UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
BACHARELADO EM ENGENHARIA ELÉTRICA

Rafael Soares do Egito

ELABORAÇÃO DE MATERIAL DIDÁTICO BÁSICO PARA CONTROLE E AUTOMAÇÃO E MODELOS EM MATLAB

Campina Grande, Paraíba

2014

Rafael Soares do Egito

ELABORAÇÃO DE MATERIAL DIDÁTICO BÁSICO PARA CONTROLE E AUTOMAÇÃO E MODELOS EM MATLAB

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Bacharelado em Engenharia Elétrica da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador:

Prof. Dr. George Acioli Junior

Campina Grande, Paraíba

2014

Folha de Aprovação

Trabalho de Conclusão de Curso sob o título *"ELABORAÇÃO DE MATERIAL DIDÁTICO BÁSICO PARA CONTROLE E AUTOMAÇÃO E MODELOS EM MATLAB"*, defendida por Rafael Soares do Egito e aprovado em Dezembro de 2014, em Campina Grande, Paraíba, pela banca examinadora constituída pelos professores:

Prof. Dr. George Acioli Junior
Orientador

Professor convidado
Universidade Federal de Campina Grande

Agradecimentos

A Deus, pela Sua soberania que me permitiu chegar até aqui. Que Ele seja glorificado acima de tudo.

À minha mãe, por tantos anos de trabalho árduo para me dar a oportunidade de realizar este sonho, me apoiando em todos os momentos e nunca me deixando desistir.

Ao meu irmão, pelos seus vários puxões de orelha que me fizeram ser um profissional excelente como ele.

Ao meu pai, que sempre se orgulhou de mim e me trouxe motivação para seguir nesta caminhada.

À minha namorada Rayanne, que me apoiou e me suportou nas minhas dificuldades e momentos de desânimo.

Aos meus amigos que, sem dúvida, foram imprescindíveis não apenas nas horas de estudos, mas também nas de lazer.

“Todas as coisas foram feitas por intermédio dele, e,
sem ele, nada do que foi feito se fez”
João 1:3

Resumo

Este trabalho tem como objetivo a elaboração de material didático sobre Controle baseado no livro *Linear Feedback Control Analysis and Design with Matlab* dos autores Dingyü Xue (Northeastern University), YangQuan Chen (Utah State University) e Derek P. Atherton (University of Sussex). Todos os códigos e funções MATLAB e modelos em Simulink aqui apresentados foram compilados e estão armazenados em arquivos para facilitar o entendimento do material e seus exemplos.

Palavras-chave: controle e automação, MATLAB, Simulink, PID

Abstract

This project's objective is the development of teaching material on Control based on the book Linear Feedback Control Analysis and Design with Matlab, from the authors Dingyu Xue (Northeastern University), Yangquan Chen (Utah State University) and Derek P. Atherton (University of Sussex). All code and MATLAB functions and Simulink models presented here were compiled and are stored in files to facilitate understanding of the material and its examples.

Keywords: control and automation, MATLAB, Simulink, PID

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

O MATLAB é uma ferramenta poderosa de cálculo numérico que tem sido, ao longo dos anos, bastante útil em diversas área do conhecimento, desde matemática e física até as engenharias. No estudo de Controle não tem sido diferente. O MATLAB fornece diversas ferramentas específicas para as diversas áreas de Controle e Automação que permitem realizar cálculos complexos de maneira praticamente instantânea, além de gerar gráficos e diagramas extremamente úteis, como o Diagrama de Bode ou o Diagrama de Nyquist. É capaz também de transformar o modo como um determinado modelo é apresentado (e.g., de espaço de estados para função de transferência).

São estas inúmeras facilidades que motivaram a elaboração deste material didático, de modo a facilitar o aprendizado de algumas ferramentas básicas de Controle do MATLAB. O material apresenta de forma gradativa a própria Teoria de Controle, começando de conceitos básicos de Controle até chegar ao mínimo necessário para o projeto de controladores PID, sendo que, em cada passo, é apresentada a ferramenta MATLAB correspondente. No entanto, não é objetivo deste material didático o ensino da Teoria de Controle, mas que ela possa ser complementada com o conhecimento das poderosas ferramentas fornecidas no MATLAB.

Inicialmente é apresentado o problema de modelagem física, seguida do conceito de transformada de Laplace com soluções em MATLAB. Em seguida, são apresentadas as representações em função de transferência, em espaço de estados e no modo zero-polo-ganho, com suas respectivas descrições MATLAB.

No capítulo seguinte é apresentada a análise no domínio do tempo de sistemas lineares e soluções numéricas, assim como maneiras de esboçar as respostas ao degrau. Após isso é mostrado o modo de ilustrar o lugar as raízes. É realizada a análise no domínio da frequência de um sistema linear e suas análises gráficas como os diagramas de Bode, Nyquist e Nichols.

Após isso, é apresentada a ferramenta Simulink que é extremamente útil em Controle, em especial na modelagem de sistemas não-lineares. São mostradas algumas particularidades deste tipo de modelo e o modo como elas podem ser incluídas no modelo Simulink.

E, para finalizar, são apresentadas várias técnicas de projeto de controladores. São explanadas as ações proporcional, integral e derivativa. É mostrado também o método de sintonia de Ziegler-Nichols. Também é apresentada a questão da identificação de modelo equivalente de primeira ordem com tempo morto e um algoritmo de otimização de controladores e uma interface gráfica que facilita esta tarefa.

1.1 Objetivos

Este trabalho tem como objetivo a elaboração de material didático que apresenta as ferramentas fornecidas no MATLAB que facilitam as atividades relacionadas ao Controle e Automação. A Teoria de Controle é apresentada desde os conceitos básicos, sendo pontuada pelas respectivas ferramentas fornecidas em MATLAB.

1.2 Metodologia

O material foi produzido a partir da tradução dos principais capítulos do livro *Linear Feedback Control Analysis and Design with Matlab* dos autores Dingyü Xue (Northeastern University), YangQuan Chen (Utah State University) e Derek P. Atherton (University of Sussex). Foi realizada uma sumarização dos exemplos do livro original, de modo a destacar os mais importantes. Todos os códigos e funções MATLAB (sejam os apresentados em exemplos ou os apresentados ao longo do texto) foram compilados e estão armazenados em arquivos. Os arquivos gerados a partir de exemplos foram salvos com o nome no padrão chXXexYY.m (ou chXXexYY.mdl para arquivos de Simulink), onde XX é o número do capítulo onde se encontra o exemplo e YY é o número do exemplo em si. Os arquivos de funções foram salvos com o nome da respectiva função. Alguns arquivos também foram salvos em duplicidade com o nome sugerido pelo livro original.

2 *Modelos Matemáticos de Sistemas de Controle com Realimentação*

A maioria, mas não todos, dos procedimentos de projeto para um sistema de controle fazem uso de modelos matemáticos. Portanto, é importante tentar obter modelos matemáticos precisos do sistema. O sistema pode então ser analisado e projetado de modo sistemáticos utilizando modelos matemáticos como aproximações de seu comportamento real.

Se o modelo não é conhecido, dois métodos podem ser utilizados para construir o modelo do sistema para questões de análise e projeto. O primeiro método é derivar o modelo utilizando leis e princípios físicos. O segundo método, mais frequentemente utilizado, é encontrar um modelo matemático aproximado baseado na resposta observada do sistema. O primeiro é chamado de modelagem física e o segundo de identificação do sistema.

Neste capítulo, o problema de modelagem física é ilustrado através de um exemplo na seção 2.1. Na seção 2.2, o conceito de transformada de Laplace é apresentado com soluções em MATLAB. A representação da função de transferência de sistemas lineares é descrita na seção 2.3. Serão apresentadas também diversas descrições em MATLAB de funções de transferência padrão. Outras descrições de sistemas comumente usadas, tal como a representação em espaço de estados e a representação zero-polo-ganho são apresentadas na seção 2.4. Os princípios de modelagem para encontrar modelos globais de sistemas a partir da conexão de submodelos são mostrados nas seções 2.5 e 2.6. Na seção 2.6 também é descrita a conversão equivalente entre diferentes tipos de modelos para o mesmo sistema. Por exemplo, uma representação em função de transferência pode ser convertido para uma representação em espaço de estados.

2.1 Exemplo de Modelagem Física

Considere o circuito mostrado na figura, onde um resistor R, um indutor L e um capacitor C são conectados em série. Para esse sistema dinâmico, o sinal de entrada é $u(t)$ e o sinal de saída é $u_c(t)$.

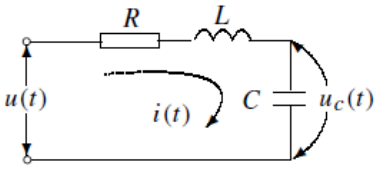


Figura 2.1: Circuito RLC série

A corrente $i(t)$ é dada por

$$i(t) = C\frac{\mathrm{d}u_c(t)}{\mathrm{d}t} \tag{2.1}$$

e a equação da tensão pode ser escrita como

$$u(t) = Ri(t) + L\frac{\mathrm{d}i(t)}{\mathrm{d}t} + u_c(t) \tag{2.2}$$

Substituindo 2.1 em 2.2, tem-se que

$$LC\frac{\mathrm{d}^2u_c(t)}{\mathrm{d}t^2} + RC\frac{\mathrm{d}u_c(t)}{\mathrm{d}t} + u_c(t) = u(t) \tag{2.3}$$

A equação diferencial ordinária (EDO) de segunda ordem mostrada em 2.3 é chamada de modelo matemático do circuito elétrico.

Em geral, o modelo matemático de um sistemas dinâmico em tempo contínuo pode ser representado por uma EDO.

2.2 A Transformada de Laplace

Na equação 2.3, a tensão $u_c(t)$ sobre o capacitor C pode ser representada por uma EDO linear de segunda ordem. Um método usado para resolver equações diferenciais lineares é utilizando a transformada de Laplace, como visto abaixo.

Definição 2.1. A transformada de Laplace de uma função no tempo f(t) é definida por

$$\mathscr{L}[f(t)] = \int_0^\infty f(t)e^{-st}\mathrm{d}t = F(s), \tag{2.4}$$

onde $\mathscr{L}[f(t)]$ é a notação abreviada para a transformada de Laplace.

O resultado da transformada de Laplace é uma função de s, uma variável complexa, geralmente denotada por $F(s)$. A unidade de s é $segundo^{-}1$.

Para uma dada função $f(t)$, é possível encontrar sua transformada de Laplace tanto através de tabela de transformação como através de funções do MATLAB.

Teorema 2.1. Algumas propriedades importantes da transformada de Laplace estão listadas abaixo (sem as demonstrações).

1. *Linearidade*: se a e b são escalares, então
$$\mathscr{L}[af(t) \pm bg(t)] = a\mathscr{L}[f(t)] \pm b\mathscr{L}[g(t)].$$
2. *Deslocamento no tempo*: $\mathscr{L}[f(t-a)] = e^{-as}F(s)$.
3. *Deslocamento em frequência (s)*: $\mathscr{L}[e^{-at}f(t)] = F(s+a)$.
4. *Diferenciação*: $\mathscr{L}[\mathrm{d}f(t)/\mathrm{d}t)] = sF(s) - f(0^+)$. A derivada de n-ésima ordem é dada por
$$\mathscr{L}\left[\frac{\mathrm{d}^n}{\mathrm{d}t^n}f(t)\right] = s^nF(s) - s^{n-1}f(0^+) - s^{n-2}\frac{\mathrm{d}f(0^+)}{\mathrm{d}} - \cdots - \frac{\mathrm{d}^{n-1}f(0^+)}{\mathrm{d}t^{n-1}}.$$
5. *Integração*: assumindo condição inicial nula, $\mathscr{L}[\int_0^t f(\tau)\mathrm{d}\tau] = F(s)/s$. A integral de n-ésima ordem da função f(t) é
$$\mathscr{L}\left[\int_0^t \cdots \int_0^t f(\tau)(\mathrm{d}\tau)^n\right] = \frac{F(s)}{s^n}.$$
6. *Tempo inicial e tempo final*:
$$\lim_{t\to 0} f(t) = \lim_{s\to\infty} sF(s), \lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s).$$
7. *Convolução*: $\mathscr{L}[f(t) * g(t)] = \mathscr{L}[f(t)]\mathscr{L}[g(t)]$, onde o operador de convolução é definido como
$$f(t) * g(t) = \int_0^t f(\tau)g(t-\tau)\mathrm{d}\tau = \int_0^t f(t-\tau)g(\tau)\mathrm{d}\tau.$$
8. *Outras*:
$$\mathscr{L}[t^n f(t)] = (-1)^n\frac{\mathrm{d}^nF(s)}{\mathrm{d}s^n}, \qquad \mathscr{L}\left[\frac{f(t)}{t^n}\right] = \int_s^\infty \cdots \int_s^\infty F(s)\mathrm{d}s^n.$$

A função do MATLAB `laplace()` pode ser utilizada para obter a transformada de Laplace de uma função $f(t)$. A sintaxe da função é `F=laplace(f)`. Note que apenas uma limitada classe de sinais $f(t)$ pode ser usada em `laplace(f)`.

Exemplo 2.1. É possível utilizar os seguintes comandos no MATLAB para encontrar a transformada de Laplace da função $e^{bt}\cos(at+b)$:

```
» syms s t a b c; F=laplace(exp(b*t)*cos(a*t+c)).
```

A forma de Laplace da função é

$$F(s) = \frac{\cos(c)(s-b)}{(s-b)^2+a^2} - \frac{\sin(c)a}{(s-b)^2+a^2}$$

Definição 2.2. A transformada inversa de Laplace de uma dada função F(s) é definida por

$$f(t) = \mathscr{L}^{-1}[F(s)] = \frac{1}{2\pi \mathrm{j}} \int_{\sigma-\mathrm{j}\infty}^{\sigma+\mathrm{j}\infty} F(s)e^{st}\mathrm{d}s,$$

onde σ é maior que a parte real das singularidades de F(s).

Dada uma função F(s), sua transformada inversa de Laplace pode ser obtida usando uma tabela ou alguma outra ferramenta. Com uso da Symbolic Toolbox do MATLAB, é possível obter a transformada inversa de Laplace com o comando `f=ilaplace(F)`. Note novamente que apenas uma classe limitada de funções F(s) pode ser usada com `ilaplace(F)`.

2.3 Modelos de Função de Transferência

2.3.1 Funções de Transferência de Sistemas de Controle

Aplicando-se a propriedade de diferenciação da transformada de Laplacade, a equação 2.3 é transformada para sua forma algébrica, como se segue:

$$LCU_c(s)s^2 + RCU_c(s)s + U_c(s) = U(s),$$

onde $U_c(s) = \mathscr{L}[u_c(t)$, $U(s) = \mathscr{L}[u(t)]$, se for assumido que as condições iniciais para $u_c(t)$ e suas derivadas são nulas.

Dividindo-se ambos os lados da equação por $U_c(s)$ obtém-se:

$$\frac{U_c(s)}{U(s)} = \frac{1}{LCs^2+RCs+1}$$

de modo que $\frac{U_c(s)}{U(s)}$ é chamado de função de transferência para um sinal de entrada $u(t)$ e um sinal de saída $u_c(t)$.

A função de transferência de um sistema linear contínuo pode ser generalizado por uma função racional de s na forma

$$G(s) = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_m s + b_{m+1}}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1} s + a_n} \tag{2.5}$$

Se os coeficientes $b_i, (i = 1, \ldots, m+1)$ e $a_i, (i = 1, \ldots, n)$ são constantes, o sistema é chamado de linear invariante no tempo (LIT). O denominador é chamado de polinômio característico do sistema. O valor de n é chamado de ordem do sistema. Para um sistema fisicamente realizável, é frequente que $m \leq n$. Nesse caso, o sistema é chamado de próprio. Se $m < n$, o sistema é chamado de estritamente próprio. O valor de $n - m$ é, algumas vezes, chamado de ordem relativa do sistema.

2.3.2 Representações em MATLAB de Funçõees de Transferência

Uma função de transferência pode ser facilmente colocada no MATLAB utilizando os seguintes comandos:

» `num=[`b_1`, `b_2`, ..., `b_m`, `b_{m+1}`]; den=[1, `a_1`, `a_2`, ..., `a_{n-1}`, `a_n`];`
`G=tf(num, den)`

Ou seja, é necessário representar os coeficientes do numerador e do denominador em dois vetores diferentes `num` e `den`. A variável G recebe o retorno da função `tf()` com a função de transferência.

Exemplo 2.2. A função de transferência

$$G(s) = \frac{s+5}{s^4 + 2s^3 + 3s^2 + 4s + 5}$$

pode ser representada em MATLAB como

```
» num=[1,5]; den=[1,2,3,4,5]; G=tf(num, den)
```

Exemplo 2.3. A função de transferência

$$G(s) = \frac{6(s+5)}{(s^2 + 3s + 1)^2 (s+6)(s^3 + 6s^2 + 5s + 3)}$$

pode ser representada em MATLAB como
`den=conv(conv(conv([1,3,1],[1,3,1]),[1,6]),[1,6,5,3]); num=6*[1,5]; G=tf(num, den)` obtendo-se a função de transferência

$$G(s) = \frac{6s + 30}{s^8 + 18s^7 + 124s^6 + 417s^5 + 740s^4 + 729s^3 + 437s^2 + 141s + 18}$$

A função `conv()` que foi utilizada realiza a convolução entre dois vetores. Os coeficientes do produto entre dois polinômios pode ser obtido pela convolução dos vetores de seus coeficientes, por isso a função `conv()` foi utilizada. Essa função pode ser aninhada quantas vezes forem necessárias.

Outra maneira de se obter a função de transferência no MATLAB é escrevendo-a em sua forma fatorada, da seguinte maneira:

```
» s=tf('s');
  G=6*(s+5)/(s^2+3*s+1)^2/(s+6)/(s^3+6*s^2+5*s+3)
```

Além das variáveis essencias do numerador e denominador, outros campos também são definidos no objeto função de transferência. É possível listar todos os possíveis campos usando o comando `set(tf)`. Outros campos úteis no objeto função de transferência incluem, por exemplo, `ioDelay` e `Ts`, que correspondem, respectivamente, ao atraso de entrada e saída e o intervalo de amostragem; este último é aplicável somente a sistemas discretos no tempo. O campo `Variable` é definido como o símbolo operador usando na função de transferência, com s e p para sistemas em tempo contínuo e z, z^{-1} e q para sistemas em tempo discreto, onde q é a forma abreviada de z^{-1}.

Caso seja necessário mudar o símbolo operador na representação da função de transferência para p e atribuir um atraso de transporte de 0,5 segundos, um dos comandos MATLAB a seguir podem ser utilizados:

```
» G.Variable='p'; G.ioDelay=0.5;
  B=[4, 6; 2, 4; 2, 2; 0, 2]; C=[0, 0, 0, 1; 0, 2, 0, 2];
  set(G,'Variable','p','ioDelay',0.5);
```

O modelo G é então mostrado como

$$e^{-0.5p}\frac{6p+30}{p^8+18p^7+124p^6+417p^5+740p^4+729p^3+437p^2+141p+18}$$

2.3.3 Matrizes de Função de Transferência para Sistemas de Multivariáveis

Sistemas com um entrada e uma saída são chamados de entrada única-saída única (SISO, do inglês single input-single output), enquanto sistemas com mais de uma entrada e mais de uma saída são chamados de múltiplas entradas-múltiplas saídas (MIMO). Para um sistemas MIMO, a

representação em função de transferência é denotada por uma matriz de funções de transferência que é chamada de matriz de função de transferência. O objeto função de transferência `tf` também pode ser utilizado para representar matrizes de função de transferência MIMO.

Exemplo 2.4. Assuma que a matriz de função de transferência de um sistema MIMO é dada por

$$G(s) = \begin{bmatrix} \frac{0.1134e^{-0.72s}}{1.78s^2+4.48s.1} & \frac{0.924}{2.07s+1} \\ \frac{0.3378e^{-0.3s}}{0.361s^2+1.09s+1} & \frac{-0.318e^{-1.29s}}{2.93s+1} \end{bmatrix}$$

Esse modelo pode ser colocado no MATLAB utilizando os seguintes comandos:

```
» g11=tf(0.1134,[1.78 4.48 1],'ioDelay',0.72);
  g12=tf(0.924,[2.07 1]);
  g21=tf(0.3378,[0.361 1.09 1],'ioDelay',0.3);
  g22=tf(-0.318,[2.93 1],'ioDelay',1.29);
  G=[g11,g12; g21,g22];
```

Os numeradores e denominadores do sistema podem ser recuperados com a função

```
» [num, den]=tfdata(G,'v')
```

2.3.4 Funções de Transferência de Sistemas em Tempo Discreto

A função de transferência em tempo discreto

$$H(z) = \frac{b_0 z^m + b_1 z^{m-1} + \cdots + b_{m-1} z + b_m}{a_1 z^n + a_2 z^{n-1} + \cdots + a_n z + a_{n+1}} z^{-d}$$

que pode ser obtida através da transformada-Z para equações de diferença, pode ser também inserido no MATLAB com as seguintes declarações

```
» num=[b0, b1, ..., bm-1, bm];
  den=[a1, a2, ..., an, an+a];
  H=tf(num, den, 'Ts', T, 'ioDelay', d);
```

onde T é o período de amostragem e m é o atraso de transporte. Alternativamente, a variável z pode ser declarada como `z=tf('z',T)` antes de especificar a função de transferência matematicamente.

Exemplo 2.5. Assuma que um modelo em tempo discreto de um sistema seja dado por

$$H(z) = \frac{6z^2 - 0.6z - 0.12}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125} z^{-5},$$

onde T=0.1s. A seguinte declaração pode ser usada:

```
» num=[6 -0.6 -0.12];
  den=[1 -1 0.25 0.25 -0.125];
  H=tf(num,den,'Ts',0.1,'ioDelay',5)
```

Alternativamente, o sistema pode ser especificamente por

```
» z=tf('z',0.1);
  H=(6*z^2-0.6*z-0.12)/(z^4-z^3+0.25*z^2+0.25*z-0.125;
  H.ioDelay=5
```

2.4 Outras Representações de Modelos Matemáticos

2.4.1 Modelagem em Espaço de Estados

Representações em espaço de estados de modelos de sistemas de controle são largamente utilizados em teoria de controle desde os anos 1960, quando a chamada "teoria de controle moderno"foi introduzida. O espaço de estados é outra maneira de descrever um modelo dinâmico de um sistema e pode ser utilizado para representar não apenas sistemas lineares, mas também não-lineares. A representação em espaço de estados de um sistema é sempre referida como um modelo de descrição interna, uma vez que as variáveis internas, assim como os estados, estão completamente descritos neste tipo de representação. De forma contrária, a representação em função de transferência é frequentemente chamada de modelo externo ou modelo de entrada e saída, uma vez que somente a relação entrada-saída do sistema é descrita.

Considere novamente o circuito RLC dado em (2.3). Assumindo $x_1 = u_C$ e $x_2 = \mathrm{d}u_C/\mathrm{d}t$, a equação diferencial ordinária de segunda ordem pode ser reescrita na seguinte forma:

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = x_2 \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = -\frac{1}{LC}x_1 - \frac{R}{L}x_2 + \frac{1}{LC}u \end{cases}$$

Em teoria de controle, $\mathrm{d}x_i/\mathrm{d}t$ é frequentemente denotado como $\dot{x}_i$ e a equação acima pode

ser escrita na forma de matriz como

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1/(LC) & -R/L \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/(LC) \end{bmatrix} u, \tag{2.6}$$

onde x_1 e x_2 são chamados de variáveis de estado, u é chamado de sinal de entrada e (2.6) é chamada de equação de estado do sistema. Note que a escolha das variáveis de estado não é única. Portanto, a equação de estado também não é única. Se as variáveis de estado escolhidas forem a tensão u_C e a corrente i, denotadas por x_1 e x_2, respectivamente, a equação de estado pode ser escrita como

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1/C \\ -1/L & -R/L \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/L \end{bmatrix} u.$$

Suponha que existem p entradas $u_i(t), (i = 1, \ldots, p)$, q saídas $y_i(t), (i = 1, \ldots, q)$ e n estados formando o vetor de estados $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathbf{T}}$. A equação de espaço de estados de um sistema dinâmico geral é dada por

$$\begin{cases} \dot{x_i} = f_i(x_1, x_2, \ldots, x_n, u_1, \ldots, u_p), & i = 1, \ldots, n, \\ y_i = g_i(x_1, x_2, \ldots, x_n, u_1, \ldots, u_p), & i = 1, \ldots, q, \end{cases}$$

onde f_i e g_i podem ser funções não-lineares. Para sistemas LIT, a equação de espaço de estado pode ser simplificada por

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t), \end{cases}$$

onde $\mathbf{u} = [u_1, \ldots, u_p]^{\mathbf{T}}$ e $\mathbf{y} = \left[y_1, \ldots, y_q\right]^{\mathbf{T}}$ são vetores de entrada e saída, respectivamente. As matrizes **A**, **B**, **C** e **D** são compatíveis. O termo "compatíveis" indica que as matrizes possuem as dimensões corretas, ou seja, **A** é uma matriz $n \times n$, **B** é uma matriz $n \times p$, **C** é uma matriz $q \times n$ e **D** é uma matriz $q \times p$.

A representação de uma equação de estado em MATLAB é bastante simples. Basta definir as matrizes de coeficientes **A**, **B**, **C** e **D** e, em seguida, utilizar o comando `G=ss(A,B,C,D)`.

Exemplo 2.6. Um sistema com duas entradas e duas saídas dado pela seguinte forma em espaço

de estados

$$\dot{\mathbf{x}} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix} \mathbf{u},\ \mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix} \mathbf{x}$$

pode ser descrito em MATLAB utilizando os seguintes comandos:

```
» A=[2.25 -5 -1.25 -0.5; 2.25 -4.25 -1.25 -0.25;
      0.25 -0.5 -1.25 -1; 1.25 -1.75 -0.25 -0.75];
  B=[4 6; 2 4; 2 2; 0 2];
  C=[0 0 0 1; 0 2 0 2];
  D=zeros(2,2);
  G=ss(A,B,C,D)
```

Caso as matrizes não sejam compatíveis, a mensagem de erro será mostrada automaticamente pela função `ss()`.

Para modelos em espaço de estados em tempo discreto

$$\begin{cases} \mathbf{x}[(k+1)T] = \mathbf{F}\mathbf{x}(kT) + \mathbf{G}\mathbf{u}(kT), \\ \mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t), \end{cases}$$

com período de amostragem T, o comando `G=ss(A,B,C,D,'Ts',T)` pode ser utilizado.

2.4.2 Descrição Zero-Polo-Ganho

A representação zero-polo-ganho é outra maneira de descrever a função de transferência de um sistema LIT SISO. O modelo zero-polo-ganho de uma dada função de transferência é usualmente representada como

$$G(s) = K\frac{(s+z_1)(s+z_2)\dots(s+z_m)}{(s+p_1)(s+p_2)\dots(s+p_n)}, \tag{2.7}$$

onde K é chamado de ganho do sistema. Note que K não é o ganho DC (ou ganho de frequência zero $G(0)$). Em (2.7), $-z_i (i = 1,\dots,m)$ são chamados de zeros e $-p_i (i = 1,\dots,n)$ são os polos do sistema. Nota-se que para funções de transferência com coeficientes reais, os polos e os zeros serão reais ou complexos conjugados em pares.

Para obter-se a representação zero-polo-ganho em MATLAB, basta utilizar os seguintes

comandos:

```
» z=-[z1; z2; ...; zm];
  p=-[p1; p2; ...; pn];
  G=zpk(z,p,K)
```

Alternativamente, o modelo zero-polo-ganho pode ser declarado antes por

```
s=zpk('s')
```

Exemplo 2.7. O modelo zero-polo-ganho

$$G(s) = 6\frac{(s+1.9294)(s+0.0353\pm 0.9287\text{j})}{(s+0.9567\pm 1.2272\text{j})(s-0.0433\pm 0.6412\text{j})}$$

pode ser facilmente representado em MATLAB através dos seguintes comandos:

```
» z=-[1.9294; 0.0353+0.9287j; 0.0353-0.9287j];
  p=-[0.9567+1.2272j; 0.9567-1.2272j; -0.0433+0.6412j; -0.0433-0.6412j];
  G=zpk(z,p,6)
```

e será mostrado como

$$G(s) = \frac{6(s+1.929)(s^2+0.0706s+0.8637)}{(s^2-0.0866s+0.413)(s^2+1.913s+2.421)}$$

2.5 Modelagem com Diagramas de Blocos Interconecatdos

Os modelos apresentados até agora são utilizados para sistemas com apenas um bloco. Em sistemas práticos, o modelo com vários blocos interconectados é mais viável. As seções a seguir mostrarão métodos para obtenção desse tipo de modelo.

2.5.1 Conexão Série

Considere a conexão série dos dois blocos como mostrado na figura 2.2(a). É possível ver que o sinal u(t) é entrada apenas do bloco G_1(s) e a sua saída será a entrada do bloco G_2(s) que, por sua vez, gera o sinal y(t), que é a saída do sistema. Este tipo de conexão é chama de conexão

série ou conexão em cascata.

A função de transferência de uma conexão série é dada por $G(s) = G_2(s)G_1(s)$. Para sistemas SISO, os bloco $G_1(s)$ e $G_2(s)$ são intercambiáveis, ou seja, $G_1G_2 = G_2G_1$. Para sistemas MIMO, entretanto, os dois blocos são geralmente não intercambiáveis.

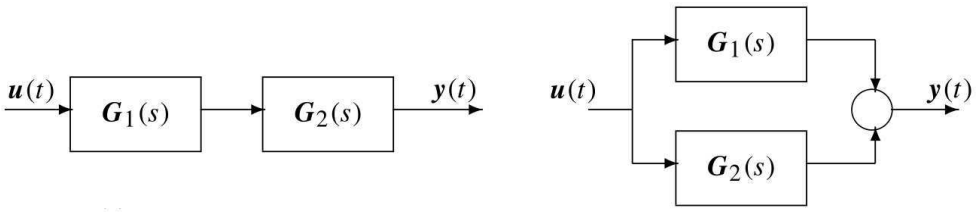
Assuma que a descrição MATLAB do modelo G_1**(s)** é representada por um objeto LIT G_1, que pode ser `tf`, `ss` ou `zpk` e que $G_2(s)$ é representado por G_2. O comando que irá gerar o sistema completo será $G = G_2 * G_1$.

A operação acima é válida mesmo que G_1 e G_2 sejam variáveis simbólicas.

2.5.2 Conexão Paralela

Uma conexão paralela típica de dois blocos $G_1(s)$ e $G_2(s)$ são mostrados na figura 2.2(b), onde os dois blocos possuem a mesma entrada $u(t)$. As saídas dos dois blocos são somadas para formar o sinal de saída do sistema $y(t)$. Então, a função de transferência da conexão paralela é $G(s)=G_1(s)+G_2(s)$.

A representação LIT da conexão paralela pode ser obtida utilizando o MATLAB através do comando $G = G_1 + G_2$, onde G_1 e G_2 são objetos LIT (`tf`, `ss` ou `zpk`) de $G_1(s)$ e $G_2(s)$, respectivamente.



(a) conexão série (b) conexão paralela

Figura 2.2: Interconexões de blocos

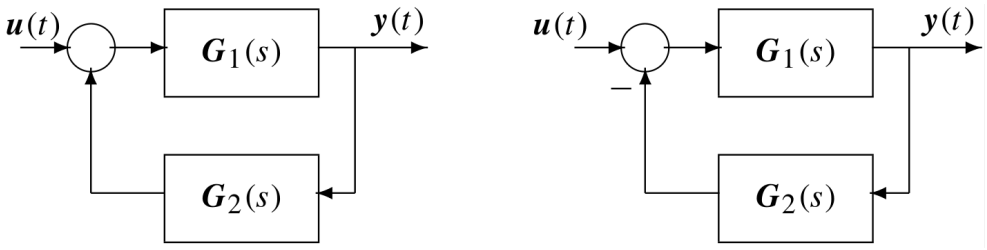
Exemplo 2.8. É possível notar que se $G_1(s)$ e $G_2(s)$ contêm os mesmos polos, então o resultado da manipulação paralela pode ser simplificada mais a frente. Considere os blocos $G_1(s) = 1/(s+1)^2$ e $G_2(s) = 1/(s+1)$. O resultado das chamadas de funções MATLAB apropriadas é mostrado através dos seguintes comandos:

```
» G1=zpk([],[-1,-1],1); G2=zpk([],[-1],1); G=G1+G2
```

Como resultado, a FT do sistema obtida é $G = (s+2)(s+1)/(s+1)^3$. Essa FT pode ser simplificada para $G = (s+2)/(s+1)^2$.

2.5.3 Conexão de Realimentação

Duas conexões de realimentação entre dois blocos são mostradas nas figuras 2.3(a) e 2.3(b), de modo que a primeira é chama de realimentação positiva e a segunda de realimentação negativa. A FT do sistema com realimentação positiva é $G(s) = G_1(s)[I - G_2(s)G_1(s)]^{-1}$ e a do sistema com realimentação negativa é $G(s) = G_1(s)[I + G_2(s)G_1(s)]^{-1}$.



(a) realimentação positiva (b) realimentação negativa

Figura 2.3: Conexões de realimentação

A função `feedback()` do MATLAB pode ser utilizada para obter a FT do modelo com realimentação, utilizando a sintaxe

`G=feedback(`G_1`,`G_2`,Sign)`

onde `Sign` é utilizado para identificar se a realimentação é positiva ou negativa. Se `Sign=1`, a realimentação será positiva. A variável `Sign` pode ser omitida caso a realimentação seja negativa. Os objetos LIT do caminho direto e do caminho de realimentação são dados por G_1 e G_2, respectivamente.

A função `feedback()` é escrita da seguinte maneira na biblioteca:

```
function H=feedback(G1,G2,key)
if nargin==2; key=-1; end
H=G1/(sym(1)-key*G1*G2); H=simple(H);
```

Exemplo 2.9. Considere novamente os modelos apresentados no exemplo 2.8. Se for considerada uma realimentação negativa, é possível encontrar a FT utilizando os seguintes comandos MATLAB:

```
» G1=tf(1,[1 2 1]); G2=tf(1,[1 1]); G=feedback(G1,G2)
```

para encontrar que

$$G(s) = \frac{s+1}{s^3+3s^2+3s+2}$$

Para uma conexão com realimentação positiva, o modelo pode ser obtido com

```
» G=feedback(G1,G2,+1)
```

onde

$$G(s) = \frac{s+1}{s^3+3s^2+3s}$$

2.6 Conversão entre Diversos Modelos

Nas seções anteriores, foram discutidos três modelos LIT. Do ponto de vista numérico, o espaço de estados é o mais adequado, especialmente para sistemas de ordem alta. De fato, cada um dos modelos podem ser convertidos em outros, desde que sejam equivalentes. Nesta seção, algumas conversões típicas de modelos serão discutidas.

2.6.1 Conversão para Funções de Transferência

Dado o espaço de estados (A,B,C,D)

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

com condições iniciais iguais a zero, a transformada de Laplace é dada

$$\begin{cases} sIX(s) = AX(s) + BU(s) \\ Y(s) = CX(s) + DU(s) \end{cases}$$

onde I é a matriz identidade com mesma dimensão que A. Então, para a primeira linhda da equação acima, temos que

$$X(s) = (sI - A)^{-1}BU(s).$$

A função de transferência equivalente pode ser obtida com

$$G(s) = Y(s)U^{-1}(s) = C(sI - A)^{-1}B + D \tag{2.8}$$

Em geral, para sistemas MIMO, a matriz $G(s)$ pode ser obtidade de 2.8. Se for dado o modelo zero-polo-ganho, basta expandir os polinômios do numerador e denominador e multiplicar pelo ganho para obter a função de transferência.

Exemplo 2.10. Suponha que um sistema seja descrito pelo espaço de estados

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -2 \end{bmatrix} u(t), \quad y(t)\,[1,0,0,0]\,x(t).$$

Utilizando os comandos MATLAB

```
» A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,5,0];
  B=[0;1;0;-2]; C=[1,0,0,0]; D=0; G=ss(A,B,C,D); G1=tf(G)
```

pode-se obter a função de transferência

$$G_1(s) = \frac{s^2 + 1.303 \cdot 10^{-13} - 3}{s^4 - 5s^2}$$

Exemplo 2.11. Para o sistema dado na forma de zero-polo-ganho

$$G(s) = 6.8\frac{(s+3)(s+7)}{s(s+1.8 \pm \mathrm{j}1.63)(s+1)^2},$$

é possível obter a função de transferência utilizando os comandos MATLAB

```
» z=[-3; -7]; p=[0; -1.8+1.63j; -1.8-1.63j; -1; −1];
  K=6.8; G=zpk(z,p,K); G1=tf(G)
```

Isto resulta em

$$G(s) = \frac{6.8s^2 + 68s + 142.8}{s^5 + 5.6s^4 + 14.1s^3 + 15.39s^2 + 5.897s}.$$

2.6.2 Conversão para Modelos Zero-Polo-Ganho

Tendo obtido a função de transferência, não é uma tarefa difícil obter o modelo zero-polo-ganho equivalente. Para isto, basta representar numerador e denominador na forma fatorada.

