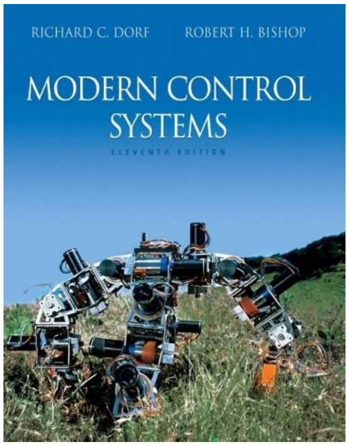
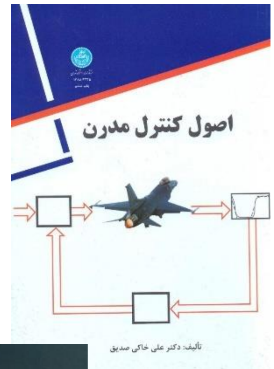
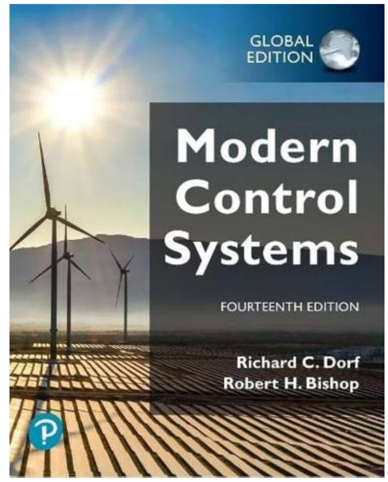
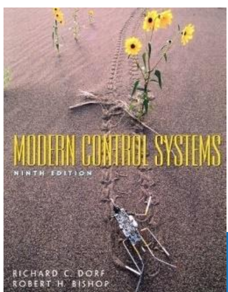
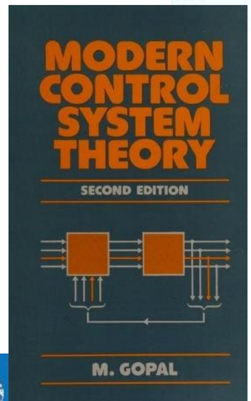
Modern Control Transfer Function report
S.Mehdi Moosaviun
ID : 99413136
Spring 2024

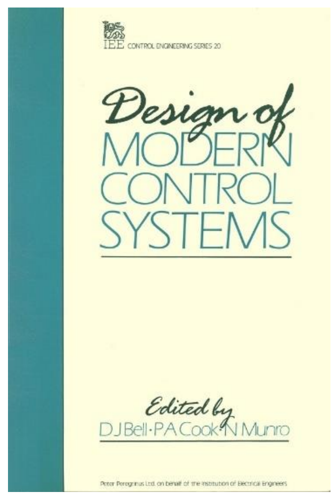


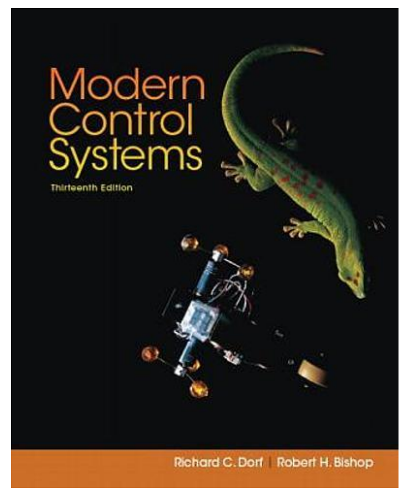


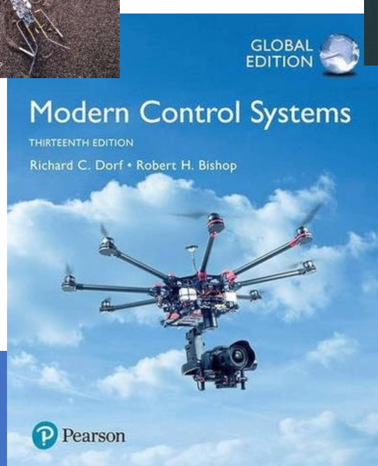












پروژه کنترل مدرن،گزارش بخش شبیه سازی تابع تبدیل

استاد: دکتر بلندی

تدریس یار: سرکارخانم قنادیان

نویسنده : سید مهدی موسویون – ۹۹۴۱۳۱۳۶

بهار۱۴۰۳

فهرست مطالب

تابع تبدیل داده شده

$$\frac{s+1.142}{2s^3+7.645s^2+18.73s+14.58}$$

۱ – شبیه سازی سیستم

صفرها و قطبهای سیستم را (با استفاده از نرم افزار متلب) محاسبه کنید؟ آیا سیستم حلقه باز پایدار است؟ آیا سیستم مینیمم فاز است؟

در ابتدا، قبل از پرداختن به پاسخ سؤالات، سیستم ارائه شده را به صورت تابع تبدیل در نرم‌افزار MATLAB پیاده‌سازی می‌کنیم. این مرحله اساسی برای تحلیل دقیق سیستم است. برای ایجاد تابع تبدیل، ابتدا صورت و مخرج تابع را به ترتیب در متغیرهای num و den تعریف می‌کنیم. سپس با استفاده از دستور tf، که مخفف Transfer Function است، تابع تبدیل را می‌سازیم. این روش به ما امکان می‌دهد تا سیستم را به شکلی دقیق و قابل تحلیل در MATLAB نمایش دهیم.

```
% Q_1 : Part A
numerator_coeffs = [3 1.677];
denominator_coeffs = [2 8.341 21.55 16.61];

sys = tf(numerator_coeffs, denominator_coeffs)
```

```
sys =

              3 s + 1.677
  -----------------------------------
  2 s^3 + 8.341 s^2 + 21.55 s + 16.61

Continuous-time transfer function.
```

۱-۱ صفر و قطب های سیستم

برای تعیین دقیق صفرها و قطب‌های تابع تبدیل در نرم‌افزارMATLAB ، از دستورات تخصصی zero(tf) و pole(tf) استفاده می‌کنیم. این دستورات به ترتیب صفرها و قطب‌های سیستم را محاسبه می‌کنند. صفرها و قطب‌ها اطلاعات مهمی درباره رفتار دینامیکی سیستم ارائه می‌دهند. پس از اجرای این دستورات، نتایج را با دقت بررسی و تحلیل می‌کنیم تا درک عمیق‌تری از خصوصیات سیستم به دست آوریم.

```
zeros = zero(sys);
poles = pole(sys);

disp('Zeros:');

disp(zeros);

disp('Poles:');

disp(poles);
```

```
Zeros:
   -0.5590
Poles:
  -1.5191 + 2.2421i
  -1.5191 - 2.2421i
  -1.1323 + 0.0000i
```

۱-۲ پایداری و مینیمم فاز بودن

به طور کلی میدانیم سیستمی که غیر مینیمم فاز باشد دو کاراکتر مشخص دارد ابتدا باید صفری در سمت راست محور jω داشته باشد که در این سیستم چنین نیست و خصوصیت دیگر سیستم غیر مینیمم فاز این است که در پاسخ پله ابتدا از پاسخ نهایی دور شده سپس به آن نزدیک میشود. بنابر نکات گفته شده اگر سیستم را مشاهده کنیم، واضح است که صفر آن قسمت حقیقی منفی دارد و اگر پاسخ پله را مشاهده کنیم خواهیم دید که سیستم از لحظه اعمال پله به سمت مقدار نهایی رفته و خصوصیت دیگر سیستم غیر مینیمم فاز را نیز ندارد. بنابراین سیستم مینیمم فاز میباشد.

اما در خصوص پایداری با توجه به اینکه تمام قطب‌های سیستم در سمت چپ محور jω میباشند سیستم پایدار مجانبی است. پایداری سیستم را در پاسخ پله آن نیز میتوان مشاهده نمود.

```
is_stable = all(real(poles) < 0);

is_minimum_phase = all(real(zeros) < 0);

if is_stable
   disp('System is a stable Open loop system');
else
   disp('System is Not a stable Open loop system');
end

if is_minimum_phase
   disp('System is a minimum phase system');
else
   disp('System is Not a minimum phase system ');
end
```

```
System is a stable Open loop system

System is a minimum phase system
```

۲- نمایش فضای حالت سیستم

یک نمایش فضای حالت برای سیستم بدست آورید؟

برای نمایش سیستم در قالب فضای حالت، که یکی از قدرتمندترین ابزارهای تحلیل و طراحی سیستم‌های کنترل است، روش‌های متنوعی وجود دارد. در این گزارش، به منظور سادگی و کارآمدی، از یکی از ساده‌ترین و در عین حال موثرترین این روش‌ها استفاده می‌کنیم.

با بهره‌گیری از دستور tf2ss() در نرم‌افزارMATLAB ، که مخفف 'Transfer Function to State Space' است، می‌توانیم تابع تبدیل موجود را به مدل فضای حالت تبدیل کنیم. این دستور به طور خودکار ماتریس‌هایA ، B، Cو D را که نمایش‌دهنده سیستم در فضای حالت هستند، محاسبه می‌کند.

پس از تبدیل، می‌توانیم از نمایش فضای حالت برای تحلیل‌های پیشرفته‌تر مانند کنترل‌پذیری، رؤیت‌پذیری، و طراحی کنترل‌کننده‌های مدرن استفاده کنیم. این رویکرد به ویژه در سیستم‌های چندورودی-چندخروجی (MIMO) بسیار کارآمد است.

```
% Convert the transfer function to state-space representation
[A, B, C, D] = tf2ss(numerator_coeffs, denominator_coeffs);
```

```
State-Space Representation:
A matrix:
   -4.1705  -10.7750   -8.3050
    1.0000         0         0
         0    1.0000         0
B matrix:
     1
     0
     0
C matrix:
         0    1.5000    0.8385
D matrix:
     0
```

۳- کنترل پذیری و مشاهده پذیری

۳-۱ بررسی کنترل‌پذیری و مشاهده‌پذیری

کنترل پذیری و رویت پذیری نمایش فضای حالت بدست آمده را بررسی کنید؟ آیا امکان طراحی فیدبک حالت و رویتگر حالت برای این نمایش وجود دارد؟ اگر نمایش فضای حالت بدست آمده کنترل ناپذیر است آن را به زیرسیستمهای کنترلپذیر و کنترلناپذیر تفکیک کنید. همچنین اگر نمایش فضای حالت بدست آمده رویت ناپذیر است آن را به زیرسیستم های رویت پذیر و رویت ناپذیر تفکیک کنید.

برای بررسی کنترل پذیری و مشاهده پذیری سیستم از دستورات نرم افزار متلب استفاده میکنیم. ابتدا برای بررسی کنترل پذیری سیستم از دستور ctrb() استفاده مینماییم که ماتریس کنترل پذیری سیستم را تشکیل میدهد. با استفاده از دستور rank() مرتبه ماتریس را بدست می‌آوریم و از این طریق تعیین میکنیم که سیستم کنترل پذیر است یا خیر. البته با توجه به اینکه سیستم تشکیل شده توسط نرم افزار متلب تحقق کنترل پذیر سیستم است، میتوان بدون بررسی این مرحله نیز متوجه شد که سیستم کنترل پذیر است. همانطور که معلوم است سیستم از مرتبه ۳ است و مرتبه کامل دارد در نتیجه این سیستم همزمان کنترل پذیر و مشاهد پذیر میباشد.

```
% Compute controllability matrix and its rank
controllability_matrix = ctrb(A, B);
rank_controllability = rank(controllability_matrix);

% Compute observability matrix and its rank
observability_matrix = obsv(A, C);
rank_observability = rank(observability_matrix);

% Check if the system is controllable
if rank_controllability == size(A, 1)
   disp('The system is controllable.')
else
   disp('The system is not controllable.')
end

% Check if the system is observable
if rank_observability == size(A, 1)
   disp('The system is observable.')
else
   disp('The system is not observable.')
end
```

```
Controllability Matrix:
    1.0000   -4.1705    6.6181
         0    1.0000   -4.1705
         0         0    1.0000
Rank of Controllability Matrix:
     3

Observability Matrix:
Rank of Observability Matrix:
     3

The system is controllable.
The system is observable.
```

۳-۲ امکان طراحی رویت‌گر و فیدبک حالت

در بخش قبلی مشخص شد که سیستم بدست آمده سیستمی رویت پذیر و کنترل پذیر میباشد. به علت کنترل پذیر بودن سیستم، قابلیت طراحی فیدبک حالت را فراهم میسازد. همچنین به علت مشاهده پذیر بودن سیستم قابلیت طراحی تخمین‌گر را نیز داراست.

۳-۳ تفکیک بخش های کنترل‌پذیر و مشاهده‌پذیر

همانطور که در بخش پیشین مطرح شد، سیستمی که داریم همزمان هم به طور کامل کنترل پذیر است هم به طور کامل مشاهده پذیر است. این ویژگی سیستم بیانگر این است که سیستم در صورت تفکیک بازهم تغییری نخواهد کرد. در نتیجه نیازی به انجام این تفکیک نخواهد بود. در صورت کنترل پذیر نبودن سیستم باید از دستور ctrbf(A,B,C) برای تفکیک استفاده میکردیم و در صورتی که سیستم مشاهده پذیر نبود باید از دستور obsvf(A,B,C) استفاده میکردیم تا به سیستم‌های تفکیک شده میرسیدیم اما حال دیگر نیازی به این کار نداریم.

۴- فضای حالت مینیمال سیستم

در صورتی که نمایش فضای حالت بدست آمده در قسمت قبل مینیمال نیست، یک نمایش فضای حالت مینیمال برای سیستم بدست آورید؟

سیستم بدست آمده سیستمی کنترل پذیر و مشاهده پذیر میباشد. در نتیجه ادعای ما بر مبنای مینیمال بودن سیستم است و در واقع نیازی برای بدست آوردن این بخش نیست. اما جهت اثبات این ادعا با استفاده از دستور mineral() متلب سیستم مینیمال را از دل سیستم خود استخراج میکنیم اگر سیستم بدست آمده برابر با سیستم اولیه ما شود میتوان گفت که ادعای انجام گرفته صحیح بوده. بنابراین به آزمایش ادعای خود میپردازیم.

```
% Check if the system is minimal
if rank_controllability && rank_observability
   disp('The system is minimal.')
else
   disp('The system is not minimal.')
end

   sys_minimal = minreal(sys);
   [Am, Bm, Cm, Dm] = ssdata(sys_minimal);
```

```
The system is minimal.

Minimal State-Space Representation:
A matrix:
   -4.1705   -2.6938   -1.0381
    4.0000         0         0
         0    2.0000         0
B matrix:
    0.5000
         0
         0
C matrix:
         0    0.7500    0.2096
D matrix:
     0
```

۵- پاسخ پله سیستم حلقه باز

پاسخ حلقه باز سیستم را (در صورت پایداری) به ازای ورودی پله واحد و شرایط اولیه دلخواه رسم کنید.

۱-۵ پایداری به ازا پله واحد

در تحلیل سیستم‌های کنترل، بررسی پاسخ پله یکی از ابزارهای کلیدی برای درک رفتار دینامیکی سیستم است. همانطور که در بخش‌های پیشین مشاهده کردیم، پاسخ پله سیستم را با استفاده از تابع تبدیل بررسی نمودیم. اکنون، با بهره‌گیری از نمایش فضای حالت، که در مراحل قبل به آن دست یافتیم، مجدداً به بررسی پاسخ پله سیستم می‌پردازیم. این رویکرد، گرچه در نتیجه نهایی تفاوتی ایجاد نمی‌کند، اما از منظر روش‌شناسی و کاربرد ابزارهای نرم‌افزاری، تفاوت قابل توجهی دارد.

میدانیم سیستمی پایدار است که تمام قطب های آن در سمت چپ محور یا کوچکتر از صفر باشند پس این شرط را بررسی میکنم. همچنین استفاده از دستور step() در محیط MATLAB برای نمایش فضای حالت، روشی متفاوت و در عین حال قدرتمند را در اختیار ما قرار می‌دهد. این تفاوت در روش، به ویژه در سیستم‌های پیچیده‌تر و چندبعدی، می‌تواند مزایای قابل توجهی داشته باشد. برای مثال، در

سیستم‌های چندورودی-چندخروجی (MIMO)، استفاده از نمایش فضای حالت می‌تواند تحلیل پاسخ پله را ساده‌تر و جامع‌تر کند.

