Laboratório 10

SEL0359 - Controle Digital - 2° Semestre de 2024

Prof. Marcos R. Fernandes

Controle LQR

Entrega:[1] entregue um arquivo PDF com os gráficos e código fonte utilizado para cada questão além da resposta para as perguntas dos enunciados. Não esqueça de colocar título na figura para identificar o que cada figura representa, nome dos eixos, e legenda.

O objetivo dessa aula de laboratório é explorar recursos computacionais do Matlab para projeto de controladores LQR em tempo discreto.

Dica: Confira sempre o help do matlab para cada comando antes de utiliza-lo!

- dare: obtem solução da equação de Riccati em tempo discreto;
- trace: calcula traço de uma matriz;
- eye: constrói matriz identidade;
- ctrb: constrói matriz de controlabilidade;
- dqlr: obtém ganho ótimo LQR de realimentação de estados.

1 Atividade 1

Nessa atividade, o objetivo implementar o controle LQR em tempo discreto a partir das equações resultantes da solução do problema de controle ótimo de horizonte finito:

$$\min_{u_0,u_1,\ldots,u_{N-1}} \sum_{k=0}^{N-1} x_k^\intercal Q x_k + u_k^\intercal R u_k + x_N^\intercal P_N x_N$$

tal que

$$x_{k+1} = Ax_k + Bu_k$$
$$x_k \in \mathbb{R}^n$$
$$u_k \in \mathbb{R}^m$$

Sabe-se que a solução ótima é dada por

$$u_k^* = -(R + B^\intercal P_{k+1} B)^{-1} B^\intercal P_{k+1} A x_k$$

[1] Última atualização: 8/11/2024

em que P_{k+1} é obtido através da Eq. de Riccati em tempo discreto:

$$P_k = Q + A^\intercal P_{k+1} A - A^\intercal P_{k+1} B (R + B^\intercal P_{k+1} B)^{-1} B^\intercal P_{k+1} A$$

cuja solução pode ser obtida "de trás para frente" fazendo $k = N-1$, $k = N-2$, ... , $k = 1$, $k = 0$.

Considere o modelo em espaço de estados a seguir

$$x_{k+1} = \begin{bmatrix} 3 & -1 \\ 0 & 2 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 2 \end{bmatrix} u_k$$

Suponha a função custo seja dada por

$$J = \sum_{k=0}^{N-1} x_k^\intercal Q x_k + u_k^\intercal R u_k + x_N^\intercal P_N x_N$$

com

$$Q = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}, \quad R = 1, \quad P_N = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}.$$

O código Matlab a seguir implementa o controle LQR de horizonte finito ($N < \infty$) para esse sistema.

```
close all
clear all
clc
%%
A=[3 -1;0 2];
B=[1 2]';
%%
rank(ctrb(A,B))
%%
N=20; %horizonte
x=zeros(2,N);
u=zeros(1,N);
P=zeros(2,2,N);
trP=zeros(1,N);
%% Resolve eq. de riccati (backward in time)
Q=[2 0;0 3];
R=1;
P(:,:,N)=0.1*eye(2);
trP(N)=trace(P(:,:,N));
for k=N-1:-1:1
    P(:,:,k)=Q+A'*P(:,:,k+1)*A-A'*P(:,:,k+1)*B/(R+B'*P(:,:,k+1)*B)*B'*P(:,:,k+1)*A;
    trP(k)=trace(P(:,:,k));
```

```
end
%% simula sistema com controle LQR de horizonte finito
x(:,1)=[10 10]';
for k=1:N-1
    u(k)=-inv(R+B'*P(:,:,k+1)*B)*B'*P(:,:,k+1)*A*x(:,k); %controle LQR
    x(:,k+1)=A*x(:,k)+B*u(k);
end
%%
figure
stairs(x')
title('estados')
figure
plot(trP,'o-')
title('Traco de P')
```

Note que o traço da matriz P_k converge para um valor constante indicando que o sistema possui uma solução de Riccati estacionária.