Exemplo 2.12. O espaço de estados mostrado no exemplo 2.10 pode ser convertido em um modelo zero-polo-ganho utilizando os seguintes comandos MATLAB:

```
» A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,5,0];
  B=[0;1;0;-2]; C=[1,0,0,0]; D=0; G=ss(A,B,C,D); G1=zpk(G)
```

onde

$$G(s) = \frac{(s-1.732)(s+1.732)}{s^2(s-2.236)(s+2.236)}.$$

Exemplo 2.13. O espaço de estados mostrado no exemplo 2.11 pode ser convertido em um modelo zero-polo-ganho utilizando os seguintes comandos MATLAB:

```
» z=[-3; -7]; p=[0; -1.8+1.63j; -1.8-1.63j; -1; -1];
  K=6.8; G=zpk(z,p,K); G1=tf(G)
```

onde

$$G_2(s) = \frac{6.8(s+7)(s+3)}{s(s+1)^2(s^2+3.6s+5.897)}.$$

2.6.3 Realizações em Espaço de Estados

Embora, para um dado modelo em espaço de estados, a função de transferência única possa ser obtida, a transformação inversa, isto é, encontrar uma espaço de estados ou realização para uma dada função de transferência, não é única. Isso pode ser mostrado através do exemplo do circuito RLC do início do capítulo, em queo espaço de estados pode ser diferente se as variáveis de estado são selecionadas de maneira diferente. A transformação de uma dada função de transferência para um espaço de estados é chamada de realização do espaço de estados da função de transferência.

Exemplo 2.14. Considere a função de transferência SISO

$$G(s) = \frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

Utilizando os comandos MATLAB

```
» num=[1,7,24,24]; den=[1,10,35,30,50,24]; G=tf(num, den); G1=ss(G)
```

o seguinte espaço de estados pode ser obtido:

$$\begin{cases} \dot{x}(t) = \begin{bmatrix} -10 & -4.375 & -3.125 & -1.5 \\ 8 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} u(t), \\ y(t) = [0.5, 0.4375, 0.75, 0.75]\, x(t), \end{cases}$$

Exemplo 2.15. Uma função de transferência MIMO pode ser convertida em um espaço de

estados utilizando a mesma função `ss()`. Considere a função de transferência MIMO

$$G(s) = \begin{bmatrix} 1/(s+1) & 0 & (s-1)/[(s+1)(s+2)] \\ -1/(s-1) & 1/(s+2) & 1/(s+2) \end{bmatrix}.$$

Utilizando comandos MATLAB

```
» s=tf('s'); h11=tf(1,[1,1]); h12=0; h13=(s-1)/(s+1)/(s+2);
  h21=tf(-1,[1,-1]); h22=tf(1,[1,2]); h23=tf(1,[1,2]);
  H=[h11,h12,h13; h21,h22,h23]; G=ss(H)
```

obtem-se o espaço de estados

$$\begin{cases} \dot{x}(t) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & -2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} u(t), \\ y(t) = \begin{bmatrix} 1 & 0 & 0 & 0.5 & -0.5 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 \end{bmatrix} x(t). \end{cases}$$

Forma canônica controlável

Suponha que a função de transferência é dada como em (2.5). A forma canônica controlável pode ser escrita como

$$\begin{cases} \dot{x} = A_C x + B_C u \\ y = C_C x + D u \end{cases} \Longrightarrow \begin{cases} \dot{x} = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_1 & -a_2 & \cdots & -a_n \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u, \\ y = [b_1, b_2, \ldots, b_n] x. \end{cases}$$

Forma canônica observável

A forma canônica observável de (2.5) é

$$\begin{cases} \dot{x} = A_O x + B_O u \\ y = C_O x + Du \end{cases} \quad \Longrightarrow \quad \begin{cases} \dot{x} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_1 \\ 1 & 0 & \cdots & 0 & -a_2 \\ 0 & 1 & \cdots & 0 & -a_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_n \end{bmatrix} x + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} u, \\ y = [0,0,\ldots,1]\,x. \end{cases}$$

É possível ver que as formas canônicas controlável e observável são duais. Isto é,

$$A_C = A_0^T, \quad B_C = C_O^T, \quad C_C = B_O^T,$$

onde (A_C, B_C, C_C, D) denota a realização do espaço de estados da forma canônica controlável e (A_C, B_C, C_C, D) denota a realização da forma canônica observável.

Forma canônica de Jordan

Assuma que os autovalores da matriz A sejam $\lambda_1, \lambda_2, \ldots, \lambda_n$ e seu i-ésimo autovetor correspondente ao i-ésimo autovalor λ_i é denotado por υ_i de modo que

$$A\upsilon_i = \lambda_i \upsilon_i, \quad i = 1,2,\ldots,n.$$

A matriz modal Λ de A é definida como

$$\Lambda = T^{-1}AT = \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_k \end{bmatrix},$$

onde os J_i's são chamadas de matrizes de Jordan. Suponha que exista uma transformação T_C de modo que um dado espaço de estados pode ser transformado para uma forma canônica controlável; então, a matriz de transformação T pode ser construída como $T = UT_C$ de modo que a realização modal pode ser obtida, onde $U = [U_1, U_2, \ldots, U_k]$. Os dois casos a seguir são considerados como formas canônicas de Jordan.

1. Se $\lambda_{i,i+1}$ é um par complexo conjugado, de modo que $\lambda_{i,i+1} = -\sigma \pm j\omega_i$, a matriz de

Jordan tem a forma

$$J_i = \begin{bmatrix} \sigma_i & \omega_i \\ -\omega_i & \sigma_i \end{bmatrix}, \quad U_i = \begin{bmatrix} 1 & 0 \\ \sigma_i & \omega_i \\ \vdots & \vdots \\ \mathrm{Re}[\lambda_i^{n-1}] & \mathrm{Im}[\lambda_i^{n-1}] \end{bmatrix}.$$

2. Se λ_i é um autovalor real com multiplicidade m_i, a matriz de Jordan é

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_i \end{bmatrix}$$

e a matriz de transformação U_i é

$$U_i = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \lambda_i & 1 & 0 & \cdots & 0 \\ \lambda_i^2 & 2\lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_i^{n-1} & \frac{\mathrm{d}}{\mathrm{d}\lambda_i(\lambda_i^{n-1})} & \frac{1}{2!}\frac{\mathrm{d}^2}{\mathrm{d}\lambda_i^2}(\lambda_i^{n-1}) & \cdots & \frac{1}{(m_i-1)!}\frac{\mathrm{d}^{m_i-1}}{\mathrm{d}\lambda_i^{m_i-1}}(\lambda_i^{n-1}) \end{bmatrix}$$

A função `canon()` do MATLAB utiliza a sintaxe `[`G_1`,`T`]=canon(`G`,type)`, onde G é o espaço de estados do sistema original e G_1 é o espaço de estados após a conversão. O argumento `type` pode ser `'companion'` ou `'modal'`. T é a matriz transformação.

Exemplo 2.16. Considere o sistema dado por

$$G(s) = \frac{3s^2 + 21s + 36}{s^4 + 5s^3 + 10s^2 + 10s + 4}.$$

Utilizando os comandos MATLAB

```
» G=tf([3 21 36],[1 5 10 10 4]); G1=canon(G,'modal')
```

a realização de Jordan pode ser obtida:

$$\begin{cases} \dot{x}(t) = \begin{bmatrix} -2 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} x(t) + \begin{bmatrix} -48.66 \\ 18.19 \\ 19.09 \\ 39.40 \end{bmatrix} u(t), \\ y = [0.061, -0.392, -0.411, 0.456]\, x(t). \end{cases}$$

2.6.4 Conversão entre Modelos em Tempo Contínuo e Discreto

Se um modelo é descrito como um objeto contínuo LIT G, sua versão discreta sob um intervalo de amostragem T pode ser facilmente obtida com a sintaxe G_d=`c2d(`G`,`T`)`. O método de discretização padrão utilizado é o *segurador de ordem zero* (*zero-order-hold* ou, simplesmente, ZOH). Outro método de discretização é o de Tustin, que pode ser utilizado com a sintaxe G_d=`c2d(`G`,`T`,'Tustin')`. Se, por outro lado, um objeito em tempo discreto G_d é conhecido, sua versão contínua pode ser obtida com G=`d2c(`G_d`)`, onde o período de amostragem T não é necessário, uma vez que esta informação já está contida em G_d.

Exemplo 2.17. Considere o sistema MIMO

$$\dot{x} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix} x + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix} u, \quad y = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix} x \ .$$

Para amostrá-lo com período de amostragem $T = 0.1$ segundos, utilizam-se os comandos

```
» A=[2.25, -5, -1.25, -0.5; 2.25, -4.25, -1.25, -0.25;
  0.25, -0.5, -1.25,-1; 1.25, -1.75, -0.25, -0.75];
  B=[4, 6; 2, 4; 2, 2; 0, 2]; C=[0, 0, 0, 1; 0, 2, 0, 2];
  D=zeros(2,2); G=ss(A,B,C,D); Gd=c2d(G,0.1)
```

dos quais obtêm-se as matrizes em tempo discreto:

$$F = \begin{bmatrix} 1.1915 & -0.4455 & -0.1013 & -0.04215 \\ 0.2008 & 0.6124 & -0.1058 & -0.01884 \\ 0.01526 & -0.03499 & 0.8849 & -0.09054 \\ 0.1147 & -0.1622 & -0.01973 & 0.9279 \end{bmatrix}, \quad G = \begin{bmatrix} 0.383253 & 0.5527 \\ 0.1906 & 0.3694 \\ 0.1879 & 0.1764 \\ 0.004833 & 0.1927 \end{bmatrix}$$

Exemplo 2.18. Se o modelo contínuo é dado por

$$G(s) = \frac{1}{(s+2)^3} e^{-2s},$$

e o período de amostragem é $T = 0.1$ segundos, os seguintes comandos podem ser utilizados para discretizar o sistema:

```
» s=tf('s'); G=1/(s+2)^3; G.ioDelay=2
```

Se forem utilizados os algoritmos ZOH e Tustin, os comandos serão

```
» G1=c2d(G,0.1) % método ZOH
  G2=c2d(G,0.1,'Tustin') % algoritmo de Tustin
```

encontrando

$$G_{ZOH}(z) = \frac{0.0001436z^2 + 0.0004946z + 0.0001064}{z^3 - 2.45z^2 + 2.011z - 0.5488} z^{-20},$$

$$G_{Tustin}(z) = \frac{9.391 \cdot 10^{-5} z^3 + 0.0002817z^2 + 0.0002817z + 9.391 \cdot 10^{-5}}{z^3 - 2.455z^2 - 0.5477} z^{-20}.$$

Se a conversão inversa for utilizada, isto é,

```
» G1c=d2c(G1), G2c=d2c(G2)
```

é possível encontrar

$$G_{1c}(s) = \frac{-6.303 \cdot 10^{-16} s^2 - 3.41 \cdot 10^{-15} s + 1}{s^3 + 6s^2 + 12s + 8},$$

$$G_{2c}(s) = \frac{9.391 \cdot 10^{-5} s^3 + 0.003096s^2 + 0.04542s + 1.01}{s^3 + 6.02s^2 + 12.08s + 8.081}.$$

3 *Análise de Sistemas de Controle Lineares*

A propriedade mais importante de um sistema linear é o princípio da sobreposição. Assuma que a resposta do sistema para um sinal $u_1(t)$ seja $y_1(t)$ e a resposta para um sinal $u_2(t)$ seja $y_2(t)$. Então, o sistema é linear se para quaisquer constantes a e b, a resposta para o sinal $au_1(t) + bu_2(t)$ possa ser representada por $ay_1(t) + by_2(t)$.

Os modelos discutidos no capítulo anterior (função de transferência, zero-polo-ganho, espaço de estados etc.) são todos lineares invariantes no tempo (LIT). Dado um modelo matemático que descreve um sistema, iremos discutir nesse capítulo outras propriedades que podem ser obtidas com relação a esse sistema linear. Primeiro, a estabilidade de sistemas LIT são discutidas na seção 3.1. Também serão discutidas estabilidade de sistemas com realimentação, propriedades de controlabilidade, observabilidade, decomposição de Kalman e norma de sistemas. São apresentadas as formas canônicas de sistemas lineares de controle e a definição e desenvolvimento dos parâmetros de Markov.

3.1 Propriedades de Sistemas de Controle Lineares

3.1.1 Análise de Estabilidade

Avaliação direta de estabilidade

Quando realimentação é utilizada, um sistema pode ser estável ou instável. Isto é, um sistema em malha aberta pode se tornar instável após a implementação do controle com realimentação, o que é indesejável. De modo oposto, um sistema instável em malha aberta pode se tornar estável após a realimentação, o que é desejável. Portanto, estabilidade é fundamental para qualquer sistema de controle.

Existem diversas noções de estabilidade. Aqui, discutiremos inicialmente a noção de estabilidade limitada em entrada-limitada em saída (BIBO, do inglês bounded input-bounded output).

Estabilidade BIBO de um sistema assegura que a saída será limitada se a entrada também for.

O modo mais fácil de verificar a estabilidade de um sistema contínuo linear é analisando a localização de seus polos no plano complexo. Se existir um polo com parte real positiva, o sistema é dito instável. Em outras palavras, se existir um ou mais polos no semiplano direito do plano complexo, o sistema é instável. O sistema é estável apenas se todos os polos tiverem parte real negativa, ou seja, todos os polos estão no semiplano esquerdo.

Para polos no eixo imaginário, existem dois casos. Se um polo específico é simples, ou seja, possuir multiplicidade 1, este polo é marginalmente estável. Se um polo específico possuir multiplicidade maior que 1, então este polo é instável. Perceba que noção de estabilidade BIBO é muito forte. Por exemplo, um integrador é estável mas não BIBO, uma vez que para uma entrada em degrau (limitada) a saída será t (não limitada).

Os polos de um dado modelo LIT G pode ser obtido no MATLAB diretamente com `eig(G)` ou `pole(G)`. O sistema será estável se a parte real dos valores forem negativas. Entretanto, se G é um sistema discreto, o sistema será instável se `abs(eig(G))` ou `abs(pole(G))` forem menor que 1, ou seja, se os polos estiverem localizados no círculo unitário.

Os zeros do sistema G podem ser obtidos com a função `zero(G)` e polos e zeros podem ser impressos com a função `pzmap(G)`.

Exemplo 3.1. Suponha que a função de transferência de um sistema é dada por

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10^3 + 35s^2 + 50s + 24}$$

.

Como ilustrado abaixo, os polos podem ser calculados e impressos com os seguintes comandos

```
»G=tf([1,7,24,24],[1,10,35,50,24]); eig(G), pzmap(G)
```

com os quais serão encontrados polos localizados em -1, -2, -3, -4, todos no semiplano esquero do plano-s, o que significa que G é estável. Isso também pode ser verificado através da figura 3.1.

Exemplo 3.2. Suponha que em um sistema com realimentação discreto no tempo o modelo da planta seja dado por

$$H(z) = \frac{6z^2 - 0.6z - 0.12}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125},$$

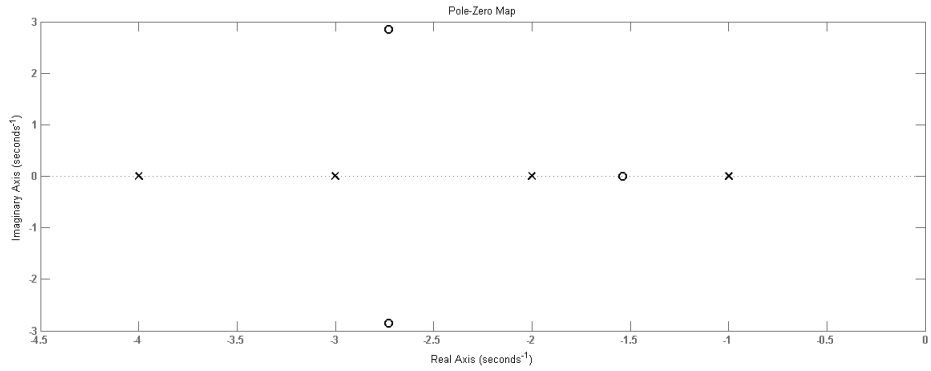


Figura 3.1: Exemplo 3.1

com intervalo de amostragem $T = 0.1$ e o controlador seja

$$G_c(z) = 0.3\frac{z-0.6}{z+0.8}.$$

A estabilidade de malha fechada pode ser obtida através dos seguintes comandos:

```
» num=[6 -0.6 -0.12]; den=[1 -1 0.25 0.25 -0.125];
  H=tf(num,den,'Ts',0.1); % Modelo da planta
  z=tf('z','Ts',0.1); Gc=0.3*(z-0.6)/(z+0.8); % Modelo do controlador
  GG=feedback(H*Gc,1); abs(eig(GG)), pzmap(GG)
```

As magnitudes dos polos são 1.1644, 1.1644, 0.5536, 0.3232 e 0.3232. Uma vez que a magnitude dos dois primeiros polos são maiores que 1, ou seja, estão fora do círculo unitário, então o sistema em malha fechada é instável. Esse resultado também pode ser visto através da figura 3.2.

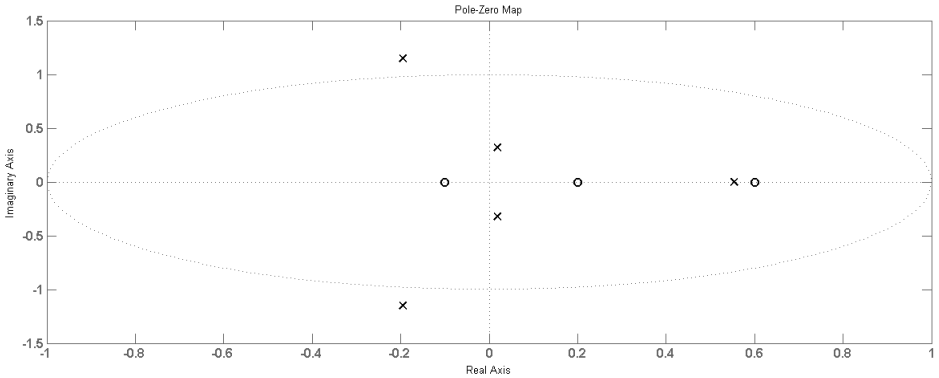


Figura 3.2: Exemplo 3.2

Bem-posto e estabilidade interna Com relação a um bom comportamento em controle em um sistema realimentado, os critérios de estabilidade nas seções anteriores não são suficientes,

uma vez que somente a estabilidade de entrada-saída do sistema é considerado. Em outras palavras, os critérios de estabilidade asseguram que se a entrada for limitada, a saída também será limitada. Se os sinais internos do sistema não forem limitados, sua estrutura física pode ser comprometida.

Considere o sistema com realimentação genérico mostrado na figura 3.3 que é uma extensão imediata da estrutura de um típico sistema com realimentação. Na figura 3.3, o sinal d é frequentemente chamado de perturbação externa e o sinal n de ruído de medição.

Antes de introduzir o conceito de estabilidade interna, a definição de bem-posto será apresentada.

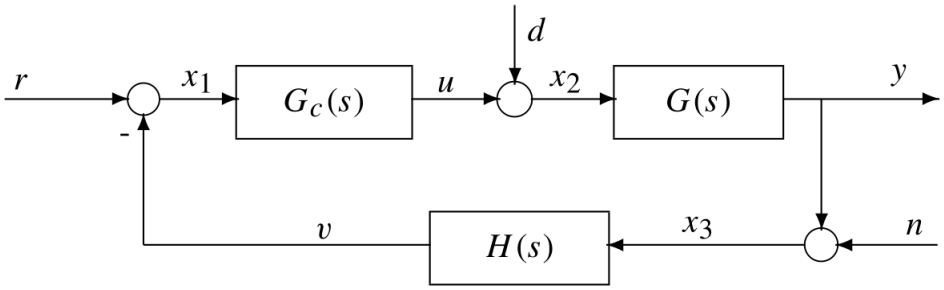


Figura 3.3: Estrutura de um sistema linear realimentado com perturbações

Definição 3.1. O sistema realimentado mostrado na Figura 3.3 é chamado de bem-posto se todas as nove funções de transferência de malha fechada de sinais de entrada (r,d,n) para sinais de saída (u,y,v) existem.

Um sistema bem-posto pode ser determinado pelo teorema a seguir.

Teorema 3.1. O sistema é bem-posto se, e somente se, a matriz 3×3

$$\begin{bmatrix} 1 & 0 & H(s) \\ -G_c(s) & 1 & 0 \\ 0 & -G(s) & 1 \end{bmatrix}$$

é não-singular, isto é, o determinante $1 + G(s)G_c(s)H(s)$ não é igual a zero.

Definição 3.2. O sistema mostrado na Figura 3.3 é dito internamente estável se todas as nove funções de transferência de malha fechada de sinais de entrada (r,d,n) para saídas (x_1,x_2,x_3) são estáveis.

As nove funções de transferência podem ser descritas por

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{1+G(s)G_c(s)H(s)} \begin{bmatrix} 1 & -G(s)H(s) & -H(s) \\ G_c(s) & 1 & -G_c(s)H(s) \\ G(s)G_c(s) & G(s) & 1 \end{bmatrix} \begin{bmatrix} r \\ d \\ n \end{bmatrix}.$$

Teorema 3.2. O sistema é internamente estável se, e somente se, as duas condições a seguir forem satisfeitas:

1. A função de transferência $1+H(s)G(s)G_c(s)$ não possui zeros em $\mathrm{Re}[s] \geq 0$.
2. $H(s)G(s)G_c(s)$ não possui cancelamento polo-zero em $\mathrm{Re}[s] \geq 0$.

A primeira condição é equivalente a dizer que o sistema em malha-fechada é estável em entrada-saída. Então, basta concentrar-se na segunda condição, a qual pode ser facilmente checada utilizando a função MATLAB `intstalbe()`:

```
function [V,c]=intstalbe(G,Gc,H)
GG=minreal(feedback(G*Gc,H)); Go=H*G*Gc; Go1=minreal(Go); p=eig(GG);
z0=eig(Go); z1=eig(Go1); zz=setdiff(z0,z1); % encontra os cancelamentos
if (G.Ts>1), % sistema discreto no tempo
    V=any(abs(p)>1);
    if V==0, V=2*any(abs(zz)>1); if V==2, c=zz(abs(zz)>1); end
    else c=p(abs(p)>1); end
else % sistema contínuo
    V=any(real(p)>0);
    if V==0, V=2*any(real(zz)>0); if V==2, c=zz(real(zz)>0); end
    else c=p(real(p)>0); end
end
```

Sua sintaxe é `[V,c]=intstable(G,Gc,H)`. Na chamada da função, as variáveis retornadas são definidas como:

1. Se o sistema é internamente estável, é retornado $V = 0$ e c é vazio.
2. Se $V = 1$, o sistema é instável em entrada-saída e os polos de malha fechada instáveis são retornados no vetor c,

3. Se $V = 2$, o sistema é estável em entrada-saída, mas não é internamente estável, e os polos instáveis internamente cancelados são retornados em c.

Exemplo 3.3. Considere o típico sistema com realimentação

$$G(s) = \frac{5(s-1)(s+2)}{s^3+4s^2+3s+4}, \quad G_c(s) = \frac{s^2+3s+4}{(s-1)(s^2+3s+2)}, \quad H(s) = 1.$$

A estabilidade do sistema pode ser testada com os seguintes comandos MATLAB:

```
» s=tf('s'); G=5*(s-1)*(s+2)/(s^3+4*s^2+3*s+4);
  Gc=(s^2+3*s+4)/((s-1)*(s^2+3*s+2)); H=1;
  G_a=minreal(ss(feedback(Gc*G,H))); eig(G_a)
```

Os polos de malha fechada do sistema, depois do cancelamento dos dois pares polo-zero, são localizados em $-0.2328 \pm \text{j}2.0546$, $-2.2672 \pm \text{j}0.6879$, que estão todos no semiplano esquerdo do plano s. É possível ver que o sistema em malha fechada é estável. Entretanto, checando a estabilidade interna com

```
» [V,cc]=intstable(G,Gc,H)
```

concluímos que o sistema não é internamente estável com pares polo-zero cancelados tendo parte real positiva $z_i = p_j = 1$ retornado em `cc`.

3.1.2 Análise de Controlabilidade e Observabilidade

Controlabilidade de sistemas lineares

Controlabilidade é uma importante propriedade de um sistema de controle e tem um papel crucial em muitos problemas de controle, tal como a estabilização de sistemas instáveis por controle realimentado.

Definição 3.3. O estado $x_i(s)$ é dito controlável se há uma entrada que, em um tempo finito, alcança um valor $x_i(t_f)$ a partir de um valor inicial $x_i(0)$. O sistema é dito totalmente controlável se todos os seus estados são controláveis.

Uma vez que a total controlabilidade de um sistema depende somente das matrizes A e B do modelo de espaço de estados, basta afirmar que (A, B) é controlável.

Construa a matriz de transformação T_c na forma

$$T_c = \left[B, AB, \ldots, A^{n-1}B\right],$$

onde n é a ordem do sistema ou o número de estados. A matriz T_c é chamada de matriz de controlabilidade e pode ser gerada utilizando a função `ctrb()` do MATLAB, com a sintaxe T_c=`ctrb(`A,B`)`.

Sob uma matriz de transformação $\widehat{T}_c$ adequada, o espaço de estados pode ser transformado na seguinte forma canônica, através da transformação para a forma escada:

$$A_c = \begin{bmatrix} \widehat{A}_{\bar{c}} & 0 \\ \widehat{A}_{21} & \widehat{A}_c \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ \widehat{B}_c \end{bmatrix}, \quad C_c = \begin{bmatrix} \widehat{C}_{\bar{c}}, & \widehat{C}_c \end{bmatrix}.$$

A representação acima é conhecida como forma escada de controlabilidade. Os autovalores de $\widehat{A}_{\bar{c}}$ são chamados de modos não-controláveis. Se o sistema é controlável, o subespaço não-controlável $\widehat{A}_{\bar{c}}$ será vazio. A função de transferência simplificada do sistema pode ser obtida do subespaço controlável

$$G(s) = \widehat{C}_c(sI - \widehat{A}_c)^{-1}\widehat{B}_c + D.$$

A função `ctrbf()` pode ser utilizada para realizar a transformação para a forma escada controlável. A sintaxe desta função é `[`A_c, B_c, C_c, T_c`]=ctrbf(`A,B,C`)`, onde (A,B,C) é o espaço de estados dado e o espaço de estados retornado tem uma forma escada que separa os subespaços controláveis e não-controláveis. A matrix retornada T_c é a matriz de transformação.

Exemplo 3.4. Considere o espaço de estados dado por

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -2 \end{bmatrix} u, \quad y = [1,0,0,0]]x.$$

É possível utilizar os seguintes comandos para checar a controlabilidade do sistema:

```
» A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,5,0]; B=[0; 1; 0; -2];
  C=[1,0,0,0]; D=0; Tc=[B, A*B, A^2*B, A^3*B]; rank(Tc)
```

Como `rank(Tc)` retorna 4, que é a ordem do sistema, então o sistema é totalmente controlável.

Exemplo 3.5. Vamos considerar outro sistema dado por

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & -2 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} u, \quad y = [1,0,0,0]]x \ .$$

A controlabilidade pode ser analisada utilizando os seguintes comandos:

```
» A=[0,1,0,0; 3,0,0,2; 0,0,0,1; 0,-2,0,0]; B=[0; 1; 0; 0];
  C=[1,0,0,0]; Tc=[B, A*B, A^2*B, A^3*B]; rank(Tc)
  [Ac,Bc,Cc,T]=ctrbf(A,B,C)
```

A forma escada controlável pode ser escrita, na forma particionada, como

$$A_c = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \hline 0.4472 & 0 & 0.8944 & 0 \\ 0 & 0 & 0 & 2.236 \\ -3.5777 & 0 & -0.4472 & 0 \end{array}\right], \quad B_c = \left[\begin{array}{c} 0 \\ \hline 0 \\ 0 \\ 1 \end{array}\right], \quad C_c = \left[\begin{array}{c|ccc} -0.8944 & 0 & 0.4472 & 0 \end{array}\right]$$

e pode ser observado que o modo não controlável está em $s = 0$, que é o autovalor da matriz $\widehat{A}_{\overline{c}}$.

Observabilidade de sistemas lineares Observabilidade é uma medição do quão bem os estados internos de um sistema pode ser inferido a partir do conhecimento de suas entradas e saídas. A observabilidade e controlabilidade de um sistema pode são matematicamente duais.

Definição 3.4. Um estado $x_i(t)$ é dito observável se, para qualquer $t_f > 0$, o estado inicial $x_i(0)$ pode ser determinado a partir dos valores da entrada $u(t)$ e da saída $y(t)$ no intervalo $\left[0, t_f\right]$. O sistema é dito totalmente observável se todos os estados do sistema são observáveis.

Uma vez que a observabilidade do sistema depende somente das matrizes A e C do modelo de espaço de estados, basta afirmar que (A, C) é observável.

Construa uma matriz de transformação T_o na seguinte forma:

$$T_o = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix},$$

onde n é a ordem do sistema. T_o é chamada de matriz de observabilidade, que pode ser gerada utilizando a função `obsv()` do MATLAB, com a sintaxe T_o=`obsv(`A,C`)`.

Sob uma matriz de transformação $\widehat{T}_o$ adequada, o espaço de estados pode ser transformado na seguinte forma canônica:

$$A_o = \begin{bmatrix} \widehat{A}_{\overline{o}} & \widehat{A}_{12} \\ 0 & \widehat{A}_o \end{bmatrix}, \quad B_o = \begin{bmatrix} \widehat{B}_{\overline{o}} \\ \widehat{B}_o \end{bmatrix}, \quad C_o = \begin{bmatrix} 0, & \widehat{C}_o \end{bmatrix},$$

conhecida como a forma escada observável. Os autovalores de $\widehat{A}_{\overline{o}}$ são chamados modos não-observáveis. Se o sistema é totalmente observável, o subespaço não-observável $\widehat{A}_{\overline{o}}$ será vazio. A função de transferência do sistema pode ser expressado por

$$\mathbf{G}(s) = \widehat{\mathbf{C}}_o(s\mathbf{I} - \widehat{\mathbf{A}}_o)^{-1}\widehat{\mathbf{B}}_o + \mathbf{D}.$$

Comparando-se os problemas de controlabilidade e observabilidade, não é difícil observar que eles são duais. Isto é, o problema de observabilidade do sistema ($\mathbf{A}^T$,$\mathbf{B}^T$,$\mathbf{C}^T$,$\mathbf{D}^T$) é exatamente o mesmo que o problema de controlabilidade do sistema ($\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}$).

A função `obsvf()` pode ser utilizada para realizar transformação para a forma escada. A sintaxe é

$$[\mathbf{A}_o,\mathbf{B}_o,\mathbf{C}_o,\mathbf{D}_o]\texttt{=obsvf(A,B,C)}$$

e os argumentos são similares aos daqueles da função `ctrbf()`.

Exemplo 3.6. Considere novamente o sistema não-controlável mostrado no exemplo anterior. A observabilidade e a forma da escada podem ser obtidas por

```
» rank([C; C*A; C*A^2; C*A^3]) % ou rank(obsv(A,C))
  [Ao,Bo,Co,T,K]=obsvf(A,B,C); Ao,Bo,Co
```

Na forma de matriz particionada, a forma da escada pode ser escrita como

$$A_o = \left[\begin{array}{c|ccc} 0 & 1 & 0 & 0 \\ \hline 0 & 0 & -2 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{array}\right], \quad B_o = \left[\begin{array}{c} 0 \\ \hline 0 \\ 1 \\ 0 \end{array}\right], \quad C_o = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 1 \end{array}\right].$$

O modo não-observável está em $s = 0$. De fato, não é difícil verificar que há dois autovalores em $s = 0$, sendo um não-controlável e o outro não-observável.

3.1.3 Decomposição de Kalman de Sistemas Lineares

As duas propriedades, controlabilidade e observabilidade, discutidas anteriormente, implicam que existem quatro possíveis modos de um sistema linear: controlável e observável, não-controlável e observável, controlável e não-observável e não-controlável e não-observável. Dado um sistema linear, como podemos fazer sua decomposição nesses quatro modos é o tópico principal dessa seção. Essa decomposição é chamada de Decomposição de Kalman. É útil entender a estrutura de herança interna de sistemas lineares.

Decomposição de Kalan Kalman demonstrou que qualquer espaço de estados pode ser decomposto na seguinte forma canônica:

$$\begin{cases} \dot{z}(t) = \left[\begin{array}{c|c|c|c} \widehat{A}_{\bar{c},\bar{o}} & \widehat{A}_{1,2} & 0 & 0 \\ \hline 0 & \widehat{A}_{\bar{c},o} & 0 & 0 \\ \hline \widehat{A}_{3,1} & \widehat{A}_{3,2} & \widehat{A}_{c,\bar{o}} & \widehat{A}_{3,4} \\ \hline 0 & \widehat{A}_{4,2} & 0 & \widehat{A}_{c,o} \end{array}\right] z(t) + \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline \widehat{B}_{c,\bar{o}} \\ \hline \widehat{A}_{c,o} \end{array}\right] u(t), \\ y(t) = \left[\begin{array}{c|c|c|c} 0 & \widehat{C}_{\bar{c},o} & 0 & \widehat{C}_{c,o} \end{array}\right] z(t), \end{cases} \tag{3.1}$$

onde o subspaço $(\widehat{A}_{\bar{c},\bar{o}}, 0, 0)$ é não-controlável/não-observável, $(\widehat{A}_{\bar{c},o}, 0, \widehat{C}_{c,\bar{o}})$ é controlável/não-observável, $(\widehat{A}_{c,\bar{o}}, \widehat{B}_{c,\bar{o}}, 0)$ é observável/não-controlável e $(\widehat{A}_{c,o}, \widehat{B}_{c,o}, \widehat{C}_{c,o})$ é controlável/observável. Essa é chamada de decomposição de Kalan e está ilustrada na Figura 3.4.

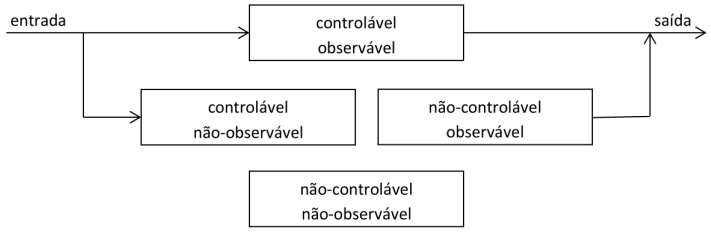


Figura 3.4: Ilustração da decomposição de Kalman

Teorema 3.3. Propriedades tais como controlabilidade e observabilidade não podem ser alteradas através de qualquer transformação.

Uma função MATLAB `kalmdec()` pode ser utilizada para realizar a decomposição de Kalman de um dado sistema:

```
function [Gk,T,K]=kalmdec(G)
G=ss(G); A=G.a; B=G.b; C=G.c; [Ac,Bc,Cc,Tc,Kc]=ctrbf(A,B,C);
nc=rank(ctrb(A,B),eps*100); n=length(A); ic=n-nc+1:n;
[Ao1,Bo1,Co1,To1,Ko1]=obsvf(Ac(ic,ic),Bc(ic),Cc(ic));
if nc<n, inc=1:n-nc;
   [Ao2,Bo2,Co2,To2,Ko2]=obsvf(Ac(inc,inc),Bc(inc),Cc(inc));
end
[m1,n1]=size(To1); [m2,n2]=size(To2); To=blkdiag(To2,To1);
T=To*Tc; e0=eps*100; n1=rank(obsv(Ac(ic,ic),Cc(ic)),e0);
n2=rank(obsv(Ac(inc,inc),Cc(inc)),e0);
K=[zeros(1,n-nc-n2),ones(1,n2), 2*ones(1,nc-n1), 3*ones(1,n1)];
Ak=T*A*inv(T); Bk=T*B; Ck=C*inv(T); Gk=ss(Ak,Bk,Ck,G.d);
```

A sintaxe da função é `[`G_k`,`T_k`,`k`]=kalmdec(`G`)`. A variável retornada G_k é a decomposição de Kalman do sistema G. A variável T_k é a matriz de transformação, enquanto k retorna um vetor que armazena as *flags* para os modos de cada estado. Se a *flag* é zero, o estado correspondente é não-controlável/não-observável. Os valores 1, 2, 3 de *flag* correspondem a não-controlável/observável, controlável/não-observável e controlável/observável, respectivamente.

Exemplo 3.7. Considere o sistema linear

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & -3 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \\ 3 \\ 0 \end{bmatrix} u, \quad y = [4, 5, 0, 0, 0, 6]\, \boldsymbol{x}.$$

Sua decomposição de Kalman pode ser realizada através dos seguintes comandos MATLAB:

```
>> A=[-1,1,0,0,0,0; 0,-1,0,0,0,0; 0,0,-2,1,0,0;
      0,0,0,-2,0,0; 0,0,0,0,-3,1; 0,0,0,0,0,-3];
   B=[1; 2; 3; 0; 4; 0]; C=[4,5,0,6,0,0]; G=ss(A,B,C,0);
   [Gk,Tk,K]=kalmdec(G),
```

que gera o familiar formato matemático a seguir:

$$\begin{cases} \dot{\boldsymbol{x}} = \left[\begin{array}{cc:cc:cc} -3 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & -2 & 0 & 0 & 0 & 0 \\ \hdashline -0.0832 & -0.9965 & -2.007 & -0.08288 & 0 & 0 \\ -0.9965 & 0.0832 & -0.08288 & -2.993 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & -1.488 & -0.6098 \\ 0 & 0 & 0 & 0 & 0.3902 & -0.5122 \end{array}\right] \boldsymbol{x} + \left[\begin{array}{c} 0 \\ \hdashline 0 \\ \hdashline -3.3223 \\ -3.7366 \\ \hdashline 0.4685 \\ 2.1864 \end{array}\right] u \\ y = \left[\begin{array}{cc:cc:cc} 0 & 6 & 0 & 0 & 0 & 6.4031 \end{array}\right] \boldsymbol{x}. \end{cases}$$

Problemas de mínima realização

Se todos os estados iniciais são zero, o sinal de saída $y(t)$ de um espaço de estados linear contínuo pode ser simplificado por

$$\boldsymbol{y}(t) = \widehat{\boldsymbol{C}}_{c,o} \int_0^t \mathrm{e}^{\widehat{\boldsymbol{A}}_{c,o}(t-\tau)} \widehat{\boldsymbol{B}}_{c,o} u(\tau) \mathrm{d}\tau,$$

que é exatamente a solução do subspaço controlável/observável. Portanto, na forma da decomposição de Kalman, o subspaço $(\widehat{A}_{c,o},\widehat{B}_{c,o},\widehat{C}_{c,o})$ é chamado de mínima realização do sistema original. Isto é, a mínima realização é sempre totalmente controlável e observável. Para funções de transferência, o "mínimo modelo realizável"é aquele em que todos os pares pólo-zero teve os mesmos valores cancelados.