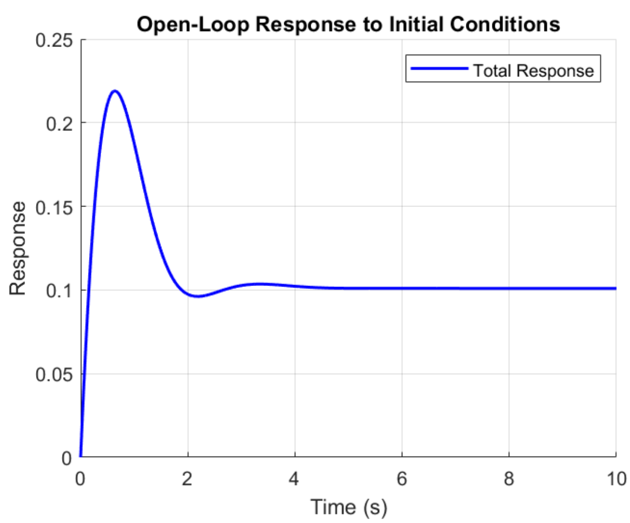
```
% Check stability
is_stable = all(real(poles) < 0);
if is_stable
    disp('System is a stable Open loop system');
else
    disp('System is Not a stable Open loop system');
    return;
end

% Step(sys)
```

```
System is a stable Open loop system
```

۵-۲ پایداری به ازا شرایط اولیه دلخواه

در ادامه تحلیل سیستم و بررسی پاسخ آن، اکنون به بررسی تأثیر شرایط اولیه بر رفتار سیستم میپردازیم. با تعیین شرایط اولیه به صورت یک بردار سه بعدی [۰.۵ ۰ ۰]، که نشان‌دهنده مقادیر اولیه متغیرهای حالت سیستم است، می‌توانیم درک عمیق‌تری از دینامیک سیستم به دست آوریم.
این انتخاب شرایط اولیه غیر صفر به ما امکان می‌دهد تا رفتار سیستم را از نقطه‌ای غیر از حالت تعادل مشاهده کنیم. این امر به ویژه در سیستم‌های واقعی که ممکن است همیشه در حالت تعادل شروع به کار نکنند، بسیار مهم است. با مشاهده خروجی سیستم تحت این شرایط، می‌توانیم چندین جنبه مهم را بررسی کنیم: اولاً، می‌توانیم ببینیم که سیستم چگونه از این حالت اولیه به سمت حالت پایدار نهایی حرکت می‌کند. این مسیر گذرا اطلاعات ارزشمندی درباره خصوصیات میرایی و نوسانی سیستم ارائه می‌دهد. ثانیاً، می‌توانیم زمان نشست سیستم را در این حالت بررسی کنیم و آن را با حالت شرایط اولیه صفر مقایسه کنیم. این مقایسه می‌تواند نشان دهد که آیا سیستم تحت شرایط مختلف، رفتار یکنواختی از خود نشان می‌دهد یا خیر. علاوه بر این، بررسی پاسخ سیستم با شرایط اولیه غیر صفر می‌تواند در تحلیل پایداری سیستم نیز مفید باشد. اگر سیستم با وجود شرایط اولیه متفاوت همچنان به نقطه تعادل مشخصی همگرا شود، این امر نشان‌دهنده پایداری قوی سیستم است. در نهایت، این تحلیل می‌تواند در طراحی کنترل‌کننده‌ها نیز مفید باشد، زیرا یک کنترل‌کننده خوب باید بتواند سیستم را از هر شرایط اولیه‌ای به حالت مطلوب برساند.

```
% Define initial conditions
x0 = [0.5; 0; 0];

% Define time vector
t = 0:0.01:10;

% Create state-space system
sys_ss = ss(A, B, C, D);

% Compute the response to the unit step input
[y_step, t, x_step] = step(sys_ss, t);

% Compute the zero-input response
[y_initial, t, x_initial] = initial(sys_ss, x0, t);

% Compute the total response
total_response = y_step + y_initial;
```



۶- اضافه نمودن فیدبک واحد

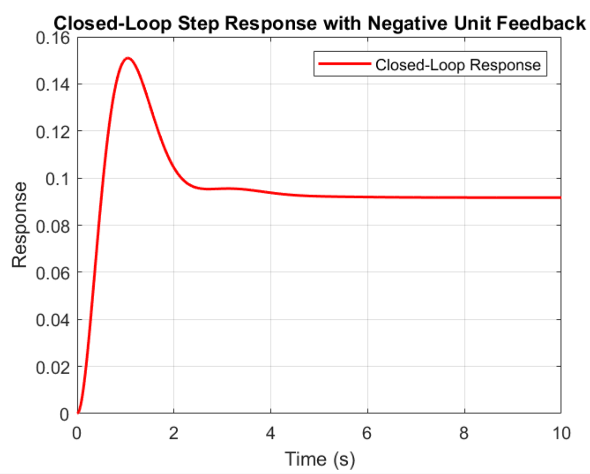
سیستم حلقه باز را بر اساس نمایش فضای حالت بدست آمده با استفاده از فیدبک واحد منفی به صورت حلقه بسته شبیهسازی کنید، پاسخ پله سیستم حلقه بسته را رسم و قطبها و صفرهای آن را بدست آورید؟

در مرحله بعدی تحلیل سیستم، به بررسی تأثیر فیدبک بر رفتار سیستم می‌پردازیم. با استفاده از دستور `feedback(sys,1)` در MATLAB، یک حلقه فیدبک واحد به سیستم اصلی اضافه می‌کنیم. این عمل، که در طراحی سیستم‌های کنترل بسیار رایج است، امکان مطالعه رفتار سیستم در حالت حلقه بسته را فراهم می‌کند.

اضافه کردن فیدبک به سیستم می‌تواند تغییرات قابل توجهی در پاسخ دینامیکی آن ایجاد کند. با مشاهده پاسخ پله سیستم جدید (سیستم با فیدبک واحد)، می‌توانیم تفاوت‌های احتمالی در پارامترهایی مانند زمان نشست، فراجهش، و خطای حالت ماندگار را بررسی کنیم. این مقایسه به ما کمک می‌کند تا درک بهتری از تأثیر فیدبک بر عملکرد سیستم داشته باشیم.

نکته مهمی که باید در نظر داشت این است که اگرچه فیدبک واحد پایداری ذاتی سیستم را تغییر نمی‌دهد، اما می‌تواند تأثیر قابل توجهی بر سایر خصوصیات سیستم داشته باشد. برای مثال، فیدبک می‌تواند سرعت پاسخ سیستم را افزایش دهد، خطای حالت ماندگار را کاهش دهد، و حساسیت سیستم به تغییرات پارامترها را تعدیل کند.

```
% Create the closed-loop system using negative unit feedback
sys_cl = feedback(sys_ss, 1);


% Compute the step response of the closed-loop system
[y_cl, t_cl] = step(sys_cl, t);
```




```
% Compute zeros and poles of the closed-loop system
zeros_cl = zero(sys_cl);
poles_cl = pole(sys_cl);
```

```
Closed-Loop Zeros:
   -0.5590
Closed-Loop Poles:
  -1.5824 + 2.5668i
  -1.5824 - 2.5668i
  -1.0056 + 0.0000i
```

واضح است که اضافه کردن فیدبک واحد باعث تغییر مکان قطب‌های سیستم شده اما تغییری در مکان صفر صورت نگرفته. همچنین همانطور که گفته شده بود سیستم همچنان پایدار میباشد.

۷-قراردهی قطب ها در محل دلخواه

برای نمایش فضای حالت بدست آمده برای سیستم، فیدبک حالت را چنان طراحی کنید تا قطبهای سیستم حلقه بسته در مکانهای دلخواهی در سمت چپ محور jω قرار بگیرند. پاسخ پله سیستم حلقه بسته و متغیرهای حالت سیستم و سیگنال کنترلی آن را رسم کنید. قطبها و صفرهای سیستم حلقه بسته و حلقه باز را با یکدیگر مقایسه کنید .

توجه: این مرحله را برای جابابی قطبهای دور و نزدیک انجام دهید و سیگنال کنترلی و بهره فیدبک حالت بدست آمده در هر دو حالت را با یکدیگر مقایسه کنید.

فرآیند با تعریف یک بازه زمانی از ۰ تا ۱۰ ثانیه با گام‌های ۰.۰۱ ثانیه‌ای آغاز می‌شود، که دقت بالایی را در شبیه‌سازی تضمین می‌کند.

```
% Define time vector
tspan = 0:0.01:10;
```

در ادامه، دو مجموعه قطب مطلوب تعریف می‌شوند: قطب‌های دور (5-، 7-، 10-):P_far و قطب‌های نزدیک (1-، 2-، 3-):P_near . این انتخاب هوشمندانه امکان مقایسه بین پاسخ‌های سریع و آهسته سیستم را فراهم می‌کند. با استفاده از تابع 'acker'، که یک روش کلاسیک برای طراحی فیدبک حالت است، ماتریس‌های بهره K_far و K_near محاسبه می‌شوند.

```
% Desired pole locations for far poles
P_far = [-5 -7 -10];
K_far = acker(A, B, P_far)

% Desired pole locations for near poles
P_near = [-1 -2 -3];
K_near = acker(A, B, P_near)
```

تابع 'acker' در MATLAB یک ابزار قدرتمند برای طراحی فیدبک حالت در سیستم‌های کنترل خطی است. این تابع از روش آکرمن برای محاسبه ماتریس بهره فیدبک حالت استفاده می‌کند، به گونه‌ای که قطب‌های سیستم حلقه بسته در مکان‌های دلخواه قرار گیرند.

در کد مورد بحث، تابع 'acker' دو بار استفاده شده است:

۱. `K_far = acker(A, B, P_far)`

۲. `K_near = acker(A, B, P_near)`

در هر دو مورد، تابع 'acker' سه ورودی دریافت می‌کند:

۱. A: ماتریس حالت سیستم

۲. B: ماتریس ورودی سیستم

۳. P_far یا P_near: بردار قطب‌های مطلوب

استفاده از تابع 'acker' در این کد امکان مقایسه مستقیم بین دو طراحی مختلف فیدبک حالت را فراهم می‌کند. این مقایسه به مهندسان کنترل اجازه می‌دهد تا تأثیر جایابی قطب‌ها را بر عملکرد سیستم، از جمله سرعت پاسخ، میزان فراجهش، و شدت سیگنال کنترلی، بررسی کنند. باید توجه داشت که روش آکرمن، اگرچه قدرتمند است، محدودیت‌هایی نیز دارد. این روش برای سیستم‌های تک ورودی بهینه است و در سیستم‌های چند ورودی ممکن است نتایج مطلوبی نداشته باشد. همچنین، در سیستم‌های با مرتبه بالا ممکن است به مشکلات عددی برخورد کند. با این حال، برای بسیاری از کاربردهای عملی، به ویژه در سیستم‌های با مرتبه پایین مانند مورد بحث در این کد، تابع 'acker' یک ابزار کارآمد و قابل اعتماد برای طراحی فیدبک حالت است.

```
K_near = 1×3
    1.8295    0.2250   -2.3050

K_far = 1×3
   17.8295  144.2250  341.6950
```

سپس، کد از تابع 'ode45' برای حل معادلات دیفرانسیل سیستم استفاده می‌کند. این تابع قدرتمند، پاسخ زمانی سیستم را برای هر دو حالت قطب‌های دور و نزدیک شبیه‌سازی می‌کند. همزمان، سیگنال‌های کنترلی متناظر (u_near و u_far) محاسبه می‌شوند، که نشان‌دهنده تلاش کنترلی لازم برای دستیابی به دینامیک مطلوب است.

```
% Simulate the system for far poles
[t_far, x_far] = ode45(@(t, x) d_Q7(t, x, A, B, K_far), tspan, x0);
u_far = -K_far * x_far';

% Simulate the system for near poles
[t_near, x_near] = ode45(@(t, x) d_Q7(t, x, A, B, K_near), tspan, x0);
u_near = -K_near * x_near';
```

تابع 'ode45' در MATLAB یک ابزار قدرتمند برای حل عددی معادلات دیفرانسیل معمولی (ODE) است. این تابع از روش رانگ-کوتای مرتبه ۴ و ۵ با گام متغیر استفاده می‌کند، که ترکیبی از دقت بالا و کارایی محاسباتی

را ارائه می‌دهد. در مورد اول، 'ode45' دینامیک سیستم را با فیدبک حالت K_far (برای قطب‌های دور) شبیه‌سازی می‌کند. در مورد دوم، همین کار را با K_near (برای قطب‌های نزدیک) انجام می‌دهد.

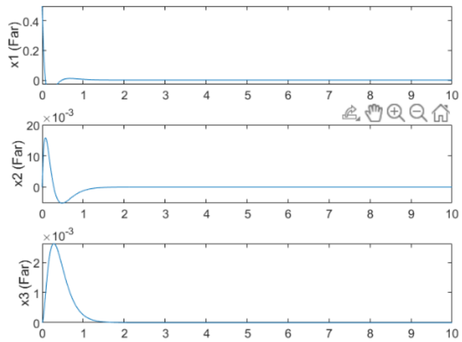
استفاده از 'ode45' در این کد چندین مزیت دارد:

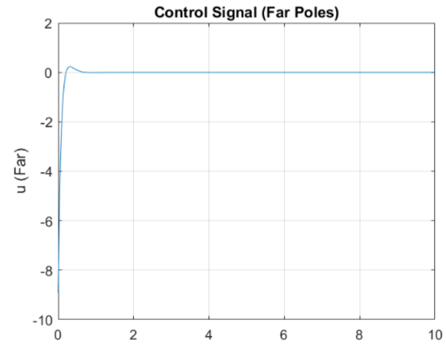
دقت بالا، گام متغیر: 'ode45' به طور خودکار اندازه گام‌های زمانی را تنظیم می‌کند تا تعادلی بین دقت و سرعت محاسبات برقرار کند.

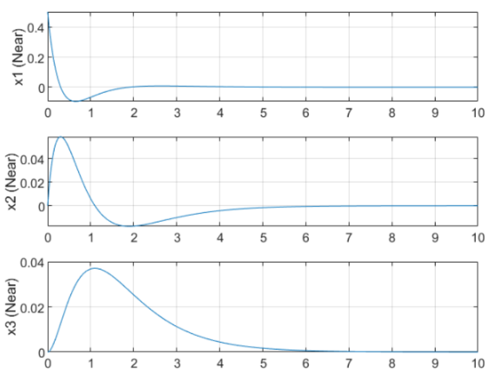
حال برای محاسبه و مقایسه قطب‌های سیستم در حالت‌های مختلف می‌پردازد، که برای درک عمیق‌تر رفتار دینامیکی سیستم بسیار مهم است. ابتدا، با استفاده از تابع 'eig'، قطب‌های سیستم حلقه باز محاسبه می‌شوند. این قطب‌ها، همان مقادیر ویژه ماتریس A هستند. سپس، کد به محاسبه قطب‌های سیستم حلقه بسته برای دو حالت مختلف فیدبک حالت می‌پردازد. در حالت اول، با استفاده از ماتریس بهره K_far، قطب‌های سیستم حلقه بسته برای حالتی که قطب‌های مطلوب دورتر از محور موهومی قرار دارند، محاسبه می‌شوند. این قطب‌ها از مقادیر ویژه ماتریس A - B * K_far به دست می‌آیند. به طور مشابه، در حالت دوم، قطب‌های سیستم حلقه بسته برای حالتی که قطب‌های مطلوب نزدیک‌تر به محور موهومی هستند، با استفاده از ماتریس بهره K_near محاسبه می‌شوند.

