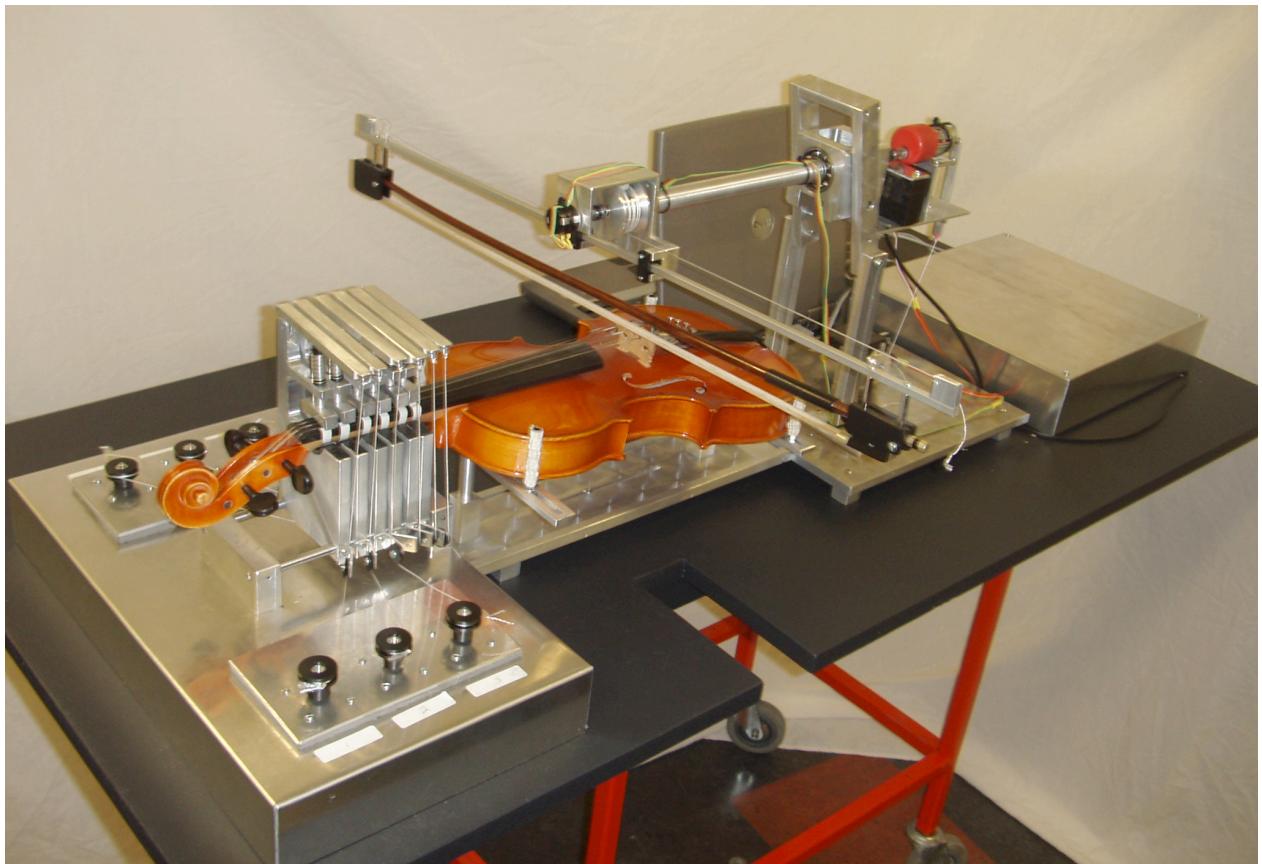


Sistemas e Sinais



Matlab



Prof. Eduardo L. L. Cabral
Prof. Vitor Alex Oliveira Alves

Índice

1. Introdução	1
1.1 Iniciando o MATLAB	1
1.2 Manipulação de matrizes	2
1.2.1 Atividades	4
1.3 Seqüências	5
1.4 Obtendo ajuda	6
1.5 Operações matemáticas	6
1.5.1 Sistemas de equações lineares	7
1.5.2 Atividade	8
1.6 Outras operações com matrizes	8
1.7 Gráficos.....	9
1.7.1 Gráficos tridimensionais	11
1.8 Polinômios	13
1.8.1 Avaliação, multiplicação, divisão e diferenciação..	14
1.9 Funções de transferência.....	14
1.10 Simulações	15
2. Análise de sistemas lineares de 1^a e 2^a ordem – atividades	18
2.1 Sistemas de 1 ^a ordem	18
2.2 Resposta temporal.....	18
2.3 Sistemas de 2 ^a ordem	18
2.4 Sistemas de 2 ^a ordem sem zeros	19
2.5 Validade do Teorema do Valor Final	19
2.6 Aproximações	19
3. Aproximações e estabilidade de sistemas lineares	20
3.1 Sistemas de 3 ^a ordem	20
3.1.1 Exemplo	20
3.2 Aproximações para modelos de 2 ^a ordem.....	20
3.2.1 Exemplos.....	21
3.3 Conexões entre sistemas	21
3.4 Estabilidade.....	22
3.4.1 Exemplos.....	22
4. Programação	24
4.1 Exemplo – análise de um sistema linear	24
4.1.1 Criando um <i>script</i>	24
4.1.2 Análise do <i>script</i>	25
4.1.3 Criando uma função	26
4.1.4 Atividade	26
4.1.5 Convenções	26
4.2 Controle de fluxo	27
4.2.1 Estrutura condicional <i>if</i>	28
Exemplo: 28	
4.2.2 Estrutura repetitiva <i>while</i>	28

4.2.3	Estrutura repetitiva <i>for</i>	29
4.3	Vetorização	29
4.4	Entrada de dados	29
4.5	Edição de funções existentes	30
4.6	Subfunções	30
4.7	Exemplos de aplicação	31
4.7.1	Aproximações	31
4.7.2	Análise do erro em regime estacionário	32
4.7.3	Atividade	33
4.7.4	Atividade – estabilidade em função de um parâmetro	33
5.	Lugar das raízes – introdução	34
5.1	Exemplo	34
5.1.1	Atividade	35
5.2	Análise gráfica do lugar das raízes	35
5.2.1	Atividade	36
5.3	Exemplos	36
6.	Projeto usando lugar das raízes – introdução.....	39
6.1	Exemplo de projeto	39
6.1.1	Atividades	42
7.	Resposta em freqüência – introdução.....	44
7.1	Determinação manual da resposta em freqüência	44
7.1.1	Atividades	44
7.2	Diagrama de Bode	45
7.2.1	Exemplo	45
7.3	Sistemas de 2 ^a ordem	46
7.3.1	Atividades	47
8.	Simulink.....	49
8.1	Exemplo de uso	49
8.1.1	Configurando os parâmetros de simulação	52
8.1.2	Simulando	54
8.2	Atividades	55
8.3	Subsistemas	57
8.3.1	Exemplo – controlador PI	57

Foto da capa: "Motor driven XY table - Series MAXY6000 ". Obtida de: <http://www.unislide.com/images/motor/maxy6012elite.jpg>

1. Introdução

O MATLAB (*Matrix Laboratory*) é um ambiente de programação de alto desempenho voltado para a resolução de problemas que possam ser expressos em notação matemática. Projeto e simulação de sistemas de controle, análise de dados e criação de gráficos são algumas das aplicações possíveis para esta ferramenta. Pacotes específicos, chamados *toolboxes*, permitem a expansão do ambiente de trabalho do MATLAB para a resolução de classes particulares de problemas como processamento de sinais, identificação de sistemas, implementação de redes neurais, lógica difusa (*fuzzy*), simulação, etc. Adicionalmente, um programa gráfico chamado *Simulink*, que trabalha juntamente com o MATLAB, permite a simulação interativa de sistemas dinâmicos lineares ou não-lineares, contínuos ou digitais.

1.1 Iniciando o MATLAB

Execute o MATLAB 6.5 a partir do menu "Iniciar". A tela principal do programa (figura 1.1) contém, em sua visualização padrão, uma janela de comandos (*command window*), uma janela para exibição da área de trabalho (*workspace*), onde ficam armazenadas as variáveis definidas pelo usuário e o histórico de comandos. A janela de comando fornece a principal forma de comunicação entre o usuário e o interpretador MATLAB, que exibe um sinal de prontidão (*prompt*) para indicar que está pronto para receber instruções.

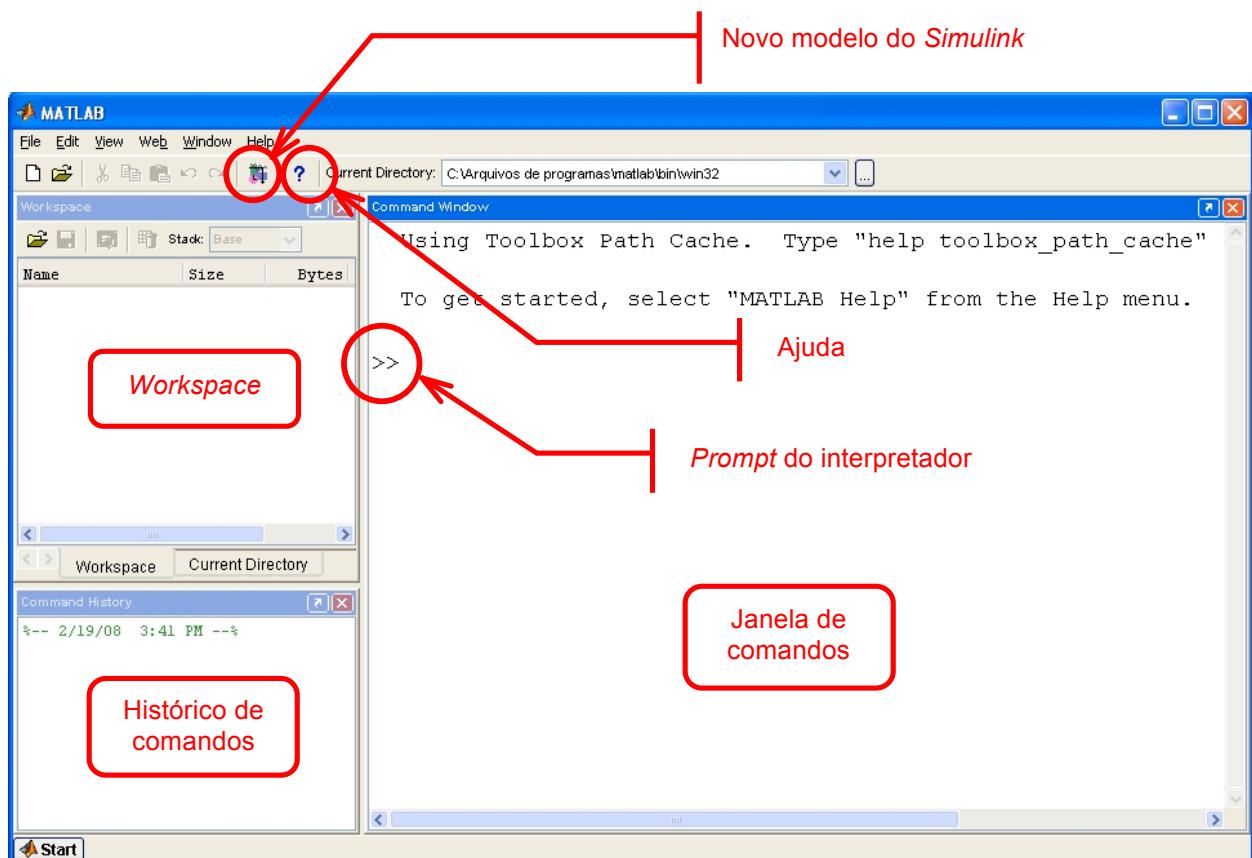


Figura 1.1: Tela principal do MATLAB

A visualização padrão da janela de comando (conforme a figura 1.1) pode ser obtida, a qualquer momento, clicando-se em *View > Desktop Layout > Default*.

Antes de iniciar a sessão de trabalho é conveniente aumentar a fonte da letra usada na janela de comando. Clique em *File > Preferences > Command Window > Fonts & Colors*, selecione a opção "Use custom font" (ver figura 1.2) e ajuste o tamanho da fonte para (no mínimo) 16 pontos. Acredite: muitos erros de digitação podem ser evitados com esta simples providência!

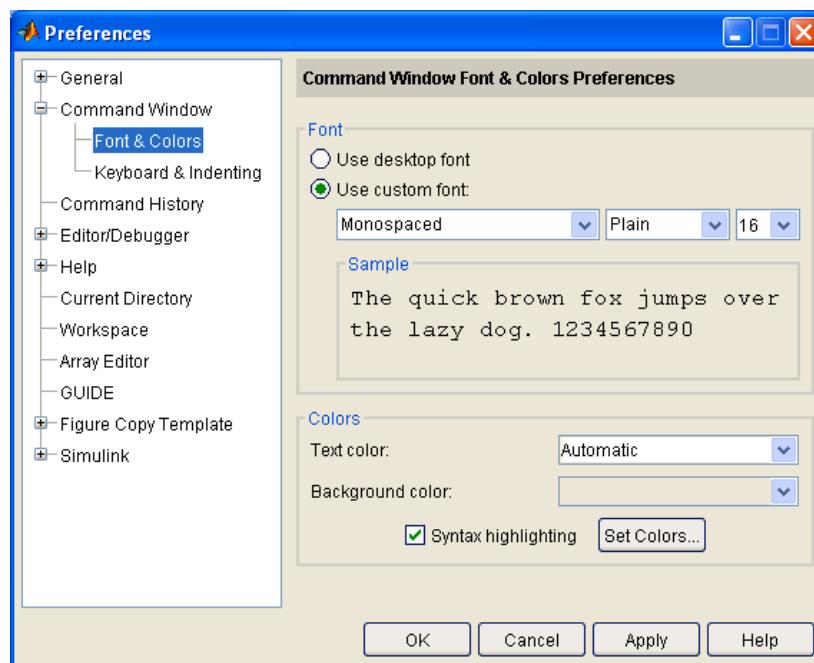


Figura 1.2: Ajuste da fonte usada na janela de comandos

1.2 Manipulação de matrizes

O tipo numérico padrão usado pelo MATLAB é a **matriz** de valores em ponto flutuante: números reais ou complexos são armazenados em matrizes 1x1. A maneira mais simples de se armazenar uma matriz na memória é com uma atribuição, como em:

```
>> A = [2 1 3 4 5] ↵
```

O resultado do comando anterior é mostrado na figura 1.3. Note que o comando passou a fazer parte do histórico do programa e que a matriz foi armazenada no *workspace*. A alocação da matriz também pode ser confirmada pelos comandos **who** (que mostra os nomes das variáveis armazenadas) ou **whos** (que mostra os nomes e espaços ocupados pelas variáveis). Exemplo:

```
>> whos ↵
Name      Size      Bytes  Class
A         1x5        40    double array
Grand total is 5 elements using 40 bytes
```

O MATLAB é **sensível à caixa**, ou seja, diferencia letras maiúsculas de minúsculas e aloca automaticamente o espaço de memória necessário para as variáveis usadas.

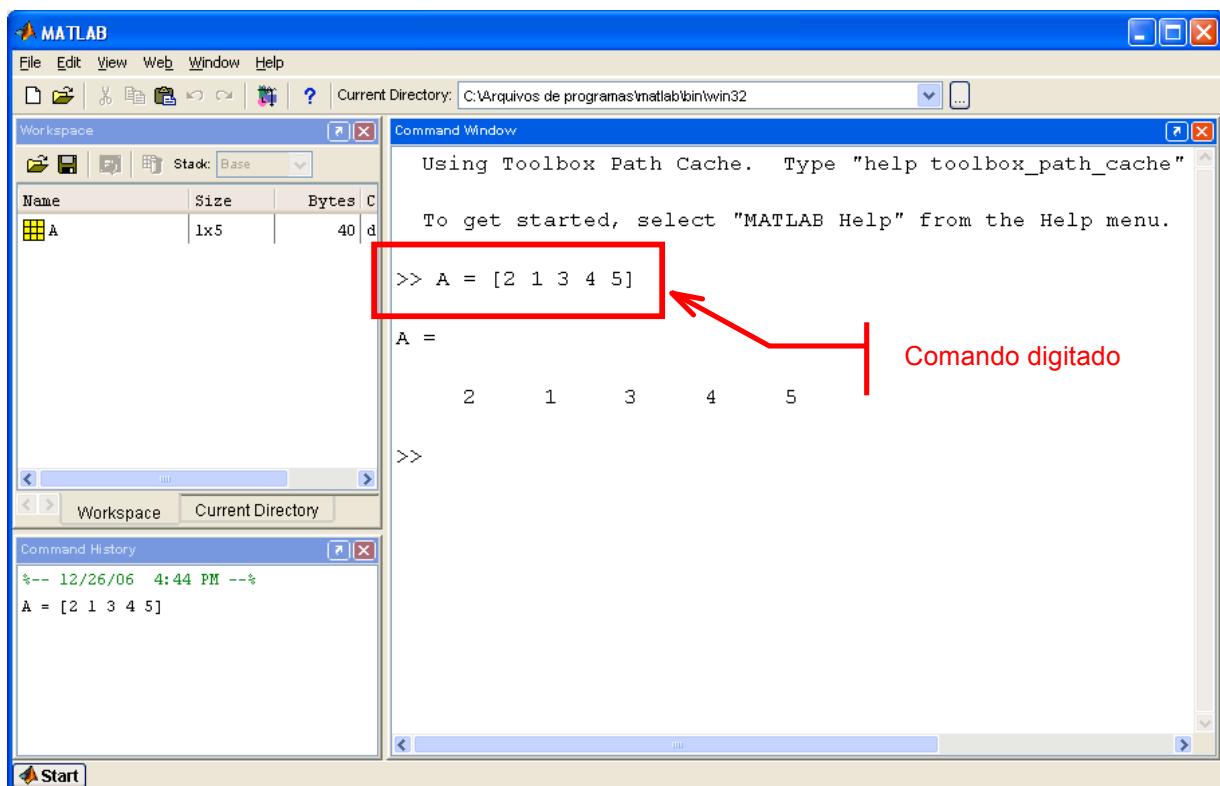


Figura 1.3: Atribuição de valores

A matriz deste exemplo é chamada de **vetor linha**, já que se trata de uma matriz com apenas 1 linha: na digitação, os valores do vetor podem ser separados por espaços, como no exemplo, ou por vírgulas. Para criar um **vetor coluna** deve-se separar cada linha das demais usando **ponto-e-vírgula**. Exemplo:

```
>> B = [5; -4; 6.5] ↵
B =
    5.0000
   -4.0000
    6.5000
```

Para criar uma matriz **bidimensional**¹ basta combinar as sintaxes anteriores:

```
>> M = [2 1 3; 4 6 7; 3 4 5] ↵
M =
    2     1     3
    4     6     7
    3     4     5
```

Quando for interessante omitir a exibição do resultado de qualquer comando basta encerrá-lo com ponto-e-vírgula. Exemplo:

```
M = [2 1 3; 4 6 7; 3 4 5]; % Cria uma matriz 3x3 (3 linhas e 3 colunas)
```

¹ A partir deste ponto, o termo matriz será usado para designar matrizes com mais de uma dimensão.

O símbolo de porcentagem serve para criar comentários de uma linha, tanto na janela de comandos quanto no ambiente de programação do MATLAB.

Os elementos de uma matriz podem ser acessados pelo nome da variável, seguido de índices entre parênteses, sendo que o primeiro elemento é **sempre o de índice 1**. Exemplo de acesso:

```
>> x = B(2) ↴
x =
-4
```

Se uma nova informação for atribuída a um vetor ou matriz os redimensionamentos necessários serão feitos automaticamente. Exemplo:

```
>> A = [4 5 9]; ↴
>> A(6) = 8
A =
4      5      9      0      0      8
```

A red bracket points from the text "Valores inseridos automaticamente" to the value 8 at index 6 of the matrix A.

Para acessar os elementos de uma matriz escreve-se o conjunto de índices entre parênteses, separados por vírgula. Exemplo:

```
>> x = M(2, 3) ↴
x =
7
```

O resultado de qualquer comando que não seja atribuído a uma variável específica é armazenado em uma variável especial chamada **ans**. Exemplo:

```
>> M(2, 1) ↴
ans =
4
```

Para facilitar a repetição de comandos é possível usar as setas para cima e para baixo do teclado ou dar um *duplo-clique* nos itens da janela de histórico. Não existem comandos específicos para desfazer atribuições feitas na janela de comando, apesar de existir a opção *undo* no menu **Edit** do programa.

1.2.1 Atividades

Antes de iniciar as atividades a seguir, limpe a janela de comando digitando **clc**. Em seguida, remova todas as variáveis da memória, usando o comando **clear**. Se quiser eliminar apenas uma variável, use a sintaxe **clear <nome da variável>**. Anote os resultados obtidos.

- a) Armazene a seguinte matriz:

$$M = \begin{bmatrix} 2 & -1 & 5 \\ 3 & 1 & 1 \\ 2 & 0 & 2 \end{bmatrix}$$

b) Obtenha a matriz **transposta** de **M**, digitando:

```
m1 = M'
```

c) Obtenha a matriz **inversa** de **M**, digitando:

```
m2 = inv(M)
```

A utilidade da inversa de uma matriz será discutida futuramente.

d) A indexação pode ser usada em conjunto com o sinal ":" para indicar "todos os elementos" de uma certa dimensão. Por exemplo, o comando a seguir cria um vetor linha com todos os elementos da segunda linha da matriz **M**:

```
v1 = M(2, :)
```

O comando anterior pode ser traduzido como "armazene em **v1** os elementos de **M** que estão na linha 2 e em todas as colunas". Da mesma forma, o comando a seguir cria um vetor coluna com os elementos da primeira coluna da matriz **M**:

```
v2 = M(:, 1)
```

e) O uso de ":" também permite a atribuição de valores a uma dimensão completa de uma matriz. Por exemplo, verifique o efeito da seguinte instrução sobre a matriz **M**:

```
M(1, :) = 5
```

f) Se a atribuição envolver uma **matriz vazia**, indicada por um par de colchetes vazios, é possível eliminar totalmente uma linha ou coluna de uma matriz. Por exemplo, a instrução

```
M(2, :) = []
```

remove a segunda linha da matriz **M**.

g) Finalmente, matrizes podem ser concatenadas por meio de atribuições diretas. Exemplo:

```
m3 = [[5; 5; 5] v2 v1'] % Cria uma nova matriz 3 x 3
```

1.3 Seqüências

O uso de ":" também serve para denotar uma seqüência igualmente espaçada de valores, entre dois limites especificados, inteiros ou não.

Por exemplo, a instrução

```
v3 = 3:8
```

cria um vetor linha com os valores 3, 4, 5, 6, 7 e 8 (o incremento padrão é unitário).

O incremento pode ser definido pelo usuário se a seqüência for criada sob a forma:

[Valor inicial: Incremento: Valor final]

Exemplo:

```
>> v4 = 2:0.5:4 ↵
v4 =
2.0000    2.5000    3.0000    3.5000    4.0000
```

Outra forma de se obter um vetor com valores igualmente espaçados é pelo uso da função ***linspace***. Por exemplo, a instrução

```
y = linspace(10,200,25)
```

gera um vetor linha com 25 valores igualmente espaçados entre 10 e 200. Se o parâmetro que controla o número de pontos for omitido, a seqüência terá 100 valores.

A diferença entre usar o operador ":" e a função ***linspace*** é que a primeira forma exige o **espaçamento** entre os valores enquanto a segunda requer a **quantidade** de valores.

Seqüências com valores linearmente espaçados são usados, normalmente, para fornecer valores de variáveis independentes para funções. Por exemplo, as instruções a seguir criam um vetor com 50 valores da função $y = \sin(x)$, para $x \in [0, 2\pi]$:

```
x = linspace(0,2*pi,50); % 'pi' é uma função interna que retorna o valor de π
y = sin(x);
```

Neste tipo de operação, chamada de *vetorizada*, o MATLAB cria ou redimensiona o vetor **y** com a mesma dimensão do vetor **x**. Em situações que exijam grandes variações de valores, como na análise de respostas em freqüência, é interessante que a variação de valores seja *logarítmica*, o que pode ser obtido com o uso da função ***logspace***, de sintaxe semelhante à de ***linspace***. Por exemplo, a instrução

```
f = logspace(0,4,50);
```

cria um vetor com 50 valores espaçados logaritmicamente entre $10^0 = 1$ e $10^4 = 1.000$.

1.4 Obtendo ajuda

Há diversas maneiras de se obter mais informações sobre uma função ou tópico do MATLAB. Se o nome da função for conhecido pode-se digitar, na janela de comando:

```
help <nome da função> ↵
```

Também é possível fazer uma busca por palavra-chave com o comando ***lookfor***. Por exemplo,

```
lookfor identity ↵
```

retorna uma descrição curta de funções relativas a matrizes identidade. Além dessas formas, pode-se consultar a documentação do MATLAB clicando no ícone de ajuda da janela de comandos.

1.5 Operações matemáticas

O MATLAB reconhece os operadores matemáticos comuns à maioria das linguagens de programação nas operações com escalares (números reais e complexos). Nas operações com matrizes é preciso respeitar as regras da Matemática em relação às dimensões envolvidas. Por exemplo, considere as seguintes matrizes:

$$A = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 3 & 9 \\ 6 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 5 & 5 \\ 4 & 6 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

Em seguida, considere as operações:

- i) $C = A+B$ (Soma)
- ii) $C = A-B$ (Subtração)
- iii) $C = A*B$ (Multiplicação matricial)
- iv) $C = A.*B$ (Multiplicação elemento-a-elemento)
- v) $C = A./B$ (Divisão elemento-a-elemento)
- vi) $C = A.^2$ (Potenciação elemento-a-elemento)

De acordo com as definições da Matemática e as convenções do MATLAB, deve-se obter os seguintes resultados:

$$\begin{array}{lll} i) \quad C = \begin{bmatrix} 6 & 9 & 10 \\ 6 & 9 & 11 \\ 9 & 4 & 6 \end{bmatrix} & ii) \quad C = \begin{bmatrix} -4 & -1 & 0 \\ -2 & -3 & 7 \\ 3 & -4 & -4 \end{bmatrix} & iii) \quad C = \begin{bmatrix} 36 & 49 & 38 \\ 49 & 64 & 61 \\ 33 & 34 & 35 \end{bmatrix} \\ iv) \quad C = \begin{bmatrix} 5 & 20 & 25 \\ 8 & 18 & 18 \\ 18 & 0 & 5 \end{bmatrix} & v) \quad C = \begin{bmatrix} 0,2 & 0,8 & 1,0 \\ 0,5 & 0,5 & 4,5 \\ 2,0 & 0 & 0,2 \end{bmatrix} & vi) \quad C = \begin{bmatrix} 1 & 16 & 25 \\ 4 & 9 & 81 \\ 36 & 0 & 1 \end{bmatrix} \end{array}$$

Atenção:

- 1) Para obter o sinal de potenciação (^) é preciso pressionar a tecla correspondente duas vezes;
- 2) A multiplicação de uma matriz A ($n \times k$) por uma matriz B ($k \times m$) produz uma matriz $n \times m$. Para as outras operações mostradas, as matrizes A e B devem ter as mesmas dimensões.

1.5.1 Sistemas de equações lineares

Todo sistema de equações lineares pode ser escrito sob a forma matricial $Ax = b$. Exemplo:

$$S = \begin{cases} 3x_1 - 2x_2 + x_3 = -4 \\ 2x_2 - x_3 = 7 \\ 4x_1 + x_2 + 2x_3 = 0 \end{cases} \Rightarrow A = \begin{bmatrix} 3 & -2 & 1 \\ 0 & 2 & -1 \\ 4 & 1 & 2 \end{bmatrix}; \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}; \quad b = \begin{bmatrix} -4 \\ 7 \\ 0 \end{bmatrix}$$

Se a matriz dos coeficientes (A) for quadrada e não-singular, ou seja, sem linhas ou colunas linearmente dependentes, a solução (única) do sistema é dada por:

$$\bar{x} = A^{-1}b.$$

Esta solução pode ser calculada de forma direta pelo MATLAB pelas instruções:

$$x = \text{inv}(A)*b \quad \text{ou} \quad x = A\b$$

As duas formas fornecem as mesmas respostas, mas os cálculos envolvidos no uso do operador "\\" exigem menos memória e são mais rápidos do que os envolvidos no cálculo de uma matriz inversa. O MATLAB também resolve sistemas sob a forma $xA = b$ ou sistemas com mais de uma solução (o que não será discutido neste material, consulte a ajuda do programa).

1.5.2 Atividade

Resolva, se possível, os seguintes sistemas lineares.

$$S_1 = \begin{cases} 3x_1 - 2x_2 + x_3 = -4 \\ 2x_2 - x_3 = 7 \\ 4x_1 + x_2 + 2x_3 = 0 \end{cases} \quad S_2 = \begin{cases} x_1 + 4x_2 + 7x_3 = 5 \\ -3x_1 - 9x_3 = 1 \\ 2x_1 + 5x_2 + 11x_3 = -2 \end{cases} \quad S_3 = \begin{cases} x_1 + 2x_2 = 4 \\ 3x_1 + 6x_2 = 5 \end{cases}$$

1.6 Outras operações com matrizes

Há uma série de funções disponíveis no MATLAB para geração ou alteração de matrizes e vetores, exemplificadas a seguir.

a) `x = max(A)`

Retorna o maior componente de **A**. Se **A** for uma matriz, o resultado é um vetor linha contendo o maior elemento de cada coluna. Para vetores, o resultado é o maior valor (ou o número complexo com maior módulo) entre seus componentes. Ainda para vetores, a sintaxe

```
[vmax imax] = max(v)
```

retorna o maior elemento do vetor **v** em **vmax** e o índice correspondente em **imax**.

b) `x = size(A)`

Retorna as dimensões da matriz **A** em um vetor linha, **x = [m n]**, contendo o número de linhas (**m**) e colunas (**n**) da matriz. A sintaxe

```
[m n] = size(A)
```

determina o número de linhas e colunas em variáveis separadas.

c) `x = length(A)`

Retorna o comprimento do vetor **A** ou o comprimento da maior dimensão da matriz **A**. Neste último caso, **length(A) = max(size(A))**.

d) `x = zeros(n)`

Cria uma matriz quadrada $n \times n$ de elementos nulos. Também é possível obter matrizes retangulares $m \times n$ usando a sintaxe **x = zeros(m,n)**. A sintaxe

```
x = zeros(size(A))
```

produz uma matriz **x** com as mesmas dimensões de **A**, preenchida com zeros.

e) `x = ones(n)`

Semelhante a **zeros**, gerando matrizes com valores unitários (preenchidas com 1's).

f) `x = eye(n)`

Retorna uma matriz identidade $n \times n$, isto é, com valores unitários na diagonal principal e nulos nas demais posições.

g) `x = det(A)`

Retorna o determinante da matriz quadrada **A**. Nota: para verificar se uma matriz possui linhas ou colunas linearmente dependentes o manual do MATLAB recomenda usar a função **cond** (cálculo do número de condição) ao invés de verificar se **det(A) = 0**.

h) `x = find(expressão)`

Encontra e retorna todos os elementos de um vetor ou matriz que satisfazem a uma certa expressão lógica. Normalmente, usa-se argumentos à esquerda da instrução de busca para armazenar os índices dos elementos de interesse. Exemplo:

```
>> A = [3 -2 1; 0 2 -1; 4 1 2]; ↴
>> [L C] = find(A>2 & A<=4) ↴
L =
    1
    3
C =
    1
    1
```

$$Obs.: \quad A = \begin{bmatrix} 3 & -2 & 1 \\ 0 & 2 & -1 \\ 4 & 1 & 2 \end{bmatrix}$$

No exemplo anterior, apenas $A(1,1)$ e $A(3,1)$ atendem ao critério desejado. As expressões válidas em MATLAB podem incluir operadores relacionais e lógicos, resumidos na tabela 1.1.

Tabela 1.1: Operadores relacionais e lógicos

Operadores relacionais		Operadores lógicos	
<	Menor que	&	Operação "E"
<=	Menor ou igual		Operação "OU"
>	Maior que	~	Negação lógica
>=	Maior ou igual		
==	Igual a		
~=	Diferente de		

Estes operadores se aplicam a escalares e matrizes, de acordo com regras que podem ser consultadas na documentação do MATLAB. Se o argumento da função for apenas o nome de uma matriz ou vetor, serão retornados os índices de todos os elementos da matriz ou vetor que forem diferentes de zero.

i) $x = \text{all}(A)$

Retorna 1 para cada coluna da matriz **A** que contenha somente valores não nulos e 0 em caso contrário, gerando um vetor linha. Exemplo:

```
>> A = [3 -2 1; 0 2 -1; 4 1 2]; ↴
>> x = all(A) ↴
x =
    0      1      1
```

$$Obs.: \quad A = \begin{bmatrix} 3 & -2 & 1 \\ 0 & 2 & -1 \\ 4 & 1 & 2 \end{bmatrix}$$

Para vetores, a função retorna 1 se todos os elementos forem não nulos e 0 em caso contrário.

j) $x = \text{any}(A)$

Retorna 1 para cada coluna da matriz **A** que contenha algum valor não nulo e 0 em caso contrário, gerando um vetor linha. A função também trabalha com vetores. Exemplo:

```
>> x = any(A) ↴
x =
    1      1      1
```

1.7 Gráficos

O MATLAB possui diversas ferramentas para traçados de gráficos bidimensionais ou tridimensionais. A maneira mais simples de traçar um gráfico *xy* é pelo uso da função **plot**. A forma **plot(x,y)** desenha um

gráfico bidimensional dos pontos do vetor y em relação aos pontos do vetor x , sendo que ambos devem ter o mesmo número de elementos. Não é obrigatório que os valores de y representem uma função em relação aos valores de x . O gráfico resultante é desenhado em uma *janela de figura* com as escalas automáticas nos eixos x e y e segmentos de reta unindo os pontos. Por exemplo, para desenhar o gráfico da função

$$y = 1 - 1,1547e^{-1,5x} \sin(2,5981x + 1,0472),$$

no intervalo $x \in [0,10]$, pode-se utilizar a seguinte seqüência de comandos:

```
>> x = 0:0.1:10; ↴
>> y = 1-1.1547*exp(-1.5*x).*sin(2.5981*x+1.0472); ↴
>> plot(x,y) ↴
```

O resultado (ver figura 1.4) é exibido em uma janela de figura identificada por um número.

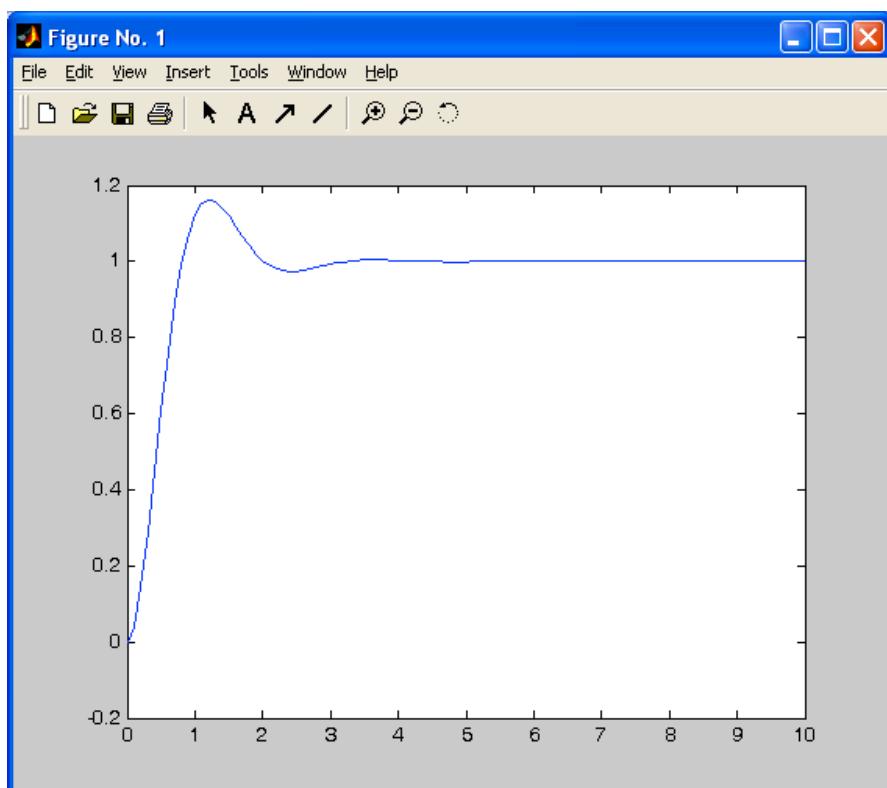


Figura 1.4: Exemplo de resultado gráfico da função *plot*

Em algumas ocasiões é interessante que as escalas dos eixos sejam representadas em escala logarítmica (ao invés da escala linear padrão). Nestes casos, é possível usar as funções *semilogx*, *semilogy* ou *loglog*, que alteram, respectivamente, a escala do eixo x , do eixo y e de ambos. Normalmente os valores que compõem tais gráficos também são gerados com espaçamentos logarítmicos, via função *logspace*.

A função *plot* pode trabalhar com várias duplas de vetores, sobrepondo mais de um gráfico em uma mesma janela. Exemplo:

```
x = linspace(0,2*pi,100); % Cria vetor 'x' com 100 pontos de 0 a 2*pi
y1 = sin(x); % Calcula y1 = sen(x)
y2 = 0.5*sin(3*x); % Calcula y2 = 0.5*sen(3x)
plot(x,y1,x,y2); % Traça os dois gráficos
xlabel('Ângulo em graus'); % Nomeia o eixo x
ylabel('sen(x) e sen(3x)'); % Nomeia o eixo y
title('Gráficos sobrepostos'); % Atribui um título ao gráfico
grid % Ativa as linhas de grade da janela
```

Note que foram usadas funções para nomear os eixos (*xlabel* e *ylabel*) e o título do gráfico (*title*), além de exibição de linhas de grade (*grid*). O resultado da seqüência de comandos anterior está representado na figura 1.5.