O procedimento para obter a mínima realização de um sistema linear está resumido a seguir:

1. Encontre a matriz de transformação T_c^{-1} para separar as partes controlável e observável:

$$\boldsymbol{A}_c = \boldsymbol{T}_c^{-1} \boldsymbol{A} \boldsymbol{T}_c = \begin{bmatrix} \widehat{\boldsymbol{A}}_{\bar{c}} & \boldsymbol{0} \\ \widehat{\boldsymbol{A}}_{21} & \widehat{\boldsymbol{A}}_c \end{bmatrix},$$
$$\boldsymbol{B}_c = \boldsymbol{T}_c^{-1} \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} \\ \widehat{\boldsymbol{B}}_c \end{bmatrix}, \quad \boldsymbol{C}_c = \boldsymbol{C} \boldsymbol{T}_c = \begin{bmatrix} \widehat{\boldsymbol{C}}_{\bar{c}} & \widehat{\boldsymbol{C}}_c \end{bmatrix};$$

2. Encontre a matriz de transformação $\widehat{T}_o$ de modo que o subsistema controlável $(\widehat{A}_c,\widehat{B}_c,\widehat{C}_c)$ possa ser decomposto para encontrar a parte observável:

3. Construa a matriz

$$\widehat{\boldsymbol{A}}_o = \widehat{\boldsymbol{T}}_o^{-1}\widehat{\boldsymbol{A}}_c\widehat{\boldsymbol{T}}_o = \begin{bmatrix} \widehat{\boldsymbol{A}}_{c,\bar{o}} & \widehat{\boldsymbol{A}}_{c,12} \\ \boldsymbol{0} & \widehat{\boldsymbol{A}}_{c,o} \end{bmatrix},$$
$$\widehat{\boldsymbol{B}}_o = \widehat{\boldsymbol{T}}_o^{-1}\widehat{\boldsymbol{B}}_c = \begin{bmatrix} \widehat{\boldsymbol{B}}_{c,\bar{o}} \\ \widehat{\boldsymbol{B}}_{c,o} \end{bmatrix}, \quad \widehat{\boldsymbol{C}}_o = \widehat{\boldsymbol{C}}_c\widehat{\boldsymbol{T}}_o = \begin{bmatrix} \boldsymbol{0} & \widehat{\boldsymbol{C}}_{c,o} \end{bmatrix}.$$

$$\widetilde{\boldsymbol{T}}_o^{-1} = \begin{bmatrix} \boldsymbol{I}_{n-\,\mathrm{rank}\{\widehat{\boldsymbol{A}}_c\}} & \boldsymbol{0} \\ \boldsymbol{0} & \widehat{\boldsymbol{T}}_o^{-1} \end{bmatrix}.$$

Então, defina a matriz de transformação $T^{-1} = \widetilde{T}_o^{-1}T_c^{-1}$ para transformar o sistema original em um sistema na forma mínima realizada ($\widehat{A}_{c,o}$,$\widehat{B}_{c,o}$,$\widehat{C}_{c,o}$). Sob a matriz de transformação T, todo o sistema pode ser transformado na forma canônica

$$\dot{z} = \begin{bmatrix} \widehat{\boldsymbol{A}}_{\bar{c}} & \boldsymbol{0} & \\ \widehat{\boldsymbol{A}}_{21} & \widehat{\boldsymbol{A}}_{c,\bar{o}} & \widehat{\boldsymbol{A}}_{c,12} \\ & \boldsymbol{0} & \widehat{\boldsymbol{A}}_{c,o} \end{bmatrix} z + \begin{bmatrix} \boldsymbol{0} \\ \widehat{\boldsymbol{B}}_{c,\bar{o}} \\ \widehat{\boldsymbol{B}}_{c,o} \end{bmatrix} \boldsymbol{u}, \qquad \boldsymbol{y} = [\boldsymbol{C}_{\bar{c}}\ \boldsymbol{0}\ \widehat{\boldsymbol{C}}_{c,o}]z + \boldsymbol{D}\boldsymbol{u}.$$

Exemplo 3.8. Considere o seguinte espaço de estados:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -5 & 8 & 0 & 0 \\ -4 & 7 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & -2 & 6 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 4 \\ -2 \\ 2 \\ 1 \end{bmatrix} u, \quad y = [2 \ \ -2 \ \ -2 \ \ 2]\boldsymbol{x}.$$

Os três passos acima podem ser implementados utilizando os seguintes comandos MATLAB para encontrar o modelo de mínima realização

```
>> A=[-5,8,0,0; -4,7,0,0; 0,0,0,4; 0,0,-2,6]; B=[4; -2; 2; 1];
   C=[2,-2,-2,2]; D=0; [Ac,Bc,Cc,Tc]=ctrbf(A,B,C);
   [Ao,Bo,Co,To]=obsvf(Ac,Bc,Cc); A_r=Ao(3:4,3:4); B_r=Bo(3:4);
   C_r=Co(3:4); G_r=zpk(ss(A_r,B_r,C_r,D))
```

O modelo mínimo realizado obtido é $G_m(s) = 10(s-2.6)/[(s-2)(s+1)]$.

3.1.4 Instante de Tempo e Parâmetros de Markov

Assuma que a função de transferência $G(s)$ é descrita por

$$G(s) = \frac{b_1 s^k + b_2 s^{k-1} + \cdots + b_k s + b_{k+1}}{a_1 s^n + a_2 s^{n-1} + \cdots + a_n s + a_{n+1}}, \quad k \leq n,$$

onde, por questão de simplicidade, assume-se que $a_{n+1} = 1$.

Considere as propriedades do valor inicial e final da transformada de Laplace. É visto que as séries de Taylor, em particular a expansão em torno de $s = 0$ e $s = \infty$, são úteis para descrever o comportamento em regime transitório e em regime estacionário.

Expansão em torno de $s = 0$: os instantes de tempo

A expansão em série de Taylor de $G(s)$ em torno de $s = 0$, ou série de Maclaurin, pode ser escrita como

$$G(s) = \sum_{i=0}^{\infty} c_i s^i = c_0 + c_1 s + c_2 s^2 + \cdots .$$

Se e^{-st} é expandido em torno de $s = 0$ na transformada de Laplace de uma reposta ao impulso $g(t)$, $G(s)$ pode ser escrita como

$$G(s) = \int_0^{\infty} g(t) \mathrm{e}^{-st} \mathrm{d}t = \int_0^{\infty} g(t) \sum_{i=0}^{\infty} \frac{(-1)^i}{i!} (st)^i \mathrm{d}t = \sum_{i=0}^{\infty} \frac{(-1)^i}{i!} M_i s^i,$$

onde $M_i = \int_0^{\infty} t^i g(t) \mathrm{d}t$ é chamado de i-ésimo momento de tempo do resposta ao impulso da função $g(t)$. Então,

$$c_i = \frac{(-1)^i}{i!} M_i.$$

Assuma que o espaço de estados é dado por (A,B,C,D). A função de transferência de um sistema pode ser obtido de forma equivalente de

$$G(s) = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}.$$

Os momentos de tempo c_i de um sistema pode então ser calculado de

$$c_i = \frac{1}{i!}\frac{\mathrm{d}^i G(s)}{\mathrm{d}s^i}\bigg|_{s=0} = -\boldsymbol{C}\boldsymbol{A}^{-(i+1)}\boldsymbol{B}, \quad i = 0, 1, \ldots.$$

A função `timmomt()` do MATLAB pode ser utilizada para calcular os momentos de tempo de um dado modelo LIT;

```
function c=timmomt(G,k)
G=ss(G); C=G.c; B=G.b; iA=inv(G.a); iA1=iA;
c=zeros(1,k); for i=1:k, c(i)=-C*iA1*B; iA1=iA*iA1; end
```

A sintaxe da função é `c=timmomt(G,k)`, onde G é um modelo LIT e k é o número de momentos a serem calculados, e o valor retornado c é um vetor contendo os primeiros k momentos de tempo.

Exemplo 3.9. Considere o modelo de quarta ordem

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

Os primeiros sete momentos de tempo do sistema podem ser obtidos dos seguintes comandos MATLAB:

```
>> G=tf([1,7,24,24],[1,10,35,50,24]);
   c=timmomt(G,7); [n,d]=rat(c)
```

o que indica que $G(s)$ pode ser aproximado pela expansão em série de Taylor

$$G(s) = 1 - \frac{13}{12}s + \frac{157}{144}s^2 - \frac{609}{571}s^3 + \frac{899}{863}s^4 - \frac{128}{125}s^5 + \frac{386}{381}s^6 + o[s^7].$$

Expansão em torno de $s = \infty$: os parâmetros de Markov

O $G(s)$ mostrado no início desta seção pode ser expandido como uma série de potências de $1/s$, i.e.,

$$G(s) = \sum_{i=n-k}^{\infty} \delta_i \left(\frac{1}{s}\right)^i,$$

onde os coeficientes δ_i são chamados de parâmetros de Markov. Alternativamente, é equivalente a realizar a expansão em série de Taylor de $G(s)$ em torno de $s = \infty$. Para o espaço de estados (A,B,C,D), os parâmetros de Markov δ_i pode ser calculado de

$$\delta_0 = \boldsymbol{CB} + \boldsymbol{D}, \quad \text{and} \quad \delta_i = \boldsymbol{CA}^i\boldsymbol{B}, \quad i = 1, 2, \ldots.$$

Relembre as propriedades da transformada de Laplace. É fácil perceber que os momentos de tempo determinam a resposta em regime estacionário do sistema, enquanto os parâmetros de Markov determinam a resposta transitória. Em termos de resposta em frequência, os momentos de tempo determinam a resposta sobre baixas e médias frequências, enquanto os parâmetros de Markov determinam a resposta sobre médias e altas frequências.

A função `markovp()` pode ser utilizada para calcular os parâmetros de Markov de uma dada função de transferência $G(s)$:

```
function m=markovp(G,k)
G=ss(G); A=G.a; B=G.b; C=G.c; D=G.d; m=[C*B+D,zeros(1,k-1)];
A1=A; for i=1:k-1, m(i+1)=C*A1*B; A1=A*A1; end
```

A sintaxe da função é `m=markovp(G,k)`, onde G é um sistema LIT e k é o número de parâmetros de Markov a serem calculados. A variável retornada m é um vetor contendo os primeiros k parâmetros de Markov.

Exemplo 3.10. Para o sistema no exemplo anterior, os primeiros sete parâmetros de Markov podem ser obtidos utilizando os seguintes comandos MATLAB:

```
» M=Markovp(G,7)
```

de modo que a série de Taylor em $s = \infty$ pode ser obtida como

$$G(s) = 1 - \frac{3}{s} + \frac{19}{s^2} - \frac{111}{s^3} + \frac{571}{s^4} - \frac{2703}{s^5} + \frac{12139}{s^6} + o\left[\frac{1}{s^7}\right].$$

3.1.5 Norma de Sinais e Sistemas

A robustez de um sistema com realimentação é muito importante na prática de engenharia de controle. Em problemas de controle atuais, sempre existem perturbações devido ao ambiente e incertezas do modelo usado no desenvolvimento do controlador. Claramente, é desejável para os sistemas controlados tem certa robustez contra essas perturbações e incertezas. Para avaliar a robustez, primeiro de tudo, é necessário uma maneira apropriada de medição. Norma de sinais e sistemas serão mostrados, a qual pode ser considerada a base do controle robusto.

Norma de sinais

O tamanho de um sinal $u(t)$ é normalmente medido em sua norma, definida como

$$\|u(t)\|_p = \left(\int_{-\infty}^{\infty} |u(t)|^p \mathrm{d}t \right)^{1/p},$$

onde p é um inteiro positivo. As seguintes normas são comumente utilizadas:

1. A norma $\mathcal{L}_1$: $||u(t)||_1 = \int_{-\infty}^{\infty} |u(t)| dt$.
2. A norma $\mathcal{L}_2$: $||u(t)||_2 = \sqrt{\int_{-\infty}^{\infty} u(t)^2 dt}$.
3. A norma $\mathcal{L}_\infty$: $||u(t)||_\infty = sup_t |u(t)|$.

Norma de sistemas

O tamanho de um sistema na forma de função de transferência pode ser medido pelas normas $\mathcal{H}_2$ e $\mathcal{H}_\infty$.

1. A norma $\mathcal{H}_2$ é definida por

$$\|G(s)\|_2 = \sqrt{\frac{1}{2\pi \mathrm{j}} \int_{-\mathrm{j}\infty}^{\mathrm{j}\infty} |G(\mathrm{j}\omega)|^2 \mathrm{d}\omega}.$$

A norma $\mathcal{H}_2$ é de fato uma medida da raiz quadrada da integral do quadrado do valor da saída quando a entrada é um impulso. Na terminologia de sistemas estocásticos, a norma $\mathcal{H}_2$ é a raiz quadrada do sinal de saída quando a entrada é um ruído branco.

$$\|G(s)\|_\infty = \sup_{u(t)\neq 0} \frac{\|y(t)\|_2}{\|u(t)\|_2},$$

2. A norma $\mathcal{H}_\infty$ é definida por

onde $u(t)$ e $y(t)$ são a entrada e saída do sistema, respectivamente. Para sistemas estáveis, a norma $\mathcal{H}_\infty$ de um sistema pode ser calculada a partir de

$$\|G(s)\|_\infty = \sup_{\omega} |G(\mathrm{j}\omega)|.$$

É fácil perceber que a norma $\mathcal{H}_\infty$ é de fato o valor de pico da magnitude da resposta em frequência.

3.2 Análise no Domínio do Tempo de Sistemas Lineares

Destacamos que a solução analítica no domínio do tempo para um sistema linear é sempre possível dado um típico sinal de entrada. Para sinais de entrada genéricos, entretanto, a análise no domínio do tempo deve ser realizada numericamente.

3.2.1 Soluções Analíticas para Respostas em Tempo Contínuo

Método do espaço de estados

Considere um sistema LIT com seu modelo em espaço de estados n-dimensional

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{cases}$$

Foi mostrado no capítulo anterior que a solução no domínio do tempo desta equação é

$$\boldsymbol{x}(t) = \mathrm{e}^{\boldsymbol{A}t}\boldsymbol{x}_0 + \int_0^t \mathrm{e}^{\boldsymbol{A}(t-\tau)}\boldsymbol{B}u(\tau)\mathrm{d}\tau,$$

onde e^{At} é chamado de matriz de transição de estado.

Pode ser visto nesta solução que a parte mais difícil é o cálculo da integral. Se uma certa transformação for introduzida para remover o termo B, a solução para o problema original pode ser significativamente simplificado.

Assuma que o sinal de entrada é um degrau unitário; então um estado extra $\dot{(x)}_{n+1} = u(t)$ pode ser introduzido. Claramente, $\dot{x}_{n+1}(t) = 0$. Assim, a equação do espaço de estados pode ser reescrita como

$$\left[\begin{array}{c} \dot{\boldsymbol{x}}(t) \\ \hline \dot{x}_{n+1}(t) \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{B} \\ \hline \boldsymbol{0} & \boldsymbol{0} \end{array}\right] \left[\begin{array}{c} \boldsymbol{x}(t) \\ \hline x_{n+1}(t) \end{array}\right].$$

Então, o espaço de estados original pode ser convertido em um sistema autônomo

$$\left\{\begin{array}{l} \dot{\tilde{\boldsymbol{x}}}(t) = \widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{x}}(t), \\ \widetilde{\boldsymbol{y}}(t) = \widetilde{\boldsymbol{C}}\widetilde{\boldsymbol{x}}(t), \end{array}\right.$$

onde $\tilde{x}^T(t) = [x^T(t), x_{n+1}(t)]$ e $\tilde{x}^T(0) = [x^T(0), 1]$. A solução analítica pode ser facilmente encontrada como

$$\tilde{x}(t) = e^{\tilde{A}t}\tilde{x(0)}.$$

Uma classe de sinais de entrada comumente usadas, que pode ser convertida em um sistema autônomo, é definida como

$$u(t) = u_1(t) + u_2(t) = \sum_{i=0}^{m} c_i t^i + e^{d_1 t}[d_2 \cos(d_4 t) + d_3 \sin(d_4 t)]. \tag{3.2}$$

É possível introduzir alguns estados extras, chamados de estados aumentados, de modo que $x_{n+1} = e^{d_1 t}\cos(d_4 t)$, $x_{n+3} = u_1(t), \ldots, x_{n+m+3} = u_1^{(m-1)}(t)$. Pode ser mostrado que as equações do espaço de estados aumentado sob um dado sinal de entrada pode ser escrito como

cuja solução analítica é

$$\tilde{x}(t) = e^{\tilde{A}t}\tilde{x}(0).$$

$$\widetilde{\boldsymbol{A}}=\left[\begin{array}{c|cc|ccccc} \boldsymbol{A} & d_2\boldsymbol{B} & d_3\boldsymbol{B} & \boldsymbol{B} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \hline \boldsymbol{0} & d_1 & -d_4 & \multicolumn{3}{c}{\multirow{2}{*}{$\boldsymbol{0}$}} \\ & d_4 & d_1 & & & & \\ \hline & & & 0 & 1 & \cdots & 0 \\ & & & 0 & 0 & \cdots & 0 \\ \boldsymbol{0} & \multicolumn{2}{c|}{\boldsymbol{0}} & \vdots & \vdots & \ddots & \vdots \\ & & & 0 & 0 & \cdots & 0 \end{array}\right], \widetilde{\boldsymbol{x}}(t)=\left[\begin{array}{c} \boldsymbol{x}(t) \\ \hline x_{n+1}(t) \\ x_{n+2}(t) \\ \hline x_{n+3}(t) \\ x_{n+4}(t) \\ \vdots \\ x_{n+m+3}(t) \end{array}\right], \widetilde{\boldsymbol{x}}(0)=\left[\begin{array}{c} \boldsymbol{x}(0) \\ \hline 1 \\ 0 \\ \hline c_0 \\ c_1 \\ \vdots \\ c_m m! \end{array}\right],$$

A função MATLAB `ss_augment()` é escrita para gerar o espaço de estados aumentado para um típico sinal de entrada:

```
function [Ga,Xa]=ss_augment(G,cc,dd,X)
G=ss(G); Aa=G.a; Ca=G.c; Xa=X; Ba=G.b; D=G.d;
if (length(dd)>0 & sum(abs(dd))>1e-5),
   if (abs(dd(4))>1e-5),
      Aa=[Aa dd(2)*Ba, dd(3)*Ba; ...
          zeros(2,length(Aa)), [dd(1),-dd(4); dd(4),dd(1)]];
      Ca=[Ca dd(2)*D dd(3)*D]; Xa=[Xa; 1; 0]; Ba=[Ba; 0; 0];
   else,
      Aa=[Aa dd(2)*B; zeros(1,length(Aa)) dd(1)];
      Ca=[Ca dd(2)*D]; Xa=[Xa; 1]; Ba=[B;0];
end, end
if (length(cc)>0 & sum(abs(cc))>1e-5), M=length(cc);
   Aa=[Aa Ba zeros(length(Aa),M-1); zeros(M-1,length(Aa)+1) ...
       eye(M-1); zeros(1,length(Aa)+M)];
   Ca=[Ca D zeros(1,M-1)]; Xa=[Xa; cc(1)]; ii=1;
   for i=2:M, ii=ii*i; Xa(length(Aa)+i)=cc(i)*ii;
end, end
Ga=ss(Aa,zeros(size(Ca')),Ca,D);
```

A sintaxe da função é `[`$\widehat{G}$`,`$\widehat{x_0}$`]=ss_augment(G,c,d,`x_0`)`, onde os vetores $c=[c_0,c_1,\ldots,c_m]$ e $d=[d_1,d_2,d_3,d_4]$ são utilizados para descrever a função de entrada $u(t)$ em (3.2). Os argumentos G e x_0 são o objeto e o vetor de estado inicial, respectivamente, onde as variáveis retornadas $\widehat{G}$ e $\widehat{x_0}$ são, respectivamente, o espaço de estados aumentado e seu vetor inicial. Uma vez que o sistema aumentado é estabelecido, as soluções analíticas para o sistema podem ser facilmente obtidas usando a *Symbolic Toolbox*.

Exemplo 3.11. Assuma que o espaço de estados é dado por

com os estados iniciais $x^T(0)=[0,1,1,2]$. Se o sinal de entrada é definido como $u(t)=2+2e^{-3t}\sin(2t)$, a função `ss_argument()` pode ser utilizada para construir o espaço de estados aumentado:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -19 & -16 & -16 & -19 \\ 21 & 16 & 17 & 19 \\ 20 & 17 & 16 & 20 \\ -20 & -16 & -16 & -19 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix} u(t), \\ y(t) = [2, 1, 0, 0]\, \boldsymbol{x}(t) \end{cases}$$

```
>> cc=[2]; dd=[-3,0,2,2]; x0=[0; 1; 1; 2];
   A=[-19,-16,-16,-19; 21,16,17,19; 20,17,16,20;
      -20,-16,-16,-19];
   B=[1; 0; 1; 2]; C=[2 1 0 0]; D=0; G=ss(A,B,C,D);
   [Ga,xx0]=ss_augment(G,cc,dd,x0); Ga.a, xx0'
```

O modelo aumentado é

$$\dot{\tilde{\boldsymbol{x}}}(t) = \left[\begin{array}{cccc:cc:c} -19 & -16 & -16 & -19 & 0 & 2 & 1 \\ 21 & 16 & 17 & 19 & 0 & 0 & 0 \\ 20 & 17 & 16 & 20 & 0 & 2 & 1 \\ -20 & -16 & -16 & -19 & 0 & 4 & 2 \\ \hdashline 0 & 0 & 0 & 0 & -3 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tilde{\boldsymbol{x}}(t), \quad \tilde{\boldsymbol{x}}(0) = \left[\begin{array}{c} 0 \\ 1 \\ 1 \\ 2 \\ \hdashline 1 \\ 0 \\ \hdashline 2 \end{array}\right].$$

Os seguintes comandos podem ser utilizados para encontrar a solução analítica do sistema:

```
>> syms t; y=Ga.c*expm(Ga.a*t)*xx0;
```

O sinal de saída do sistema é obtido como

$$y(t) = -54 + \frac{127}{4} t\mathrm{e}^{-t} + 57\mathrm{e}^{-3t} + \frac{119}{8}\mathrm{e}^{-t} + 4t^2\mathrm{e}^{-t} - \frac{135}{8}\mathrm{e}^{-3t}\cos 2t + \frac{77}{4}\mathrm{e}^{-3t}\sin 2t.$$

Método da transformada de Laplace

Consideremos a seguinte função de trasnferência:

$$G(s) = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_m s + b_{m+1}}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1} s + a_n}.$$

Para qualquer sinal de entrada $u(t)$ com $U(s)$ como sua transformada de Laplace, o sinal de saída pode ser obtido de $Y(s) = G(s)U(s)$. Assim, a fim de encontrar $y(t)$, a transformada inversa de Laplace é necessária, uma vez que $y(t) = \mathcal{L}^{-1}[Y(s)]$. A *Symbolic Toolbox* do MATLAB pode ser utilizado para realizar a transformada de Laplace de dados sinais de entrada e a transformada inversa pode ser utilizada para realizar a solução analítica do sistema.

Exemplo 3.12. Assuma que

$$G(s) = \frac{s^3 + 7s^2 + 3s + 4}{s^4 + 7s^3 + 17s^2 + 17s + 6}$$

é a função de transferência a ser analisada e que o sinal de entrada é dado por $u(t) = 2 + 2e^{-3t}\sin 2t$. A solução analítica para o sinal de saída pode ser realizada utilizando os comandos

```
>> syms s t;
   G=(s^3+7*s^2+3*s+4)/(s^4+7*s^3+17*s^2+17*s+6);
   u=2+2*exp(-3*t)*sin(2*t); U=laplace(u);
   y=ilaplace(G*U)
```

e a solução analítica pode ser escrita como

$$y(t) = \frac{4}{3} - \frac{31}{12}\mathrm{e}^{-3t} - \frac{23}{20}\mathrm{e}^{-3t}\cos 2t + \left(6 - \frac{21}{4}t\right)\mathrm{e}^{-t} - \frac{18}{5}\,\mathrm{e}^{-2t} - \frac{103}{40}\mathrm{e}^{-3t}\sin 2t.$$

3.2.2 Soluções Analíticas para Respostas em Tempo Discreto

Similar a abordagem do domínio s para solução analítica de sistemas contínuos, a transformada Z pode ser utilizada para sistemas discretos para calcular a resposta a um sinal de entrada $U(z)$. Então, a solução analítica o sistema $H(z)$ pode ser obtida pela resolução da transformada Z inversa, uma vez que $y(n) = \mathcal{Z}^{-1}[H(z)U(z)]$.

Exemplo 3.13. Assuma que

é a função de transferência discreta no tempo do sistema. Assuma também que o sinal de entrada é o degrau unitário. A solução analítica pode ser obtida usando os comandos

$$G(z) = \frac{(z - 1/3)}{(z - 1/2)(z - 1/4)(z + 1/5)}$$

```
>> syms z; u=sym(1); U=ztrans(sym(u));
   H=(z-1/3)/(z-1/2)/(z-1/4)/(z+1/5);
   y=iztrans(H*U)
```

$$y(n) = \frac{40}{27} - \frac{80}{81}\left(\frac{1}{4}\right)^n + \frac{800}{567}\left(-\frac{1}{5}\right)^n - \frac{40}{21}\left(\frac{1}{2}\right)^n.$$

e a solução analítica pode ser escrita como

Se o período de amostragem T é dado, a solução analítica pode ser reescrita como

$$y(kT) = \frac{40}{27} - \frac{80}{81}\left(\frac{1}{4}\right)^{kT} + \frac{800}{567}\left(-\frac{1}{5}\right)^{kT} - \frac{40}{21}\left(\frac{1}{2}\right)^{kT}.$$

3.3 Simulação Numérica de Sistemas Lineares

As soluções analiticas de sistemas lineares foram estudadas nas seções anteriores. Em aplicações reais, é preferível ter soluções numéricas e, baseado nos resultados, as respostas no domínio do tempo podem ser plotadas. Visualização gráfica das respostas do sistema é usualmente mais simples e informativa para engenheiros de controle.

Nesta seção, as técnicas de solução numérica para sistemas lineares serão apresentadas com foco em algumas respostas comuns, tais como resposta ao degrau, resposta ao impulso e resposta no domínio do tempo para sinais de entrada arbitrários.

3.3.1 Resposta ao Degrau de Sistemas Lineares

Sinais de entrada em degrau e suas respostas são comumente utilizadas em análise e desenvolvimento de sistemas de controle. A resposta típica ao degrau em um sistema de segunda ordem são estudadas e especificações são dadas. Então, o cálculo baseado em MATLAB de respostas ao degrau é possível.

Análise de sistemas de segunda ordem

Em disciplinas de controle clássico, sistemas de segunda ordem são frequentemente utilizados como exemplo, onde a maioria das propriedades de sistemas de controle linear são

ilustradas.

Teorema 3.4. A resposta ao degrau em malha fechada de um sistema de segunda ordem

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

pode ser obtida facilmente considerando os quatro casos a seguir:

1. Quando $\zeta = 0$, a resposta ao degrau é $y(t) = 1 - \cos(\omega_n t)$.
2. Quando $0 < \zeta < 1$, a resposta ao degrau é

$$y(t) = 1 - e^{-\zeta\omega_n t}\frac{1}{\sqrt{1-\zeta^2}}\sin(\omega_d t + \theta),$$

   onde $\theta = \tan^{-1}\sqrt{1-\zeta^2}/\zeta$ e $\omega_n\sqrt{1-\zeta^2}$.
3. Quando $\zeta = 1$, a resposta ao degrau é $y(t) = 1 - (1+\omega_n t)e^{-\omega_n t}$.
4. Quando $\zeta > 1$, a resposta ao degrau é

$$y(t) = 1 - \frac{\omega_n}{2\sqrt{\zeta^2-1}}\left(\frac{e^{\lambda_1 t}}{\lambda_1} - \frac{e^{\lambda_1 t}}{\lambda_1}\right),$$

   onde $\lambda_1 = -\zeta - \sqrt{\zeta^2-1}$, $\lambda_2 = -\zeta + \sqrt{\zeta^2-1}$.

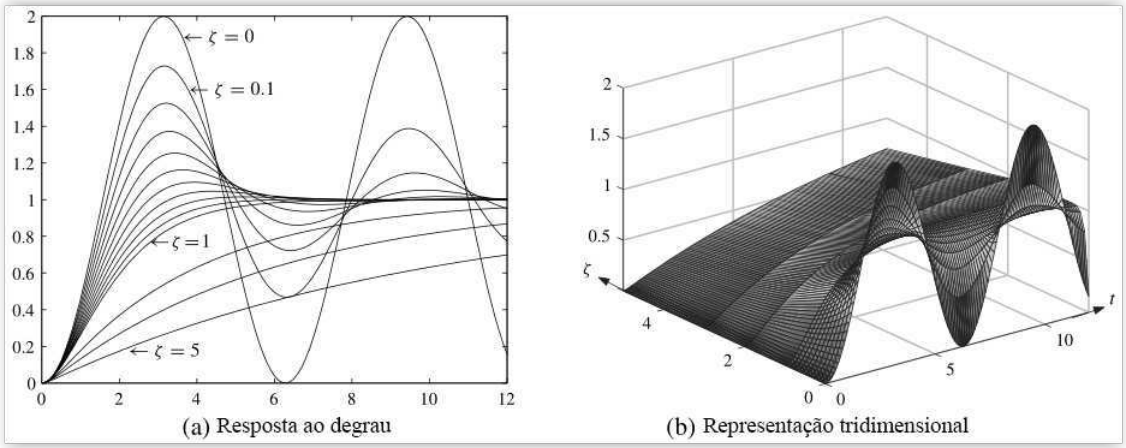


Figura 3.5: Respostas ao degrau de sistemas de segunda ordem

```
>> syms z s, syms wn positive
   y=ilaplace(wn^2/(s*(s^2+2*z*wn*s+wn^2)))
```

Exemplo 3.14. Com o uso da poderosa *Symbolic Toolbox*, as soluções analíticas para um sistema de segunda ordem podem ser facilmente obtidas:

Com algumas simplificações óbvias, segue o resultado

$$y(t) = 1 - \omega_n e^{-\zeta\omega_n t}\left[\frac{\cosh(\omega_d t)}{\omega_n} + \frac{\zeta\sinh(\omega_d t)}{\omega_d}\right], \tag{3.3}$$

onde $\omega_d = \sqrt{\zeta^2 - 1}\omega_n$. É possível perceber que o formato dos novos resultados são muito mais concisos que aqueles dados no Teorema (3.4) se variáveis complexas são permitidas. A única restrição em (3.3) é que $\zeta \neq 1$; caso contrário, zero pode ser utilizado no denominador. É possível evitar esse caso particular definindo $\zeta = 1 + \varepsilon$, onde ε é um número muito pequeno.

A resosta ao degrau do sistema pode ser calculada facilmente como mostrado na Figura 3.5(a). A versão tridimensional é mostrada na Figura 3.5(b).

Os seguintes comandos MATLAB mostram as respostas ao degrau:

```
>> wn=1; yy=[]; t=0:.1:12; zet=[0:0.1:0.9, 1+eps,2,3,5];
   for z=zet
      wd=sqrt(z^2-1)*wn;
      y=1-wn*exp(-z*wn*t).*[cosh(wd*t)/wn+z*sinh(wd*t)/wd];
      yy=[yy; y];
   end
   plot(t,yy), figure, surf(t,zet,yy)
```

Pode ser visto que quando $\zeta = 0$, a saída é uma oscilação não-amortecida. Quando ζ é menor que 1, existe uma oscilação amortecida e, com o incremento no valor de ζ, a oscilação e o *overshoot* tendem a ser menores. Quando o valor de ζ é maior que ou igual a 1, não existe oscilação no sinal de saída. O comportamento do sinal de saída *versus* as mudanças no valor de ζ podem ser melhor observadas na superfície tridimensional na Figura 3.5(b).

Especificações quantitativas em respostas ao degrau

Para curvas típicas de resposta do degrau, várias especificações quantitativas úteis são definidas e mostradas na Figura 3.6. Detalhes dessas especificações comumente utilizadas estão resumidas abaixo:

1. *O valor estacionário* $y(\infty)$: O valor estacionário do sistema sob a resposta ao degrau é a saída quando $t \to \infty$. Para a função de transferência, utilizando o teorema do valor final

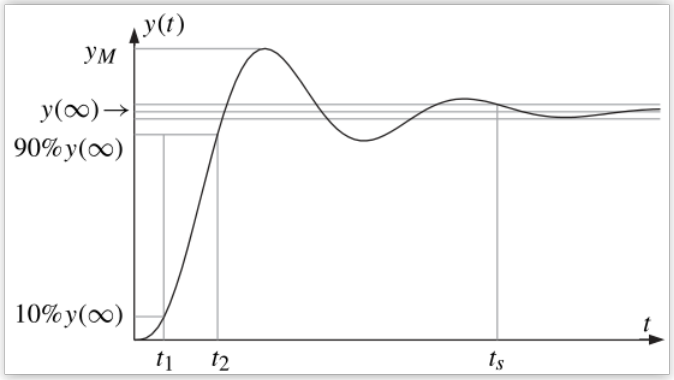


Figura 3.6: Especificações de resposta ao degrau típica

da transformada de Laplace, o valor estacionário do sistema pode ser facilmente obtido de

$$y(\infty) = \lim_{s\to 0} sG(s)\frac{1}{s} = G(0) = \frac{b_m}{a_n}.$$

Se o sistema é dado com o espaço de estados (**A,B,C,D**), o valor estacionário do sistema pode ser obtido de

$$y(\infty) = \lim_{s\to 0} sG(s)\frac{1}{s} = -\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B} + \boldsymbol{D}.$$

O valor estacionário do sistema G pode ser calculado utilizado a função

```
K=dcgain(G)
```

fornecido pela *Control Systems Toolbox*.

2. *O tempo de subida* t_r: O tempo de subida é definido como $t_r = t_2 - t_1$ com t_2 e t_1, respectivamente, o tempo quando $y(t)$ alcança 90% e 10% do valor estacionário

3. *O tempo de acomodação* t_s: Quando o sinal de saída $y(t)$ é mantido dentro de um intervalo de $[y(\infty) - \Delta y, y(\infty) + \Delta y]$, o instante em que $y(t)$ entra no intervalo é chamado de tempo de acomodação. De acordo com diferentes definições, Δy pode ser definido como 2% ou 5% do valor estacionário $y(\infty)$.

4. *O* overshoot M_p *e o valor de pico* y_M: M_p é também conhecido como o percentual de *overshoot* e é definido como

$$M_p = \frac{y_M - y(\infty)}{y(\infty)} \times 100\%$$

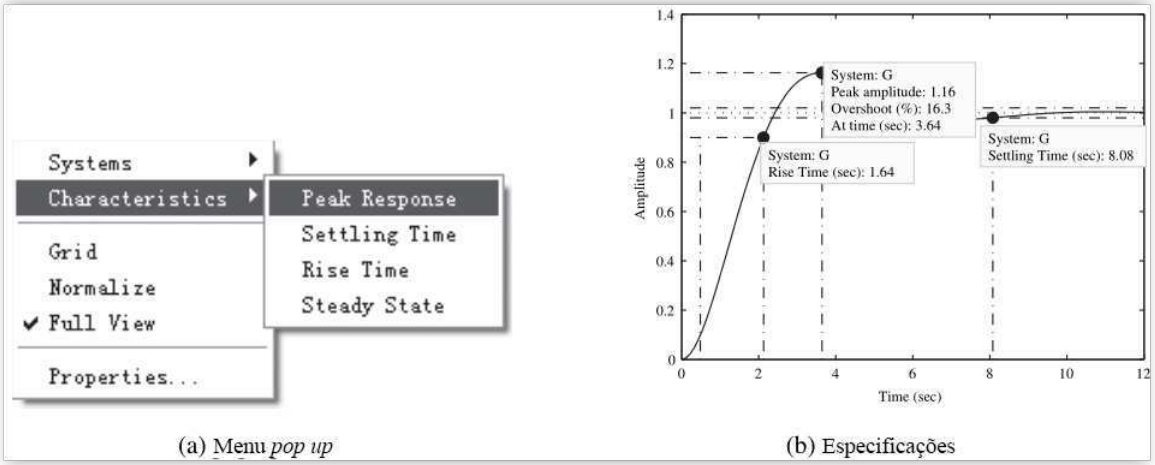


Figura 3.7: Resposta ao degrau com especificações

Em projeto de sistemas de controle, é esperado projetar um sistema com baixo tempo de subida e baixo tempo de acomodação e uma pequena porcentagem ou nenhum *overshoot*.