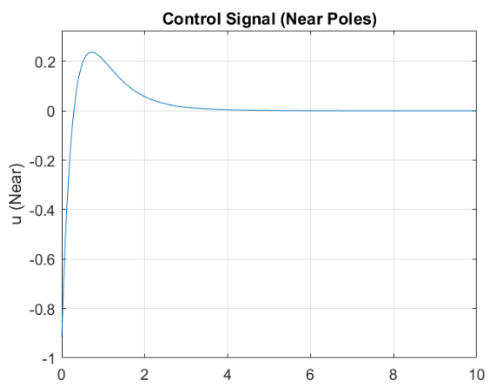
```
% Open loop poles
openLoop_poles = eig(A);

% Closed loop poles for far poles
closedLoop_poles_far = eig(A - B * K_far);

% Closed loop poles for near poles
closedLoop_poles_near = eig(A - B * K_near);
```

```
Open-Loop Poles:
  -1.5191 + 2.2421i
  -1.5191 - 2.2421i
  -1.1323 + 0.0000i
Closed-Loop Poles (Far):
  -10.0000
   -7.0000
   -5.0000
Closed-Loop Poles (Near):
   -3.0000
   -2.0000
   -1.0000
```









حال در این قسمت از کد به مقایسه و تحلیل دو سیستم حلقه بسته با قطب‌های دور و نزدیک می‌پردازد. ابتدا سیستم حلقه بسته با قطب‌های دور ایجاد شده و با استفاده از تابع ss، ماتریس‌های سیستم و ضریب فیدبک K_far تعریف می‌شود. سپس پاسخ پله و پاسخ حالت اولیه این سیستم با استفاده از توابع step و initial محاسبه می‌شود. پاسخ کلی سیستم از جمع این دو پاسخ به دست می‌آید. در ادامه، مشخصات پاسخ پله سیستم با تابع stepinfo بررسی می‌شود. همین روند برای سیستم حلقه بسته با قطب‌های نزدیک نیز تکرار می‌شود، با این تفاوت که از ضریب فیدبک K_near استفاده می‌شود. این کد امکان مقایسه عملکرد دو سیستم با قطب‌های متفاوت را فراهم می‌کند و می‌تواند برای تحلیل تأثیر جایگذاری قطب‌ها بر رفتار سیستم کنترلی مورد استفاده قرار گیرد.

```
% Closed-loop system for far poles
closeLoop_far = ss(A - B * K_far, B, C, D);
[y_s_far, t_s_far, x_s_far] = step(closeLoop_far, tspan);
[y_i_far, t_i_far, x_i_far] = initial(closeLoop_far, x0, tspan);
y_far = y_s_far + y_i_far;
stepinfo(closeLoop_far)
```

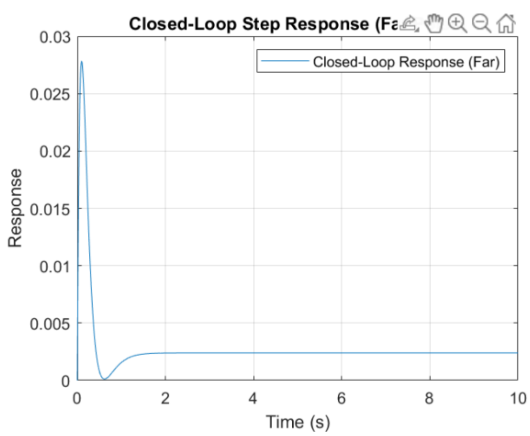
```
% Closed-loop system for near poles
closeLoop_near = ss(A - B * K_near, B, C, D);
[y_s_near, t_s_near, x_s_near] = step(closeLoop_near, tspan);
[y_i_near, t_i_near, x_i_near] = initial(closeLoop_near, x0, tspan);
y_near = y_s_near + y_i_near;
stepinfo(closeLoop_near)
```

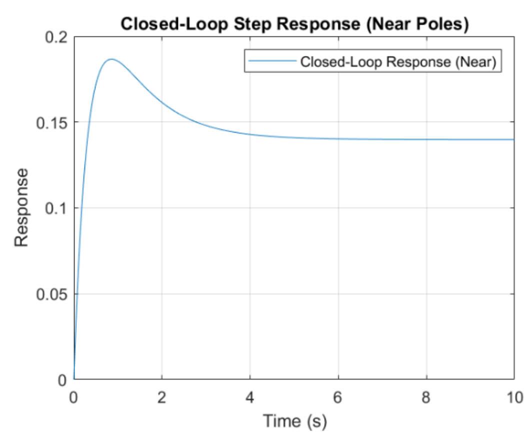
```
stepinfo(closeLoop_far)
```

```
ans = struct with fields:
         RiseTime: 0.0492
    TransientTime: 1.3704
     SettlingTime: 1.5707
      SettlingMin: 0.0024
      SettlingMax: 0.0087
        Overshoot: 261.5084
       Undershoot: 0
             Peak: 0.0087
         PeakTime: 0.3132
```

```
stepinfo(closeLoop_near)
```

```
ans = struct with fields:
         RiseTime: 0.6452
    TransientTime: 4.7450
     SettlingTime: 4.7450
      SettlingMin: 0.1290
      SettlingMax: 0.1678
        Overshoot: 20.0904
       Undershoot: 0
             Peak: 0.1678
         PeakTime: 1.7193
```





۸- طراحی ردیاب استاتیک

برای نمایش فضای حالت بدست آمده یک ردیاب استاتیکی طراحی کنید.

این بخش از کد به طراحی و شبیه‌سازی یک ردیاب استاتیک می‌پردازد. ابتدا مکان قطب‌های مطلوب سیستم تعیین شده و با استفاده از تابع acker، ماتریس بهره فیدبک K محاسبه می‌شود. سپس شرایط اولیه سیستم و بازه زمانی شبیه‌سازی تعریف می‌شوند.

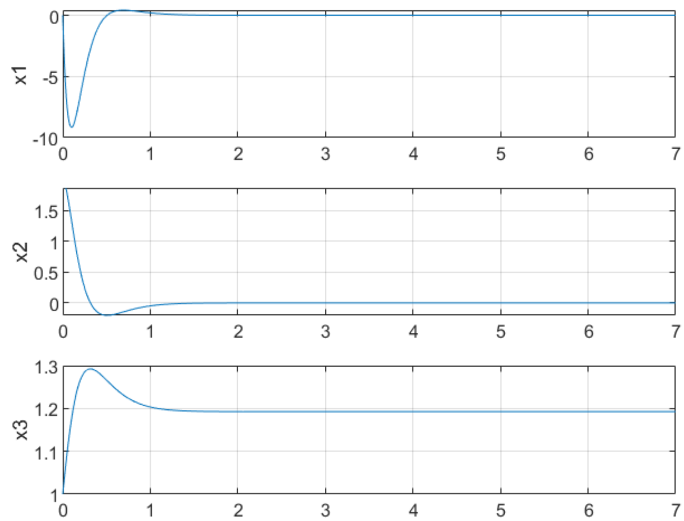
```
% Desired pole locations
P = [-5 -7 -10];
K = acker(A, B, P);

% Define initial conditions
x0 = [0; 2; 1];

% Define time vector
tspan = 0:0.01:7
```

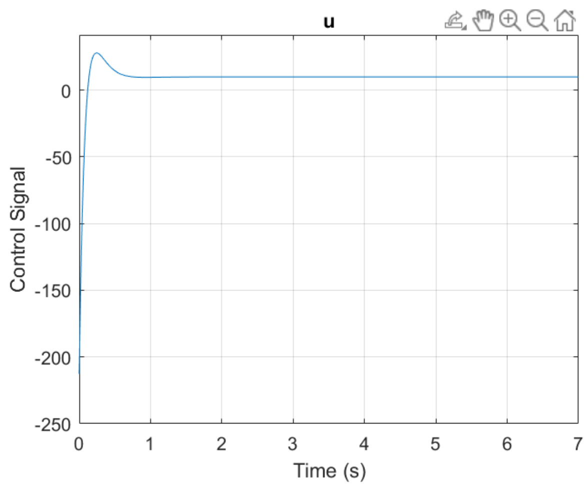
در ادامه، بهره ردیاب استاتیک با استفاده از روابط جبری محاسبه می‌شود. این محاسبات شامل تعیین تابع تبدیل حلقه باز و محاسبه بهره p برای حالت ماندگار است.

```
% Compute the gain for the static tracker
syms s;
Ga = C * inv(s * eye(size(A)) - (A - B * K)) * B;
p = inv(-C * inv(A - B * K) * B); % for s=0
r = 1; % reference input
```

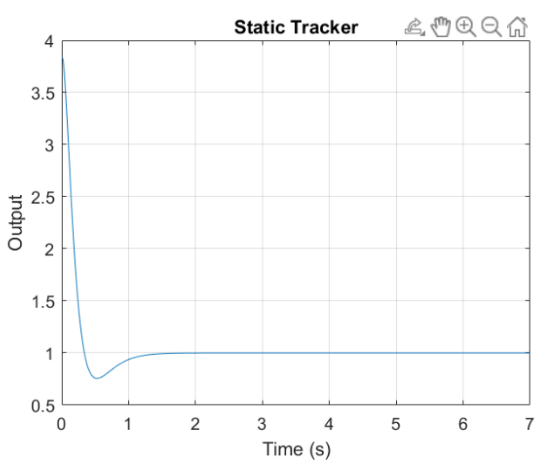
برای شبیه‌سازی سیستم، از تابع ode45 استفاده می‌شود که معادلات دیفرانسیل سیستم را حل می‌کند. تابع این سوال همانگونه که در زیر آمده است، معادلات دینامیکی سیستم را تعریف می‌کند. پس از شبیه‌سازی، سیگنال کنترل و خروجی سیستم محاسبه می‌شوند.

```
% Simulate the system
[t, x] = ode45(@(t, x) d_Q8(t, x, A, B, C, K, p, r), tspan, x0);

% Compute the control signal and output
u = -K * x' + p * r;
y = C * x';

% Q -8 fcn:
function dxdt = d_Q8(t, x, A, B, C, K, p, r)
   u = -K * x + p * r;
   dxdt = A * x + B * u;
end
```





در نهایت ردیاب استاتیک بصورت زیر نمایش داده میشود:



۹- طراحی ردیاب انتگرالی

برای نمایش فضای حالت بدست آمده یک ردیاب انتگرالی طراحی کنید.

برای طراحی ردیاب انتگرالی دو شر ط باید چک شود. اول آنکه ماتریس کنترل پذیر باشد سپس ماتریس $\begin{bmatrix} A & B \\ 0 & -C \end{bmatrix}$ Full Rank باشد.

کد به بررسی کنترل‌پذیری سیستم می‌پردازد. با استفاده از تابع ctrb و بررسی رتبه ماتریس کنترل‌پذیری، قابلیت کنترل سیستم ارزیابی می‌شود. اگر رتبه این ماتریس برابر با تعداد متغیرهای حالت باشد، سیستم کنترل‌پذیر است. کد به بررسی قابلیت کنترل انتگرالی سیستم می‌پردازد. برای این منظور، ماتریس‌های Abar و Bbar که مربوط به سیستم افزوده شده (augmented system) هستند، تشکیل می‌شوند. سپس با بررسی رتبه ماتریس ترکیبی [B A; zeros(m, 1) -C]، قابلیت کنترل انتگرالی سیستم ارزیابی می‌شود. اگر رتبه این ماتریس برابر با مجموع تعداد متغیرهای حالت و تعداد خروجی‌ها باشد، سیستم قابلیت کنترل انتگرالی دارد.

```
% Check if the system is controllable
n = size(A, 1); % number of states
m = size(C, 1); % number of outputs

if rank(ctrb(A, B)) == n
   disp('System is controllable')
else
   disp('System is not controllable')
   return;
end

% Check if the augmented matrix is full rank
Abar = [A zeros(n, m); -C zeros(m, m)];
Bbar = [B; zeros(m, 1)];

if rank([B A; zeros(m, 1) -C]) == (n + m)
   disp('System is integrally controllable')
else
   disp('System is not integrally controllable')
   return;
end
```

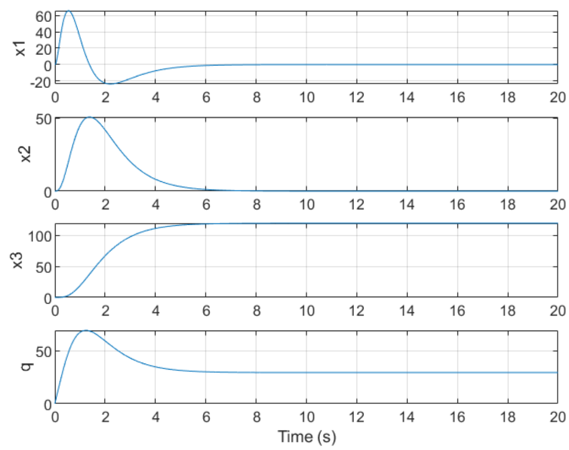
این بخش از کد اساس طراحی ردیاب انتگرالی را فراهم می‌کند و با بررسی شرایط لازم برای کنترل‌پذیری و کنترل‌پذیری انتگرالی، امکان‌پذیری طراحی کنترل‌کننده انتگرالی را تأیید می‌کند. این مرحله برای اطمینان از قابلیت طراحی یک کنترل‌کننده مؤثر برای سیستم ضروری است.

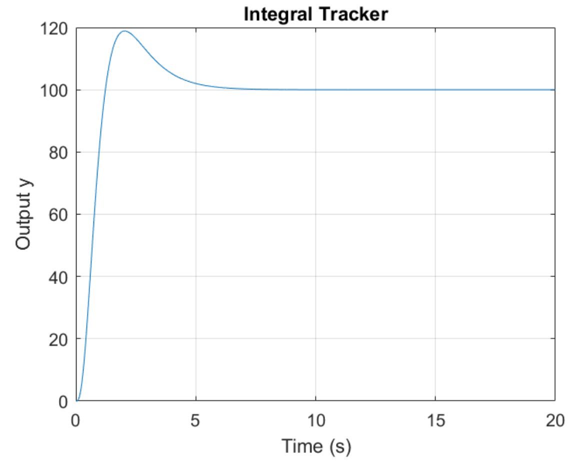
```
System is integrally controllable
System is integrally controllable
```

در ادامه کد به طراحی و شبیه‌سازی ردیاب انتگرالی می‌پردازد. ابتدا، مکان قطب‌های مطلوب برای سیستم افزوده شده تعیین می‌شود. سپس با استفاده از تابع acker، بهره فیدبک K برای سیستم افزوده محاسبه می‌شود. این بهره به دو بخش K1 و K2 تقسیم می‌شود که به ترتیب برای متغیرهای حالت اصلی و متغیر انتگرالی استفاده می‌شوند. در ادامه، شرایط اولیه سیستم، بردار زمان و خروجی مطلوب تعریف می‌شوند. یک اغتشاش صفر نیز در نظر گرفته شده است.

برای شبیه‌سازی سیستم، یک حلقه زمانی ایجاد می‌شود. در هر گام زمانی، سیگنال کنترل با استفاده از قانون کنترل u = -K1*x - K2*xi محاسبه می‌شود، که در آن xi متغیر انتگرالی است. سپس، وضعیت سیستم با استفاده از روش اویلر به‌روزرسانی می‌شود. این به‌روزرسانی شامل دینامیک سیستم اصلی، متغیر انتگرالی، ورودی مرجع و اغتشاش است.

```
% Desired pole locations for the augmented system
P = [-1 -2 -3 -4];
% Calculate the feedback gain for the augmented system
K = acker(Abar, Bbar, P);
K1 = K(1:n);
K2 = K(n+1:n+m);
% Initial state
x = [0; 0; 0; 0];
% Time vector
t = 0:0.01:20;
[rt, ct] = size(t);
% Desired output
yd = 1;

% Disturbance
w = zeros(n, 1); % disturbance

% Simulate the system
x_traj = zeros(n+m, ct);
u_traj = zeros(1, ct);
y_traj = zeros(m, ct);
for k = 2:ct
    u_traj(k) = [-K1 -K2] * x;
    x = x + 0.01 * (Abar * x + Bbar * u_traj(k)) + [zeros(n, 1); ones(m, 1)] * yd + [w; zeros(m, 1)];
    x_traj(:, k) = x;
end
% Output
y_traj = C * x_traj(1:n, :);
```





این کد یک ردیاب انتگرالی را پیاده‌سازی می‌کند که قادر است خروجی سیستم را به مقدار مرجع ثابت برساند، حتی در حضور اغتشاشات ثابت. استفاده از انتگرال‌گیر در ساختار کنترل‌کننده باعث می‌شود که خطای حالت ماندگار به صفر برسد و سیستم بتواند ورودی مرجع را به طور دقیق دنبال کند.

۱۰- مقایسه و تحلیل پاسخ های دو ردیاب

پاسخهای دو ردیاب طراحی شده در سوال ۸ و۹ را مقایسه و نتیجه را از دیدگاههای زیرتحلیل کنید.