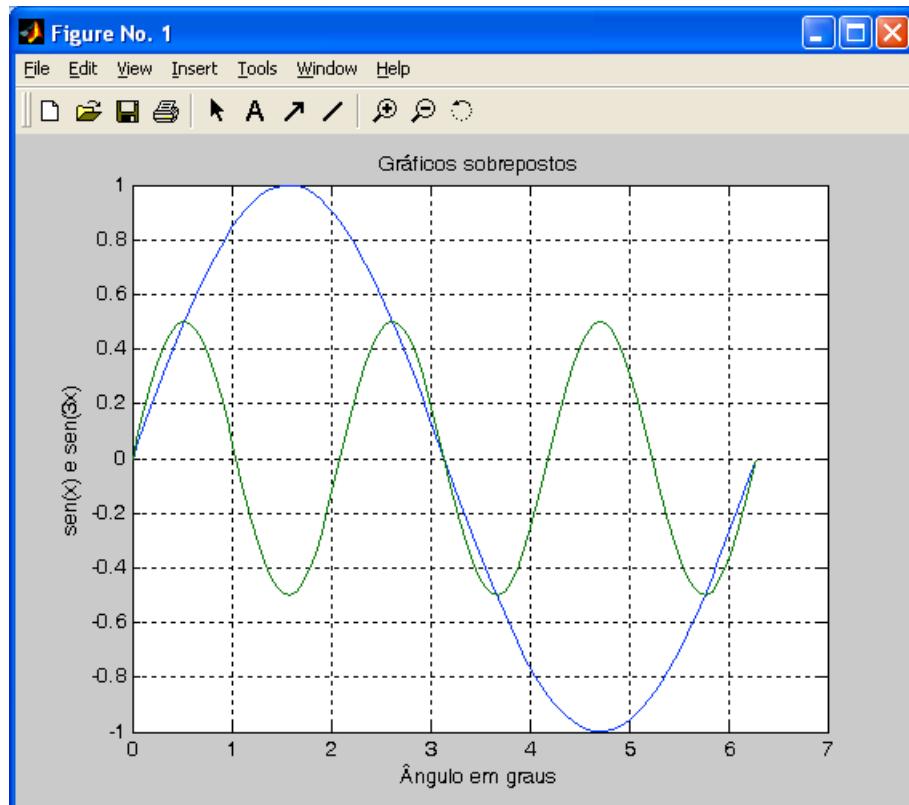


Figura 1.5: Gráfico de duas funções superpostas

Outra forma de se obter gráficos sobrepostos é com o uso da função *hold*, que faz com que todos os resultados gráficos subseqüentes ao seu uso sejam desenhados em uma mesma janela de figura. Exemplo (considerando as variáveis do exemplo anterior):

```
plot(x,y1); % Desenha o gráfico de uma função
hold on    % Ativa a 'trava' de exibição gráfica
plot(x,y2); % Desenha outro gráfico na mesma janela de figura
hold off   % Desativa a 'trava' de exibição gráfica
```

Todos os resultados gráficos aparecem na janela de figura *ativa*. Uma nova janela pode ser criada ou ativada pelo comando *figure*. Quando usada sem argumentos, esta função cria uma janela de título *Figure No. xx*, sendo *xx* um número seqüencial, considerado disponível pelo MATLAB. O uso de *figure(n)* cria a janela de figura *n*, se necessário, e a torna ativa. Outros recursos da função *plot* podem ser consultados na documentação do MATLAB.

1.7.1 Gráficos tridimensionais

Gráficos em três dimensões podem ser traçados pelo MATLAB com a mesma facilidade que os bidimensionais. A função *plot3* funciona de forma semelhante à *plot* para o traçado de *gráficos de linha*. Por exemplo, a seqüência de comandos a seguir produz um gráfico de uma *hélice tridimensional*. Note o uso da função *zlabel* para nomear o eixo *z* do gráfico.

```
t = linspace(0,6*pi,100); %
plot3(sin(t),cos(t),t); %
xlabel('seno(t)'); %
ylabel('cosseno(t)');
```

```

zlabel('z = t'); ↵
title('Gráfico de hélice'); ↵
grid on; ↵

```

O resultado está representado na figura 1.6.

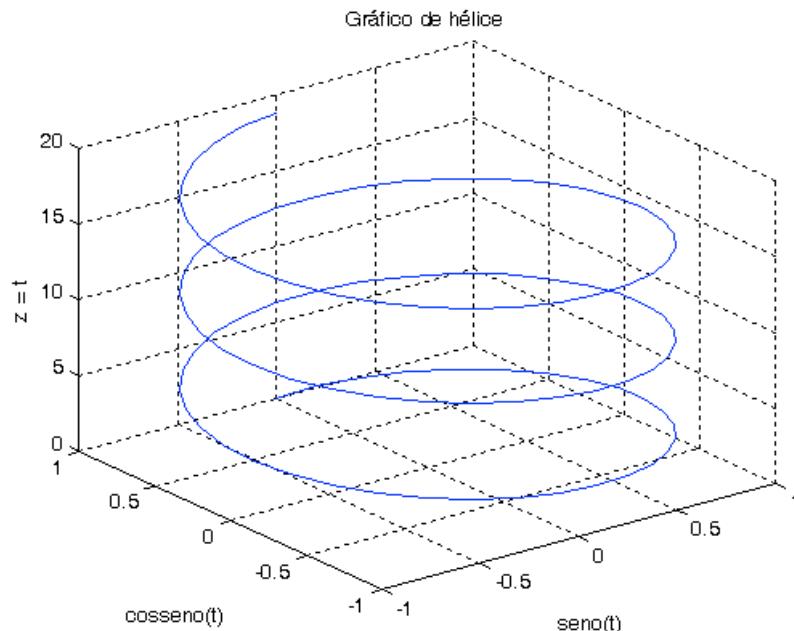


Figura 1.6: Gráfico de linha tridimensional

O MATLAB também pode construir *gráficos de superfícies*, a partir de um conjunto de coordenadas tridimensionais xyz . Inicialmente, é preciso gerar matrizes X e Y com, respectivamente, linhas e colunas repetidas, preenchidas com os valores das variáveis x e y . Isto pode ser feito diretamente pela função **meshgrid**, como no exemplo mostrado a seguir:

```

x = linspace(0,2,20);           % Geração de valores para 'x' e 'y',
y = linspace(1,5,20);           % ambos com a mesma dimensão!
[X,Y] = meshgrid(x,y);         % Criação da matriz da malha 'xy'

```

A partir dessas matrizes, que representam uma grade retangular de pontos no plano xy , qualquer função de duas variáveis pode ser calculada em uma matriz Z e desenhada pelo comando **mesh**. Exemplo para o gráfico de um *parabolóide elíptico*:

```

x = -5:0.5:5;                  % Definição da malha de pontos no eixo 'x'
y = x;                          % Repetição da malha do eixo x para o eixo 'y'
[X,Y] = meshgrid(x,y);          % Criação da matriz da malha 'xy'
Z = X.^2 + Y.^2;                % Cálculo da função z = f(x,y)
mesh(X,Y,Z)                     % Traçado do gráfico da função 'z'

```

O resultado deste exemplo é mostrado na figura 1.7. A função **mesh** cria uma *malha tridimensional* em que cada ponto é unido por segmentos de reta aos vizinhos na malha. Usando a função **surf** é possível gerar um gráfico de superfície em que os espaços entre os segmentos são coloridos. Em ambos os casos, uma quarta matriz pode ser usada como parâmetro para estabelecer as cores a serem usadas no desenho. Se esta matriz for omitida, como no exemplo anterior, as cores das linhas serão relacionadas com a altura da malha sobre o plano xy . As duas funções podem receber somente a matriz Z como parâmetro, traçando um gráfico de malha cujos valores de x e y correspondem aos índices da matriz.

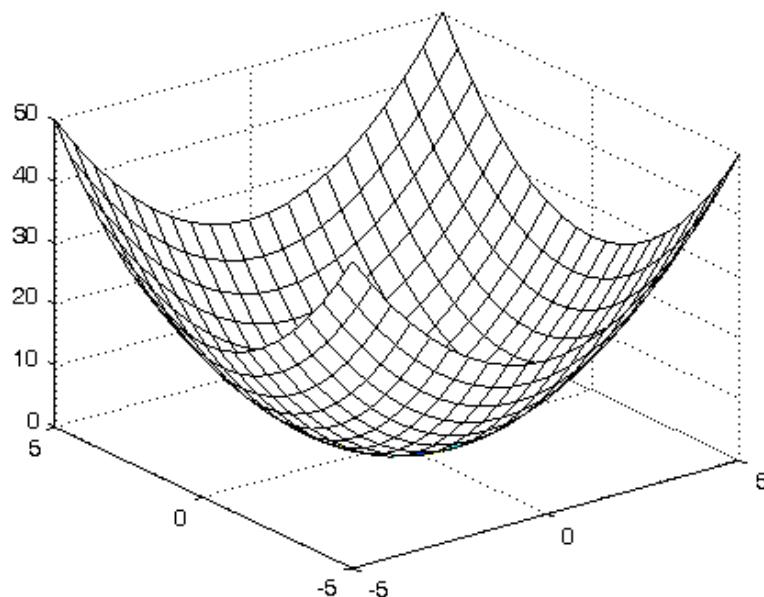


Figura 1.7: Gráfico de superfície tridimensional

1.8 Polinômios

O MATLAB possui funções específicas para operações com polinômios, como a determinação de raízes, avaliação, diferenciação, etc. Uma função polinomial da forma

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

pode ser representada no MATLAB por um **vetor de coeficientes**, em ordem decrescente de potência:

$$p = [a_n \quad a_{n-1} \quad \cdots \quad a_2 \quad a_1 \quad a_0].$$

Por exemplo, o polinômio $g(x) = x^3 - 2x - 5$ pode ser representado pelo seguinte vetor:

`g = [1 0 -2 -5];`

As raízes (reais ou complexas) de um polinômio podem ser calculadas diretamente em um vetor coluna pela função **roots**. Exemplo:

```
>> r = roots(g) ↴
r =
    2.0946
   -1.0473 + 1.1359i
   -1.0473 - 1.1359i
```

Note a forma de representação de números complexos no MATLAB (parte real + parte imaginária + *i*). De fato, o MATLAB reconhece automaticamente as letras *i* e *j* como a unidade imaginária da matemática.

De forma inversa, se forem conhecidas as raízes de um polinômio, a função **poly** reconstrói o polinômio original. Por exemplo, os coeficientes do vetor **g** do exemplo anterior, podem ser recuperados pela instrução:

```
p1 = poly(r) % Atenção: o argumento da função poly deve ser um vetor coluna!
```

1.8.1 Avaliação, multiplicação, divisão e diferenciação

Avaliar um polinômio significa determinar o valor de $p(x)$ para um dado valor de x . Para calcular, por exemplo, $g(2.4)$ usa-se a função ***polyval***, como em:

```
>> y = polyval(g, 2.4) ↴
```

```
y =
```

```
4.0240
```

As operações de multiplicação e divisão entre polinômios correspondem, respectivamente, a operações de *convolução* e *deconvolução*, implementadas pelas funções ***conv*** e ***deconv***. Por exemplo, considere:

$$n(s) = 3s^2 + s + 1 \quad \text{e} \quad d(s) = s + 1.$$

O produto $n(s)d(s)$ pode ser calculado com a seguinte seqüência de comandos:

```
>> n = [3 1 1]; ↴
>> d = [1 1]; ↴
>> prod = conv(n, d) ↴

prod =
3     4     2     1
```

Note que o grau do polinômio resultante é dado pela soma dos graus dos polinômios envolvidos na multiplicação. Finalmente, a derivada de uma função polinomial pode ser obtida diretamente a partir do vetor que representa a função com o uso da função ***polyder***.

Por exemplo, a derivada de $f(x) = 2x^3 + x^2 - 3x$ pode ser calculada com:

```
>> f = [2 1 -3 0] ↴
>> f1 = polyder(p) ↴

f1 =
6     2    -3
```

1.9 Funções de transferência

Considere um sistema linear em que se possa monitorar uma variável de saída, gerada pela ação de uma variável de entrada. Neste caso, define-se a **função de transferência** do sistema como a relação entre a transformada de Laplace da variável de saída e a transformada de Laplace da variável de entrada, considerando condições iniciais nulas. Existe uma classe própria no MATLAB para funções de transferência, criadas pela função ***tf*** e definidas pelo quociente de dois polinômios na variável s . Por exemplo, a função de transferência

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

pode ser armazenada em uma variável no MATLAB pela seguinte seqüência de comandos:

```
>> n = 3; ↴
>> d = [1 2 3]; ↴
```

```
>> G = tf(n,d) ↵
Transfer function:
3
-----
s^2 + 2 s + 3
```

1.10 Simulações

Existem funções específicas para simular o comportamento de sistemas lineares a entradas tipo impulso, degrau ou entradas genéricas. Para simular a resposta a um *impulso unitário* (em $t = 0$ s) de um sistema linear utiliza-se a função ***impulse***, fornecendo os polinômios representativos da função de transferência do sistema ou a própria função. Considerando as variáveis **n** e **d** do exemplo anterior, pode-se usar indistintamente

`impulse(n,d)` ou `impulse(G)`

O resultado da simulação é apresentado em uma janela gráfica, como mostra a figura 1.8. Opcionalmente, pode-se fornecer um valor em segundos para o tempo final de simulação:

```
impulse(G,10); % Simula a resposta ao impulso por 10 s.
```

É possível, ainda, armazenar os vetores do tempo de simulação (criado automaticamente pelo MATLAB) e da resposta do sistema, sem desenhar o gráfico correspondente. Exemplo:

```
[y t] = impulse(G,10); % Simula por 10 s. Retorna vetores de tempo e saída
```

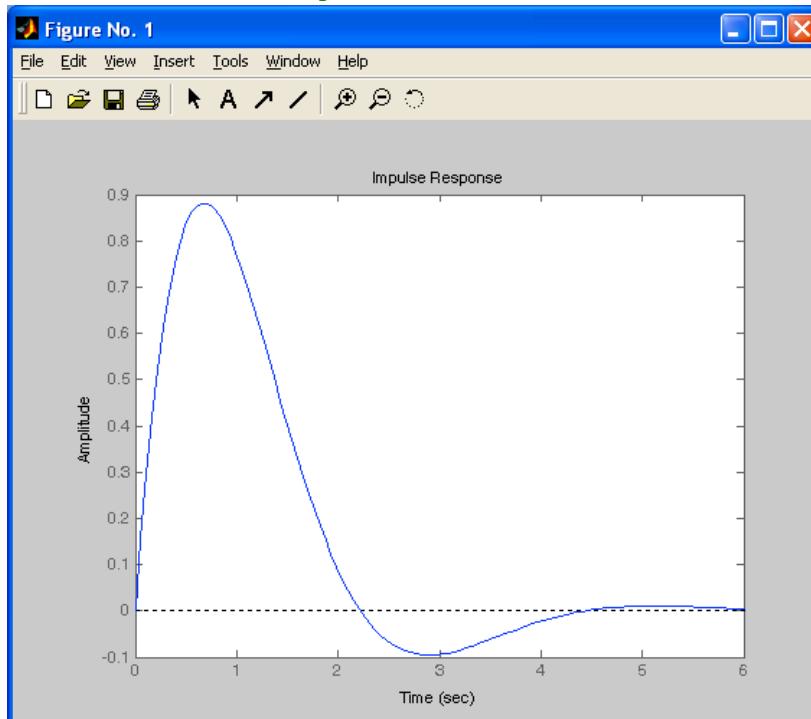


Figura 1.8: Resposta ao impulso

A simulação da resposta a uma entrada em *degrau unitário* é feita pela função ***step***, como em:

```
step(G); % Opção: step(n,d);
```

O resultado desta simulação está representado na figura 1.9.

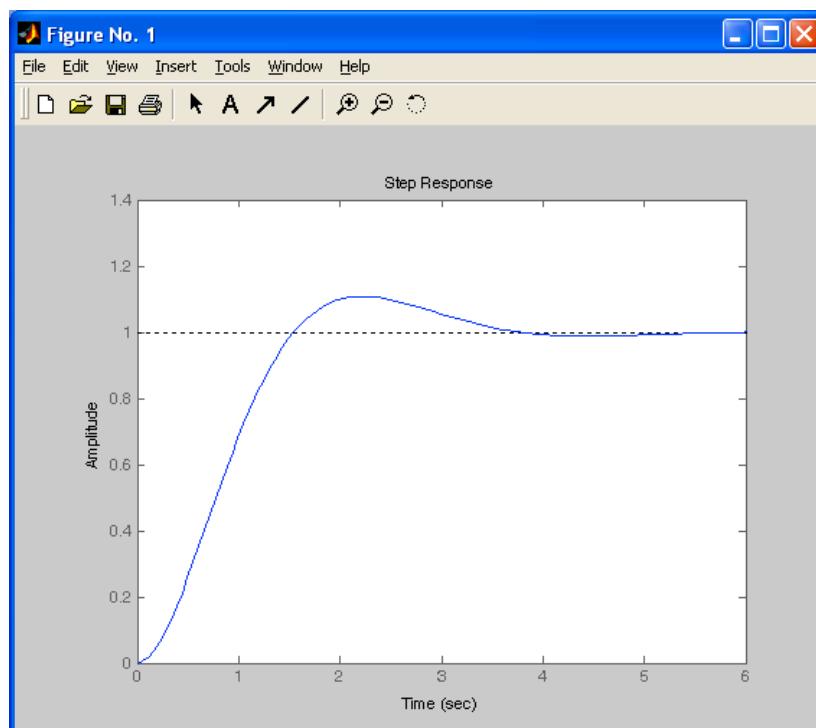


Figura 1.9: Resposta ao degrau unitário

Não é possível alterar a amplitude do degrau usado na simulação. No entanto, é possível controlar o tempo de simulação e armazenar os vetores de resposta (saída e tempo). Exemplo:

```
>> [y t] = step(G,10); ↴
```

Como se trata da simulação de um sistema linear, a saída para uma entrada em degrau de amplitude A pode ser calculada como $y_2(t) = Ay(t)$. Finalmente as funções **impulse** e **step** permitem que o usuário forneça um vetor de tempos a ser usado na simulação. Exemplo:

```
t = 0:0.01:15; ↴
step(n,d,t); ↴
```

Assim como no caso da função **plot**, pode-se sobrepor dois gráficos em uma mesma janela de figura. Finalmente, para simular a resposta de um sistema linear a uma entrada genérica é preciso usar a função **lsim**, fornecendo a especificação do sistema e os vetores de entrada e de tempo de simulação. Exemplo (usando o sistema **G** definido anteriormente):

```
t = 0:0.1:10;           % Vetor de tempo de simulação
u = zeros(length(t),1); % Vetor de entrada, com mesma dimensão de 't'
u(21:30) = 0.5;         % Atribuição de valores não nulos
lsim(G,u,t);           % Simulação
```

O resultado da simulação é apresentado em uma janela gráfica, como mostra a figura 1.10.

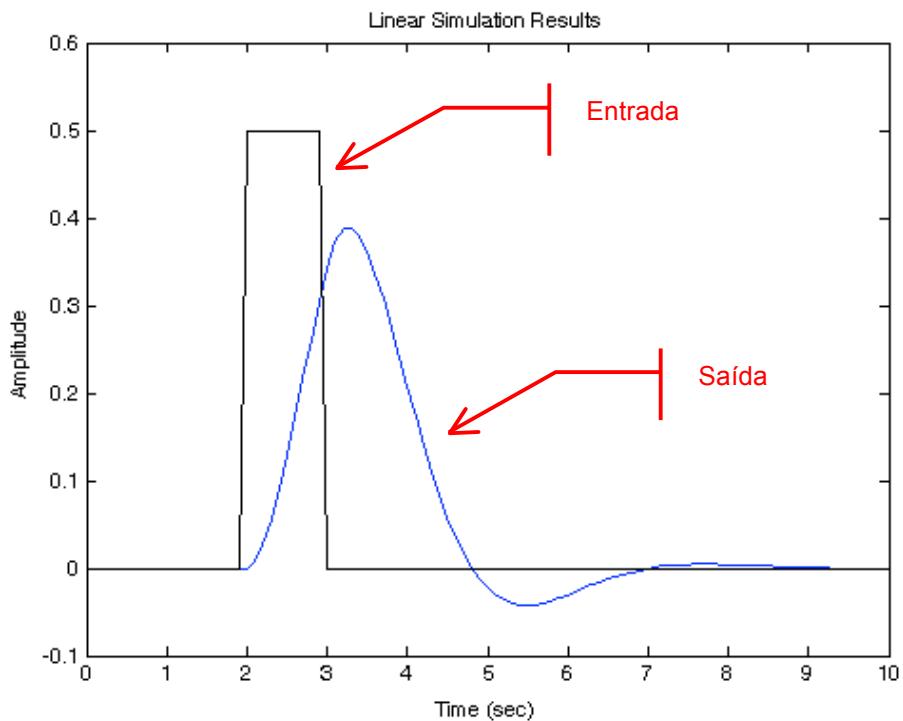


Figura 1.10: Resposta a um sinal genérico

Se for usada uma sintaxe com argumentos à esquerda a simulação será feita mas o gráfico não será desenhado. O vetor de saída criado pela função terá sempre o mesmo número de elementos do vetor de tempo fornecido.

2. Análise de sistemas lineares de 1^a e 2^a ordem – atividades

2.1 Sistemas de 1^a ordem

Obtenha a resposta ao degrau unitário dos sistemas definidos pelas seguintes funções de transferência:

a) $G_1(s) = \frac{10}{s + 5}$

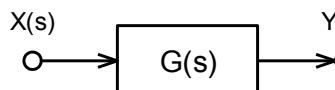
b) $G_2(s) = \frac{0,8}{s + 2}$

c) $G_3(s) = \frac{1}{0,01s + 1}$

Em seguida, determine:

- i) os pólos do sistema;
- ii) o valor final teórico da resposta (usando o Teorema do Valor Final);
- iii) o valor final do sinal de resposta (a partir da simulação);
- iv) a constante de tempo do sistema.

Finalmente, discuta a estabilidade de cada sistema, classificando-os como estáveis (E), marginalmente estáveis (ME) ou instáveis (I). Teorema do Valor Final:



$$y(\infty) = \lim_{s \rightarrow 0} s [X(s)G(s)] = \lim_{s \rightarrow 0} s Y(s)$$

2.2 Resposta temporal

Reescreva as funções de transferência anteriores, sob a forma

$$\bar{G}(s) = \frac{K}{\tau s + 1}$$

e verifique a relação entre esta forma e os parâmetros da resposta ao degrau: constante de tempo e valor final.

2.3 Sistemas de 2^a ordem

Obtenha a resposta ao degrau unitário dos sistemas definidos pelas seguintes funções de transferência:

a) $G_4(s) = \frac{1}{s^2 + s + 1}$

b) $G_5(s) = \frac{9}{s^2 + 3s + 9}$

c) $G_6(s) = \frac{25}{s^2 + 7s + 25}$

d) $G_7(s) = \frac{25}{s^2 + 10s + 25}$

Observação: Nas simulações de resposta ao degrau criadas pela função *step* as principais características de desempenho podem ser obtidas diretamente na janela gráfica: clique com o botão direito do *mouse* sobre uma área livre do gráfico e selecione, no menu *Characteristics*, as opções *Peak Response* (ultrapassagem ou sobressinal), *Settling Time* (tempo de assentamento), *Rise Time* (tempo de subida) e *Steady State* (valor final).

Determine:

- i) os pólos e zeros do sistema;
- ii) o valor final teórico da resposta (usando o Teorema do Valor Final);
- iii) o valor final do sinal de resposta (a partir da simulação);

- iv) o tempo de subida e de acomodação do sinal de resposta;
- v) o valor da ultrapassagem (ou sobressinal).

Em seguida, discuta a estabilidade de cada sistema e classifique-os em subamortecidos (SB), sobreamortecidos² (SO), criticamente amortecidos (CA) ou oscilatórios (O).

2.4 Sistemas de 2^a ordem sem zeros

Reescreva as funções de transferência do item anterior sob a forma

$$\bar{G}(s) = K \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$$

e identifique os valores de ξ e ω_n . Verifique a influência da *frequência natural* (ω_n) na velocidade da resposta de sistemas com a mesma *relação de amortecimento* (ξ). Em seguida, analise a influência da relação ξ em sistemas com a mesma freqüência ω_n . Para qual caso o sobressinal da resposta se mantém igual?

2.5 Validade do Teorema do Valor Final

Usando o Teorema do Valor Final, determine o valor estacionário da resposta ao degrau unitário do sistema definido por:

$$G_8(s) = \frac{s+2}{s^3+1}.$$

Simule a resposta ao degrau e compare com o resultado analítico. Por que o teorema falhou?

2.6 Aproximações

- a) Obtenha em uma mesma janela de figura as respostas ao degrau de

$$G_4(s) = \frac{1}{s^2 + s + 1} \quad \text{e} \quad G_9(s) = \frac{10}{s^3 + 11s^2 + 11s + 10}$$

Repita (em outra janela de figura) para uma entrada em impulso. O que se pode concluir?

- b) Obtenha em uma mesma janela de figura as respostas ao degrau unitário de

$$G_{10}(s) = \frac{2}{s^2 + 2s + 2} \quad \text{e} \quad G_{11}(s) = \frac{s+2}{s^2 + 2s + 2}.$$

Discuta os efeitos causados na resposta ao degrau de $G_{10}(s)$ pela inclusão do zero ($s = -2$).

² Também chamados de superamortecidos.

3. Aproximações e estabilidade de sistemas lineares

3.1 Sistemas de 3^a ordem

A característica dominante de um sistema de 3^a ordem sem zeros pode ser identificada por meio de *expansão em frações parciais*. Por exemplo, a função de transferência

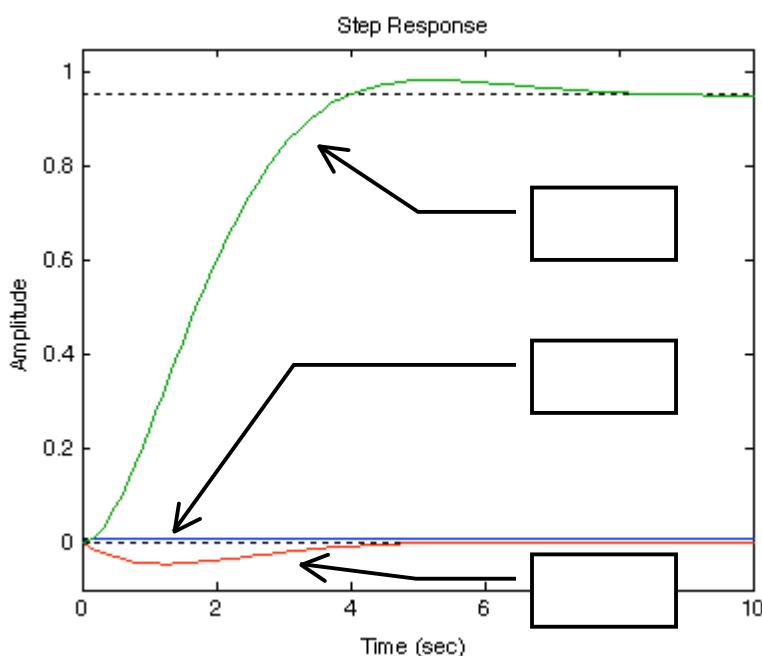
$$G(s) = \frac{7,98}{s^3 + 11,4s^2 + 14s + 7,98},$$

pode ser reescrita como:

$$\bar{G}(s) = G_1(s) + G_2(s) + G_3(s) = \frac{0,0893}{s + 10,091} + \frac{0,7530}{s^2 + 1,3090s + 0,7908} - \frac{0,0893s}{s^2 + 1,3090s + 0,7908}$$

3.1.1 Exemplo

Obtenha em uma mesma janela de figura as respostas ao degrau de cada termo da função $\bar{G}(s)$, dada anteriormente. Verifique qual termo possui maior influência na resposta dinâmica do sistema. Confirme sua análise a partir da resposta ao degrau do sistema de 3^a ordem, $G(s)$.



3.2 Aproximações para modelos de 2^a ordem

Os passos a seguir descrevem uma forma de se aproximar um sistema de ordem superior, sem zeros, para um sistema de 2^a ordem:

- 1) Calcule (ou visualize no plano complexo) os pólos originais da função de transferência – use as funções *roots* e *pzmap*;
- 2) Despreze o pólo (ou pólos) que tiver menor influência na resposta dinâmica do sistema para obter um novo denominador de 2^a ordem;
- 3) Obtenha um numerador que mantenha o ganho estático (ganho DC) do sistema original;
- 4) Verifique (por exemplo, graficamente) a qualidade da aproximação obtida.

3.2.1 Exemplos

a) Usando o conceito de *pólos dominantes*, escreva modelos de 2^a ordem para:

$$i) \quad G_1(s) = \frac{7,98}{s^3 + 11,4s^2 + 14s + 7,98}$$

$$ii) \quad G_2(s) = \frac{73,626}{(s+3)(s^2 + 4s + 24,542)}$$

b) Considere um sistema definido pela função de transferência:

$$G(s) = \frac{1,278s + 12,78}{s^3 + 11,72s^2 + 17,626s + 4,26}$$

i) Reescreva a função sob a forma $T(s) = K \frac{(s - z_1)}{(s - p_1)(s - p_2)(s - p_3)}$.

ii) Usando o resultado anterior, obtenha uma aproximação de 2^a ordem para $G(s)$.

c) Escreva uma seqüência de comandos que crie uma aproximação de 2^a ordem, $G_2(s)$, para:

$$G_3(s) = \frac{156,25}{s^4 + 16s^3 + 78,75s^2 + 81,25s + 78,125}.$$

Use apenas instruções literais, isto é, que não envolvam valores numéricos diretamente.

3.3 Conexões entre sistemas

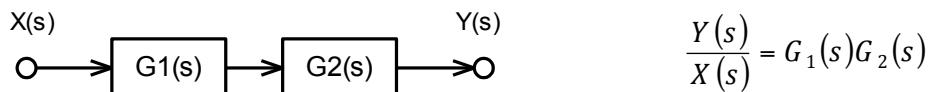
O MATLAB possui funções para determinar o efeito de algumas formas de interconexão entre funções de transferência. Nos exemplos a seguir, considere que se deseja obter

$$\frac{Y(s)}{X(s)} = \frac{n(s)}{d(s)}$$

a partir das funções:

$$G_1(s) = \frac{n_1(s)}{d_1(s)} \text{ e } G_2(s) = \frac{n_2(s)}{d_2(s)}.$$

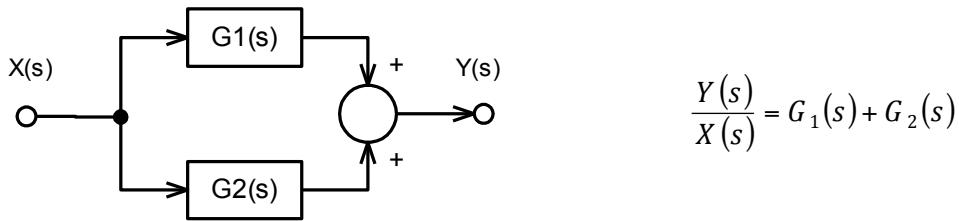
a) Conexão em cascata



Comandos:

<pre>[n d] = series(n1,d1,n2,d2) [n d] = series(G1,G2) FT = G1*G2</pre>	$\left. \right\} \quad \begin{matrix} 3 \text{ opções para a} \\ \text{mesma operação} \end{matrix}$
---	--

b) Conexão em paralelo



Comandos:

$[n \ d] = \text{parallel}(n1, d1, n2, d2)$ $[n \ d] = \text{parallel}(G1, G2)$ $FT = G1 + G2$	$\left. \right\} 3 \text{ opções para a mesma operação}$
--	--

3.4 Estabilidade

Sabe-se que um sistema em malha fechada é estável se sua função de transferência não apresentar pólos no *semiplano direito* do plano complexo, ou seja, se nenhum polo tiver parte real positiva. Por exemplo, considere o sistema da figura 3.1.

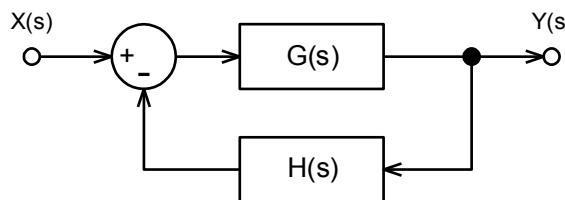


Figura 3.1: Sistema de controle em malha fechada

A função de transferência em malha fechada deste sistema é dada por:

$$FTMF = \frac{Y(s)}{X(s)} = \frac{G(s)}{1 + G(s)H(s)}.$$

O denominador da função de transferência em malha fechada dá origem à *equação característica* do sistema, definida como:

$$q(s) = 1 + G(s)H(s) = 0$$

Para um sistema como o da figura 3.1, a função **feedback** do MATLAB permite determinar diretamente a função de transferência em malha fechada. Exemplo:

```
>> MF = feedback(G, H) ↴
```

O padrão da função **feedback** é trabalhar com realimentação negativa. Para sistemas com *realimentação negativa unitária*, isto é, quando $H(s) = 1$, usa-se:

```
>> MF = feedback(G, 1) ↴
```

3.4.1 Exemplos

Investigue a estabilidade dos sistemas a seguir, admitindo a configuração da figura 3.1, com realimentação negativa unitária. Sistemas marginalmente estáveis devem ser classificados como instáveis.

a) $G(s) = \frac{1}{s^3 + s^2 + 2s + 1,5}$ () Estável () Instável

b) $G(s) = \frac{1}{s^2 + 2s - 4}$ () Estável () Instável

c) $G(s) = \frac{1}{s^4 + 6,5s^3 + 14s^2 + 11,5s + 2}$ () Estável () Instável

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

4. Programação

Um dos aspectos mais poderosos do MATLAB é a possibilidade de se criar programas em uma linguagem de programação interpretada³ usando a mesma notação aceita na janela de comando. Arquivos contendo código MATLAB são arquivos de texto com a extensão **.m** chamados de *arquivos-M* (*M-files*). Estes arquivos podem conter o código de **scripts** ou **funções**, cujas principais características estão relacionadas na tabela 4.1.

Tabela 4.1: Características das formas de programação MATLAB

Arquivos de <i>script</i>	Arquivos de funções
Não aceitam argumentos nem retornam valores ao <i>workspace</i> .	Aceitam argumentos e retornam valores ao <i>workspace</i> .
Trabalham com as variáveis definidas no <i>workspace</i> .	Trabalham com variáveis definidas localmente ao arquivo.
Principal aplicação: automatização de comandos que precisam ser executados em uma certa seqüência.	Principal aplicação: adaptação da linguagem MATLAB a qualquer situação de programação necessária.

4.1 Exemplo – análise de um sistema linear

Para ilustrar as formas de programação possíveis vamos criar um *script* para a análise de desempenho de um sistema linear de 2^a ordem definido, pela função de transferência:

$$F(s) = \frac{5}{s^2 + 2s + 5}.$$

4.1.1 Criando um *script*

Os *scripts* constituem a forma mais simples de programação em ambiente MATLAB porque apenas automatizam uma série de comandos. É possível usar qualquer editor de textos para a criação de *scripts*, mas o uso do editor embutido no MATLAB é preferido por fornecer recursos úteis ao programador como auto-indentação, destaque de palavras reservadas, ferramentas de depuração, etc. Digite **edit** na janela de comando ou clique em *File > New > M-file* para invocar o editor. Para este exemplo, digite o código listado a seguir e salve-o com o nome **Analise1.m**. Os comentários podem ser omitidos.

```
% ANALISE1.M - Script para análise de desempenho.
% Simula a resposta ao degrau e ao impulso de:
%
%      5
% F(s) = -----
%           s^2 + 2s + 5
%
% Atenção: cria os polinômios 'n' e 'd' e a função
% de transferência 'F' no workspace!
```

```
clear; % Limpa todas as variáveis da memória
```

³ Como a linguagem de programação do MATLAB é *interpretada*, todos os códigos precisam ser executados a partir do MATLAB. É possível criar executáveis independentes, assunto que não será discutido neste material.

```

clc; % Limpa a janela de comando
n = 5; % Define o numerador da função de transferência
d = [1 2 5]; % Define o denominador da função de transferência
F = tf(n,d); % Cria a função de transferência F
figure(1); % Cria a janela gráfica 1
step(F,10); % Simula por 10 s e desenha a resposta ao degrau de F
figure(2); % Cria a janela gráfica 2
impulse(F,10); % Simula por 10 s e desenha a resposta ao impulso de F

```

Antes de executar o *script* é preciso que o MATLAB reconheça a pasta em que o arquivo **.m** foi gravado como um diretório de trabalho. Digite **cd** na janela de comando para descobrir o diretório de trabalho atual e, se necessário, altere-o para o diretório onde o *script* foi gravado. Por exemplo, se o *script* foi gravado em **C:\TEMP**, digite:

```
>> cd c:\temp ↵
```

Execute o *script*, digitando seu nome (**Analise1**) na janela de comando. Neste exemplo, o resultado gráfico é exibido em duas janelas (ver figura 4.1). Verifique que as variáveis **F**, **n** e **d** (e somente elas!) permanecem no *workspace* após a execução do *script*.

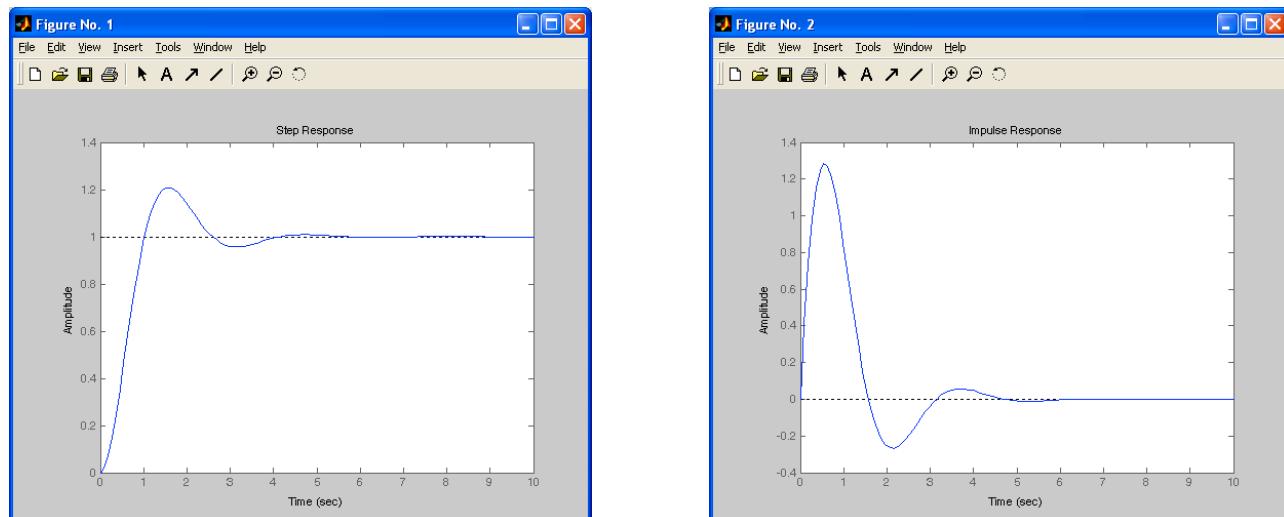


Figura 4.1: Resultados gráficos do script

4.1.2 Análise do *script*

As primeiras linhas do código proposto contém comentários que são exibidos pelo comando **help** quando o usuário pede ajuda sobre o *script*:

```
>> help Analise1 ↵
```

ANALISE1.M – Script para análise de desempenho.
Simula a resposta ao degrau e ao impulso de:

$$F(s) = \frac{5}{s^2 + 2s + 5}$$

Atenção: cria os polinômios '**n**' e '**d**' e a função de transferência '**F**' no *workspace*!

