



Laboratório de Engenharia de Controle

Rafael Cavalcanti Neto

Departamento de Engenharia Elétrica
Centro de Tecnologia e Geociências
Universidade Federal de Pernambuco



Abril, 2024



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

Copyright ©2024 Universidade Federal de Pernambuco

[WWW.HTTPS://WWW.UFPE.BR/DEE](https://www.ufpe.br/dee)

Primeira Edição, April 30, 2024

Contents

1	MATLAB para Engenharia de Controle	1
1.1	Introdução ao MATLAB	1
1.1.1	Interface	1
1.1.2	Operações Básicas	2
1.1.3	Operações com Vetores	3
1.1.4	Operações com Matrizes	4
1.1.5	Gráficos e Figuras	7
1.1.6	Controle de Fluxo	15
1.2	Ferramentas do MATLAB para Eng. de Controle	18
1.2.1	Função de Transferência	18
1.2.2	Funções do MATLAB para Sistemas de Controle	20
1.2.3	Simulink	25
	References	29

MATLAB para Engenharia de Controle

1.1 | Introdução ao MATLAB

1.1.1 | Interface

Compreender as diferentes maneiras de programação no ambiente MATLAB é o primeiro passo para um aproveitamento completo do *software*. A programação pode ser realizada das seguintes formas:

- Modo Imediato: cada linha de comando é executada após o <enter> na [janela de comandos](#);
- Arquivo de comandos: É criado um arquivo .m, composto por uma série de instruções a serem executadas linha a linha em sequência, as variáveis definidas no arquivo de comandos farão parte do [espaço de trabalho](#), assim como no modo imediato;
- Arquivo de funções: Apesar de ser também um arquivo .m, ele deve ser visto como uma caixa preta, onde são fornecidos os argumentos de entrada e são obtidos os resultados, até por isso, o MATLAB define uma nova área de memória para armazenar as variáveis, evitando choque com as variáveis do espaço de trabalho. Para utilizá-lo, deve-se seguir uma sintaxe específica para que o MATLAB identifique o arquivo como uma função. O nome da Função deve ser o mesmo do arquivo (sem o .m) e a primeira linha de comando do arquivo é a declaração da função, como mostra o comando abaixo.

```
1 function [y1,y2,y3] = nomedaFuncao(x1,x2,x3)
```

Ainda no contexto dos arquivos, é válido mencionar que as variáveis do espaço de trabalho não persistem após o MATLAB ser fechado, por isso, existem os .mat ou MAT-File, que são binários que armazenam as variáveis do espaço de trabalho, caso exista a intenção de utilizá-las posteriormente. Através do comando a seguir é possível salvar as variáveis atuais do espaço de trabalho no .mat.

```
1 save('variaveis');  
2 %Criando um arquivo 'variaveis.mat'
```

1.1.2 | Operações Básicas

Primeiramente, é importante esclarecer o uso do ';' no final de uma linha de comando. Quando não utilizado, o resultado da operação é exibido na janela de comandos, portanto, em um arquivo de comandos é de boa prática utilizar sempre o ';', exceto, quando deseja-se verificar algum resultado.

Confere-se a seguir as principais formas de declaração de variáveis numéricas de diferentes tipos:

Forma Numérica	Declaração MATLAB	Tipo
1,7	1.7	Números decimais
$33,3 \cdot 10^{-3}$	33.3e-3	Notação científica
$4 + 3 \cdot i$	4+3*i	Complexos em notação algébrica
$3 \cdot e^{\frac{i\pi}{6}}$	3*exp(i*pi/6)	Complexos na forma fasorial

Observação: apesar da variável i já estar definida como **i=sqrt(-1)**, é possível utilizar outra denominação. Em circuitos elétricos, a letra 'j' é utilizada como unidade dos números imaginários, para isso, basta declarar:

```
>> j=sqrt(-1)
      0 + 1.000i
```

Seguindo a lógica mencionada sobre as funções no MATLAB, é válido destacar algumas funções simples que serão utilizadas com certa frequência. Estas são: cos(x), sin(x), tan(x), log(x), exp(x), sqrt(x), todas estas funções recebem uma única entrada numérica e retornam uma saída única.

Além das funções pré-definidas, o MATLAB utiliza os símbolos tradicionais (+ - * / ^) nas operações matemáticas. Quando múltiplas operações estiverem envolvidas, a ordem das mesmas deve ser estabelecida através do uso dos parênteses.

A seguir, confere-se os operadores relacionais e lógicos no MATLAB e como são aplicados. Operadores relacionais trabalham em comparações, igualdades e desigualdades, onde o resultado retornado é booleano (verdadeiro ou falso). Já operadores lógicos são utilizados quando são necessárias expressões condicionais complexas.

<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
==	Igual
~=	Diferente

&	AND
	OR
~	NOT

1.1.3 | Operações com Vetores

O MATLAB permite a declaração de vetores de diferentes formas, como exibido a seguir:

```
a=[1, 2, 3] % Vetor linha: elementos sao separados por virgulas ou
            espacos
b=[4; 5; 6] % Vetor Coluna: elementos sao separados por ponto e
            virgula
```

Também é possível obter um vetor coluna através da transposta de um vetor linha, utilizando o operador (`'`). Atenção na utilização do operador de transposição, pois, quando utilizado em variáveis complexas, o resultado retornado será o seu complexo conjugado.

```
>> c=[4, 5, 6]'; % Vetor igual ao vetor 'b'

>> i'

    0 - 1.000i
```

Para criar um vetor onde os elementos são igualmente dispostos, o comando é **(a:step:b)**, onde **a** é o número inicial, **b** é o final e **step** é o intervalo entre os elementos, o passo, caso o passo seja omitido, o valor padrão é 1. Esta técnica é especialmente importante para definição de intervalos de tempo. Outra maneira de se obter vetores deste gênero é através do comando **linspace(a, b, n)**, que retorna um vetor de **n** elementos igualmente espaçados entre **a** e **b**.

```
>> d = 1:3

    1    2    3

>> e = 9:-2:0

    9    7    5    3    1

>> f = linspace(0,6,4)

    0    2    4    6
```

Definindo vetor $a = [a_1 \ a_2 \ \dots \ a_n]$, vetor $b = [b_1 \ b_2 \ \dots \ b_n]$ e c um escalar qualquer. A tabela a seguir apresenta as operações aritméticas com vetores e seus respectivos resultados. As operações a seguir só são possíveis considerando vetores de mesmo tamanho e orientação (linha ou coluna).

Operação	Expressão	Resultado
Adição escalar	$a + c$	$[a_1 + c, \ a_2 + c, \ \dots, \ a_n + c]$
Adição vetorial	$a + b$	$[a_1 + b_1, \ a_2 + b_2, \ \dots, \ a_n + b_n]$
Multiplicação escalar	$a * c$	$[a_1 * c, \ a_2 * c, \ \dots, \ a_n * c]$
Multiplicação vetorial	$a .* b$	$[a_1 * b_1, \ a_2 * b_2, \ \dots, \ a_n * b_n]$
Divisão	$a ./ b$	$[a_1 / b_1, \ a_2 / b_2, \ \dots, \ a_n / b_n]$
Potenciação (vetor-escalar)	$a.^c$	$[a_1^c, \ a_2^c, \ \dots, \ a_n^c]$
Potenciação (escalar-vetor)	$c.^a$	$[c^{a_1}, \ c^{a_2}, \ \dots, \ c^{a_n}]$
Potenciação (vetor-vetor)	$a.^b$	$[a_1^{b_1}, \ a_2^{b_2}, \ \dots, \ a_n^{b_n}]$

1.1.4 | Operações com Matrizes

A maneira mais básica de declarar uma matriz no MATLAB é através do comando a seguir, onde cada linha é delimitada pelo uso do (;) e os elementos da linha são separados por vírgulas ou espaços.

```
>> A = [1, 2, 3; 4, 5, 6] % Matriz A 2x3
      1     2     3
      4     5     6
```