عملکرد ردیابی/ مقاومت در حضور تغییر پارامترهای مدل/ عملکرد سیستم حلقه بسته با وارد کردن اغتشاشات ثابت

در این بخش، به مقایسه و تحلیل نتایج حاصل از دو روش کنترلی پیشین می‌پردازیم. این مقایسه جامع در سه محور اصلی صورت می‌گیرد تا تصویری کامل از عملکرد هر دو سیستم ارائه دهد:

۱. پاسخ پله: ابتدا واکنش هر دو سیستم به ورودی پله را بررسی می‌کنیم. این مقایسه به ما امکان می‌دهد تا سرعت پاسخ، میزان فراجهش و زمان نشست هر سیستم را ارزیابی کنیم.

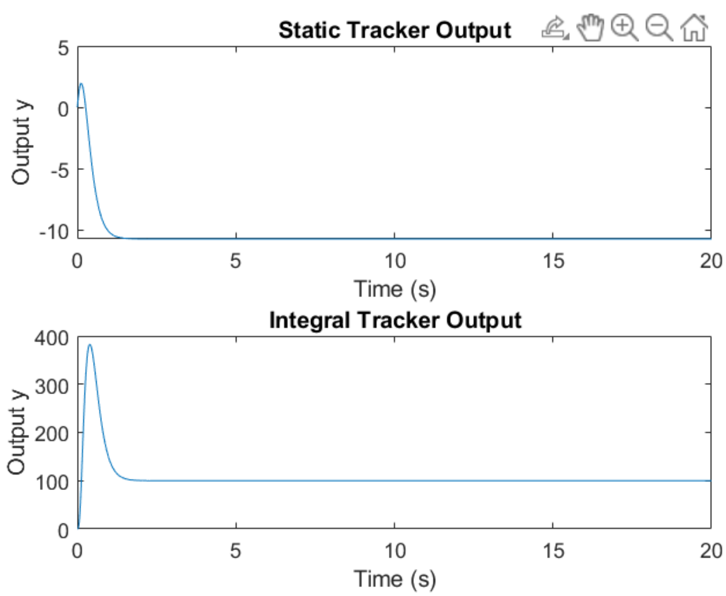
۲. تغییر پارامترها: در این مرحله، تأثیر تغییر پارامترهای سیستم را در دو حالت مختلف مطالعه می‌کنیم. این بررسی، مقاومت و انعطاف‌پذیری هر روش کنترلی را در برابر تغییرات سیستم نشان می‌دهد.

۳. پاسخ به اغتشاش: در نهایت، عملکرد هر دو سیستم را در حضور اغتشاشات خارجی ارزیابی می‌کنیم. این مقایسه توانایی هر روش را در حفظ پایداری و عملکرد مطلوب در شرایط غیر ایده‌آل نشان می‌دهد.

۱۰-۱ مقایسه پاسخ پله

در ابتدا به مقایسه خروجی های آنها میپردازیم

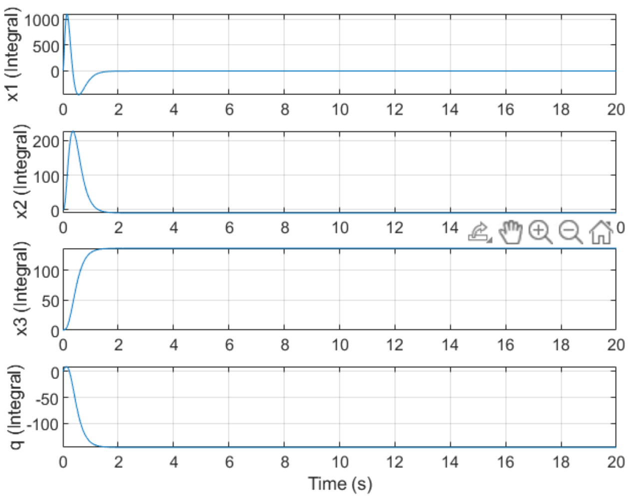
```
figure;
subplot(2,1,1);
plot(t, y_static_traj);
title('Static Tracker Output');
xlabel('Time (s)');
ylabel('Output y');

subplot(2,1,2);
plot(t, y_integral_traj);
title('Integral Tracker Output');
xlabel('Time (s)');
ylabel('Output y');
```

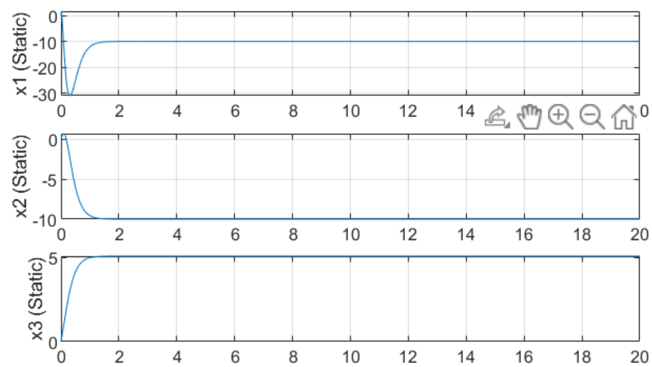


در مرحله بعد برای مقایسه حالت های دو سیستم را بررسی مینماییم.

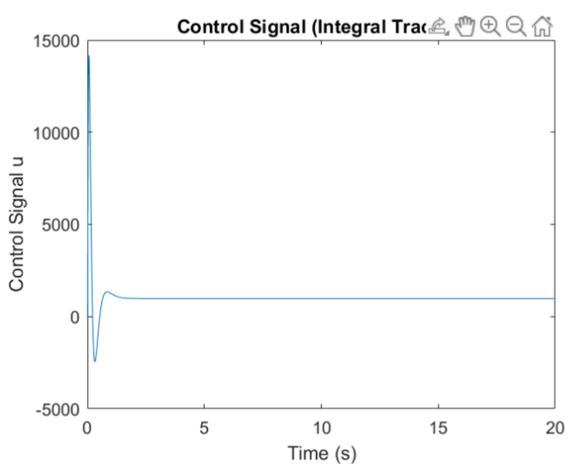
```
figure;
subplot(4, 1, 1);
plot(t, x_static_traj(1, :));
ylabel('x1 (Static)');
grid on;

subplot(4, 1, 2);
plot(t, x_static_traj(2, :));
ylabel('x2 (Static)');
```

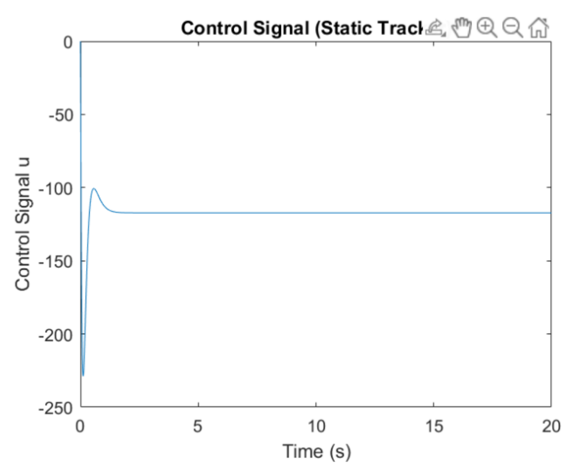
```
grid on;

subplot(4, 1, 3);
plot(t, x_static_traj(3, :));
ylabel('x3 (Static)');
grid on;
```





سیگنال های کنترلی آنها نیز در ادامه آورده شده است:



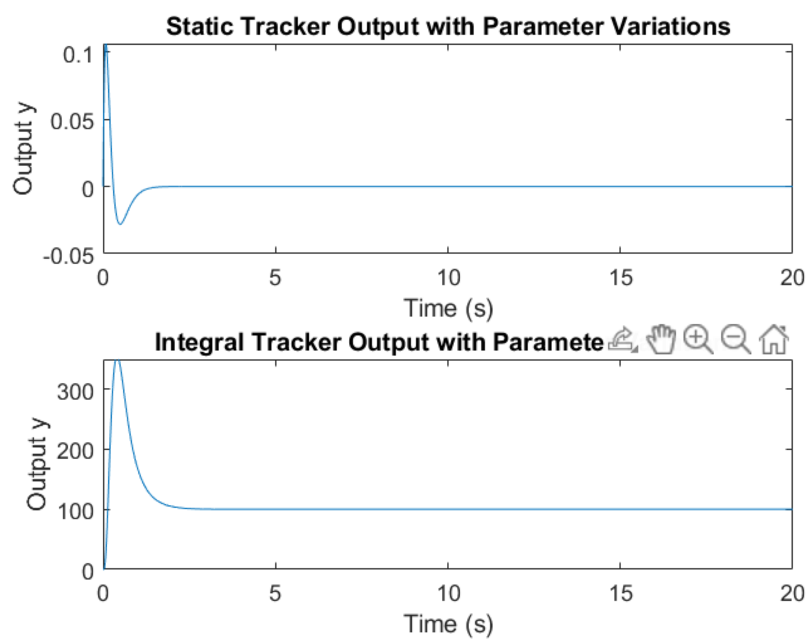


مقایسه عملکرد سیستم‌های کنترلی با ردیاب استاتیک و انتگرالی نشان می‌دهد که ردیاب استاتیک از سرعت پاسخ اندکی بالاتری برخوردار است. این تفاوت در سرعت، اگرچه قابل مشاهده، اما چندان چشمگیر نیست و در اکثر کاربردها می‌تواند نادیده گرفته شود. علت اصلی این اختلاف جزئی، وجود قطب اضافی در ساختار ردیاب انتگرالی است که طبیعتاً منجر به کمی تأخیر در پاسخ سیستم می‌شود.

با این حال، نکته قابل توجه این است که هر دو سیستم در این مرحله از ارزیابی، عملکرد مطلوب و قابل قبولی از خود نشان داده‌اند. ردیاب استاتیک با برتری جزئی در سرعت پاسخ، و ردیاب انتگرالی با قابلیت‌های خاص خود که در مراحل بعدی ارزیابی مشخص خواهد شد، هر دو پتانسیل بالایی برای کاربردهای مختلف دارند.

۲-۱۰ تغییر پارامتر ها

```
% Simulate the static tracker with parameter variations
A_variation = A + 0.1 * eye(size(A));
x_static = x0;
x_static_traj_variation = zeros(n, ct);
for k = 2:ct
    u_static_traj(k) = -K_static * x_static;
    x_static = x_static + 0.01 * (A_variation * x_static + B * u_static_traj(k));
    x_static_traj_variation(:, k) = x_static;
end
y_static_traj_variation = C * x_static_traj_variation;
```

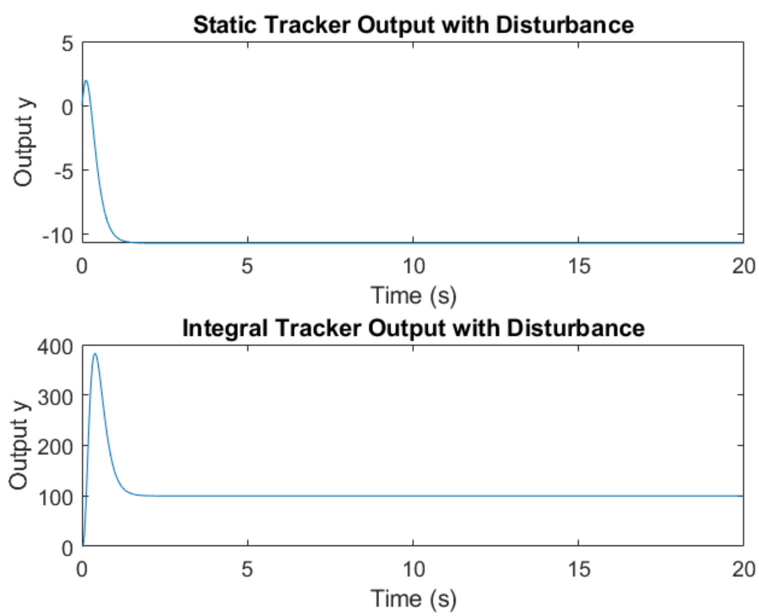


با بررسی نمودارهای ارائه شده،ردیاب استاتیک رفتاری پایدار و کنترل شده از خود نشان می‌دهد. خروجی این سیستم پس از نوسانات اولیه کوچک و میرا شونده، به سرعت به صفر همگرا می‌شود. زمان نشست این سیستم نسبتاً کوتاه بوده و حدود ۵ ثانیه است، و دامنه تغییرات آن بسیار محدود و بین ۰.۱ و -۰.۰۵ قرار دارد. این ویژگی‌ها نشان‌دهنده مقاومت و پایداری بالای ردیاب استاتیک در برابر تغییرات پارامتری است.

در مقابل، ردیاب انتگرالی واکنش متفاوت و قابل توجهی به تغییرات پارامتری نشان می‌دهد. این سیستم با یک فراجهش اولیه بزرگ (تا حدود ۳۵۰) شروع می‌شود و سپس به تدریج به یک مقدار ثابت غیر صفر (حدود ۱۰۰) همگرا می‌شود. زمان نشست این سیستم حدود ۵ ثانیه است، و دامنه تغییرات آن بسیار وسیع‌تر از ردیاب استاتیک است. این رفتار نشان می‌دهد که ردیاب انتگرالی حساسیت بیشتری به تغییرات پارامتری دارد.

۱۰-۳ پاسخ به اغتشاش

```
% Simulate the integral tracker with disturbance
x_integral = x0_integral;
x_integral_traj_disturbance = zeros(n + m, ct);
for k = 2:ct
    u_integral_traj(k) = [-K1 -K2] * x_integral;
    x_integral = x_integral + 0.01 * (Abar * x_integral + Bbar * u_integral_traj(k)) + [zeros(n, 1); ones(m, 1)] * yd + [disturbance; zeros(m, 1)];
    x_integral_traj_disturbance(:, k) = x_integral;
end
y_integral_traj_disturbance = C * x_integral_traj_disturbance(1:n, :);
```



ردیاب استاتیک در مواجهه با اغتشاش، واکنشی سریع نشان می‌دهد. خروجی سیستم ابتدا یک افزایش کوتاه‌مدت تا حدود ۲ را تجربه می‌کند، اما سپس به سرعت کاهش یافته و در نهایت در یک مقدار منفی ثابت به حالت پایدار می‌رسد. این رفتار نشان می‌دهد که ردیاب استاتیک قادر به کاهش اثر اغتشاش است، اما نمی‌تواند آن را به طور کامل حذف کند و یک خطای حالت ماندگار باقی می‌ماند.

در مقابل، ردیاب انتگرالی رفتار متفاوتی را نشان می‌دهد. این سیستم با یک فراجهش قابل توجه تا حدود ۴۰۰ شروع می‌شود، که نشان‌دهنده حساسیت بالای آن به اغتشاش است. با این حال، پس از این فراجهش اولیه، سیستم به تدریج به سمت یک مقدار ثابت مثبت حدود ۱۰۰ همگرا می‌شود. این رفتار نشان می‌دهد که ردیاب انتگرالی، علی‌رغم واکنش شدید اولیه، قادر است اثر اغتشاش را تا حدی جبران کند، اما همچنان یک انحراف ثابت از مقدار مطلوب باقی می‌ماند.

۱۱- طراحی رویتگر

برای نمایش فضای حالت بدست آمده برای سیستم مفروض، یک رویتگر مرتبه کامل طراحی کنید. ملاک انتخاب قطبهای رویتگر حالت چیست؟ متغیرهای حالت سیستم و خطای تخمین را رسم کنید. در این سوال دو دسته قطب کند و سریع برای رویتگر انتخاب کنید و رویتگر مرتبه کامل متناظر را طراحی کنید. پاسخهای متغیرهای حالت اصلی سیستم و متغیرهای حالت تخمین زده شده توسط دو رویتگر مرتبه کامل را رسم و مقایسه کنید.

قطب‌های رویتگر با توجه به دو معیار مهم انتخاب می‌شوند: سرعت پاسخ‌دهی و توانایی فیزیکی سیستم در جایگذاری قطب‌ها. سرعت پاسخ‌دهی به این معناست که هرچه قطب‌ها از مبدأ در صفحه مختلط فاصله بیشتری داشته باشند، سیستم رویتگر با سرعت بیشتری حالت‌های داخلی سیستم را تخمین می‌زند. به عبارت دیگر، قطب‌های دورتر منجر به سرعت تخمین بالاتری می‌شوند، اما باید توجه داشت که این انتخاب نباید به پایداری سیستم لطمه بزند.