A primeira linha de comentário, chamada de **linha H1** é usada nas buscas por palavra-chave do comando **lookfor**. Exemplo:

```
>> lookfor desempenho ↵
```

Analisel.m: % ANALISE1.M - Script para análise de desempenho.

4.1.3 Criando uma função

Funções são *arquivos-M* que estendem a capacidade de processamento dos *scripts* por aceitar argumentos de entrada e retornar valores para o *workspace*. Cada função trabalha com variáveis locais, isoladas do espaço de memória do *workspace*. Além disso, as funções podem ser executadas mais rapidamente que os *scripts* por serem compiladas internamente em um *pseudo-código* que é mantido em memória, aumentando a velocidade de execução caso a função seja chamada mais de uma vez.

A primeira linha de um arquivo de função depois dos comentários iniciais deve conter a palavra-chave **function** seguida pela definição dos valores de retorno, nome da função e pela lista de argumentos de entrada⁴. Como exemplo, analise o código listado a seguir de uma função (**Media**) para cálculo da média dos elementos de um vetor (o uso da estrutura **if** dera explicado futuramente).

```
% MEDIA.M - Função para cálculo da média dos elementos de um vetor
% Versão simplificada, com verificação básica de dimensão do vetor
% Forma de uso: y = media(V) , sendo V um vetor linha ou coluna

function y = Media(V)

[nl nc] = size(V);      % Obtém dimensões do vetor
if (nl==1 & nc==1)      % Se nl=nc=1, V é um escalar
    disp('Erro: a função não trabalha com escalares');
else
    y = sum(V)/length(V);    % Se não for escalar, calcula a média
end    % Fim da estrutura condicional
```

Exemplo de uso da função:

```
>> a = [1 2 4 6 7 7]; ↵
>> b = Media(a) ↵

b =
4.5000
```

Note que o valor de retorno da função foi o último valor atribuído internamente à variável **y**. O final do código de uma função não precisa de nenhum identificador específico.

4.1.4 Atividade

Pesquise detalhes de funcionamento das funções **disp** e **sum**, usadas no código da função **Media**. Consulte também a documentação da função **mean**.

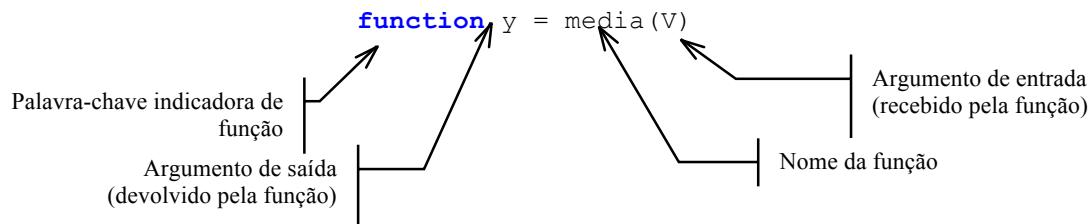
4.1.5 Convenções

Os nomes de funções no MATLAB têm as mesmas restrições que os nomes de variáveis: devem ter, no máximo, 31 caracteres e iniciar por uma letra seguida por qualquer combinação de caracteres, números ou caracteres de sublinhado. O nome do arquivo que contém o código da função deve ser formado pelo nome da

⁴ São válidas as mesmas observações feitas para os *scripts* em relação à utilidade das primeiras linhas do código.

função, com extensão **.m**. Se houver divergências, o nome do arquivo prevalecerá sobre o nome interno da função.

A linha de definição da função deve seguir uma forma padronizada que identifique a quantidade de parâmetros recebidos ou devolvidos pela função. Para a função **Media**, usada anteriormente como exemplo, pode-se identificar:



Se a função retornar mais de um valor deve-se especificar uma lista de argumentos de retorno, delimitada por colchetes. Argumentos de entrada, se existirem, devem estar entre parênteses. As duas listas devem ter seus valores, *obrigatoriamente*, separados por vírgulas. Exemplo:

```

% PONTOMEDIO.M - Calculo das coordenadas do ponto médio entre
% pontos. A função deve receber 2 vetores de coordenadas (x,y)

function [x,y] = PontoMedio(coord_x,coord_y)

if(length(coord_x)==length(coord_y))
    x = mean(coord_x);
    y = mean(coord_y);
else
    disp('Erro nas dimensões dos vetores!');
end
  
```

Exemplo de execução da função **PontoMedio**:

```

>> cx = [1 2 4 1 4 5]; ↴
>> cy = [7 2 3 5 0 8]; ↴
>> [x,y] = PontoMedio(cx,cy) ↴

x =
2.8333
y =
4.1667
  
```

A lista de retorno pode ser omitida ou deixada vazia se não houver resultado a ser devolvido, ou seja, pode-se escrever um cabeçalho como

```
function Calcula(x)     ou     function [] = Calcula(x)
```

Como mostrado no exemplo da função **Media** as variáveis passadas para a função não precisam ter os mesmos nomes usados na definição da função.

4.2 Controle de fluxo

Como a maioria das linguagens de programação, o MATLAB permite o uso de estruturas condicionais e repetitivas para controle de fluxo⁵. Neste material discutiremos algumas formas de uso dos comandos ***if***, ***while*** e ***for***.

4.2.1 Estrutura condicional ***if***

O comando ***if*** avalia uma expressão e, dependendo de seu valor, executa um determinado conjunto de instruções. Em sua forma mais simples, usa-se:

```
if expressão
    <COMANDOS>
end
```

Se a expressão lógica for verdadeira (diferente de zero), todos os comandos até o finalizador ***end*** serão executados. Em caso contrário (expressão igual a zero), a execução do código continuará na instrução após o finalizador ***end***. A forma completa da estrutura inclui a declaração ***else*** para a indicação de comandos a serem executados se a expressão for falsa:

```
if expressão
    <COMANDOS V>
else
    <COMANDOS F>
end
```

As expressões podem incluir operadores relacionais e lógicos, como mostrado na tabela 4.2.

Tabela 4.2: Operadores relacionais e lógicos

Operadores relacionais		Operadores lógicos	
<	Menor que	&	Operação "E"
<=	Menor ou igual		Operação "OU"
>	Maior que	~	Negação lógica
>=	Maior ou igual		
==	Igual a		
~=	Diferente de		

Exemplo:

```
...
if x == 0
    y = sin(3*t+a);
else
    y = cos(3*t-a);
end
...
```

4.2.2 Estrutura repetitiva ***while***

⁵ O MATLAB reconhece 6 estruturas de controle de fluxo: *if*, *switch*, *while*, *for*, *try-catch* e *return*.

O laço **while** executa um grupo de comandos enquanto uma expressão de controle for avaliada como verdadeira. A sintaxe para este tipo de laço é:

```
while expressão
    <COMANDOS>
end
```

Exemplo de um trecho de código que simula a operação da função interna **sum**:

```
...
S = 0;
i = 1;
while i<=length(V)
    S = S+V(i);
    i = i+1;
end    % Neste ponto, S = sum(V)
...
```

4.2.3 Estrutura repetitiva **for**

O laço **for** executa repetidamente um conjunto de comandos por um número especificado de vezes. Sua forma geral é:

```
for variável de controle = valor inicial: incremento: valor final
    <COMANDOS>
end
```

O incremento pode ser negativo ou omitido (caso em que será adotado um incremento unitário). Exemplo de um trecho de código que simula a operação da função interna **max**:

```
...
M = V(1);
for i = 2:length(V)    % O incremento foi omitido!
    if V(i) > M
        M = V(i);
    end
end    % Neste ponto, M = max(V)
...
```

4.3 Vetorização

O acesso *vetorizado* é uma opção aos laços de acesso individual a elementos de vetores ou matrizes. Normalmente, o acesso vetorizado é mais rápido que o acesso individual, feito com laços de repetição. Exemplo de uso:

Acesso convencional

```
i = 0;
for t = 0:0.001:99.999
    i = i+1;
    y(i) = sin(t);
end
```

Acesso vetorizado

```
t = 0:0.001:99.999;
y = sin(t);
```

Versões recentes do MATLAB (versão 6 em diante) fazem operações automáticas de otimização que, na prática, produzem ganhos de desempenho compatíveis com os da vetorização.

4.4 Entrada de dados

A função **input** permite a entrada de dados ou expressões durante a execução de *scripts* ou funções, exibindo (opcionalmente) um texto ao usuário. Exemplo:

```
Kp = input('Digite o ganho da ação proporcional: ');
```

Se o valor de entrada for uma expressão, seu valor será avaliado antes da atribuição à variável usada no comando. Se o valor de entrada for um texto o caractere 's' (*string*) deve ser incluído na lista de argumentos da função. Exemplo:

```
Titulo_Grafico = input('Título do gráfico: ','s');
```

Outra forma disponível de interação via teclado é dada pela função **pause**. Quando usada sem argumentos, a instrução interrompe a execução de um *script* ou função até que o usuário pressione alguma tecla⁶. A função **pause** é especialmente útil para permitir ao usuário a leitura de várias informações impressas em tela ou durante a fase de depuração do programa.

4.5 Edição de funções existentes

O código da maioria das funções discutidas neste material pode ser visualizado ou editado, digitando-se:

```
>> edit <nome da função> ↵
```

Não é possível editar o código de funções internas do MATLAB como **inv**, **max**, etc. Apesar de possível, não é recomendável alterar diretamente o código das funções que acompanham o MATLAB. Se desejar⁷, crie uma nova versão com outro nome.

4.6 Subfunções

Os *arquivos-M* podem conter mais de uma função. A primeira delas, cujo nome deve coincidir com o nome do arquivo, é a função **primária** enquanto as demais são **subfunções**. As subfunções podem ser definidas em qualquer ordem após a função primária e suas variáveis sempre tem escopo local⁸. Não é preciso usar qualquer indicação especial de fim de função porque a presença de um novo cabeçalho indica o fim da função (ou subfunção) anterior. Como exemplo, analise o código da função **Baskara**, listado a seguir.

```
% BASKARA.M - Exemplo de uso de uma subfunção para
% cálculo de raízes de equação do 2o grau

% Função primária: mesmo nome que o do arquivo .M
function x = Baskara(v)

a = v(1); b = v(2); c = v(3);      % Obtém coeficientes
D = Delta(a,b,c);                  % Calcula "delta"
```

```
% Calcula raízes reais, se existirem
if isreal(D)
    r1 = (-b+D)/(2*a);    % Calcula raiz
    r2 = (-b-D)/(2*a);    % Calcula raiz
    if r1 == r2
        x = r1;            % Retorna apenas uma raiz
```

⁶ A função **pause** também pode ser usada com argumentos. Quando usada sob a forma **pause(N)**, a função interrompe a execução do código atual por N segundos.

⁷ E souber o que está fazendo.

⁸ É possível criar variáveis globais, reconhecidas em todo o código – consulte a documentação do MATLAB.

```

else
    x = [r1; r2];      % Retorna raízes distintas
end
else
    disp('A equação não possui raízes reais');
    x = [];      % Retorno nulo
end

% Subfunção para cálculo de "delta"
function d = Delta(a,b,c)
d = sqrt(b^2-4*a*c);

```

4.7 Exemplos de aplicação

4.7.1 Aproximações

Crie uma função chamada *Aprox2* que obtenha uma aproximação de 2^a ordem a partir de uma função de transferência de 3^a ordem, da forma:

$$G_3(s) = \frac{n_3}{d_3(s)}.$$

A função deve receber como parâmetros o numerador (um número real) e o polinômio de 3º grau correspondente ao denominador. Os valores de retorno devem ser os polinômios do numerador e denominador da aproximação. Adicionalmente, devem ser superpostos os gráficos da resposta ao degrau das duas funções de transferência. Exemplo de uso:

$$G_3(s) = \frac{7,98}{s^3 + 11,4s^2 + 14s + 7,98}$$

- Instruções:

```

>> n3 = 7.98 ↴
>> d3 = [1 11.4 14 7.98]; ↴
>> [n2 d2] = Aprox2(n3,d3) ↴

```

- Técnica: eliminação do pólo não-dominante (supostamente, o único pólo real).

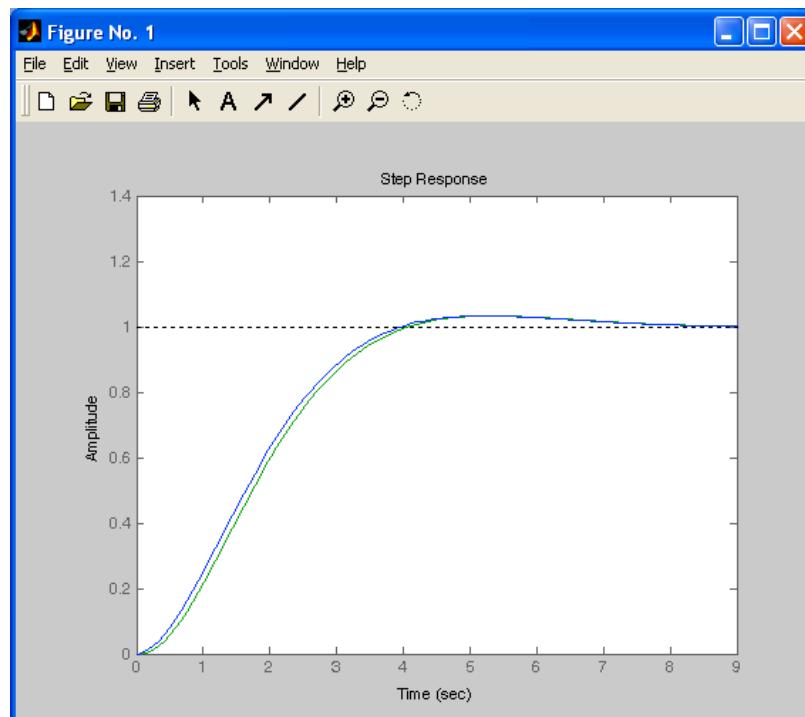
- Valores de retorno da função (para este exemplo)

```

n2 = 0.7908
d2 = 1.0000    1.3090    0.7908

```

- Resultado gráfico: ver exemplo na figura 4.2.

Figura 4.2: Exemplo do resultado gráfico da função *Aprox2*

4.7.2 Análise do erro em regime estacionário

Para sistemas realimentados estáveis o erro em regime estacionário é dado por:

$$e(\infty) = \lim_{t \rightarrow \infty} e(t) = \lim_{s \rightarrow 0} s E(s).$$

Como o MATLAB *sempre* calcula a resposta de sistemas lineares em **tempo discreto**, isto é, para apenas alguns instantes de tempo, pode-se calcular o vetor de erro (**e**) a partir dos vetores de entrada (**u**) e saída (**y**) de uma simulação, usando:

```
e = u-y; % ATENÇÃO: u e y devem ter a mesma dimensão!
```

Um exemplo deste cálculo é mostrado no código da função **Erro**, listado a seguir, que desenha o gráfico do erro e da resposta ao degrau do sistema mostrado na figura 4.3.

```
function [ess] = Erro(K)

clc
G = tf(K,conv([1 1],[1 5])); % Limpa a tela
MF = feedback(G,1); % Cria a função de ramo direto
[y t] = step(MF); % Cria a função em malha fechada
u = ones(length(y),1); % Obtém os vetores de simulação
e = u-y; % Cria o vetor representativo da entrada
plot(t,y,t,e) % Calcula o vetor de erros
% Traça os gráficos da saída e do erro
ess = e(length(e)); % Retorna o valor "final" do erro
```

No código, observe o uso da função **step** com argumentos à esquerda e como a função **plot** foi usada para sobrepor dois gráficos na mesma janela (uma alternativa ao uso de **hold**).

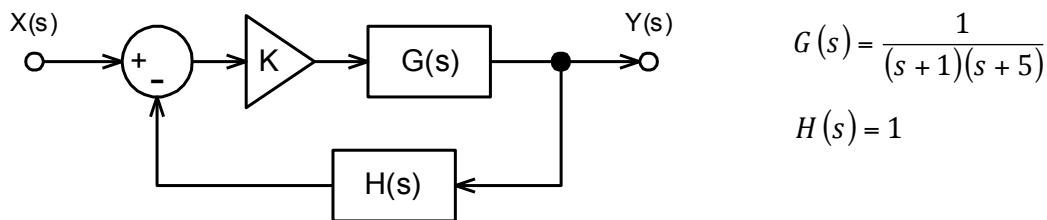


Figura 4.3: Sistema de controle realimentado

4.7.3 Atividade

Modifique o código da função **Erro** para aceitar como argumento de entrada o tempo de simulação e os parâmetros de $G(s)$ e $H(s)$. Verifique a influência do valor de K na resposta dinâmica do sistema.

4.7.4 Atividade – estabilidade em função de um parâmetro

A função **Pulos**, listada a seguir, calcula e desenha em um plano *xy* os pólos do sistema de controle mostrado na figura 4.4, para alguns valores de K entre 0 e 20.

```
% POLOS.M - Função para desenhar a posição dos pólos de um
% sistema realimentado em função do ganho K

function [] = Polos

K = [0:0.5:20]; % Cria vetor de ganhos
for i=1:length(K)
    q = [1 2 4 K(i)]; % Polinômio da equação característica
    p(:,i) = roots(q); % Calcula raízes para o ganho K(i)
end
plot(real(p),imag(p), 'bx'); % Desenha gráfico dos pólos
grid on;
xlabel('Eixo real'); % Nomeia os eixos
ylabel('Eixo imaginário'); % do gráfico
```

a) Analise o gráfico criado pela função e verifique se existem valores de K para os quais o sistema realimentado é instável.

b) Faça com que a função aceite a entrada dos valores do ganho máximo e do passo. Exemplo:

```
Kmax = input('Digite o valor do ganho máximo: ');
```

c) Modifique a função para detectar o valor de K que leva o sistema ao limite da estabilidade.

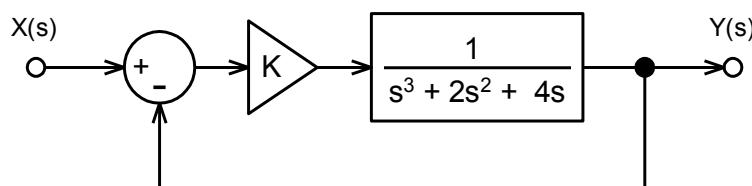


Figura 4.4: Sistema de controle realimentado

5. Lugar das raízes – introdução

O desempenho de um sistema linear pode ser analisado pelos pólos da sua função de transferência em malha fechada. Na prática, tenta-se ajustar a posição destes pólos para que o comportamento do sistema atenda a certas especificações de desempenho (sobressinal, tempo de assentamento, etc). Por exemplo, considere o sistema representado na figura 5.1.

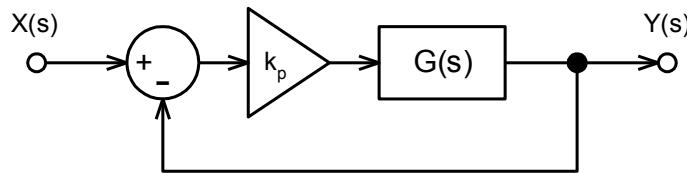


Figura 5.1: Sistema de controle em malha fechada

Neste caso, percebe-se que as raízes da equação característica do sistema, dada por

$$q(s) = 1 + k_p G(s) = 1 + k_p \frac{n(s)}{d(s)} = 0$$

dependem do valor do ganho K_p . O traçado do **lugar das raízes** de um sistema em malha fechada, em função de um ganho de ramo direto, pode ser obtido diretamente no MATLAB pela função **rlocus**. A partir da função de transferência em malha aberta,

$$G(s) = \frac{n(s)}{d(s)}$$

obtém-se o lugar das raízes do sistema em malha fechada, usando

```
rlocus(n, d); ou rlocus(G); % Sendo G = tf(n, d)
```

Neste caso, o MATLAB utiliza um vetor de ganhos criado automaticamente, com k_p variando de 0 a $+\infty$. Este vetor também pode ser fornecido pelo usuário, como no exemplo a seguir:

```
VG = [0:0.5:20]; % Cria vetor com ganhos de 0 a 20
rlocus(n, d, VG); % Obtém o lugar das raízes
```

O resultado é apresentado em uma janela de figura com os pólos e zeros de malha aberta indicados por "x" e "o", respectivamente. A função **rlocus** também pode fornecer as raízes calculadas e o vetor de ganhos utilizado nos cálculos, sem traçar o gráfico, usando-se:

```
[R Kp] = rlocus(n, d) % R = matriz de raízes; Kp = vetor de ganhos
```

Analizando-se estes valores de retorno é possível descobrir, por exemplo, quais valores de k_p podem tornar o sistema instável.

5.1 Exemplo

Considere um sistema como o da figura 5.1, com:

$$G(s) = \frac{1}{s^3 + 4s^2 + 6s + 1}.$$

O lugar das raízes deste sistema pode ser obtido, digitando-se:

```
>> MA = tf(1, [1 4 6 1]); ↴
>> rlocus(MA); ↴
```

O resultado gráfico está representado na figura 5.2. As setas indicam o sentido de deslocamento dos pólos de malha fechada do sistema com o aumento de k_p .

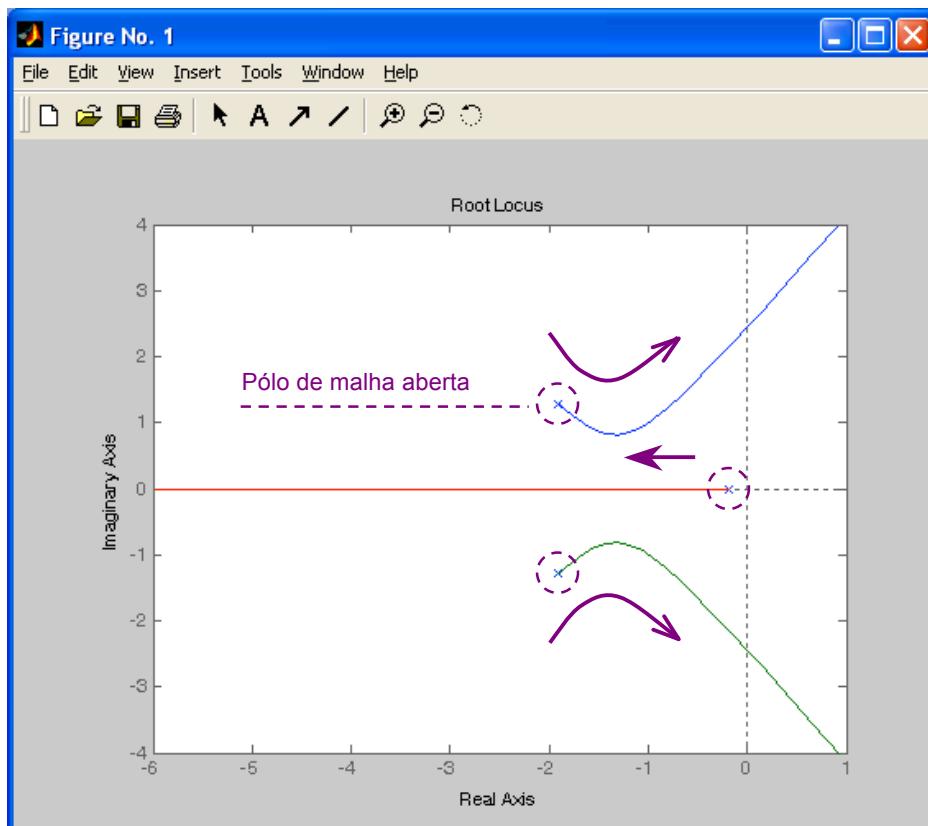


Figura 5.2: Lugar das raízes

5.1.1 Atividade

Descubra quantos (e quais) valores de ganho k_p foram usados pela função **rlocus** para a obtenção do lugar das raízes do sistema da figura 5.2.

5.2 Análise gráfica do lugar das raízes

A análise dos pólos do sistema em relação ao ganho k_p também pode ser feita de forma gráfica pelo uso da função **rlocfind**. Esta função calcula o ganho de ramo direto necessário para que um certo polo, selecionado pelo *mouse* na janela do lugar das raízes, seja obtido.

Para ver um exemplo de como isto pode ser feito, obtenha o lugar das raízes do sistema desejado e digite:

```
>> [g p] = rlocfind(MA) ↴
```

A mensagem "Select a point in the graphics window" será mostrada na janela de comando do MATLAB e um cursor e forma de cruz aparecerá na janela de figura, aguardando que um ponto seja selecionado. No exemplo a seguir (figura 5.3) foi considerado o lugar das raízes do sistema descrito como exemplo no item 5.1.

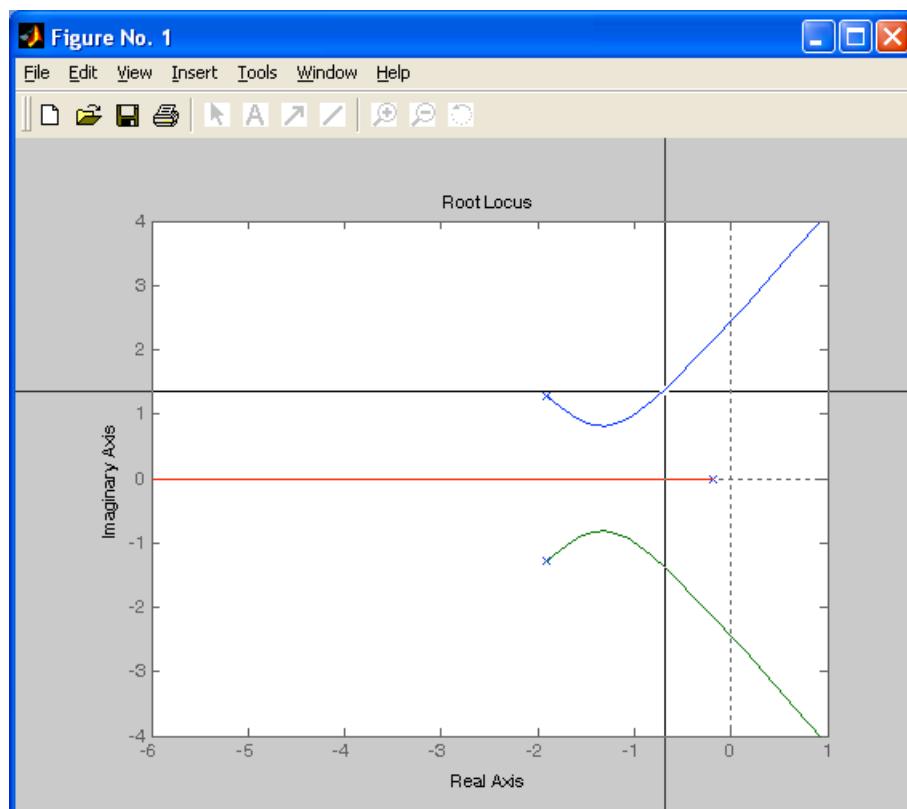


Figura 5.3: Seleção de um ponto sobre o lugar das raízes

Os resultados correspondentes são exibidos na janela de comando assim que um ponto for selecionado. Para o exemplo da figura 5.3, tem-se:

```
selected_point = -0.6836 + 1.3540i
g = 5.1271
p = -2.5984
-0.7008 + 1.3664i
-0.7008 - 1.3664i
```

Os pólos calculados no vetor **p** ficam destacados na janela de figura até que o gráfico seja redesenhado. Importante: a função **rlocfind** só funciona se existir uma janela de lugar das raízes criada pelo comando **rlocus**.

5.2.1 Atividade

Use a função **rlocfind** para estimar o ganho k_p que leva o sistema definido em 5.1 ao limite da estabilidade.

5.3 Exemplos

Obs.: todas as análises a seguir se referem a sistemas com a estrutura mostrada na figura 5.1.

- a) Determine o ganho crítico (limite da estabilidade) para um sistema com:

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

Na condição crítica, quais pólos são dominantes?

- b) Considere um sistema realimentado definido por:

$$G(s) = \frac{1}{s(s+2)(s^2 + 4s + 5)}.$$

Verifique se é possível obter pólos dominantes com parte real igual a -0,35 e determine para qual valor de k_p isto acontece.

- c) Considere um sistema definido pela função de transferência:

$$G(s) = \frac{(s+1)}{s^2(s+9)}.$$

Determine o valor do ganho k_p para o qual os três pólos de malha fechada são reais e iguais.

- d) Um colega traçou o lugar das raízes de um sistema com

$$G(s) = \frac{s+20}{s(s^2 + 24s + 144)},$$

para determinar a faixa de valores de k_p que tornassem o sistema realimentado oscilatório. Usando a função **rlocfind** e selecionando o ponto em que os pólos do sistema (realimentado) passavam a ter parte complexa, obteve:

```
selected_point = -4.7698
```

```
g = 16.3718
p = -14.4939
      -4.7698
      -4.7363
```

O procedimento adotado pelo colega está correto, ou seja, o sistema em questão terá realmente comportamento oscilatório para $k_p \approx 16,37$? Justifique.

- e) Considere o sistema de controle mostrado na figura 5.4 e os controladores:

i) $G_c(s) = k_p$ (Controlador proporcional)

ii) $G_c(s) = k_i / s$ (Controlador integral)

iii) $G_c(s) = k_p + \frac{k_i}{s} = k_{pi} \left(\frac{s+1}{s} \right)$ (Controlador proporcional-integral *simplificado*)

Obtenha o lugar das raízes de cada sistema compensado. Discuta a estabilidade dos sistemas em função do ganho e, quando possível, determine o valor do ganho crítico.

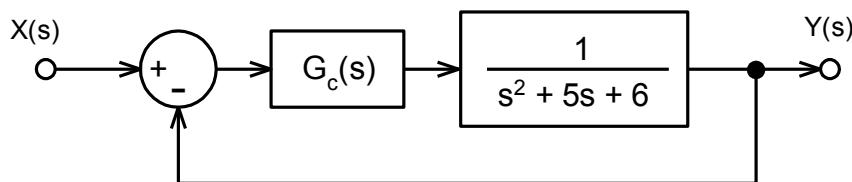


Figura 5.4: Sistema de controle em malha fechada

- f) A função **rlocus** do MATLAB utiliza a estrutura mostrada na figura 5.5.

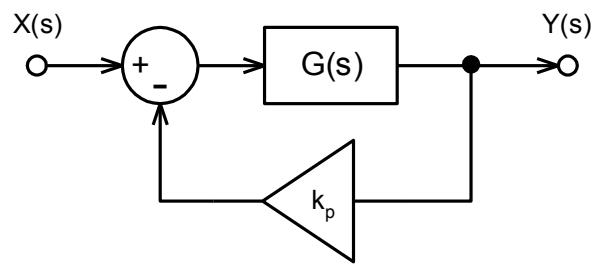


Figura 5.5: Estrutura usada na análise de lugar das raízes

Mostre que, para a análise de lugar das raízes, esta estrutura é equivalente à da figura 5.1.

6. Projeto usando lugar das raízes – introdução

O MATLAB permite o projeto interativo de sistemas de controle por meio de uma ferramenta gráfica chamada *SISO Design Tool*. A abreviatura SISO indica que o projeto é limitado a sistemas com apenas uma entrada e uma saída (*single-input, single-output*). É possível usar como base de projeto o diagrama do lugar das raízes do sistema, o diagrama de Bode ou o diagrama de Nichols. A configuração padrão para o projeto está representada na figura 6.1.

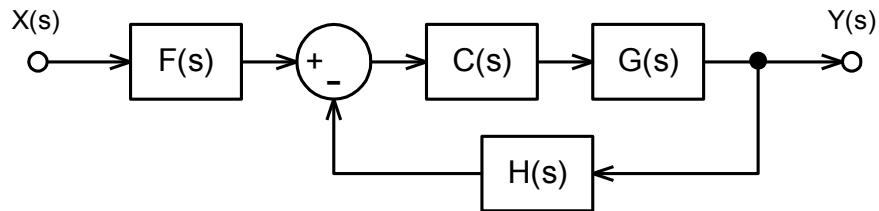


Figura 6.1: Configuração padrão para a ferramenta de projeto SISO

Nesta configuração:

- $F(s)$ é o pré-filtro do sistema;
- $C(s)$ é o compensador (ou controlador);
- $G(s)$ representa a planta ou processo controlado;
- $H(s)$ representa o elemento de medição (sensor).

As funções de transferência do compensador e da planta, geralmente, são importadas do *workspace*, como será mostrado a seguir. Por padrão, adota-se $F(s)=H(s)=1$.

6.1 Exemplo de projeto

Considere o projeto de um controlador proporcional-derivativo (PD) para uma planta com a seguinte função de transferência:

$$G_p(s) = \frac{1.000}{s(s + 125)}.$$

O sistema controlado deve apresentar:

- i) Erro estacionário nulo para entrada em degrau;
- ii) Sobressinal máximo de 10%;
- iii) Tempo de subida menor ou igual a 10 ms.

Resolução: a função de transferência do controlador PD é dada por

$$G_{pd}(s) = k_p + k_d s = k_p(1 + T_d s),$$

sendo $T_d = k_d/k_p$, o *tempo derivativo* do controlador.

Para utilizarmos o diagrama de lugar das raízes no projeto é preciso que exista apenas um parâmetro livre. Assim, incluiremos a contribuição dinâmica do controlador na planta, para um valor fixo de T_d , de modo que o ganho de ramo direto seja apenas k_p . (ver figura 6.2).

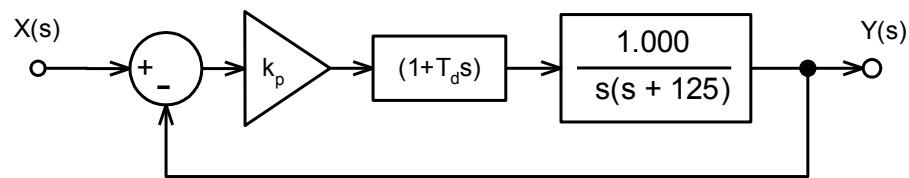


Figura 6.2: Sistema de controle com ganho de ramo direto ajustável

Assim, as funções de transferência necessárias para o uso da ferramenta de projeto são:

$$F(s) = H(s) = 1; \quad C(s) = k_p; \quad G(s) = \frac{1.000(1 + T_d s)}{s(s + 125)}$$

Neste exemplo, o zero do sistema (em $s = -1/T_d$) será posicionado arbitrariamente entre os pólos de malha aberta, com $T_d = 1/80$ (ver figura 6.3).

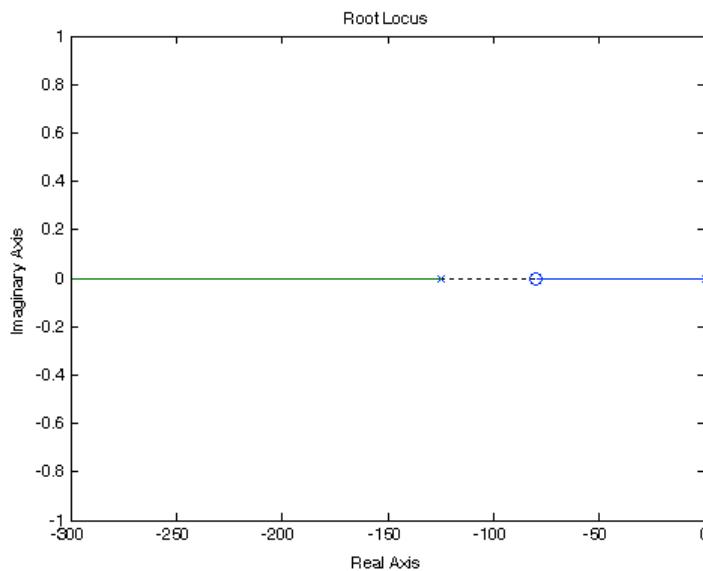


Figura 6.3: Lugar das raízes do sistema para $T_d = 1/80$

Para executar a ferramenta de projeto com o diagrama de lugar das raízes do sistema usa-se a função ***rltool***. Seqüência de comandos:

```
>> Td = 1/80; ↵
>> n = 1000*[Td 1]; ↵
>> d = conv([1 0],[1 125]); ↵
>> G = tf(n,d); ↵
>> rltool(G); ↵
```

A interface da ferramenta de projeto será mostrada (ver figura 6.4) com os pólos correspondentes ao valor atual de k_p (valor inicial = 1) destacados.

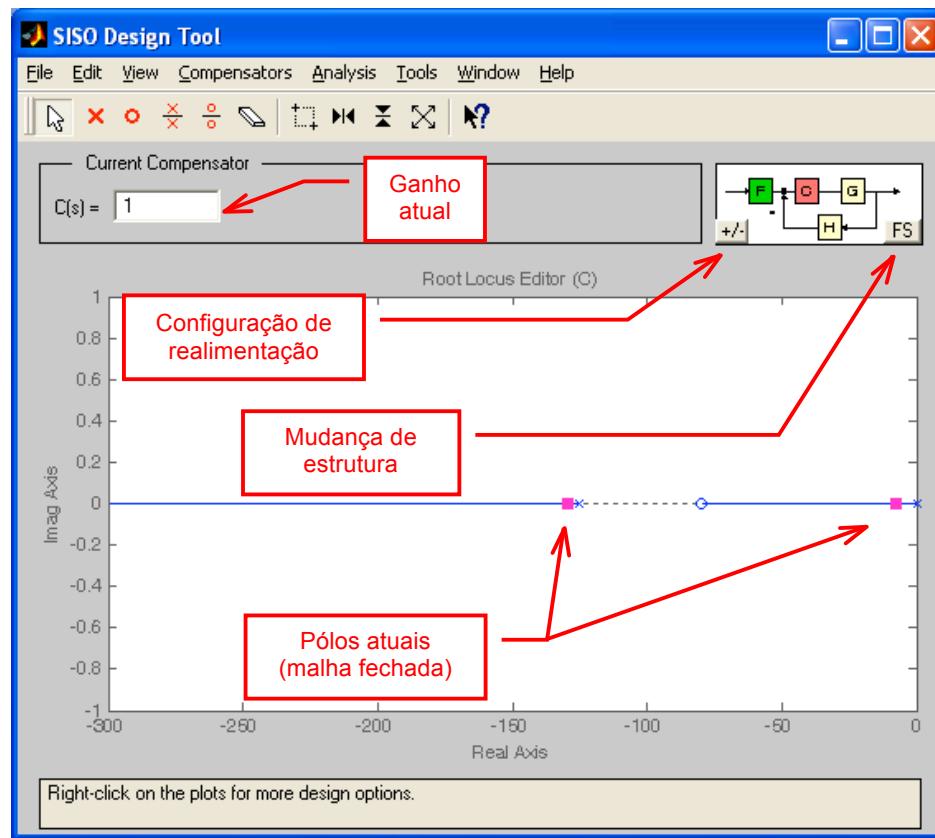


Figura 6.4: Interface da ferramenta de projeto SISO

Para verificar se os critérios de desempenho estão sendo satisfeitos abra uma janela com a resposta ao degrau do sistema controlado, clicando em *Analysis > Response to Step Command*. Inicialmente, são mostrados os gráficos da saída do sistema e do esforço de controle (saída do controlador), conforme a figura 6.5.