Usando a mesma lógica implemente um controle LQR de horizonte finito com $N = 30$ para o sistema:

$$x_{k+1} = \begin{bmatrix} 3 & 0 & 0 \\ 5 & 4 & 0 \\ 1 & 2 & 3 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} u_k$$

Considere

$$Q = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R = 3, \quad P_N = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

2 Atividade 2

Repita a atividade anterior, porém, agora considere um horizonte infinito ($N \to \infty$) para a formulação do controle ótimo. Ou seja, considere o problema de controle ótimo na forma:

$$\min_{u_0, u_1, \ldots} \lim_{N \to \infty} \sum_{k=0}^{N-1} x_k^\intercal Q x_k + u_k^\intercal R u_k + x_N^\intercal P_N x_N$$

tal que

$$x_{k+1} = Ax_k + Bu_k$$

$$x_k \in \mathbb{R}^n$$

$$u_k \in \mathbb{R}^m$$

A solução ótima é dada por

$$u_k^* = -(R + B^\intercal P_\infty B)^{-1} B^\intercal P_\infty A x_k$$

em que P_∞ é a solução estacionária da Eq. de Riccati em tempo discreto:

$$P_\infty = Q + A^\intercal P_\infty A - A^\intercal P_\infty B (R + B^\intercal P_\infty B)^{-1} B^\intercal P_\infty A$$

cuja solução pode ser obtida "de trás para frente" fazendo $k = N-1$, $k = N-2$, ... , $k = 1$, $k = 0$ com N suficientemente grande (suficiente para que a eq. Riccati entre em regime estacionário) ou também pode-se utilizar o comando **dare** do Matlab. Por simplicidade, considere $P_N = 0$.

O código do Matlab a seguir compara a solução de Riccati obtida via recursão "de trás para frente" com a solução via comando **dare**.

```
close all
clear all
clc
%%
A=[3 -1;0 2];
B=[1 2]';
%%
rank(ctrb(A,B))
%%
N=20; %horizonte
x=zeros(2,N);
u=zeros(1,N);
P=zeros(2,2,N);
trP=zeros(1,N);
%% Resolve eq. de riccati (backward in time)
Q=[2 0;0 3];
R=1;
P(:,:,N)=zeros(2);
trP(N)=trace(P(:,:,N));
for k=N-1:-1:1
    P(:,:,k)=Q+A'*P(:,:,k+1)*A-A'*P(:,:,k+1)*B/(R+B'*P(:,:,k+1)*B)*B'*P(:,:,k+1)*A;
    trP(k)=trace(P(:,:,k));
end
Pinf=dare(A,B,Q,R);
trPinf=trace(Pinf);
%%
figure
plot(trP,'o-')
hold on
```

```
line([0 N],[trPinf trPinf],'linestyle','--','color','black','linewidth',1.5)
title('Traco de P')
legend('trP','trP_\infty')
```

3 Atividade 3

Nessa atividade, o objetivo é projetar um controlador ótimo de realimentação de estados.

Nesse exemplo, será utilizado o modelo do *self-balanced robot* conforme ilustrado na Figura 1. As EDO que descrevem o comportamento dessa planta são dadas a seguir:

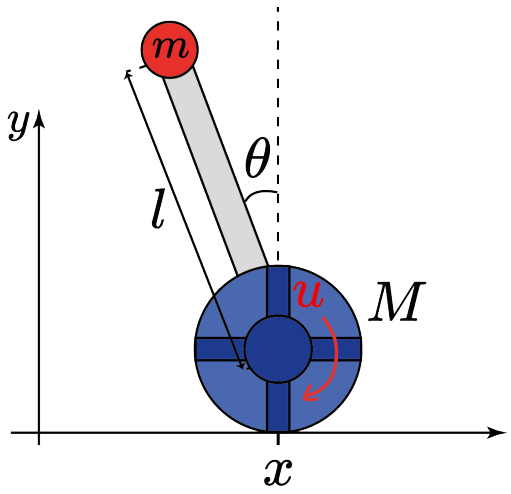

Figura 1: Self-balanced robot.

$$(M+m)\ddot{x} - ml\cos(\theta)\ddot{\theta} + ml\dot{\theta}^2\sin(\theta) + \alpha_1\dot{x} = u, \tag{1}$$

$$l^2 m\ddot{\theta} - ml\cos(\theta)\ddot{x} - gml\sin(\theta) + \alpha_2\dot{\theta} = 0. \tag{2}$$

o vetor de estados é dado por

$$x = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}, \quad \begin{cases} x \rightarrow \text{posição} \\ \dot{x} \rightarrow \text{velocidade} \\ \theta \rightarrow \text{posição angular} \\ \dot{\theta} \rightarrow \text{velocidade angular} \end{cases}$$

e o sinal de entrada é o torque que atua na roda.

Adotando,

$$x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad u_0 = 0,$$

como ponto de operação do sistema (posição vertical com velocidade zero), pode-se obter um modelo linearizado na forma

$$\dot{x} = A(x_0, u_0)x + B(x_0, u_0)u,$$

válido para regiões próximas do ponto (x_0, u_0). Após obter o modelo linearizado, pode-se utilizar o comando **c2d** para obter o equivalente em tempo discreto.

Considere a condição inicial do sistema real como $x_0 = [2\ 0\ \frac{10\pi}{180}\ 0]^\intercal$.