همچنین لازم به ذکر است که سیستم مشاهده‌پذیر است. مشاهده‌پذیری به این معناست که تمامی حالت‌های داخلی سیستم می‌توانند از طریق خروجی‌های قابل اندازه‌گیری قابل دسترسی باشند. بنابراین، شرط لازم برای طراحی رویتگر فراهم است و می‌توانیم با اطمینان بیشتری به طراحی و پیاده‌سازی آن بپردازیم. این امر به ما اجازه می‌دهد تا یک رویتگر بهینه و موثر برای سیستم مورد نظر طراحی کنیم که می‌تواند با دقت بالا حالت‌های داخلی سیستم را تخمین بزند.

۱-۱۱ طراحی رویت گر با قطب های نزدیک

میدانیم که طراحی رویتگر برای سیستم رویت پذیر مانند سیستم ما به صورت مرتبه کامل از رابطه زیر قابل محاسبه است.

$$\dot{e} = (A - LC)e \qquad (۱۱,۱)$$

```
% Desired pole locations for the observer
P_slow = [-1 -2 -3];  % Poles for slow observer
P_fast = [-10 -20 -30];  % Poles for fast observer
```

در این بخش قطب‌های مورد نظر برای دو نوع رویتگر تعریف شده‌اند: یکی با قطب‌های آهسته و دیگری با قطب‌های سریع. قطب‌های آهسته و سریع نمایانگر پاسخ سیستم با دینامیک‌های مختلف هستند.

```
% Calculate the observer gain for slow poles
K_slow = acker(A', C', P_slow);
L_slow = K_slow';


% Calculate the observer gain for fast poles
K_fast = acker(A', C', P_fast);
L_fast = K_fast';
```

اینجا از تابع acker برای محاسبه بهره‌های رویتگر استفاده شده است. تابع acker ماتریس‌های حالت و قطب‌های مورد نظر را گرفته و بهره‌های رویتگر را محاسبه می‌کند. سپس این بهره‌ها با استفاده از K’ به L تبدیل می‌شوند.

```
% Initial state and observer initial state
x0 = [2; 0; 0];
x0_est_slow = [0; 0; 0];
x0_est_fast = [0; 0; 0];


% Initial combined state for ODE solver
x0_combined_slow = [x0; x0_est_slow];
x0_combined_fast = [x0; x0_est_fast];
```

اینجا حالت اولیه سیستم (x0) و حالت اولیه رویتگر (x0_est_slow و x0_est_fast) تعریف شده‌اند. سپس این حالت‌ها برای شبیه‌سازی با هم ترکیب می‌شوند.

```
% Simulate the system with slow poles
[t_slow, x_slow] = ode45(@(t, x) observer_dynamics(t, x, A, B, C, L_slow), tspan, x0_combined_slow);


% Simulate the system with fast poles
[t_fast, x_fast] = ode45(@(t, x) observer_dynamics(t, x, A, B, C, L_fast), tspan, x0_combined_fast);
```

در این بخش از تابع ode45 برای شبیه‌سازی دینامیک سیستم و رویتگر استفاده شده است. تابع observer_dynamics دینامیک سیستم را با در نظر گرفتن بهره‌های رویتگر شبیه‌سازی می‌کند.

```
% Extract true states and estimated states for slow poles
    x_true_slow = x_slow(:, 1:3);
    x_est_slow = x_slow(:, 4:6);
% Extract true states and estimated states for fast poles
    x_true_fast = x_fast(:, 1:3);
    x_est_fast = x_fast(:, 4:6);
```

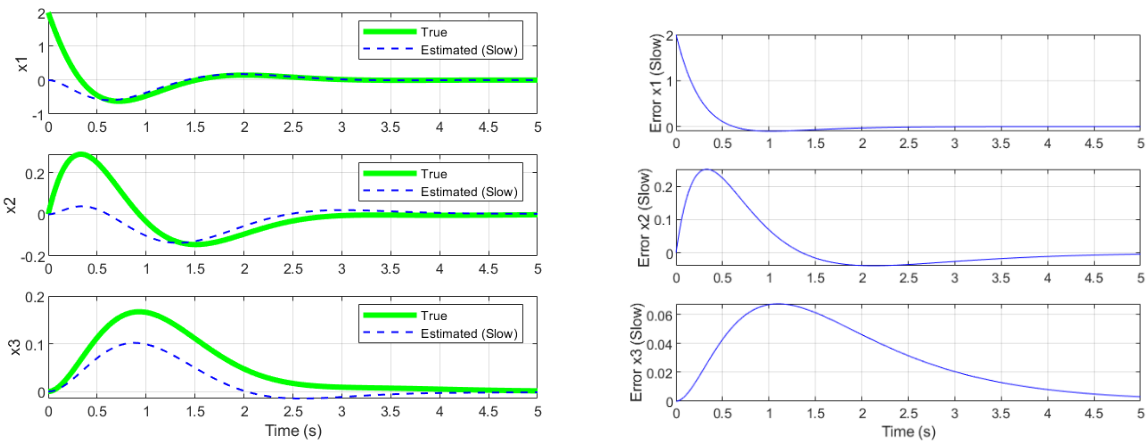
در اینجا حالت‌های واقعی و تخمینی از نتایج شبیه‌سازی استخراج می‌شوند. حالت‌های واقعی (x_true_slow و x_true_fast) مربوط به سیستم اصلی و حالت‌های تخمینی (x_est_slow و x_est_fast) مربوط به رویتگر هستند.

```
% Calculate estimation error for slow poles
    error_slow = x_true_slow - x_est_slow;
% Calculate estimation error for fast poles
    error_fast = x_true_fast - x_est_fast;
```

در این بخش خطای تخمین برای قطب‌های آهسته و سریع محاسبه می‌شود که تفاوت بین حالت‌های واقعی و تخمینی است.

و در نهایت نمایش تخمینگر در برابر مقدار اصلی و همچنین نمایش مقدار خطای ما در زیر آورده شده است.

```
figure;
subplot(3,1,1);
    plot(t_slow, x_true_slow(:, 1), 'g', 'linewidth', 3);
    hold on;
    plot(t_slow, x_est_slow(:, 1), 'b--', 'linewidth', 1);
    ylabel('x1');
    legend('True', 'Estimated (Slow)');
grid on;
subplot(3,1,2);
    plot(t_slow, x_true_slow(:, 2), 'g', 'linewidth', 3);
    hold on;
    plot(t_slow, x_est_slow(:, 2), 'b--', 'linewidth', 1);
    ylabel('x2');
    legend('True', 'Estimated (Slow)');
grid on;
subplot(3,1,3);
    plot(t_slow, x_true_slow(:, 3), 'g', 'linewidth', 3);
    hold on;
    plot(t_slow, x_est_slow(:, 3), 'b--', 'linewidth', 1);
    ylabel('x3');
    legend('True', 'Estimated (Slow)');
    xlabel('Time (s)');
grid on;
```



پس مشاهده میشود که با قطب های نزدیک مقادیر واقعی و مقادیر تخمین زده شده به همراه خطاهای این تخمین برای قطب های زیر بصورت بالا میباشد.

```
P_slow = [-1 -2 -3];  % Poles for slow observer
```

```
Observer gain L (slow poles):
   -5.5048
    1.0165
    0.3634
```

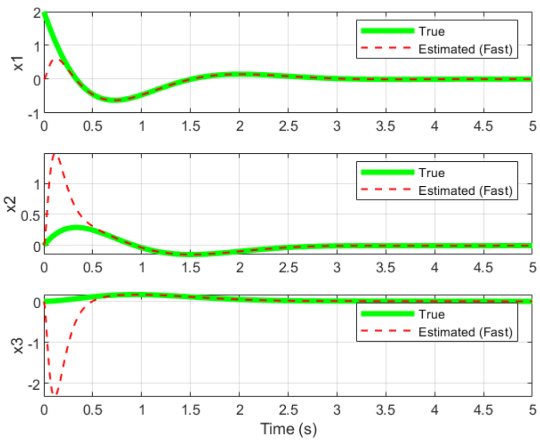
۲-۱۱ طراحی رویت‌گر با قطب های دور

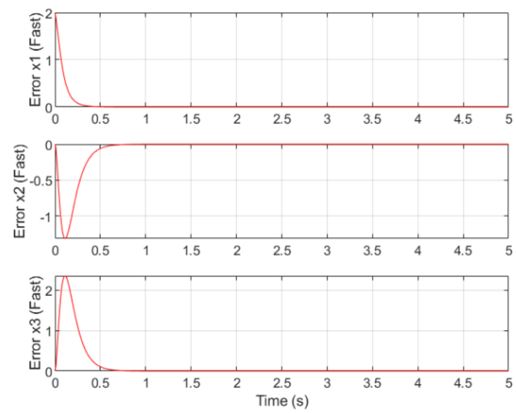
در این بخش، همانند قسمت قبلی، به طراحی رویتگر پرداخته و قطب‌ها را در مکان‌های مناسب جایگذاری می‌کنیم. برای این مرحله، قطب‌ها در مقادیر دورتر انتخاب شده‌اند. پس از انجام طراحی و شبیه‌سازی پاسخ سیستم، نتایج حاصل را مشاهده می‌کنیم. همانطور که در شکل‌ها به وضوح قابل مشاهده است، برای این بخش نیز پاسخ مناسبی به دست می‌آید.

به طور کلی، می‌دانیم که هرچه قطب‌های سیستم از مبدأ در صفحه مختلط دورتر باشند، سرعت و دقت تخمین افزایش می‌یابد. این موضوع در مورد سیستم ما نیز کاملاً صادق است و نتایج شبیه‌سازی‌ها این نکته را تأیید می‌کنند. با جایگذاری قطب‌ها در مکان‌های دورتر از مبدأ، رویتگر قادر خواهد بود با سرعت و دقت بیشتری حالت‌های داخلی سیستم را تخمین بزند. این امر باعث بهبود عملکرد سیستم و کاهش خطای تخمین می‌شود.

```
P_fast = [-10 -20 -30];  % Poles for fast observer
```

```
Observer gain L (fast poles):
   1.0e+03 *

    0.2202
    0.6273
   -1.0557
```





در نهایت، نتایج نشان می‌دهد که طراحی رویتگر با قطب‌های انتخاب شده منجر به بهبود چشمگیری در دقت و سرعت تخمین حالت‌های داخلی سیستم شده است. این موضوع اهمیت انتخاب مناسب قطب‌ها در طراحی رویتگر را برجسته می‌کند و تأکید می‌کند که انتخاب بهینه قطب‌ها می‌تواند تأثیر بسزایی در عملکرد نهایی سیستم داشته باشد.

۱۲- تغییر پارامترها و مقایسه سیستم

در این بخش، با تغییر برخی از پارامترهای سیستم، اثر این تغییرات را بر روی تخمین حالت‌های سیستم توسط رویتگر بررسی می‌کنیم. ابتدا ماتریس A و سپس ماتریس C را تغییر داده و نتایج را تحلیل می‌کنیم.

۱۲-۱ تغییر ماتریس A

```
% Change A matrix (5 times) and simulate
A_changed = 5 * A;
```

در این قسمت، ماتریس A را ۵ برابر می‌کنیم. این کار باعث می‌شود تا دینامیک سیستم تغییر کند و رفتار جدیدی از سیستم مشاهده شود.

```
[t_slow_A, x_slow_A] = ode45(@(t, x) observer_dynamics(t, x, A_changed, B, C, L_slow), tspan, x0_combined_slow);
[t_fast_A, x_fast_A] = ode45(@(t, x) observer_dynamics(t, x, A_changed, B, C, L_fast), tspan, x0_combined_fast);
```

در این قسمت، سیستم را با ماتریس تغییر یافته شبیه‌سازی می‌کنیم. از تابع ode45 برای حل معادلات دیفرانسیل استفاده می‌شود. پارامترهای ورودی شامل ماتریس‌های A تغییر یافته،B،Cو بهره‌های رویتگر L برای قطب‌های آهسته و سریع هستند.

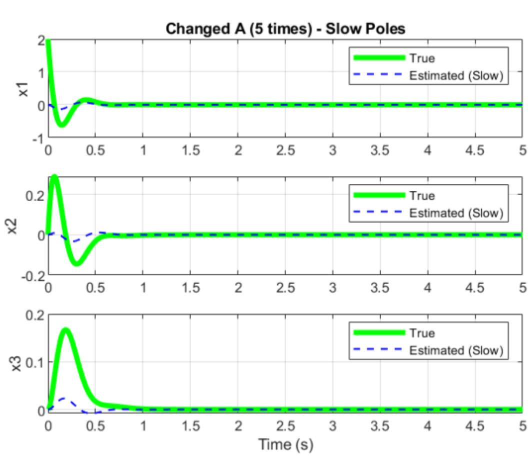
```
x_true_slow_A = x_slow_A(:, 1:3);
x_est_slow_A = x_slow_A(:, 4:6);
x_true_fast_A = x_fast_A(:, 1:3);
x_est_fast_A = x_fast_A(:, 4:6);
error_slow_A = x_true_slow_A - x_est_slow_A;
error_fast_A = x_true_fast_A - x_est_fast_A;
```

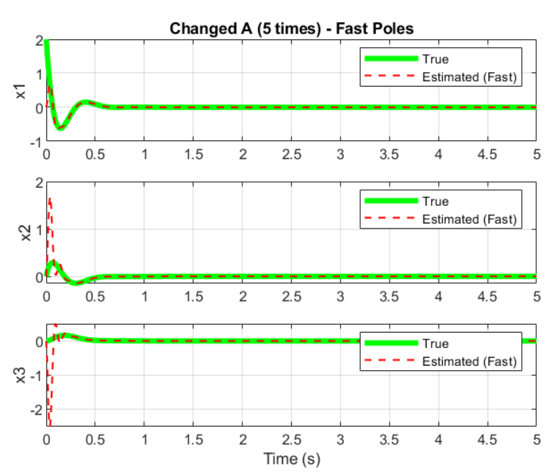
در این بخش، حالات واقعی و تخمینی از نتایج شبیه‌سازی استخراج می‌شوند. حالات واقعی در ستون‌های اول تا سوم و حالات تخمینی در ستون‌های چهارم تا ششم ذخیره شده‌اند. همچنین خطای تخمین نیز محاسبه می‌شود.

در نهایت مطابق کدهای زیر نمودار ها را رسم میکنیم.

```
% Plot true states and estimated states with changed A for slow poles
figure;
subplot(3,1,1);
        plot(t_slow_A, x_true_slow_A(:, 1), 'g', 'linewidth', 3);
        hold on;
        plot(t_slow_A, x_est_slow_A(:, 1), 'b--', 'linewidth', 1);
        ylabel('x1');
        legend('True', 'Estimated (Slow)');
        title('Changed A (5 times) - Slow Poles');
grid on;
subplot(3,1,2);
        plot(t_slow_A, x_true_slow_A(:, 2), 'g', 'linewidth', 3);
        hold on;
        plot(t_slow_A, x_est_slow_A(:, 2), 'b--', 'linewidth', 1);
        ylabel('x2');
        legend('True', 'Estimated (Slow)');
grid on;
subplot(3,1,3);
        plot(t_slow_A, x_true_slow_A(:, 3), 'g', 'linewidth', 3);
        hold on;
        plot(t_slow_A, x_est_slow_A(:, 3), 'b--', 'linewidth', 1);
        ylabel('x3');
        legend('True', 'Estimated (Slow)');
        xlabel('Time (s)');
grid on;
```

```
% Plot true states and estimated states with changed A for fast poles
figure;
subplot(3,1,1);
    plot(t_fast_A, x_true_fast_A(:, 1), 'g', 'linewidth', 3);
    hold on;
    plot(t_fast_A, x_est_fast_A(:, 1), 'r--', 'linewidth', 1);
    ylabel('x1');
    legend('True', 'Estimated (Fast)');
    title('Changed A (5 times) - Fast Poles');
grid on;
subplot(3,1,2);
    plot(t_fast_A, x_true_fast_A(:, 2), 'g', 'linewidth', 3);
    hold on;
    plot(t_fast_A, x_est_fast_A(:, 2), 'r--', 'linewidth', 1);
    ylabel('x2');
    legend('True', 'Estimated (Fast)');
grid on;
subplot(3,1,3);
    plot(t_fast_A, x_true_fast_A(:, 3), 'g', 'linewidth', 3);
    hold on;
    plot(t_fast_A, x_est_fast_A(:, 3), 'r--', 'linewidth', 1);
    ylabel('x3');
    legend('True', 'Estimated (Fast)');
    xlabel('Time (s)');
grid on;
```

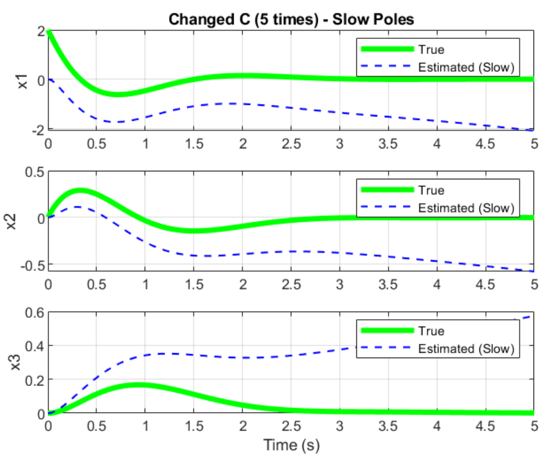


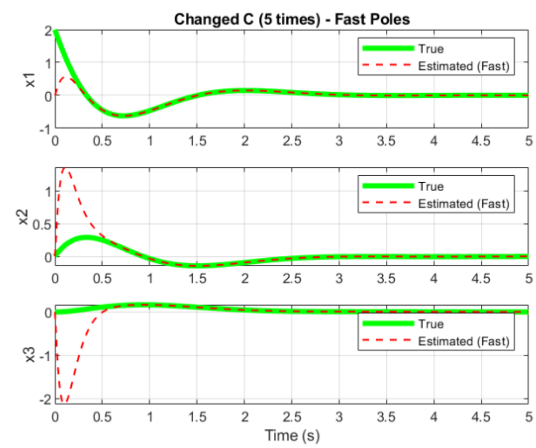


همانگونه که مشاهده میشود اگر ماتریس A را ۵ برابر کنیم زمانی که طول می کشد تا خطا صفر شود حدودا ۲.۵ برابر می شود.