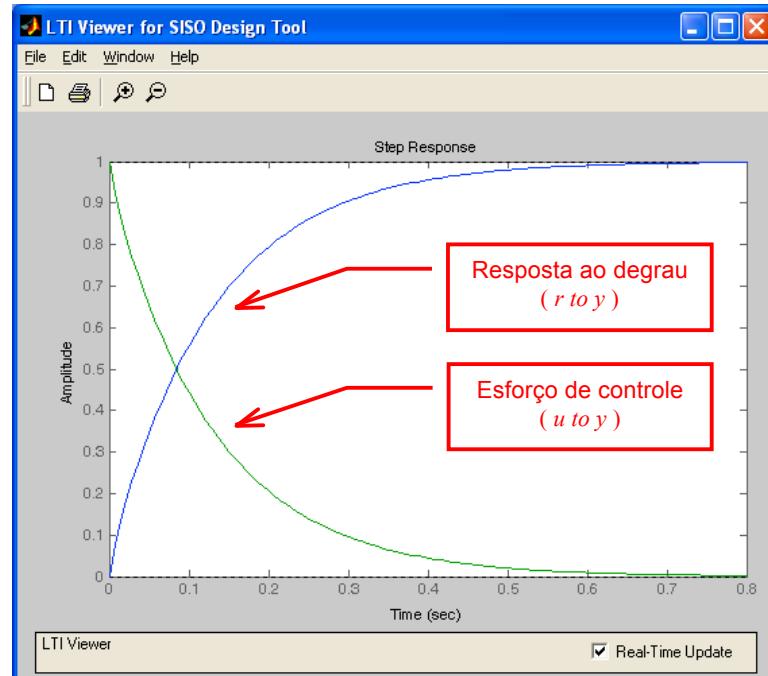


Figura 6.5: Resposta ao degrau do sistema controlado

As curvas que aparecem na janela de resposta podem ser alteradas clicando-se em *Analysis > Other Loop Responses*. No momento, oculte o gráfico do *esforço de controle*, desmarcando a opção "r to u (FCS)" (ver figura 6.6).

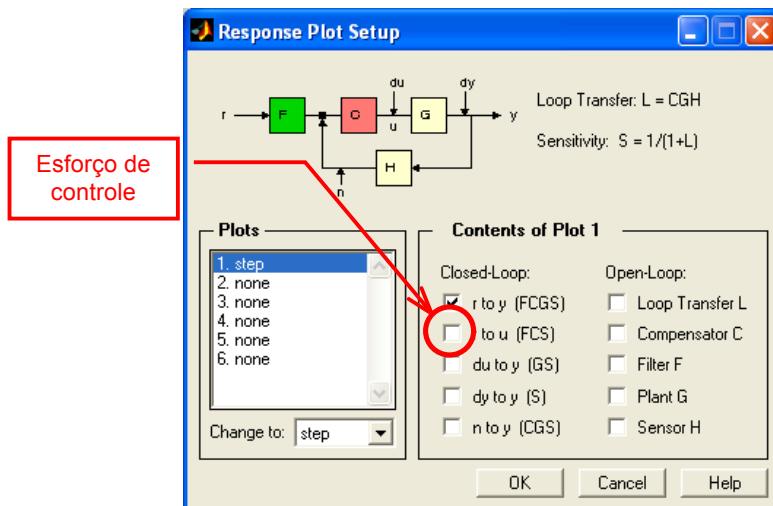


Figura 6.6: Configuração da resposta ao degrau

Clique com o botão direito na janela de resposta ao degrau e selecione *Characteristics > Rise Time*. Para $k_p = 1$ o tempo de subida deve ser de aproximadamente 0,283 s.

Mantenha as janelas da resposta e da ferramenta de projeto lado a lado e altere o ganho k_p arrastando um dos pólos atuais do sistema. O novo valor de ganho também pode ser digitado na caixa de edição "Current Compensator". O desempenho exigido neste projeto deve ser obtido para $k_p \geq 27$. Se for necessário repetir o projeto, por exemplo, com outro valor de T_d , altere a função de transferência $G(s)$ e importe-a para a ferramenta de projeto, clicando em *File > Import* e selecionando a variável G como "função da planta" (ver figura 6.7).

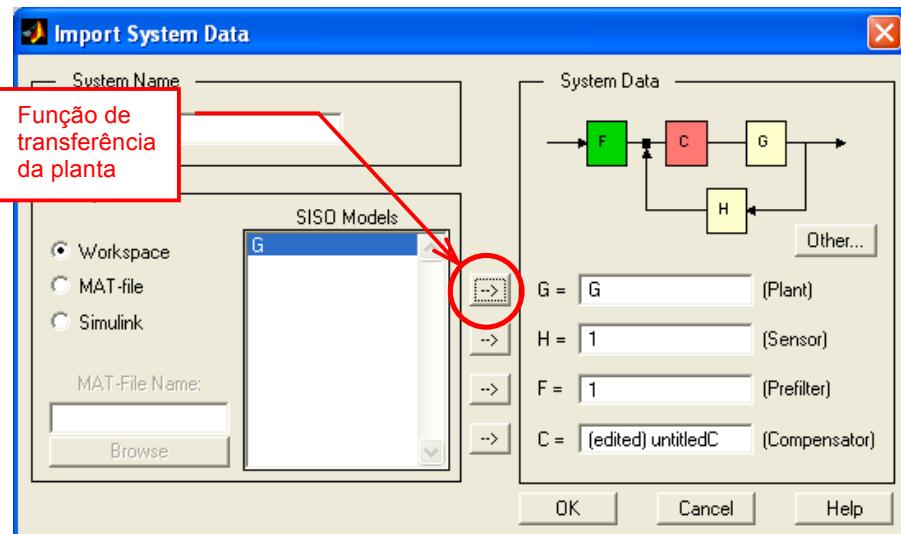


Figura 6.7: Alteração da função de transferência da planta

6.1.1 Atividades

- Repita o projeto com $T_d = 1/200$. Verifique se ainda é possível atender aos critérios de desempenho exigidos e discuta os resultados obtidos.
- Ainda com $T_d = 1/200$, repita o projeto usando um controlador proporcional. Discuta os resultados obtidos.

c) Usando os mesmos critérios de desempenho dos projetos anteriores, projete um controlador proporcional-integral (PI) para a planta:

$$G_p(s) = \frac{1.000}{(s + 125)(s + 500)}.$$

A função de transferência do controlador PI é:

$$G_{pi}(s) = k_p + \frac{k_i}{s} = k_p \left(1 + \frac{1}{T_i s} \right) = k_p \left(\frac{s + 1/T_i}{s} \right),$$

Adote um valor para o tempo integral e determine o valor de k_p que permita atender às especificações de desempenho propostas. Justifique a escolha feita para a posição do zero em relação aos pólos do sistema.

7. Resposta em freqüência – introdução

O estudo da resposta em freqüência de um sistema linear estável visa determinar como sua saída em *regime permanente* varia em função da freqüência ω de um sinal de entrada puramente senoidal. Nestas condições, a saída e todos os sinais internos do sistema são senoidais com a mesma freqüência do sinal de entrada, diferindo apenas em *amplitude* e *fase*.

7.1 Determinação manual da resposta em freqüência

Considere um sistema linear estável definido por uma função de transferência conhecida, $G(s)$, submetido a uma entrada senoidal da forma $x(t) = A \operatorname{sen}(\omega t)$. Neste caso, a saída em regime permanente é da forma:

$$y(t) = A |G(\omega)| \operatorname{sen}(\omega t + \phi).$$

O ganho do sistema, dado por

$$G(j\omega) = G(s) \Big|_{s=j\omega},$$

é um **número complexo** cujo módulo e fase podem ser calculados como

$$|G(j\omega)| = \sqrt{\operatorname{Im}(G(j\omega))^2 + \operatorname{Re}(G(j\omega))^2} \quad \text{e} \quad \phi = \operatorname{arctg}(\operatorname{Im}(G(j\omega)) / \operatorname{Re}(G(j\omega))),$$

sendo $\operatorname{Re}(G(j\omega))$ e $\operatorname{Im}(G(j\omega))$, respectivamente, a parte real e imaginária do ganho $G(j\omega)$.

7.1.1 Atividades

a) Crie um *script* ou função que calcule o valor da amplitude e fase (em graus) do ganho de um sistema definido por

$$G(s) = \frac{10.000}{s^2 + 141,4s + 10.000} \Rightarrow G(j\omega) = \frac{10.000}{(10.000 - \omega^2) + (141,4\omega)j},$$

para entradas senoidais de freqüências: 1, 10, 100, 1.000 e 10.000 rad/s. Use as funções **abs** e **angle** nos cálculos e converta os ângulos de radianos para graus, multiplicando os valores por 180 e dividindo por π . O que se pode concluir à respeito do ganho do sistema?

b) Analise o código listado a seguir.

```

function Ordem1(n,d,wmax)
w = logspace(-2,ceil(log10(wmax)),100);
for k=1:100
    G = n/(d(1)*w(k)*i+d(2));
    Mod(k) = abs(G);
    Fase(k) = angle(G)*180/pi;
end
figure(1); semilogx(w,Mod); title('Modulo');
figure(2); semilogx(w,Fase); title('Fase (graus)');

```

Teste a função para **n=1**, **d=[1 1]** e **wmax = 100**. Explique, usando suas próprias palavras, qual é a utilidade da função.

c) Crie uma nova versão da função **Ordem1** que use acesso vetorizado (ver seção 1.3, pág. 6).

d) Elabore uma função (*RespFreq*) que desenhe os gráficos da resposta em freqüência (módulo e fase do ganho) de um sistema linear da forma:

$$G(s) = \frac{n(s)}{d(s)} = \frac{K}{as^2 + bs + c}.$$

Sintaxe da função

```
[Mod, Fase, w] = RespFreq(n, d, wmax);
```

Dados de entrada

- Função de transferência (polinômios do numerador e denominador);
- Freqüência máxima a ser usada nos cálculos (em rad/s).

Valores de retorno

- Vetores de módulo e fase dos ganhos calculados;
- Vetor de freqüências usadas nos cálculos.

Considerações

- Freqüência mínima = 0,01 rad/s;
- 1.000 valores de freqüência;
- Gráfico de fases em graus.

e) Altere o código de *RespFreq* para trabalhar com funções de transferência genéricas.

7.2 Diagrama de Bode

A representação gráfica do ganho de um sistema (módulo e fase) em função da freqüência do sinal de entrada fornece informações importantes sobre seu comportamento dinâmico. Normalmente os gráficos de módulo e fase usam *escalas logarítmicas* de freqüência, que facilitam a representação de grandes variações de ω . Além disso, é usual representar os valores de ganho em **decibéis** (dB), calculados como:

$$G_{dB} = 20 \log_{10} |G(\omega)|.$$

Os gráficos desenhados sob as especificações anteriores formam o **diagrama de Bode** de um sistema, que pode ser obtido diretamente no MATLAB pelo uso da função *bode*.

7.2.1 Exemplo

A seqüência de comandos a seguir desenha o diagrama de Bode do sistema definido por:

$$G(s) = \frac{10.000}{s^2 + 141,4s + 10.000}.$$

Comandos:

```
>> n = 1e4; ↴
>> d = [1 141.4 n]; ↴
>> G = tf(n,d); ↴
>> bode(G); ↴ % Opção: bode(n,d);
```

O resultado é representado em uma janela gráfica, conforme a figura 7.1.

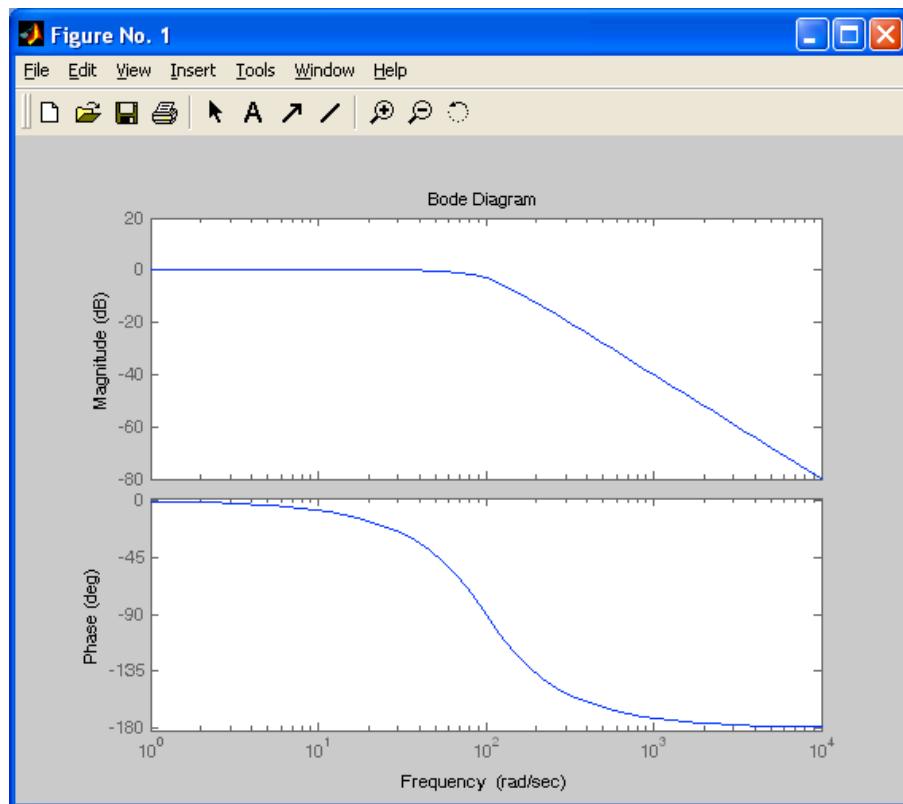


Figura 7.1: Diagrama de Bode

Neste caso, as freqüências usadas na criação dos gráficos são determinadas automaticamente pelo MATLAB. Como outras funções com resultados gráficos, a função **bode** também aceita mais de uma função de transferência como argumento. Neste caso, os diagramas são sobrepostos na mesma janela de figura. Os valores calculados pela função **bode** (ou seja, a resposta em freqüência) podem ser obtidos, sem o traçado dos gráficos, usando-se a sintaxe:

```
[mod,fase,w] = bode(G) % Sendo G = tf(n,d), um sistema linear
```

Nesta forma de uso, os ganhos retornados pela função são *adimensionais*, e não em decibéis. O vetor de freqüências a ser usado nos cálculos pode ser fornecido pelo usuário, como no exemplo a seguir (que aproveita os polinômios do exemplo anterior):

```
>> G = tf(n,d); ↴
>> w = logspace(0,3,200); ↴
>> bode(G,w); ↴
```

7.3 Sistemas de 2^a ordem

Muitos sistemas lineares podem ser representados ou aproximados por funções de transferência de 2^a ordem padrão, da forma:

$$G(s) = K \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}.$$

Desconsiderando-se o ganho K , cujo efeito pode ser incluído posteriormente, pode-se traçar o diagrama de Bode do sistema para uma freqüência normalizada ($u = \omega/\omega_n$) em função da relação de amortecimento ξ (ver figura 7.2).

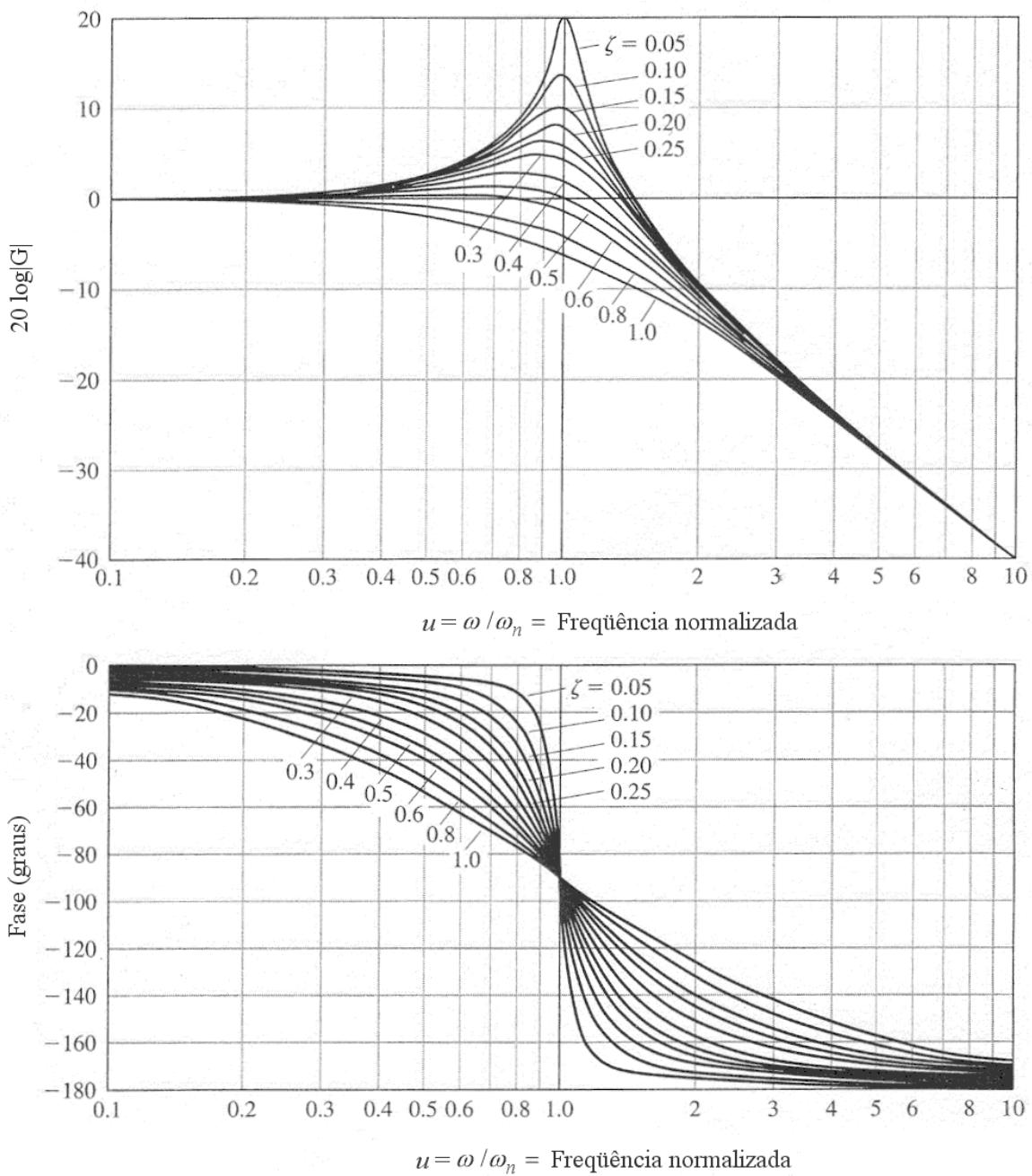


Figura 7.2: Diagrama de Bode para sistemas de 2^a ordem padrão

Em geral, há três parâmetros de interesse na resposta em freqüência de um sistema de 2^a ordem:

- O pico de ressonância (ganho máximo) do sistema, denominado $M_{p\omega}$ (em dB);
- A freqüência de ressonância (ω_r) em que o pico de ressonância ocorre;
- A largura de banda passagem do sistema (ω_B), definida pela freqüência em que o ganho cai 3 dB abaixo de seu valor em baixas freqüências.

Esses parâmetros podem ser usados para estabelecer critérios de desempenho no domínio do tempo, apesar de não existirem formulações exatas que relacionem a resposta em freqüência com a resposta temporal de um sistema.

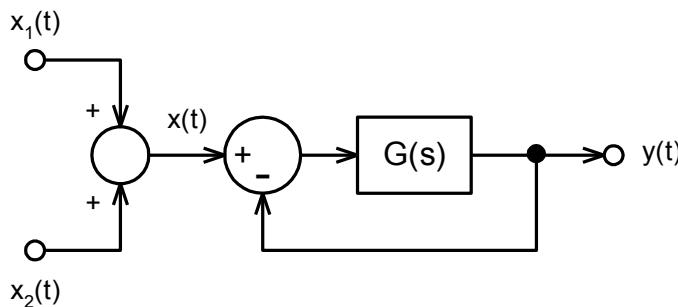
7.3.1 Atividades

a) Determine o pico de ressonância, a freqüência de ressonância e a largura de banda dos sistemas definidos por:

$$i) \quad G_1(s) = \frac{4,494}{s^2 + 1,484s + 4,494}$$

$$ii) \quad G_2(s) = \frac{s + 1}{s^2 + s + 2}$$

b) A saída de um sistema linear excitado por uma entrada que não seja puramente senoidal pode ser obtida pela composição dos efeitos associados a cada *componente espectral* do sinal de entrada. Por exemplo, considere o sistema da figura 7.3.



$$x_1(t) = \text{sen}(5t)$$

$$x_2(t) = 0,5 \text{sen}(50t)$$

$$G(s) = \frac{1.000}{s^3 + 20s^2 + 200s + 1.000}$$

Figura 7.3: Sistema realimentado com duas entradas senoidais

Obtenha a simulação da saída $y(t)$ usando a função **lsim** e verifique a atuação do sistema como um *filtro de freqüências*. Sugestão para o vetor de tempo de simulação: $t = 0: 0.005: 5;$

c) Simule a saída do sistema anterior para cada sinal de entrada, $x_1(t)$ e $x_2(t)$, isoladamente. Verifique a defasagem e atenuação sofridas em cada caso e confira os resultados usando o diagrama de Bode do sistema realimentado.

8. Simulink

O Simulink é um programa que funciona de forma integrada ao MATLAB, usado para modelagem e simulação de sistemas dinâmicos lineares ou não-lineares, em tempo contínuo, tempo discreto ou uma combinação dos dois modos. Os resultados das simulações podem ser visualizados, gravados em variáveis do MATLAB ou em arquivos de dados.

8.1 Exemplo de uso

A criação de modelos no Simulink é feita de forma gráfica pelo posicionamento, interligação e configuração de blocos funcionais. Para ilustrar o uso do Simulink será mostrado como obter a simulação da resposta ao degrau do sistema representado na figura 8.1.

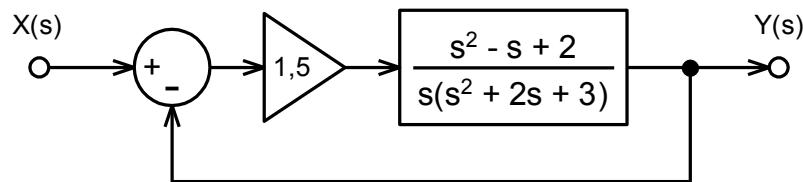


Figura 8.1: Sistema de controle em malha fechada

Inicie o Simulink a partir da linha de comando (digitando **simulink**) ou clicando no ícone do programa na barra de comandos do MATLAB. A janela principal do Simulink será exibida com as bibliotecas de blocos disponíveis para uso.

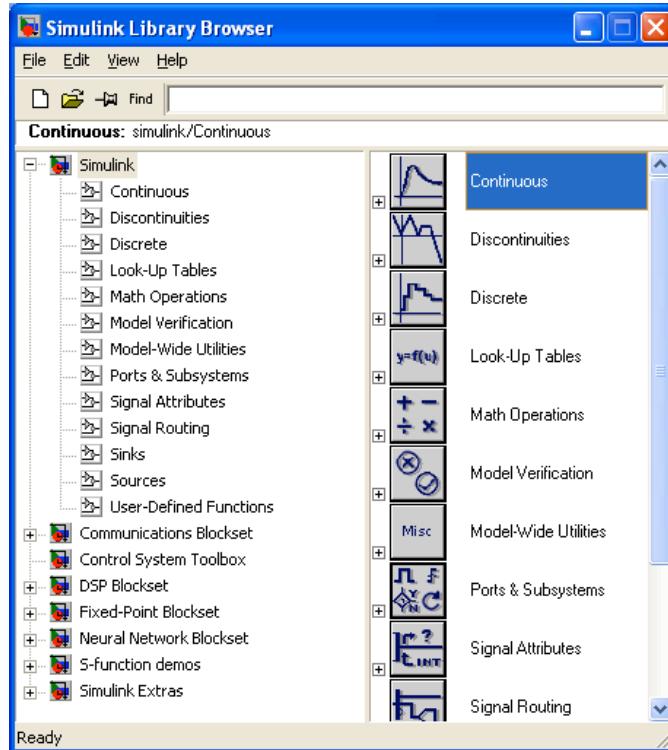


Figura 8.2: Bibliotecas de blocos do Simulink

Uma janela para edição de um novo modelo (*untitled*) deve ser aberta automaticamente. Se isso não acontecer, clique em *File > New > Model* na janela do Simulink.

Os blocos necessários para a simulação do exemplo estão nas bibliotecas **Sources**, **Sinks**, **Continuous** e **Math Operations**. Por exemplo, a figura 8.3 mostra alguns blocos da biblioteca de geradores de sinais.

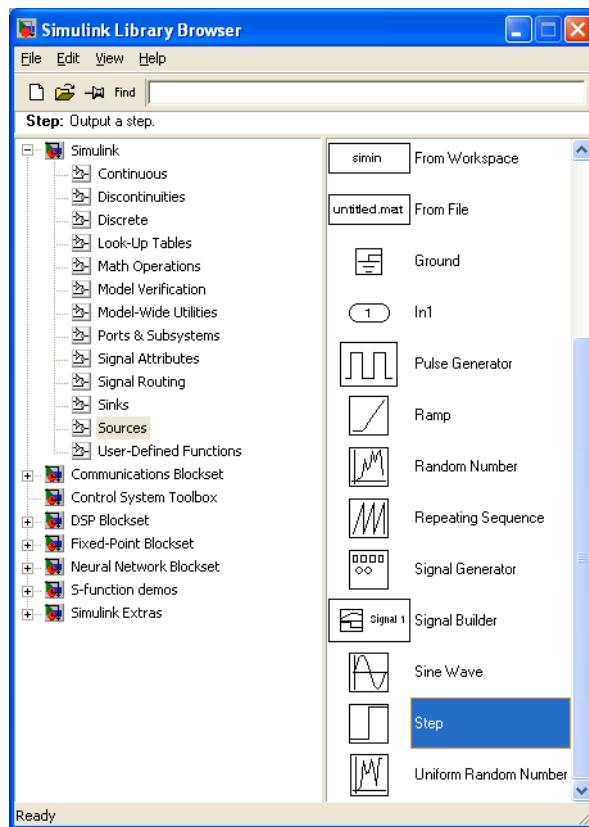


Figura 8.3: Biblioteca de blocos geradores de sinais

Para inserir um gerador de sinal tipo degrau no sistema basta clicar no bloco **Step** e arrastá-lo para a janela do modelo (ver figura 8.4).

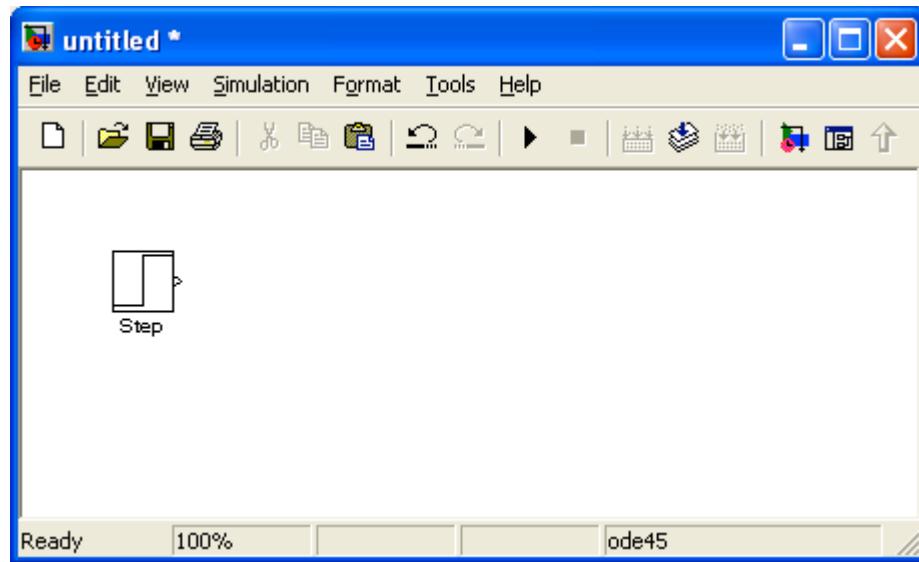


Figura 8.4: Janela do modelo em edição

Em seguida, insira um somador no modelo a partir da biblioteca **Math Operations** e dê um *duplo-clique* no somador para editar suas propriedades. Altere a lista de sinais para '+ -' (sem as aspas) e clique em **OK** (ver figura 8.5).

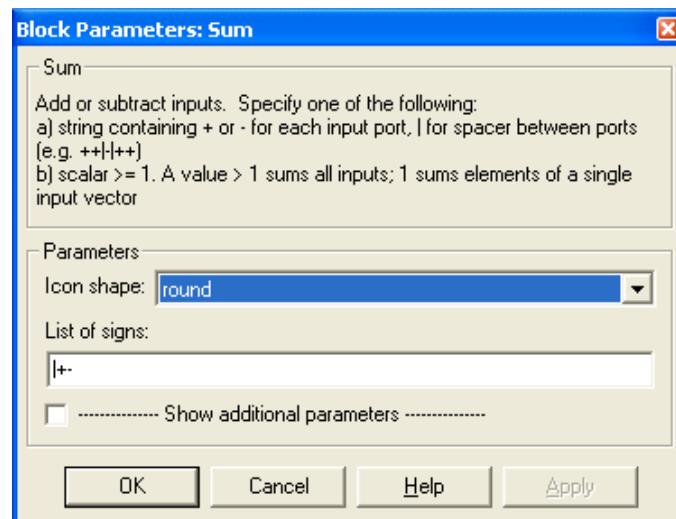


Figura 8.5: Caixa de diálogo com as propriedades do bloco somador

Crie uma ligação entre o gerador e a entrada '+' do somador. Para isso, posicione o *mouse* sobre a saída do gerador até que o cursor mude para a forma de uma cruz. Em seguida, clique e arraste o cursor até a entrada do somador – o cursor deve mudar para uma cruz dupla – para completar a ligação (ver figura 8.6).

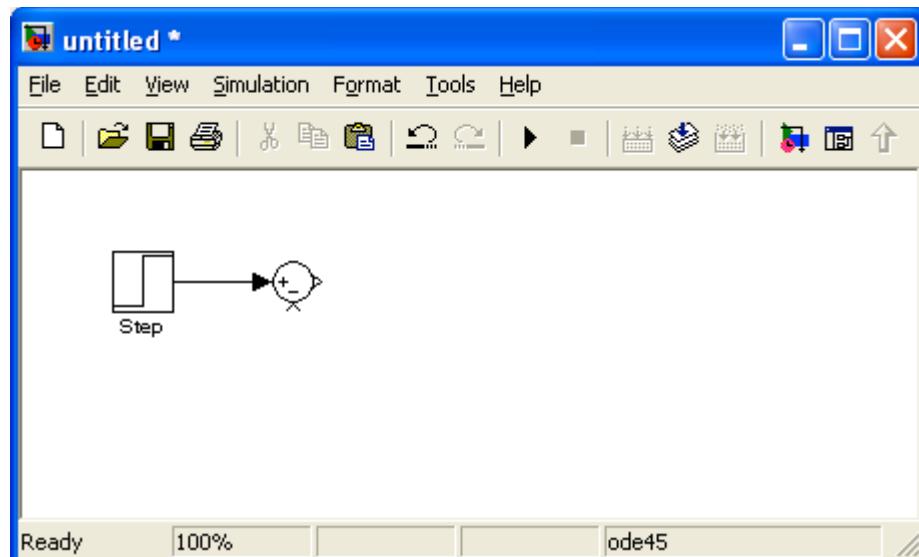


Figura 8.6: Ligação entre dois blocos

Insira os outros blocos no sistema (navegue pelas bibliotecas **Continuous** e **Sinks**) e complete as ligações até obter um modelo semelhante ao da figura 8.7. As técnicas válidas na maioria dos programas para ambiente Windows, como seleção múltipla de objetos, redimensionamento e movimentação também funcionam no Simulink. Operações de rotação e espelhamento estão disponíveis em menus acionados pelo clique do botão direito do *mouse* sobre os blocos.

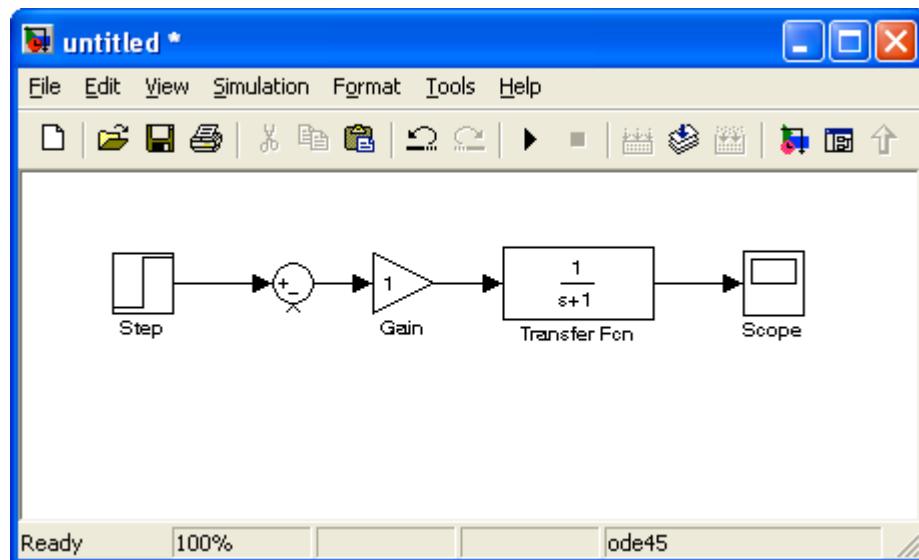


Figura 8.7: Modelo de simulação parcialmente construído

Crie a ligação de realimentação ligando a entrada do somador com a linha de ligação entre o bloco da função de transferência e o osciloscópio (*scope*) – podem ser necessárias algumas tentativas. Outra maneira de se fazer esta ligação é posicionar o cursor sobre a linha de ligação, pressionar a tecla **Ctrl** e arrastar o mouse até a entrada do somador. O processo de clicar e arrastar pode ser feito em etapas para que as linhas de ligação sejam posicionadas de forma conveniente.

Para completar o modelo, edite o valor do ganho do bloco amplificador e altere a função de transferência, digitando os polinômios do numerador ([1 -1 2]) e do denominador ([1 2 3 0]). Se desejar, edite os nomes dos blocos para descrever a função de cada um. Finalmente, grave o modelo como **sistema_01.mdl** antes de gerar a simulação. O resultado final deve ser semelhante ao mostrado na figura 8.8.

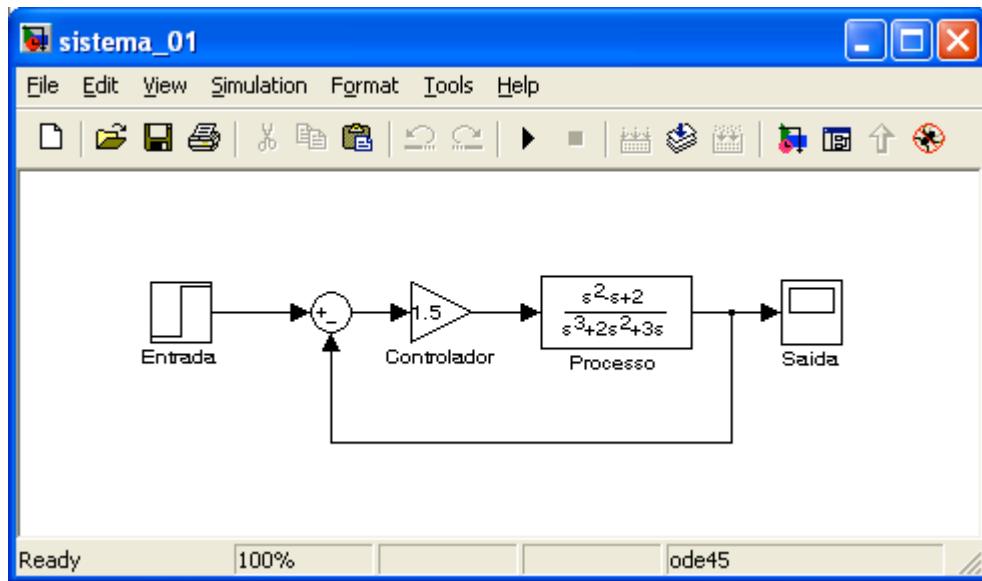


Figura 8.8: Modelo de simulação completo

8.1.1 Configurando os parâmetros de simulação

Clique em *Simulation > Parameters* para acessar as opções de simulação (ver figura 8.9).

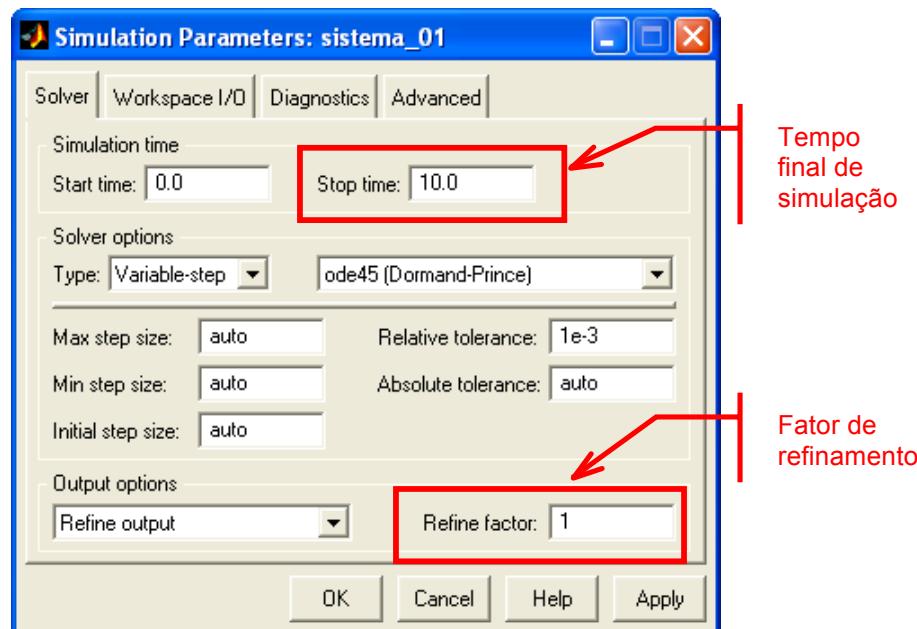


Figura 8.9: Parâmetros de simulação

Os botões da caixa de diálogo têm as seguintes funções:

- **Apply** – aplica os parâmetros de simulação atuais ao sistema, mesmo que exista uma simulação em curso, mantendo a caixa de diálogo aberta;
- **Cancel** – retorna os parâmetros da simulação aos últimos valores ajustados;
- **Help** – abre uma janela de ajuda para os comandos da caixa de diálogo;
- **OK** – aplica os parâmetros de simulação ao sistema e fecha a caixa de diálogo.