Existem diversas formas de acessar ou atribuir valores em uma posição específica ou em partes da Matriz, confere-se algumas destas a seguir:

B(2,:)	Linha 2 da matriz B
B(:,3)	Coluna 3 da matriz B
B(1:3,:)	Linhas 1, 2 e 3 da matriz B
A(2,1:2)	Vetor Linha [B(2,1) B(2,2)];
B(1:4,3)	Vetor coluna [B(1,3); B(2,3); B(3,3); B(4,3)]
B(2:3,4:5)	Matriz [B(2,4) B(2,5); B(3,4) B(3,5)]
B(5,3)	Elemento B(5,3)
B(:)	Vetor coluna das colunas de B "empilhadas" em ordem

```
>> A(:) == [1, 4, 2, 5, 3, 6] '
      True
```

Atenção ao inserir elementos complexos em uma matriz no MATLAB. Quando deseja-se incluir números como $1 + \sqrt{3}i$ não se declara utilizando espaços, pois o espaço indica separação entre elementos na matriz, portanto, deve ser feito assim **1+sqrt(3)*i**.

```
>> B = [1, 2, 1+sqrt(3)*i; 0, 0, 0];
```

Visto que o operador de transposição (') pode indicar tanto a transposta de uma matriz, quanto o conjugado de um número complexo, deve-se tomar alguns cuidados ao transpor matrizes com elementos complexos. Exemplo:

```
>> C = [1, i; -i*5, 2]
      1     i
     -5i     2

>> D = C '
      1     5i
     -i     2
```

Nota-se que a Matriz foi transposta incorretamente, pois, apesar do deslocamento correto dos elementos, os números complexos foram conjugados. Para realizar uma transposição não conjugada, deve-se utilizar um dos dois comandos a seguir:

G.' ou conj(G')

Sendo M uma matriz qualquer, confere-se a seguir outras funções de matrizes disponíveis no MATLAB:

inv(M)	Matriz inversa
pinv(M)	Matriz pseudoinversa
eye(l,c)	Matriz (lxc) Identidade
zeros(l,c)	Matriz (lxc) de zeros
ones(l,c)	Matriz (lxc) de 1's
rand(l,c)	Matriz (lxc) de números aleatórios entre [0,1]
diag(M)	Matriz quadrada com os elementos de M na diagonal

```
>> [l,c]=size(A); %Retorna o vetor das dimensoes de A
```

```
>> E = eye(3) % Matriz Identidade 3x3
```

```
1    0    0
0    1    0
0    0    1
```

```
>> F = eye(2,3) % Preenche a diagonal principal
```

```
1    0    0
0    1    0
```

```
>> G = zeros(2,5)
```

```
0    0    0    0    0
0    0    0    0    0
```

```
>> H = ones(3,2)
```

```
1    1
1    1
1    1
```

Adição e subtração: É possível somar e subtrair matrizes de mesmas dimensões, onde a operação será executada elemento a elemento de mesma posição relativa.

```
>> I = [2, 4; 6, 8]
```

```
2    4
6    8
```

```
>> J = [1, 0; 3, -1]
```

```
1    0
3   -1
```

```
>> K = I + J
      3      4
      9      7
```

Multiplicação: A multiplicação entre matrizes é realizada normalmente e é possível apenas se o número de linhas da primeira matriz for igual ao número de colunas da segunda matriz. Também é possível multiplicar matrizes por um escalar. Por fim, existe a multiplicação elemento a elemento de matrizes de mesmas dimensões (linha e coluna), esta, é definida pelo comando **(A.*B)**.

```
>> L = [2, 4, 7; 6, 8, 1]
      2      4      7
      6      8      1

>> M = [1, 0; 3, -1; 4, 6]
      1      0
      3     -1
      4      6

>> N = L*M % Multiplicacao matricial
      42      38
      34     -2

>> O = 0.5*N % Multiplicacao escalar
      21      19
      17     -1
```

Divisão: A divisão matricial (**A/B**) corresponde à operação (**A*inv(B)**). E assim, como na multiplicação, também existe a divisão elemento a elemento de matrizes, definida similarmente como: (**A./B**), para matrizes de mesmas dimensões.

Potenciação: elevar uma matriz a n-ésima potência (A^n), sendo n um escalar inteiro maior que 1, significa realizar n-1 multiplicações matriciais de $A*A$. Exemplo: A^3 equivale a $A*A*A$. Além da potenciação padrão, pode-se realizar a potenciação elemento a elemento de 2 matrizes de mesmas dimensões ($A.^B$), nesse caso, cada elemento de A é elevado às potências correspondentes em B. Finalmente, existe a potenciação elemento a elemento de uma matriz com um escalar ($A.^c$), onde cada elemento é elevado à potência do escalar c.

```
>> P = 0^3
      22504      14136
      12648      6136

>> R = 0.^[2, 0; 3, 1]
      441      1
```

```

        4913    1

>> S = 0.^2
        441    361
        289    1

```

Outro tipo especial de vetor são os vetores de células, estes, são aqueles cujos elementos são endereços de memória, cada célula aponta para um endereço de memória e pode armazenar qualquer tipo de dado (números, strings, matrizes, até mesmo outros vetores de células, etc). O comando a seguir cria uma matriz 2x3 de células vazias:

```

>> Z = cell(2,2)
        []    []
        []    []

```

Os comandos $[Z(i, j) = \{x\}]$ e $[Z\{i, j\} = x]$ são duas maneiras de armazenar o conteúdo x no endereço de memória apontado pela posição (i, j) . O primeiro comando (*cell indexing*) funciona da seguinte forma, a célula de conteúdo x estará na posição i, j do vetor de células Z . Já o segundo comando (*content addressing*) atribui que a célula na posição i, j de Z aponte para o conteúdo x .

```

1 Z(1,1) = {2+3i};           % cell indexing
2 Z(1,2) = {12:-2:0};       % cell indexing
3 Z{2,1} = 'Uma string';    % content addressing
4 Z{2,2} = {12:-2:0};       % content addressing

```

```

>> Z =
        [2.0000+ 3.0000i]    [12 10 8 6 4 2 0]
        'Uma string'        1x1 cell

```

Para acessar o conteúdo apontado por uma célula é através do *content addressing*. Confere-se a seguir a diferença ao utilizar o *cell indexing*.

```

>> Z{2,2}
        12    10    8    6    4    2    0

>> Z(2,2)
        {1 1    cell}

```

1.1.5 | Gráficos e Figuras

O MATLAB oferece uma variedade de ferramentas para visualização de dados, tanto contínuos quanto discretos. O procedimento padrão para plotar uma função contém 3 passos: define-se o intervalo de valores a ser plotados; define-se a função; por fim, é passado o comando para plotagem da função. A função mais básica relativa à plotagem de figuras é o comando **plot(x,y,'opcionais')**.

No exemplo a seguir, como previamente explicado, são utilizados vetores para armazenar as coordenadas dos pontos e assim plotar a função desejada. A Figura 1.1 apresenta o *output* do código mencionado.

```

1 x=[1,2,3];
2 y=[0,2,1];
3 plot(x,y) % pontos ligados por uma linha

```

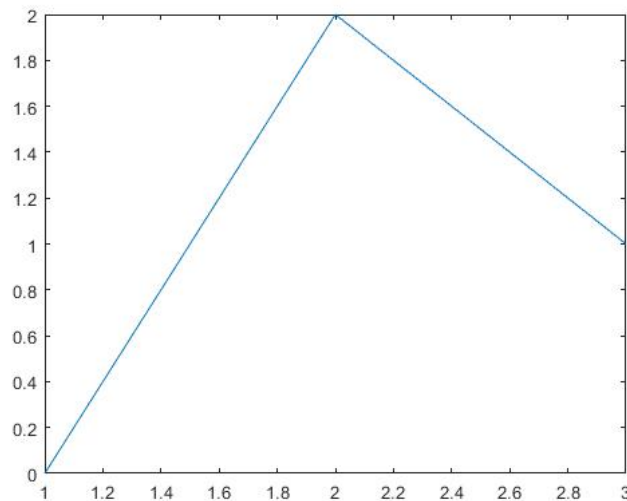


Figure 1.1: Plot de 3 pontos genéricos ligados por linha

A função **plot(x,y,'opcionais')** permite algumas personalizações na exibição das curvas plotadas, é possível configurar o estilo da linha e/ou dos pontos, assim como a cor. Algumas possibilidades podem ser conferidas na tabela a seguir, onde a *string* que determina os opcionais pode conter um caractere de cada coluna. Para mais informações deve-se conferir a [documentação oficial do MATLAB](#).