۱۲-۲ تغییر ماتریس C

مجددا همانند بخش قبل ابتدا در این قسمت، ماتریس Cرا ۵ برابر می‌کنیم. این کار باعث می‌شود تا ویژگی‌های مشاهده‌پذیری سیستم تغییر کند و رفتار جدیدی از سیستم مشاهده شود. سپس همانند قبل سیستم را با ماتریس Cتغییر یافته شبیه‌سازی می‌کنیم. پارامترهای ورودی شامل ماتریس‌های A، B، C(تغییر یافته) و بهره‌های رویتگر (L) برای قطب‌های آهسته و سریع هستند.





اگر ماتریس C را ۵ برابر کنیم تغییر زیادی ایجاد نمی شود.

۱۳- تخمینگر کاهش مرتبه

برای طراحی این رویتگر، ابتدا باید بررسی کنیم که آیا ماتریس (C) به فرم $[I_q \quad 0]$ است یا خیر. به وضوح، این شرط برقرار نیست؛ بنابراین، لازم است از تبدیل‌هایی استفاده کنیم تا سیستم مطلوب را به دست آوریم.

تبدیل مورد نظر، همان‌طور که در شرح درس گفته شد، شامل استفاده از ماتریس (C) و یک ماتریس دلخواه دیگر برای ساخت یک ماتریس مرتبه کامل است که مرتبه آن برابر با (n) باشد. این فرآیند به ما کمک می‌کند تا سیستم جدیدی را تعریف کنیم که دارای خصوصیات مورد نظر برای طراحی رویتگر است.

ابتدا یک ماتریس دلخواه (T) را انتخاب می‌کنیم که همراه با (C) یک ماتریس مرتبه کامل تشکیل دهد. این ماتریس باید به گونه‌ای انتخاب شود که ماتریس ترکیبی $[C \ ; \ T]$ دارای رتبه کامل باشد و تمام خصوصیات مشاهده‌پذیری سیستم را حفظ کند. برای انجام این تبدیل‌ها، ماتریس تبدیل (P) را محاسبه می‌کنیم که

ماتریس‌های سیستم اصلی را به سیستم جدید تبدیل می‌کند. این ماتریس تبدیل به ما کمک می‌کند تا حالت‌های جدید و ماتریس‌های جدید سیستم را بدست آوریم. با استفاده از سیستم جدید، رویتگر مورد نظر را طراحی می‌کنیم. رویتگر باید به گونه‌ای طراحی شود که حالت‌های سیستم را به طور دقیق تخمین بزند و خطای تخمین را به حداقل برساند. بهره‌های رویتگر (L) با توجه به قطب‌های مورد نظر سیستم انتخاب می‌شوند تا سرعت و دقت تخمین افزایش یابد.

```
% Define P matrix
P = [C; 1 0 0; 0 1 0];

% Check the rank of P
disp('Rank of P:');
disp(rank(P));
```

```
Rank of P:
     3
```

در این بخش ماتریس P تعریف می‌شود که شامل ماتریس C و دو سطر دیگر است. سپس رتبه ماتریس P محاسبه و نمایش داده می‌شود. این بررسی برای اطمینان از اینکه P دارای رتبه کامل است انجام می‌گیرد.

```
% Desired poles for the reduced-order observer
P_desire = [-8 -9];

% Transform the system
Abar = P * A / P;
Bbar = P * B;
Cbar = C / P;
```

قطب‌های مورد نظر برای رویتگر با استفاده از P_desire تعریف می‌شوند. سپس سیستم اصلی با استفاده از ماتریس P تبدیل می‌شود تا به فرم مطلوب برای طراحی رویتگر برسیم.

```
% Partition the transformed system
A22 = Abar(2:3, 2:3);
A12 = Abar(1, 2:3);
A21 = Abar(2:3, 1);
A11 = Abar(1, 1);
B1 = Bbar(1, 1);
B2 = Bbar(2:3, 1);
```

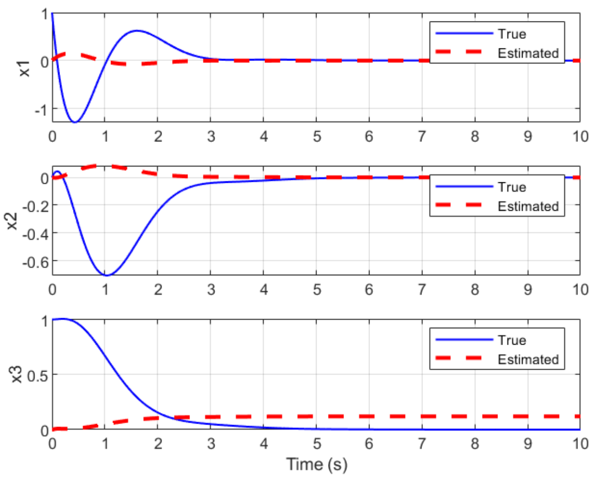
```
% Calculate the observer gain
K = place(A22', A12', P_desire);
L = K';
% Display the observer gain
        disp('Observer gain L:');
        disp(L);
```

سیستم تبدیل شده به زیر ماتریس‌هایی تقسیم می‌شود که برای طراحی رویتگر لازم هستند. تابع place برای جایگذاری قطب‌ها و محاسبه بهره رویتگر L استفاده می‌شود.

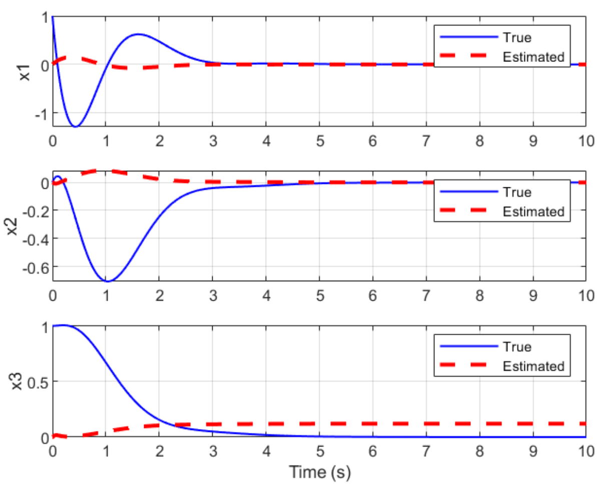
```
Observer gain L:
    4.3436
    7.5303
```

```
% Initial conditions
xn = [0; 0; 0];
tspan = 0:0.01:10;
ct = length(tspan);
z = zeros(2, ct);
y = zeros(1, ct);

% Simulate the system
for k = 2:ct
   y(k) = Cbar * xn(:, k-1);
   xn(:, k) = xn(:, k-1) + 0.01 * (Abar * xn(:, k-1) + Bbar);
   z(:, k) = z(:, k-1) + 0.01 * ((A22 - L * A12) * z(:, k-1) + ((A22 - L * A12) * L + A21) * y(k-1) + (B2 - L * B1));
end

% Calculate the estimated state
x_hat = inv(P) * [1 0 0; L eye(2)] * [y; z];

% Simulate the true system
[t1, x] = ode45(@(t, x) system_dynamics(t, x, A, B), tspan, [1; 0; 1]);
```

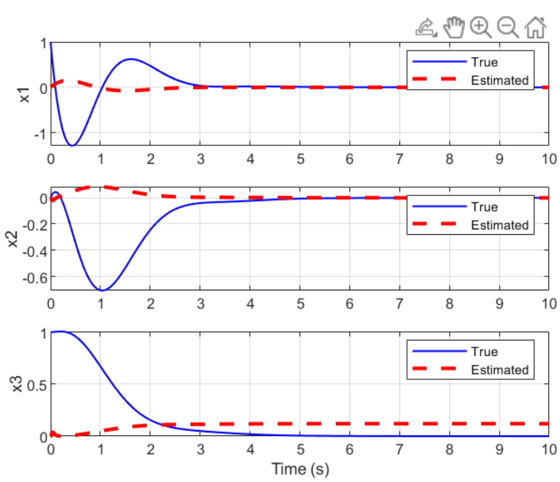
شرایط اولیه و بردار زمان تعریف می‌شوند. یک حلقه for برای شبیه‌سازی سیستم با استفاده از روش اویلر نوشته شده است. خروجی سیستم و تخمین حالات سیستم محاسبه می‌شوند. در انتها، حالات تخمینی با استفاده از ماتریس P و ماتریس‌های دیگر محاسبه می‌شوند. این بخش از کد با استفاده از تابع ode45 سیستم واقعی را شبیه‌سازی می‌کند. سپس نتایج شبیه‌سازی شامل حالات واقعی و تخمینی سیستم در سه زیر نمودار رسم می‌شوند.



همانطور که مشاهده میشود تخمینگر طراحی شده توانسته تخمینی از سیستم داشته باشد اما عملکرد آن قوی نیست، پس در مرحله بعد قطب تغییر میدهیم.




```
Observer gain L:
    3.6771
   26.6117
```




```
Observer gain L:
  -19.4957
  105.0364
```

۱۴- رگولاتور با تخمینگر مرتبه کامل

طبق خواسته سوال این طراحی را انجام میدهیم اما به طور کلی سیستم از ابتدا از نظر overshoot مشکلی نداشتیم. برای تعیین زمان نشست نیز طبق خواسته طراحی میتوان قطب‌های فیدبک را در مکان دورتری قرار داد تا پاسخ سریع تری پیدا کند اما این عمل باعث پایین آمدن مقدار نهایی خواهد شد که با تغییر دامنه پله میتوانیم این مشکل را نیز برطرف کنیم. یا از کنترل کننده های دیگری مانند PI استفاده کنیم. اما برای این مرحله از ما چنین چیزی خواسته نشده است.

```
% Simulate the system with observer and state feedback
[t, x] = ode45(@d_Q14, tspan, zeros(6,1));
X = x';
```

در این بخش، سیستم با استفاده از تابع ode45 شبیه‌سازی می‌شود. تابع ode45 یک حل‌کننده عددی برای معادلات دیفرانسیل است d_Q14. تابعی است که دینامیک سیستم را تعریف می‌کند. شرایط اولیه برای سیستم به عنوان یک بردار صفر با ابعاد ۶ (۳ حالت اصلی و ۳ حالت تخمین زده شده) تنظیم می‌شود. نتایج شبیه‌سازی در متغیر x ذخیره می‌شوند و سپس به X انتقال داده می‌شوند.

```
% Q -14 fcn:
function Z = d_Q14(t, x)
   numerator_coeffs = [3 1.677];
   denominator_coeffs = [2 8.341 21.55 16.61];
   [A, B, C, D] = tf2ss(numerator_coeffs, denominator_coeffs);

   P_observer = [-20 -25 -30];
   P_controller = [-3 -4 -5];

   % Calculate observer gain
   K_observer = acker(A', C', P_observer);
   L = K_observer';

   % Calculate state feedback gain
   K = acker(A, B, P_controller);

   r = 1;
   Ga = inv(C * inv(-A + B * K) * B);
   u = Ga * r;

   G = [A -B * K; L * C A - L * C - B * K];
   Z = G * x + [B; B] * u;
end
```

تحلیل کد:

این تابع دینامیک سیستم حلقه بسته با رویتگر مرتبه کامل و فیدبک حالت را تعریف می‌کند. با استفاده از قطب‌های مناسب برای رویتگر و کنترل‌کننده، می‌توان عملکرد سیستم را بهینه کرد. قطب‌های سریع‌تر برای رویتگر، تضمین می‌کنند که تخمین حالات سیستم به سرعت همگرا می‌شوند و قطب‌های کنترل‌کننده، پاسخ مناسب سیستم را تضمین می‌کنند. در نهایت، نرخ تغییرات حالات سیستم برای شبیه‌سازی و تحلیل عملکرد سیستم مورد استفاده قرار می‌گیرد.

قطب‌های مورد نظر برای رویتگر مرتبه کامل و کنترل‌کننده فیدبک حالت تعریف می‌شوند. قطب‌های رویتگر باید بیشتر از قطب‌های سیستم باشند تا رویتگر سریع‌تر از سیستم اصلی باشد. قطب‌های کنترل‌کننده نیز باید به گونه‌ای انتخاب شوند که پاسخ پله به لحاظ فراجهش و زمان نشست رفتار قابل قبولی داشته باشد.

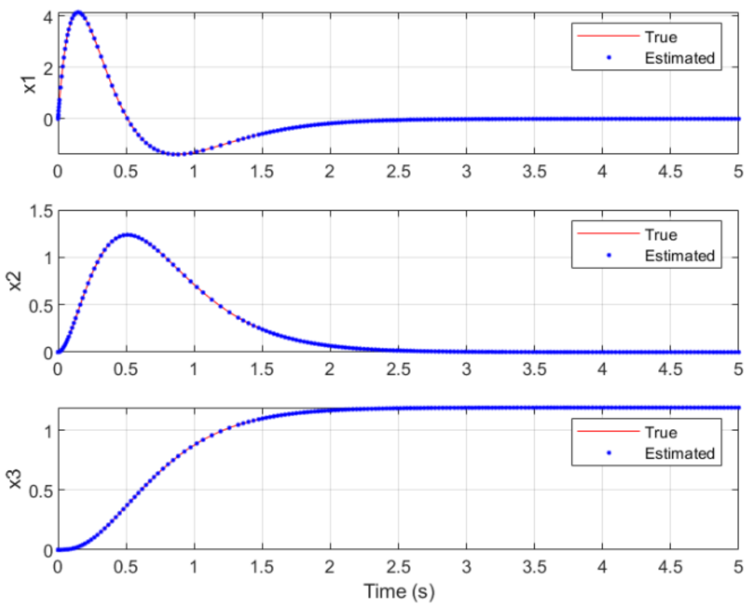
```
P_observer = [-20 -25 -30];
P_controller = [-3 -4 -5];
```

در این بخش، بهره رویتگر با استفاده از تابع acker محاسبه می‌شود. این تابع ماتریس‌های حالت (A') و خروجی (C') و قطب‌های مورد نظر رویتگر (P_observer) را دریافت کرده و بهره رویتگر (K_observer) را محاسبه می‌کند. سپس بهره رویتگر (L) به دست می‌آید.

```
% Calculate observer gain
K_observer = acker(A', C', P_observer);
L = K_observer';
```

در این بخش، بهره فیدبک حالت با استفاده از تابع acker محاسبه می‌شود. این تابع ماتریس‌های حالت (A) و ورودی (B) و قطب‌های مورد نظر کنترل‌کننده (P_controller) را دریافت کرده و بهره فیدبک حالت (K) را محاسبه می‌کند.