Algumas opções de interesse da aba **Solver**:

- Simulation time** – Intervalo de tempo em que a simulação é feita. O tempo que a simulação leva para ser completada depende de fatores como a complexidade do modelo e capacidade de processamento do computador.
- Solvers** – A simulação de um sistema dinâmico envolve a integração numérica de sistemas de equações diferenciais ordinárias. Para isso, o Simulink oferece vários métodos de resolução com passos de integração fixos ou variáveis. Normalmente, o algoritmo de passo variável **ode45**, fundamentado nos métodos de Runge-Kutta, fornece bons resultados.
- Step sizes** – É possível controlar os valores dos passos de integração dos algoritmos de passo variável, como **ode45**. Como regra geral, pode-se deixar o controle desses valores a cargo do Simulink em uma primeira simulação e alterá-los caso os resultados obtidos não sejam adequados. De preferência, deve-se manter valores iguais para os valores de passo máximo e inicial. Esta regra prática funciona de forma conveniente para a maioria dos problemas de simulação embora não seja a única nem a mais adequada para todos os casos. Em muitas situações é possível melhorar os resultados de uma simulação ajustando-se o fator de refinamento da simulação, como será discutido adiante.
- Error tolerances** – Os algoritmos de resolução usam técnicas de controle de erro a cada passo de simulação. Os valores estimados dos erros são comparados com um erro aceitável, definido pelos valores de tolerância relativa e absoluta, indicados na caixa de diálogo. Os algoritmos de passo variável reduzem o passo de integração automaticamente se o erro for maior que o aceitável. Em geral não é preciso alterar estes parâmetros.
- Output options** – Permite o controle dos instantes de tempo em que serão gerados os resultados da simulação. A opção mais útil é a do controle do fator de refinamento (*Refine factor*) que permite obter um número adicional de pontos de simulação entre aqueles que o algoritmo usaria normalmente. Por exemplo, se o fator de refinamento for definido como 5, cada passo de integração (de tamanho variável) será dividido em 5

subintervalos. Na prática, é mais simples e eficiente (do ponto de vista computacional) melhorar os resultados de uma simulação aumentando o fator de refinamento do que reduzindo o tamanho do passo de integração⁹.

8.1.2 Simulando

Ajuste o tempo de simulação para 80 segundos, alterando o valor de *Stop Time* (80). Simule a resposta do modelo, clicando em *Simulation > Start* ou no ícone em forma de seta na barra de ferramentas do Simulink. O programa avisa que a simulação terminou emitindo um *beep* e exibindo a palavra "*Ready*" na parte inferior da janela do modelo. Dê um duplo clique no bloco do osciloscópio (*Scope*) para ver a simulação do sinal de saída. O resultado deve ser como mostrado na figura 8.10.

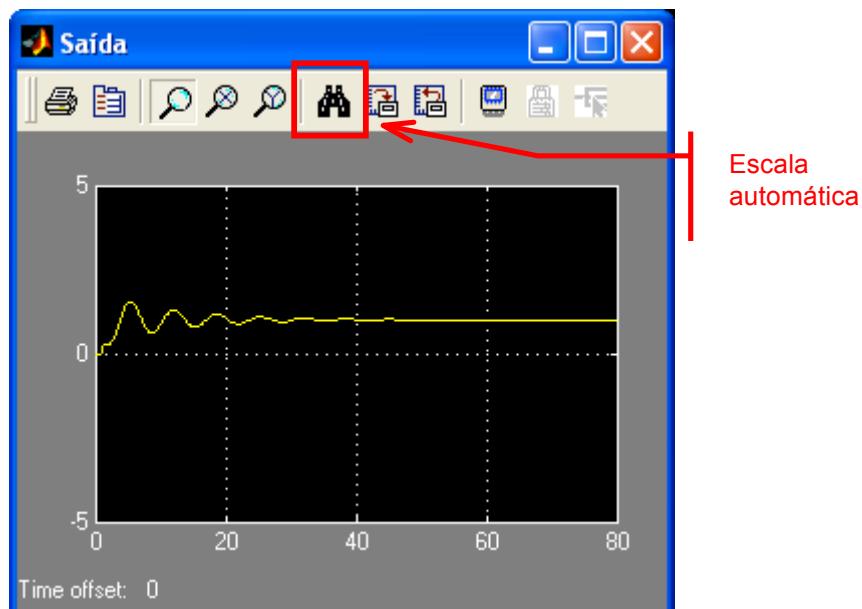


Figura 8.10: Resultado da simulação

É possível alterar as escalas dos eixos a partir das opções de configuração do bloco (clicando com o botão direito do *mouse* em algum ponto do gráfico) mas, geralmente, basta clicar no botão de escala automática (ver figura 8.10), indicado por um binóculo. O resultado final (ajuste de escala automática) é mostrado na figura 8.11. Se o resultado parecer pouco preciso, aumente o fator de refinamento (3 ou 5 costumam ser valores adequados) e simule novamente. Como padrão, o Simulink armazena o vetor de tempo usado na simulação em variável do *workspace* chamada **tout**. A criação desta variável, incluindo seu nome, podem ser ajustados na aba **Workspace I/O** da caixa de diálogo dos parâmetros de simulação.

⁹ Por questões de estabilidade numérica, o fator de refinamento não deve ser aumentado indefinidamente.

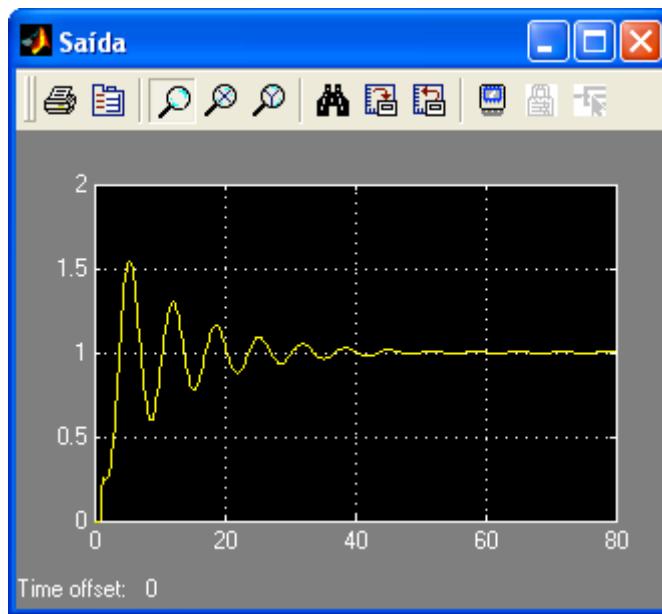


Figura 8.11: Resultado da simulação apóis ajuste de escala

8.2 Atividades

Atenção: Nas simulações pedidas a seguir use fator de refinamento = 5 e ajuste o tempo de simulação para valores convenientes.

- a) Simule o comportamento do sistema da figura 8.12 para uma entrada senoidal de 5 rad/s.

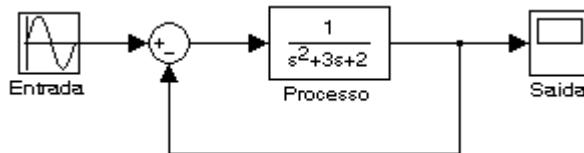


Figura 8.12: Sistema da atividade 8.2 a)

Determine o valor aproximado do ganho do sistema, analisando sua saída em regime permanente. Em seguida, calcule o valor exato do ganho fazendo $s = j5$ na função de transferência de *malha fechada* e compare os resultados.

- b) Determine o valor aproximado do módulo do ganho do sistema da figura 8.13 para sinais senoidais de 3 rad/s. Repita para sinais de 5 e 10 rad/s. O MATLAB pode emitir alguns avisos de "loop algébrico" porque a função de transferência do sistema realimentado possui numerador e denominador com mesmo grau.

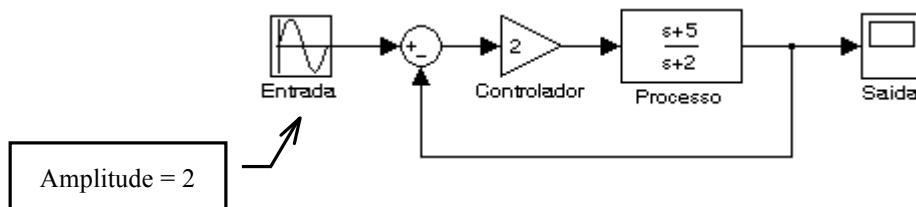


Figura 8.13: Sistema da atividade 8.2 b)

- c) Obtenha o valor aproximado do ganho do sistema da figura 8.14 para um sinal de entrada senoidal de 5 rad/s.

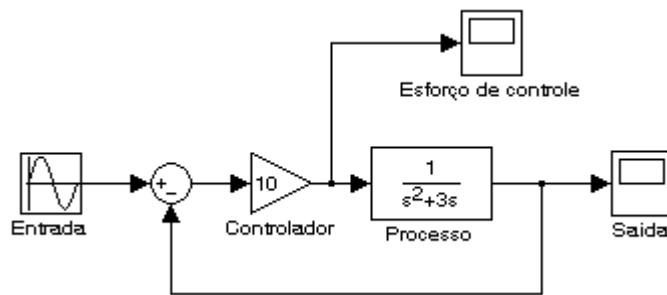


Figura 8.14: Sistema da atividade 8.2 c)

Em seguida, considere que a saída do controlador sature em **±8,5**. Simule o sistema nesta nova condição e verifique as alterações ocorridas em sua saída. Represente a saturação do controlador usando o bloco *Saturation* da biblioteca **Discontinuities** (ver figura 8.15).

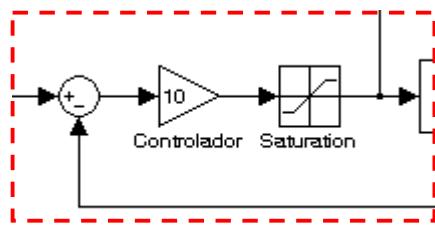


Figura 8.15: Ponto de inserção do bloco de saturação

- d) Investigue a atuação dos demais blocos da biblioteca **Discontinuities**.
- e) Simule o comportamento do sistema mostrado na figura 8.16, analisando a saída do sistema em relação ao sinal de entrada, após o misturador. Descreva a atuação do sistema. Sugestão para o tempo total de simulação: 5 s.

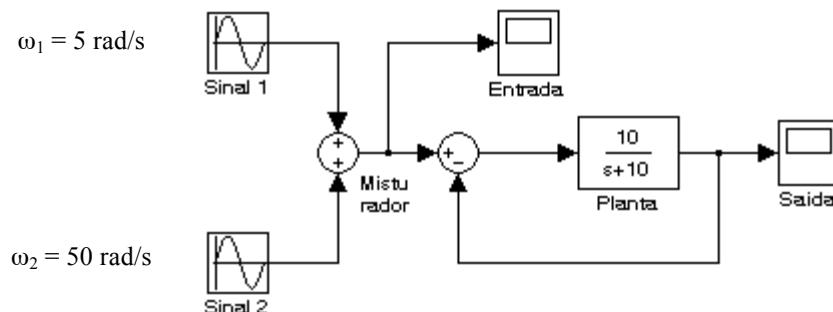


Figura 8.16: Sistema da atividade 8.2 e)

- f) Repita a simulação anterior, substituindo a planta original por:

$$i) \quad G_1(s) = \frac{100}{s^2 + 14s + 100}$$

$$ii) \quad G_2(s) = \frac{1.000}{s^3 + 20s^2 + 200s + 1.000}$$

Verifique as mudanças ocorridas no desempenho do sistema de acordo com a ordem do sistema que representa a planta. Descreva a atuação do sistema.

- g) É possível enviar os resultados de uma simulação para o *workspace* com o bloco **To Workspace** da biblioteca **Sinks**: o nome da variável criada no *workspace* é definido pelas propriedades do bloco. Simule o comportamento do sistema da figura 8.17 durante 10 segundos, ajustando o gerador de pulsos para um

período de 2 segundos. Na janela de comandos do MATLAB, trace o gráfico do vetor **saida** em função de **tout**.

Importante: configure o bloco **To Workspace** para gerar valores de saída no formato *array* (o formato padrão é *Structure*).

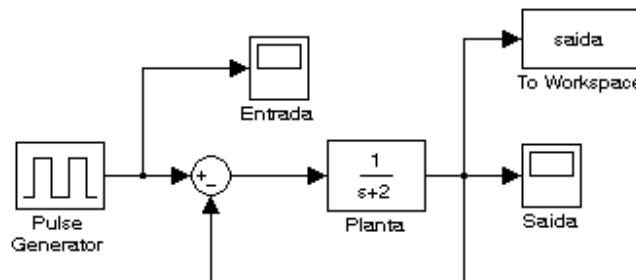


Figura 8.17: Sistema da atividade 8.2 g)

h) É possível usar variáveis do *workspace* como entradas para sistemas do Simulink, usando o bloco **From Workspace** da biblioteca **Sources**. Como exemplo, simule por 10 segundos o comportamento do sistema da figura 8.18 para a entrada $u(t) = 0,75 \sin(5t)$. Para isso, crie uma matriz de estímulos, **VU**, com os tempos de simulação na primeira coluna e os valores da função $u(t)$ na segunda.

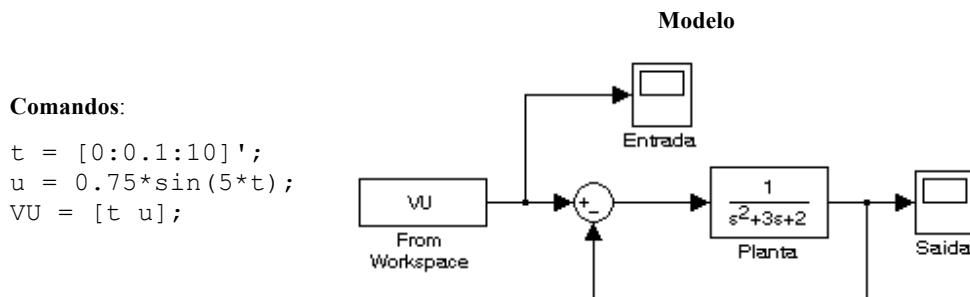


Figura 8.18: Sistema da atividade 8.2 h)

Observação: o bloco **From Workspace** deve estar associado à variável correta (**VU**).

8.3 Subsistemas

Modelos complexos podem ser simplificados pela criação de subsistemas, que são conjuntos de blocos relacionados logicamente. A maneira mais simples de se criar um subsistema é pelo agrupamento de blocos já existentes em um modelo. Também é possível adicionar um bloco **subsystem** vazio a um modelo e editá-lo.

8.3.1 Exemplo – controlador PI

Crie um modelo no Simulink, de acordo com a figura 8.19.

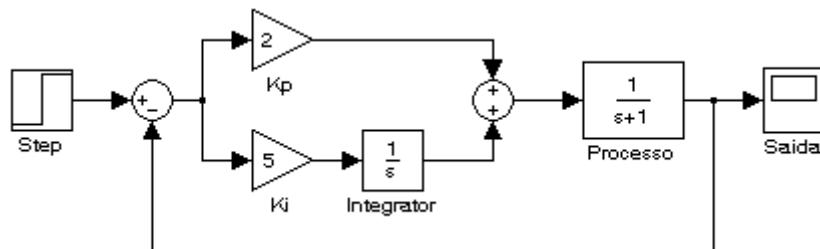


Figura 8.19: Sistema de controle com controlador PI

Usando o *mouse*, selecione os blocos do controlador PI (ver figura 8.20).

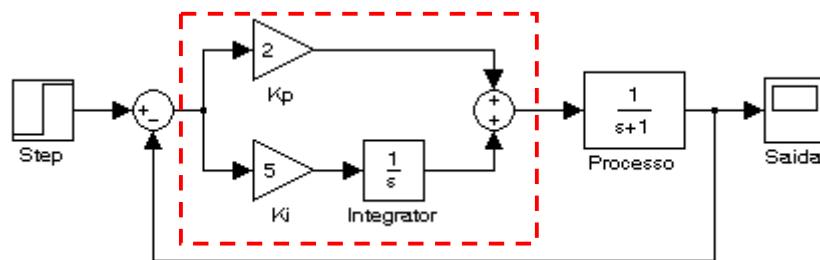


Figura 8.20: Seleção dos blocos do controlador PI

Clique em *Edit > Create Subsystem*. Os blocos selecionados serão substituídos por um bloco chamado *Subsystem*, com uma entrada (**In1**) e uma saída (**Out1**). Altere o nome do bloco para "Controlador PI", conforme a figura 8.21.

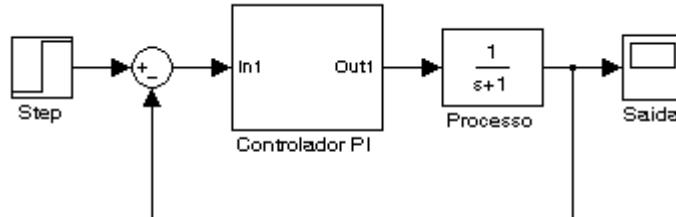


Figura 8.21: Modelo de simulação com subsistema

Dê um *duplo-clique* no subsistema para editá-lo, modificar os nomes das entradas e saídas ou mesmo para gravá-lo como um modelo independente do Simulink (ver figura 8.22).

A entrada e a saída do subsistema são representadas por blocos da biblioteca **Ports & Subsystems** e podem ter seus nomes ocultados individualmente pela opção **Format - Hide Name**, do menu de contexto acionado pelo botão direito do *mouse*. Se esta operação for feita no subsistema, fora de sua janela de edição, todos os nomes serão escondidos e o subsistema será representado por um ícone padrão.

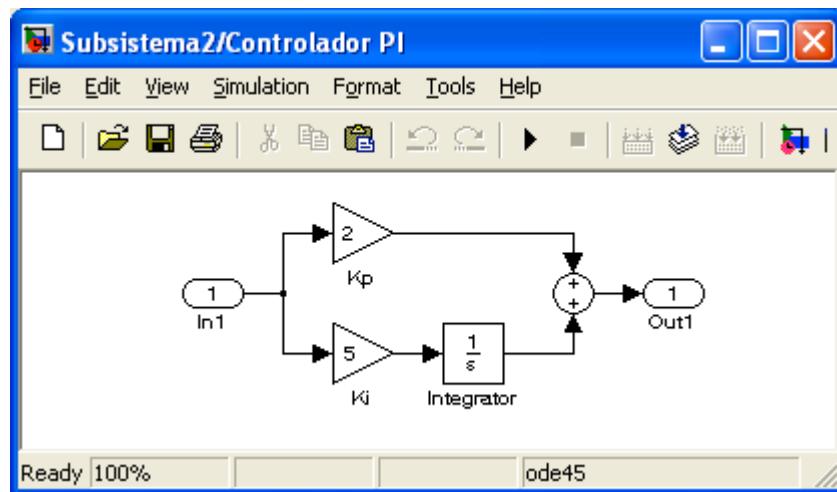


Figura 8.22: Janela de edição do subsistema

Existem ainda ferramentas avançadas, como as *máscaras*, que permitem a personalização da aparência dos subsistemas e a criação de caixas de diálogo (ver exemplo na figura 8.23).

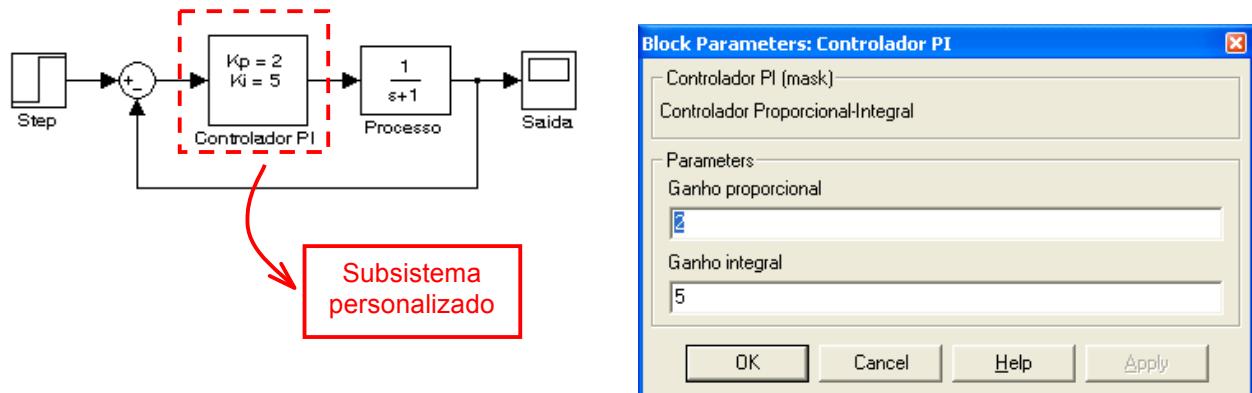


Figura 8.23: Subsistema personalizado e caixa de diálogo de propriedades

Se desejar, consulte a ajuda do MATLAB para aprender a trabalhar com estes recursos.

Índice

1. Introdução	1
1.1 Iniciando o MATLAB	1
1.2 Manipulação de matrizes	2
1.2.1 Atividades	4
1.3 Seqüências	5
1.4 Obtendo ajuda	6
1.5 Operações matemáticas	6
1.5.1 Sistemas de equações lineares	7
1.5.2 Atividade	8
1.6 Outras operações com matrizes	8
1.7 Gráficos.....	9
1.7.1 Gráficos tridimensionais	11
1.8 Polinômios	13
1.8.1 Avaliação, multiplicação, divisão e diferenciação..	14
1.9 Funções de transferência.....	14
1.10 Simulações	15
2. Análise de sistemas lineares de 1^a e 2^a ordem – atividades	18
2.1 Sistemas de 1 ^a ordem	18
2.2 Resposta temporal.....	18
2.3 Sistemas de 2 ^a ordem	18
2.4 Sistemas de 2 ^a ordem sem zeros	19
2.5 Validade do Teorema do Valor Final	19
2.6 Aproximações	19
3. Aproximações e estabilidade de sistemas lineares	20
3.1 Sistemas de 3 ^a ordem	20
3.1.1 Exemplo	20
3.2 Aproximações para modelos de 2 ^a ordem.....	20
3.2.1 Exemplos.....	21
3.3 Conexões entre sistemas	21
3.4 Estabilidade.....	22
3.4.1 Exemplos.....	22
4. Programação	24
4.1 Exemplo – análise de um sistema linear	24
4.1.1 Criando um <i>script</i>	24
4.1.2 Análise do <i>script</i>	25
4.1.3 Criando uma função	26
4.1.4 Atividade	26
4.1.5 Convenções	26
4.2 Controle de fluxo	27
4.2.1 Estrutura condicional <i>if</i>	28
Exemplo: 28	
4.2.2 Estrutura repetitiva <i>while</i>	28

4.2.3	Estrutura repetitiva <i>for</i>	29
4.3	Vetorização	29
4.4	Entrada de dados	29
4.5	Edição de funções existentes	30
4.6	Subfunções	30
4.7	Exemplos de aplicação	31
4.7.1	Aproximações	31
4.7.2	Análise do erro em regime estacionário	32
4.7.3	Atividade	33
4.7.4	Atividade – estabilidade em função de um parâmetro	33
5.	Lugar das raízes – introdução	34
5.1	Exemplo	34
5.1.1	Atividade	35
5.2	Análise gráfica do lugar das raízes	35
5.2.1	Atividade	36
5.3	Exemplos	36
6.	Projeto usando lugar das raízes – introdução.....	39
6.1	Exemplo de projeto	39
6.1.1	Atividades	42
7.	Resposta em freqüência – introdução.....	44
7.1	Determinação manual da resposta em freqüência	44
7.1.1	Atividades	44
7.2	Diagrama de Bode	45
7.2.1	Exemplo	45
7.3	Sistemas de 2 ^a ordem	46
7.3.1	Atividades	47
8.	Simulink.....	49
8.1	Exemplo de uso	49
8.1.1	Configurando os parâmetros de simulação	52
8.1.2	Simulando	54
8.2	Atividades	55
8.3	Subsistemas	57
8.3.1	Exemplo – controlador PI	57

Foto da capa: "Motor driven XY table - Series MAXY6000 ". Obtida de: <http://www.unislide.com/images/motor/maxy6012elite.jpg>

1. Introdução

O MATLAB (*Matrix Laboratory*) é um ambiente de programação de alto desempenho voltado para a resolução de problemas que possam ser expressos em notação matemática. Projeto e simulação de sistemas de controle, análise de dados e criação de gráficos são algumas das aplicações possíveis para esta ferramenta. Pacotes específicos, chamados *toolboxes*, permitem a expansão do ambiente de trabalho do MATLAB para a resolução de classes particulares de problemas como processamento de sinais, identificação de sistemas, implementação de redes neurais, lógica difusa (*fuzzy*), simulação, etc. Adicionalmente, um programa gráfico chamado *Simulink*, que trabalha juntamente com o MATLAB, permite a simulação interativa de sistemas dinâmicos lineares ou não-lineares, contínuos ou digitais.

1.1 Iniciando o MATLAB

Execute o MATLAB 6.5 a partir do menu "Iniciar". A tela principal do programa (figura 1.1) contém, em sua visualização padrão, uma janela de comandos (*command window*), uma janela para exibição da área de trabalho (*workspace*), onde ficam armazenadas as variáveis definidas pelo usuário e o histórico de comandos. A janela de comando fornece a principal forma de comunicação entre o usuário e o interpretador MATLAB, que exibe um sinal de prontidão (*prompt*) para indicar que está pronto para receber instruções.

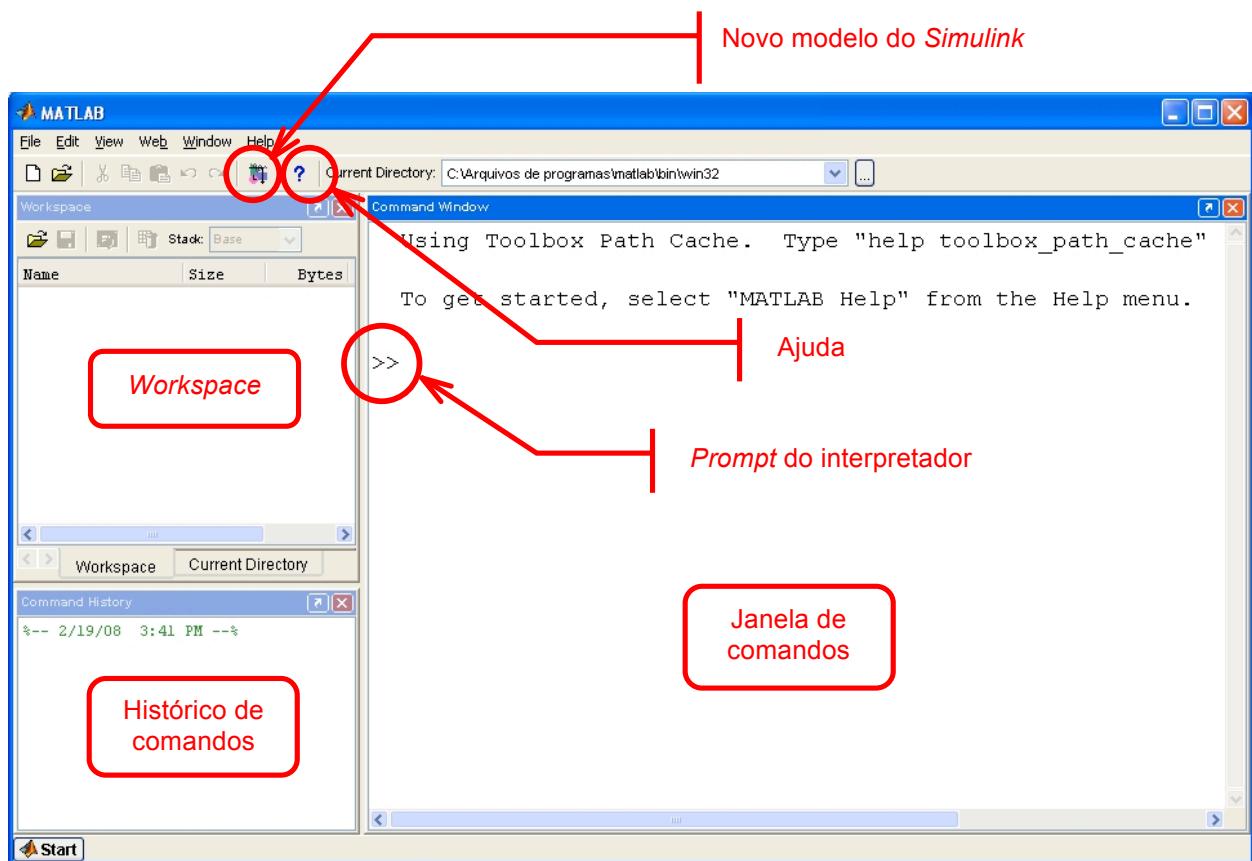


Figura 1.1: Tela principal do MATLAB

A visualização padrão da janela de comando (conforme a figura 1.1) pode ser obtida, a qualquer momento, clicando-se em *View > Desktop Layout > Default*.

Antes de iniciar a sessão de trabalho é conveniente aumentar a fonte da letra usada na janela de comando. Clique em *File > Preferences > Command Window > Fonts & Colors*, selecione a opção "Use custom font" (ver figura 1.2) e ajuste o tamanho da fonte para (no mínimo) 16 pontos. Acredite: muitos erros de digitação podem ser evitados com esta simples providência!

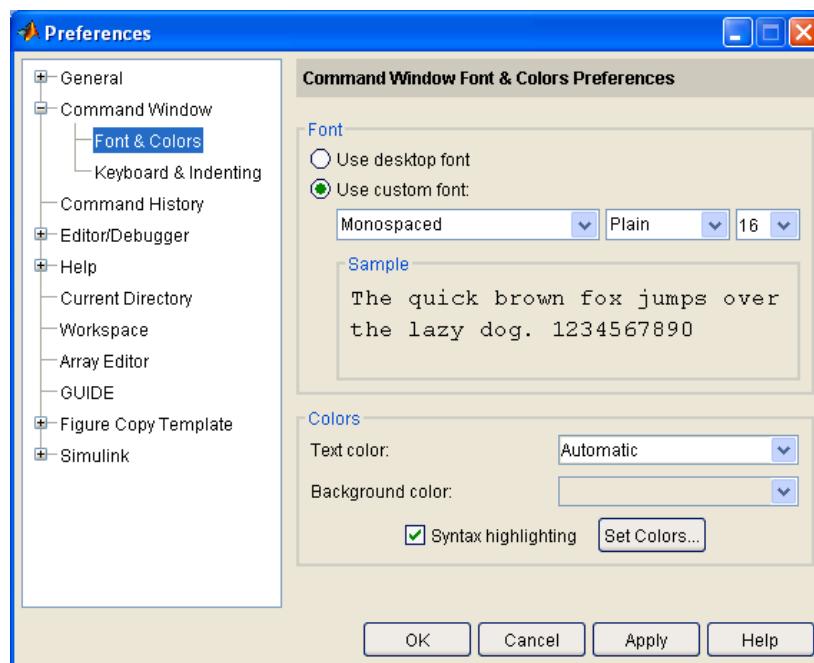


Figura 1.2: Ajuste da fonte usada na janela de comandos

1.2 Manipulação de matrizes

O tipo numérico padrão usado pelo MATLAB é a **matriz** de valores em ponto flutuante: números reais ou complexos são armazenados em matrizes 1x1. A maneira mais simples de se armazenar uma matriz na memória é com uma atribuição, como em:

```
>> A = [2 1 3 4 5] ↵
```

O resultado do comando anterior é mostrado na figura 1.3. Note que o comando passou a fazer parte do histórico do programa e que a matriz foi armazenada no *workspace*. A alocação da matriz também pode ser confirmada pelos comandos **who** (que mostra os nomes das variáveis armazenadas) ou **whos** (que mostra os nomes e espaços ocupados pelas variáveis). Exemplo:

```
>> whos ↵
Name      Size      Bytes  Class
A         1x5        40    double array
Grand total is 5 elements using 40 bytes
```

O MATLAB é **sensível à caixa**, ou seja, diferencia letras maiúsculas de minúsculas e aloca automaticamente o espaço de memória necessário para as variáveis usadas.

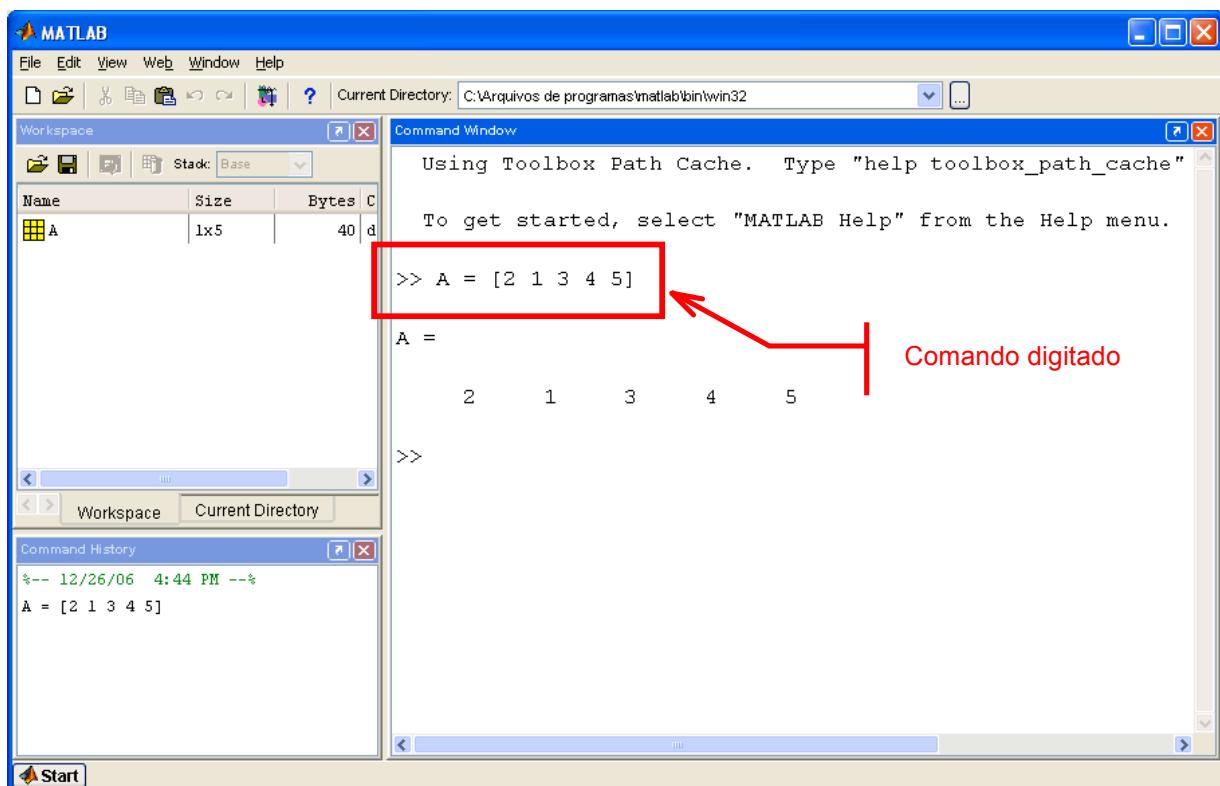


Figura 1.3: Atribuição de valores

A matriz deste exemplo é chamada de **vetor linha**, já que se trata de uma matriz com apenas 1 linha: na digitação, os valores do vetor podem ser separados por espaços, como no exemplo, ou por vírgulas. Para criar um **vetor coluna** deve-se separar cada linha das demais usando **ponto-e-vírgula**. Exemplo:

```
>> B = [5; -4; 6.5] ↵
B =
    5.0000
   -4.0000
    6.5000
```

Para criar uma matriz **bidimensional**¹ basta combinar as sintaxes anteriores:

```
>> M = [2 1 3; 4 6 7; 3 4 5] ↵
M =
    2     1     3
    4     6     7
    3     4     5
```

Quando for interessante omitir a exibição do resultado de qualquer comando basta encerrá-lo com ponto-e-vírgula. Exemplo:

```
M = [2 1 3; 4 6 7; 3 4 5]; % Cria uma matriz 3x3 (3 linhas e 3 colunas)
```

¹ A partir deste ponto, o termo matriz será usado para designar matrizes com mais de uma dimensão.

O símbolo de porcentagem serve para criar comentários de uma linha, tanto na janela de comandos quanto no ambiente de programação do MATLAB.

Os elementos de uma matriz podem ser acessados pelo nome da variável, seguido de índices entre parênteses, sendo que o primeiro elemento é **sempre o de índice 1**. Exemplo de acesso:

```
>> x = B(2) ↴
x =
-4
```

Se uma nova informação for atribuída a um vetor ou matriz os redimensionamentos necessários serão feitos automaticamente. Exemplo:

```
>> A = [4 5 9]; ↴
>> A(6) = 8
A =
4      5      9      0      0      8
```

A red bracket points from the text "Valores inseridos automaticamente" to the value 8 at index 6 of the matrix A.

Para acessar os elementos de uma matriz escreve-se o conjunto de índices entre parênteses, separados por vírgula. Exemplo:

```
>> x = M(2, 3) ↴
x =
7
```

O resultado de qualquer comando que não seja atribuído a uma variável específica é armazenado em uma variável especial chamada **ans**. Exemplo:

```
>> M(2, 1) ↴
ans =
4
```

Para facilitar a repetição de comandos é possível usar as setas para cima e para baixo do teclado ou dar um *duplo-clique* nos itens da janela de histórico. Não existem comandos específicos para desfazer atribuições feitas na janela de comando, apesar de existir a opção *undo* no menu **Edit** do programa.

1.2.1 Atividades

Antes de iniciar as atividades a seguir, limpe a janela de comando digitando **clc**. Em seguida, remova todas as variáveis da memória, usando o comando **clear**. Se quiser eliminar apenas uma variável, use a sintaxe **clear <nome da variável>**. Anote os resultados obtidos.