Exemplo 3.15. Considere o sistema de segunda ordem com $\zeta = 0.5$ e $\omega_n = 1$ rad/s. A função `step()`, que será descrita posteriormente, provida em *Control Systems Toolbox*, pode ser utilizada diretamente para desenhar a curva de resposta ao degrau

```
>> z=0.5; wn=1; G=tf(wn^2,[1,2*z*wn,wn^2]); step(G).
```

Clique com o botão direito na janela que aparece para ver o menu *pop-up* mostrado na Figura 3.7(a). É possível selecionar diferentes especificações a partir do menu e as correspondentes especificações serão sobrepostas na curva de resposta ao degrau, como mostrado na Figura 3.7(b).

Cálculo de resposta ao degrau com MATLAB

A resposta ao degrau de sistemas lineares pode ser calculado e plotado utilizando a função `step()`, que pode ser chamada com uma variedade de sintaxes:

```
step(G)
[y,t]=step(G)
[y,t]=step(G,tf)
y=step(G,t)
[y,t,x]=step(G)
```

Na chamada da função, *G* é um modelo LIT, que pode ser uma função de transferência, um espaço de estados ou um modelo polo-zero-ganho. Esta função aplica-se a sistemas contínuos e sistemas discretos. Ela pode ser também utilizada para sistemas SISO e MIMO e sistemas com e sem atraso. Assim, a função fornece um modo unificado de encontrar a resposta ao degrau de sistemas lineares. Se nenhum argumento é retornado na chamada da função, então a resposta ao degrau é plotada automaticamente. Se algo é retornado, não haverá plotagem da resposta. Os dados pode ser plotados posteriormente utilizando a função `plot()`. Entretanto, as curvas planas de `plot()` podem perder propriedades úteis, tais como o menu *pop-up* mostrado na Figura 3.7(a).

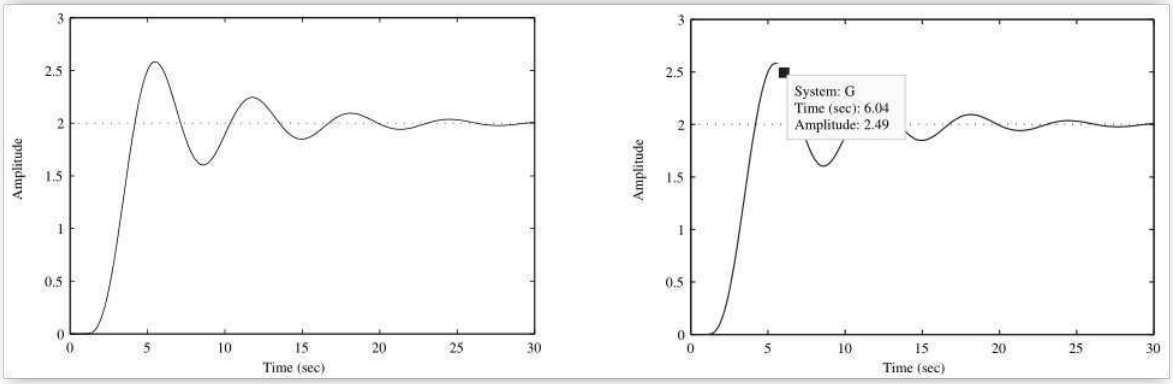


Figura 3.8: Resposta ao degrau de um sistema contínuo com atraso

Exemplo 3.16. Se o modelo contínuo

$$G(s) = \frac{10s + 20}{10s^4 + 23s^3 + 26s^2 + 23s + 10} e^{-s}$$

é uma função de transferência que contém um atraso no tempo, os seguintes comandos podem ser utilizados para plotar a resposta no tempo como mostrado na Figura 3.8(a).

```
>> G=tf([10 20],[10 23 26 23 10],'ioDelay',1); % model input
   step(G,30); % step response with terminate time of 30 sec.
```

Na plotagem da resposta ao degrau, se for clicado em algum ponto na curva, sua magnitude e tempo serão mostrados, como mostra a Figura 3.8(b). O *overshoot*, tempo de acomodação e outras especificações poem ser facilmente mostradas na curva, utilizando os métodos mostrados anteriormente.

Exemplo 3.17. Se um sistema é dado por

com período de amostragem $T = 0.01, 0.1, 0.5, 1.2$ segundos, respectivamente, os seguintes comandos podem ser utilizados para obter os modelos de tempo discreto para diferentes intervalos de amostragem. As respostas ao degrau são mostradas na Figura 3.9. É possível mostrar que as informações do sistema original podem ser perdidas se o intervalo de amostragem selecionado for grande demais.

$$G(s) = \frac{1}{s^2 + 0.2s + 1} e^{-s},$$

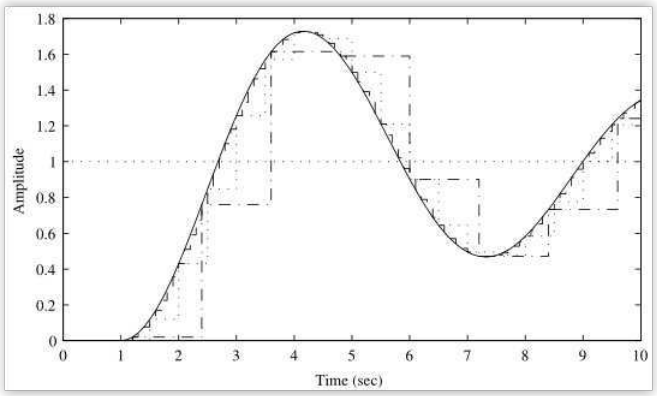


Figura 3.9: Comparações de respostas ao degrau de sistemas discretizados

Os modelos em tempo discreto obtidos são, respectivamente,

Deve-se notar que a curva de resposta ao degrau do sistema em tempo discreto é automaticamente plotado no formato escada.

3.3.2 Resposta ao Impulso de Sistemas Lineares

A resposta ao impulso de um sistema pode ser plotada facilmente utilizando a função `impulse()` fornecida na *Control Systems Toolbox*. A sintaxe da função é exatamente a mesma da função `step()` mostrada anteriormente.

Exemplo 3.18. Considere novamente o sistema estudado no Exemplo 3.16. A resposta ao impulso do sistema pode ser obtido como mostrado na Figura 3.11:

3.3.3 Respostas no Tempo a Entradas Arbitrárias

Nas discussões anteriores, dois tipos de sinais de entrada foram estudados. Aqui, dois outros tipos de sinais serão estudados.

Se a transformada de Laplace $R(s)$ do sinal de entrada pode ser escrito como uma função racional, a saída do sistema pode ser expressa como $Y(s) = G(s)R(s)$, que também é racional. Assim, o tempo de resposta sob $R(s)$ pode ser equivalentemente calculado com a função

```
>> G=tf(1,[1 0.2 1],'ioDelay',1); G1=c2d(G,0.01,'zoh');
   G2=c2d(G,0.1); G3=c2d(G,0.5); G4=c2d(G,1.2);
   step(G,'-',G2,'--',G3,':',G4,'-.',10)
```

$$G_1(z) = \frac{4.997\times10^{-5}z+4.993\times10^{-5}}{z^2-1.998z+0.998}z^{-100}, \qquad G_2(z) = \frac{0.004963z+0.00493}{z^2-1.97z+0.9802}z^{-10}$$

$$G_3(z) = \frac{0.1185z+0.1145}{z^2-1.672z+0.9048}z^{-2}, \qquad G_4(z) = \frac{0.01967z^2+0.7277z+0.3865}{z^3-0.6527z^2+0.7866z}.$$

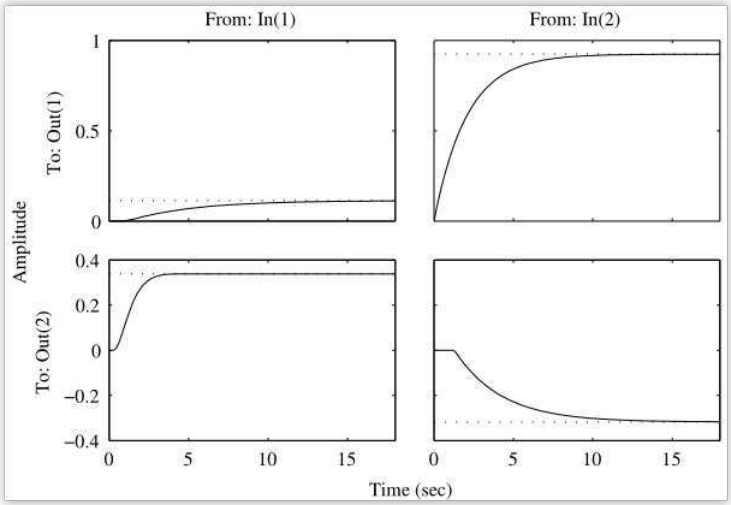


Figura 3.10: Resposta ao degrau de um sistema multivariável

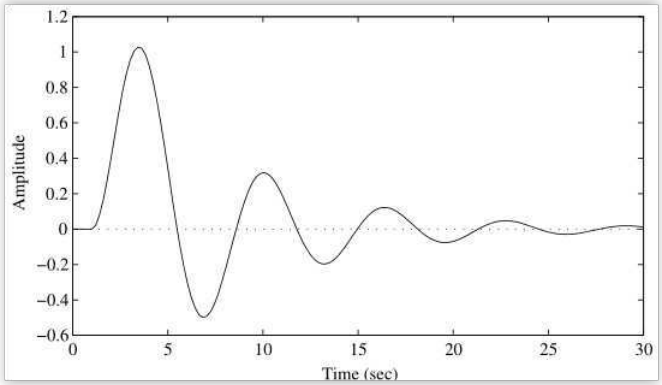


Figura 3.11: Resposta ao impulso do sistema

```
>> G=tf([10 20],[10 23 26 23 10],'ioDelay',1); impulse(G, 30);
```

`impulse()` se $Y(s)$ é assumido como a função de transferência do sistema.

Exemplo 3.19. Considere novamente

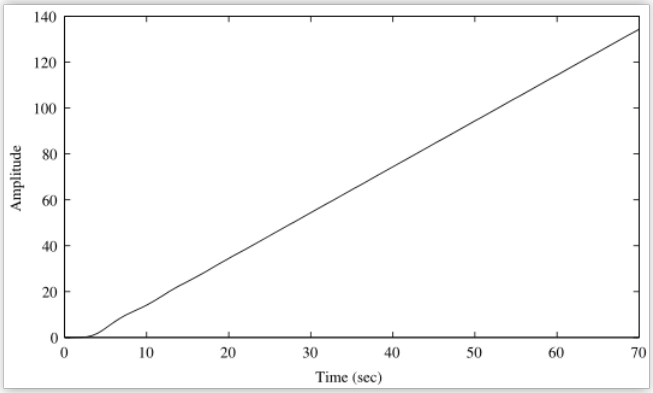


Figura 3.12: Resposta à rampa

$$G(s) = \frac{10s + 20}{10s^4 + 23s^3 + 26s^2 + 23s + 10} e^{-s}.$$

A resposta à rampa do sistema pode ser obtida com a ajuda da função `impulse()`.

Sabe-se que a transformada de Laplace de uma rampa é a função $1/s^2$; então, a resposta à rampa do sistema pode ser calculada como a resposta ao degrau do sistema $G(s)/s$ ou a resposta ao impulso do sistema $G(s)/s^2$. Os seguintes comandos podem ser então utilizados para calcular a resposta à rampa do sistema, como mostrado na Figura 3.12.

```
>> G=tf([10 20],[10 23 26 23 10],'ioDelay',1);
   s=tf('s'); step(G/s);  % or use impulse(G/s^2)
```

Se o sinal de entrada não pode ser expresso por equações matemáticas ou a transformada de Laplace não pode ser a uma função racional, os métodos acima não pode ser utilizados. Nesse caso, a função `lsim()` pode ser utilizada para calcular a resposta no domínio do tempo do sistema. A sintaxe da função `lsim()` é similar a da função `step()`. A diferença é que o vetor de entrada *u* deve ser utilizado da seguinte maneira: `lsim(G,u,t)`.

3.4 Local das Raízes de Sistemas Lineares

Assuma que o sistema de controle realimentado é estabelecido por uma realimentação negativa, um ganho K e um modelo $G(s)$ de malha aberta. Para cada valor de K, um conjunto de polos de malha fechada podem ser encontrados pela resolução da equação $1 + KG(s) = 0$. Com contínuas mudanças no ganho K, as trajetórias dos polos do sistema em malha fechada podem ser construídas. As trajetórias dos polos do sistema em malha fechada são chamadas de local das raízes do sistema. Deve-se notar que o modelo em malha aberta $G(s)$ deve ser utilizado para

plotar o local das raízes e o local das raízes pode ser utilizado para descreber as posições dos polos do sistema em malha fechada.

A função MATLAB `rlocus()` é fornecida na *Control Systems Toolbox* para plotar o local das raízes de um dado sistema. A função pode ser chamada em um dos seguintes modos:

```
rlocus(G)          % automatic draw of the root locus
rlocus(G, [k_min, k_max])          % root locus over the gain range
rlocus(G, K)          % root locus for a given gain vector K
[R, K]=rlocus(G)          % evaluate the closed-loop pole positions R
rlocus(G1,'-',G2,'-.b',G3,':r')          % root locus for several models
```

Deve-se notar que esta função aplica-se em ambas os tipos de sistemas, contínuo e discreto. Somente modelos SISO LIT pode ser processados nesta função. Ela também pode ser utilizada para plotagem do local das raízes de funções de transferência discretas no tempo.

No local das raízes do sistema, é possível clicar com o botão esquero em qualquer ponto para mostrar o ganho, a posição do polo, a taxa de amortecimento e o *overshoot* do sistema para aquele valor de K. É possível facilmente encontrar os valores do ganho de malha aberta K para as quais o sistema em malha fechada é estável. O comando `grid` pode ser utilizado para sobrepor as linhas de iso-amortecimento e isofrequência do sistema. Estas linhas fornecem informações úteis em projeto de sistemas de controle.

Exemplo 3.20. Seja

$$G(s)=\frac{s^2+4s+8}{s^5+18s^4+120.3s^3+357.5s^2+478.5s+306}$$

o modelo em malha aberta do sistema sob investigação. Utilizando os seguintes comandos MATLAB, o local das raízes do sistema pode ser facilmente e precisamente plotado, como mostra a Figura 3.13(a). Ao clicar com o botão esquerdo no ponto de intersecção com o eixo imaginário, a informação sobre o ponto crítico é mostrado como na Figura 3.13(b), do qual é imediatamente visto que o ganho crítico é 772. Conclui-se que quando o ganho $K > 772$, então o sistema em malha fechada é instável.

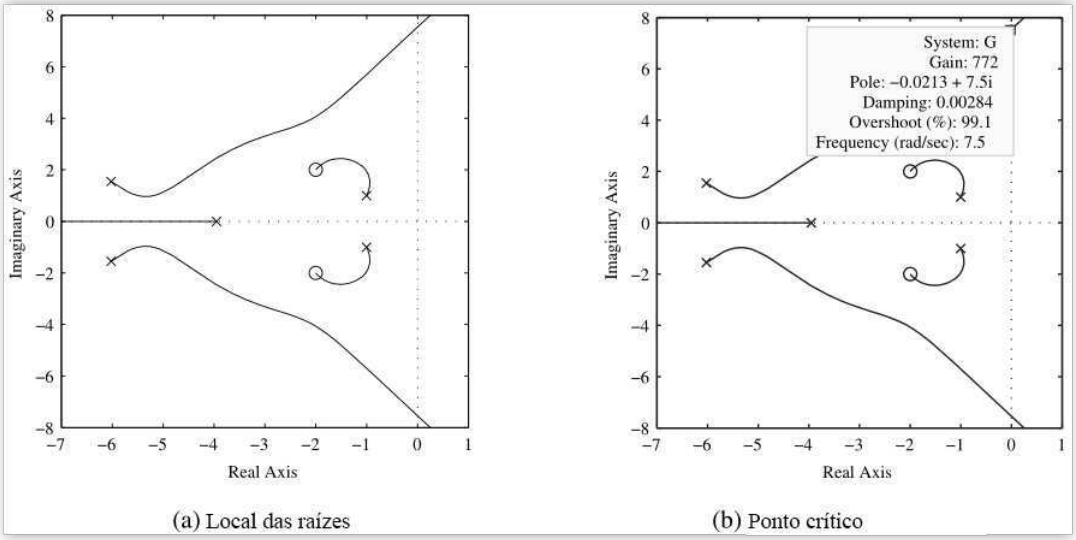


Figura 3.13: Análise do local das raízes do sistema e sua inversa

```
>> num=[1 4 8]; den=[1,18,120.3,357.5,478.5,306];
   G=tf(num,den); rlocus(G)
```

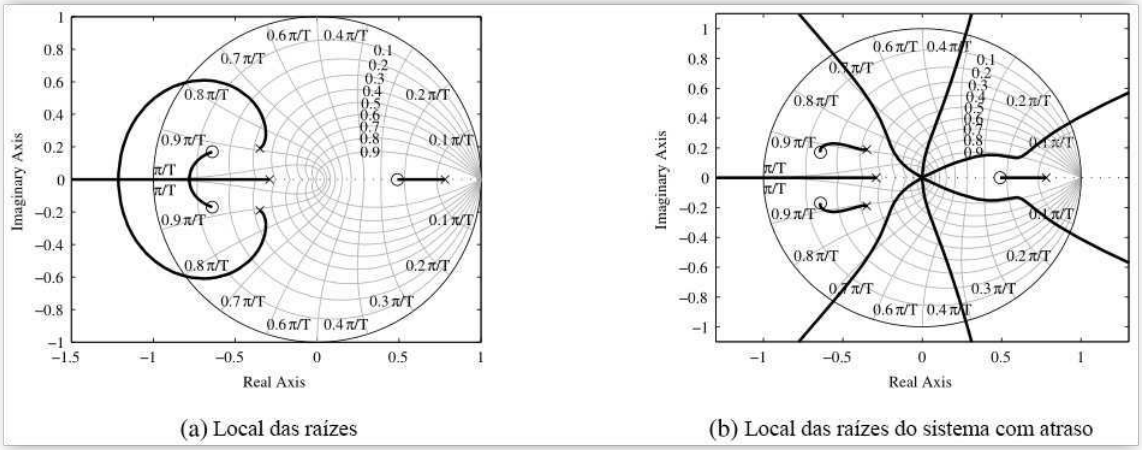


Figura 3.14: Local das raízes de um sistema discreto no tempo

Exemplo 3.21. Se um modelo em malha aberta discreto no tempo é dado por

$$G(z) = \frac{0.52(z-0.49)(z^2+1.28z+0.4385)}{(z-0.78)(z+0.29)(z^2+0.7z+0.1586)}$$

com período de amostragem de $T = 0.1$s, os seguintes comandos podem ser utilizados para plotar o local das raízes do sistema, como mostrado na Figura 3.14(a). Clicando em pontos relevantes, é possível ver que o ganho crítico é $K = 2.83$.

Se existe um termo de atraso z^{-6} no sistema original, o local das raízes do sistema com

```
>> z=tf('z','Ts',0.1);
   G=0.52*(z-0.49)*(z^2+1.28*z+0.4385)/(z+0.29)/(z^2+0.7*z+0.1586);
   rlocus(G), grid
```

atraso pode ser replotado, como mostrado na Figura 3.14(b):

```
>> G.ioDelay=6; rlocus(G), grid
```

É possível encontrar que o ganho crítico é reduzido para 1.16. Sendo assim, um termo de atraso em um modelo em tempo discreto reduz o ganho crítico do sistema.

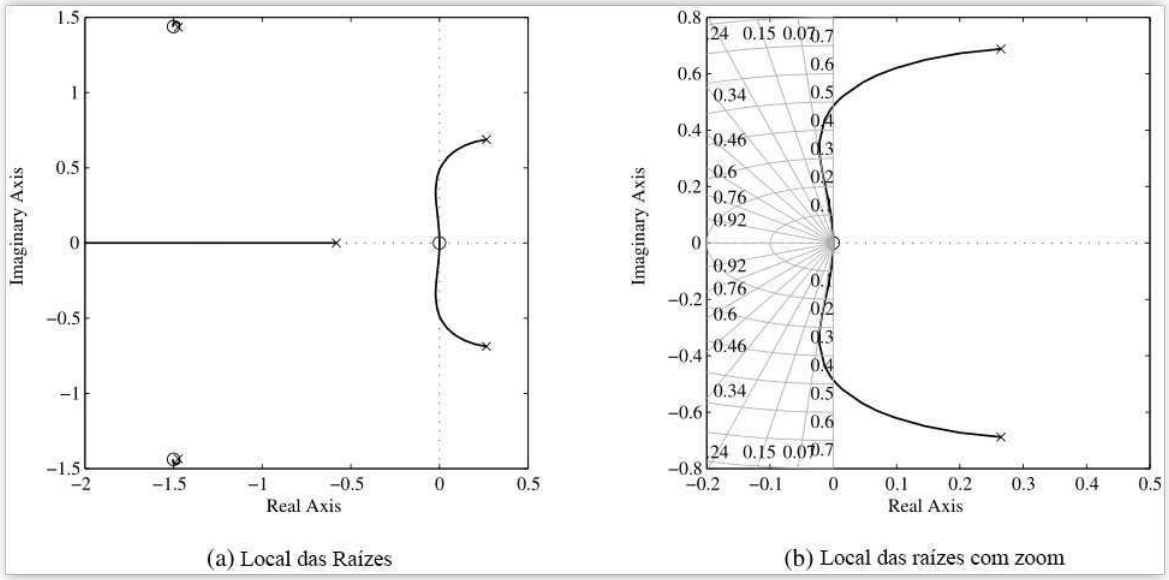


Figura 3.15: Local das raízes de acordo com a varável a

Exemplo 3.22. Para o modelo em malha aberta

$$G(s) = \frac{0.3(s+2)(s^2+2.1s+2.23)}{s^2(s^2+3s+4.32)(s+a)},$$

se é desejável encontrar o local das raízes de acordo com a variável a, a equação característica $1+G(s)=0$ pode ser reescrita como

$$a(s^4+3s^3+4.32s^2)+(s^5+3s^4+4.62s^3+1.23s^2+1.929s+1.338)=0$$

de onde pode ser visto que

$$1+a\frac{s^4+3s^3+4.32s^2}{s^5+3s^4+4.62s^3+1.23s^2+1.929s+1.338}=0.$$

Seja

$$\widehat{G}(s)=\frac{s^4+3s^3+4.32s^2}{s^5+3s^4+4.62s^3+1.23s^2+1.929s+1.338}.$$

A equação característica pode ser escrita como $1+a\widehat{G}(s)=0$. O local das raízes de acordo com a variável a pode ser plotado para o modelo $\widehat{G}(s)$. Os comandos abaixo podem ser realizados para se obter o local das raízes como mostrado na Figura 3.15(a). A Figura 3.15(b) é a versão com zoom da primeira.

```
>> G1=tf([1,3,4.32,0,0],[1,3,4.62,1.23,1.929,1.338]); rlocus(G1)
```

O local das raízes pode ser utilizado em projeto de controle para selecionar um valor apropriado para o ganho K. Se existe um par de polos complexos dominantes no local das raízes, o qual tem relativamente baixo amortecimento para um valor específico de K, então, selecionar este valor de K deve ser o mais apropriado. Assume-se que se os polos complexos dominantes e os efeitos de quaisquer zeros são ignorados, então a resposta em malha fechada resultante será aproximadamente a mesma de um sistema de segunda ordem com esses mesmos polos.

3.5 Análise em Domínio da Frequência de Sistemas Lineares

Métodos de análise no domínio da frquência compõem uma classe de métodos extremamente importantes em análise e projeto de sistemas de controle. Em 1932, Nyquist apresentou um método gráfico que pode ser utilizado para avaliar a estabilidade de um sistema de controle. Em poucos anos, a análise e projeto utilizando domínio no frequência já havia se popularizado. Esta se revelou como uma abordagem muito útil.

3.5.1 Gráficos em Domínio da Frequência em MATLAB

Para uma função de transferência linear $G(s)$, se a variável no domínio da frequência $\mathrm{j}\omega$ é utilizada para substituir a variável complexa s, então $G(\mathrm{j}\omega)$ pode ser considerado como o "ganho complexo"do sistema. Há muitos modos diferentes de descrever a quantidade complexa $G(\mathrm{j}\omega)$. Baseado nessas descrições, diferentes métodos no domínio da frequência podem ser estabelecidos, como a seguir.

1. *Representação das partes real e imaginária*: O ganho complexo pode ser representado como parte real e parte imaginária, tal como

$$G(\mathrm{j}\omega) = P(\omega) + \mathrm{j}Q(\omega).$$

É possível ver que $P(\omega)$ e $Q(\omega)$ são funções da frequência ω. Se o eixo horizontal é utilizado para representar a parte real e o eixo verticial para a parte imaginária, a trajetória do ganho complexo $G(\mathrm{j}\omega)$ é chamada de Diagrama de Nyquist.

A função MATLAB `nyquist()` fornecida na *Control Systems Toolbox* pode ser utilizada para plotar o Diagrama de Nyquist do modelo LIT G. As sintaxes da função são Um

```
nyquist(G)          % automatic drawing of Nyquist plot
nyquist(G,{ωm,ωM})  % draw Nyquist plot over range (ωm, ωM)
nyquist(G, ω)       % draw Nyquist plot over frequency vector ω
[R, I, ω]=nyquist(G)  % Nyquist response data evaluation
nyquist(G1,'-',G2,'-.b',G3,':r')  % several systems
```

clique em algum ponto do Diagrama mostra informações sobre a frequência juntamente com o valor do ganho complexo. Esta facilidade fornece uma ferramente extremamente útil em análise e projeto de sistemas lineares. O comando `grid` pode ser utilizado para sobrepor os circulos iso-M no topo da curva de Nyquist.

2. *Representação de magnitude e fase*: A quantidade complexa $G(\mathrm{j}\omega)$ pode ser expressa na forma de magnitude e fase, isto é, Assim, a frequência ω pode ser utilizada como o eixo

$$G(\mathrm{j}\omega) = A(\omega)\mathrm{e}^{-\mathrm{j}\phi(\omega)}.$$

horizontal e magnitude $A(\omega)$ e fase $\phi(\omega)$ podem ser, separadamente, o eixo vertical. Um novo conjunto de diagramas pode ser construído. Se a frequência for plotada em escala logarítmica, a magnitude M for plotada em decibéis (dB), isto é, $M(\omega) = 20\log[A(\omega)]$ e a fase for plotada em graus, o diagrama é chamado de Diagrama de Bode.

A função `bode()` é fornecida na *Control Systems Toolbox* e pode ser utilizada para plotar o Diagrama de Bode do sistema linear G. As sintaxes da função são

```
bode(G)          % automatic draw of the Bode diagram
bode(G,{ω_m, ω_M})          % draw Bode diagram over range (ω_m, ω_M)
bode(G,ω)          % draw Bode diagram over frequency vector ω
[A,φ,ω]=bode(G)          % Bode diagram data evaluation
bode(G_1,'-',G_2,'-.b',G_3,':r')          % several systems
```

3. *Representação em única plotagemde magnitude e fase*: A representação de magnitude e fase do ganho complexo é novamente utilizada. Selecionando magnitude e fase como eixos vertical e horizontal, respectivamente, encontraremos o Diagrama de Nichols.

   A função `nichols()` fornecida na *Control Systems Toolbox* pode ser utilizada para plotar o Diagrama de Nichols para o dado sistema G. O comando `grid` pode ser utilizado para sobrepor as constantes M e N e contornar o gráfico.

   Para sistemas discretos $H(z)$, é possível substituir $z = \mathrm{e}^{\mathrm{j}\omega T}$ na função de transferência, de modo que a relação entre a magnitude complexa $\widehat{H}(\mathrm{j}\omega)$ e a frequência possa ser estabelecida. As funções mencionadas acima `nyquist()`, `bode()` e `nichols` podem ser utilizadas em modelos discretos no tempo diretamente.

3.5.2 Análise de Estabilidade usando Métodos em Domínio da Frequência

O Diagrama de Nyquist, frequentemente desenhado para modelos em malha aberta, pode ser utilizado para inferir o comportamento em malha fechada em termos de estabilidade e, inclusive, resposta no domínio do tempo. A estabilidade pode ser concluída utilizando o conhecido Teorema de Nyquist.

Teorema 3.5. Para sistemas em malha fechada com função de malha aberta $G(s)$ serem estáveis, o Diagrama de Nyquist de $G(s)$ deve circular o ponto $(-1,\mathrm{j}0)$ no sentido anti-horário tantas vezes quanto o número de polos de $G(s)$ que estão localizados no semiplano direito do plano s.

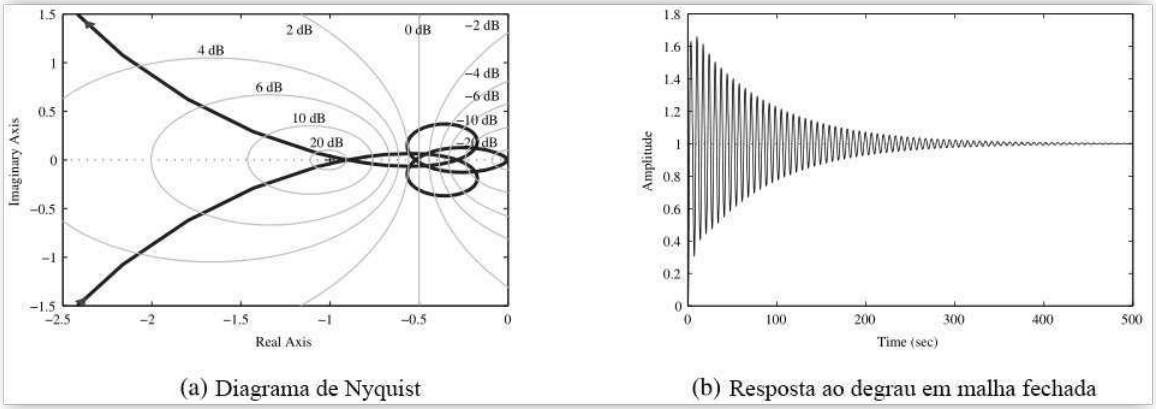
O Teorema de Nyquist pode ser interpretado nos dois casos a seguir:

1. Para um modelo em malha aberta estável $G(s)$, o sistema em malha fechada é estável se, e somente se, o Diagrama de Nyquist de $G(s)$ não circular o ponto $(-1,\mathrm{j}0)$. Se o Diagrama de Nyquist circula o ponto $(-1,\mathrm{j}0)$ p vezes no sentido horário, então existem p polos de malha fechada instáveis.

2. Para um modelo em malha aberta instável de $G(s)$ com p modos instáveis, o sistema em malha fechada é estável se, e somente se, o Diagrama de Nyquist de $G(s)$ circula o ponto $(-1,\mathrm{j}0)$ p vezes no sentido anti-horário. Se o ponto $(-1,\mathrm{j}0)$ é circulado q vezes no sentido anti-horário, então existem $q-p$ polos instáveis de malha fechada.

Exemplo 3.23. Considere o modelo em malha aberta O modelo pode ser facilmente colocado

$$G(s)=\frac{2.7778(s^2+0.192s+1.92)}{s(s+1)^2(s^2+0.384s+2.56)}.$$



(a) Diagrama de Nyquist (b) Resposta ao degrau em malha fechada

Figura 3.16: Análise do sistema

no MATLAB através dos seguintes comandos. O Diagrama de Nyquist do sistema pode ser plotado como mostrado na Figura 3.16(a): Pode ser visto que, embora a trajetória seja bastante complicada, o ponto $(-1,\mathrm{j}0)$ não é circulado nenhuma vez. Já que não existem polos instáveis, é possível mostrar, por meio do uso do Teorema de Nyquist, o sistema em malha

```
>> s=tf('s');
   G=2.7778*(s^2+0.192*s+1.92)/(s*(s+1)^2*(s^2+0.384*s+2.56));
   nyquist(G); axis([-2.5,0,-1.5,1.5]); grid
   figure; step(feedback(G,1))
```

fechada é estável. A resposta ao degrau do sistema em malha fechada é obtida como mostrado na Figura 3.16(b).

Embora o sistema em malha fechada seja estável, a resposta ao degrau possui uma oscilação forte, então o desempenho do sistema provavelmente não será satisfatório. Neste caso, um controlador será necessário para melhorar o desempenho.

3.5.3 Margens de Ganho e Fase de um Sistema

É possível ver através dos exemplos anteriores que, embora a estabilidade do sistema seja extremamente importante, não é o único fator importante na descrição do comportamento de sistemas. Margens de resposta em frequência são indicadores efetivos no endereçamento de estabilidade relativa e problemas de desempenho.

Nas Figuras 3.17(a) e (b), os esboços das margens de ganho e fase são ilustrados, respectivamente, nos Diagramas de Nyquist e Nichols. As margens de ganho e fase podem ser também ilustradas em Diagrama de Bode.

Se para um sistema com uma função de transferência de malha aberta estável o Diagrama de Nyquist intercepta o eixo real negativo na frequência ω_{cg}, a margem de ganho é definida como o inverno do ganho, isto é, $G_m = 1/A(\omega_{cg})$, frequentemente expressa em dBs. Se o Diagrama de Nyquist intercepta o círculo unitário na frequência ω_{cp}, a margem de fase é definida como $\gamma = \phi(\omega_{cp}) - 180^\circ$.

Pode ser visto que, normalmente, quanto maior o valor da margem de ganho G_m, mais rápida é a resposta ao degrau. Se $G_m < 1$, o sistema em malha fechada é instvel. De maneira similar, se a resposta em frequência em malha aberta é relativamente suave na região das margens de ganho e fase, quando maior a margem de fase, menor será o *overshoot* na resposta ao degrau em malha fechada. Entretanto, se $\gamma < 0$, o sistema em malha fechada é instável. Os seguites casos especiais devem ser também considerados.

1. Se não existe instersecção entre a curva no Diagrama de Nyquist com o semi-eixo real negativo, a margem de ganho é infinita

2. Se a curva no Diagrama de Nyquist intercepta muitas vezes o semi-eixo real negativo

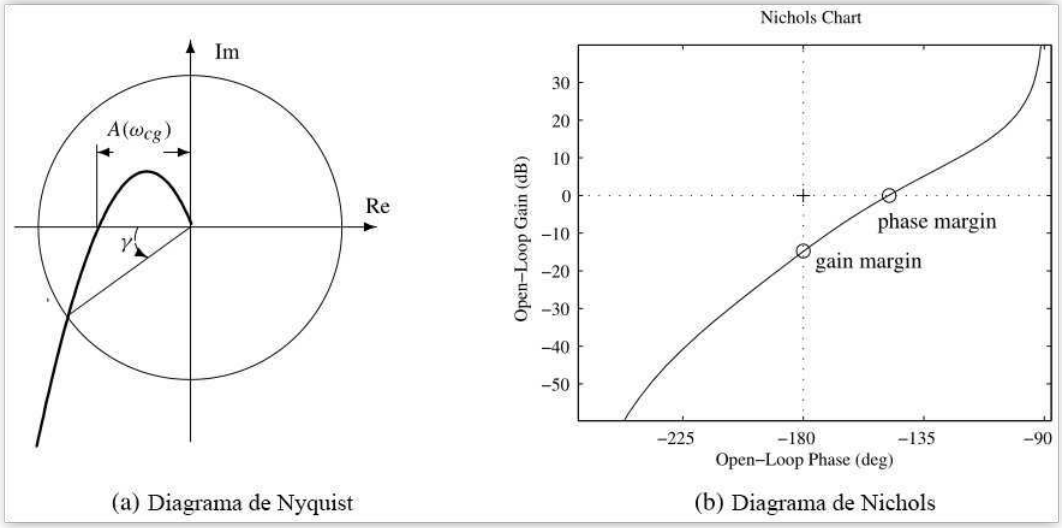


Figura 3.17: Representações gráficas das margens de ganho e fase

entre $(-1,\text{j}0)$ e $(0,\text{j}0)$, o ponto mais próximo de $(-1,\text{j}0)$ pode ser considerado como o ponto da margem de ganho.

3. Se não existe intersecção entre a curva no Diagrama de Nyquist com o cículo unitário, a margem de fase é infinita.

4. Se a curva no Diagrama de Nyquist intercepta muitas vezes o círculo unitário no terceiro quadrante, o ponto mais próximo do semi-eixo real negativo pode ser considerado como o ponto da margem de fase.

A função `margin()` é fornecida na *Control Systems Toolbox* para calcular as margens de ganho e fase. A sintaxe da funação é Como resultado, se uma margem é infinita, o valor

```
[Gm,γ,ωcg,ωcp]=margin(G);
```

retornado será `Inf`, enquanto a frequência correspondente será `NaN`.

Exemplo 3.24. Considere novamente o modelo em malha aberta no exemplo anterior. Os seguintes comandos podem ser utilizados para analisar as margens de fanho e fase do sistema:

```
>> s=tf('s');
   G=2.7778*(s^2+0.192*s+1.92)/(s*(s+1)^2*(s^2+0.384*s+2.56));
   [gm,pm,wg,wp]=margin(G)
```

A margem de ganho é 1.105 na frequência 0.9621 rad/s e a margem de fase é 2.0985°, na frequência 0.9261 rad/s. Já que ambas são muito pequenas, então o sistema em malha fechada terá uma oscilação muito forte, embora seja estável.