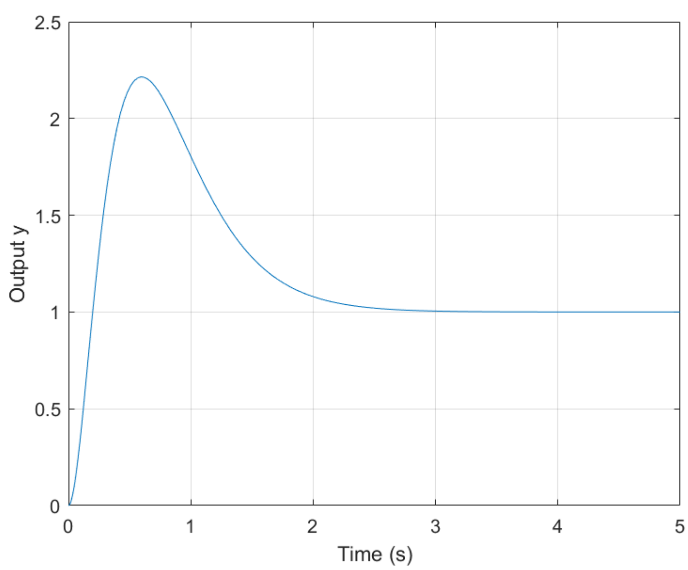
```
% Calculate state feedback gain
K = acker(A, B, P_controller);
```



همانطور که از شکل پیداست طراحی سیستم موفق بود و توانستیم حالت های سیستم فیدبک را با استفاده از تخمینگر تخمین بزنیم.

در ضمن خروجی پاسخ پله سیستم اصلی و تخمین زده شده نیز به شرح زیر است.

```
figure;
plot(t, y, '-');
ylabel('Output y');
xlabel('Time (s)');
grid on;
```



۱۵- رگولاتور با تخمینگر کاهش مرتبه

برای سیستم مفروض یک سیستم کنترل (رگولاتور) فیدبک حالت با رویتگر مرتبه کاهش یافته طراحی کنید.

این کد به طراحی و شبیه‌سازی یک سیستم کنترل فیدبک حالت با رویتگر مرتبه کاهش یافته می‌پردازد. هدف اصلی این است که برای سیستمی که تمامی حالات آن قابل دسترس نیست، یک رویتگر مرتبه کاهش یافته طراحی شود تا بتوانیم حالات سیستم را تخمین بزنیم و از آن‌ها برای فیدبک حالت استفاده کنیم.در این بخش نیز مانند بخش پیشین عمل میکنیم با این تفاوت که تخمینگر مربوطه در این بخش تخمینگر کاهش مرتبه است.

تحلیل کد

با استفاده از این کد، سیستم حلقه بسته‌ای طراحی شده که از رویتگر مرتبه کاهش یافته برای تخمین حالات و فیدبک حالت استفاده می‌کند. قطب‌های رویتگر به گونه‌ای انتخاب شده‌اند که سریع‌تر از سیستم اصلی عمل کند، به طوری که تخمین حالات به سرعت همگرا شود. همچنین قطب‌های کنترل‌کننده به گونه‌ای انتخاب شده‌اند که پاسخ سیستم به لحاظ فراجهش و زمان نشست رفتار قابل قبولی داشته باشد. نمودارهای نهایی نشان می‌دهند که حالات تخمین زده شده به خوبی با حالات واقعی تطابق دارند و سیستم حلقه بسته عملکرد مطلوبی دارد.

```
% Define P matrix
P = [C; 1 0 0; 0 1 0];
inv_p = inv(P);
P1 = inv_p(1:3,1);
P2 = inv_p(1:3,2:3);
Abar = P * A * inv(P);
Bbar = P * B;
```

در این بخش، ماتریس P تعریف می‌شود که ترکیبی از ماتریس C و ماتریسR است. سپس معکوس P محاسبه شده و به دو بخش P1 و P2 تقسیم می‌شود. ماتریس‌های A و B با استفاده از P به فرم جدیدی تبدیل می‌شوند تا محاسبات رویتگر مرتبه کاهش یافته ساده‌تر شود.

```
A22 = Abar(2:3, 2:3);
A12 = Abar(1, 2:3);
A21 = Abar(2:3, 1);
A11 = Abar(1, 1);
B1 = Bbar(1, 1);
B2 = Bbar(2:3, 1);
```

ماتریس‌های Abar و Bbar به زیرماتریس‌های کوچک‌تری تقسیم می‌شوند که در مراحل بعدی برای محاسبه بهره‌های رویتگر و فیدبک حالت استفاده خواهند شد.

```
% Desired poles for the controller and reduced-order observer
P_controller = [-5 -7 -10];
P_observer = [-7 -10];

% Calculate state feedback gain
K = place(A, B, P_controller)

% Calculate observer gain
k = place(A22', A12', P_observer);
L = k'
```

```
K = 1×3
   17.8295  144.2250  341.6950
L = 2×1
    4.4657
    7.3117
```

قطب‌های مطلوب برای رویتگر مرتبه کاهش یافته و کنترل‌کننده فیدبک حالت تعیین می‌شوند. تابع place برای محاسبه بهره فیدبک حالت K و بهره رویتگر L استفاده می‌شود. بهره رویتگر به صورت L=K′ تعریف می‌شود.

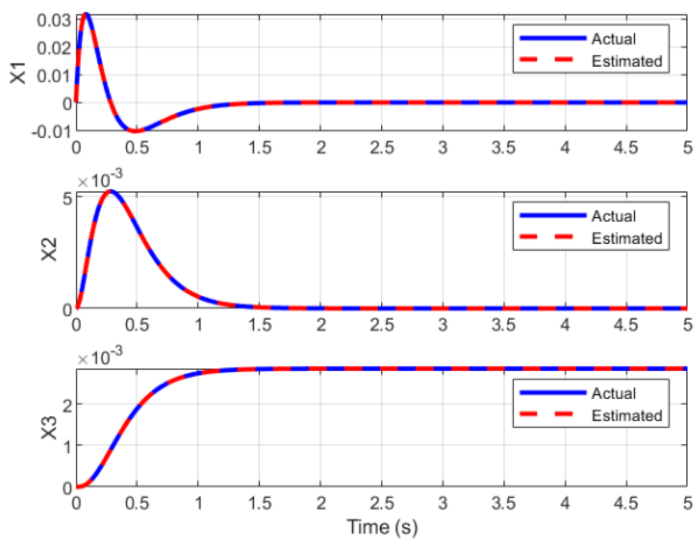
```
% Reduced-order observer parameters
B_l = B2 - L * B1;
AA = [A - B * K * P1 * C - B * K * P2 * L * C, -B * K * P2;
    ((A22 - L * A12) * L + (A21 - L * A11)) * C - B_l * K * P1 * C - B_l * K * P2 * L * C, (A22 - L * A12) - B_l * K * P2];
BB = [B; B_l];
CC = [C zeros(1, 2)];
```

در این بخش، پارامترهای رویتگر مرتبه کاهش یافته محاسبه می‌شوند. این پارامترها شامل ماتریس‌های AA، BB و CC هستند که برای شبیه‌سازی سیستم حلقه بسته استفاده می‌شوند.

```
% Time vector
t = linspace(0, 5, 1000);

% Simulate the closed-loop system
sys = ss(AA, BB, CC, 0);
[y,~,x] = lsim(sys, ones(1,1000), t);

% Extract true and estimated states
x_t = x(:,1:3);
z = x(:,4:5);
x_est = inv(P) * [y'; L * y' + z'];
```

یک بردار زمان برای شبیه‌سازی تعریف می‌شود و سپس سیستم حلقه بسته با استفاده از تابع lsim شبیه‌سازی می‌شود. تابع ss یک سیستم فضای حالت با استفاده از ماتریس‌های AA، BBو CC تعریف می‌کند و تابع lsim

پاسخ سیستم به ورودی واحد را شبیه‌سازی می‌کند. حالات واقعی سیستم (xtx_txt) و حالات تخمین زده شده (xestx_estxest) استخراج می‌شوند. این حالات برای مقایسه عملکرد رویتگر و کنترل‌کننده استفاده می‌شوند.



نتایج شبیه‌سازی شامل حالات واقعی و تخمین زده شده سیستم در سه زیرنمودار رسم می‌شوند. این نمودارها مقایسه‌ای از حالات واقعی و تخمین زده شده را نشان می‌دهند و عملکرد رویتگر را ارزیابی می‌کنند.

۱۶- ردیاب با تخمینگر مرتبه کامل

برای سیستم مفروض، یک سیستم کنترل (ردیاب) فیدبک حالت با رویتگر مرتبه کامل طراحی کنید.

تحلیل کد

با استفاده از این کد، سیستم حلقه بسته‌ای طراحی شده که از رویتگر مرتبه کامل برای تخمین حالات و فیدبک حالت استفاده می‌کند. قطب‌های رویتگر به گونه‌ای انتخاب شده‌اند که سریع‌تر از سیستم اصلی عمل کند، به طوری که تخمین حالات به سرعت همگرا شود. همچنین قطب‌های کنترل‌کننده به گونه‌ای انتخاب شده‌اند که پاسخ سیستم به لحاظ فراجهش و زمان نشست رفتار قابل قبولی داشته باشد. نمودارهای نهایی نشان می‌دهند که حالات تخمین زده شده به خوبی با حالات واقعی تطابق دارند و سیستم حلقه بسته عملکرد مطلوبی دارد.

```
% Desired pole locations for the controller and observer
controller_poles = [-7 -10 -12];
observer_poles = [-5 -7 -10];

% Calculate state feedback gain
K = place(A, B, controller_poles)
```

```
% Calculate observer gain
L = place(A', C', observer_poles)'
```

```
K = 1×3
   24.8295  263.2250  831.6950
L = 3×1
   23.6454
   41.0244
  -52.1253
```

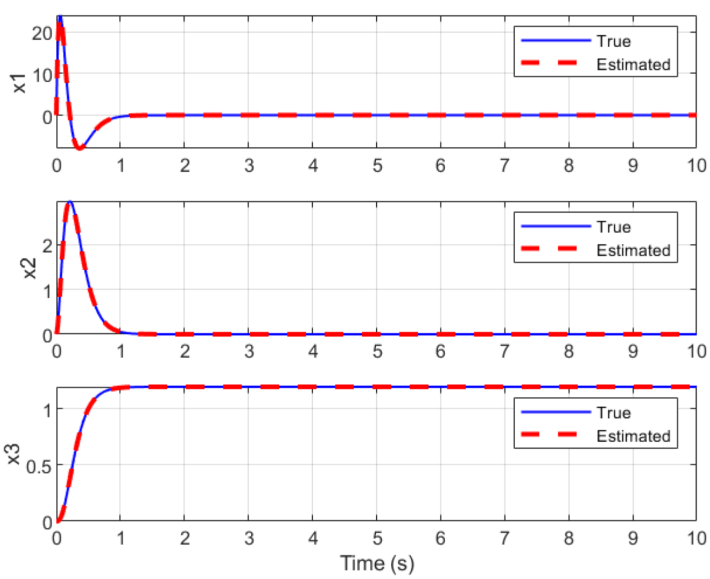
در این بخش، قطب‌های مطلوب برای کنترل‌کننده و رویتگر تعیین می‌شوند. تابع place برای محاسبه بهره فیدبک حالت K و بهره رویتگر L استفاده می‌شود. بهره رویتگر به صورت L=K′ تعریف می‌شود.

```
% Calculate the feedforward gain
Ga = inv(C * inv(-A + B * K) * B);
```

Ga بهره پیش‌خور که تضمین می‌کند سیستم به ورودی مرجع (r) پاسخ می‌دهد. این بهره به سیستم اجازه می‌دهد که به ورودی مرجع بدون خطای حالت ماندگار پاسخ دهد.

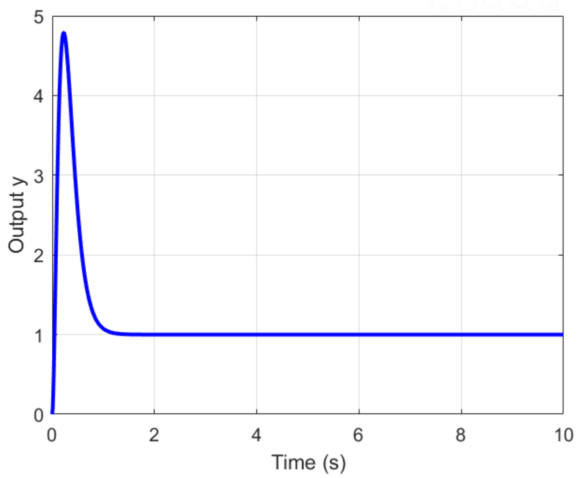
```
% Create state-space system
sys = ss(Abar, Bbar, Cbar, 0);

% Time vector for simulation
t = linspace(0, 10, 1000);

% Simulate the system response
[y, ~, x] = lsim(sys, Ga * ones(1, 1000), t);
```

ss این تابع یک سیستم فضای حالت با ماتریس‌های(Abar) ، (Bbar)، (Cbar)و ماتریس (۰) ایجاد می‌کند. linspaceبردار زمان را برای شبیه‌سازی تعریف می‌کند.lsim تابع پاسخ سیستم فضای حالت به ورودی مرجع را شبیه‌سازی می‌کند.



با توجه به نتایج شبیه‌سازی، سیستم طراحی شده به ورودی مرجع به خوبی پاسخ می‌دهد و تخمین حالات با دقت قابل قبولی انجام می‌شود. این نشان می‌دهد که طراحی سیستم کنترل و رویتگر به درستی انجام شده است.

در ضمن خروجی پاسخ پله سیستم اصلی و تخمین زده شده نیز به شرح زیر است.




```
figure;
plot(t, y, 'b-', 'linewidth', 2);
ylabel('Output y');
xlabel('Time (s)');
grid on;
```

۱۷- ردیاب با تخمینگر کاهش مرتبه

برای سیستم مفروض، یک سیستم کنترل (ردیاب) فیدبک حالت با رویتگر مرتبه کاهش یافته طراحی کنید.

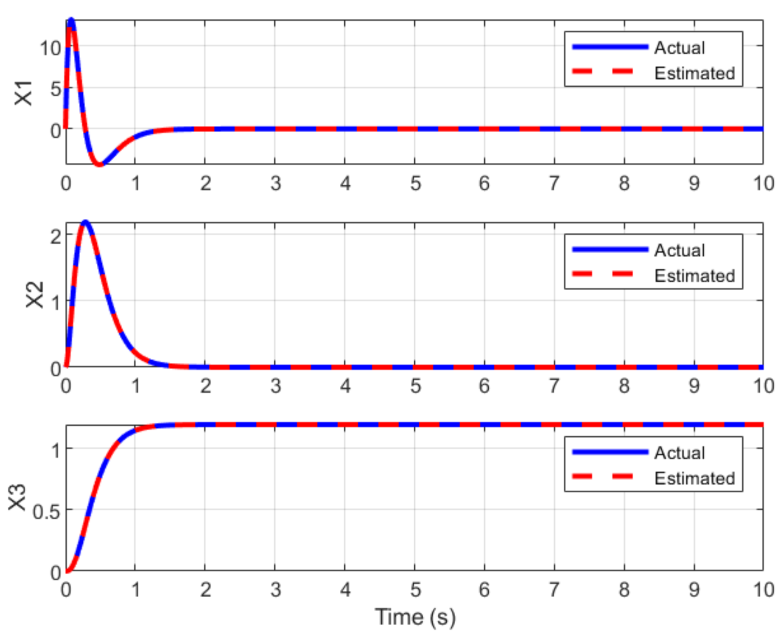
تحلیل کد

با استفاده از این کد، یک سیستم حلقه بسته طراحی شده که از رویتگر مرتبه کاهش یافته برای تخمین حالات و فیدبک حالت استفاده می‌کند. قطب‌های رویتگر به گونه‌ای انتخاب شده‌اند که سریع‌تر از سیستم اصلی عمل کند تا تخمین حالات به سرعت همگرا شود. همچنین، قطب‌های کنترل‌کننده به گونه‌ای انتخاب شده‌اند که پاسخ سیستم به لحاظ فراجهش و زمان نشست رفتار قابل قبولی داشته باشد. نمودارهای نهایی نشان می‌دهند که حالات تخمین زده شده به خوبی با حالات واقعی تطابق دارند و سیستم حلقه بسته عملکرد مطلوبی دارد.