- a) Armazene a seguinte matriz:

$$M = \begin{bmatrix} 2 & -1 & 5 \\ 3 & 1 & 1 \\ 2 & 0 & 2 \end{bmatrix}$$

b) Obtenha a matriz **transposta** de **M**, digitando:

```
m1 = M'
```

c) Obtenha a matriz **inversa** de **M**, digitando:

```
m2 = inv(M)
```

A utilidade da inversa de uma matriz será discutida futuramente.

d) A indexação pode ser usada em conjunto com o sinal ":" para indicar "todos os elementos" de uma certa dimensão. Por exemplo, o comando a seguir cria um vetor linha com todos os elementos da segunda linha da matriz **M**:

```
v1 = M(2, :)
```

O comando anterior pode ser traduzido como "armazene em **v1** os elementos de **M** que estão na linha 2 e em todas as colunas". Da mesma forma, o comando a seguir cria um vetor coluna com os elementos da primeira coluna da matriz **M**:

```
v2 = M(:, 1)
```

e) O uso de ":" também permite a atribuição de valores a uma dimensão completa de uma matriz. Por exemplo, verifique o efeito da seguinte instrução sobre a matriz **M**:

```
M(1, :) = 5
```

f) Se a atribuição envolver uma **matriz vazia**, indicada por um par de colchetes vazios, é possível eliminar totalmente uma linha ou coluna de uma matriz. Por exemplo, a instrução

```
M(2, :) = []
```

remove a segunda linha da matriz **M**.

g) Finalmente, matrizes podem ser concatenadas por meio de atribuições diretas. Exemplo:

```
m3 = [[5; 5; 5] v2 v1'] % Cria uma nova matriz 3 x 3
```

1.3 Seqüências

O uso de ":" também serve para denotar uma seqüência igualmente espaçada de valores, entre dois limites especificados, inteiros ou não.

Por exemplo, a instrução

```
v3 = 3:8
```

cria um vetor linha com os valores 3, 4, 5, 6, 7 e 8 (o incremento padrão é unitário).

O incremento pode ser definido pelo usuário se a seqüência for criada sob a forma:

[Valor inicial: Incremento: Valor final]

Exemplo:

```
>> v4 = 2:0.5:4 ↵
v4 =
2.0000    2.5000    3.0000    3.5000    4.0000
```

Outra forma de se obter um vetor com valores igualmente espaçados é pelo uso da função ***linspace***. Por exemplo, a instrução

```
y = linspace(10,200,25)
```

gera um vetor linha com 25 valores igualmente espaçados entre 10 e 200. Se o parâmetro que controla o número de pontos for omitido, a seqüência terá 100 valores.

A diferença entre usar o operador ":" e a função ***linspace*** é que a primeira forma exige o **espaçamento** entre os valores enquanto a segunda requer a **quantidade** de valores.

Seqüências com valores linearmente espaçados são usados, normalmente, para fornecer valores de variáveis independentes para funções. Por exemplo, as instruções a seguir criam um vetor com 50 valores da função $y = \sin(x)$, para $x \in [0, 2\pi]$:

```
x = linspace(0,2*pi,50); % 'pi' é uma função interna que retorna o valor de π
y = sin(x);
```

Neste tipo de operação, chamada de *vetorizada*, o MATLAB cria ou redimensiona o vetor **y** com a mesma dimensão do vetor **x**. Em situações que exijam grandes variações de valores, como na análise de respostas em freqüência, é interessante que a variação de valores seja *logarítmica*, o que pode ser obtido com o uso da função ***logspace***, de sintaxe semelhante à de ***linspace***. Por exemplo, a instrução

```
f = logspace(0,4,50);
```

cria um vetor com 50 valores espaçados logaritmicamente entre $10^0 = 1$ e $10^4 = 1.000$.

1.4 Obtendo ajuda

Há diversas maneiras de se obter mais informações sobre uma função ou tópico do MATLAB. Se o nome da função for conhecido pode-se digitar, na janela de comando:

```
help <nome da função> ↵
```

Também é possível fazer uma busca por palavra-chave com o comando ***lookfor***. Por exemplo,

```
lookfor identity ↵
```

retorna uma descrição curta de funções relativas a matrizes identidade. Além dessas formas, pode-se consultar a documentação do MATLAB clicando no ícone de ajuda da janela de comandos.

1.5 Operações matemáticas

O MATLAB reconhece os operadores matemáticos comuns à maioria das linguagens de programação nas operações com escalares (números reais e complexos). Nas operações com matrizes é preciso respeitar as regras da Matemática em relação às dimensões envolvidas. Por exemplo, considere as seguintes matrizes:

$$A = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 3 & 9 \\ 6 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 5 & 5 \\ 4 & 6 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

Em seguida, considere as operações:

- i) $C = A+B$ (Soma)
- ii) $C = A-B$ (Subtração)
- iii) $C = A*B$ (Multiplicação matricial)
- iv) $C = A.*B$ (Multiplicação elemento-a-elemento)
- v) $C = A./B$ (Divisão elemento-a-elemento)
- vi) $C = A.^2$ (Potenciação elemento-a-elemento)

De acordo com as definições da Matemática e as convenções do MATLAB, deve-se obter os seguintes resultados:

$$\begin{array}{lll} i) \quad C = \begin{bmatrix} 6 & 9 & 10 \\ 6 & 9 & 11 \\ 9 & 4 & 6 \end{bmatrix} & ii) \quad C = \begin{bmatrix} -4 & -1 & 0 \\ -2 & -3 & 7 \\ 3 & -4 & -4 \end{bmatrix} & iii) \quad C = \begin{bmatrix} 36 & 49 & 38 \\ 49 & 64 & 61 \\ 33 & 34 & 35 \end{bmatrix} \\ iv) \quad C = \begin{bmatrix} 5 & 20 & 25 \\ 8 & 18 & 18 \\ 18 & 0 & 5 \end{bmatrix} & v) \quad C = \begin{bmatrix} 0,2 & 0,8 & 1,0 \\ 0,5 & 0,5 & 4,5 \\ 2,0 & 0 & 0,2 \end{bmatrix} & vi) \quad C = \begin{bmatrix} 1 & 16 & 25 \\ 4 & 9 & 81 \\ 36 & 0 & 1 \end{bmatrix} \end{array}$$

Atenção:

- 1) Para obter o sinal de potenciação (^) é preciso pressionar a tecla correspondente duas vezes;
- 2) A multiplicação de uma matriz A ($n \times k$) por uma matriz B ($k \times m$) produz uma matriz $n \times m$. Para as outras operações mostradas, as matrizes A e B devem ter as mesmas dimensões.

1.5.1 Sistemas de equações lineares

Todo sistema de equações lineares pode ser escrito sob a forma matricial $Ax = b$. Exemplo:

$$S = \begin{cases} 3x_1 - 2x_2 + x_3 = -4 \\ 2x_2 - x_3 = 7 \\ 4x_1 + x_2 + 2x_3 = 0 \end{cases} \Rightarrow A = \begin{bmatrix} 3 & -2 & 1 \\ 0 & 2 & -1 \\ 4 & 1 & 2 \end{bmatrix}; \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}; \quad b = \begin{bmatrix} -4 \\ 7 \\ 0 \end{bmatrix}$$

Se a matriz dos coeficientes (A) for quadrada e não-singular, ou seja, sem linhas ou colunas linearmente dependentes, a solução (única) do sistema é dada por:

$$\bar{x} = A^{-1}b.$$

Esta solução pode ser calculada de forma direta pelo MATLAB pelas instruções:

$$x = \text{inv}(A)*b \quad \text{ou} \quad x = A\b$$

As duas formas fornecem as mesmas respostas, mas os cálculos envolvidos no uso do operador "\\" exigem menos memória e são mais rápidos do que os envolvidos no cálculo de uma matriz inversa. O MATLAB também resolve sistemas sob a forma $xA = b$ ou sistemas com mais de uma solução (o que não será discutido neste material, consulte a ajuda do programa).

1.5.2 Atividade

Resolva, se possível, os seguintes sistemas lineares.

$$S_1 = \begin{cases} 3x_1 - 2x_2 + x_3 = -4 \\ 2x_2 - x_3 = 7 \\ 4x_1 + x_2 + 2x_3 = 0 \end{cases} \quad S_2 = \begin{cases} x_1 + 4x_2 + 7x_3 = 5 \\ -3x_1 - 9x_3 = 1 \\ 2x_1 + 5x_2 + 11x_3 = -2 \end{cases} \quad S_3 = \begin{cases} x_1 + 2x_2 = 4 \\ 3x_1 + 6x_2 = 5 \end{cases}$$

1.6 Outras operações com matrizes

Há uma série de funções disponíveis no MATLAB para geração ou alteração de matrizes e vetores, exemplificadas a seguir.

a) `x = max(A)`

Retorna o maior componente de **A**. Se **A** for uma matriz, o resultado é um vetor linha contendo o maior elemento de cada coluna. Para vetores, o resultado é o maior valor (ou o número complexo com maior módulo) entre seus componentes. Ainda para vetores, a sintaxe

```
[vmax imax] = max(v)
```

retorna o maior elemento do vetor **v** em **vmax** e o índice correspondente em **imax**.

b) `x = size(A)`

Retorna as dimensões da matriz **A** em um vetor linha, **x = [m n]**, contendo o número de linhas (**m**) e colunas (**n**) da matriz. A sintaxe

```
[m n] = size(A)
```

determina o número de linhas e colunas em variáveis separadas.

c) `x = length(A)`

Retorna o comprimento do vetor **A** ou o comprimento da maior dimensão da matriz **A**. Neste último caso, **length(A) = max(size(A))**.

d) `x = zeros(n)`

Cria uma matriz quadrada $n \times n$ de elementos nulos. Também é possível obter matrizes retangulares $m \times n$ usando a sintaxe **x = zeros(m,n)**. A sintaxe

```
x = zeros(size(A))
```

produz uma matriz **x** com as mesmas dimensões de **A**, preenchida com zeros.

e) `x = ones(n)`

Semelhante a **zeros**, gerando matrizes com valores unitários (preenchidas com 1's).

f) `x = eye(n)`

Retorna uma matriz identidade $n \times n$, isto é, com valores unitários na diagonal principal e nulos nas demais posições.

g) `x = det(A)`

Retorna o determinante da matriz quadrada **A**. Nota: para verificar se uma matriz possui linhas ou colunas linearmente dependentes o manual do MATLAB recomenda usar a função **cond** (cálculo do número de condição) ao invés de verificar se **det(A) = 0**.

h) `x = find(expressão)`

Encontra e retorna todos os elementos de um vetor ou matriz que satisfazem a uma certa expressão lógica. Normalmente, usa-se argumentos à esquerda da instrução de busca para armazenar os índices dos elementos de interesse. Exemplo:

```
>> A = [3 -2 1; 0 2 -1; 4 1 2]; ↴
>> [L C] = find(A>2 & A<=4) ↴
L =
    1
    3
C =
    1
    1
```

$$Obs.: \quad A = \begin{bmatrix} 3 & -2 & 1 \\ 0 & 2 & -1 \\ 4 & 1 & 2 \end{bmatrix}$$

No exemplo anterior, apenas $A(1,1)$ e $A(3,1)$ atendem ao critério desejado. As expressões válidas em MATLAB podem incluir operadores relacionais e lógicos, resumidos na tabela 1.1.

Tabela 1.1: Operadores relacionais e lógicos

Operadores relacionais		Operadores lógicos	
<	Menor que	&	Operação "E"
<=	Menor ou igual		Operação "OU"
>	Maior que	~	Negação lógica
>=	Maior ou igual		
==	Igual a		
~=	Diferente de		

Estes operadores se aplicam a escalares e matrizes, de acordo com regras que podem ser consultadas na documentação do MATLAB. Se o argumento da função for apenas o nome de uma matriz ou vetor, serão retornados os índices de todos os elementos da matriz ou vetor que forem diferentes de zero.

i) $x = \text{all}(A)$

Retorna 1 para cada coluna da matriz **A** que contenha somente valores não nulos e 0 em caso contrário, gerando um vetor linha. Exemplo:

```
>> A = [3 -2 1; 0 2 -1; 4 1 2]; ↴
>> x = all(A) ↴
x =
    0      1      1
```

$$Obs.: \quad A = \begin{bmatrix} 3 & -2 & 1 \\ 0 & 2 & -1 \\ 4 & 1 & 2 \end{bmatrix}$$

Para vetores, a função retorna 1 se todos os elementos forem não nulos e 0 em caso contrário.

j) $x = \text{any}(A)$

Retorna 1 para cada coluna da matriz **A** que contenha algum valor não nulo e 0 em caso contrário, gerando um vetor linha. A função também trabalha com vetores. Exemplo:

```
>> x = any(A) ↴
x =
    1      1      1
```

1.7 Gráficos

O MATLAB possui diversas ferramentas para traçados de gráficos bidimensionais ou tridimensionais. A maneira mais simples de traçar um gráfico *xy* é pelo uso da função **plot**. A forma **plot(x,y)** desenha um

gráfico bidimensional dos pontos do vetor y em relação aos pontos do vetor x , sendo que ambos devem ter o mesmo número de elementos. Não é obrigatório que os valores de y representem uma função em relação aos valores de x . O gráfico resultante é desenhado em uma *janela de figura* com as escalas automáticas nos eixos x e y e segmentos de reta unindo os pontos. Por exemplo, para desenhar o gráfico da função

$$y = 1 - 1,1547e^{-1,5x} \sin(2,5981x + 1,0472),$$

no intervalo $x \in [0,10]$, pode-se utilizar a seguinte seqüência de comandos:

```
>> x = 0:0.1:10; ↴
>> y = 1-1.1547*exp(-1.5*x).*sin(2.5981*x+1.0472); ↴
>> plot(x,y) ↴
```

O resultado (ver figura 1.4) é exibido em uma janela de figura identificada por um número.

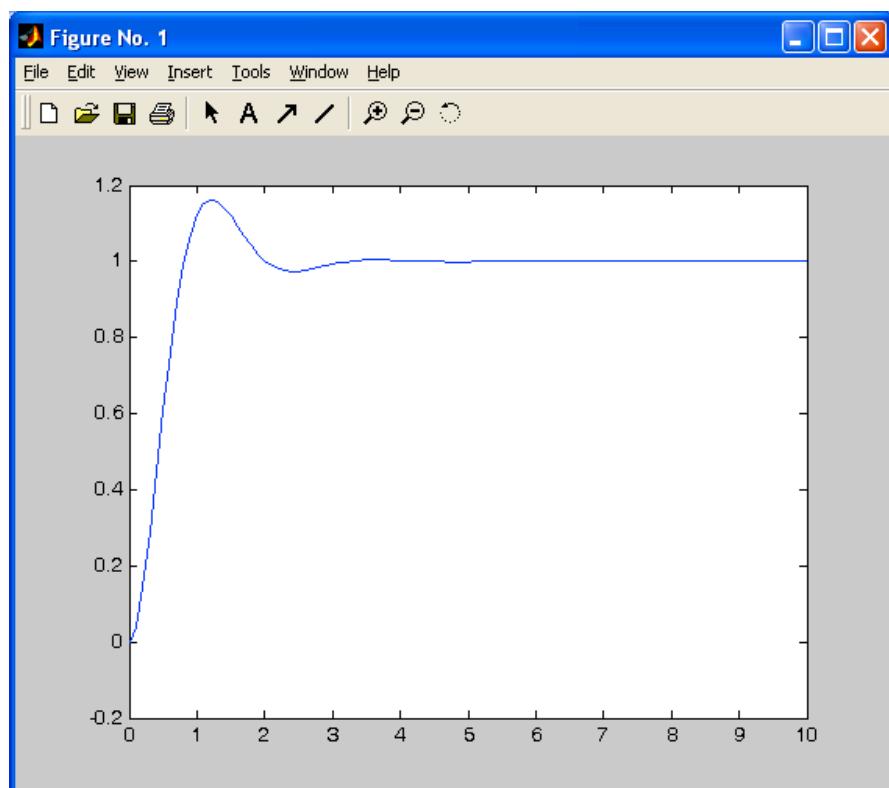


Figura 1.4: Exemplo de resultado gráfico da função *plot*

Em algumas ocasiões é interessante que as escalas dos eixos sejam representadas em escala logarítmica (ao invés da escala linear padrão). Nestes casos, é possível usar as funções *semilogx*, *semilogy* ou *loglog*, que alteram, respectivamente, a escala do eixo x , do eixo y e de ambos. Normalmente os valores que compõem tais gráficos também são gerados com espaçamentos logarítmicos, via função *logspace*.

A função *plot* pode trabalhar com várias duplas de vetores, sobrepondo mais de um gráfico em uma mesma janela. Exemplo:

```
x = linspace(0,2*pi,100); % Cria vetor 'x' com 100 pontos de 0 a 2*pi
y1 = sin(x); % Calcula y1 = sen(x)
y2 = 0.5*sin(3*x); % Calcula y2 = 0.5*sen(3x)
plot(x,y1,x,y2); % Traça os dois gráficos
xlabel('Ângulo em graus'); % Nomeia o eixo x
ylabel('sen(x) e sen(3x)'); % Nomeia o eixo y
title('Gráficos sobrepostos'); % Atribui um título ao gráfico
grid % Ativa as linhas de grade da janela
```

Note que foram usadas funções para nomear os eixos (*xlabel* e *ylabel*) e o título do gráfico (*title*), além de exibição de linhas de grade (*grid*). O resultado da seqüência de comandos anterior está representado na figura 1.5.

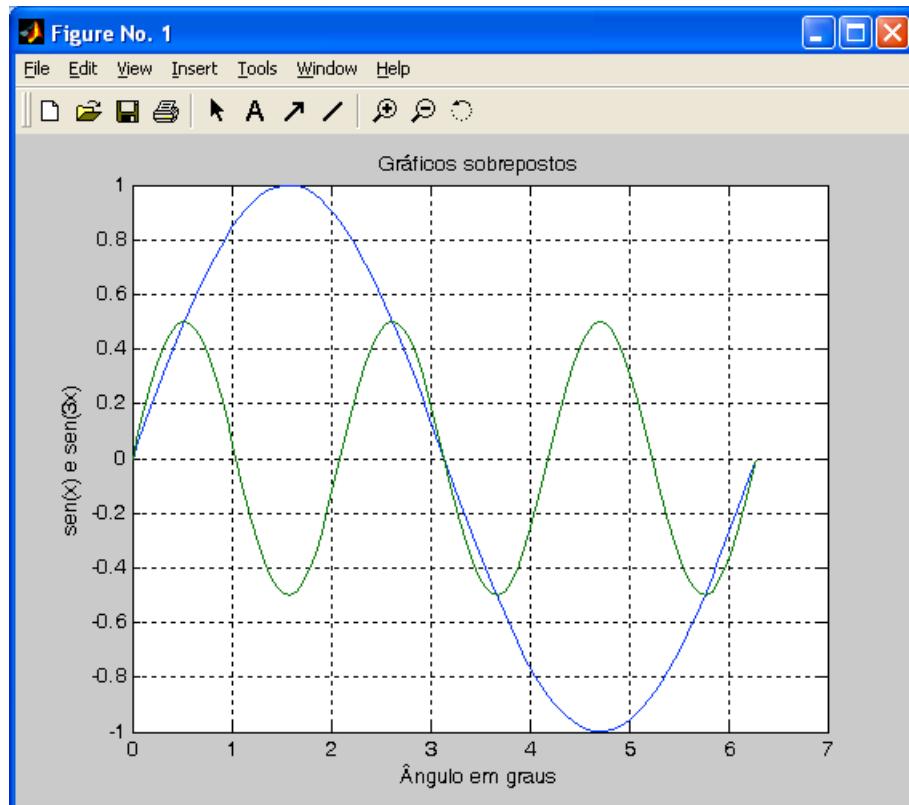


Figura 1.5: Gráfico de duas funções superpostas

Outra forma de se obter gráficos sobrepostos é com o uso da função *hold*, que faz com que todos os resultados gráficos subseqüentes ao seu uso sejam desenhados em uma mesma janela de figura. Exemplo (considerando as variáveis do exemplo anterior):

```
plot(x,y1); % Desenha o gráfico de uma função
hold on      % Ativa a 'trava' de exibição gráfica
plot(x,y2); % Desenha outro gráfico na mesma janela de figura
hold off     % Desativa a 'trava' de exibição gráfica
```

Todos os resultados gráficos aparecem na janela de figura *ativa*. Uma nova janela pode ser criada ou ativada pelo comando *figure*. Quando usada sem argumentos, esta função cria uma janela de título *Figure No. xx*, sendo *xx* um número seqüencial, considerado disponível pelo MATLAB. O uso de *figure(n)* cria a janela de figura *n*, se necessário, e a torna ativa. Outros recursos da função *plot* podem ser consultados na documentação do MATLAB.

1.7.1 Gráficos tridimensionais

Gráficos em três dimensões podem ser traçados pelo MATLAB com a mesma facilidade que os bidimensionais. A função *plot3* funciona de forma semelhante à *plot* para o traçado de *gráficos de linha*. Por exemplo, a seqüência de comandos a seguir produz um gráfico de uma *hélice tridimensional*. Note o uso da função *zlabel* para nomear o eixo *z* do gráfico.

```
t = linspace(0,6*pi,100); %
plot3(sin(t),cos(t),t); %
xlabel('seno(t)'); %
ylabel('cosseno(t)');
```

```

zlabel('z = t'); ↵
title('Gráfico de hélice'); ↵
grid on; ↵

```

O resultado está representado na figura 1.6.

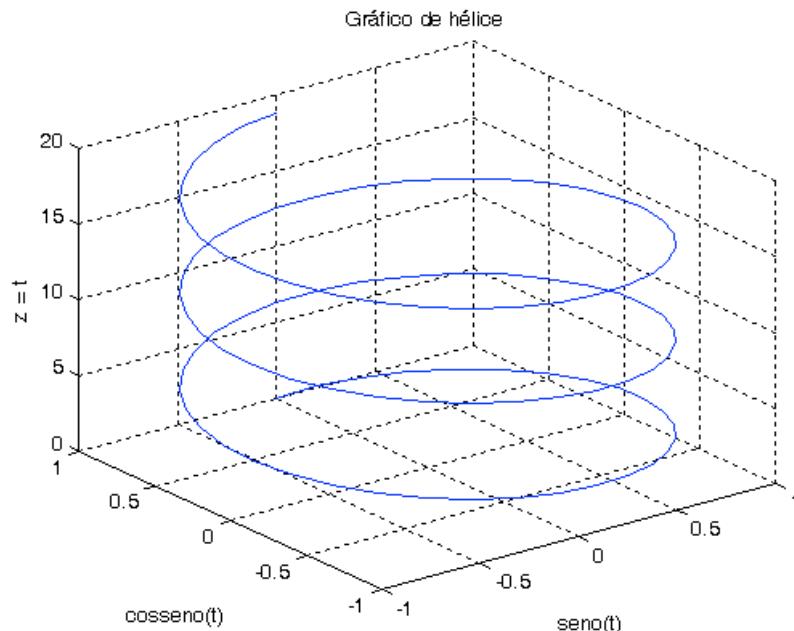


Figura 1.6: Gráfico de linha tridimensional

O MATLAB também pode construir *gráficos de superfícies*, a partir de um conjunto de coordenadas tridimensionais xyz . Inicialmente, é preciso gerar matrizes X e Y com, respectivamente, linhas e colunas repetidas, preenchidas com os valores das variáveis x e y . Isto pode ser feito diretamente pela função **meshgrid**, como no exemplo mostrado a seguir:

```

x = linspace(0,2,20);           % Geração de valores para 'x' e 'y',
y = linspace(1,5,20);           % ambos com a mesma dimensão!
[X,Y] = meshgrid(x,y);         % Criação da matriz da malha 'xy'

```

A partir dessas matrizes, que representam uma grade retangular de pontos no plano xy , qualquer função de duas variáveis pode ser calculada em uma matriz Z e desenhada pelo comando **mesh**. Exemplo para o gráfico de um *parabolóide elíptico*:

```

x = -5:0.5:5;                  % Definição da malha de pontos no eixo 'x'
y = x;                          % Repetição da malha do eixo x para o eixo 'y'
[X,Y] = meshgrid(x,y);          % Criação da matriz da malha 'xy'
Z = X.^2 + Y.^2;                % Cálculo da função z = f(x,y)
mesh(X,Y,Z)                     % Traçado do gráfico da função 'z'

```

O resultado deste exemplo é mostrado na figura 1.7. A função **mesh** cria uma *malha tridimensional* em que cada ponto é unido por segmentos de reta aos vizinhos na malha. Usando a função **surf** é possível gerar um gráfico de superfície em que os espaços entre os segmentos são coloridos. Em ambos os casos, uma quarta matriz pode ser usada como parâmetro para estabelecer as cores a serem usadas no desenho. Se esta matriz for omitida, como no exemplo anterior, as cores das linhas serão relacionadas com a altura da malha sobre o plano xy . As duas funções podem receber somente a matriz Z como parâmetro, traçando um gráfico de malha cujos valores de x e y correspondem aos índices da matriz.

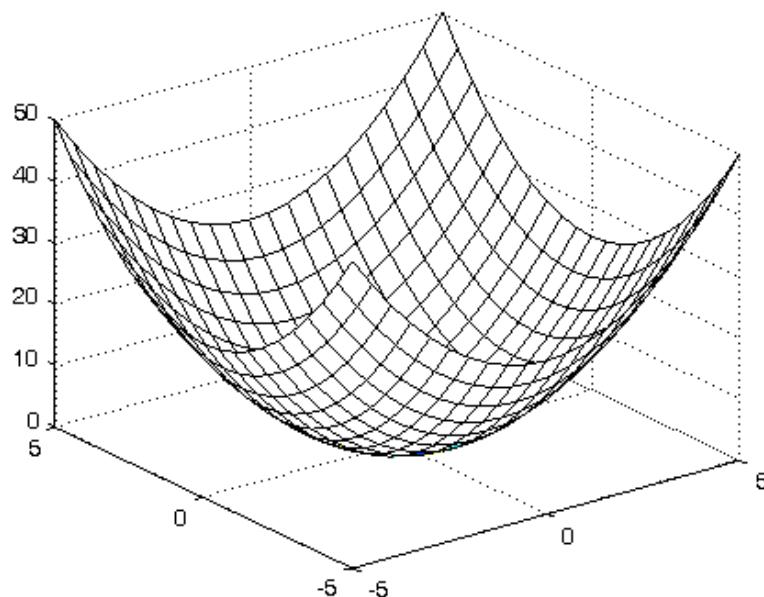


Figura 1.7: Gráfico de superfície tridimensional

1.8 Polinômios

O MATLAB possui funções específicas para operações com polinômios, como a determinação de raízes, avaliação, diferenciação, etc. Uma função polinomial da forma

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

pode ser representada no MATLAB por um **vetor de coeficientes**, em ordem decrescente de potência:

$$p = [a_n \quad a_{n-1} \quad \cdots \quad a_2 \quad a_1 \quad a_0].$$

Por exemplo, o polinômio $g(x) = x^3 - 2x - 5$ pode ser representado pelo seguinte vetor:

`g = [1 0 -2 -5];`

As raízes (reais ou complexas) de um polinômio podem ser calculadas diretamente em um vetor coluna pela função **roots**. Exemplo:

```
>> r = roots(g) ↴
r =
    2.0946
   -1.0473 + 1.1359i
   -1.0473 - 1.1359i
```

Note a forma de representação de números complexos no MATLAB (parte real + parte imaginária + *i*). De fato, o MATLAB reconhece automaticamente as letras *i* e *j* como a unidade imaginária da matemática.

De forma inversa, se forem conhecidas as raízes de um polinômio, a função **poly** reconstrói o polinômio original. Por exemplo, os coeficientes do vetor **g** do exemplo anterior, podem ser recuperados pela instrução:

```
p1 = poly(r) % Atenção: o argumento da função poly deve ser um vetor coluna!
```

1.8.1 Avaliação, multiplicação, divisão e diferenciação

Avaliar um polinômio significa determinar o valor de $p(x)$ para um dado valor de x . Para calcular, por exemplo, $g(2.4)$ usa-se a função ***polyval***, como em:

```
>> y = polyval(g, 2.4) ↴
y =
4.0240
```

As operações de multiplicação e divisão entre polinômios correspondem, respectivamente, a operações de *convolução* e *deconvolução*, implementadas pelas funções ***conv*** e ***deconv***. Por exemplo, considere:

$$n(s) = 3s^2 + s + 1 \quad \text{e} \quad d(s) = s + 1.$$

O produto $n(s)d(s)$ pode ser calculado com a seguinte seqüência de comandos:

```
>> n = [3 1 1]; ↴
>> d = [1 1]; ↴
>> prod = conv(n, d) ↴
prod =
3     4     2     1
```

Note que o grau do polinômio resultante é dado pela soma dos graus dos polinômios envolvidos na multiplicação. Finalmente, a derivada de uma função polinomial pode ser obtida diretamente a partir do vetor que representa a função com o uso da função ***polyder***.

Por exemplo, a derivada de $f(x) = 2x^3 + x^2 - 3x$ pode ser calculada com:

```
>> f = [2 1 -3 0] ↴
>> f1 = polyder(p) ↴
f1 =
6     2    -3
```

1.9 Funções de transferência

Considere um sistema linear em que se possa monitorar uma variável de saída, gerada pela ação de uma variável de entrada. Neste caso, define-se a **função de transferência** do sistema como a relação entre a transformada de Laplace da variável de saída e a transformada de Laplace da variável de entrada, considerando condições iniciais nulas. Existe uma classe própria no MATLAB para funções de transferência, criadas pela função ***tf*** e definidas pelo quociente de dois polinômios na variável s . Por exemplo, a função de transferência

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

pode ser armazenada em uma variável no MATLAB pela seguinte seqüência de comandos:

```
>> n = 3; ↴
>> d = [1 2 3]; ↴
```

```
>> G = tf(n,d) ↵
Transfer function:
3
-----
s^2 + 2 s + 3
```

1.10 Simulações

Existem funções específicas para simular o comportamento de sistemas lineares a entradas tipo impulso, degrau ou entradas genéricas. Para simular a resposta a um *impulso unitário* (em $t = 0$ s) de um sistema linear utiliza-se a função ***impulse***, fornecendo os polinômios representativos da função de transferência do sistema ou a própria função. Considerando as variáveis **n** e **d** do exemplo anterior, pode-se usar indistintamente

`impulse(n,d)` ou `impulse(G)`

O resultado da simulação é apresentado em uma janela gráfica, como mostra a figura 1.8. Opcionalmente, pode-se fornecer um valor em segundos para o tempo final de simulação:

```
impulse(G,10); % Simula a resposta ao impulso por 10 s.
```

É possível, ainda, armazenar os vetores do tempo de simulação (criado automaticamente pelo MATLAB) e da resposta do sistema, sem desenhar o gráfico correspondente. Exemplo:

```
[y t] = impulse(G,10); % Simula por 10 s. Retorna vetores de tempo e saída
```

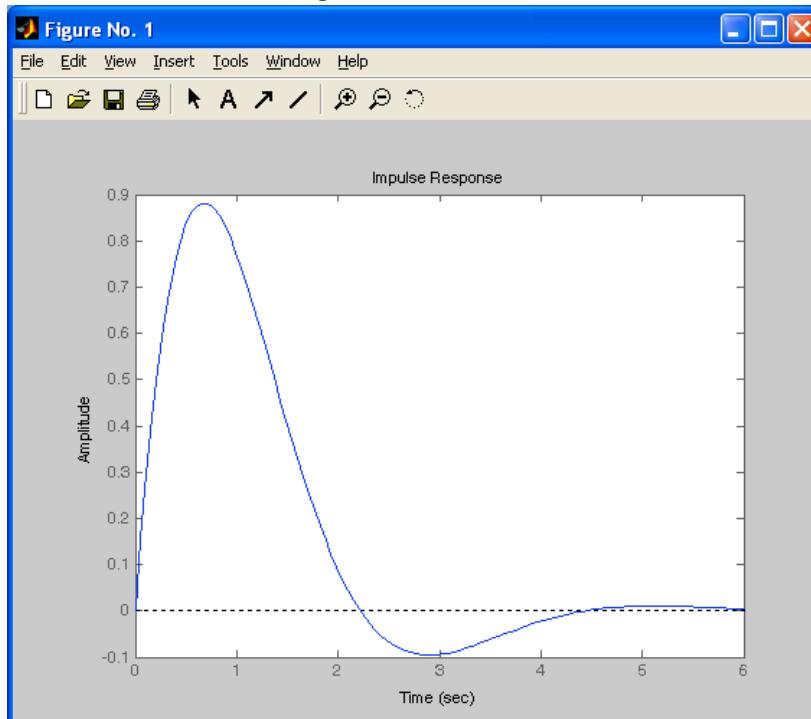


Figura 1.8: Resposta ao impulso

A simulação da resposta a uma entrada em *degrau unitário* é feita pela função ***step***, como em:

```
step(G); % Opção: step(n,d);
```

O resultado desta simulação está representado na figura 1.9.

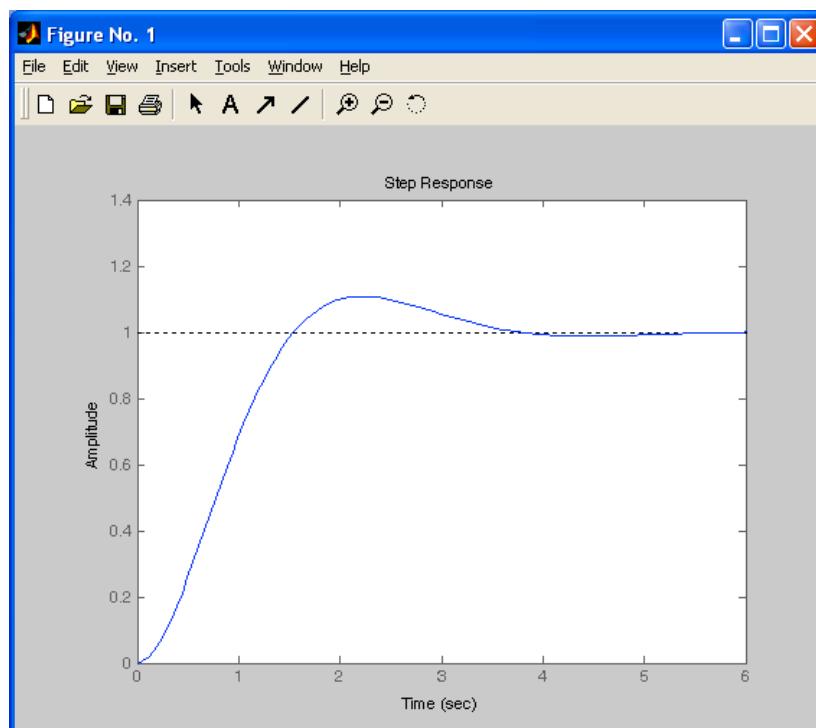


Figura 1.9: Resposta ao degrau unitário

Não é possível alterar a amplitude do degrau usado na simulação. No entanto, é possível controlar o tempo de simulação e armazenar os vetores de resposta (saída e tempo). Exemplo:

```
>> [y t] = step(G,10); ↴
```

Como se trata da simulação de um sistema linear, a saída para uma entrada em degrau de amplitude A pode ser calculada como $y_2(t) = Ay(t)$. Finalmente as funções **impulse** e **step** permitem que o usuário forneça um vetor de tempos a ser usado na simulação. Exemplo:

```
t = 0:0.01:15; ↴
step(n,d,t); ↴
```

Assim como no caso da função **plot**, pode-se sobrepor dois gráficos em uma mesma janela de figura. Finalmente, para simular a resposta de um sistema linear a uma entrada genérica é preciso usar a função **lsim**, fornecendo a especificação do sistema e os vetores de entrada e de tempo de simulação. Exemplo (usando o sistema **G** definido anteriormente):

```
t = 0:0.1:10;           % Vetor de tempo de simulação
u = zeros(length(t),1); % Vetor de entrada, com mesma dimensão de 't'
u(21:30) = 0.5;         % Atribuição de valores não nulos
lsim(G,u,t);           % Simulação
```

O resultado da simulação é apresentado em uma janela gráfica, como mostra a figura 1.10.

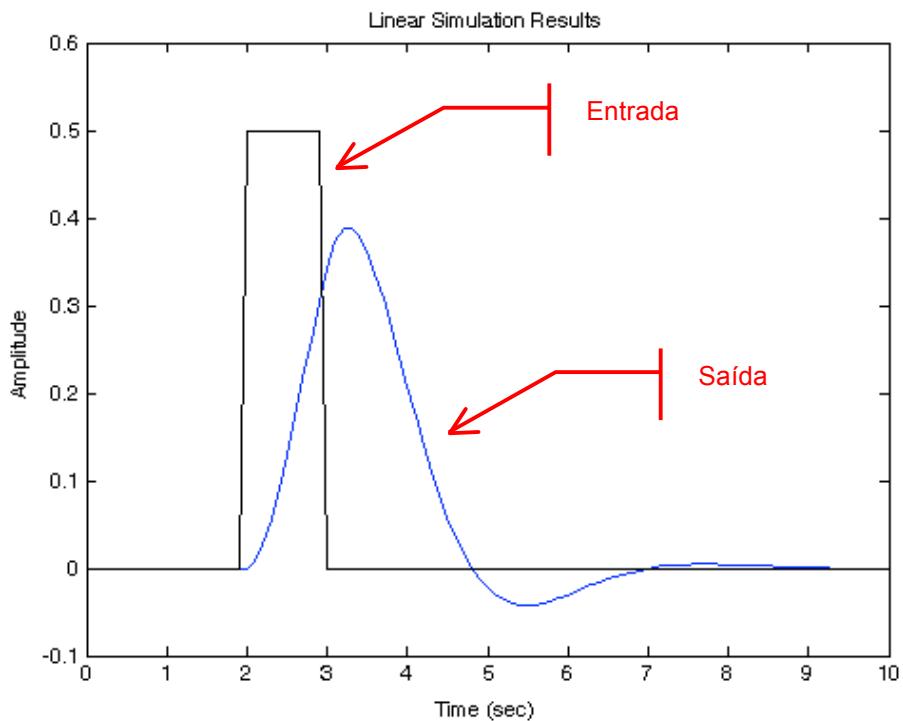


Figura 1.10: Resposta a um sinal genérico

Se for usada uma sintaxe com argumentos à esquerda a simulação será feita mas o gráfico não será desenhado. O vetor de saída criado pela função terá sempre o mesmo número de elementos do vetor de tempo fornecido.

2. Análise de sistemas lineares de 1^a e 2^a ordem – atividades

2.1 Sistemas de 1^a ordem

Obtenha a resposta ao degrau unitário dos sistemas definidos pelas seguintes funções de transferência:

a) $G_1(s) = \frac{10}{s + 5}$

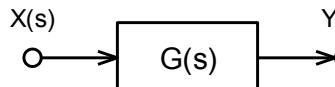
b) $G_2(s) = \frac{0,8}{s + 2}$

c) $G_3(s) = \frac{1}{0,01s + 1}$

Em seguida, determine:

- i) os pólos do sistema;
- ii) o valor final teórico da resposta (usando o Teorema do Valor Final);
- iii) o valor final do sinal de resposta (a partir da simulação);
- iv) a constante de tempo do sistema.