Cor	Marcação do Ponto	Estilo da Linha
b (azul)	. (ponto)	- (sólida)
g (verde)	o (circulo)	: (pontilhada)
r (vermelho)	x (marcador x)	- (tracejada)
c (ciano)	+ (marcador +)	-. (traço-ponto)
m (magenta)	* (marcador *)	
y (amarelo)	s (quadrado)	
k (preto)	d (diamante)	
w (branco)	^(triângulo p/ cima)	
	v (triângulo p/ baixo)	
	p (pentagrama)	
	h (hexagrama)	

Outros comandos e funções que complementam a apresentação de figuras no MATLAB estão exibidas na Tabela a seguir, sendo possível adicionar título, rótulos nos eixos, linhas de grade, etc.

O comando *hold on* assim como as variantes *hold off* e *hold all* serão explicadas mais detalhadamente a seguir.

<code>xlabel('string')</code>	Rótulo de texto no eixo x
<code>ylabel('string')</code>	Rótulo de texto no eixo y
<code>title('string')</code>	Título da Figura
<code>grid</code>	adiciona linhas de grade
<code>axis on</code>	ativa a exibição dos eixos
<code>axis off</code>	desativa a exibição dos eixos
<code>axis square</code>	Torna a Figura quadrada
<code>axis equal</code>	Torna a unidade de medida dos eixos iguais
<code>axis ([xmin xmax ymin ymax])</code>	Define os limites dos eixos
<code>hold on</code>	Segura a exibição do gráfico atual
<code>legend('primeiro', 'segundo', ...)</code>	Define a legenda das curvas em ordem

Agora, será apresentada maneiras distintas para exibir diferentes curvas no mesmo gráfico, assim como também será explicado como obter diferentes gráficos na mesma Figura e por fim, como gerar Figuras distintas no mesmo bloco de código. **Múltiplas curvas - mesmo gráfico:** o padrão do MATLAB é exibir para o usuário apenas a última plotagem, deletando as anteriores, com isto em mente, existem 3 maneiras principais de contornar este problema:

Primeiro, através da passagem de múltiplas curvas no mesmo sistema de coordenadas, em um único comando `plot(x1,y1,"x2,y2",...)`. No exemplo a seguir são plotados os sinais seno e cosseno no intervalo de tempo $[0, 2\pi]$ definindo diferentes personalizações das curvas para melhor distinção.

```

1 t=0:0.2:2*pi;
2 y1=sin(t);
3 y2=cos(t);
4 plot(t,y1,'--r',t,y2,'*b')
5 % '--r': linha tracejada vermelha
6 % '*b': asteriscos azuis
7 grid % ativa o grid

```

As outras 2 maneiras é através do comando **hold**, que, como explicado anteriormente, controla a retenção dos plots quando deseja-se apresentar múltiplas curvas. Este controle pode ser feito através da passagem do comando **hold on** e depois o **hold off**, ativando e desativando a retenção, ou através do **hold all** que vai segurar todos os plots. Confere-se as duas maneiras nas linhas de código a seguir, e o resultado obtido por ambos os métodos é o exibido na Figura 1.3 abaixo.

```

1 t = linspace(-pi,pi);
2 y1 = sin(t);
3 plot(t,y1) % primeiro plot
4 hold on % ativa a retencao
5 y2 = cos(t);
6 plot(t,y2) % segundo plot

```

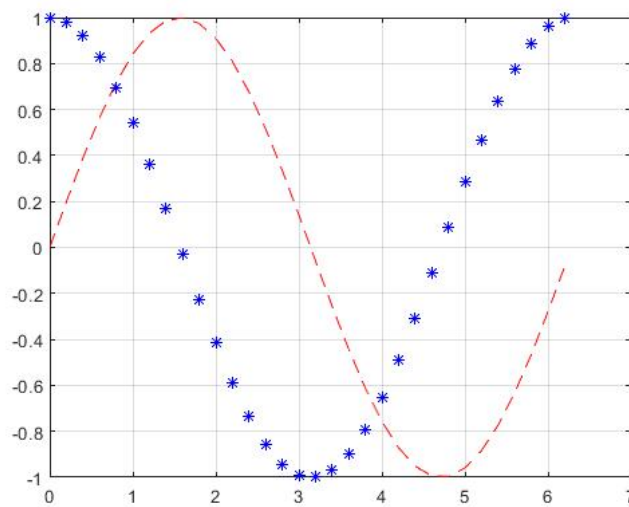


Figure 1.2: Funções seno e cosseno com configurações distintas no intervalo de $[0, 2\pi]$

```

7 hold off      % desativa a retencao

1 t = linspace(-pi,pi);
2 y1 = sin(t);
3 plot(t,y1)    % primeiro plot
4 hold all      % ativa a retencao total
5 y2 = cos(t);
6 plot(t,y2)    % segundo plot

```

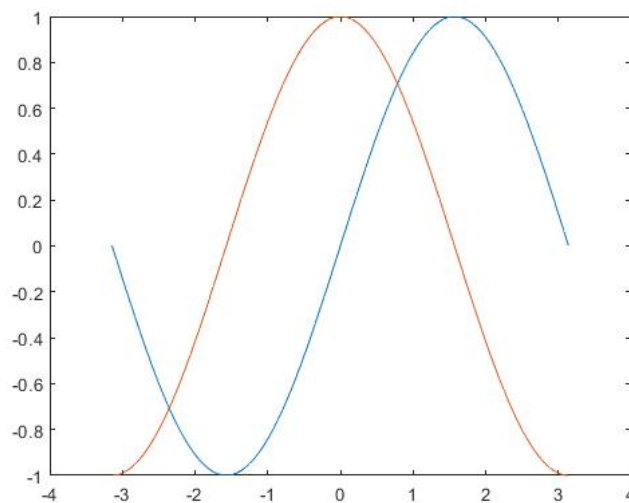


Figure 1.3: Funções seno e cosseno no intervalo de $[-\pi, \pi]$

Através dos 3 métodos anteriores exibe-se 2 curvas distintas no mesmo gráfico e figura, obs: o

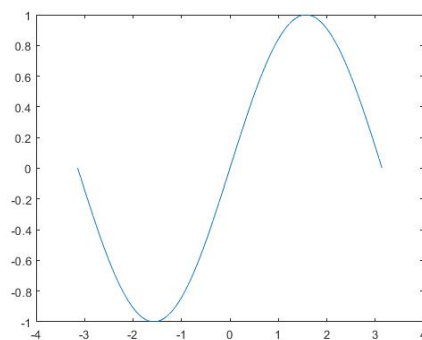
próprio MATLAB irá definir diferentes cores para as curvas mesmo que não sejam definidas pelo usuário.

Figuras distintas - mesmo bloco de código: caso seja desejado apresentar diferentes *plots* em janelas distintas, é necessário passar o comando **figure(n)** que irá criar uma Figura enumerada n em branco. Dessa forma, os plots passados após a declaração do comando serão visualizados na Figura n definida. Confere-se no exemplo a seguir, a plotagem do seno e cosseno em figuras diferentes, o resultado esperado está apresnetado na Figura 1.4 a seguir.