3.6 Introdução às Técnicas de Redução de Modelo

Uma técnica de redução de modelo foi introduzida pela primeira vez por Davison em 1966. O método introduzido era reduzir a dimensão da matriz de coeficientes do sistema enquanto preservava alguns autovalores dominantes ou estados mais influentes do sistema original. Técnicas de redução de funções de transferência são algumas vezes chamdas de "simplificação". Aqui, o termo "redução"será utilizado, uma vez que esta terminologia aparece mais frequentemente na literatura. Nesta seção, técnicas de redução de modelo para funções de transferência e espaços de estados serão mostradas.

O modelo reduzido é denotado por onde $k < n$ com n sendo a ordem do sistema original.

$$G_{r/k}(s) = \frac{\beta_1 s^r + \beta_2 s^{r-1} + \cdots + \beta_{r+1}}{\alpha_1 s^k + \alpha_2 s^{k-1} + \cdots + \alpha_k s + \alpha_{k+1}},$$

Novamente, por simplificação, assume-se que $\alpha_{k+1} = 1$.

3.6.1 Aproximações de Padé e Aproximações de Routh

Suponha que a série de Taylor do sistema original $G(s)$ seja escrita como

$$G(s) = c_0 + c_1 s + c_2 s^2 + \cdots,$$

onde os instantes de tempo c_i podem ser calculados. Uma função de transferência de baixa ordem pode ser construída para aproximar o modelo original. Se é desejável preservar os primeiros $r+m+1$ instantes de tempo $c_i (i = 0, \ldots, r+k)$ do modelo original, as seguintes fórmulas podem ser estabelecidas:

$$\begin{cases} \beta_{r+1} = c_0, \\ \beta_r = c_1 + \alpha_k c_0, \\ \quad \vdots \\ \beta_1 = c_r + \alpha_k c_{r-1} + \cdots + \alpha_{k-r+1} c_0, \\ 0 = c_{r+1} + \alpha_k c_r + \cdots + \alpha_{k-r} c_0, \\ 0 = c_{r+2} + \alpha_k c_{r+1} + \cdots + \alpha_{k-r-1} c_0, \\ \quad \vdots \\ 0 = c_{k+r} + \alpha_k c_{k+r-1} + \cdots + \alpha_2 c_{r+1} + \alpha_1 c_r. \end{cases}$$

Para as primeiras m fórmulas, as seguintes equações algébricas podem ser obtidas:

$$\begin{bmatrix} c_r & c_{r-1} & \cdots & . \\ c_{r+1} & c_r & \cdots & . \\ \vdots & \vdots & \ddots & \vdots \\ c_{k+r-1} & c_{k+r-2} & \cdots & c_r \end{bmatrix} \begin{bmatrix} \alpha_k \\ \alpha_{k-1} \\ \vdots \\ \alpha_1 \end{bmatrix} = - \begin{bmatrix} c_{r+1} \\ c_{r+2} \\ \vdots \\ c_{k+r} \end{bmatrix}$$

a partir das quais os coeficientes α_i do denominador podem ser facilmente calculados através da solução de equações algébricas lineares. Para as primeiras $r+1$ fórmulas, os coeficientes β_i do numerador pode ser calculados.

$$\begin{bmatrix} \beta_{r+1} \\ \beta_r \\ \vdots \\ \beta_1 \end{bmatrix} = \begin{bmatrix} c_0 & 0 & \cdots & 0 \\ c_1 & c_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ c_r & c_{r-1} & \cdots & c_0 \end{bmatrix} \begin{bmatrix} 1 \\ \alpha_k \\ \vdots \\ \alpha_{k-r+1} \end{bmatrix}.$$

O algoritmo acima é chamado de aproximação de Padé. A função MATLAB `pademod()` é escrita baseado nesta técnica de redução:

```
function G_red=pademod(G_Sys,r,k)
c=timmomt(G_Sys,r+k+1); G_red=pade_app(c,r,k);
```

A função também chama a função de mais baixo nível `pade_app()`, onde

```
function Gr=pade_app(c,r,k)
w=-c(r+2:r+k+1)'; vv=[c(r+1:-1:1)'; zeros(k-1-r,1)];
W=rot90(hankel(c(r+k:-1:r+1),vv)); V=rot90(hankel(c(r:-1:1)));
x=[1 (W\w)']; dred=x(k+1:-1:1)/x(k+1);
y=[c(1) x(2:r+1)*V'+c(2:r+1)]; nred=y(r+1:-1:1)/x(k+1);
Gr=tf(nred,dred);
```

A sintaxe de `pademod()` é G_r=`pademod(G,r,k)`, onde G é a função de transferência do modelo original e r e k são as ordens desejadas do numerador e do denominador, respectivamente. As variáveis retornadas G_r são o modelo reduzido.

Exemplo 3.25. Considere a função de transferência de quarta ordem

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

Os instantes de tempo c_i e o modelo reduzido de segunda ordem podem ser obtidos utilizandos os seguintes comandos MATLAB:

```
>> G=tf([1 7 24 24],[1 10 35 50 24]); Gr=pademod(G,1,2)
   bode(G,Gr), figure, step(G,Gr)
```

O modelo de segunda ordem aproximado é

$$G_r(s) = \frac{0.8544s + 0.6957}{s^2 + 1.091s + 4.174}.$$

As comparações doss diagramas de Bode e respostas ao degrau são obtidas como mostrado nas Figuras 3.18(a) e (b), respectivamente. É possível ver que o modelo reduzido pode aproximar o modelo original de quarta ordem satisfatoriamente.

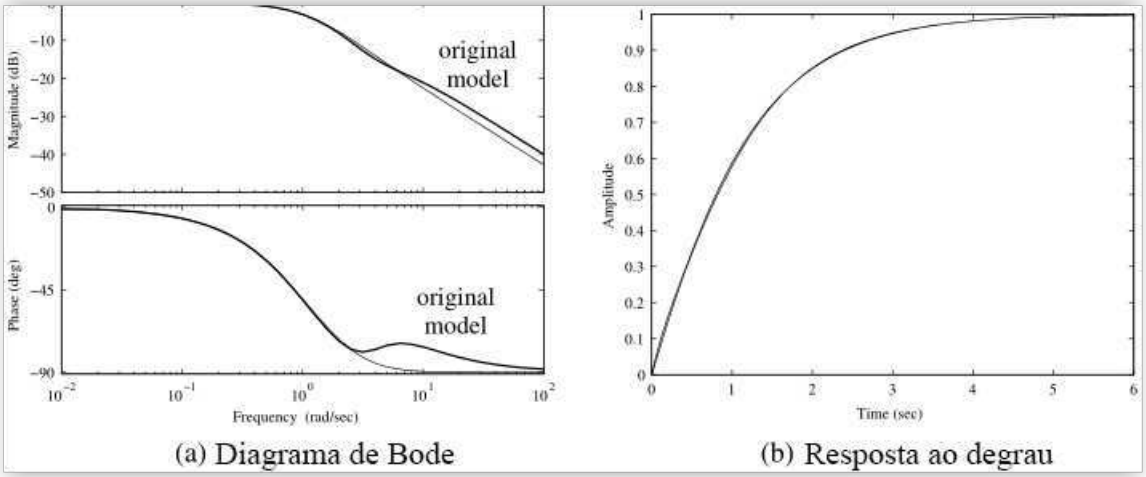


Figura 3.18: Comparações do Diagrama de Bode e da resposta ao degrau

Exemplo 3.26. Assuma que o modelo original é dado por

$$G(s) = \frac{0.067s^5 + 0.6s^4 + 1.5s^3 + 2.016s^2 + 1.55s + 0.6}{0.067s^6 + 0.7s^5 + 3s^4 + 6.67s^3 + 7.93s^2 + 4.63s + 1}.$$

Os polos de $G(s)$ são encontrados:

```
>> num=[0.067,0.6,1.5,2.016,1.66,0.6];
   den=[0.067 0.7 3 6.67 7.93 4.63 1]; G=tf(num,den); zpk(G)
```

O formato zero-polo-ganho do modelo original é

$$G(s) = \frac{(s+5.92)(s+1.221)(s+0.897)(s^2+0.9171s+1.381)}{(s+2.805)(s+1.856)(s+1.025)(s+0.501)(s^2+4.261s+5.582)}.$$

É possível ver que o sistema original é estável. A aproximação de Padé de terceira ordem do sistema original pode ser obtida por

```
>> Gr=zpk(pademod(G,1,3))
```

O modelo reduzido obtido é

$$G_r(s) = \frac{-0.6328(s+0.7695)}{(s-2.598)(s^2+1.108s+0.3123)},$$

Obviamente o sistema reduzido é instável. Isso significa que a aproximação de Padé pode não preservar a estabilidade do sistema original.

Uma vez que a abordagem de Padé pode falhar em preservar a estabilidade do sistema original, a aproximação de Routh é utilizda. O método de aproximação de Routh proposto por Hutton era encontrar um modelo estável de ordem reduzida para o sistema original assintoticamente estável.

A função MATLAB `routhmod()` é escrita de acordo com o algoritmo da aproximação de Routh:

```
function Gr=routhmod(G,nr)
num=G.num{1}; den=G.den{1}; n0=length(den); n1=length(num);
a1=den(end:-1:1); b1=[num(end:-1:1) zeros(1,n0-n1-1)];
for k=1:n0-1,
   k1=k+2; alpha(k)=a1(k)/a1(k+1); beta(k)=b1(k)/a1(k+1);
   for i=k1:2:n0-1,
      a1(i)=a1(i)-alpha(k)*a1(i+1); b1(i)=b1(i)-beta(k)*a1(i+1);
end, end
nn=[]; dd=[1]; nn1=beta(1); dd1=[alpha(1),1]; nred=nn1; dred=dd1;
for i=2:nr,
   nred=[alpha(i)*nn1, beta(i)]; dred=[alpha(i)*dd1, 0];
   n0=length(dd); n1=length(dred); nred=nred+[zeros(1,n1-n0),nn];
   dred=dred+[zeros(1,n1-n0),dd]; nn=nn1; dd=dd1; nn1=nred; dd1=dred;
end
Gr=tf(nred(nr:-1:1),dred(end:-1:1));
```

e pode ser utilizada para encontrar a aproximação de Routh e um dado sistema. A sintaxe desta função é G_r=`routhmod(G,k)`, onde G e G_r são os modelos original e reduzido, respectivamente, e k é a ordem esperada da aproximação de Routh. Note que, no modelo reduzido, a ordem do numerador é uma unidade menor que a do denominador.

3.6.2 Redução de Modelo de Espaço de Estados

Método da realização balanceada

Suponha que a realização balanceada do sistema original pode ser particionada como

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}B_1\\ B_2\end{bmatrix}u,\ y = [C_1\ \ C_2]\begin{bmatrix}x_1\\ x_2\end{bmatrix} + Du$$

e assuma que os estados no subvetor x_2 serão cortados. Então, o modelo reduzido é escrito como a seguir:

$$\begin{aligned}\dot{x}_1 &= [A_{11} - A_{12}A_{22}^{-1}A_{21}]x_1 + [B_1 - A_{12}A_{22}^{-1}B_2]u,\\ y &= [C_1 - C_2A_{22}^{-1}A_{21}]x_1 + [D - C_2A_{22}^{-1}B_2]u.\end{aligned}$$

A função `modred()` implementa o algoritmo acima é fornecida na *Control Systems Toolbox* com a sintaxe G_r=`modred(G,elim)`, onde G é o espaço de estados realizado balanceadao e `elim` contém os estados a serem cortados. O modelo reduzido G_r é então retornado.

Método do truncamento da realização balanceada de Schur

A função de truncamento da realização balanceada de Schur `schmr()`, fornecida na *Robust Control Toolbox*, pode ser utilizada para realizar uma realização de modelo de maneira similar à `modred()`. A diferença entre estas duas técnicas é que um sistema instável pode ser manipulado em `schmr()`. A sintaxe de `schmr()` é G_r=`schmr(G,1,`n_r, onde G é o modelo original no formato de espaços de estados, n_r é a ordem esperada do modelo retornado e G_r retorna o modelo reduzido, também na forma de espaço de estados.

Exemplo 3.27. Considere o modelo abaixo

$$G(s) = \frac{1 + 8.8818s + 29.9339s^2 + 67.087s^3 + 80.3787s^4 + 68.6131s^5}{1 + 7.6194s + 21.7611s^2 + 28.4472s^3 + 16.5609s^4 + 3.5338s^5 + 0.0462s^6}.$$

Para aplicar o algoritmo de redução de Schur, o modelo em espaço de estados deve ser obtido primeiramente. Isto pode ser feito utilizando os seguintes comandos MATLAB:

```
>> num=[68.6131,80.3787,67.087,29.9339,8.8818,1];
   den=[0.0462,3.5338,16.5609,28.4472,21.7611,7.6194,1];
   G=ss(tf(num,den)); Gr=zpk(schmr(G,1,3))
```

Três estados são removidos e o modelo de terceira ordem reduzido utilizando o método de Schur pode ser escrito como

$$G_r(s) = \frac{1485.3076(s^2 + 0.1789s + 0.2601)}{(s + 71.64)(s^2 + 3.881s + 4.188)}.$$

4 *Análise de Simulação de Sistemas Não-Lineares*

Nos capítulos anteriores, apresentamos métodos de modelagem e análise de sistemas lineares. No mundo real, entretanto, sistemas de controle sempre contêm efeitos não-lineares, que podem ser inerentes e inevitáveis ou podem ser introduzidos intencionalmente para fornecer melhor desempenho tecnicamente ou economicamente. Um bom exemplo é o uso de relés para controle *on-off* de temperatura. De fato, é possível imaginar que um sistema de controle que não opera sob a saturação do atuador em algum momento é um projeto ruim do ponto de vista econômico. O MATLAB inclui a linguagem de simulação Simulink e, embora a análise de sistemas não-lineares seja mais difícil que a de sistemas lineares, esta simulação é simples. Não é a intenção aqui apresentar métodos teóricos para o estudo de sistemas não-lineares, mas apresentar o Simulink e suas facilidades na simulação de sistemas não-lineares. Isto é realizado através da discussão de alguns exemplos. Além disso, uma vez que projetos iniciais para diferentes sistemas não-lineares envolvem considerações de modelos linearizados, este importante tópico também será coberto.

4.1 Introdução ao Simulink

Simulink foi desenvolvida pela MathWorks em 1990. Seu nome original era SimuLAB, que foi alterado para o nome atual em 1992. Dois significados estão implícitos neste nome, "simu"e "link". A palavra "link"significa que o diagrama de bloco pode ser connstruído pela "ligação"(do inglês, *linkage*) entre blocos. A palavra "simu"mostra sua característica de simulação. Com o uso das ferramentas poderas fornecidas no Simulink, diferentes sistemas podem ser simulados facilmente e de maneira simples.

4.1.1 Blocos Comumente Utilizados no Simulink

Os algoritmos de modelagem apresentados nos capítulos anteriores não podem ser diretamente aplicados em sistemas não-lineares. Neste caso, o sofisticado ambiente do Simulink pode ser utilizado para representar tais sistemas não-lineares.

Para modelar um sistema não-linear, a biblioteca de blocos do Simulink deve ser aberta primeiramente. Isto pode ser feito de duas maneiras. Iremos utilizar a primeira maneira ao longo deste capítulo.

1. Escreva `open_system('simulink')` no prompt de comando do MATLAB. Então, a janela principal do Simulink será mostrada, como na Figura 4.1.

2. Clique no ícone Simulink na barra de ferramentas na janela do MATLAB, como na Figura 4.2.

É possível ver que um grande número de blocos é fornecido no Simulink. Aqui, os blocos mais comumente utilizados em sistemas de controle são:

1. *Blocos de sistemas lineares*: A função de transferência contínua, o espaço de estados e o modelo zero-polo-ganho são fornecidos no grupo Continuous

2. *Blocos não lineares*: As não-linearidades comumente utilizadas são fornecidas no grupo Discontinuity, como mostrado na Figura 4.4, onde não-linearidades tais como saturação, zona-morta e outras pode ser utilizadas diretamente.

3. *Blocos de entrada e saída*: Os sinais de entrada e saída podem ser modelados utilizando os blocos do grupo Sources, mostrado na Figura 4.5. Degrau, pulso e outros sinais de entrada podem ser representados por blocos deste grupo. Em particular, o bloco Inport pode ser utilizado para modelar a porta de entrada do sistema.

   A saída do sistema pode ser mostrada com os blocos do grupo Sink, mostrado na Figura 4.6. É possível utilizar o bloco Scope para mostrar as curves dos sinais selecionados durante a simulação. O bloco outport deste grupo pode ser utilizado para indicar a porta de saída do sistema.

4. *Blocos matemáticos*: Os sinais no sistema devem ser calculados utilizando os sinais $+, -, \times, \div$, entre outros. Os blocos do grupo Math mostrados na Figura 4.7 podem ser utilizados para modelar estas operações.

Para construir uma simulação mais facilmente, o usuário deve familiarizar-se com os blocos fornecidos no ambiente Simulink. Nas seções a seguir, a modelagem utilizando Simulink será ilustrada através de exemplos.

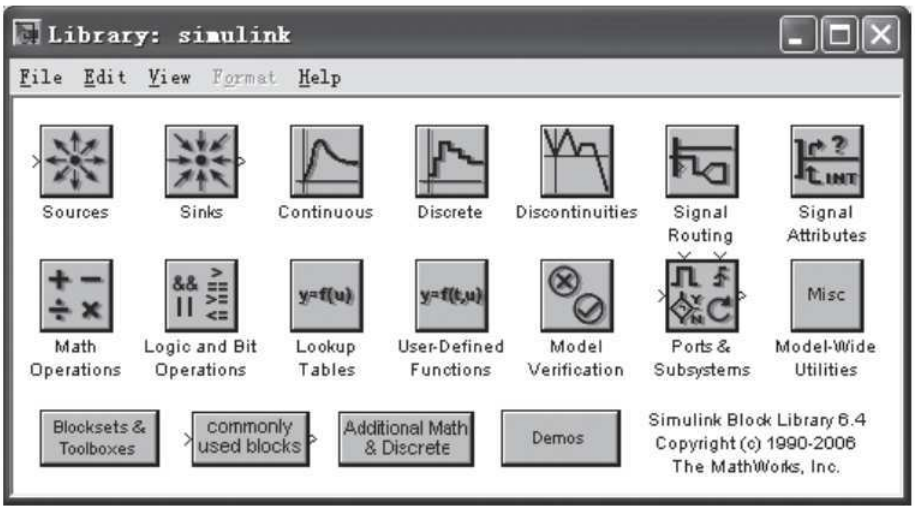


Figura 4.1: Biblioteca de blocos do Simulink

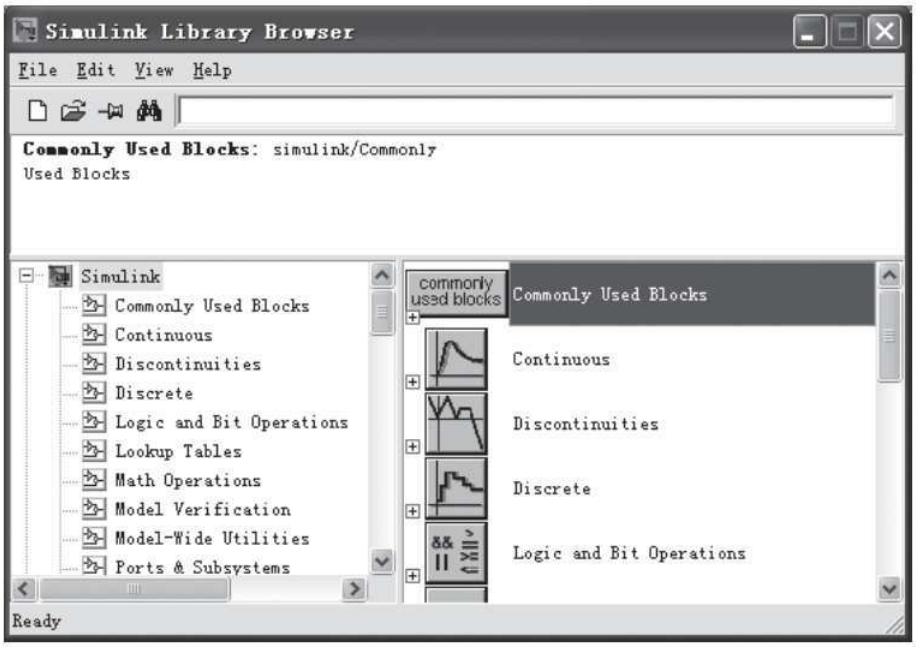


Figura 4.2: Browser da biblioteca de blocos do Simulink

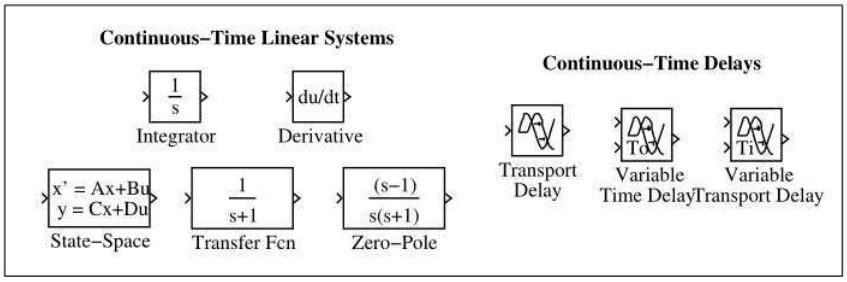


Figura 4.3: Blocos contínuos lineares

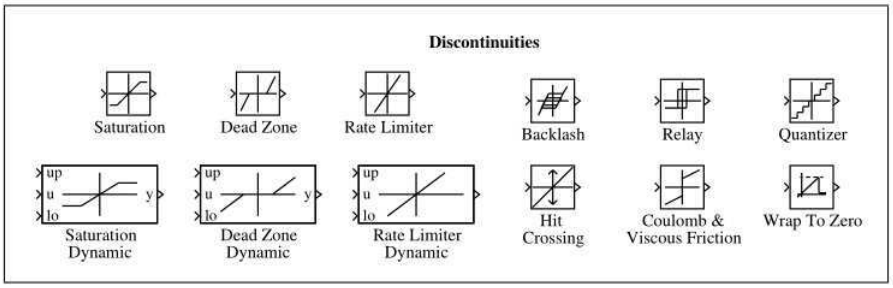


Figura 4.4: Blocos não-lineares

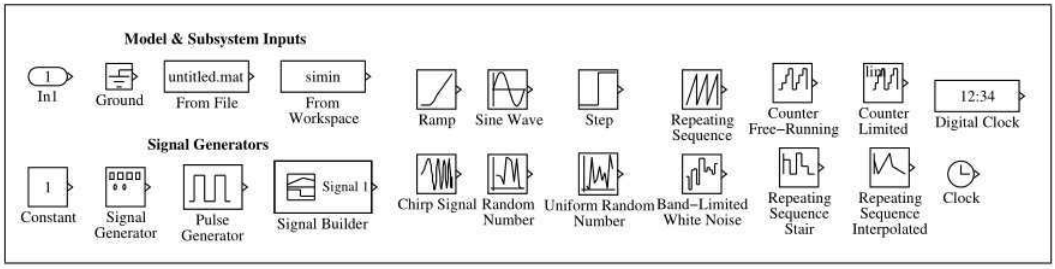


Figura 4.5: Blocos de entrada (fontes)

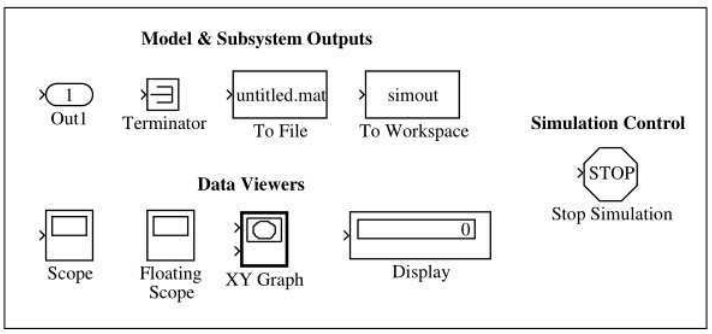


Figura 4.6: Blocos de saída

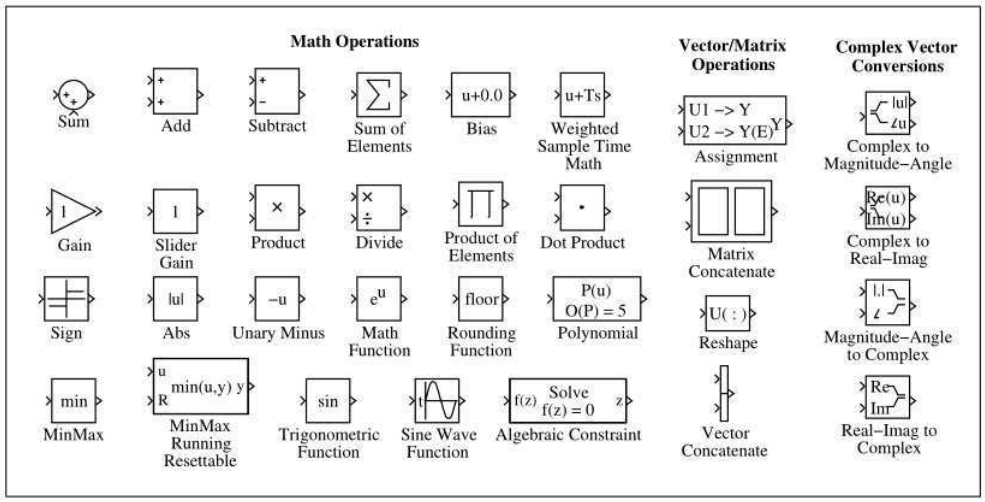


Figura 4.7: Blocos matemáticos

4.1.2 Modelagem em Simulink

Aqui tem temos um breve passo-a-passo para modelagem em Simulink:

1. *Preparação inicial*: Para introduzir um modelo no formato Simulink, é necessário primeiramente iniciar o ambiente Simulink. Então, abre-se uma janela em branco para o novo sistema clicando em File | New.

2. *Adicione os blocos do sistema*: Abra as bibliotecas de blocos relevantes para que os componentes do sistema possam ser copiados para ele. Por exemplo, o ícone nomeado como Continuous na Figura 4.3 contém os blocos mostrados na Figura 4.4, enquanto o ícone Discontinuities contém os mostrados na Figura 4.4. É possível selecionar os blocos nestes grupos e, então, arrastá-los para dentro da nova janela.

3. *Especifique os parâmetros*: Deve-se notar que as bibliotecas mostradas na Figura 4.3 contêm somente modelos padrões de certos tipos. Por exemplo, a função de transferência linear está contida na Figura 4.3, mas somente com o modelo padrão $1/(s+1)$. Para especificar os parâmetros de tal modelo, deve-se dar um duplo clique na caixa de diálogo, como mostrado na Figura 4.8 e, então, preencher os parâmetros requisitados. Deve-se notar também que os numeradores e denominadores requisitados na caixa de diálogo são coeficientes em ordem descendente de s.

4. *Desenhe as conexões*: Uma vez que todos os blocos necessários são colocados na janela, é necessário desenhar as conexões entre os blocos para que se possa concluir o sistema. As conexões entre os blocos podem ser desenhadas clicando na porta de saída de um

bloco e arrastando o mouse até a porta de entrada de outro. Uma conexão entre os dois blocos será internamente estabelecida pelo Simulink.

5. *Especficações de Entrada e Saída*: Usa-se o ícone Inport do grupo Sources para se obter o sinal de entrada entrada do sistema e o ícone Outport do grupo Sinks para conectar à saída do sistema.

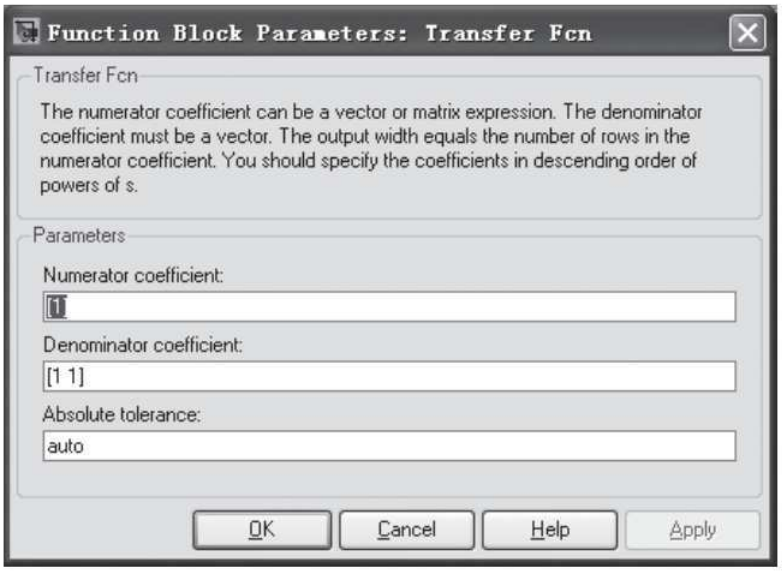


Figura 4.8: Caixa de diálogo dos parâmetros da função de transferência

Exemplo 4.1. Considere o sistema não-linear mostrado na Figura 4.9. É possível ver que existem dois elementos não-lineares no sistema. Utilizando o Simulink, é possivel facilmente desenhar o diagrama de blocos do sistema, como mostrado na Figura 4.10.

Utilizando o Simulink o usuário pode, em teoria, desenhar o diagrama de bloco de um sistema de controle de qualquer complexidade. O Simulink também permite ao usuário realizar a simulação através de itens de menu e chamadas de função relevantes. Os resultados da simulação podem ser mostrados nos escopos fornecidos dentro do Simulink ou retornado ao ambiente de trabalho do MATLAB para que possam ser realizadas as operações de plotagem.

4.1.3 Algoritmos de Simulação e Parâmetros de Controle

Quando o menu Simulation na janela do Simulink é selecionado, aparecerá o menu semelhante ao da Figura 4.11, onde o menu Configuration Parameters pode ser selecionado para especificar os algoritmos de simulação e os parâmetros de controle. A caixa de diálogo é mostrada na Figura 4.12. Os seguintes parâmetros pode ser configurados a partir da caixa de diálogo.

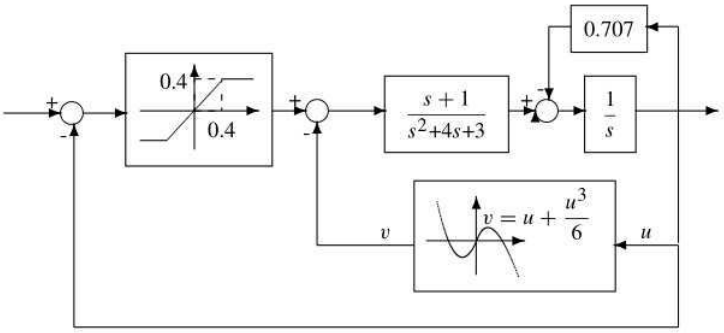


Figura 4.9: Um exemplo de sistema não-linear

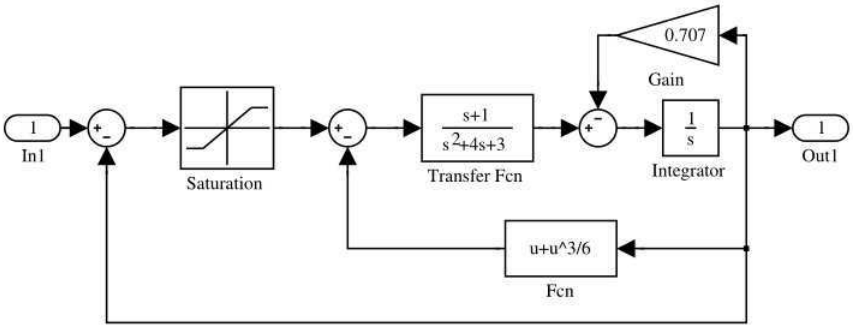


Figura 4.10: O modelo Simulink

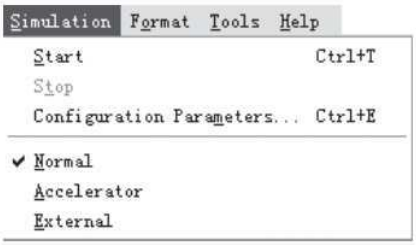


Figura 4.11: Menu simualação

1. As caixa de edição Start time e Stop time pode especificar os tempos de início e parada do processo de simulação.

2. A lista Type na coluna Solver options tem duas opções, onde os algoritmos de degrau variável e degrau fixo podem ser selecionados. Com o fim de manter a alta precisão na simulação, é sugerido selecionar os algoritmos de degrau variável. Frequentemente os

algoritmos ode45 (Dormand-Prince) e ode15s (stiff/NDF) são adequados para problemas de controle.

3. A precisão dos resultados da simulação pode ser controlada pelas opções Relative Tolerance e Absolute Tolerance. A tolerância padrão é `1e-3`, o que significa que o erro relativo pode chegar a um milésimo. O valor padrão é usualmente muito grande e sugere-se diminuir para `1e-6` ou `1e-7`. Deve-se notar que, embora a tolerância relativa seja reduzia significativamente, o esforço computacional não é muito aumentado, devido ao uso de algoritmos de degrau variável.

4. Os degraus máximos e mínimos permitidos podem ser configurados pelo preenchimento das caixas Min step e Max step. Se o real valor do degrau for além do intervalo especificado, uma mensagem de erro será mostrada.

5. As mensagem de aviso e erro podem ser configuradas na coluna Diagnostics.

Depois de completar as especificações dos parâmetros de controle, o item Simulation/Start pode ser selecionado para iniciar o processo de simulação. A variável `tout` será retornada automaticamente e a matriz `yout` pode ser gerada quando a saída é utilizada no modelo Simulink. A função `plot(tout,yout)` pode ser utilizada para mostrar os resultados da simulação.

A função `sim()` pode ser utilizada para iniciar o processo de simulação. A sintaxe da função é

```
[t,x,y]=sim(model_name, t_f, options)
```

onde "model_name"é o nome do arquivo do modelo Simulink, com o sufixo `.mdl` omitido. O argumento t_f é o tempo de parada da simulação. As variáveis retornadas t, x e y são, respectivamente, o vetor tempo, a matriz de estados e a matriz de saída do sistema.

Os parâmetros de controle options pode ser especificado pela função simset(), cuja sintaxe é

onde `property` é o nome do parâmetro, enquanto `parameter` é o valor associado a ele. As propriedades relevantes podem ser listada com o comando `help simset`. Por exemplo, se for desejável mudar o valor da tolerância relativa para 10^{-7}, pode-se fazer pelo comando `options=simset('RelTol',1e-7)` ou simplesmente por `options.RelTol=1e-7`.

Quando a variável options é modificada, ela pode ser utilizada no processo de simulação pelo preenchimento da função `sim()`.

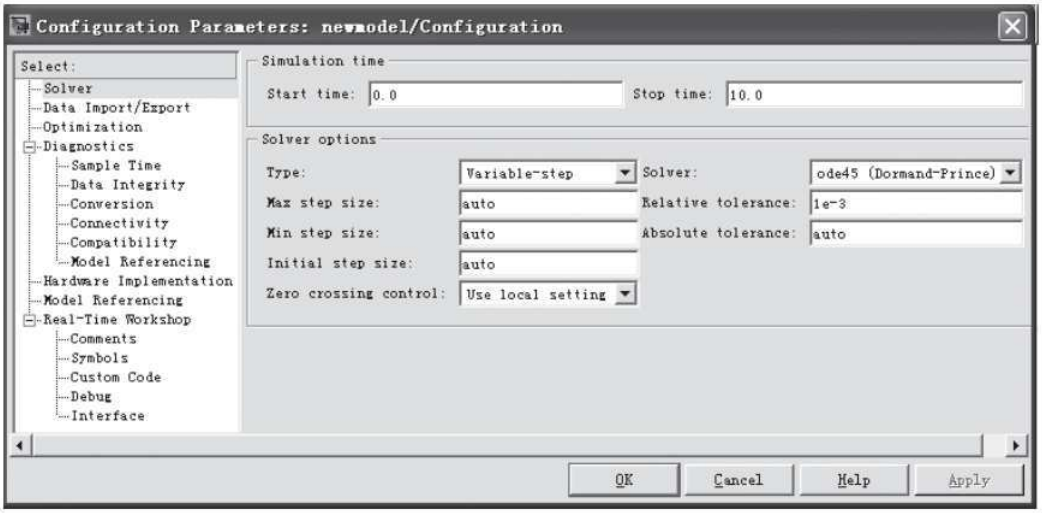


Figura 4.12: Caixa de diálogo dos parâmetros de controle

```
options=simset(property, parameter 1, property 2, parameter 2, ···)
```

4.2 Modelagem de Sistemas Não-Lineares

Uma série de exemplos relacionados com sistemas de controle será utilizados para ilustrar o uso do Simulink. Uma equação diferencial ordinária (EDO) não-linear é utilizada primeiramente como exemplo, seguida de um sistema multivariável, um sistema controlado por computador e um sistema variante no tempo. Será visto nesses exemplos que sistemas com complexidades significantes podem ser simulados utilizando o Simulink.

Exemplo 4.2. Considere a equação caótica de Rössler, cuja forma matemática é

$$\begin{cases} \dot{x}(t) = -y(t) - z(t), \\ \dot{y}(t) = x(t) + ay(t), \\ \dot{z}(t) = b + [x(t) - c]z(t). \end{cases}$$

Selecionando $a = b = 0.2, c = 5.7$ e $x(0) = y(0) = z(0) = 0$, o modelo em Simulink pode ser construído e a simulação pode ser aplicada para o dado sistema.