- در این بخش تمامی قسمت‌ها مانند بخش ۱۵ میباشد تنها تفاوت این دو بخش حضور ضریب ردیاب استاتیک است.

```
P = [C; 1 0 0; 0 1 0];
inv_p = inv(P);
P1 = inv_p(1:3, 1);
P2 = inv_p(1:3, 2:3);
Abar = P * A / P;
Bbar = P * B;

A22 = Abar(2:3, 2:3);
A12 = Abar(1, 2:3);
A21 = Abar(2:3, 1);
A11 = Abar(1, 1);
B1 = Bbar(1, 1);
B2 = Bbar(2:3, 1);

P_controller = [-5 -7 -10];
P_observer = [-12 -15];

% Calculate state feedback gain
K = place(A, B, P_controller);

% Calculate observer gain
k = place(A22', A12', P_observer)
L = k'
```

```
k = 1×2
    4.7544   18.7213
L = 2×1
    4.7544
   18.7213
```



۱۸- بهره فیدبک بهینه LQR

بهره فیدبک حالت بهینه سیستم را برای حداقل سازی تابع هزینه زیر برای مقادیر مختلف ماتریس R و ماتریس Q بدست آورید.

$$J = \int (x^T Q x + u^T R u) dt$$

یک بار ماتریس Q را ثابت فرض کنید و ماتریس R تغییر دهید. نتایج شبیهسازی را با یکدیگر مقایسه کنید. بار دیگر ماتریس R را ثابت در نظر بگیرید و ماتریس Q را تغییر دهید. نتایج شبیهسازی را با یکدیگر مقایسه کنید.

در این سوال، هدف طراحی یک کنترل‌کننده بهینه با استفاده از روش LQR (Linear Quadratic Regulator) برای حداقل‌سازی تابع هزینه J است. تابع هزینه به صورت زیر تعریف شده است:

$$J = \int (x^T Q x + u^T R u) dt$$

که در آن Q ماتریس وزن‌دهی به حالات سیستم و R ماتریس وزن‌دهی به ورودی کنترل است. دو حالت در نظر گرفته شده است:

۱. تغییر ماتریس R با ماتریس Q ثابت.
۲. تغییر ماتریس Q با ماتریس R ثابت.

۱۸-۱ Q ثابت و R متغیر

در طراحی کنترل‌کننده با استفاده از روش تنظیم کننده خطی درجه دوم (LQR)، ماتریس R نقش مهمی در تعیین استراتژی کنترلی ایفا می‌کند. به طور کلی، تعریف ما از ماتریس R به گونه‌ای است که با افزایش مقادیر آن، سیستم تمایل به استفاده از انرژی کنترلی کمتری پیدا می‌کند. این امر منجر به چندین پیامد قابل توجه در رفتار سیستم می‌شود:

۱. موقعیت قطب‌ها: با افزایش R، قطب‌های سیستم حلقه بسته معمولاً به سمت مبدأ صفحه s حرکت می‌کنند. این به معنای پایداری بیشتر سیستم است، اما ممکن است منجر به پاسخ کندتری شود.
۲. دامنه پاسخ: افزایش R معمولاً باعث می‌شود که دامنه پاسخ سیستم بزرگتر شود. این به این معنی است که سیستم ممکن است به ورودی‌ها با شدت بیشتری واکنش نشان دهد.
۳. سرعت پاسخ: با افزایش R، سیستم تمایل به عملکرد کندتری پیدا می‌کند. این به دلیل تلاش سیستم برای حفظ انرژی کنترلی و اجتناب از تغییرات سریع در سیگنال کنترل است.

۴. مقادیر متغیرهای حالت: اصولاً، با افزایش R، مقادیر مربوط به متغیرهای حالت بزرگتر می‌شوند. این بدان معناست که سیستم اجازه می‌دهد متغیرهای حالت از مقدار مطلوب خود بیشتر منحرف شوند.

برای صحت‌سنجی این ادعاها و بررسی دقیق‌تر اثرات تغییر ماتریس R، از دستور lqr() در MATLAB استفاده می‌کنیم. این دستور امکان حل سریع و دقیق مسئله کنترل بهینه LQR را فراهم می‌کند. خروجی‌های این دستور به شرح زیر هستند:

۱. خروجی اول (K): این ماتریس شامل ضرایب فیدبک بهینه است که برای رسیدن به پاسخ مطلوب استفاده می‌شود. K در واقع ماتریس بهره فیدبک حالت است که سیگنال کنترل را تعیین می‌کند.

۲. خروجی دوم (S): این ماتریس، حل معادله جبری ریکاتی است که در فرآیند بهینه‌سازی LQR استفاده می‌شود.

۳. خروجی سوم (e): این بردار شامل مقادیر ویژه (قطب‌ها) سیستم حلقه بسته است که با استفاده از ماتریس K به دست می‌آید. این قطب‌ها نشان‌دهنده دینامیک سیستم کنترل شده هستند.

با تغییر مقادیر R و مشاهده تغییرات در این خروجی‌ها، می‌توانیم به طور دقیق تأثیر R بر عملکرد سیستم را بررسی کنیم. این تحلیل به ما امکان می‌دهد تا درک عمیق‌تری از ارتباط بین انتخاب ماتریس R و رفتار سیستم کنترل شده به دست آوریم، و در نهایت به طراحی بهتر کنترل‌کننده LQR منجر شود.

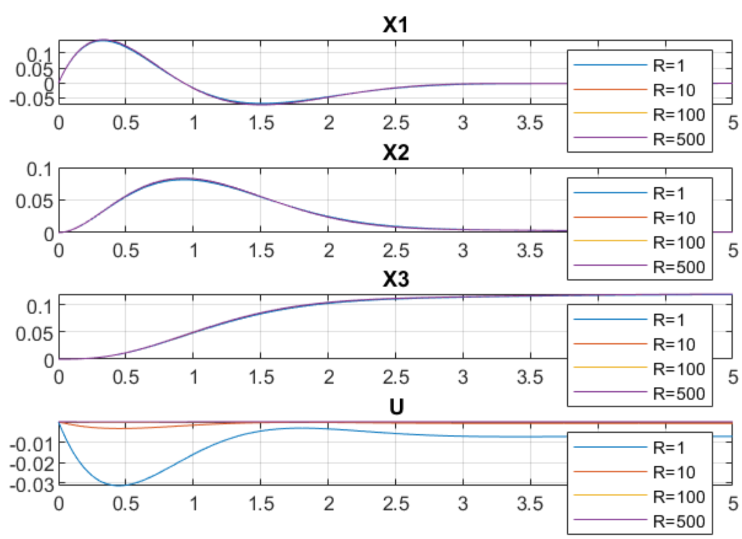
```
% Case 1: Varying R with constant Q
Q = eye(3)
R_values = [1, 10, 100, 500]

figure('Position', [100, 100, 1200, 800]);

for i = 1:length(R_values)
    R = R_values(i);
    [K,~,P] = lqr(A, B, Q, R);
    sys = ss(A-B*K, B, C, 0);
    [y,~,x] = lsim(sys, ones(1, 1000), t);
    u = -K * x';
```

در این بخش، ماتریس Q به عنوان ماتریس همانی ۳ ثابت در نظر گرفته می‌شود، در حالی که مقادیر مختلفی برای R تعریف می‌شوند. هدف اصلی، بررسی تأثیر تغییرات R بر عملکرد سیستم کنترلی است.

افزایش R به معنای افزایش هزینه کنترل است. انتظار می‌رود که با افزایشR ، سیستم محافظه‌کارانه‌تر عمل کند، یعنی تلاش کنترلی کمتری اعمال شود اما زمان رسیدن به حالت پایدار افزایش یابد. این امر در سیستم‌های عملی که محدودیت‌های فیزیکی در اعمال کنترل دارند، بسیار مهم است.

حلقه for برای محاسبه بهره فیدبک (K) برای هر مقدار R استفاده می‌شود. تابع lqr بهره بهینه را محاسبه می‌کند. سپس، سیستم حلقه بسته با استفاده از این بهره شکل می‌گیرد و با تابع lsim شبیه‌سازی می‌شود. ورودی پله واحد ((ones(1, 1000)) برای بررسی پاسخ پله سیستم استفاده می‌شود.

```
Q = 3×3
     1     0     0
     0     1     0
     0     0     1
R_values = 1×4
     1    10   100   500
```



با افزایش R، که به معنای افزایش هزینه کنترل است، مشاهده می‌شود که دامنه پاسخ افزایش می‌یابد. این افزایش دامنه را می‌توان به تمایل کمتر سیستم برای اعمال سیگنال‌های کنترلی قوی نسبت داد. با این حال، به طور متناقض‌نمایی، خطای حالت دائم کاهش می‌یابد که نشان‌دهنده بهبود دقت طولانی‌مدت سیستم است. این رفتار می‌تواند ناشی از رویکرد محافظه‌کارانه‌تر سیستم کنترل باشد که به دنبال پایداری بیشتر در بلندمدت است. تغییرات در سرعت پاسخ متغیرهای حالت نیز قابل توجه است؛ متغیرهای حالت اول و سوم سریع‌تر می‌شوند، در حالی که متغیر حالت دوم کندتر می‌شود. این تفاوت‌ها می‌تواند ناشی از تغییر در اولویت‌بندی کنترل و حساسیت متفاوت متغیرها نسبت به تغییرات R باشد.

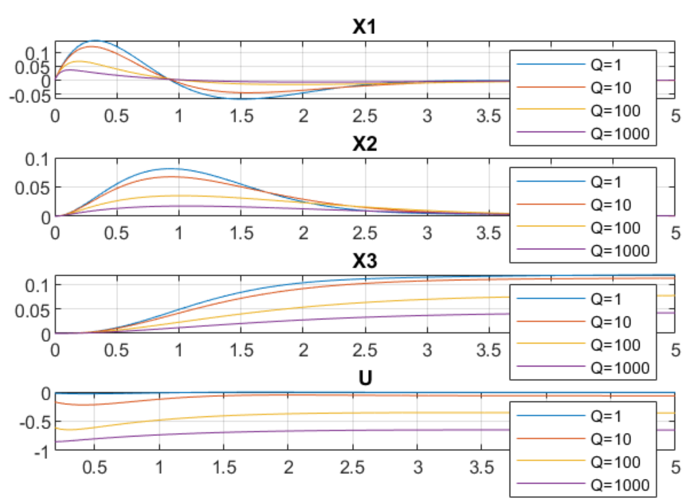
۲-۱۸ Q متغیر R ثابت

```
% Case 2: Varying Q with constant R

R = 1

Q_values = {eye(3), 10*eye(3), 100*eye(3), 500*eye(3)}

figure('Position', [100, 100, 1200, 800]);

for i = 1:length(Q_values)
    Q = Q_values{i};
    [K,~,P] = lqr(A, B, Q, R);
    sys = ss(A-B*K, B, C, 0);
    [y,~,x] = lsim(sys, ones(1, 1000), t);
    u = -K * x';
```

در این بخش، روندی مشابه با حالت ۱-۱۸ دنبال می‌شود، اما این بار R ثابت نگه داشته می‌شود (R = 1) و Q تغییر می کند. هدف اصلی، بررسی تأثیر افزایش اهمیت خطای حالت بر عملکرد سیستم است. افزایش Q به معنای افزایش اهمیت دقت در رسیدن به حالت مطلوب است. انتظار می‌رود که با افزایشQ ، سیستم سریع‌تر و با دقت بیشتری به حالت مطلوب برسد. این امر ممکن است منجر به افزایش تلاش کنترلی شود، که در نمودارu قابل مشاهده خواهد بود. مجدداً، یک حلقه for برای محاسبه K و شبیه‌سازی سیستم برای هر مقدار Q استفاده می‌شود. این بار، Q به صورت ضرایبی از ماتریس همانی تعریف می‌شود تا تأثیر افزایش یکنواخت تمام عناصر Q بررسی شود

```
R = 1
Q_values = 1×4 cell
```

	1	2	3	4
1	[1,0,0;0,1,0;0,0,1]	[10,0,0;0,10,0;0,0,10]	[100,0,0;0,100,0;0,0,100]	[500,0,0;0,500,0;0,0,500]

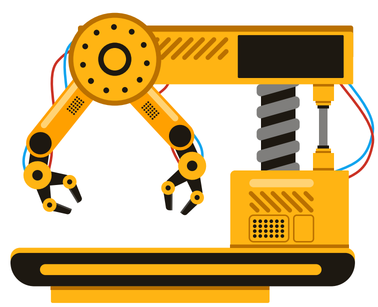


تحلیل نتایج شبیه‌سازی نشان می‌دهد که کاهش مقادیر ماتریس Q منجر به بهبود قابل توجهی در پاسخ سیستم می‌شود. این بهبود را می‌توان در کاهش نوسانات، زمان نشست کوتاه‌تر و خطای حالت ماندگار کمتر مشاهده کرد. همزمان، افزایش مقدار R نیز تأثیر مثبتی بر عملکرد کلی سیستم دارد. با این حال، باید توجه داشت که در پروژه‌های واقعی، انتخاب بهینه پارامترهای Q و R فرآیندی پیچیده و چند بعدی است. این انتخاب باید با در نظر گرفتن الزامات خاص پروژه، محدودیت‌های فیزیکی سیستم (مانند حداکثر نیروی محرکه یا سرعت عملگرها)، و اهداف کنترلی (مانند حداقل مصرف انرژی یا دقت بالا) صورت گیرد. در برخی موارد، ممکن است نیاز به مصالحه بین سرعت پاسخ و پایداری سیستم وجود داشته باشد. بنابراین، گرچه در تئوری افزایش R و کاهش Q می‌تواند به پاسخ مطلوب منجر شود، در عمل باید یک رویکرد متعادل و سازگار با شرایط واقعی اتخاذ کرد. این امر مستلزم آزمایش‌های گسترده، تنظیم دقیق پارامترها و در نظر گرفتن جنبه‌های مختلف عملکرد سیستم است تا بتوان به بهترین ترکیب ممکن از پایداری، سرعت پاسخ و دقت کنترل دست یافت.

در انتهای فایل تمام توابع لازمه برای کد ها آورده شده است، زیرا در Live Script توابع باید در انتهای فایل باشد.

```
function dxdt = d_Q7(t, x, A, B, K)
   u = -K * x;
   dxdt = A * x + B * u;
end
%----------------------------------------------------------------
% Q -8 fcn:
function dxdt = d_Q8(t, x, A, B, C, K, p, r)
   u = -K * x + p * r;
   dxdt = A * x + B * u;
end
%----------------------------------------------------------------
% Q -11 & 12 fcn:
function d_sys = observer_dynamics(~, x, A, B, C, L)
   n = size(A, 1);
   x_true = x(1:n);
   x_hat = x(n+1:end);
   u = 0;
   y = C * x_true;
   dx_true = A * x_true + B * u;
   dx_hat = A * x_hat + B * u + L * (y - C * x_hat);
   d_sys = [dx_true; dx_hat];
end
%----------------------------------------------------------------
% Q -13 fcn:
function dx = system_dynamics(~, x, A, B)
   u = 0;
```

```
    dx = A * x + B * u;
end
%------------------------------------------------------------------
% Q -14 fcn:
function Z = d_Q14(t, x)
    numerator_coeffs = [3 1.677];
    denominator_coeffs = [2 8.341 21.55 16.61];
    [A, B, C, D] = tf2ss(numerator_coeffs, denominator_coeffs);

    P_observer = [-20 -25 -30];
    P_controller = [-3 -4 -5];

    % Calculate observer gain
    K_observer = acker(A', C', P_observer);
    L = K_observer';

    % Calculate state feedback gain
    K = acker(A, B, P_controller);

    r = 1;
    Ga = inv(C * inv(-A + B * K) * B);
    u = Ga * r;

    G = [A -B * K; L * C A - L * C - B * K];
    Z = G * x + [B; B] * u;
end
```



با تشکر و سپاس از توجه شما

سید مهدی موسویون – ۹۹۴۱۳۱۳۶ – تیرماه ۱۴۰۳ – درس کنترل مدرن: پروژه شبیه سازی تابع تبدیل

References:

- MATLAB document