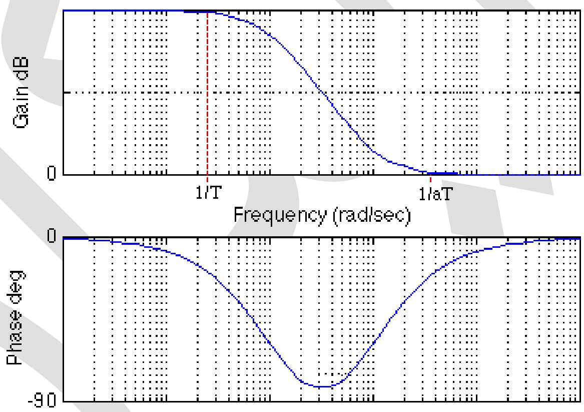
```
        sys_ol = C_lag*P;
```

جبران‌ساز پس‌فاز به کمک پاسخ فرکانسی

جبران‌ساز پس‌فاز مرتبه اول را می‌توان با استفاده از روش پاسخ فرکانسی نیز طراحی نمود. جبران‌ساز پیش‌فاز به فرم پاسخ فرکانسی به صورت زیر نشان داده می‌شود:

$$C(s) = \frac{1}{a}\left(\frac{1+aT_s}{1+T_s}\right) \quad [a>1] \qquad (۹)$$

جبران‌ساز پس‌فاز بسیار مشابه‌ی جبران‌ساز پیش‌فاز می‌باشد با این تفاوت که در اینجا مقدار a کمتر از ۱ می‌باشد. تفاوت اصلی آن این است که جبران‌ساز پس‌فاز مقدار فاز **منفی** را در بازه‌ی فرکانسی مشخص به سیستم اضافه می‌نماید اما جبران‌ساز پیش‌فاز مقدار فاز **مثبت** را در بازه‌ی فرکانسی مشخص به سیستم اضافه می‌کند. دیاگرام بودی جبران‌ساز پس‌فاز به شکل زیر رسم می‌گردد:



دو فرکانس گوشه برابر 1/T و 1/aT می‌باشد. تاثیر اصلی جبران‌ساز پس‌فاز در نمودار اندازه مشخص می‌شود. جبران‌ساز پس‌فاز در فرکانس‌های پایین بهره را اضافه کرده و مقدار این بهره برابر a می‌باشد. تاثیر این اضافه شدن بهره، کاهش خطای حالت ماندگار سیستم حلقه بسته با ضریب a می‌باشد. به دلیل اینکه در فرکانس‌های وسط و بالا مقدار بهره‌ی جبران‌ساز پس‌فاز برابر ۱ می‌باشد، پاسخ گذرا و پایداری سیستم بدون تغییر باقی می‌ماند.

عوارض جانبی جبران‌ساز پس‌فاز، اضافه کردن فاز منفی به سیستم مابین دو فرکانس گوشه می‌باشد. بسته به مقدار a، تا 90- درجه فاز می‌تواند به سیستم اضافه شود. باید دقت نمود که بعد از استفاده از جبران‌ساز پس‌فاز، سیستم همچنان دارای حد فاز رضایت‌بخشی باشد. برای اینکار معمولا فرکانس ماکزیمم فاز را به صورت زیر محاسبه کرده و پایین‌تر از فرکانس گذر بهره‌ی جدید قرار می‌دهند:

$$\omega_m = \frac{1}{T\sqrt{a}} \qquad (۱۰)$$

برای پیاده‌سازی جبران‌ساز پس‌فاز C(s) به فرم پاسخ فرکانسی از دستور زیر استفاده می‌نماییم:

```
        s = tf('s');
```

```
C_lag = (a*T*s+1)/(a*(T*s+1));
```

برای ادغام جبران‌ساز با سیستم از دستور زیر استفاده می‌نماییم:

```
sys_ol = C_lag*P;
```

جبران‌ساز پیش-پس‌فاز به کمک مکان هندسی ریشه‌ها یا پاسخ فرکانسی

جبران‌ساز پیش-پس‌فاز تاثیر هر دو جبران‌ساز پیش‌فاز و پس‌فاز را با یکدیگر ادغام می‌کند. نتیجه‌ی آن سیستمی است که پاسخ گذرا، پایداری و خطای حالت ماندگار بهبود یافته‌ای دارد. برای پیاده‌سازی جبران‌ساز پیش-پس‌فاز، ابتدا جبران‌ساز پیش‌فاز را برای دستیابی به پاسخ گذرا و پایداری مطلوب طراحی کرده و سپس جبران‌ساز پس‌فاز را برای بهبود خطای حالت ماندگار سیستم کنترل شده با جبران‌ساز پیش‌فاز، طراحی می‌نماییم.