Finalmente, discuta a estabilidade de cada sistema, classificando-os como estáveis (E), marginalmente estáveis (ME) ou instáveis (I). Teorema do Valor Final:



$$y(\infty) = \lim_{s \rightarrow 0} s [X(s)G(s)] = \lim_{s \rightarrow 0} s Y(s)$$

2.2 Resposta temporal

Reescreva as funções de transferência anteriores, sob a forma

$$\bar{G}(s) = \frac{K}{\tau s + 1}$$

e verifique a relação entre esta forma e os parâmetros da resposta ao degrau: constante de tempo e valor final.

2.3 Sistemas de 2^a ordem

Obtenha a resposta ao degrau unitário dos sistemas definidos pelas seguintes funções de transferência:

a) $G_4(s) = \frac{1}{s^2 + s + 1}$

b) $G_5(s) = \frac{9}{s^2 + 3s + 9}$

c) $G_6(s) = \frac{25}{s^2 + 7s + 25}$

d) $G_7(s) = \frac{25}{s^2 + 10s + 25}$

Observação: Nas simulações de resposta ao degrau criadas pela função *step* as principais características de desempenho podem ser obtidas diretamente na janela gráfica: clique com o botão direito do *mouse* sobre uma área livre do gráfico e selecione, no menu *Characteristics*, as opções *Peak Response* (ultrapassagem ou sobressinal), *Settling Time* (tempo de assentamento), *Rise Time* (tempo de subida) e *Steady State* (valor final).

Determine:

- i) os pólos e zeros do sistema;
- ii) o valor final teórico da resposta (usando o Teorema do Valor Final);
- iii) o valor final do sinal de resposta (a partir da simulação);

- iv) o tempo de subida e de acomodação do sinal de resposta;
- v) o valor da ultrapassagem (ou sobressinal).

Em seguida, discuta a estabilidade de cada sistema e classifique-os em subamortecidos (SB), sobreamortecidos² (SO), criticamente amortecidos (CA) ou oscilatórios (O).

2.4 Sistemas de 2^a ordem sem zeros

Reescreva as funções de transferência do item anterior sob a forma

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

e identifique os valores de ζ e ω_n . Verifique a influência da *frequência natural* (ω_n) na velocidade da resposta de sistemas com a mesma *relação de amortecimento* (ζ). Em seguida, analise a influência da relação ζ em sistemas com a mesma freqüência ω_n . Para qual caso o sobressinal da resposta se mantém igual?

2.5 Validade do Teorema do Valor Final

Usando o Teorema do Valor Final, determine o valor estacionário da resposta ao degrau unitário do sistema definido por:

$$G_8(s) = \frac{s+2}{s^3+1}.$$

Simule a resposta ao degrau e compare com o resultado analítico. Por que o teorema falhou?

2.6 Aproximações

- a) Obtenha em uma mesma janela de figura as respostas ao degrau de

$$G_4(s) = \frac{1}{s^2 + s + 1} \quad \text{e} \quad G_9(s) = \frac{10}{s^3 + 11s^2 + 11s + 10}$$

Repita (em outra janela de figura) para uma entrada em impulso. O que se pode concluir?

- b) Obtenha em uma mesma janela de figura as respostas ao degrau unitário de

$$G_{10}(s) = \frac{2}{s^2 + 2s + 2} \quad \text{e} \quad G_{11}(s) = \frac{s+2}{s^2 + 2s + 2}.$$

Discuta os efeitos causados na resposta ao degrau de $G_{10}(s)$ pela inclusão do zero ($s = -2$).

² Também chamados de superamortecidos.

3. Aproximações e estabilidade de sistemas lineares

3.1 Sistemas de 3^a ordem

A característica dominante de um sistema de 3^a ordem sem zeros pode ser identificada por meio de *expansão em frações parciais*. Por exemplo, a função de transferência

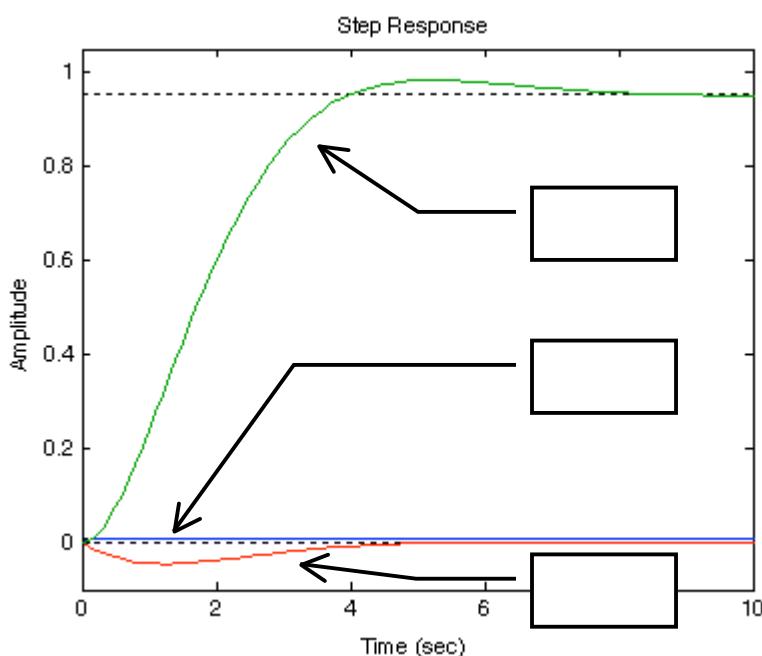
$$G(s) = \frac{7,98}{s^3 + 11,4s^2 + 14s + 7,98},$$

pode ser reescrita como:

$$\bar{G}(s) = G_1(s) + G_2(s) + G_3(s) = \frac{0,0893}{s + 10,091} + \frac{0,7530}{s^2 + 1,3090s + 0,7908} - \frac{0,0893s}{s^2 + 1,3090s + 0,7908}$$

3.1.1 Exemplo

Obtenha em uma mesma janela de figura as respostas ao degrau de cada termo da função $\bar{G}(s)$, dada anteriormente. Verifique qual termo possui maior influência na resposta dinâmica do sistema. Confirme sua análise a partir da resposta ao degrau do sistema de 3^a ordem, $G(s)$.



3.2 Aproximações para modelos de 2^a ordem

Os passos a seguir descrevem uma forma de se aproximar um sistema de ordem superior, sem zeros, para um sistema de 2^a ordem:

- 1) Calcule (ou visualize no plano complexo) os pólos originais da função de transferência – use as funções *roots* e *pzmap*;
- 2) Despreze o pólo (ou pólos) que tiver menor influência na resposta dinâmica do sistema para obter um novo denominador de 2^a ordem;
- 3) Obtenha um numerador que mantenha o ganho estático (ganho DC) do sistema original;
- 4) Verifique (por exemplo, graficamente) a qualidade da aproximação obtida.

3.2.1 Exemplos

a) Usando o conceito de *pólos dominantes*, escreva modelos de 2^a ordem para:

$$i) \quad G_1(s) = \frac{7,98}{s^3 + 11,4s^2 + 14s + 7,98}$$

$$ii) \quad G_2(s) = \frac{73,626}{(s+3)(s^2 + 4s + 24,542)}$$

b) Considere um sistema definido pela função de transferência:

$$G(s) = \frac{1,278s + 12,78}{s^3 + 11,72s^2 + 17,626s + 4,26}$$

i) Reescreva a função sob a forma $T(s) = K \frac{(s - z_1)}{(s - p_1)(s - p_2)(s - p_3)}$.

ii) Usando o resultado anterior, obtenha uma aproximação de 2^a ordem para $G(s)$.

c) Escreva uma seqüência de comandos que crie uma aproximação de 2^a ordem, $G_2(s)$, para:

$$G_3(s) = \frac{156,25}{s^4 + 16s^3 + 78,75s^2 + 81,25s + 78,125}.$$

Use apenas instruções literais, isto é, que não envolvam valores numéricos diretamente.

3.3 Conexões entre sistemas

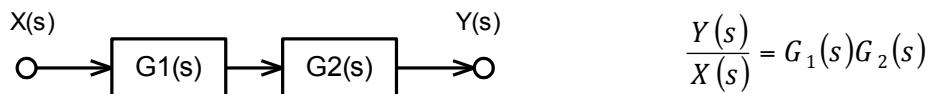
O MATLAB possui funções para determinar o efeito de algumas formas de interconexão entre funções de transferência. Nos exemplos a seguir, considere que se deseja obter

$$\frac{Y(s)}{X(s)} = \frac{n(s)}{d(s)}$$

a partir das funções:

$$G_1(s) = \frac{n_1(s)}{d_1(s)} \text{ e } G_2(s) = \frac{n_2(s)}{d_2(s)}.$$

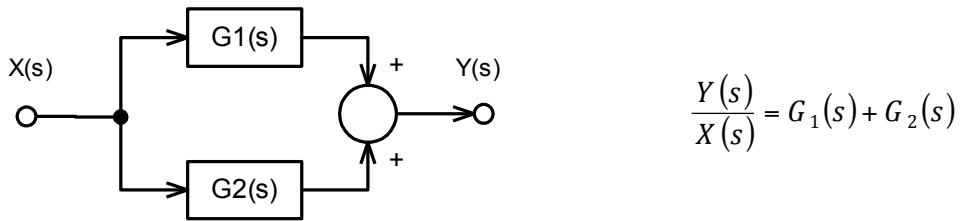
a) Conexão em cascata



Comandos:

<pre>[n d] = series(n1,d1,n2,d2) [n d] = series(G1,G2) FT = G1*G2</pre>	$\left. \right\} \quad \begin{matrix} 3 \text{ opções para a} \\ \text{mesma operação} \end{matrix}$
---	--

b) Conexão em paralelo



Comandos:

$[n \ d] = \text{parallel}(n1, d1, n2, d2)$ $[n \ d] = \text{parallel}(G1, G2)$ $FT = G1 + G2$	$\left. \right\} 3 \text{ opções para a mesma operação}$
--	--

3.4 Estabilidade

Sabe-se que um sistema em malha fechada é estável se sua função de transferência não apresentar pólos no *semiplano direito* do plano complexo, ou seja, se nenhum polo tiver parte real positiva. Por exemplo, considere o sistema da figura 3.1.

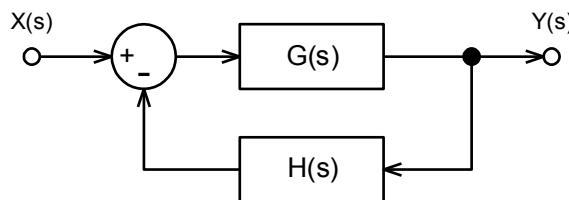


Figura 3.1: Sistema de controle em malha fechada

A função de transferência em malha fechada deste sistema é dada por:

$$FTMF = \frac{Y(s)}{X(s)} = \frac{G(s)}{1 + G(s)H(s)}.$$

O denominador da função de transferência em malha fechada dá origem à *equação característica* do sistema, definida como:

$$q(s) = 1 + G(s)H(s) = 0$$

Para um sistema como o da figura 3.1, a função **feedback** do MATLAB permite determinar diretamente a função de transferência em malha fechada. Exemplo:

```
>> MF = feedback(G, H) ↴
```

O padrão da função **feedback** é trabalhar com realimentação negativa. Para sistemas com *realimentação negativa unitária*, isto é, quando $H(s) = 1$, usa-se:

```
>> MF = feedback(G, 1) ↴
```

3.4.1 Exemplos

Investigue a estabilidade dos sistemas a seguir, admitindo a configuração da figura 3.1, com realimentação negativa unitária. Sistemas marginalmente estáveis devem ser classificados como instáveis.

a) $G(s) = \frac{1}{s^3 + s^2 + 2s + 1,5}$ () Estável () Instável

b) $G(s) = \frac{1}{s^2 + 2s - 4}$ () Estável () Instável

c) $G(s) = \frac{1}{s^4 + 6,5s^3 + 14s^2 + 11,5s + 2}$ () Estável () Instável

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

4. Programação

Um dos aspectos mais poderosos do MATLAB é a possibilidade de se criar programas em uma linguagem de programação interpretada³ usando a mesma notação aceita na janela de comando. Arquivos contendo código MATLAB são arquivos de texto com a extensão **.m** chamados de *arquivos-M* (*M-files*). Estes arquivos podem conter o código de **scripts** ou **funções**, cujas principais características estão relacionadas na tabela 4.1.

Tabela 4.1: Características das formas de programação MATLAB

Arquivos de <i>script</i>	Arquivos de funções
Não aceitam argumentos nem retornam valores ao <i>workspace</i> .	Aceitam argumentos e retornam valores ao <i>workspace</i> .
Trabalham com as variáveis definidas no <i>workspace</i> .	Trabalham com variáveis definidas localmente ao arquivo.
Principal aplicação: automatização de comandos que precisam ser executados em uma certa seqüência.	Principal aplicação: adaptação da linguagem MATLAB a qualquer situação de programação necessária.

4.1 Exemplo – análise de um sistema linear

Para ilustrar as formas de programação possíveis vamos criar um *script* para a análise de desempenho de um sistema linear de 2^a ordem definido, pela função de transferência:

$$F(s) = \frac{5}{s^2 + 2s + 5}.$$

4.1.1 Criando um *script*

Os *scripts* constituem a forma mais simples de programação em ambiente MATLAB porque apenas automatizam uma série de comandos. É possível usar qualquer editor de textos para a criação de *scripts*, mas o uso do editor embutido no MATLAB é preferido por fornecer recursos úteis ao programador como auto-indentação, destaque de palavras reservadas, ferramentas de depuração, etc. Digite **edit** na janela de comando ou clique em *File > New > M-file* para invocar o editor. Para este exemplo, digite o código listado a seguir e salve-o com o nome **Analise1.m**. Os comentários podem ser omitidos.

```
% ANALISE1.M - Script para análise de desempenho.
% Simula a resposta ao degrau e ao impulso de:
%
%      5
% F(s) = -----
%           s^2 + 2s + 5
%
% Atenção: cria os polinômios 'n' e 'd' e a função
% de transferência 'F' no workspace!
```

```
clear; % Limpa todas as variáveis da memória
```

³ Como a linguagem de programação do MATLAB é *interpretada*, todos os códigos precisam ser executados a partir do MATLAB. É possível criar executáveis independentes, assunto que não será discutido neste material.

```

clc; % Limpa a janela de comando
n = 5; % Define o numerador da função de transferência
d = [1 2 5]; % Define o denominador da função de transferência
F = tf(n,d); % Cria a função de transferência F
figure(1); % Cria a janela gráfica 1
step(F,10); % Simula por 10 s e desenha a resposta ao degrau de F
figure(2); % Cria a janela gráfica 2
impulse(F,10); % Simula por 10 s e desenha a resposta ao impulso de F

```

Antes de executar o *script* é preciso que o MATLAB reconheça a pasta em que o arquivo **.m** foi gravado como um diretório de trabalho. Digite **cd** na janela de comando para descobrir o diretório de trabalho atual e, se necessário, altere-o para o diretório onde o *script* foi gravado. Por exemplo, se o *script* foi gravado em **C:\TEMP**, digite:

```
>> cd c:\temp ↵
```

Execute o *script*, digitando seu nome (**Analise1**) na janela de comando. Neste exemplo, o resultado gráfico é exibido em duas janelas (ver figura 4.1). Verifique que as variáveis **F**, **n** e **d** (e somente elas!) permanecem no *workspace* após a execução do *script*.

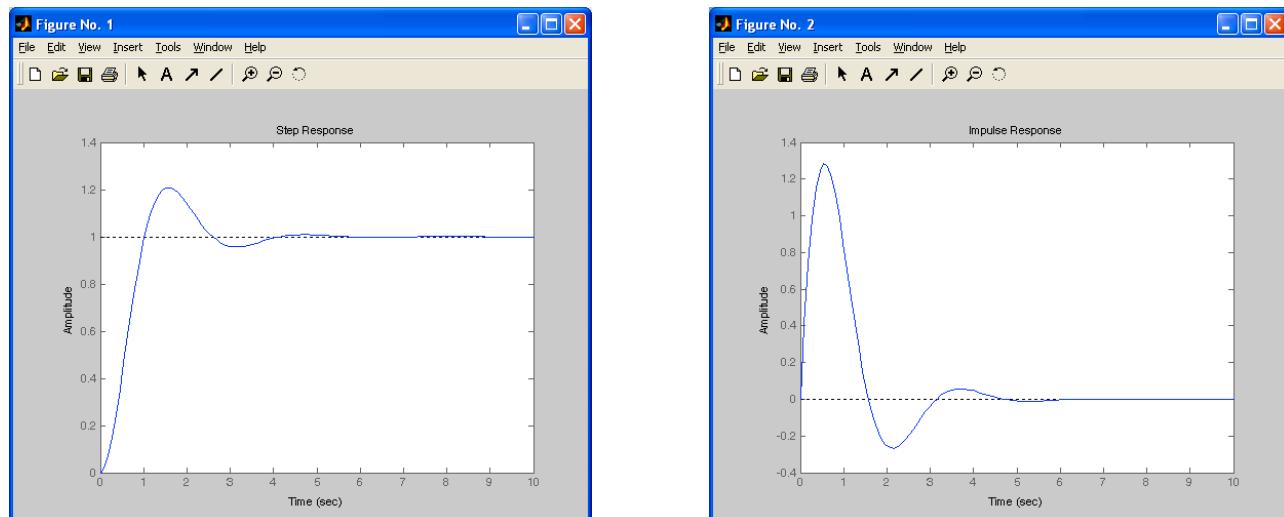


Figura 4.1: Resultados gráficos do script

4.1.2 Análise do *script*

As primeiras linhas do código proposto contém comentários que são exibidos pelo comando **help** quando o usuário pede ajuda sobre o *script*:

```
>> help Analise1 ↵
```

ANALISE1.M – Script para análise de desempenho.
Simula a resposta ao degrau e ao impulso de:

$$F(s) = \frac{5}{s^2 + 2s + 5}$$

Atenção: cria os polinômios '**n**' e '**d**' e a função de transferência '**F**' no *workspace*!

A primeira linha de comentário, chamada de **linha H1** é usada nas buscas por palavra-chave do comando **lookfor**. Exemplo:

```
>> lookfor desempenho ↵
```

Analisel.m: % ANALISE1.M - Script para análise de desempenho.

4.1.3 Criando uma função

Funções são *arquivos-M* que estendem a capacidade de processamento dos *scripts* por aceitar argumentos de entrada e retornar valores para o *workspace*. Cada função trabalha com variáveis locais, isoladas do espaço de memória do *workspace*. Além disso, as funções podem ser executadas mais rapidamente que os *scripts* por serem compiladas internamente em um *pseudo-código* que é mantido em memória, aumentando a velocidade de execução caso a função seja chamada mais de uma vez.

A primeira linha de um arquivo de função depois dos comentários iniciais deve conter a palavra-chave **function** seguida pela definição dos valores de retorno, nome da função e pela lista de argumentos de entrada⁴. Como exemplo, analise o código listado a seguir de uma função (**Media**) para cálculo da média dos elementos de um vetor (o uso da estrutura **if** dera explicado futuramente).

```
% MEDIA.M - Função para cálculo da média dos elementos de um vetor
% Versão simplificada, com verificação básica de dimensão do vetor
% Forma de uso: y = media(V) , sendo V um vetor linha ou coluna

function y = Media(V)

[nl nc] = size(V);      % Obtém dimensões do vetor
if (nl==1 & nc==1)      % Se nl=nc=1, V é um escalar
    disp('Erro: a função não trabalha com escalares');
else
    y = sum(V)/length(V);    % Se não for escalar, calcula a média
end    % Fim da estrutura condicional
```

Exemplo de uso da função:

```
>> a = [1 2 4 6 7 7]; ↵
>> b = Media(a) ↵

b =
4.5000
```

Note que o valor de retorno da função foi o último valor atribuído internamente à variável **y**. O final do código de uma função não precisa de nenhum identificador específico.

4.1.4 Atividade

Pesquise detalhes de funcionamento das funções **disp** e **sum**, usadas no código da função **Media**. Consulte também a documentação da função **mean**.

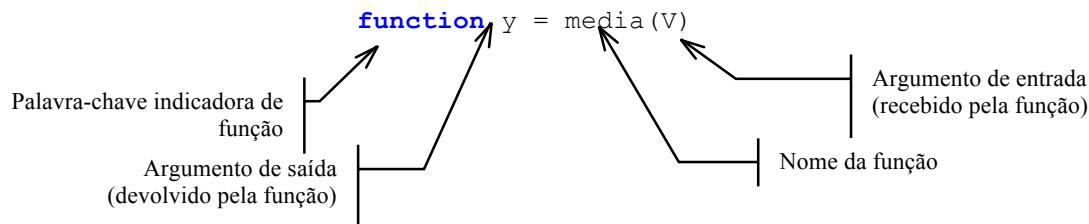
4.1.5 Convenções

Os nomes de funções no MATLAB têm as mesmas restrições que os nomes de variáveis: devem ter, no máximo, 31 caracteres e iniciar por uma letra seguida por qualquer combinação de caracteres, números ou caracteres de sublinhado. O nome do arquivo que contém o código da função deve ser formado pelo nome da

⁴ São válidas as mesmas observações feitas para os *scripts* em relação à utilidade das primeiras linhas do código.

função, com extensão **.m**. Se houver divergências, o nome do arquivo prevalecerá sobre o nome interno da função.

A linha de definição da função deve seguir uma forma padronizada que identifique a quantidade de parâmetros recebidos ou devolvidos pela função. Para a função **Media**, usada anteriormente como exemplo, pode-se identificar:



Se a função retornar mais de um valor deve-se especificar uma lista de argumentos de retorno, delimitada por colchetes. Argumentos de entrada, se existirem, devem estar entre parênteses. As duas listas devem ter seus valores, *obrigatoriamente*, separados por vírgulas. Exemplo:

```

% PONTOMEDIO.M - Calculo das coordenadas do ponto médio entre
% pontos. A função deve receber 2 vetores de coordenadas (x,y)

function [x,y] = PontoMedio(coord_x,coord_y)

if(length(coord_x)==length(coord_y))
    x = mean(coord_x);
    y = mean(coord_y);
else
    disp('Erro nas dimensões dos vetores!');
end
  
```

Exemplo de execução da função **PontoMedio**:

```

>> cx = [1 2 4 1 4 5]; ↴
>> cy = [7 2 3 5 0 8]; ↴
>> [x,y] = PontoMedio(cx,cy) ↴

x =
2.8333
y =
4.1667
  
```

A lista de retorno pode ser omitida ou deixada vazia se não houver resultado a ser devolvido, ou seja, pode-se escrever um cabeçalho como

```
function Calcula(x)     ou     function [] = Calcula(x)
```

Como mostrado no exemplo da função **Media** as variáveis passadas para a função não precisam ter os mesmos nomes usados na definição da função.

4.2 Controle de fluxo

Como a maioria das linguagens de programação, o MATLAB permite o uso de estruturas condicionais e repetitivas para controle de fluxo⁵. Neste material discutiremos algumas formas de uso dos comandos ***if***, ***while*** e ***for***.

4.2.1 Estrutura condicional ***if***

O comando ***if*** avalia uma expressão e, dependendo de seu valor, executa um determinado conjunto de instruções. Em sua forma mais simples, usa-se:

```
if expressão
    <COMANDOS>
end
```

Se a expressão lógica for verdadeira (diferente de zero), todos os comandos até o finalizador ***end*** serão executados. Em caso contrário (expressão igual a zero), a execução do código continuará na instrução após o finalizador ***end***. A forma completa da estrutura inclui a declaração ***else*** para a indicação de comandos a serem executados se a expressão for falsa:

```
if expressão
    <COMANDOS V>
else
    <COMANDOS F>
end
```

As expressões podem incluir operadores relacionais e lógicos, como mostrado na tabela 4.2.

Tabela 4.2: Operadores relacionais e lógicos

Operadores relacionais		Operadores lógicos	
<	Menor que	&	Operação "E"
<=	Menor ou igual		Operação "OU"
>	Maior que	~	Negação lógica
>=	Maior ou igual		
==	Igual a		
~=	Diferente de		

Exemplo:

```
...
if x == 0
    y = sin(3*t+a);
else
    y = cos(3*t-a);
end
...
```

4.2.2 Estrutura repetitiva ***while***

⁵ O MATLAB reconhece 6 estruturas de controle de fluxo: *if*, *switch*, *while*, *for*, *try-catch* e *return*.

O laço **while** executa um grupo de comandos enquanto uma expressão de controle for avaliada como verdadeira. A sintaxe para este tipo de laço é:

```
while expressão
    <COMANDOS>
end
```

Exemplo de um trecho de código que simula a operação da função interna **sum**:

```
...
S = 0;
i = 1;
while i<=length(V)
    S = S+V(i);
    i = i+1;
end    % Neste ponto, S = sum(V)
...
```

4.2.3 Estrutura repetitiva **for**

O laço **for** executa repetidamente um conjunto de comandos por um número especificado de vezes. Sua forma geral é:

```
for variável de controle = valor inicial: incremento: valor final
    <COMANDOS>
end
```

O incremento pode ser negativo ou omitido (caso em que será adotado um incremento unitário). Exemplo de um trecho de código que simula a operação da função interna **max**:

```
...
M = V(1);
for i = 2:length(V)    % O incremento foi omitido!
    if V(i) > M
        M = V(i);
    end
end    % Neste ponto, M = max(V)
...
```

4.3 Vetorização

O acesso *vetorizado* é uma opção aos laços de acesso individual a elementos de vetores ou matrizes. Normalmente, o acesso vetorizado é mais rápido que o acesso individual, feito com laços de repetição. Exemplo de uso:

Acesso convencional

```
i = 0;
for t = 0:0.001:99.999
    i = i+1;
    y(i) = sin(t);
end
```

Acesso vetorizado

```
t = 0:0.001:99.999;
y = sin(t);
```

Versões recentes do MATLAB (versão 6 em diante) fazem operações automáticas de otimização que, na prática, produzem ganhos de desempenho compatíveis com os da vetorização.

4.4 Entrada de dados

A função **input** permite a entrada de dados ou expressões durante a execução de *scripts* ou funções, exibindo (opcionalmente) um texto ao usuário. Exemplo:

```
Kp = input('Digite o ganho da ação proporcional: ');
```

Se o valor de entrada for uma expressão, seu valor será avaliado antes da atribuição à variável usada no comando. Se o valor de entrada for um texto o caractere 's' (*string*) deve ser incluído na lista de argumentos da função. Exemplo:

```
Titulo_Grafico = input('Título do gráfico: ','s');
```

Outra forma disponível de interação via teclado é dada pela função **pause**. Quando usada sem argumentos, a instrução interrompe a execução de um *script* ou função até que o usuário pressione alguma tecla⁶. A função **pause** é especialmente útil para permitir ao usuário a leitura de várias informações impressas em tela ou durante a fase de depuração do programa.

4.5 Edição de funções existentes

O código da maioria das funções discutidas neste material pode ser visualizado ou editado, digitando-se:

```
>> edit <nome da função> ↵
```

Não é possível editar o código de funções internas do MATLAB como **inv**, **max**, etc. Apesar de possível, não é recomendável alterar diretamente o código das funções que acompanham o MATLAB. Se desejar⁷, crie uma nova versão com outro nome.

4.6 Subfunções

Os *arquivos-M* podem conter mais de uma função. A primeira delas, cujo nome deve coincidir com o nome do arquivo, é a função **primária** enquanto as demais são **subfunções**. As subfunções podem ser definidas em qualquer ordem após a função primária e suas variáveis sempre tem escopo local⁸. Não é preciso usar qualquer indicação especial de fim de função porque a presença de um novo cabeçalho indica o fim da função (ou subfunção) anterior. Como exemplo, analise o código da função **Baskara**, listado a seguir.

```
% BASKARA.M - Exemplo de uso de uma subfunção para
% cálculo de raízes de equação do 2o grau

% Função primária: mesmo nome que o do arquivo .M
function x = Baskara(v)

a = v(1); b = v(2); c = v(3);      % Obtém coeficientes
D = Delta(a,b,c);                  % Calcula "delta"
```

```
% Calcula raízes reais, se existirem
if isreal(D)
    r1 = (-b+D)/(2*a);    % Calcula raiz
    r2 = (-b-D)/(2*a);    % Calcula raiz
    if r1 == r2
        x = r1;            % Retorna apenas uma raiz
```

⁶ A função **pause** também pode ser usada com argumentos. Quando usada sob a forma **pause(N)**, a função interrompe a execução do código atual por N segundos.

⁷ E souber o que está fazendo.

⁸ É possível criar variáveis globais, reconhecidas em todo o código – consulte a documentação do MATLAB.

```

else
    x = [r1; r2];      % Retorna raízes distintas
end
else
    disp('A equação não possui raízes reais');
    x = [];      % Retorno nulo
end

% Subfunção para cálculo de "delta"
function d = Delta(a,b,c)
d = sqrt(b^2-4*a*c);

```

4.7 Exemplos de aplicação

4.7.1 Aproximações

Crie uma função chamada *Aprox2* que obtenha uma aproximação de 2^a ordem a partir de uma função de transferência de 3^a ordem, da forma:

$$G_3(s) = \frac{n_3}{d_3(s)}.$$

A função deve receber como parâmetros o numerador (um número real) e o polinômio de 3º grau correspondente ao denominador. Os valores de retorno devem ser os polinômios do numerador e denominador da aproximação. Adicionalmente, devem ser superpostos os gráficos da resposta ao degrau das duas funções de transferência. Exemplo de uso:

$$G_3(s) = \frac{7,98}{s^3 + 11,4s^2 + 14s + 7,98}$$

- Instruções:

```

>> n3 = 7.98 ↴
>> d3 = [1 11.4 14 7.98]; ↴
>> [n2 d2] = Aprox2(n3,d3) ↴

```

- Técnica: eliminação do pólo não-dominante (supostamente, o único pólo real).

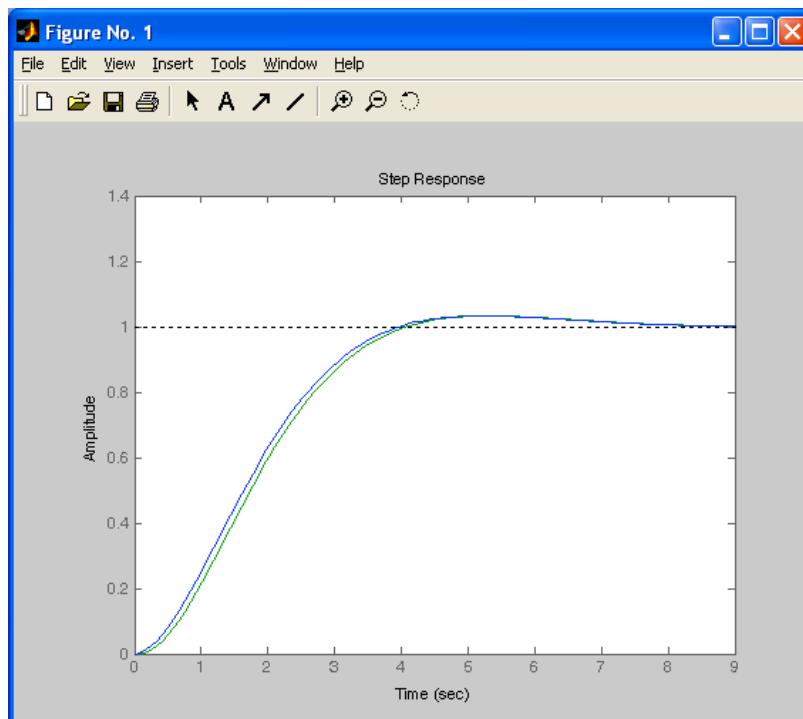
- Valores de retorno da função (para este exemplo)

```

n2 = 0.7908
d2 = 1.0000    1.3090    0.7908

```

- Resultado gráfico: ver exemplo na figura 4.2.

Figura 4.2: Exemplo do resultado gráfico da função *Aprox2*

4.7.2 Análise do erro em regime estacionário

Para sistemas realimentados estáveis o erro em regime estacionário é dado por:

$$e(\infty) = \lim_{t \rightarrow \infty} e(t) = \lim_{s \rightarrow 0} s E(s).$$

Como o MATLAB *sempre* calcula a resposta de sistemas lineares em **tempo discreto**, isto é, para apenas alguns instantes de tempo, pode-se calcular o vetor de erro (**e**) a partir dos vetores de entrada (**u**) e saída (**y**) de uma simulação, usando:

```
e = u-y; % ATENÇÃO: u e y devem ter a mesma dimensão!
```

Um exemplo deste cálculo é mostrado no código da função **Erro**, listado a seguir, que desenha o gráfico do erro e da resposta ao degrau do sistema mostrado na figura 4.3.

```
function [ess] = Erro(K)

clc
G = tf(K,conv([1 1],[1 5])); % Limpa a tela
MF = feedback(G,1); % Cria a função de ramo direto
[y t] = step(MF); % Cria a função em malha fechada
u = ones(length(y),1); % Obtém os vetores de simulação
e = u-y; % Cria o vetor representativo da entrada
plot(t,y,t,e) % Calcula o vetor de erros
% Traça os gráficos da saída e do erro
ess = e(length(e)); % Retorna o valor "final" do erro
```

No código, observe o uso da função **step** com argumentos à esquerda e como a função **plot** foi usada para sobrepor dois gráficos na mesma janela (uma alternativa ao uso de **hold**).

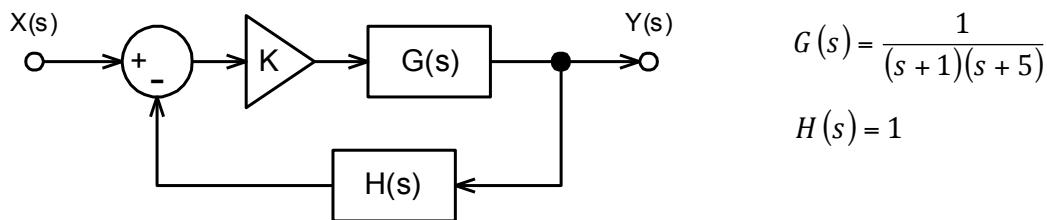


Figura 4.3: Sistema de controle realimentado

4.7.3 Atividade

Modifique o código da função **Erro** para aceitar como argumento de entrada o tempo de simulação e os parâmetros de $G(s)$ e $H(s)$. Verifique a influência do valor de K na resposta dinâmica do sistema.

4.7.4 Atividade – estabilidade em função de um parâmetro

A função **Pulos**, listada a seguir, calcula e desenha em um plano *xy* os pólos do sistema de controle mostrado na figura 4.4, para alguns valores de K entre 0 e 20.

```
% POLOS.M - Função para desenhar a posição dos pólos de um
% sistema realimentado em função do ganho K

function [] = Polos

K = [0:0.5:20]; % Cria vetor de ganhos
for i=1:length(K)
    q = [1 2 4 K(i)]; % Polinômio da equação característica
    p(:,i) = roots(q); % Calcula raízes para o ganho K(i)
end
plot(real(p),imag(p), 'bx'); % Desenha gráfico dos pólos
grid on;
xlabel('Eixo real'); % Nomeia os eixos
ylabel('Eixo imaginário'); % do gráfico
```

a) Analise o gráfico criado pela função e verifique se existem valores de K para os quais o sistema realimentado é instável.

b) Faça com que a função aceite a entrada dos valores do ganho máximo e do passo. Exemplo:

```
Kmax = input('Digite o valor do ganho máximo: ');
```

c) Modifique a função para detectar o valor de K que leva o sistema ao limite da estabilidade.

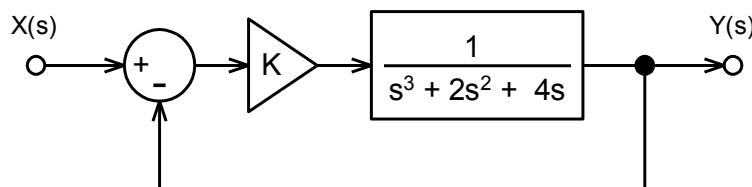


Figura 4.4: Sistema de controle realimentado

5. Lugar das raízes – introdução

O desempenho de um sistema linear pode ser analisado pelos pólos da sua função de transferência em malha fechada. Na prática, tenta-se ajustar a posição destes pólos para que o comportamento do sistema atenda a certas especificações de desempenho (sobressinal, tempo de assentamento, etc). Por exemplo, considere o sistema representado na figura 5.1.

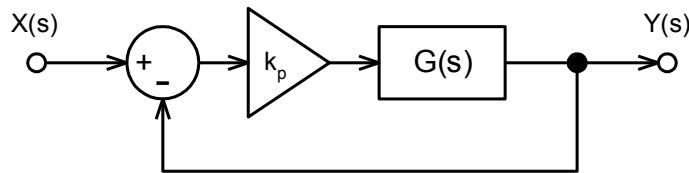


Figura 5.1: Sistema de controle em malha fechada

Neste caso, percebe-se que as raízes da equação característica do sistema, dada por

$$q(s) = 1 + k_p G(s) = 1 + k_p \frac{n(s)}{d(s)} = 0$$

dependem do valor do ganho K_p . O traçado do **lugar das raízes** de um sistema em malha fechada, em função de um ganho de ramo direto, pode ser obtido diretamente no MATLAB pela função **rlocus**. A partir da função de transferência em malha aberta,

$$G(s) = \frac{n(s)}{d(s)}$$

obtém-se o lugar das raízes do sistema em malha fechada, usando

```
rlocus(n, d); ou rlocus(G); % Sendo G = tf(n, d)
```

Neste caso, o MATLAB utiliza um vetor de ganhos criado automaticamente, com k_p variando de 0 a $+\infty$. Este vetor também pode ser fornecido pelo usuário, como no exemplo a seguir:

```
VG = [0:0.5:20]; % Cria vetor com ganhos de 0 a 20
rlocus(n, d, VG); % Obtém o lugar das raízes
```

O resultado é apresentado em uma janela de figura com os pólos e zeros de malha aberta indicados por "x" e "o", respectivamente. A função **rlocus** também pode fornecer as raízes calculadas e o vetor de ganhos utilizado nos cálculos, sem traçar o gráfico, usando-se:

```
[R Kp] = rlocus(n, d) % R = matriz de raízes; Kp = vetor de ganhos
```

Analizando-se estes valores de retorno é possível descobrir, por exemplo, quais valores de k_p podem tornar o sistema instável.

5.1 Exemplo

Considere um sistema como o da figura 5.1, com:

$$G(s) = \frac{1}{s^3 + 4s^2 + 6s + 1}.$$

O lugar das raízes deste sistema pode ser obtido, digitando-se:

```
>> MA = tf(1, [1 4 6 1]); ↴
>> rlocus(MA); ↴
```

O resultado gráfico está representado na figura 5.2. As setas indicam o sentido de deslocamento dos pólos de malha fechada do sistema com o aumento de k_p .