Um truque simples para simular EDO's é assumir que cada integrador para um variável de estado representa a saída deste estado $x_i(t)$; então, a entrada do integrador é $\dot{x}_i(t)$. O modelo em Simulink na Figura 4.13 pode ser facilmente construído. Os parâmetros de controle para a simulação podem então ser configuradas adequadamente. Se o processo de simulação for iniciado, duas variáveis serão retornada, `tout` e `yout`. O tempo de resposta dos três estados são obtidas com o comando `plot(tout,yout)`, como mostrado na Figura 4.14(a).

Se os estados $x_1(t), x_2(t)$ e $x_3(t)$ são utilizados como os três eixos, a trajetória no espaço de

fases pode ser plotado como mostrado na Figura 4.14(b). A função `comet3()` pode ser utilizada para plotar dinamicamente as trajetórias da curva no espaço de fases:

```
>> plot3(yout(:,1),yout(:,2),yout(:,3))  % or further using comet3
   comet3(yout(:,1),yout(:,2),yout(:,3))
```

Muitos blocos no Simulink suportam operações com vetores, isto é, o bloco pode facilmente processar o caso em que várias entradas são colocadas em um sinal de vetor utilizando o bloco Mux. Se tal sinal é colocado em um bloco integrador, o sinal de saída é também um vetor, cujos canais são integrais dos canais de entrada. Assim, os blocos na Figura 4.15(a) podem ser utilizado para reescrever os blocos no modelo Simulink.

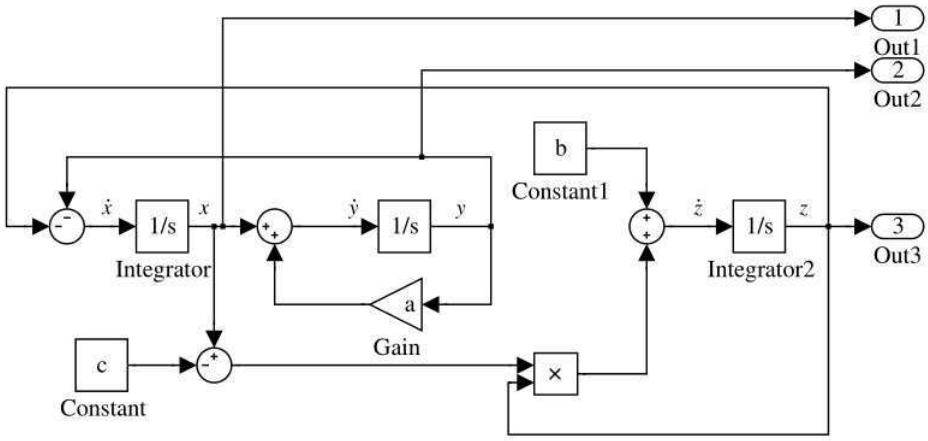


Figura 4.13: Modelo Simulink das equações de Rössler

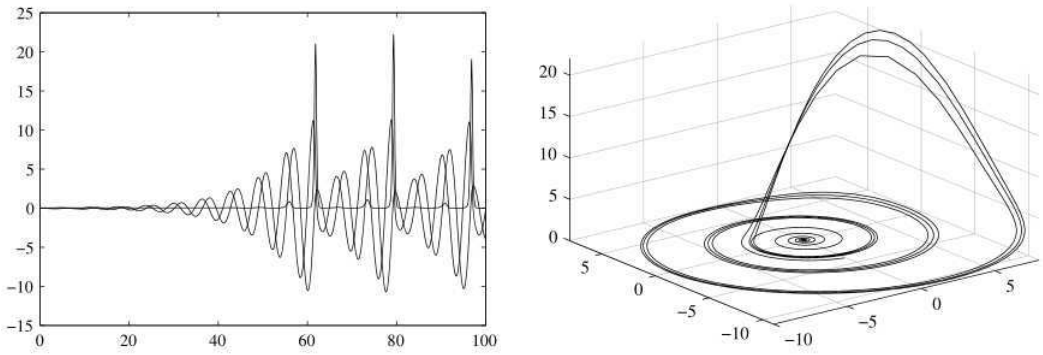


Figura 4.14: Resultados da simulação das equações de Rössler

No modelo, o bloco Fcn é utilizado para definir a operação matemática nos sinais de entrada. A entrada do bloco é o estado do sistema e a entrada do bloco Fcn é denotado por `u`.

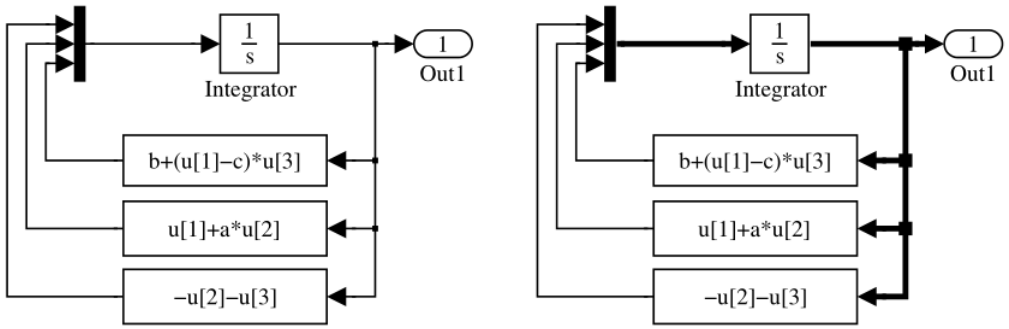


Figura 4.15: Outra descrição para as equações de Rössler

Se `u` é um vetor, `u[`i`]` pode ser utilizado para denotar os componentes dos seus i-ésimos canais. Assim, o modelo construído é muito mais fácil que aquele mostrado na Figura 4.13 e é muito mais facilmente mantido.

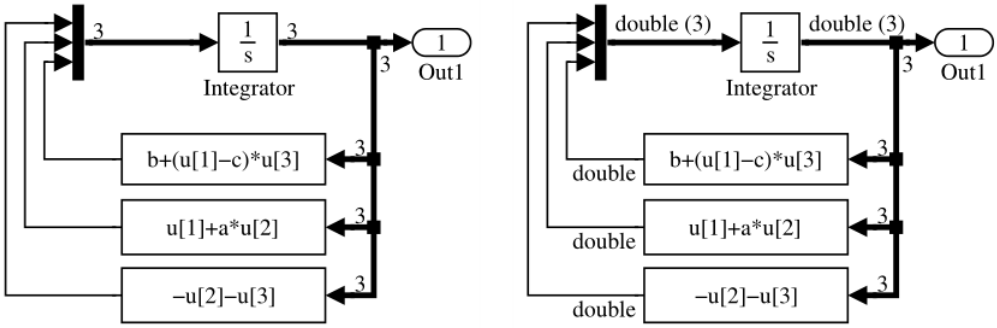


Figura 4.16: Melhorias nos vetores

Os vetores podem ser visualmente aprimorados em Simulink. Por exemplo, o menu Format/Wide nonscalar lines na janela permite expressões com linhas mais grossas para vetores, como mostrado na Figura 4.15(b). Se o usuário selecionar o menu Format/Signal dimensions, as dimensões do sinal serão mostradas sobre os vetores. Por exemplo, uma vez que a variável de estados é tridimensional, um "3"é mostrado, como mostrado na Figura 4.16(a). O menu Format fornece o item Format/Port data types, que permite mostrar os tipos de dados, como mostrado na Figura 4.16(b).

Exemplo 4.3. Considere o sistema multivariável

$$\boldsymbol{G}(s) = \begin{bmatrix} \dfrac{0.1134\mathrm{e}^{-0.72s}}{1.78s^2 + 4.48s + 1} & \dfrac{0.924}{2.07s + 1} \\ \dfrac{0.3378\mathrm{e}^{-0.3s}}{0.361s^2 + 1.09s + 1} & \dfrac{-0.318\mathrm{e}^{-1.29s}}{2.93s + 1} \end{bmatrix}.$$

Uma vez que existem atrasos no tempo na função de transferência do sistema em malha

aberta, a função de malha fechada não pode ser expressada com a função `feedback()`. Aproximações de Padé podem ser utilizadas para aproximar os termos de atraso. Este é o único modo de verificar a precisão da simulação. Com o uso do Simulink, o modelo preciso pode ser construído como mostrado na Figura 4.19. No modelo de simulação, as duas entradas são atribuídas às variáveis `u1` e `u2`.

Utilizamos a função `step()` para encontrar os resultados aproximados da simulação para o sistema multivariável acima quando cada entrada atua individualmente:

```
>> g11=tf(0.1134,[1.78 4.48 1],'ioDelay',0.72);
   g21=tf(0.3378,[0.361 1.09 1],'ioDelay',0.3);
   g12=tf(0.924,[2.07 1]); g22=tf(-0.318,[2.93 1],'ioDelay',1.29);
   G=[g11, g12; g21, g22];
   [n1,d1]=paderm(0.72,0,2); g11.ioDelay=0; g11=tf(n1,d1)*g11;
   [n1,d1]=paderm(0.30,0,2); g21.ioDelay=0; g21=tf(n1,d1)*g21;
   [n1,d1]=paderm(1.29,0,2); g22.ioDelay=0; g22=tf(n1,d1)*g22;
   G1=[g11, g12; g21, g22];
   Kp=[0.1134,0.924; 0.3378,-0.318]; G2=ss(G1*Kp);
   [y1,x1,t1]=step(G2.a,G2.b,G2.c,G2.d,1,15);
   [y2,x2,t2]=step(G2.a,G2.b,G2.c,G2.d,2,15);
```

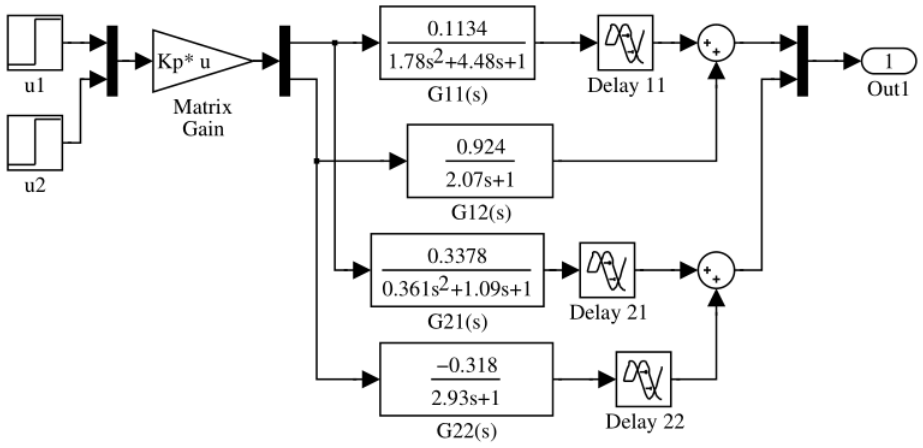


Figura 4.17: Modelo Simulink do sistema multivariável

Através da simulação do sistema com o Simulink, obtemos a resposta ao degrau do sistema com duas entradas. Elas podem ser plotadas juntas com os resultados aproximados, como mostrado na Figura 4.18:

Pode ser visto de 4.18 que os resultados aproximados da simulação estão muito próximos dos resultados exatos.

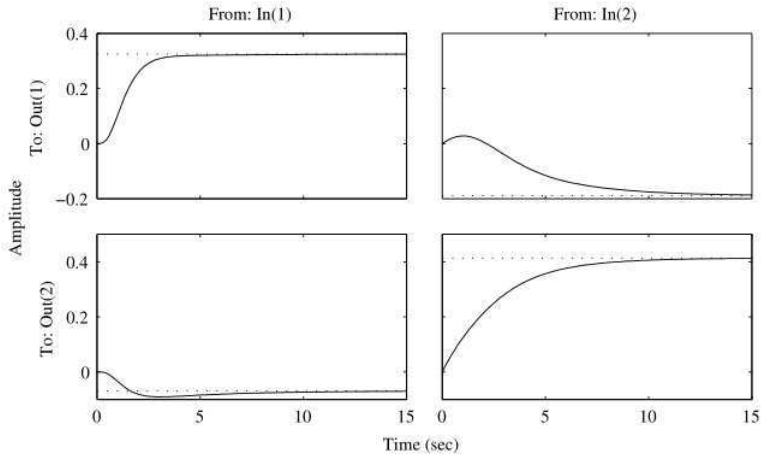


Figura 4.18: Comparações das simulações dos sistemas multivariáveis

```
>> u1=1; u2=0; [tt1,x1,yy1]=sim('c4mmimo',15);
   u1=0; u2=1; [tt2,x2,yy2]=sim('c4mmimo',15);
   subplot(221), plot(t1,y1(:,1),':',tt1,yy1(:,1))
   subplot(222), plot(t1,y1(:,2),':',tt1,yy1(:,2))
   subplot(223), plot(t2,y2(:,1),':',tt2,yy2(:,1))
   subplot(224), plot(t2,y2(:,2),':',tt2,yy2(:,2))
```

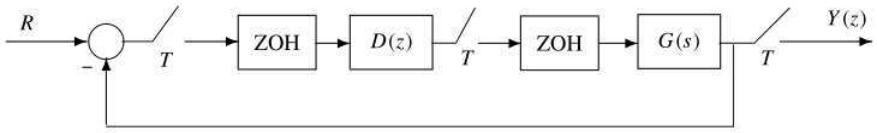


Figura 4.19: Diagrama de blocos de um sistema controlado por computador

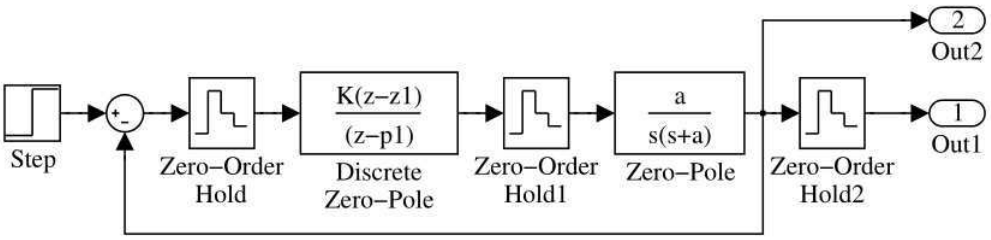


Figura 4.20: Modelo Simulink para um sistema controlado por computador

Exemplo 4.4.

Considere o clássico sistema controlado por computador mostrado na Figura 4.19, onde o controlador é discreto com período de amostragem T segundos. O ZOH é o segurador de ordem zero e a planta é dada por um modelo contínuo. Assuma que a planta e o controlador são dados, respectivamente, por

$$G(s) = \frac{a}{s(s+1)}, \quad D(z) = \frac{1-\mathrm{e}^{-T}}{1-\mathrm{e}^{-0.1T}} \frac{z-\mathrm{e}^{-0.1T}}{z-\mathrm{e}^{-T}},$$

onde $a = 0.1$. Não é possível escrever a equação diferencial correspondente para este sistema, uma vez que ambos os elementos, contínuo e discreto, existem no sistema.

Simulink possui a vantagem de poder resolver este tipo de problema hibrido. Do dado diagrama de bloco do sistema, o modelo Simulink pode ser facilmente construído, como mostrado na Figura 4.20. No modelo do sistema, as variáveis a, T, z_1, p_1, K são utilizadas, onde os dois primeiros devem ser especificados pelo usuário, enquanto as três últimas podem ser calculadas. No primeiro bloco ZOH, o intervalo de amostragem é configurado para T e, por simplicidade, ao resot dos blocos são atribuidos -1, indicando que eles herdam o período de amostragem dos sinais de entrada. Não é necessário colocar o valor de T em cada bloco discreto.

Se $a = 0.1$ e o período de amostragem é dado por $T = 0.2$ segundos, a resposta ao degrau do sistema em malha fechada pode ser obtida como mostrado na Figura 4.21(a):

```
>> T=0.2; a=0.1; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   [t,x,y]=sim('c4mcompc',20);
   plot(t,y(:,2)); hold on; stairs(t,y(:,1))
```

Se o período de amostragem é aumentado para $T = 1$ segundo, a resposta ao degrau do sistema em malha fechada pode ser obtido como visto na Figura 4.21(b). Pode-se ver que quando o período de amostragem aumenta, a diferença entre os sinais contínuo e discreto também aumenta:

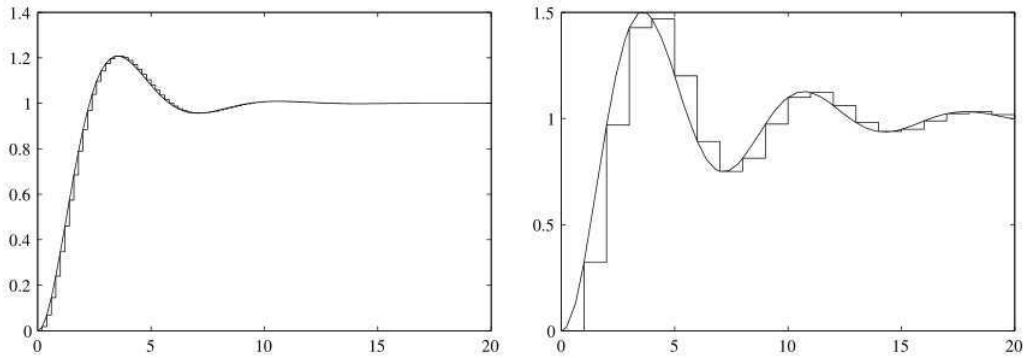
```
>> T=1; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   [t,x,y]=sim('c4mcompc',20);
   plot(t,y(:,2)); hold on; stairs(t,y(:,1))
```

De fato, com o algoritmo de conversão dado no primeiro capítulo, a versão discreta da planta pode ser encontrado sob um intervalo de amostrado T. Assim, o sistema discreto em malha fechada pode ser obtido. A resposta ao degrau do sistema pode ser obtida com os seguintes comandos:

```
>> T=0.2; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   Dz=zpk(z1,p1,K,'Ts',T); G=zpk([],[0;-a],a);  Gz=c2d(G,T);
   GG=zpk(feedback(Gz*Dz,1)), step(GG)
```

O controlador pode ser obtido como

Os comandos podem ser utilizados para obter os mesmo resultados do modelo Simulink, de maneira mais simples, por sinal. Entretanto, este método possui suas limitações.



(a) $T = 0.2$ (b) $T = 1$

Figura 4.21: Respostas ao degrau para diferentes períodos de amostragem

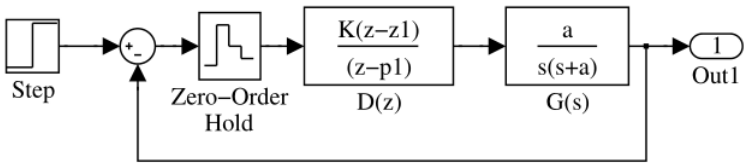


Figura 4.22: Sistema de controlado por computador simplificado

$$G_c(z) = \frac{0.018187(z + 0.9934)(z - 0.9802)}{(z - 0.9802)(z^2 - 1.801z + 0.8368)}.$$

Uma investigação posterior do modelo Simulink mostra que o ZOH depois do controlador $D(z)$ é redundante, uma vez que a saída de $D(z)$ já é um sinal discreto e permanece a mesma após o intervalo de amostragem. Sendo assim, ele pode ser removido. O ZOH no sinal de saída também pode ser removido. A modelo de simulação final pode finalmente ser reduzido para aquele mostrado na Figura 4.22 sem nenhum problema.

Claro que o sistema pode ser simplificado no modelo Simulink, uma vez que todos os ZOHs podem ser removidos, como mostrado na Figura 4.23. Embora este não seja um método oficial, a aproximação é correta no Simulink.

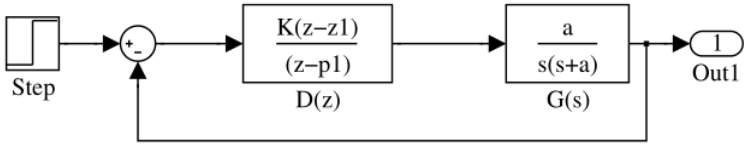


Figura 4.23: Modelo Simulink simplificado

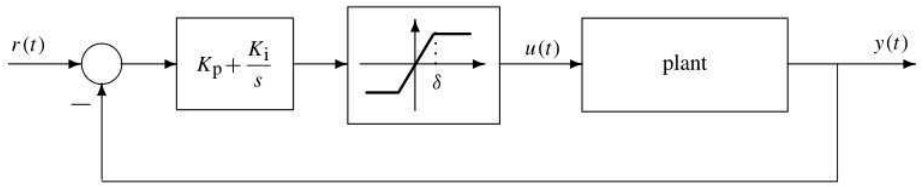


Figura 4.24: Diagrama de blocos do sistema variante no tempo

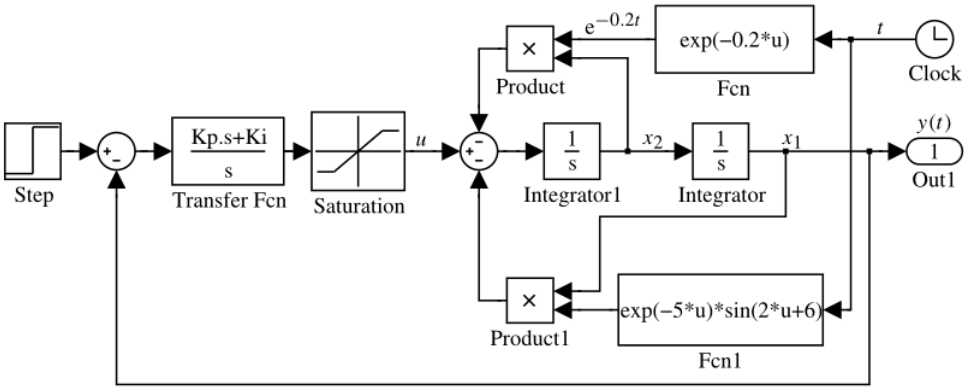


Figura 4.25: Modelo Simulink

Exemplo 4.5. Assuma que o modelo é dado por $\ddot{y}(t) + \mathrm{e}^{-0.2t}\dot{y}(t) + \mathrm{e}^{-5t}\sin(2t+6)y(t) = u(t)$. Agora considere um sistema de controle PI, mostrado na Figura 4.24, onde $K_p = 200$ e $K_i = 10$. A largura do elemento de saturação é $\delta = 2$.

É possível ver que, exceto o bloco variante no tempo, a modelagem do resto do sistema é muito simples. Na parte variante no tempo, assuma que as equações diferenciais explícitas de primeira ordem $x_1(t) = y(t)$, $x_2(t) = \dot{y(t)}$ podem ser utilizadas para estabelecer que

$$\begin{cases} \dot{x}_1(t) = x_2(t), \\ \dot{x}_2(t) = -\mathrm{e}^{-0.2t}x_2(t) - \mathrm{e}^{-5t}\sin(2t+6)x_1(t) + u(t). \end{cases}$$

Utilizando um método similar ao do Exemplo 4.2, um integrador deve ser atribuido para cada variável de estado. O modelo Simulink na Figura 4.25 pode ser construído, onde a função variante no tempo pode ser configurada com blocos.

Uma vez que o modelo de simulação foi construído, os seguintes comandos MATLAB podem ser utilizados para simular o sistema. A resposta ao degrau do sistema variante no tempo pode ser obtida como mostrado na Figura 4.26.

```
>> opt=simset('RelTol',1e-8); Kp=200; Ki=10;
   [t,x,y]=sim('c4mtimv',10,opt); plot(t,y)
```

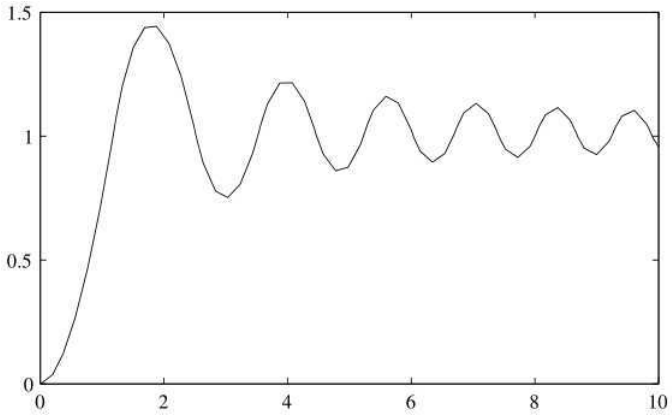


Figura 4.26: Resposta ao degrau do sistema variante no tempo

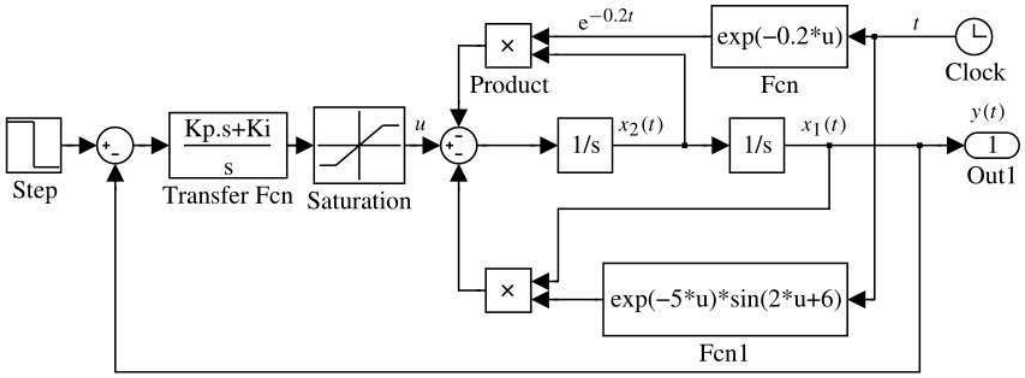


Figura 4.27: Resposta ao impulso de um sistema variante no tempo

Exemplo 4.6. Considere novamente o modelo variante no tempo do Exemplo 4.5. Assuma que o sinal de entrada é um impulso. Simulink é utilizado para encontrar a resposta ao impulso do sistema.

Uma vez que não existe um bloco de impulso no Simulink, o bloco de degrau pode ser utilizado para aproximá-lo. Se o tempo do degrau é a, onde a é um valor extremamente pequeno, o valor inicial do degrau pode ser configurado para $1/a$ e o valor final para 0. O modelo de simulação mostrado na Figura 4.27 pode ser utilizado para modelar o sistema inteiro.

Teoricamente, quando $a \to 0$, o impulso pode ser aproximado. Na simulação real, a pode ser configurado para valores relativamente pequenos, por exemplo, $a = 0.001$. A resposta ao impulso do sistema pode ser obtida com os seguintes comandos MATLAB, como mostrado na Figura 4.28:

```
>> opt=simset('RelTol',1e-8); Kp=200; Ki=10; a=0.001;
   [t,x,y]=sim('c4mtimva',10,opt); plot(t,y)
```

De fato, mesmo a sendo um valor grande, por exemplo $a = 0.1$, uma aproximação muito boa pode ser obtida.

Em aplicações reais, as entradas com sinais periódicos arbitrários podem ser obtidas com o uso do bloco Repeating Sequence. Sinais e sistemas com comportamento ainda mais complicados podem ser modelados com o uso das funções-S.

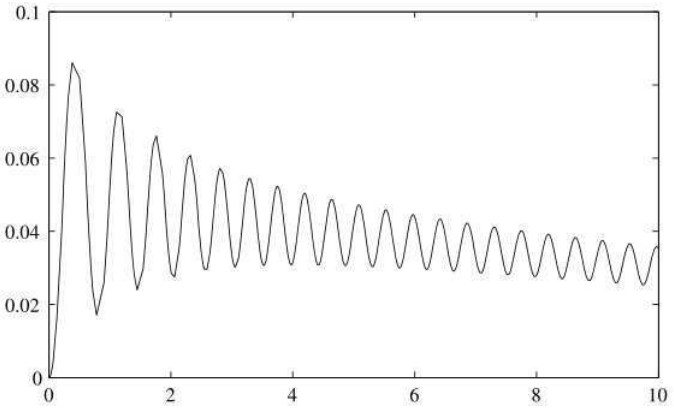


Figura 4.28: Resposta ao impulso de um sistema variante no tempo

4.3 Modelagem de Elementos Não-Lineares

Uma técnica para tentar predizer o ciclo limite em sistemas não-lineares é descrever o método da função. Uma vez que este é um método aproximado, é muito útil para propósitos de comparação quando determina-se soluções por simulação. Nesta seção, modelagem de não-linearidades será discutida em mais detalhes e, então, um sistema simples que possui um ciclo limite é simulado.

4.3.1 Modelagem de Não-Linearidades por Partes Lineares

Não-linearidades estáticas de qualquer complexidade podem ser construídas utilizando os blocos Simulink existentes. Nesta seção, é apresentada a modelagem em Simulink de não linearidades de valor único e valor duplo.

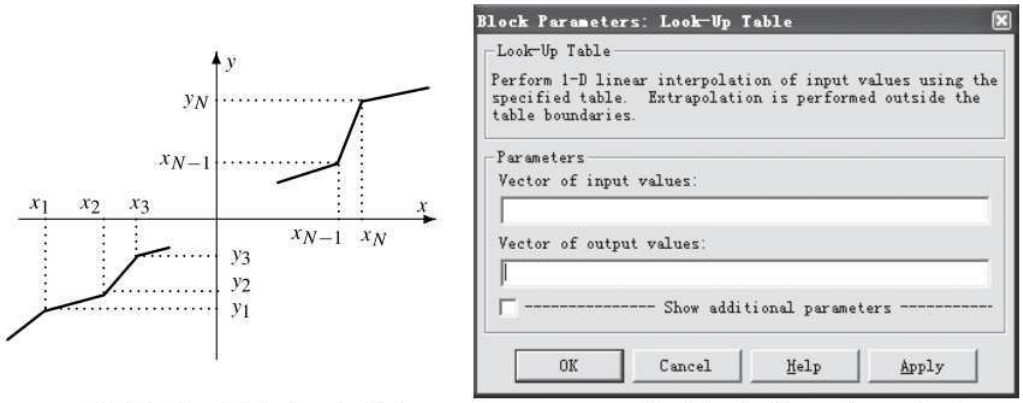
A não-linearidade de valor único estática pode ser facilmente construída com o bloco de *look-up table* unidimensional. Por exemplo, considere a não-linearidade mostrada na Figura 4.29(a), para os pontos de inflexão $(x_1, y_1), (x_2, y_2), \ldots, (x_{N-1}, y_{N-1}), (x_N, y_N)$. É possível selecionar um ponto x_0 de tal modo que $x_0 < x_1$. O valor y_0 pode ser facilmente calculado a partir

do comportamente não-linear. Para outro ponto x_{N+1} de tal modo que $x_{N+1} > x_N$, o valor de y_{N+1} pode ser obtido. Assim, os dois vetores x e y pode ser estabelecidos como

$$\boldsymbol{x} = [x_0, x_1, x_2, \ldots, x_N, x_{N+1}]\,; \quad \boldsymbol{y} = [y_0, y_1, y_2, \ldots, y_N, y_{N+1}]\,;$$

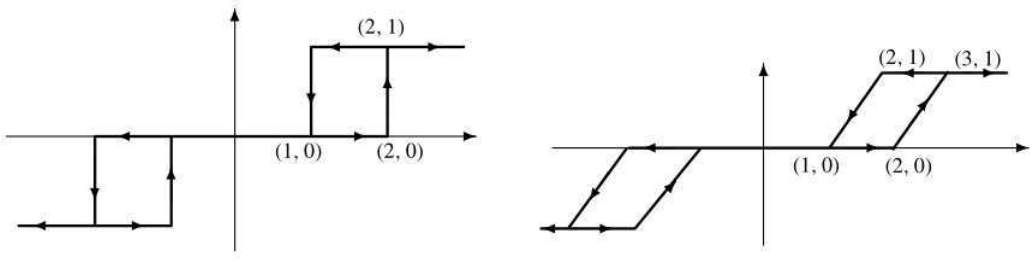
Um duplo clique no bloco Look-Up Table unidimensional abrirá a caixa de diálogo mostrada na Figura 4.29(b). Pode-se especificar nas caixas Vector of input values e Vector of output values os vetores x e y, respectivamente, e então a não-linearidade de valor único pode ser configurada com sucesso.

A construção de uma não-linearidade de valor duplo genérica não é tão simples quando a do caso de valor único. Para sinais de entrada específicos, é possível definidir estas não-linearidades em termos da entrada e suas derivadas, já que um caminho em torno do elemento não-linear será obtido quando a entrada está aumentando e outro quando está diminuindo. Assim, a abordagem será válida para uma entrada senoidal mas não para uma entrada aleatória ou para uma entrada cuja derivada muda a região de valor duplo. Uma abordagem que pode ser utilizada para estas situações é mostrada abaixo.



(a) Não-linearidade de valor único (b) Caixa de diálogo dos parâmetros

Figura 4.29: Construção das não-linearidades de valor único



(a) relay loop (b) saturated relay loop

Figura 4.30: Expressão da função periódica

Exemplo 4.7. Quando existem ciclos na não-linearidade, além de alguns blocos existentes na biblioteca do Simulink, uma não-linearidade genérica não pode ser facilmente construída. O bloco Switch pode ser utilizado para lidar com este problema.

Agora, considere as duas não-linearidades de valor duplo mostradas nas Figuras 4.30(a) e (b). Primeiro, considere a não-linearidade cíclica mostrada na Figura 4.30(a). É possível ver que a função periódica pode ser expressada por uma não-linearidade de valor único quando o sinal de entrada está aumentando e por outra não-linearidade de valor único quando está diminuindo. Isto significa que a não-linearidade de valor único é condicional. Por exemplo, as duas não-linearidades de valor único na Figura 4.31 podem ser utilizadas para expressar a não-linearidade cíclica na Figura 4.29(a).

O bloco Memory pode ser utilizado para extrair o sinal de entrada na instância de tempo anterior. Assim, o modelo Simulink mostrado na Figura 4.32 pode ser utilizado para expressar a não-linearidade cíclica de valor duplo. Um bloco comparador é utilizado para checar se o sinal de entrada está aumentando ou não, isto é, se o valor atual é maior que o valor anterior ou não. Um bloco de chaveamento pode ser utilizado para controlar o bloco de valor único, com o Threshold configurado para 0.5.

As duas não-linearidades de valor único podem ser expressadas por dois blocos *table-look-up* dados por

$$\boldsymbol{x}_1 = [-3, -1, -1+\epsilon, 2, 2+\epsilon, 3], \quad \boldsymbol{y}_1 = [-1, -1, 0, 0, 1, 1],$$
$$\boldsymbol{x}_2 = [-3, -2, -2+\epsilon, 1, 1+\epsilon, 3], \quad \boldsymbol{y}_2 = [-1, -1, 0, 0, 1, 1],$$

onde ε pode ser configurado para um valor muito pequeno. Por exemplo, pode ser configurado para a constante reservada do MATLAB `eps`.

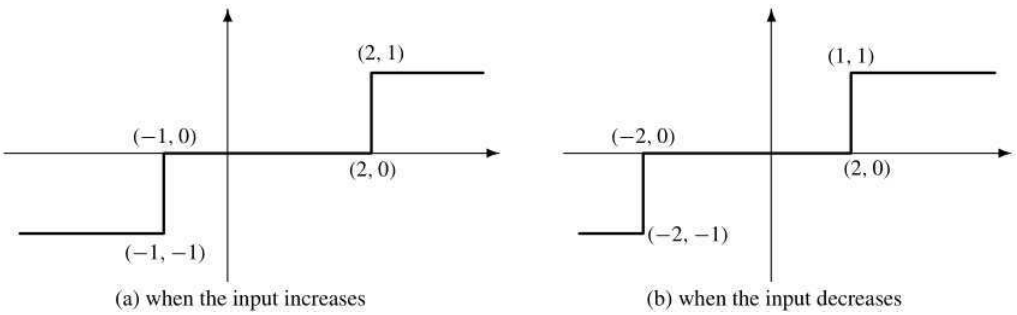


Figura 4.31: A função periódica pode ser expressada como função de valor simples

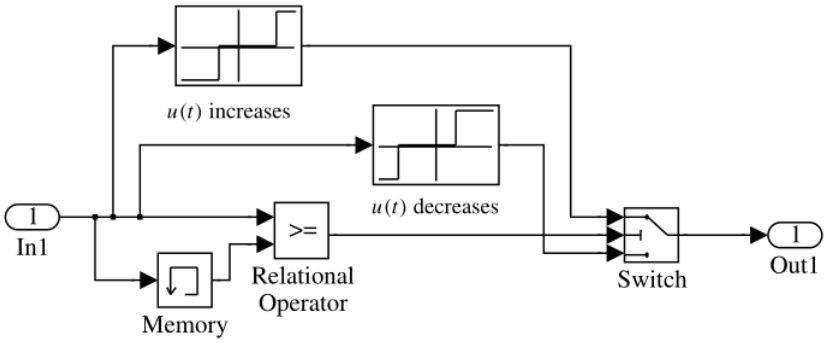


Figura 4.32: Não-linearidade de valor duplo

Considere agora a não-linearidade mostrada na Figura 4.30(b). O modelo Simulink construído anteriormente pode ainda ser utilizado para modelar a não-linearidade. Os novos conjuntos de dados podem ser utilizados para modificar os blocos *table-look-up*

$$\boldsymbol{x}_1 = [-3, -2, -1, 2, 3, 4], \quad \boldsymbol{y}_1 = [-1, -1, 0, 0, 1, 1],$$
$$\boldsymbol{x}_2 = [-3, -2, -1, 1, 2, 3], \quad \boldsymbol{y}_2 = [-1, -1, 0, 0, 1, 1].$$

É possível ver, portanto, que as não-linearidades estáticas de valor único e valor duplo de qualquer complexidade podem ser facilmente modeladas em Simulink. A não-linearidade pode ser utilizada diretamente na simulação.