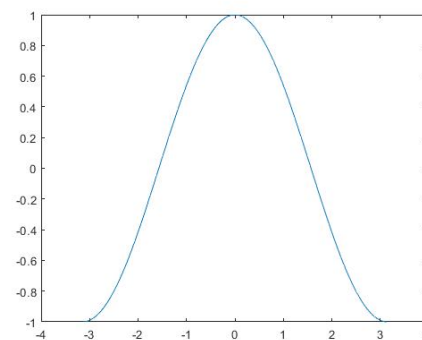
```

1 t = linspace(-pi,pi);
2 y1 = sin(t);
3 y2 = cos(t);
4 figure(1)           % cria-se a Figura 1
5 plot(t,y1)          % plot na Figura 1
6 figure(2)           % cria-se a Figura 2
7 plot(t,y2)          % plot na Figura 2

```



(a) Seno



(b) Cosseno

Figure 1.4: Funções trigonométricas plotadas em figuras distintas no intervalo de $[-\pi, \pi]$

Gráficos distintos - mesma Figura: Na situação onde queira-se apresentar diferentes gráficos em uma mesma figura, o MATLAB disponibiliza o comando **subplot(n,m,i)**, responsável por criar uma figura e em 'n' linhas e 'm' colunas. Cada novo comando de **plot** será visualizado na 'sub-figura' na posição i, onde a posição é contada da esquerda para a direita e de cima para baixo. No código a seguir está evidenciada a criação de uma figura dividida em 4 partes, onde cada sub-figura plotada é acompanhada de um título a fim de facilitar a compreensão dos conceitos. A Figura 1.5 exhibe o resultado fornecido pelo MATLAB.

```

1 t=0:0.2:2*pi;
2 subplot(2,2,1) % sub-figura 1
3 plot(t,cos(t))
4 title('cosseno')
5 subplot(2,2,2) % sub-figura 2

```

```

6 plot(t,sin(t))
7 title('seno')
8 subplot(2,2,3) % sub-figura 3
9 plot(t,exp(t))
10 title('exponencial')
11 subplot(2,2,4) % sub-figura 4
12 plot(t,log(t))
13 title('log')

```

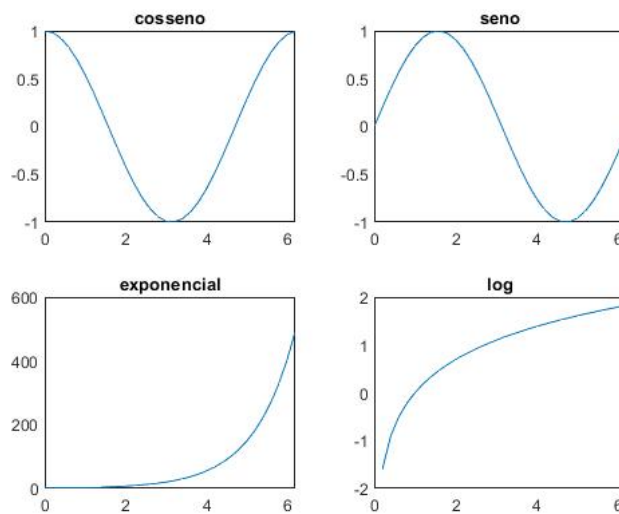


Figure 1.5: Funções no tempo subdivididas na Figura do MATLAB

Todos os procedimentos anteriores descritos para o **plot(x,y)** são análogos aos comandos indicados na Tabela a seguir. Estes comandos funcionam do mesmo modo que o **plot()** e vão apenas afetar a escala dos eixos e consequente a disposição dos dados na figura retornada.

Comando	Output
plot(x,y)	gráficos lineares x-y
loglog(x,y)	gráficos em escala logarítmica x-y
semilogx(x,y)	eixo x em escala logarítmica e eixo y linear
semilogy(x,y)	eixo y em escala logarítmica e eixo x linear

Dados imaginários (números complexos): Sendo **z** um vetor de números complexos, **plot(z)** é equivalente a **plot(real(z),imag(y))**, ou seja, os dados do vetor complexo serão plotados no plano imaginário.

Além das funções mencionadas, que são mais generalistas e vão ser usadas na maior parte do tempo, o MATLAB disponibiliza outras funções de plotagem de dados mais direcionadas e específicas. Por exemplo:

Com o intuito de melhorar a apresentação e entendimento das figuras, é possível adicionar textos com diferentes caracteres especiais em gráficos no MATLAB. Na tabela a seguir são ap-

scatterplot	Gráficos de dispersão
bar	Gráficos de barra
piechart	Gráficos de pizza
outros tipos	

representados alguns caracteres e como devem ser referenciados. Confere-se a tabela completa na [documentação do MATLAB](#).

Símbolo	Comando	Símbolo	Comando
α	<code>\alpha</code>		
β	<code>\beta</code>		
γ	<code>\gamma</code>		
δ	<code>\delta</code>	Δ	<code>\Delta</code>
ω	<code>\omega</code>	Ω	<code>\Omega</code>
σ	<code>\sigma</code>	Σ	<code>\Sigma</code>
ϕ	<code>\phi</code>	Φ	<code>\Phi</code>
ψ	<code>\psi</code>	Ψ	<code>\Psi</code>
θ	<code>\theta</code>	Θ	<code>\Theta</code>
ζ	<code>\zeta</code>	π	<code>\pi</code>

No código abaixo é apresentado o procedimento para inserir o ' π ' nos *ticks* da figura. A Figura 1.6 apresenta os *ticks* inseridos no eixo x do gráfico.

```

1 t = linspace(-3*pi,3*pi);
2 y1 = sin(t);
3 y2 = cos(t);
4 plot(t,y1)
5 hold all
6 plot(t,y2)
7 %procedimento para caracteres especiais nos ticks:
8 xticks([-3*pi, -2*pi, -pi, 0, pi, 2*pi, 3*pi])
9 xticklabels({'-3\pi', '-2\pi', '-\pi', '0', '\pi', '2\pi', '3\pi'})

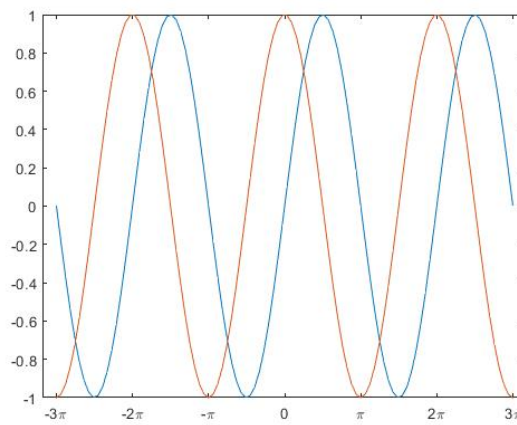
```

Para finalizar o subtópico, foi idealizada a seguinte sequência de comandos com o intuito de ilustrar como o tempo de amostragem pode influenciar a visualização de um sinal, o resultado retornado pelo *script* está visível na Figura 1.7 abaixo.

```

1 t1=0:0.05:2*pi;
2 t2=0:0.2:2*pi;
3 t3=0:0.5:2*pi;
4 t4=0:0.8:2*pi;
5 subplot(2,2,1) % sub-figura 1
6 plot(t1,sin(t1))
7 title('intervalo de 0,05')
8 subplot(2,2,2) % sub-figura 2
9 plot(t2,sin(t2))

```

Figure 1.6: Personalização dos *ticks* na eixo x do gráfico

```

10 title('intervalo de 0,2')
11 subplot(2,2,3) % sub-figura 3
12 plot(t3,sin(t3))
13 title('intervalo de 0,5')
14 subplot(2,2,4) % sub-figura 4
15 plot(t4,sin(t4))
16 title('intervalo de 0,8')