1. Projete um controle de realimentação de estados do tipo LQR com horizonte infinito de forma a trazer o sistema para a origem ($x_k = [0\ 0\ 0\ 0]^\intercal$) em menos de 5s. Assuma que os estados são acessíveis diretamente (não tem necessidade de um observador). (defina as matrizes Q e R de forma ter um desempenho que você ache razoável). Dica: use o comando **dlqr** do matlab para obter o ganho ótimo de realimentação de estados.

2. Repita o item anterior com valores crescente de Q e R e para cada caso apresente os pólos de malha fechada do sistema. O que ocorre com os pólos na medida que as matrizes Q e R aumentam?

3. Faça uma modificação na simulação da planta de forma a ser possível controlar a planta para posições diferentes de zero usando LQR.

   dica: uma vez que o Jacobiano do sistema não depende da posição, pode-se usar o modelo linearizado na forma de **desvio de estado e controle**

   $$\delta x_{k+1} = A_d \delta x_k + B_d \delta u_k$$

   em que $\delta x_k = x_k - x_0$, $\delta u_k = u_k - u_0$ e x_0 é o ponto de operação e u_0 é a ação de controle de equilíbrio. Dessa forma, a lei de controle é dada por $u_k = -K\delta x_k + u_0$. Assim, mudando-se o ponto de operação a lei de controle vai "regular" o sistema para próximo de x_0 sem a necessidade de recalcular o ganho de realimentação de estados! Note que $u_0 = 0$ nesse exemplo do self-balanced robot.

   Para avaliar o desempenho faça o self-balanced-robot se mover da posição $x = 0$ até a posição $x = 4$ e permancer por 5s nessa posição. Em seguida, faça ele se mover

para a posição $x = -4$;

Dica: Use o código de exemplo disponível no e-disciplina (arquivo **self-balanced-robot.zip**) como ponto de partida.

4 Atividade 4

Nessa atividade, o objetivo é projetar um controlador LQR com rastreamento para modelos em espaço de estados em tempo discreto.

Nesse exemplo, será utilizado o modelo simplificado de um *quadricóptero* restrito a movimentos em um plano conforme ilustrado na Figura 2. As EDO que descrevem o compor-

Figura 2: Quadricóptero.

tamento dessa planta são dadas a seguir:

$$\ddot{z} = \frac{1}{M}(F_1 + F_2)\cos(\theta) - g \tag{3}$$

$$\ddot{y} = \frac{1}{M}(F_1 + F_2)\sin(\theta) \tag{4}$$

$$\ddot{\theta} = \frac{l}{I_{xx}}(F_1 - F_2) \tag{5}$$

o vetor de estados é dado por

$$x = \begin{bmatrix} z \\ y \\ \dot{z} \\ \dot{y} \\ \theta \\ \dot{\theta} \end{bmatrix}, \quad \begin{cases} z \rightarrow \text{posição na direção z} \\ x \rightarrow \text{posição na direção y} \\ \dot{z} \rightarrow \text{velocidade na direção z} \\ \dot{y} \rightarrow \text{velocidade na direção y} \\ \theta \rightarrow \text{posição angular (roll)} \\ \dot{\theta} \rightarrow \text{velocidade angular} \end{cases}$$

e o sinal de entrada são as forças de empuxo F_1 e F_2 geradas pelos motores:

$$u = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

Adotando,

$$x_0 = \begin{bmatrix} z_{\text{ref}} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad u_0 = \begin{bmatrix} \frac{1}{2}gM \\ \frac{1}{2}gM \end{bmatrix},$$

como ponto de operação do sistema (altura z_{ref} com velocidade zero e roll zero), pode-se obter um modelo linearizado na forma

$$\delta\dot{x} = A(x_0, u_0)\delta x + B(x_0, u_0)\delta u,$$

válido para regiões próximas do ponto (x_0, u_0). Após obter o modelo linearizado, pode-se utilizar o comando **c2d** para obter o equivalente em tempo discreto. Considere a condição inicial do sistema real como $x_0 = [0\ 0\ 0\ 0\ 0\ 0]^\intercal$.

Assuma que os estados do sistema estão disponíveis e então projete um controlador de realimentação de estados do tipo LQR que estabilize o quadricóptero na posição $z = 5$ e $y = 0$ durante 5s e depois leve o quadricóptero para $z = 3$. Use o código exemplo disponível no site e-disciplina **quadcopter-control.zip** como ponto de partida.

Em seguida, crie uma trajetória no plano zy e faça o quadricóptero percorrer essa trajetória usando controle LQR (use sua imaginação para criar diferentes trajetórias).