4.3.2 Ciclo Limite de Sistemas Não-Lineares

Sistemas não-lineares podem ter comportamentos que não estão presentes em sistemas lineares. Uma dada situação é a existência de um ciclo limite, ou oscilação auto-excitada, que pode ser atingido quando o sistema é partido de diferentes condições iniciais.

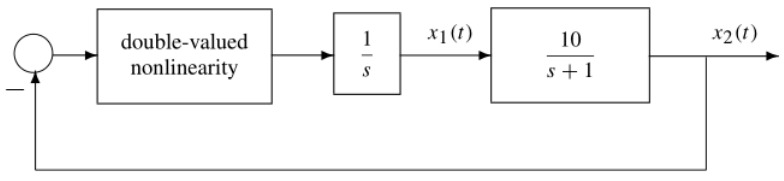


Figura 4.33: Diagrama de bloco do sistema não-linear

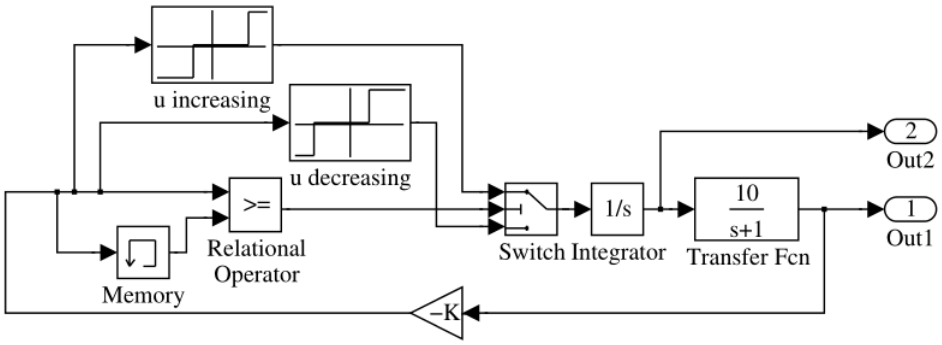
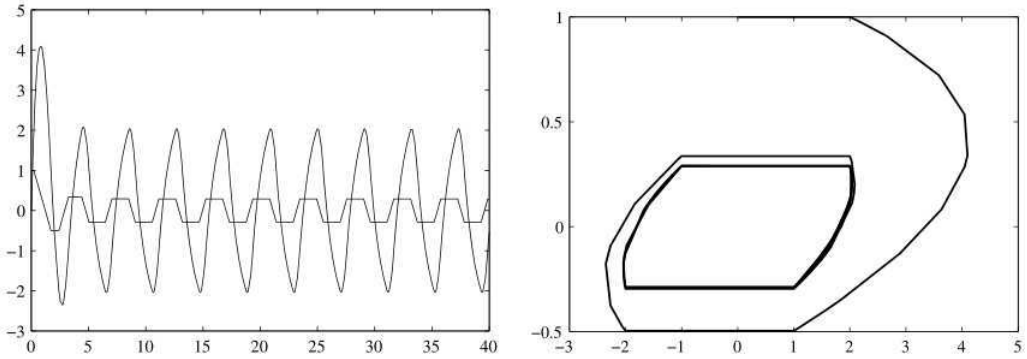


Figura 4.34: Modelo Simulink



(a) Simulação no domínio do tempo (b) Trajetória no plano de fase

Figura 4.35: Resultado da simulação do sistema não-linear

Exemplo 4.8. Considere o típico sistema não-linear realimentado mostrado na Figura 4.33, com o elemento não-linear mostrado na Figura 4.30(a), cujo modelo Simulink é construído na Figura 4.32. Para tal sistema, o modelo Simulink pode ser construído como mostrado na Figura 4.34. No modelo de simulação, o valor inicial do integrador é configurado para um.

Configure o tempo de término da simulação para 40 segundos e mantenha a tolerância relativa precisa configurando a Relative tolerance para 10^{-8} ou um valor menor. Com o seguinte código MATLAB, o tempo de resposta do sistema, quando não existe um sinal de entrada externo, pode ser obtido como mostrado na Figura 4.35(a).

```
>> [t,x,y]=sim('c4mlimcy',40); plot(t,y)
```

É possível ver que os sinais $x_1(t)$ e $x_2(t)$ alcançam oscilações estacionárias depois do período transitório. Com a função gráfica `plot(y(:,1),y(:,2))` fornecida no MATLAB, a trajetória do plano de fase aparece como mostrado na Figura 4.35(b). É possível ver que a trajetória é uma curva fechada, que é chamada de ciclo limite. Um ciclo limite é uma característica interessante que pode ocorrer em um sistema não-linear.

4.4 Linearização de Modelos Não-Lineares

Sistemas lineares são muito mais fáceis de analisar e projetar que os não-lineares. Infelizmente, sistemas que devem ser tratados na prática raramente são lineares. Neste caso, uma aproximação linear do sistema é frequentemente necessária para simplificar os procedimentos de análise e projeto.

Linearização de sistemas tem como objetivo encontrar um modelo linear aproximado, isto é, um modelo linear na vizinhança do ponto de operação.

Considere o sistema dinâmico não-linear

$$\dot{x}_i(t) = f_i(x_1, x_2, \ldots, x_n, u, t), i = 1, 2, \ldots, n. \tag{4.1}$$

Ponto de operação é definido como os valores de estado e variáveis de entrada cujas derivadas das variáveis de estados se aproximam de zero. Isso pode ser obtido pela solução de equações não-lineares definidas em (4.1), tal que

$$f_i(x_1, x_2, \ldots, x_n, \boldsymbol{u}, t) = 0, \quad i = 1, 2, \ldots, n,$$

que pode ser resolvida numericamente. Denote por x_0 o ponto de operação comum sinal de entrada u_0. O sistema não-linear pode ser aproximado por

$$\Delta\dot{x}_i = \sum_{j=1}^{n} \left.\frac{\partial f_i(\boldsymbol{x}, \boldsymbol{u})}{\partial x_j}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} \Delta x_j + \sum_{j=1}^{p} \left.\frac{\partial f_i(\boldsymbol{x}, \boldsymbol{u})}{\partial u_j}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} \Delta u_j.$$

Utilizando as novas variáveis de estado $z(t) = \Delta x(t)$ para o sistema, o modelo linearizado pode ser obtido como:

$$\dot{\boldsymbol{z}}(t) = \boldsymbol{A}_l|_{\boldsymbol{x}_0, u_0}\, \boldsymbol{z}(t) + \boldsymbol{B}_l|_{\boldsymbol{x}_0, u_0}\, \boldsymbol{v}(t),$$

onde $v(t) = \Delta u(t)$ e

$$\boldsymbol{A}_l = \begin{bmatrix} \partial f_1/\partial x_1 & \cdots & \partial f_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \cdots & \partial f_n/\partial x_n \end{bmatrix}, \quad \boldsymbol{B}_l = \begin{bmatrix} \partial f_1/\partial u_1 & \cdots & \partial f_1/\partial u_p \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial u_1 & \cdots & \partial f_n/\partial u_p \end{bmatrix}.$$

Funções úteis para a realização da linearização de sistemas não-lineares são fornecidas no Simulink. O usuário pode utilizar a função `trim()` para encontrar o ponto de operação. A sintaxe da função `trim()` é

```
[x,u,y,z]=trim(model_name,x0,u0)
```

onde `model_name` é o nome do modelo Simulink. As variáveis x_0, u_0 são o chute inicial para os estados e a entrada no ponto de operação. Uma técnica de otimização é utilizada para obter o ponto de operação. Para sistemas sem elementos não-lineares, o chute inicial de x_0, u_0 pode ser omitido. O ponto de operação real é retornado em x, u, y e os valores das derivadas das variáveis de estado são retornadas em z. Teoricamente, as derivadas das variáveis de estado no ponto de operação devem ser iguais a 0.

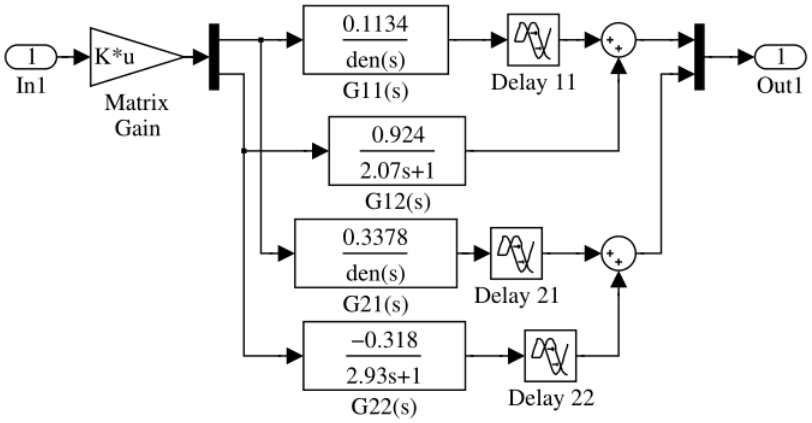


Figura 4.36: Modelo Simulink MIMO

Exemplo 4.9. Considere o sistema multivariável do Exemplo 4.3. Se um modelo linearizado é esperado, a aproximação de Padé deve ser utilizada na linearização dos termos de atraso. Deve-se configurar a caixa Pade order (for linearization) para 2 nos blocos de atraso, onde a aproximação de Padé de segunda ordem pode ser utilizada. O modelo Simulink para o sistema multivariável pode ser então construído como mostrado na Figura 4.36.

Quando o modelo Simulink é construído, os seguintes comandos podem ser utilizados no processo de linearização para obter-se o espaço de estados linear. Os resultados exatos da simulação podem ser obtidos, juntamente com o modelo linearizado, como mostrado na Figura 4.37. Pode-se ver que os resultados da simulação do modelo linearizado são muito precisos.

```
>> Kp=[0.1134,0.924; 0.3378,-0.318];
   [A,B,C,D]=linmod('c4mmdly1'), step(ss(A,B,C,D))
```

O espaço de estados pode ser obtido do sistema

$$\boldsymbol{A} = \begin{bmatrix}
-8.33 & -23.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0637 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -0.4831 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -20 & -133.33 & 0 & 0 & 0 & 0 & 0 & 0.9357 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -4.6512 & -7.2111 & 0 & 0 & 0 & 0 & -0.1085 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -2.5169 & -0.5618 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3.0194 & -2.7701 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.3413
\end{bmatrix},$$

$$\boldsymbol{B}^{\mathbf{T}} = \begin{bmatrix}
0 & 0 & 0.3378 & 0 & 0 & 0 & 0 & 0.1134 & 0 & 0.1134 & 0 & 0.3378 \\
0 & 0 & -0.318 & 0 & 0 & 0 & 0 & 0.924 & 0 & 0.924 & 0 & -0.318
\end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix}
-16.667 & 0 & 0.44638 & 0 & 0 & 0 & 0 & 0 & 0.063708 & 0 & 0 & 0 \\
0 & 0 & 0 & -40 & 0 & -9.3023 & 0 & 0 & 0 & 0 & 0.93573 & -0.10853
\end{bmatrix}.$$

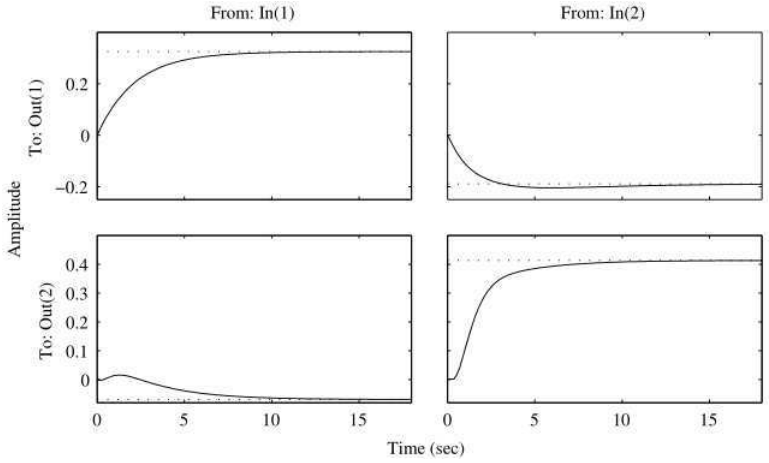


Figura 4.37: Comparação entre os resultados exato e aproximado

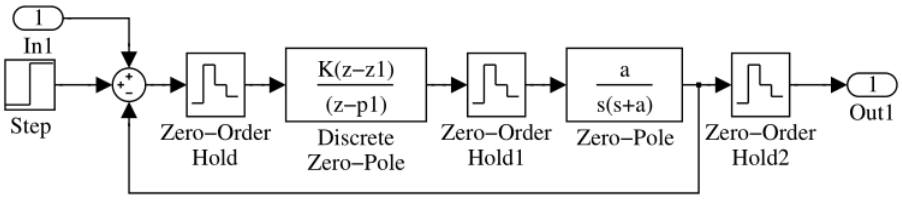


Figura 4.38: Outro modelo Simulink

5 *Projeto de Controladores PID*

Controle PID (proporcinal-integral-derivada) é uma das primeiras estratégias de controle desenvolvidas. Suas primeiras implementações eram em dispositivos pneumáticos, seguidos da eletrônica de estado sólido, chegando às atuais implementações digitais de microprocessadores. Possui uma estrutura de controle simples que pôde ser facilmente entendida pelos operadores da planta e é relativamente fácil de sintonizar. Uma vez que muitos sistemas de controle que utilizam PID tem se mostrado satisfatórios, ele continua com uma larga escala de aplicações em controle industrial. De acordo com uma pesquisa para sistemas de controle realizada em 1989, mais de 90% das malhas de controle eram do tipo PID. Controle PID tem sido constante alvo de pesquisa e desenvolvimento por muitos anos. Uma vez que muitas plantas controladas por PID tem dinâmicas similares, tem sido possível configurar satisfatoriamente os parâmetros do controlador com menos informações sobre a planta. Estas técnicas vieram da necessidade de ajustas os parâmetros com o mínimo esforço e também pela possível dificuldade e baixo custo benefício da dependência dos modelos matemáticos.

5.1 Introdução

5.1.1 As Ações PID

Uma estrutura típica de um sistema de controle PID é mostrado na Figura 5.1, onde é possível ver que, em um controlador PID, um sinal de erro $e(t)$ é utilizado para gerar as ações proporcional, integral e derivativa, com os sinais resultantes ponderados e somados para formar o sinal de controle $u(t)$ aplicado à planta. Uma descrição matemática do controlador PID é

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(\tau) \mathrm{d}\tau + T_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \right],$$

onde $u(t)$ é o sinal de entrada para a planta, o sinal de erro $e(t)$ é definido como $e(t) = r(t) - y(t)$ e $r(t)$ é chamado de sinal de referência.

Os comportamentos das ações proporcional, integral e derivativa serão demonstrados individualmente através do exemplo a seguir.

Exemplo 5.1. Considere o modelo de terceira ordem dado por $G(s) = 1/(s+1)^3$. Se uma estratégia de controle proporcional é selecionada, isto é, $T_i \to \infty$ e $T_d = 0$, para diferentes valores de K_p, a resposta em malha fechada do sistema pode ser obtida utilizando os seguintes comandos MATLAB:

```
>> G=tf(1,[1,3,3,1]);
   for Kp=[0.1:0.1:1], H=feedback(Kp*G,1); step(H), hold on; end
   figure; rlocus(G,[0,15])
```

A resposta ao degrau é obtida como mostrado na Figura 5.2(a). É possível ver que quando K_p aumenta, a velocidade de resposta do sistema e o *overshoot* também aumentam e o erro diminui. Entretanto, quando K_p é grande o suficiente, o sistema em malha fechada se torna instável, o que pode ser diretamente concluído a partir da análise do local das raízes, como mostrado na Figura 5.2b), onde é visto que quando K_p está fora do trecho $(0,8)$, o sistema torna-se instável.

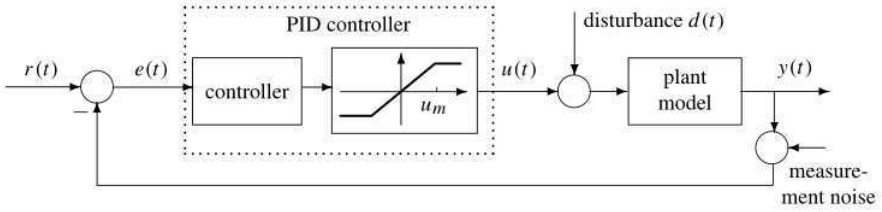
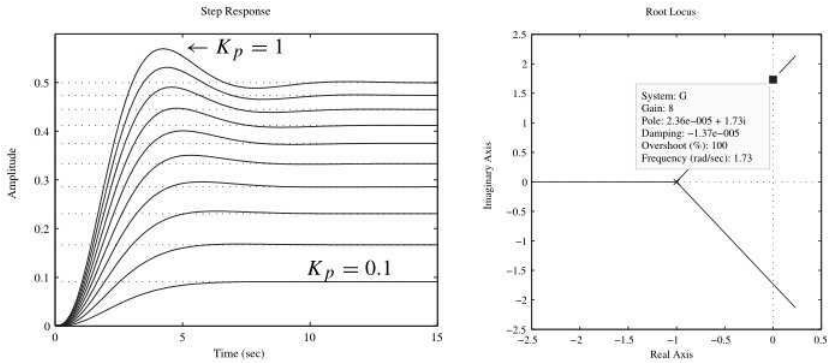
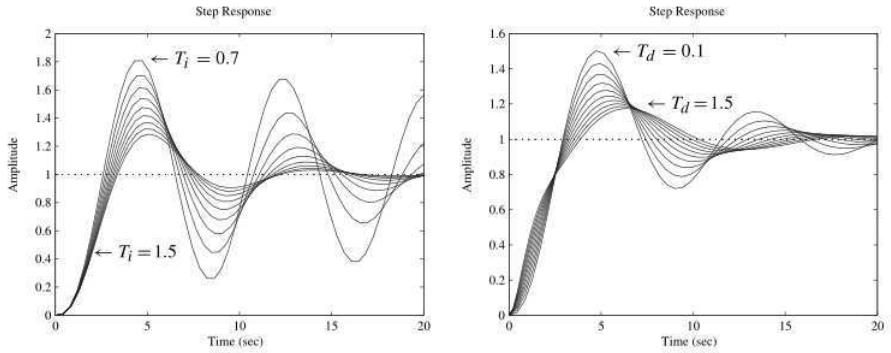


Figura 5.1: Ums típica estrutura de controle PID



(a) resposta ao degrau em malha fechada (b) local das raízes

Figura 5.2: Resposta ao degrau em malha fechada



(a) Controle PI (b) Controle PID

Figura 5.3: Resposta ao degrau em malha fechada

Se fixarmos $K_p = 1$ e aplicarmos o controle PI para diferentes valores de T_i, podemos utilizar os seguintes comandos MATLAB

```
>> Kp=1; s=tf('s');
   for Ti=[0.7:0.1:1.5]
      Gc=Kp*(1+1/Ti/s); G_c=feedback(G*Gc,1); step(G_c), hold on
   end
```

para gerar a resposta ao degrau em malha fechada do sistema mostrada na Figura 5.3(a). A característica mais importante de um controlador PI é que não existe erro estacionário na resposta ao degrau, se o sistema for estável. Uma análise posterior mostra que se T_i é menor que 0.6, o sistema em malha fechada não será estável. É possível ver que quando T_i aumenta, o *overshoot* tende a ser menor, mas o tempo de resposta tende a ser maior.

Fixando K_p e T_i em 1, isto é, $T_i = K_p = 1$, quando o controle PID é utilizado, com diferntes valores de T_d, podemos utilizar os seguintes comandos MATLAB

```
>> Kp=1; Ti=1; s=tf('s');
   for Td=[0.1:0.2:2]
      Gc=Kp*(1+1/Ti/s+Td*s); step(feedback(G*Gc,1)); hold on
   end
```

para obter a resposta ao degrau em malha fechada mostrada na Figura 5.3(b). Claramente, quando T_d aumenta, a resposta tem um *overshoot* menor e um tempo de subida levemente menor, mas com o mesmo tempo de acomodação.

Em aplicações práticas, a ação derivativa pura nunca é utilizada, devido ao impulso gerado no sinal de controle para uma entrada em degrau e à indesejável amplificação de ruído. Normalmente possui um filtro de primeira ordem em cascata. Assim, a transformada de Laplace do controlador PID pode ser escrito como

O efeito de N é ilustrado no exemplo a seguir.

$$U(s) = K_p \left(1 + \frac{1}{T_i s} + \frac{sT_d}{1 + s\frac{T_d}{N}} \right) E(s).$$

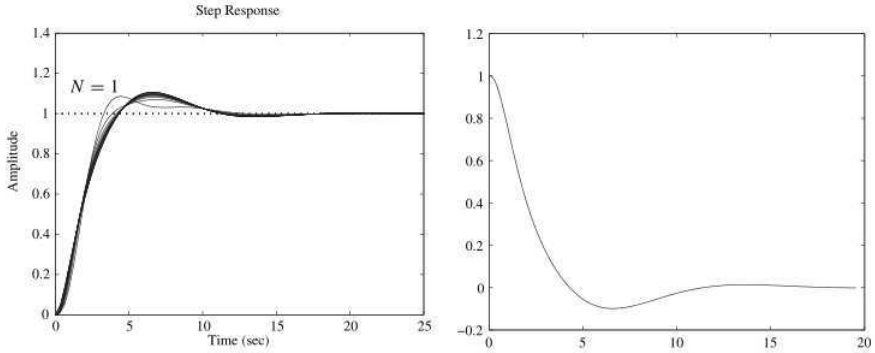
Exemplo 5.2. Considere o modelo mostrado no exemplo anterior. Os parâmetros do controlador PID são $K_p = 1$, $T_i = 1$ e $T_d = 1$. Com diferentes valores de N, podemos utilizar os seguintes comandos MATLAB

```
>> Td=1; Gc=Kp*(1+1/Ti/s+Td*s); step(feedback(G*Gc,1)), hold on
   for N=[100,1000,10000,1:10]
      Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N)); step(feedback(G*Gc,1));
   end
   figure; [y,t]=step(feedback(G*Gc,1)); err=1-y; plot(t,err)
```

para obter a resposta em malha fechada com os termos derivativos aproximados como mostrado na Figura 5.4(a). O sinal de erro $e(t)$ quando $N = 10$ é mostrado na Figura 5.4(b). É possível ver que com $N = 10$, a aproximação é razoavelmente satisfatória.

5.1.2 Controle PID na Realimentação da Malha

Da 5.4(b), é possível ver que existe um *jump* no sinal de erro da resposta ao degrau quando $t = 0$. Isto significa que a ação derivativa pode não ser desejável em tal estratégia de controle.



(a) output signal (b) error signal

Figura 5.4: Controle PID com derivadas aproximadas

Assim, na prática, o termo derivativo pode ser preferível na realimentação. Uma vez que a saída não muda instantaneamente, para uma entrada em degrau, um sinal mais suave é produzido. Esta estratégia de controle PID, que é denotada como PI-D, é mostrada na Figura 5.5.

As funções de transferência do controlador e da realimentação podem ser escritas como

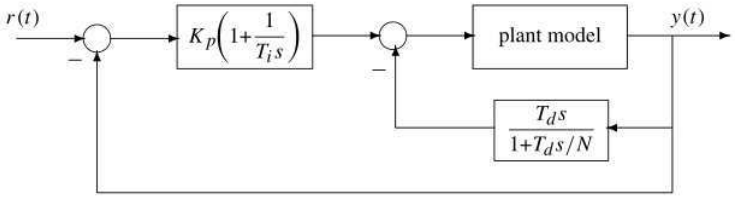


Figura 5.5: Controle PID com derivada no sinal de saída

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s}\right),$$

$$H(s) = \frac{(1 + K_p/N)T_i T_d s^2 + K_p(T_i + T_d/N) + K_p}{K_p(T_i s + 1)(T_d s/N + 1)}.$$

O exemplo a seguir ilustra a consequência do uso do termo derivativo na realimentação.

Exemplo 5.3. Para o modelo mostrado no Exemplo 5.1, através dos seguintes comandos MATLAB

```
>> G=tf(1,[1,3,3,1]); Ti=1; Td=1; Kp=1; N=10; s=tf('s');
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
   G_c=feedback(G*Gc,1); Gc1=Kp*(1+1/s/Ti);
   H=((1+Kp/N)*Ti*Td*s^2+Kp*(Ti+Td/N)*s+Kp)/(Kp*(Ti*s+1)*(Td/N*s+1));
   G_c1=feedback(G*Gc1,H); step(G_c,G_c1)
```

é obtida a resposta ao degrau do sistema em malha fechada em PID e PI-D, mostrada na Figura 5.6. Por observação, a resposta com o controlador PI-D é mais lenta e tem maior *overshoot* neste exemplo em particular.

5.2 Métodos de Sintonia de Ziegler-Nichols

5.2.1 Método Empírico de Ziegler-Nichols

Um método empírico de sintonia muito útil foi proposto por Ziegler e Nichols em 1942. O método de sintonia é obtido quando a planta tem um modelo de primeira ordem com tempo morto (FOPDT, sigla que vem do inglês *First-Order Plus Dead Time*) expressado por

$$G(s) = \frac{k}{1+sT}\mathrm{e}^{-sL}. \tag{5.1}$$

Em sistemas de controle em tempo real, uma grande variedade de plantas podem ser aproximadamente modeladas por 5.1. Se o modelo não pode ser fisicamente derivado, experimentos podem ser realizados para extrair os parâmetros para o modelo aproximado. Por exemplo, se a resposta ao degrau do modelo pode ser medida através de um experimento, o sinal de saída

pode ser esboçado como na Figura 5.7(a), de onde os parâmetros k, L e T (ou a, onde $a = kL/T$) podem ser extraídos de maneira simples. Com L e a, o método de Ziegler-Nichols da Tabela 5.1 pode ser utilizado para obter os parâmetros do controlador.

Tabela 5.1: Método de sintonia de Ziegler-Nichols

Tipo do controlador	da resposta ao degrau			da resposta em frequência		
	K_p	T_i	T_d	K_d	T_i	T_d
P	$1/a$			$0,5K_c$		
PI	$0,9/a$	$3L$		$0,4K_c$	$0,8T_c$	
PID	$1,2/a$	$2L$	$L/2$	$0,6K_c$	$0,5T_c$	$0.12T_c$

Se o experimento para a resposta em frequência pode ser realizado, a frequência ω_c e o ganho K_c podem ser obtidos a partir do Diagrama de Nyquist, como mostrado na Figura 5.7(b) ($T_c = 2\pi/\omega_c$). Os parâmetros do controlador PID podem também ser obtidos a partir da Tabela 5.1. Deve-se notar que a tabela 5.1 aplica-se ao projeto de controladores P, PI e PID utilizando o mesmo conjunto de dados experimentais da planta.

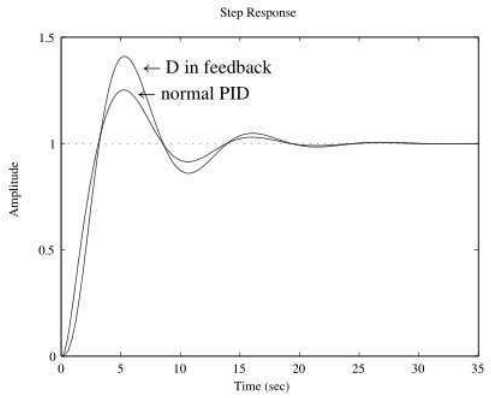


Figura 5.6: Comparação das respostas ao degrau em malha fechada

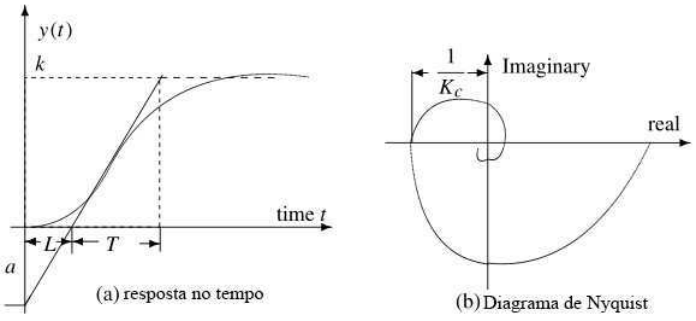


Figura 5.7: Emboços das respostas do modelo FOPDT

A função MATLAB `ziegler()` é criada para projetar controladores PI/PID utilizando o método de sintonia de Ziegler-Nichols:

```
function [Gc,Kp,Ti,Td,H]=ziegler(key,vars)
Ti=[]; Td=[]; H=1;
if length(vars)==4,
   K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T;
   if key==1,  Kp=1/a;
   elseif key==2, Kp=0.9/a; Ti=3.33*L;
   elseif key==3 | key==4, Kp=1.2/a; Ti=2*L; Td=L/2; end
elseif length(vars)==3,
   K=vars(1); Tc=vars(2); N=vars(3);
   if key==1, Kp=0.5*K;
   elseif key==2, Kp=0.4*K; Ti=0.8*Tc;
   elseif key==3 | key==4, Kp=0.6*K; Ti=0.5*Tc; Td=0.12*Tc; end
elseif length(vars)==5,
   K=vars(1); Tc=vars(2); rb=vars(3); N=vars(5);
   pb=pi*vars(4)/180; Kp=K*rb*cos(pb);
   if key==2, Ti=-Tc/(2*pi*tan(pb));
   elseif key==3|key==4, Ti=Tc*(1+sin(pb))/(pi*cos(pb)); Td=Ti/4; end
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

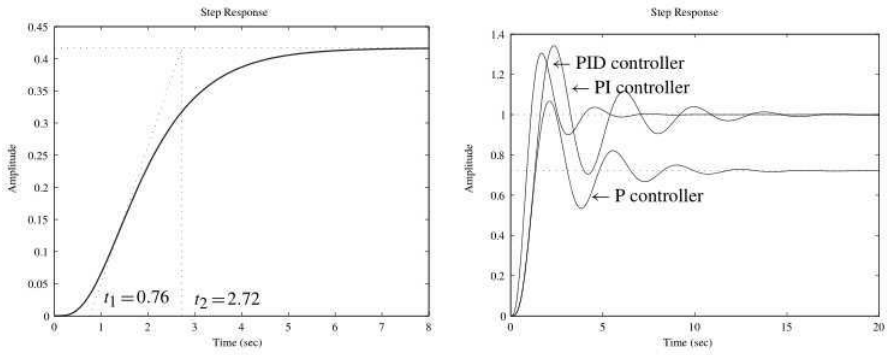
Existe a função de mais baixo nível `writepid()` que pode ser utilizada para o projeto; o conteúdo desta função é

```
function [Gc,H]=writepid(Kp,Ti,Td,N,key)
switch key
case 1, Gc=Kp;
case 2, Gc=tf(Kp*[Ti,1],[Ti,0]); H=1;
case 3, nn=[Kp*Ti*Td*(N+1)/N,Kp*(Ti+Td/N),Kp];
   dd=Ti*[Td/N,1,0]; Gc=tf(nn,dd); H=1;
case 4,  d0=sqrt(Ti*(Ti-4*Td)); Ti0=Ti; Kp=0.5*(Ti+d0)*Kp/Ti;
   Ti=0.5*(Ti+d0); Td=Ti0-Ti; Gc=tf(Kp*[Ti,1],[Ti,0]);
   nH=[(1+Kp/N)*Ti*Td, Kp*(Ti+Td/N), Kp];
   H=tf(nH,Kp*conv([Ti,1],[Td/N,1]));
case 5, Gc=tf(Kp*[Td*(N+1)/N,1],[Td/N,1]); H=1;
end
```

Esta função parece bastante longa para um método tão simples. Na verdade, a função MATLAB também incorpora um outro algoritmo que será discutido posteriormente. Aqui, devemos considerar somente a sintaxe para o método, que é

$$[G_c, K_p, T_i, T_d] = \texttt{ziegler(key,vars)},$$

onde `key` determina o tipo de controlador, com `key=1` para o controlador P, `key=2` para o controlador PI, `key=3` para o controlador PID. Quando os dados da resposta ao degrau estão disponíveis, deve-se especificar `vars=`$[K,L,T,N]$, enquanto `vars=`$[K_c,T_c,N]$ é usado para dados de resposta em frequência.



(a) resposta ao degrau em malha aberta (b) resposta ao degrau em malha fechada

Figura 5.8: Respostas com parâmetros em domínio do tempo

Exemplo 5.4. Considere o modelo de quarta ordem

$$G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)}.$$

O modelo pode ser colocado em MATLAB com:

```
>> s=tf('s'); G=10/(s+1)/(s+2)/(s+3)/(s+4);
   step(G); k=dcgain(G)
```

A resposta ao degrau é mostrada na Figura 5.8(a) com o valor estacionário de 0.4167. Da resposta ao degrau e dos parâmetros do modelo FOPDT aproximado são $k = 0.2941$, $L = 0.76$ e $T = 2.72 - 0.76 = 1.96$, podem ser projetados os controladores P, PI e PID utilizando os comandos MATLAB:

```
>> L=0.76; T=2.72-L; [Gc1,Kp1]=ziegler(1,[k,L,T,10])
   [Gc2,Kp2,Ti2]=ziegler(2,[k,L,T,10])
   [Gc3,Kp3,Ti3,Td3]=ziegler(3,[k,L,T,10])
```

Os controladores P, PI e PID projetados são, respectivamente,

$$G_{\mathrm{p}}(s) = 6.1895,\ G_{\mathrm{PI}}(s) = 5.57\left(1 + \frac{1}{2.5308s}\right),\ G_{\mathrm{PID}}(s) = 7.4274\left(1 + \frac{1}{1.52s} + 0.38s\right).$$

As respostas em malha fechada para esses controladores são obtidos utilizando os comandos MATLAB

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); step(G_c1,G_c2,G_c3);
```

e mostrados na Figura 5.8(b). É possível observar que o errado estacionário existe quando o controlador P é utilizado e a resposta do controlador PID é mais rápida que a do controlador PI.

Se a resposta em frequência do modelo pode ser medida, o ganho K_c e a frequência ω_c podem ser lidas do Diagrama de Nyquist, como mostrado na Figura 5.7(b). Com K_c e ω_c, os parâmetros dos diferentes tipos de controladores PID podem ser obtidos da Tabela 5.1. Neste caso, a função MATLAB `ziegler()` pode ser utilizada.

De fato, uma vez que a frequência ω_c e o ganho K_c são a margem de ganho do modelo em malha aberta, é possível obter os parâmetros utilizando a função `margin()`.