```

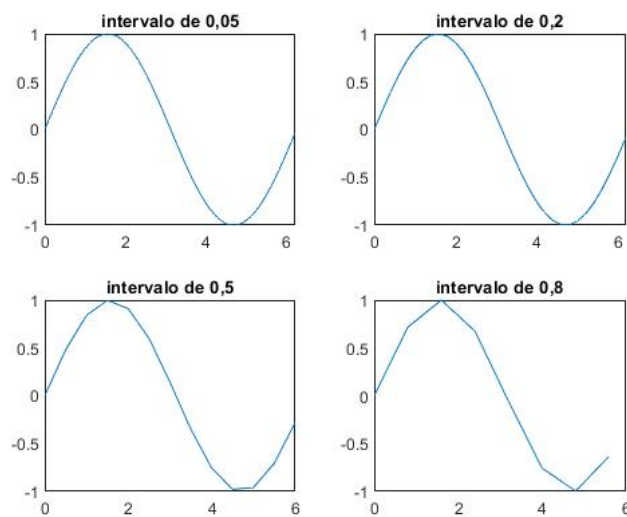


Figure 1.7: Influência do tempo de amostragem na fidedignidade ao sinal

1.1.6 | Controle de Fluxo

À depender do contexto, pode ser necessário um controle maior no fluxo dos comandos, seja através da repetição de parte do código, da adição de condições, ou até da introdução de interrupções. Cada estrutura, laço ou comando vai depender do contexto e da necessidade. Neste sentido, o MATLAB disponibiliza algumas estruturas de controle de fluxo para o usuário:

- estrutura condicional: *if-elseif-else-end*;
- laços: *for* e *while*;
- comandos: *break* e *switch*.

Uma estrutura condicional consiste em avaliar uma ou mais condições lógicas para executar ou não determinadas sequencias de comandos. A estrutura condicional disponibilizada pelo MATLAB é o **if-elseif-else-end** e sua formatação geral pode ser visualizada no bloco de código abaixo. Deve-se atentar ao fato de que uma estrutura condicional pode conter múltiplos blocos **elseif** e que os blocos de **elseif** e **else** são opcionais, além disso, o comando **end** define o limite inferior do corpo da estrutura.

```
if <condição 0>
    <comandos 0>
elseif <condição 1>
    <comandos 1>
elseif <condição 2>
    <comandos 2>
...
else
    <comandos n>
end
```

Observa-se que cada grupo de <comandos> é executado somente quando a <condição> booleana correspondente for verdadeira. As condições são avaliadas em ordem e uma condição só é analisada quando todas as anteriores são falsas. Uma vez que uma condição for verdadeira, as condições subsequentes são desconsideradas e o bloco condicional é encerrado, dessa forma, em uma única estrutura condicional apenas uma lista de <comandos> será executada por iteração.

O laço de repetição **for** permite a execução de uma sequência de comandos um número específico de vezes. Sua estrutura é composta por uma variável de controle (<variavel>) e um vetor <arranjo>, os quais determinam quantas vezes o laço será repetido. A cada iteração do laço, a variável de controle recebe, sequencialmente, os valores do vetor <arranjo>, permitindo a execução dos <comandos> no corpo do **for**. O número de repetições dos <comandos> é igual à quantidade de elementos no vetor <arranjo>. A estrutura descrita está explicitada no bloco de código a seguir.

```
1 for <variavel> = <arranjo>
2     <comandos>
3 end
```

No exemplo a seguir, verifica-se que o laço terá 20 iterações. A variável de controle 'i' assumirá valores de 1 a 5, com incremento padrão de 1, e em cada iteração será calculado um valor para 'x'. Obs: os índices do vetor <arranjo> devem ser **números inteiros positivos** ou valores lógicos.

```
1 for i=1:5
2     x(i) = 2^i;
3 end
```

```
>> x =
      2      4      8     16     32
```

Fazer uma atribuição, como por exemplo $i = 10$, dentro do laço **for** para forçar a saída da estrutura, não funciona. A variável de controle não pode ser redefinida dentro do laço. Além disso, é uma boa prática evitar o uso do **for** quando há uma forma equivalente de resolver o mesmo problema por meio de vetores ou matrizes, conhecida como solução vetorizada, geralmente mais eficiente. Aplicando este tipo de solução no exemplo anterior, é possível substituí-lo pelos seguintes comandos:

```
1 >> i = 1:5;
2 >> x = 2.^i
3      2      4      8     16     32
```

Outro tipo de laço disponível no MATLAB é o comando de repetição condicional **while**, que, ao contrário do **for**, repete os comandos um número indefinido de vezes até que o programador dê a ordem de interrupção. No laço **while**, apenas a condição é testada, ou seja, enquanto a condição for verdadeira, o laço continuará sendo executado. A cada iteração, a condição é reavaliada. O laço é encerrado somente quando o teste da condição resultar em falso, tornando-o uma estrutura de repetição útil para contextos onde é desconhecido o número de iterações desejadas. A sintaxe descrita é da seguinte forma:

```
1 while <condicao>
2     <comandos>
3 end
```

Como explicado anteriormente, no laço **while**, os comandos são repetidos indefinidamente, onde a condição de interrupção é testada no início da rotina. No entanto, em algumas situações, pode ser necessário interromper a execução do laço no meio da estrutura e não somente no seu início. Para solucionar este problema, o MATLAB dispõe do comando **break**, que permite uma saída antecipada de um laço **while** ou **for**. Ao utilizá-lo, o comando interrompe o laço e continua para o próximo comando fora do *loop*. Em casos de múltiplos laços, o comando **break** interrompe a execução do laço mais interno.

No exemplo acima, o comando **break** interrompe o laço **while** apenas quando a 'condição 2' for satisfeita.

```

while <condição 1>
    <comandos 1>
    if <condição 2>
        break
    end
    <comandos 2>
end

```

Por fim, outro comando de controle de fluxo possível, através do MATLAB, é o comando **switch**, uma estrutura condicional baseada em diferentes testes de igualdade para um mesmo argumento. A estrutura geral do comando **switch** está representada na tabela seguir.

```

switch <variavel>
    case <opção 1>
        <comandos 1>
    case <opção 2>
        <comandos 2>
    ...
    otherwise
        <comandos n>
end

```

A estrutura *switch-case* é caracterizada por uma variável (escalar ou *string*) que será testada em igualdade para as diversas opções. Quando um teste de igualdade for verdadeiro, os comandos associados serão executados e o bloco é encerrado. O bloco **switch** testa cada **case** até que uma das expressões seja verdadeira. Em cada **case** a opção pode ser única ou múltipla, caso seja múltipla, as opções devem estar entre chaves e separadas por vírgulas.

No exemplo a seguir, foi desenvolvido um código que solicita ao usuário um tamanho em cm e a unidade em que deseja-se realizar a conversão, a depender da unidade solicitada pelo usuário, o código irá realizar diferentes cálculos para converter o comprimento.

```

1 x = input('Comprimento em cm: ');
2 unidades = input('Digite a unidade: ');
3 switch unidades
4     case {'polegadas','pol'}
5         y=x/2.54;
6     case {'pes','p'}
7         y = x/30.48;
8     case {'metros','m'}
9         y = x/100;
10    case {'milimetros','mm'}
11        y =x*10;
12    otherwise
13        disp(['Unidade desconhecida: ' unidades])
14        y = NaN;

```

15 `end`

```
>> script
Comprimento em cm: 56
Digite a unidade: 'pes'
>> disp(y)
1.8373
```

1.2 | Ferramentas do MATLAB para Eng. de Controle

1.2.1 | Função de Transferência

Uma função de transferência é uma representação matemática de um sistema dinâmico. Ela expressa como a saída se comporta em resposta a uma entrada ao longo do tempo, geralmente em termos do domínio de Laplace. Ela é geralmente representada na forma de uma razão de polinômios, onde as raízes do polinômio do numerador, são os chamados zeros do sistema e as raízes do polinômio do denominador, são os polos do sistema.