Exemplo 5.5. Considere o modelo do exemplo 5.4. Pelos comandos MATLAB

```
>> G=tf(10, [1,10,35,50,24]);
   nyquist(G); axis([-0.2,0.6,-0.4,0.4])
   [Kc,pp,wg,wp]=margin(G); [Kc,wg], Tc=2*pi/wg;
   [Gc1,Kp1]=ziegler(1,[Kc,Tc,10]); Kp1
   [Gc2,Kp2,Ti2]=ziegler(2,[Kc,Tc,10]); [Kp2,Ti2]
   [Gc3,Kp3,Ti3,Td3]=ziegler(3,[Kc,Tc,10]); [Kp3,Ti3,Td3]
```

a margem de ganho e sua frequência são encontradas como, respectivamente, 12.6 e 2.2361 rad/s. Os controladores são projetados como

$$G_{\mathrm{p}}(s)=6.3, \quad G_{\mathrm{PI}}(s)=5.04\left(1+\frac{1}{2.2479s}\right), \quad G_{\mathrm{PID}}(s)=7.56\left(1+\frac{1}{1.405s}+0.3372s\right).$$

O Diagrama de Nyquist do sistema pode ser obtido como mostrado na Figura 5.9(a). Com esses controladores, a resposta do sistema em malha fechada pode ser obtido utilizando os seguintes comando MATLAB

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); step(G_c1,G_c2,G_c3);
```

e a resposta ao degrau do sistema em malha fechada é mostrada na Figura 5.9(b).

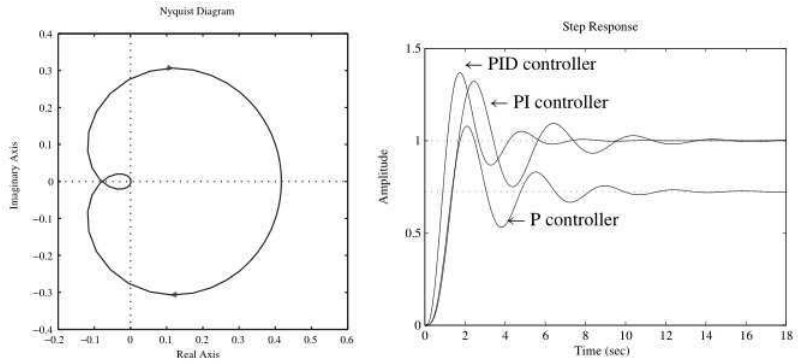
5.2.2 Ação Derivada na Realimentação

Assuma que a ação derivativa é colocada na realimentação, então os parâmetros PID (K_p, T_i, T_d) podem ser obtidos como

$$K_p = K_p'\left(1+\frac{T_d'}{T_i'}\right), \quad T_i = T_i' + T_d', \quad T_d = \frac{T_i'T_d'}{T_i'+T_d'},$$

onde (K_p', T_i', T_d') são os parâmetros PID com o termo derivativo na realimentação.

Em outras palavras, se um controlador PID com ação derivativa no caminha direto é projetado, então um controlador PID equivalente com a ação derivativa não realimentação pode ser obtido pela resolução da seguinte equação algébrica:



(a) Diagrama de Nyquist (b) Resposta ao degrau em malha fechada

Figura 5.9: Respostas do controlador

$$x^2 - T_i x + T_i T_d = 0, \quad \Rightarrow \quad x_{1,2} = \frac{T_i \pm \sqrt{T_i(T_i - 4T_d)}}{2}.$$

É razoável assumir que $T_i > 4T_d$ na maioria dos controladores PID. Neste caso, a equação acima terá raízes reais $x_{1,2}$. Assim, os parâmetros PID equivalentes para a nova estrutura, isto é, com o termo derivativo na realimentação, pode ser calculado como:

$$T_i' = \frac{T_i + \sqrt{T_i(T_i - 4T_d)}}{2}, \quad T_d' = \frac{T_i - \sqrt{T_i(T_i - 4T_d)}}{2},$$
$$K_p' = \frac{2T_i K_p}{T_i + \sqrt{T_i(T_i - 4T_d)}}.$$

A função MATLAB `ziegler()` pode ser utilizada para projetar tal controlador PID. A sintaxe da função agora é

```
[Gc,Kp,Ti,Td,H]=ziegler(key,vars)
```

com `key=4` e H sendo a função de transferência equivalente.

Exemplo 5.6. Considere o modelo do Exemplo 5.4. O controlador PID normal pode ser projetado utilizando o algoritmo de Ziegle-Nichols. Um controlador PID com o termo derivativo na realimentação pode ser obtido utilizando os seguintes comandos MATLAB:

```
>> G=tf(10,[1,10,35,50,24]); N=10; [Kc,Pm,wc,wp]=margin(G);
   Tc=2*pi/wc; [Gc1,Kp1,Ti1,Td1]=ziegler(3,[Kc,Tc,N]),
   [Gc2,Kp2,Ti2,Td2,H]=ziegler(4,[Kc,Tc,N]),
   G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,H);
   step(G_c1,G_c2)
```

Os controladores projetados são $G_{PID}(s) = 7,5600(1 + 1/1,4050s + 0,3372s)$, com $K_p' = 4,5360, T_i' = 0,8430, T_d' = 0,5620$. A resposta ao degrau é mostrada na Figura 5.10(a).

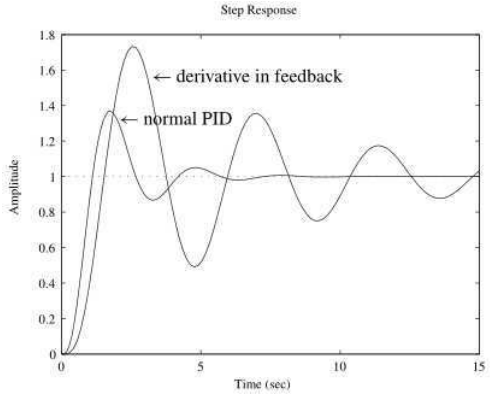


Figura 5.10: Comparação dos controladores PID

É possível ver que, embora o controlador PID com o termo derivativo na realimentação possa ser mais fácil e rápido de implementar comparado com o controlador normal, seu desempenho pode não ser muito satisfatório. Algumas vezes, tal controlador PID deve ser projetado utilizando um algoritmo dedicado para garantir um bom desempenho de controle.

5.2.3 Métodos para Ajuste de Primeira Ordem com Tempo Morto (FOPDT)

É possível ver que o modelo (5.1) é útil para o projeto de controladores PID por causa da viabilidade de uma fórmula simples. O método visto na seção 5.2.1 para encontrar os valores de L e T de uma dada planta é simples para utilizar com o gráfico da resposta ao degrau da planta. Embora em computação moderna seja necessário reduzir o modelo para sua forma para encontrar-se os parâmetros adequados do controlador PID, ainda assim isso pode ser útil. Para a função de transferência da planta , podemos utilizar alguns dos métodos de redução já vistos. Nesta seção, dois outros algoritmos efetivos e frequentemente utilizados serão apresentados.

Método da resposta em frequência

Considere a resposta em frequência do modelo de primeira ordem

$$G(\mathrm{j}\omega) = \frac{k}{Ts+1}\mathrm{e}^{-Ls}\Big|_{s=\mathrm{j}\omega} = \frac{k}{T\mathrm{j}\omega+1}\mathrm{e}^{-\mathrm{j}\omega L}.$$

O ganho K_c na frequência ω_c é a primeira interseção da curva do Diagrama de Nyquist com o semi-eixo negativo real, isto é,

$$\begin{cases} \dfrac{k(\cos\omega_c L - \omega_c T\sin\omega_c L)}{1+\omega_c^2 T^2} = -\dfrac{1}{K_c}, \\ \sin\omega_c L + \omega_c T\cos\omega_c L = 0, \end{cases}$$

onde k é o valor estacionário ou ganho DC do sistema, o que pode ser diretamente calculado a partir da função de transferência. Defina duas variáveis $x_1 = L$ e $x_2 = T$ que satisfaçam

$$\begin{cases} f_1(x_1, x_2) = kK_c(\cos\omega_c x_1 - \omega_c x_2 \sin\omega_c x_1) + 1 + \omega_c^2 x_2^2 = 0, \\ f_2(x_1, x_2) = \sin\omega_c x_1 + \omega_c x_2 \cos\omega_c x_1 = 0. \end{cases}$$

A matriz Jacobiana é

$$\begin{aligned} \boldsymbol{J} &= \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{bmatrix} \\ &= \begin{bmatrix} -kK_c\omega_c \sin\omega_c x_1 - kK_c\omega_c^2 x_2 \cos\omega_c x_1 & 2\omega_c^2 x_2 - kK_c\omega_c \sin\omega_c x_1 \\ \omega_c \cos\omega_c x_1 - \omega_c^2 x_2 \sin\omega_c x_1 & \omega_c \cos\omega_c x_1 \end{bmatrix}. \end{aligned}$$

A função MATLAB `[K,L,T]=getfod(G)` é escrita para resolver x_1 e x_2 a fim de encontrar os parâmetros K, L, T do sistema

```
function [K,L,T]=getfod(G,method)
K=dcgain(G);
if nargin==1
   [Kc,Pm,wc,wcp]=margin(G); ikey=0; L=1.6*pi/(3*wc); T=0.5*Kc*K*L;
   if finite(Kc), x0=[L;T];
      while ikey==0, u=wc*x0(1); v=wc*x0(2);
         FF=[K*Kc*(cos(u)-v*sin(u))+1+v^2; sin(u)+v*cos(u)];
         J=[-K*Kc*wc*sin(u)-K*Kc*wc*v*cos(u), -K*Kc*wc*sin(u)+2*wc*v;
            wc*cos(u)-wc*v*sin(u), wc*cos(u)];
         x1=x0-inv(J)*FF;
         if norm(x1-x0)<1e-8, ikey=1; else, x0=x1;
      end, end
      L=x0(1); T=x0(2);
   end
elseif nargin==2 & method==1
   [n1,d1]=tfderv(G.num{1},G.den{1}); [n2,d2]=tfderv(n1,d1);
   K1=dcgain(n1,d1); K2=dcgain(n2,d2);
   Tar=-K1/K; T=sqrt(K2/K-Tar^2); L=Tar-T;
end
function [e,f]=tfderv(b,a)
f=conv(a,a); na=length(a); nb=length(b);
e1=conv((nb-1:-1:1).*b(1:end-1),a);
e2=conv((na-1:-1:1).* a(1:end-1),b); maxL=max(length(e1),length(e2));
e=[zeros(1,maxL-length(e1)) e1]-[zeros(1,maxL-length(e2)) e2];
```

Método da função de tranferência

Considere o modelo de primeira-ordem com atraso dado por

$$G_n(s) = \frac{ke^{-Ls}}{Ts+1}.$$

Com as derivadas de primeira e segunda ordem de $G_n(s)$ em relação a s, é possível imediatamente encontrar que

$$\frac{G_n'(s)}{G_n(s)} = -L - \frac{T}{1+Ts}, \quad \frac{G_n''(s)}{G_n(s)} - \left(\frac{G_n'(s)}{G_n(s)}\right)^2 = \frac{T^2}{(1+Ts)^2}.$$

$$T_{\text{ar}} = -\frac{G_n'(0)}{G_n(0)} = L + T, \quad T^2 = \frac{G_n''(0)}{G_n(0)} - T_{\text{ar}}^2,$$

Calculando os valores em $s = 0$, tem-se que

onde T_{ar} é também chamado de tempo médio de residência. Da equação, tem-se que $l = T_{ar} - T$. Novamente, o ganho DC k pode ser calculador a partir de $G_n(0)$.

A solução para o modelo FOPDT é então obtida pelo uso de derivadas de sua função de tranferência $G(s)$.

A função MATLAB `getfod)` mostrada anteriormente pode ser utilizada com a sintaxe `[`K,L,T`]=getfod(G,1)` para encontrar os parâmetros K,L,T do sistema.

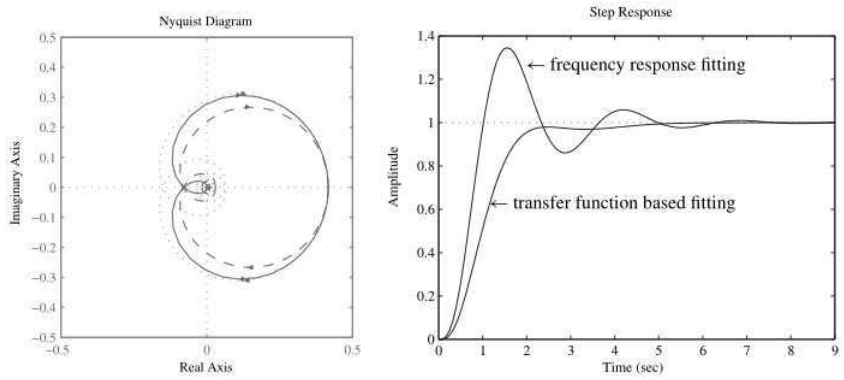
Exemplo 5.7. Considere o modelo de quarta ordem utilizado no Exemplo 5.4. Os parâmetros do seu modelo aproximado FOPDT pode ser obtido utilizando os seguintes comandos MATLAB

```
>> G=tf(10,[1,10,35,50,24]);
   [k,L,T]=getfod(G); G1=tf(k,[T 1]); G1.ioDelay=L;
   [Gc1,Kp3,Ti3,Td3]=ziegler(3,[k,L,T,10])
   [k,L,T]=getfod(G,1), G2=tf(k,[T 1]); G2.ioDelay=L;
   nyquist(G,'-',G1,'--',G2,':'); figure
   [Gc2,Kp4,Ti4,Td4]=ziegler(3,[k,L,T,10])
   G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1); step(G_c1,G_c2)
```

A comparação entre o Diagrama de Nyquist do modelo com os das duas aproximações está mostrado na Figura 5.11(a).

Com o método de resposta em frequência, os parâmetros K,L,T são obtidos como 0.4167, 0.7882, 2.3049. O controlador PID projetado com o método de Ziegler-Nichols é $G_{c1}(s) = 8.4219(1 + 1/1.5764s + 0.3941s)$. Enquanto os parâmetros utilizando o método da função de transferência são 0.4167, 0.8902, 1.1932 e o controlador PID é $G_{c2} = 3.8602(1 + 1/1.7804s + 0.4451s)$. A resposta ao degrau em malha fechada com os dois controladores acima são mostrados na Figura 5.11(b).

É possível ver que embora o controlador PID projetado com o algoritmo de identificação de função de transferência pareça melhor, isso não reflete as usuais características de *overshoot* da sintonia de Ziegler-Nichols, presumivelmente devido aos parâmetros do modelo FOPDT identificados de forma imprecisa.



(a) Diagrama de Nyquist (b) Resposta ao degrau em malha fechada

Figura 5.11: Respostas do controlador PID

5.3 Algoritmos de Sintonia de Controladores PID para Outros Tipos de Planta

5.3.1 Configuração de Parâmetros PD e PID para Modelos IPDT

Uma grande quantidade de plantas encontradas podem ser descritas pelo modelo matemático $G(s) = Ke^{-Ls}/s$, que é chamado de integrador com tempo morto (IPDT, sigla que vem do inglês *Integrator Plus Dead Time*). Este tipo de planta não pode ser controlada pelos controladores PD e PID utilizando os algoritmos mostrados anteriormente.

Uma vez que já existe um integrador no modelo, não é necessário um integrador adicional no controlador para remover o erro estacionário para uma entrada em degrau, mas é necessário para remover o error causado por uma perturbação na entrada. Controladores PD podem também ser utilizados para evitar um grande *overshoot*. Os modelos matemáticos dos controladores PD e PID são, respectivamente,

$$G_{\mathrm{PD}}(s) = K_p(1 + T_d s), \;\; G_{\mathrm{PID}}(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right).$$

Os algoritmos para configuração dos parâmetros PD e PID geram

Tabela 5.2: Os coeficientes do controlador para modelos FOPDT

critério	a_1	a_2	a_3	a_4	a_5
ISE	1,03	0,49	1,37	1,49	0,59
ITSE	0,96	0,45	1,36	1,66	0,53
ISTSE	0,90	0,45	1,34	1,83	0.49

$$\text{PD controller } K_p = \frac{a_1}{KL}, \quad T_d = a_2 L,$$
$$\text{PID controller } K_p = \frac{a_3}{KL}, \quad T_i = a_4 L, \quad T_d = a_5 L,$$

onde para diferentes critérios, os coeficientes a_i podem ser selecionados como mostrado na Tabela 5.2. A seguinte função MATLAB pode ser escrita para implementar os algoritmos acima:

```
function [Gc,Kp,Ti,Td]=ipdtctrl(key,key1,K,L,N)
a=[1.03,0.49,1.37,1.49,0.59; 0.96,0.45,1.36,1.66,0.53;
    0.9,0.45,1.34,1.83,0.49]; s=tf('s'); Ti=inf;
if key==1
   Kp=a(key1,1)/K/L; Td=a(key1,2)*L; Gc=Kp*(1+Td*s/(1+Td/N*s));
else
   Kp=a(key1,3)/K/L; Ti=a(key1,4)*L; Td=a(key1,5)*L;
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td/N*s));
end
```

Na função, `key` é a chave de seleção entre os controladores PD e PID, com `key=1` para PD e 2 para PID. A variável `key1` é configurada como 1, 2, 3 para ISE, ITSE e ISTSE, respectivamente.

Tabela 5.3: Os coeficientes do controlador para modelos IPDT

critério	a_1	b_1	a_2	b_2	a_3	b_3	γ
ISE	1,32	0,92	4,00	0,47	3,78	0,84	0,95
ITSE	1,38	0,90	4,12	0,90	3,62	0,85	0,93
ISTSE	1,35	0,95	4,52	1,13	3,70	0,86	0,97

5.3.2 Configuração de Parâmetros PID para Modelos FOPDT Instáveis

Em sistemas de controle práticos, o modelo pode ser aproximado para um modelo FOPDT instável, isto é,

$$G(s) = \frac{K\mathrm{e}^{-Ls}}{Ts - 1}.$$

Os seguintes algoritmos podem ser utilizados para projetar o controlador PID:

$$K_p = \frac{a_1}{K} A^{b_1}, \quad T_i = a_2 T A^{b_2}, \quad T_d = a_3 T \left[1 - b_3 A^{-0.02}\right] A^{\gamma},$$

onde $A = L/T$. Para diferentes critérios, os coeficientes a_i, b_i, γ do controlador PID podem ser obtidos da Tabela 5.3. Baseado no algoritmo, a função MATKAB para projeto de controlador PID para modelos FOPDT instáveis pode ser escrita como

```
function [Gc,Kp,Ti,Td]=ufopdt(key,K,L,T,N)
Tab=[1.32, 0.92, 4.00, 0.47, 3.78, 0.84, 0.95;
     1.38, 0.90, 4.12, 0.90, 3.62, 0.85, 0.93;
     1.35, 0.95, 4.52, 1.13, 3.70, 0.86, 0.97];
a1=Tab(key,1); b1=Tab(key,2); a2=Tab(key,3); b2=Tab(key,4);
a3=Tab(key,5); b3=Tab(key,6); gam=Tab(key,7); A=L/T;
Kp=a1*A^b1/K; Ti=a2*T*A^b2; Td=a3*T*(1-b3*A^(-0.02))*A^gam;
s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s/(1+Td/N*s));
```

5.4 PID_Tuner: Um Programa de Projeto de Controladores PID para Modelos FOPDT

A maioria de algoritmos de parâmetros de sintonia PID são baseados em modelos FOPDT. Assim, uma interface gráfica foi criada para se utilizada no projeto de controladores PID e também para simulação em malha fechada. Com a interface, os seguintes procedimentos podem ser utilizados para projetar controladores PID:

1. Escreva `pid_tuner` no prompt do MATLAB e aperte *enter*. A interface da Figura 5.12 será aberta.

2. Clique em Plant model; uma caixa de diálogo será apresentada para que se possa descrever o modelo. Qualquer modelo SISO, com ou sem atraso no tempo, pode ser definido. O botão Modify Plant Model pode ser utilizado para modificar os modelos.

3. Uma vez que o modelo foi especificado, o botão Get FOPDT pode ser clicado para extrair os parâmetros FOPDT, isto é, para encontrar K, L, T.

4. Com os parâmetros K, L, T, o controlador pode ser projetado. O tipo do controlador pode ser selecionado pela combinação das listas Choose controller type, Apply to e Tuning algorithm selection.

5. O botão Design Controller pode ser utilizado para projetar o controlador PID.

6. O botão Closed-loop Simulation pode ser utilizado para mostrar a resposta ao degrau em malha fechada do sistema sob os controladores projetados.

5.5 Projeto de Controle Ótimo

Controle ótimo é definido como a otimização de certos índices de desempenho predefinidos. Algumas vezes, funções parmétricas podem ser utilizadas, por exemplo, o problema do

Figura 5.12: Interface do programa para controlador PID

regulador linear quadrático ótimo, onde as matrizes Q, R precisam ser definidas. Existe um métodos ainda não aceito universalmente para definir essas duas matrizes.

Nesta seção, iremos primeiramente sumarizar e ilustrar algumas soluções para problemas de otimização utilizando MATLAB. Então, o método pode ser aplicado para problemas de projeto de controlador ótimo.

5.5.1 Soluções para Problemas de Otimização com MATLAB

Problemas de otimização não-restrita

A formulação matemática do problema de otimização não-restrita é

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}),$$

onde $x = [x_1, x_2, \ldots, x_n]^T$. A interpretação da fórmula é: encontre o vetor x de modo que a função objetiva $F(\mathbf{x})$ é minimizada. Se um problema de maximização é tratado, a função pode ser mudada para $-F(\mathbf{x})$ de modo que pode ser convertido para um problema de minimização.

A função MATLAB `fminsearch()` é fornecida. A sintaxe da função é

onde `Fun` é uma função MATLAB, um função explícita ou uma função anônima para des-

```
[x, f_opt, key, c]=fminsearch(Fun, x_0, OPT)
```

crever a função objetiva. A variável x_0 é o ponto inicial para o método de busca. O argumento `OPT` contém opções de controle para o processo de otimização.

Exemplo 5.8. Se uma função com duas variáveis é dada por $z = f(x,y) = (x^2 - 2x)\mathrm{e}^{-x^2-y^2-xy}$ e o ponto mínimo é desejado, deve-se primeiramente introduzir o vetor x para as variáveis desconhecidos x e y. Seleciona-se $x_1 = x$ e $x_2 = y$. A função objetiva pode então ser reescrita como $f(\mathbf{x}) = (x^2 - 2x)\mathrm{e}^{-x_1^2-x_2^2-x_1x_2}$. A função objetiva pode ser expressada como uma função anônima de modo que

```
>> f=@(x)[(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2))];
```

Se for escolhido um ponto de busca inicial em $(0,0)$, o ponto mínimo pode ser encontrado com os comandos

```
>> x0=[0; 0]; x=fminsearch(f,x0).
```

Então a solução obtida é $\mathbf{x} = [0.6110, -3055]^T$.

Problemas de otimização restrita

A forma geral do problema de otimização restrita é

$$\underset{\boldsymbol{x}\ \text{s.t.}}{\min} \begin{cases} F(\boldsymbol{x}) \\ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{B} \\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq} \\ \boldsymbol{x}_m \leq \boldsymbol{x} \leq \boldsymbol{x}_M \\ \boldsymbol{C}(\boldsymbol{x}) \leq \boldsymbol{0} \\ \boldsymbol{C}_{eq}(\boldsymbol{x}) = \boldsymbol{0}, \end{cases}$$

onde $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$. As restições são classificadas como restrições de igualdade linear $A_{eq} = B_{eq}$, restrições de desigualdade linear $Ax \leq B$ e restrições não-lineares $C_{eq} = 0$ e $C(x) \leq 0$. Os limites superior e inferior das variáveis de otimização podem também ser definidas como $x_m \leq x \leq x_M$.

A interpretação do problema de otimização é: encontre o vetor $\mathbf{x}$ que minimiza a função objetiva $F(\mathbf{x})$ enquanto satisfaz todas as restrições.

A função MATLAB `fmincon()` pode ser utilizada para resolver problemas de otimização restrita. A sintaxe da função é

```
[x, fopt, key, c]=fmincon(Fun, x0, A, B, Aeq, Beq, xm, xM, CFun, OPT)
```

onde `Fun` novamente pode ser funções MATLAB, funções explícitas ou funções anônimas para a função objetiva e x_0 é o ponto de busca inicial. As restrições não-lineares podem ser descritas pela função `CFun`.

Exemplo 5.9. Considere o seguinte problema de programação não-linear:

$$\underset{\boldsymbol{x}\ \text{s.t.}}{\min} \begin{cases} 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3. \\ x_1^2+x_2^2+x_3^2-25=0 \\ 8x_1+14x_2+7x_3-56=0 \\ x_1, x_2, x_3 \geq 0 \end{cases}$$

A função objetiva pode ser expressada com a função anônima

```
>> f=@(x)1000-x(1)*x(1)-2*x(2)*x(2)-x(3)*x(3)-x(1)*x(2)-x(1)*x(3);
```

Também, as duas restrições são igualdades, uma delas sendo não-linear. As restrições não-lineares podem ser descritas na seguinte função MATLAB, onde duas variáveis de restrição `ceq` e `c` são retornadas. Uma vez que não existe uma restrição de desigualdade, a variável `c` retorna uma matriz vazia.

```
function [c,ceq]=opt_con(x)
ceq=x(1)*x(1)+x(2)*x(2)+x(3)*x(3)-25; c=[];
```

A restrição de igualdade linear pode ser expressada pelas matrizes A_{eq}, B_{eq}, enquanto as matrizes A e B devem ser vazias, uma vez que não existem desigualdades lineares no problema. Selecionando a posição de busca inicial em $x_0 = [1,1,1]^T$, o problema pode então ser resolvido utilizando os seguintes comandos:

```
>> x0=[1;1;1]; xm=[0;0;0]; xM=[]; A=[]; B=[]; Aeq=[8,14,7]; Beq=56;
   [x,f_opt,c,d]=fmincon(f,x0,A,B,Aeq,Beq,xm,xM,'opt_con')
```

A solução ótima pode então ser encontrada, onde $x^* = [3,5121, 0,2170, 3,5522]^T$ e $f_{opt} = 961,7151$.

5.5.2 Projeto de Controlador Ótimo

Com as poderosas ferramentas fornecidas no MATLAB, muitos problemas de controle ótimo podem ser convertidos em problemas de otimização. Com as função mencionadas anteriormente, alguns problemas de projeto de controlador ótimo pode ser facilmente resolvidos. Embora não permitindo soluções analíticas elegantes, métodos numéricos são técnicas extremamente eficientes para projeto de controladores.

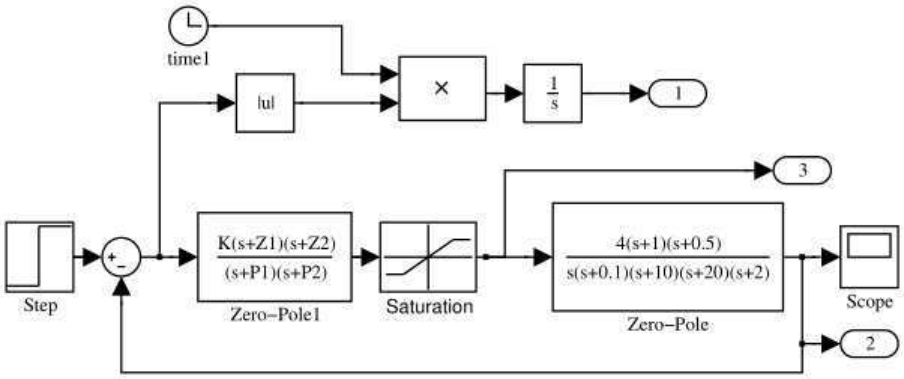


Figura 5.13: Modelo Simulink com saturação

Exemplo 5.10. Assuma que o sinal de controle deve ser mantido entre ± 20, o modelo Simulink pode ser modificado como mostrado na Figura 5.13. A função objetiva pode ser escrita como

Os seguintes comandos podem ser utilizados para buscar pelo controlador ótimo para o sistema:

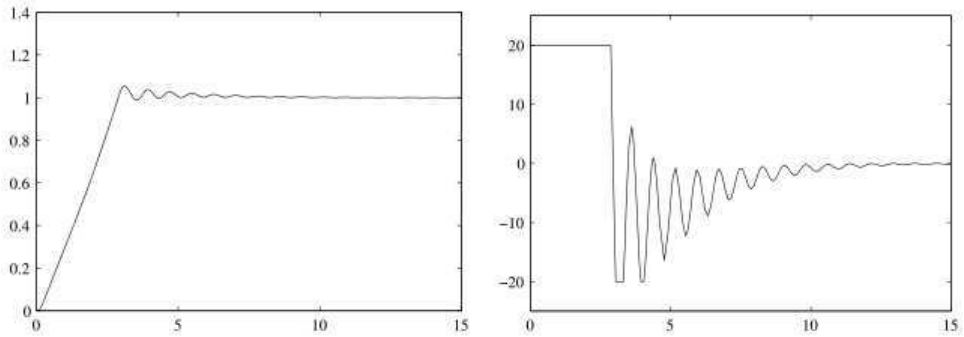
e o controlador

```
function y=c6optm3(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % assign variables in MATLAB workspace
[t,xx,yy]=sim('c6moptm3.mdl',15); y=yy(end,1);   %   evaluate objective function
if max(yy(:,2))>1.03, y=1.4*y; end    % update the objective function
```

```
>> x=fmincon('c6optm3',x0,[],[],[],[],0.01*ones(5,1))
```

$$G_c(s) = 37.1595\frac{(s+142.6051)(s+62.6172)}{(s+20.3824)(s+27.6579)}$$

pode ser projetado. O sinal de saída e o sinal de controle podem ser obtidos como mostrado na Figura 5.14. É possível ver que os resultados do controle são satisfatórios.



(a) Sinal de saída (b) Sinal de controle

Figura 5.14: Resposta ao degrau do sistema quando a saturação é introduzida

5.5.3 Projeto de Controlador Ótimo Utilizando MATLAB/Simulink

Dos exemplos na seção anterior, utilizando algoritmos de otimização numérica, o projeto de controle ótimo pode ser feito de maneira simples. Nesta seção, iremos apresentar o programa de projeto de controlador ótimo (OCD, sigla do inglês *Optimal Controller Designer*) com base em MATLAB/Simulink com alguns exemplos de aplicação.

O procedimento para aplicar o programa OCD é o seguinte:

1. Escreva `ocd` no *prompt* do MATLAB; a interface principal é mostrada na Figura 5.15.

2. O modelo Simulink pode ser construído e deve conter pelo menos dois elementos: os nomes das variáveis indeterminadas e a saída refletindo o critério ótimo. Por exemplo,

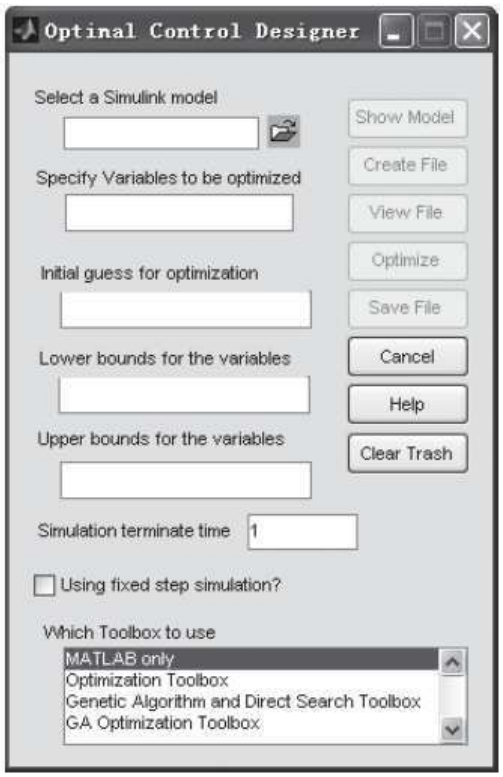


Figura 5.15: Interface OCD

no projeto do controlador PI, as variáveis K_p e K_i podem ser associadas. O critério ITAE pode ser representado no modelo Simulink como a saída 1.

3. Preencha o nome do modelo Simulink na caixa Select a Simulink model name.

4. Preencha os nomes das variáveis a serem otimizadas na caixa Specify Variables to be optimized com os nomes das variáveis separadas por vírgula.

5. Estime o tempo para o erro ir a zero e coloque na caixa Simulation terminate time.

6. Clique em Create File para automaticamente gerar a função MATLAB `optfun_*.m` e clique em Clear Trash para deletar os arquivos temporários de funções objetivas.

7. Clique em Optimize para iniciar o processo de otimização. Algumas vezes, o botão deve ser clicado novamente para garantir soluções otimizadas mais precisas. As funções `fminsearch()`, `nonlin()` e `fmincon()` podem ser chamadas automaticamente para parâmetros de otimização.

8. As variáveis de limite superior e inferior podem ser utilizadas e um ponto inicial de busca podem ser especificados, se necessário.

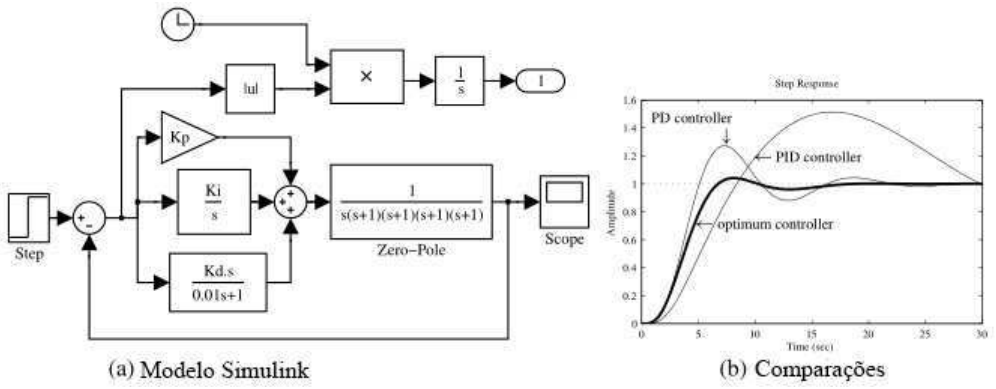
Exemplo 5.11. Considere novamente o modelo

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

O modelo Simulink para o controle PID, com descrições ITAE, é construído como mostrado na Figura 5.16(a).

Preencha a caixa Select a Simulink model name com o nome do modelo. Os nomes das variáveis a serem otimizadas K_p, K_i, K_d devem ser colocados na caixa Specify Variables to be optimized. Na caixa Simulation terminate time deve ser colocado o valor 30. Então, clique em Create File para automaticamente gerar a função MATLAB que descreve a função objetiva onde a segunda, terceira e quarta linhas no código irão associar o vetor **x** às variáveis K_p, K_i, K_d no MATLAB. Simulação é então realizada para calcular a função objetiva.

```
function y=optfun_2(x)
assignin('base','Kp',x(1));
assignin('base','Ki',x(2));
assignin('base','Kd',x(3));
[t_time,x_state,y_out]=sim('c6mopt4.mdl',[0,30.000000]);
y=y_out(end);
```



(a) Modelo Simulink (b) Comparações

Figura 5.16: Comparações das respostas

Clique no botão Optimize para iniciar o processo de otimização. Após a otimização, o controlador PID ótimo será obtido como

$$G_c(s) = 0.2583 + \frac{0.0001}{s} + \frac{0.7159s}{0.01s+1}$$

que minimiza o critério ITAE. É possível ver que $K_i = 0.00001$ é muito pequeno, e então pode ser desprezada. Assim, o controlador PD é suficiente para o sistema. A resposta ao degrau em malha fechada é mostrada na Figura 5.16(b).

$$G(s)=\frac{s+2}{s^4+8s^3+4s^2-s+0.4}.$$

Exemplo 5.12. Considere o modelo instável

Pelo uso direto do programa OCD, o controlador PID não é factível. Entretanto, ainda é possível construir um modelo Simulink como mostrado na Figura 5.17, que é o mesmo do exemplo anterior.

A fim de garantir que a ação de controle não seja muito grande, um elemento de saturação pode ser adicionado ao controlador, com a saturação $\Delta = 5$. Do programa OCD, o controlador PID ótimo pode ser projetado como

$$G_c(s) = 47.8313 + \frac{0.2041}{s} + \frac{55.3632s}{0.01s+1}.$$

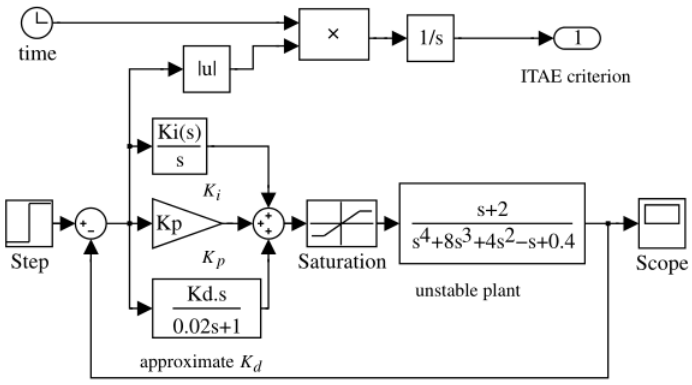


Figura 5.17: Modelo Simulink para controle PID

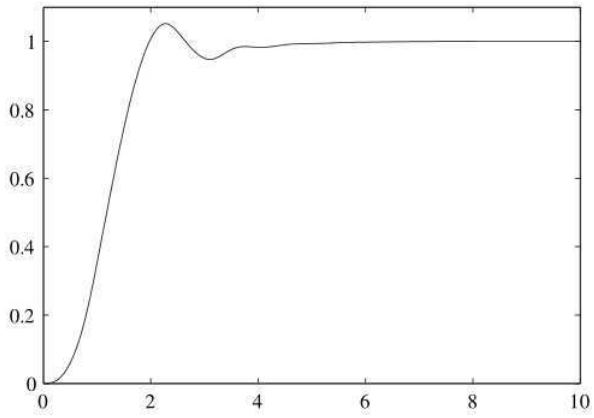


Figura 5.18: Resultado da simulação para um modelo instável com controlador PID

A resposta ao degrau do sistema em malha fechada sob o controlador ótimo é mostrado na Figura 5.18. É possível ver que o controlador PID pode ainda ser projetado e a resposta transitória é satisfatória.

6 Conclusão

O trabalho atingiu seu objetivo de elaboração de material didático para Controle e Automação com foco em MATLAB. O material elaborado pode ser de grande auxílio no ensino das disciplinas básicas de Controle, uma vez que aborda todos os temas apresentados nas disciplinas, desde modelagem de sistemas até projeto de controladores PID. Os modelos em MATLAB e Simulink facilitam o entendimento de conceitos que são bastante complexos para iniciantes no estudo de Controle, uma vez que podem facilmente gerar gráficos e diagramas úteis como os diagramas de Bode e Nyquist, além de permitir o projeto de controladores a partir dos parâmetros do sistema e a verificação da eficácia de métodos empíricos como o de Ziegler-Nichols.

Bibliografia

1. Xue, Dingyu, YangQuan Chen, and Derek P. Atherton. *Linear feedback control: analysis and design with MATLAB*. Vol. 14. Siam, 2007.

2. Callier F. M., Desoer C. A. *Multivariable Feedback Systems*. New York: Springer Verlag, 1982

3. Postlethwaite I., MacFarlane A. G. J. A *Complex Variable Approach to the Analysis of Linear Multivariable Feedback Systems*. Berlin: Springer-Verlag, 1979

4. Vardulakis A.I.G. *Linear Multivariable Control — Algebraic Analysis and Synthesis Methods*. Chichester, England: John Wiley & Sons, 1991

5. Ziegler J.G., Nichols N.B. *Optimum settings for automatic controllers*. Transactions of the ASME, 1942, 64:759–768

6. Bode H. W. *Network Analysis and Feedback Amplifier Design*. Princeton, NJ: Van Nostrand, 1945

7. Pontryagin L. S., Boltyanskii V. G., Gamkrelidze R. V., Mischenko E. F. *The Mathematical Theory of Optimal Processes*. New York: Interscience Publishers, 1962.

8. Moore B. *Principal component analysis in linear systems: controllability, observability, and model reduction*. IEEE Transactions onAutomatic Control, 1981, 26:17–32

9. Lucas T. N. *Some further observations on the differential method of model reduction*. IEEE Transactions on Automatic Control, 1992, 37:1389–1391

10. Stahl H., Hippe P. Comments on "FF-Padé method of model reduction in frequency domain." IEEE Transactions on Automatic Control, 1988, 33:415–416

11. *Using Simulink Version 4.1*. The MathWorks, Natick, MA, 2001

12. *Bennett S. Development of the PID controllers*. IEEE Control Systems Magazine, 1993, 13(2):58–65