Como mencionado, é comum a apresentação da função de transferência ser dada em uma razão de polinômios, a Equação 1.1 a seguir indica este formato de uma FT de maneira genérica. Ao se deparar com este tipo de representação, o MATLAB disponibiliza ao usuário duas abordagens principais, ambas, através da mesma função: `tf(num,den)`. Primeiro, pode-se especificar os polinômios do numerador e denominador como entradas da função `tf(num,den)`, ou segundo, pode-se definir uma variável 's' como variável da função de transferência e definir o sistema por meio de operações matemáticas básicas. Das duas formas, será alcançado um sistema no domínio de Laplace a partir dos parâmetros passados. Confere-se no bloco de código a seguir os dois caminhos explicados.

$$G(s) = \frac{b_ms^m + b_{m-1}s^{m-1} + \dots + b_1s + b_0}{a_ns^n + a_{n-1}s^{n-1} + \dots + a_1s + a_0} = \frac{num(s)}{den(s)} \quad (1.1)$$

```
1 % Q1) Representar: G(s) = (2s + 3) / (s^2 - 5)
2
3 % Metodo 1 - Definindo a razao de polinomios
4 num = [2 3];
5 den = [1 0 -5];
6 sys_m1 = tf(num, den);
7
8 % Metodo 2 - 's' como variavel da FT
9 s = tf('s');
10 sys_m2 = (2*s + 3) / (s^2 - 5);
```

Outra maneira de definir uma função de transferência é através dos polos, zeros e ganho do sistema, para assim, representar a função na sua maneira fatorada. A representação genérica de um sistema no modelo zero-polo-ganho está explícita na Equação 1.2 a seguir (considerando apenas polos simples). O bloco de código subsequente é responsável por demonstrar a aplicação do modelo **zero-polo-ganho** no MATLAB.

$$G(s) = K \frac{(s - z_1)(s - z_2) \dots (s - z_n)}{(s - p_1)(s - p_2) \dots (s - p_n)} \quad (1.2)$$

```

1 % Q1) Representar: G(s) = (s + 3) / ((s - 5)(s+7))
2
3 z = [-3];           %caso nao tenha zeros: z = []
4 p = [5, -7];       %vetor polos
5 k = 1;             %ganho
6 sys = zpk(z,p,k);  %retorna sys no modelo zpk

```

Conhecendo a representação polinomial e a fatorada, é importante inteirar-se como o MATLAB possibilita a conversão de um modelo para outro. O comando **[z,p,k] = tf2zp(num,den)** é responsável por converter a razão de polinômios e retornar as variáveis [z,p,k] de um sistema equivalente. Já o comando **[num,den] = zp2tf(z,p,k)** realiza o inverso, interpreta o sistema zero-polo-ganho e extrai a razão de polinômios. Podem ser conferidas no bloco de código abaixo, as duas conversões de modelos, utilizando como exemplo as FTs exemplificadas anteriormente.

```

1 % polinomial -> fatorada
2 num = [2 3];
3 den = [1 0 -5];
4 [z,p,k] = tf2zp(num,den);
5 sys1 = zpk(z,p,k);
6
7 % fatorada -> polinomial
8 z = [-3];
9 p = [5 -7];
10 k = 1;
11 [num,den] = zp2tf(z,p,k);
12 sys2 = tf(num,den);

```

Apesar do modelo zero-polo-ganho fornecer as informações de maneira completa, o MATLAB dispõe de uma ferramenta adicional, o comando **r = roots(pol)** que retorna as raízes do polinômio 'pol'. Esta funcionalidade pode ser utilizada no contexto de funções de transferência para identificar os polos ou zeros da função. A sequência de comandos a seguir apresenta como a função **roots** pode ser aplicada.

```

1 % Encontre os polos e zeros de G(s) = (s + 3) / (s^2 - 5s + 2)
2 num = [1 3];

```

```

3 den = [1 -5 2];
4
5 zeros = roots(num);
6 polos = roots(den);

```

Por fim, outra ferramenta importante ao lidar com funções de transferência é a expansão em frações parciais. Esta expansão permite decompor uma função de transferência complexa em termos de frações simples, além de explicitar os polos como denominadores das frações parciais. A expansão em frações parciais de uma FT qualquer, considerando apenas polos simples, está destacada na Equação 1.3 a seguir.

$$\frac{B(s)}{A(s)} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + \dots + \frac{r_n}{s - p_n} + k(s) \quad (1.3)$$

No MATLAB, a expansão em frações parciais é realizada a partir da razão de polinômios da FT, e através do comando **[r,p,k] = residue(num,den)** são encontrados os resíduos, polos e a função **k(s)** resultantes. Confere-se nos comandos a seguir um exemplo do uso da função **residue** no MATLAB. A interpretação do resultado apresentado no terminal do MATLAB é a equação 1.4 abaixo.

```

>> num = [-4 8];
>> den = [1 6 8];
>> [r,p,k] = residue(num,den)

r =
    -12
     8
p =
    -4
    -2
k =
    []

```

$$\frac{-4s + 8}{s^2 + 6s + 8} = \frac{-12}{s + 4} + \frac{8}{s + 2} \quad (1.4)$$

É válido ressaltar que para funções de transferência com polos múltiplos a análise é um pouco mais complexa. Nesse caso, os resíduos são passados da menor para a maior multiplicidade. Para maiores dúvidas, consultar a [documentação do MATLAB - Residue](#).

1.2.2 | Funções do MATLAB para Sistemas de Controle

No que diz respeito à sistemas de controle, o MATLAB oferece uma gama de funções e comandos essenciais para estudos e análises de sistemas. Confere-se a seguir, uma lista das principais funções, seus respectivos objetivos e funcionamento.

Sobre as características do sistema:

pole(sys) Retorna os polos da função de transferência de sistema 'sys';

zero(sys) Retorna os zeros da função de transferência de sistema 'sys';

damp(sys) Exibe as frequências naturais, taxas de amortecimento e constantes de tempo de cada polo do sistema 'sys';

pzmap(sys) Cria uma figura mapa dos polos e zeros do modelo, indicando a localização dos polos e zeros, conforme mostrado na Figura 1.8 a seguir;

K=dcgain(sys) Retorna o ganho DC do modelo 'sys'.

```
1 % Características do sistema
2 num = [1, 2];
3 den = [1, 4, 3];
4 sys=tf(num,den);
5 polos = pole(sys); % retorna os polos
6 zeros = zero(sys); % retorna os zeros
7 K = dcgain(sys);   % retorna o ganho DC
8 pzmap(sys);        % Mapa de polos e zeros
9 damp(sys)
```

Resposta ao degrau:

step(sys) Cria uma figura com a resposta ao degrau do sistema 'sys';

step(sys, Tfinal) Cria uma figura da resposta ao degrau do sistema 'sys' no intervalo de tempo [0, Tfinal];

step(sys, t) Plota a resposta ao degrau do sistema 'sys' nos pontos do vetor 't' (a ser definido pelo usuário);

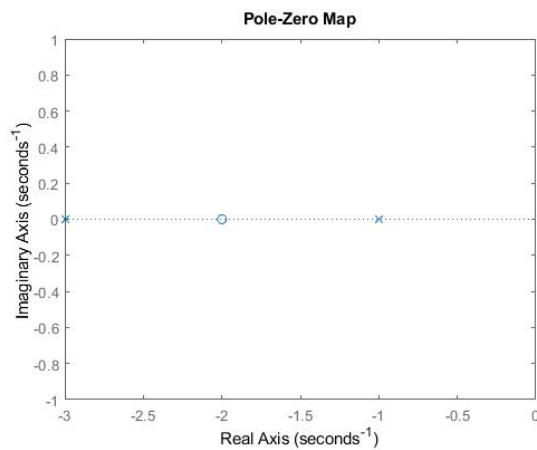
data=stepinfo('sys') Retorna as características da resposta ao degrau: tempo de pico, tempo de assentamento, máximo sobressinal, tempo de subida, etc;

v=step(sys) Salva a resposta em um vetor.

```
1 % resposta ao degrau
2 num = [2];
3 den = [1, 0.5, 2];
4 Tfinal = 18;
5 t = 0:0.8:18;
6 sys=tf(num,den);
7 step(sys, Tfinal, 'r')
```

Workspace	
Name ▲	Value
den	[1,4,3]
K	0.6667
num	[1,2]
polos	[-3;-1]
sys	1x1 tf
zeros	-2

(a) Variáveis



(b) Mapa de polos e zeros

>> Script

Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-1.00e+00	1.00e+00	1.00e+00	1.00e+00
-3.00e+00	1.00e+00	3.00e+00	3.33e-01

(c) Frequências

Figure 1.8: Resultados do script exemplo sobre características do sistema

```

8 hold on
9 step(sys, t, 'xb')
10 hold off
11 stepinfo(sys)

```

```

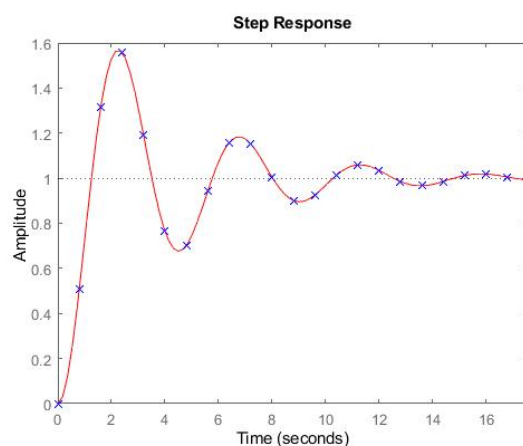
ans =

  struct with fields:

    RiseTime: 0.8441
    SettlingTime: 14.2451
    SettlingMin: 0.6791
    SettlingMax: 1.5675
    Overshoot: 56.7549
    Undershoot: 0
    Peak: 1.5675
    PeakTime: 2.2105

```

(a) Informações da resposta



(b) Resposta ao degrau de 0 a 18s

Figure 1.9: Resultados do script exemplo sobre resposta ao degrau

Resposta ao impulso:

impulse(sys) Cria uma figura com a resposta ao impulso do sistema 'sys';

impulse(sys, Tfinal) Cria uma figura da resposta ao impulso do sistema 'sys' no intervalo de tempo [0, Tfinal];

impulse(sys, t) Plota a resposta ao impulso do sistema 'sys' nos pontos do vetor 't' (a ser definido pelo usuário);

Resposta à entrada arbitrária:

lsim(sys,u,t) Define-se o vetor entrada 'u' e o vetor tempo 't', este comando retorna a resposta do sistema 'sys' à entrada 'u' nos pontos do vetor tempo 't';

lsim(sys,u,t,x0) Resposta do sistema 'sys' à entrada 'u' definindo as condições iniciais 'x0';

v=lsim(sys,u,t) Salva a resposta em um vetor.

```

1 % resposta a uma entrada qualquer
2 sys=tf([1 2],[1 4 3]);
3 t=0:0.1:10;      % vetor tempo
4 u=sin(1.2*t);    % vetor entrada
5 lsim(sys,u,t)    % plot da resposta

```

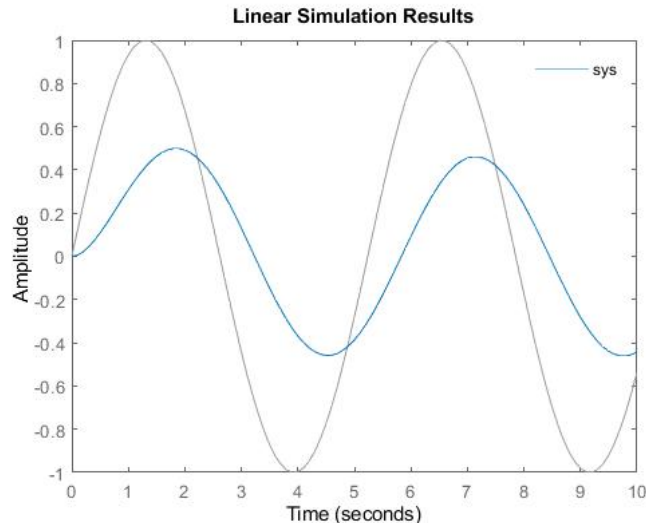


Figure 1.10: Resposta à entrada arbitrária 'u'

Sistemas com atraso:

sys=zpk(z,p,k,'InputDelay',T) É atribuído à variável 'sys' um sistema com atraso no modelo ZPK, 'sys' é o sistema inicial [z,p,k] multiplicado pelo atraso e^{-Ts} ;

sys=tf(num,den,'InputDelay',T) É atribuído à variável 'sys' um sistema com atraso na forma polinomial, 'sys' é o sistema inicial [num,den] multiplicado pelo atraso e^{-Ts} ;

Interconexões do sistema:

sys = series(sys1,sys2,...,sysn) Conexão em série de múltiplos sistemas;

sys = sys1*sys2*...*sysn Conexão em série de múltiplos sistemas;

sys = parallel(sys1,sys2,...,sysn) Conexão em paralelo de múltiplos sistemas;

sys = sys1+sys2+...+sysn Conexão em paralelo de múltiplos sistemas;

sys = feedback(sys1, sys2) Sistema 'sys' com realimentação NEGATIVA, onde 'sys1' é a malha direta e 'sys2' é a realimentação;

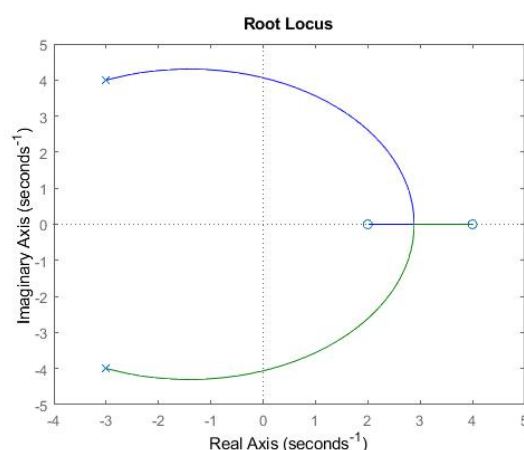
sys = feedback(sys1, sys2, 1) Sistema 'sys' com realimentação POSITIVA, onde 'sys1' é a malha direta e 'sys2' é a realimentação.

Estabilidade do sistema:

isstable(sys) Avalia a estabilidade do sistema 'sys' e retorna um valor lógico, caso sistema estável: 1 (true), caso contrário: 0 (false);

rlocus(sys) Calcula e plota em um figura o lugar geométrico das raízes do sistema 'sys' (assumindo realimentação negativa).

```
1 % Estabilidade do sistema
2 sys=zpk([2 4],[-3+4i -3-4i],1);
3 rlocus(sys)
4 isstable(sys)
```



(a) Lugar Geométrico das Raízes

```
ans =
    logical
     1
```

(b) Estabilidade do Sistema

Figure 1.11: Resultados do script exemplo sobre estabilidade do sistema

Análises na resposta em frequência:

bode(H) Gera o diagrama de bode da função de transferência definida como H;

bode(H, w) onde w pode ser um ponto específico ou um vetor de frequência logarítmica (função logspace)

allmargin(sys) Calcula a margem de ganho, margem de fase, frequências de corte, etc correspondentes do modelo de malha aberta 'sys'.

nyquist(sys) Cria e plota o diagrama de Nyquist do sistema 'sys', onde a frequência assume valores de zero a infinito;

```
1 % Resposta em frequencia
2 sys = zpk([2 -0.2],[1 4],5);
3 figure(1)
4 nyquist(sys)
5 grid on
6 figure(2)
7 bode(sys)
8 grid on
9 allmargin(sys)
```

1.2.3 | Simulink

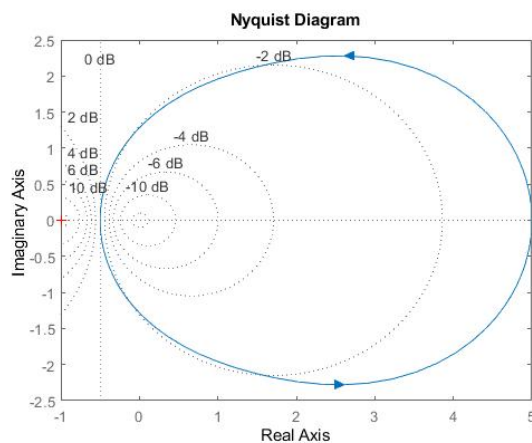
O Simulink é um aplicativo dentro do MATLAB que permite a modelagem, simulação e análise de sistemas por meio de diagramas de blocos. Com o Simulink, é possível representar cada componente de um sistema e todas suas conexões, além das ferramentas de medição e visualização dos dados.

Para acessar o Simulink, basta digitar o comando "simulink" na janela de comandos do MATLAB e o programa será iniciado. Alternativamente, é possível acessar o Simulink através do menu superior do MATLAB, onde existe a opção "Simulink". Clicando nessa opção, o ambiente do Simulink será iniciado.

```
>> simulink
```

Para abrir uma página em branco seleciona-se a opção 'Blank Model'. Com o modelo aberto, é necessário montar o sistema através dos blocos disponíveis no simulink, para isso deve-se abrir a biblioteca clicando no botão 'Library Browser', esta ação abre a janela exibida na Figura 1.13 abaixo.

A janela 'Library Browser' é dividida em duas partes principais, na esquerda estão as categorias de blocos, e na direita são exibidos os blocos da categoria selecionada. Para adicionar um bloco deve-se localizá-lo na biblioteca, clicar arrastar e soltar o bloco no modelo. Ao adicionar um bloco é possível alterar suas propriedades dando dois cliques nele. Cada bloco possui uma entrada e saída, que servem para conectar os blocos entre si, novamente, isto é editado clicando e



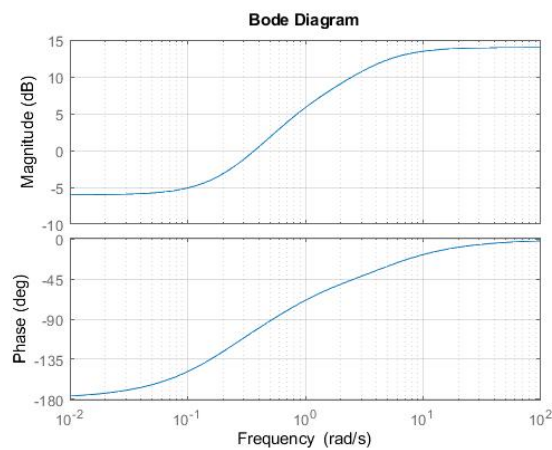
(a) Diagrama de Nyquist

ans =

struct with fields:

```
GainMargin: 2
GMFrequency: 0
PhaseMargin: 76.7916
PMFrequency: 0.3708
DelayMargin: 3.6150
DMFrequency: 0.3708
Stable: 0
```

(b) Informações das margens

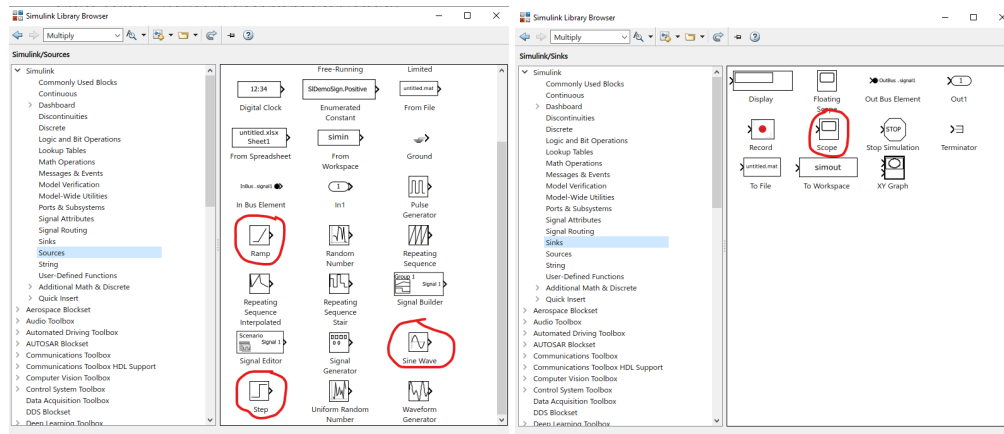


(c) Diagrama de Bode

Figure 1.12: Resultados do script exemplo sobre resposta em frequência

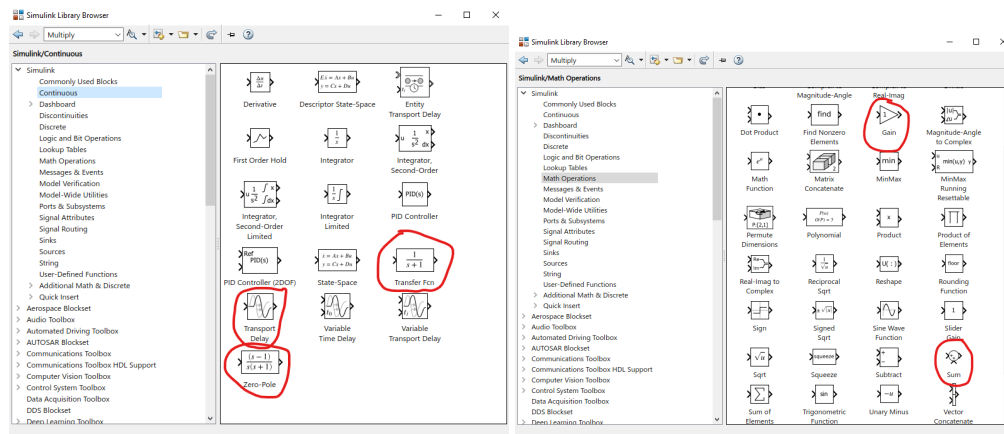
arrastando. A Figura 1.13 acima evidencia os blocos e as bibliotecas mais utilizados em sistemas de controle.

Ao montar o sistema apresentado na Figura 1.14, seguindo os procedimentos descritos, foi possível simular o sistema e captar a resposta ao degrau através do 'Scope'. O sinal obtido está indicado na Figura 1.15 a seguir. Além disso, vale ressaltar que para obter um resultado coerente, deve-se configurar o tempo de simulação assim como o intervalo de amostragem.



(a) Sources

(b) Sinks



(c) Continuous

(d) Math Operations

Figure 1.13: Blocos e bibliotecas mais utilizados no Simulink

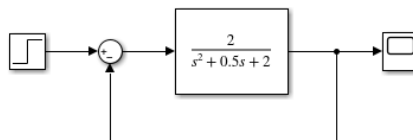


Figure 1.14: Sistema em Malha Fechada

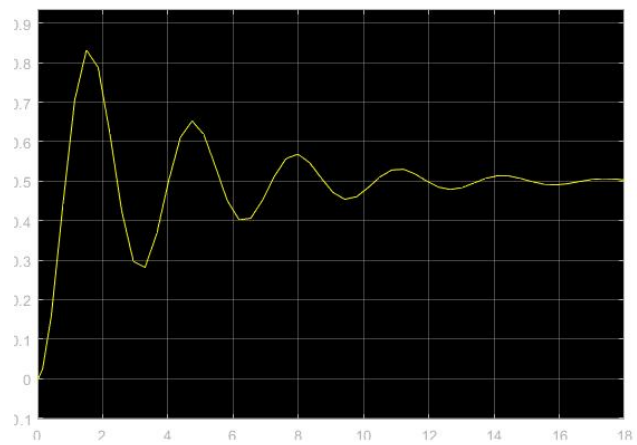


Figure 1.15: Resposta ao degrau do sistema

References