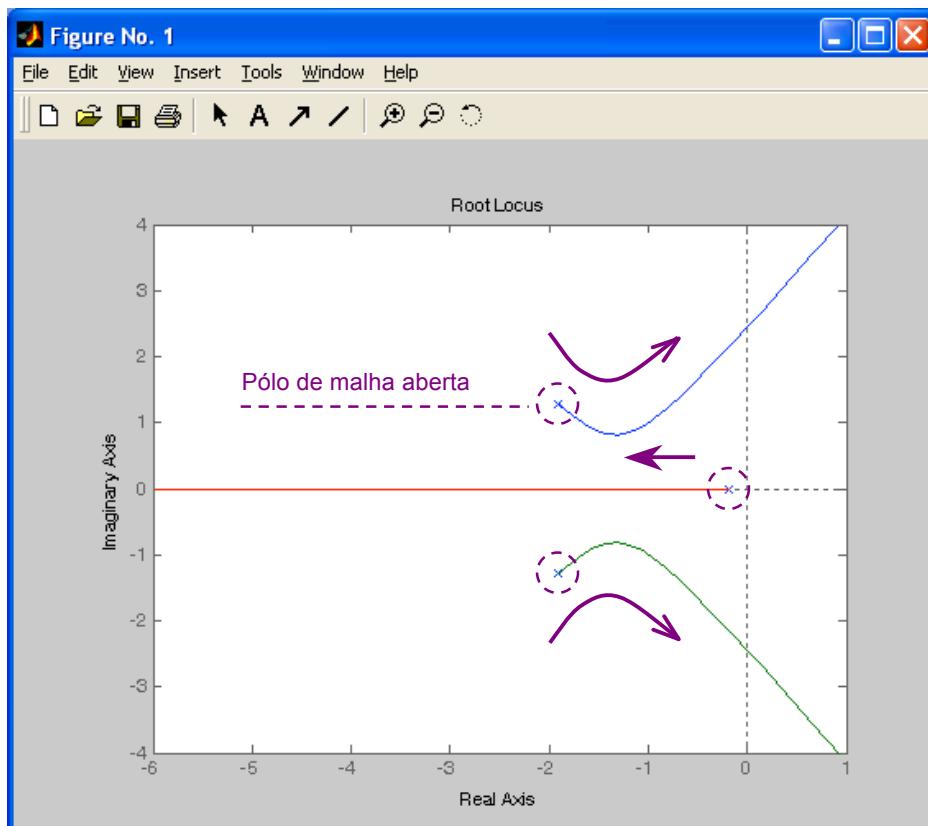


Figura 5.2: Lugar das raízes

5.1.1 Atividade

Descubra quantos (e quais) valores de ganho k_p foram usados pela função **rlocus** para a obtenção do lugar das raízes do sistema da figura 5.2.

5.2 Análise gráfica do lugar das raízes

A análise dos pólos do sistema em relação ao ganho k_p também pode ser feita de forma gráfica pelo uso da função **rlocfind**. Esta função calcula o ganho de ramo direto necessário para que um certo polo, selecionado pelo *mouse* na janela do lugar das raízes, seja obtido.

Para ver um exemplo de como isto pode ser feito, obtenha o lugar das raízes do sistema desejado e digite:

```
>> [g p] = rlocfind(MA) ↴
```

A mensagem "*Select a point in the graphics window*" será mostrada na janela de comando do MATLAB e um cursor e forma de cruz aparecerá na janela de figura, aguardando que um ponto seja selecionado. No exemplo a seguir (figura 5.3) foi considerado o lugar das raízes do sistema descrito como exemplo no item 5.1.

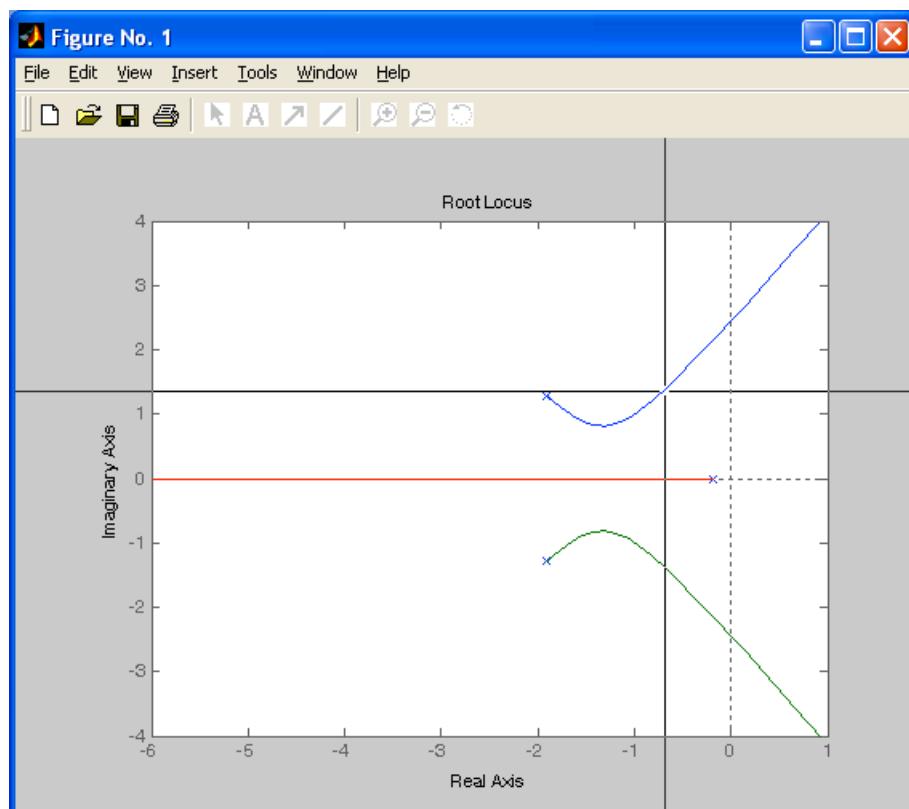


Figura 5.3: Seleção de um ponto sobre o lugar das raízes

Os resultados correspondentes são exibidos na janela de comando assim que um ponto for selecionado. Para o exemplo da figura 5.3, tem-se:

```
selected_point = -0.6836 + 1.3540i
g = 5.1271
p = -2.5984
-0.7008 + 1.3664i
-0.7008 - 1.3664i
```

Os pólos calculados no vetor **p** ficam destacados na janela de figura até que o gráfico seja redesenhado. Importante: a função **rlocfind** só funciona se existir uma janela de lugar das raízes criada pelo comando **rlocus**.

5.2.1 Atividade

Use a função **rlocfind** para estimar o ganho k_p que leva o sistema definido em 5.1 ao limite da estabilidade.

5.3 Exemplos

Obs.: todas as análises a seguir se referem a sistemas com a estrutura mostrada na figura 5.1.

- a) Determine o ganho crítico (limite da estabilidade) para um sistema com:

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

Na condição crítica, quais pólos são dominantes?

- b) Considere um sistema realimentado definido por:

$$G(s) = \frac{1}{s(s+2)(s^2 + 4s + 5)}.$$

Verifique se é possível obter pólos dominantes com parte real igual a -0,35 e determine para qual valor de k_p isto acontece.

- c) Considere um sistema definido pela função de transferência:

$$G(s) = \frac{(s+1)}{s^2(s+9)}.$$

Determine o valor do ganho k_p para o qual os três pólos de malha fechada são reais e iguais.

- d) Um colega traçou o lugar das raízes de um sistema com

$$G(s) = \frac{s+20}{s(s^2 + 24s + 144)},$$

para determinar a faixa de valores de k_p que tornassem o sistema realimentado oscilatório. Usando a função **rlocfind** e selecionando o ponto em que os pólos do sistema (realimentado) passavam a ter parte complexa, obteve:

```
selected_point = -4.7698
```

```
g = 16.3718
p = -14.4939
      -4.7698
      -4.7363
```

O procedimento adotado pelo colega está correto, ou seja, o sistema em questão terá realmente comportamento oscilatório para $k_p \approx 16,37$? Justifique.

- e) Considere o sistema de controle mostrado na figura 5.4 e os controladores:

i) $G_c(s) = k_p$ (Controlador proporcional)

ii) $G_c(s) = k_i / s$ (Controlador integral)

iii) $G_c(s) = k_p + \frac{k_i}{s} = k_{pi} \left(\frac{s+1}{s} \right)$ (Controlador proporcional-integral *simplificado*)

Obtenha o lugar das raízes de cada sistema compensado. Discuta a estabilidade dos sistemas em função do ganho e, quando possível, determine o valor do ganho crítico.

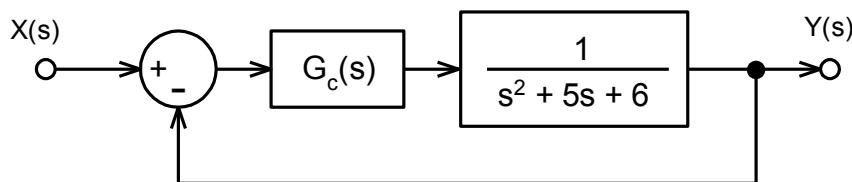


Figura 5.4: Sistema de controle em malha fechada

- f) A função **rlocus** do MATLAB utiliza a estrutura mostrada na figura 5.5.

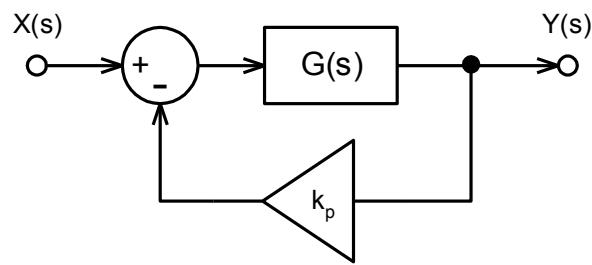


Figura 5.5: Estrutura usada na análise de lugar das raízes

Mostre que, para a análise de lugar das raízes, esta estrutura é equivalente à da figura 5.1.

6. Projeto usando lugar das raízes – introdução

O MATLAB permite o projeto interativo de sistemas de controle por meio de uma ferramenta gráfica chamada *SISO Design Tool*. A abreviatura SISO indica que o projeto é limitado a sistemas com apenas uma entrada e uma saída (*single-input, single-output*). É possível usar como base de projeto o diagrama do lugar das raízes do sistema, o diagrama de Bode ou o diagrama de Nichols. A configuração padrão para o projeto está representada na figura 6.1.

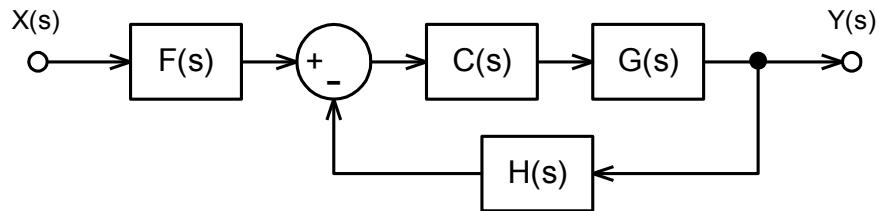


Figura 6.1: Configuração padrão para a ferramenta de projeto SISO

Nesta configuração:

- $F(s)$ é o pré-filtro do sistema;
- $C(s)$ é o compensador (ou controlador);
- $G(s)$ representa a planta ou processo controlado;
- $H(s)$ representa o elemento de medição (sensor).

As funções de transferência do compensador e da planta, geralmente, são importadas do *workspace*, como será mostrado a seguir. Por padrão, adota-se $F(s)=H(s)=1$.

6.1 Exemplo de projeto

Considere o projeto de um controlador proporcional-derivativo (PD) para uma planta com a seguinte função de transferência:

$$G_p(s) = \frac{1.000}{s(s + 125)}.$$

O sistema controlado deve apresentar:

- i) Erro estacionário nulo para entrada em degrau;
- ii) Sobressinal máximo de 10%;
- iii) Tempo de subida menor ou igual a 10 ms.

Resolução: a função de transferência do controlador PD é dada por

$$G_{pd}(s) = k_p + k_d s = k_p(1 + T_d s),$$

sendo $T_d = k_d/k_p$, o *tempo derivativo* do controlador.

Para utilizarmos o diagrama de lugar das raízes no projeto é preciso que exista apenas um parâmetro livre. Assim, incluiremos a contribuição dinâmica do controlador na planta, para um valor fixo de T_d , de modo que o ganho de ramo direto seja apenas k_p . (ver figura 6.2).

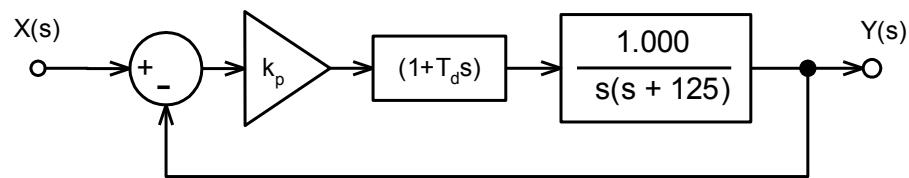
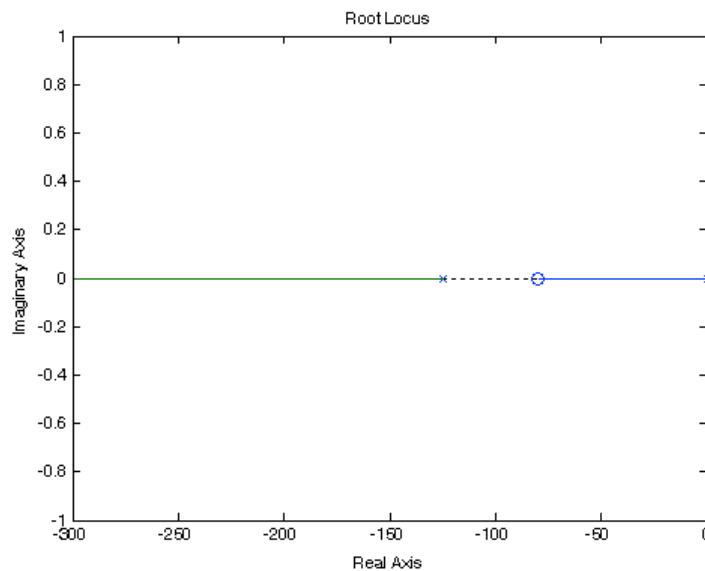


Figura 6.2: Sistema de controle com ganho de ramo direto ajustável

Assim, as funções de transferência necessárias para o uso da ferramenta de projeto são:

$$F(s) = H(s) = 1; \quad C(s) = k_p; \quad G(s) = \frac{1.000(1 + T_d s)}{s(s + 125)}$$

Neste exemplo, o zero do sistema (em $s = -1/T_d$) será posicionado arbitrariamente entre os pólos de malha aberta, com $T_d = 1/80$ (ver figura 6.3).

Figura 6.3: Lugar das raízes do sistema para $T_d = 1/80$

Para executar a ferramenta de projeto com o diagrama de lugar das raízes do sistema usa-se a função ***rltool***. Seqüência de comandos:

```
>> Td = 1/80; ↴
>> n = 1000*[Td 1]; ↴
>> d = conv([1 0],[1 125]); ↴
>> G = tf(n,d); ↴
>> rltool(G); ↴
```

A interface da ferramenta de projeto será mostrada (ver figura 6.4) com os pólos correspondentes ao valor atual de k_p (valor inicial = 1) destacados.

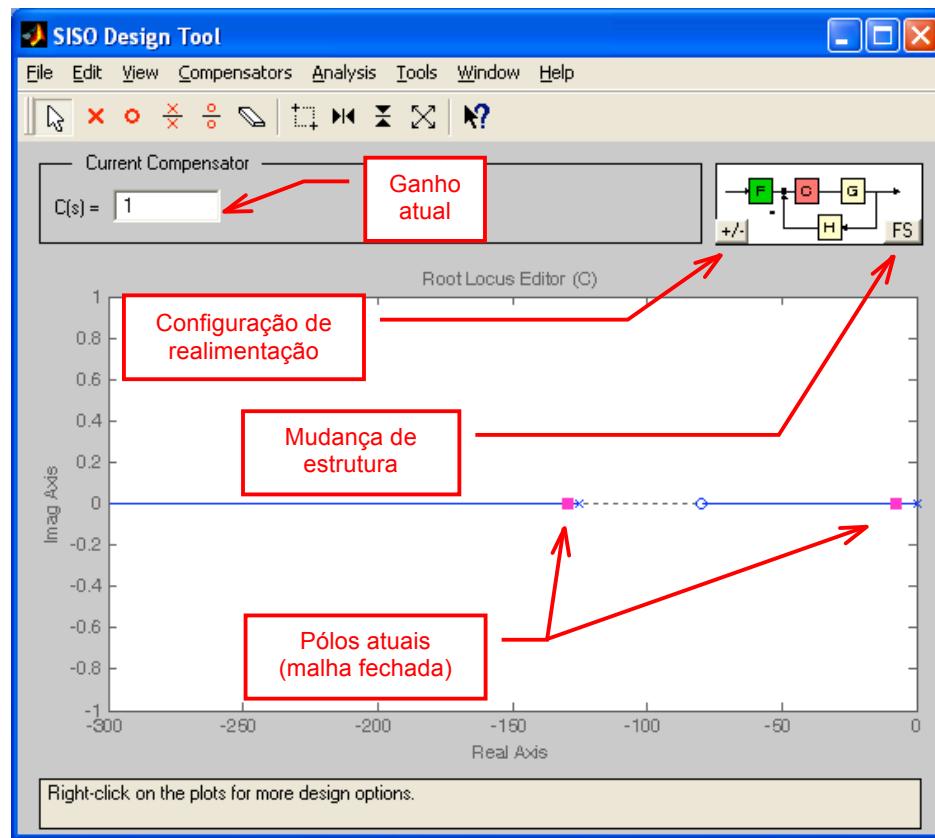


Figura 6.4: Interface da ferramenta de projeto SISO

Para verificar se os critérios de desempenho estão sendo satisfeitos abra uma janela com a resposta ao degrau do sistema controlado, clicando em *Analysis > Response to Step Command*. Inicialmente, são mostrados os gráficos da saída do sistema e do esforço de controle (saída do controlador), conforme a figura 6.5.

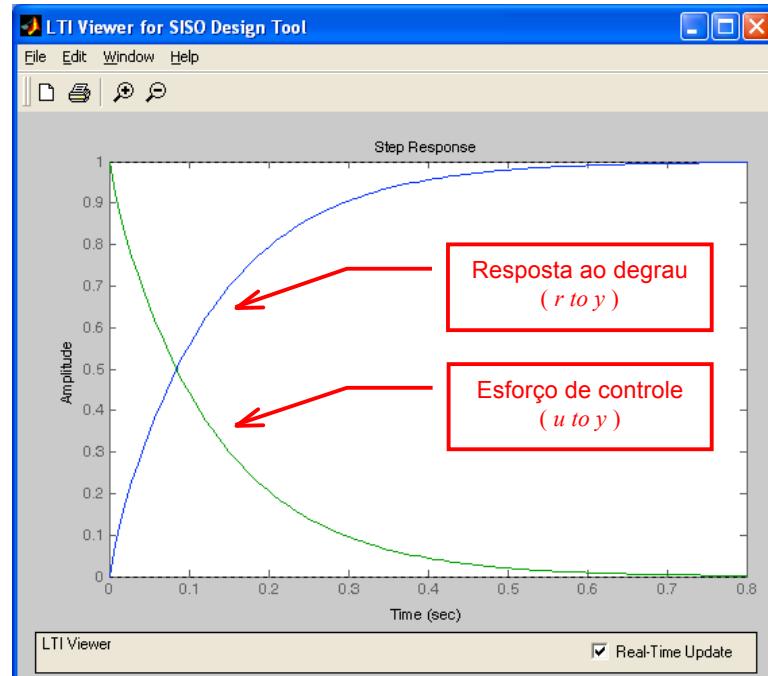


Figura 6.5: Resposta ao degrau do sistema controlado

As curvas que aparecem na janela de resposta podem ser alteradas clicando-se em *Analysis > Other Loop Responses*. No momento, oculte o gráfico do *esforço de controle*, desmarcando a opção "r to u (FCS)" (ver figura 6.6).

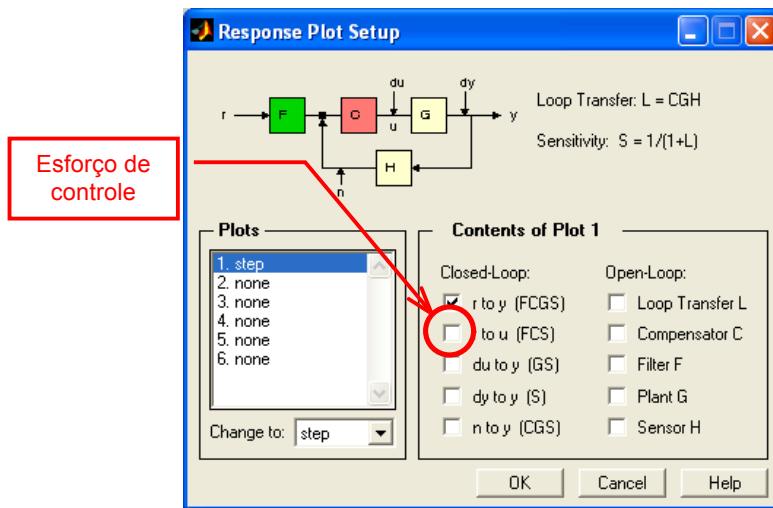


Figura 6.6: Configuração da resposta ao degrau

Clique com o botão direito na janela de resposta ao degrau e selecione *Characteristics > Rise Time*. Para $k_p = 1$ o tempo de subida deve ser de aproximadamente 0,283 s.

Mantenha as janelas da resposta e da ferramenta de projeto lado a lado e altere o ganho k_p arrastando um dos pólos atuais do sistema. O novo valor de ganho também pode ser digitado na caixa de edição "Current Compensator". O desempenho exigido neste projeto deve ser obtido para $k_p \geq 27$. Se for necessário repetir o projeto, por exemplo, com outro valor de T_d , altere a função de transferência $G(s)$ e importe-a para a ferramenta de projeto, clicando em *File > Import* e selecionando a variável G como "função da planta" (ver figura 6.7).

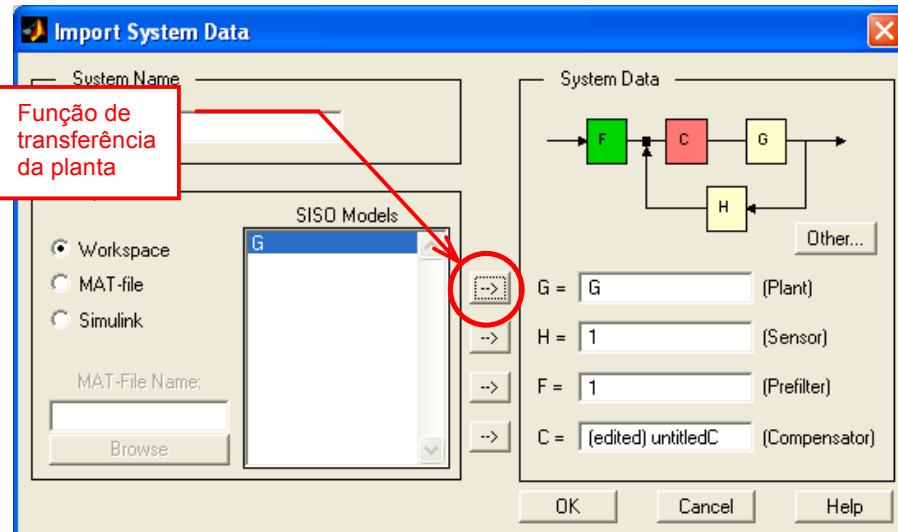


Figura 6.7: Alteração da função de transferência da planta

6.1.1 Atividades

- Repita o projeto com $T_d = 1/200$. Verifique se ainda é possível atender aos critérios de desempenho exigidos e discuta os resultados obtidos.
- Ainda com $T_d = 1/200$, repita o projeto usando um controlador proporcional. Discuta os resultados obtidos.

c) Usando os mesmos critérios de desempenho dos projetos anteriores, projete um controlador proporcional-integral (PI) para a planta:

$$G_p(s) = \frac{1.000}{(s + 125)(s + 500)}.$$

A função de transferência do controlador PI é:

$$G_{pi}(s) = k_p + \frac{k_i}{s} = k_p \left(1 + \frac{1}{T_i s} \right) = k_p \left(\frac{s + 1/T_i}{s} \right),$$

Adote um valor para o tempo integral e determine o valor de k_p que permita atender às especificações de desempenho propostas. Justifique a escolha feita para a posição do zero em relação aos pólos do sistema.

7. Resposta em freqüência – introdução

O estudo da resposta em freqüência de um sistema linear estável visa determinar como sua saída em *regime permanente* varia em função da freqüência ω de um sinal de entrada puramente senoidal. Nestas condições, a saída e todos os sinais internos do sistema são senoidais com a mesma freqüência do sinal de entrada, diferindo apenas em *amplitude* e *fase*.

7.1 Determinação manual da resposta em freqüência

Considere um sistema linear estável definido por uma função de transferência conhecida, $G(s)$, submetido a uma entrada senoidal da forma $x(t) = A \operatorname{sen}(\omega t)$. Neste caso, a saída em regime permanente é da forma:

$$y(t) = A |G(\omega)| \operatorname{sen}(\omega t + \phi).$$

O ganho do sistema, dado por

$$G(j\omega) = G(s) \Big|_{s=j\omega},$$

é um **número complexo** cujo módulo e fase podem ser calculados como

$$|G(j\omega)| = \sqrt{\operatorname{Im}(G(j\omega))^2 + \operatorname{Re}(G(j\omega))^2} \quad \text{e} \quad \phi = \operatorname{arctg}(\operatorname{Im}(G(j\omega)) / \operatorname{Re}(G(j\omega))),$$

sendo $\operatorname{Re}(G(j\omega))$ e $\operatorname{Im}(G(j\omega))$, respectivamente, a parte real e imaginária do ganho $G(j\omega)$.

7.1.1 Atividades

a) Crie um *script* ou função que calcule o valor da amplitude e fase (em graus) do ganho de um sistema definido por

$$G(s) = \frac{10.000}{s^2 + 141,4s + 10.000} \Rightarrow G(j\omega) = \frac{10.000}{(10.000 - \omega^2) + (141,4\omega)j},$$

para entradas senoidais de freqüências: 1, 10, 100, 1.000 e 10.000 rad/s. Use as funções **abs** e **angle** nos cálculos e converta os ângulos de radianos para graus, multiplicando os valores por 180 e dividindo por π . O que se pode concluir à respeito do ganho do sistema?

b) Analise o código listado a seguir.

```
function Ordem1(n,d,wmax)
w = logspace(-2,ceil(log10(wmax)),100);
for k=1:100
    G = n/(d(1)*w(k)*i+d(2));
    Mod(k) = abs(G);
    Fase(k) = angle(G)*180/pi;
end
figure(1); semilogx(w,Mod); title('Modulo');
figure(2); semilogx(w,Fase); title('Fase (graus)');
```

Teste a função para **n=1**, **d=[1 1]** e **wmax = 100**. Explique, usando suas próprias palavras, qual é a utilidade da função.

c) Crie uma nova versão da função **Ordem1** que use acesso vetorizado (ver seção 1.3, pág. 6).

d) Elabore uma função (*RespFreq*) que desenhe os gráficos da resposta em freqüência (módulo e fase do ganho) de um sistema linear da forma:

$$G(s) = \frac{n(s)}{d(s)} = \frac{K}{as^2 + bs + c}.$$

Sintaxe da função

```
[Mod, Fase, w] = RespFreq(n, d, wmax);
```

Dados de entrada

- Função de transferência (polinômios do numerador e denominador);
- Freqüência máxima a ser usada nos cálculos (em rad/s).

Valores de retorno

- Vetores de módulo e fase dos ganhos calculados;
- Vetor de freqüências usadas nos cálculos.

Considerações

- Freqüência mínima = 0,01 rad/s;
- 1.000 valores de freqüência;
- Gráfico de fases em graus.

e) Altere o código de *RespFreq* para trabalhar com funções de transferência genéricas.

7.2 Diagrama de Bode

A representação gráfica do ganho de um sistema (módulo e fase) em função da freqüência do sinal de entrada fornece informações importantes sobre seu comportamento dinâmico. Normalmente os gráficos de módulo e fase usam *escalas logarítmicas* de freqüência, que facilitam a representação de grandes variações de ω . Além disso, é usual representar os valores de ganho em **decibéis** (dB), calculados como:

$$G_{dB} = 20 \log_{10} |G(\omega)|.$$

Os gráficos desenhados sob as especificações anteriores formam o **diagrama de Bode** de um sistema, que pode ser obtido diretamente no MATLAB pelo uso da função *bode*.

7.2.1 Exemplo

A seqüência de comandos a seguir desenha o diagrama de Bode do sistema definido por:

$$G(s) = \frac{10.000}{s^2 + 141,4s + 10.000}.$$

Comandos:

```
>> n = 1e4; ↴
>> d = [1 141.4 n]; ↴
>> G = tf(n,d); ↴
>> bode(G); ↴ % Opção: bode(n,d);
```

O resultado é representado em uma janela gráfica, conforme a figura 7.1.

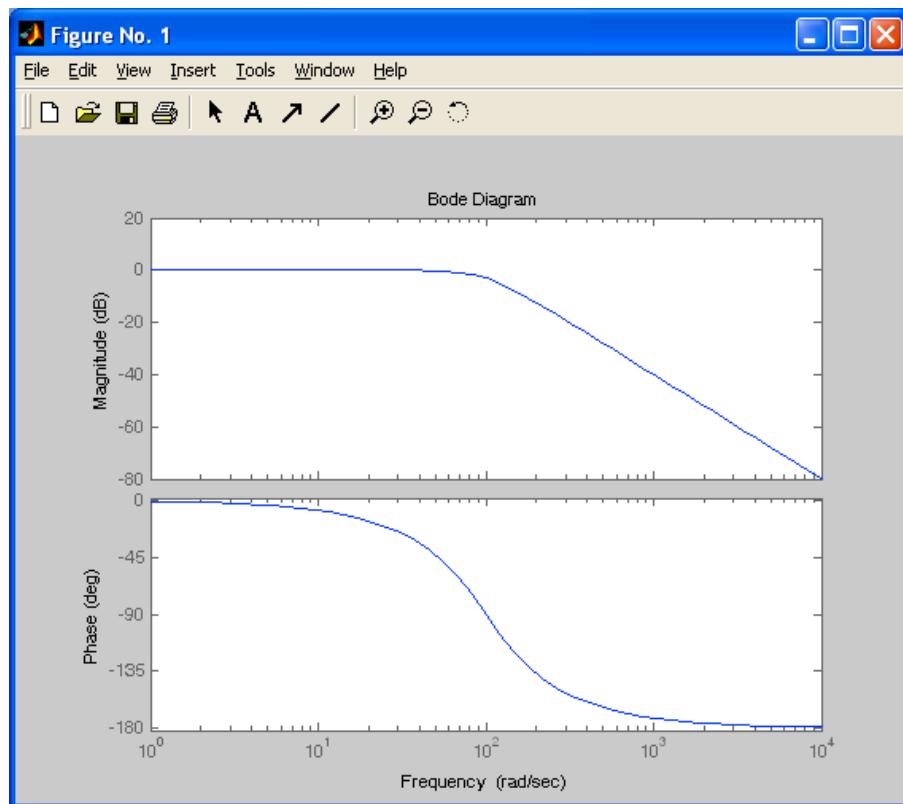


Figura 7.1: Diagrama de Bode

Neste caso, as freqüências usadas na criação dos gráficos são determinadas automaticamente pelo MATLAB. Como outras funções com resultados gráficos, a função **bode** também aceita mais de uma função de transferência como argumento. Neste caso, os diagramas são sobrepostos na mesma janela de figura. Os valores calculados pela função **bode** (ou seja, a resposta em freqüência) podem ser obtidos, sem o traçado dos gráficos, usando-se a sintaxe:

```
[mod,fase,w] = bode(G) % Sendo G = tf(n,d), um sistema linear
```

Nesta forma de uso, os ganhos retornados pela função são *adimensionais*, e não em decibéis. O vetor de freqüências a ser usado nos cálculos pode ser fornecido pelo usuário, como no exemplo a seguir (que aproveita os polinômios do exemplo anterior):

```
>> G = tf(n,d); ↴
>> w = logspace(0,3,200); ↴
>> bode(G,w); ↴
```

7.3 Sistemas de 2^a ordem

Muitos sistemas lineares podem ser representados ou aproximados por funções de transferência de 2^a ordem padrão, da forma:

$$G(s) = K \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}.$$

Desconsiderando-se o ganho K , cujo efeito pode ser incluído posteriormente, pode-se traçar o diagrama de Bode do sistema para uma freqüência normalizada ($u = \omega/\omega_n$) em função da relação de amortecimento ξ (ver figura 7.2).

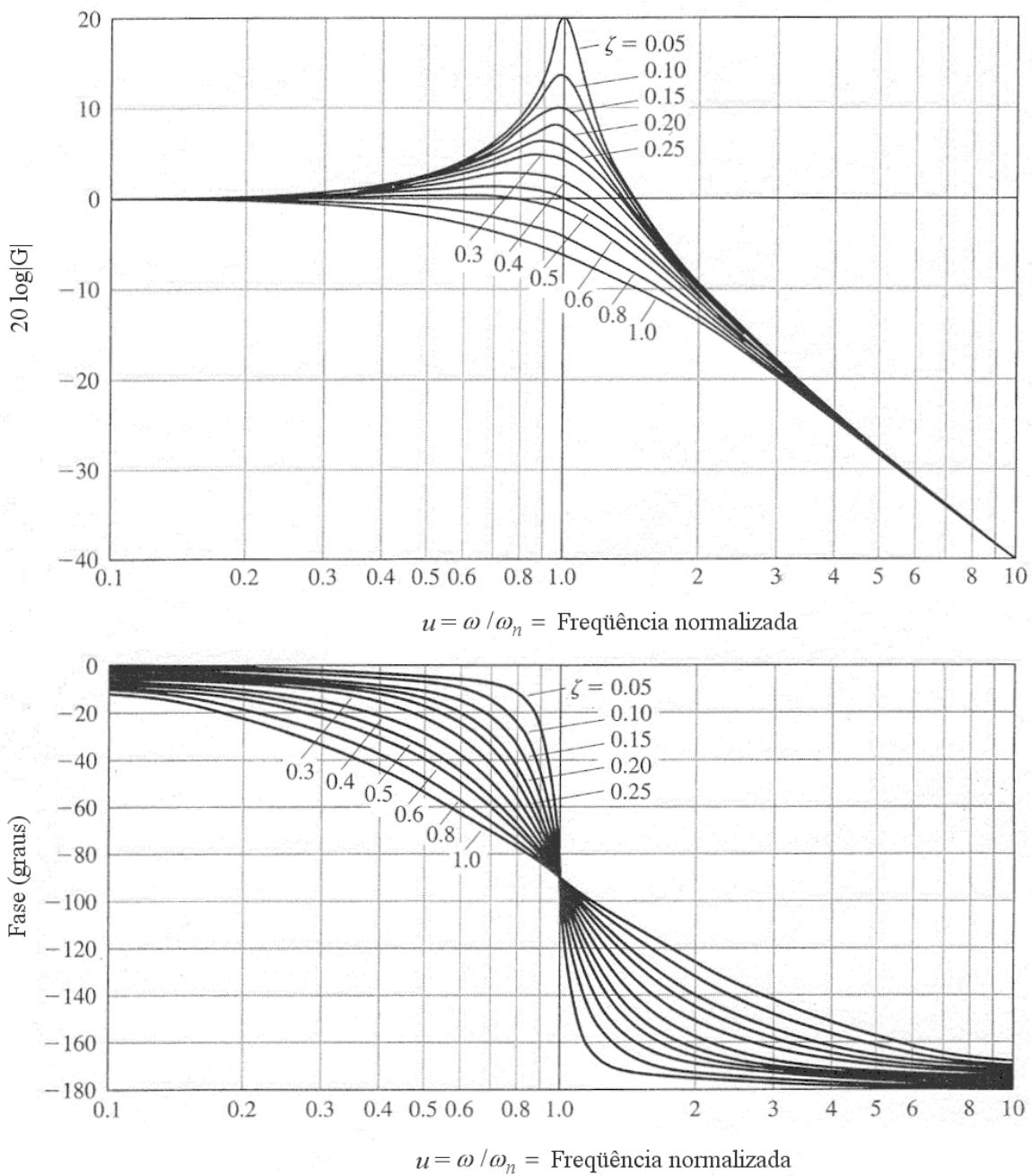


Figura 7.2: Diagrama de Bode para sistemas de 2^a ordem padrão

Em geral, há três parâmetros de interesse na resposta em freqüência de um sistema de 2^a ordem:

- O pico de ressonância (ganho máximo) do sistema, denominado $M_{p\omega}$ (em dB);
- A freqüência de ressonância (ω_r) em que o pico de ressonância ocorre;
- A largura de banda passagem do sistema (ω_B), definida pela freqüência em que o ganho cai 3 dB abaixo de seu valor em baixas freqüências.

Esses parâmetros podem ser usados para estabelecer critérios de desempenho no domínio do tempo, apesar de não existirem formulações exatas que relacionem a resposta em freqüência com a resposta temporal de um sistema.

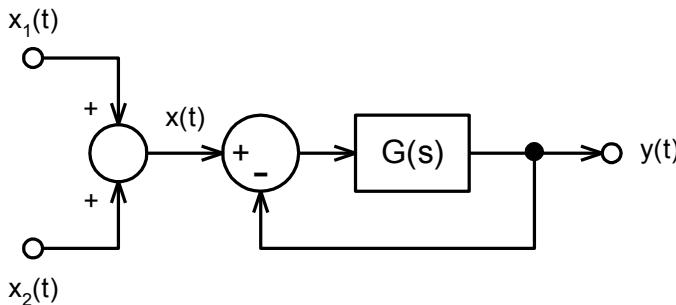
7.3.1 Atividades

a) Determine o pico de ressonância, a freqüência de ressonância e a largura de banda dos sistemas definidos por:

$$i) \quad G_1(s) = \frac{4,494}{s^2 + 1,484s + 4,494}$$

$$ii) \quad G_2(s) = \frac{s + 1}{s^2 + s + 2}$$

b) A saída de um sistema linear excitado por uma entrada que não seja puramente senoidal pode ser obtida pela composição dos efeitos associados a cada *componente espectral* do sinal de entrada. Por exemplo, considere o sistema da figura 7.3.



$$x_1(t) = \text{sen}(5t)$$

$$x_2(t) = 0,5 \text{sen}(50t)$$

$$G(s) = \frac{1.000}{s^3 + 20s^2 + 200s + 1.000}$$

Figura 7.3: Sistema realimentado com duas entradas senoidais

Obtenha a simulação da saída $y(t)$ usando a função **lsim** e verifique a atuação do sistema como um *filtro de freqüências*. Sugestão para o vetor de tempo de simulação: $t = 0: 0.005: 5;$

c) Simule a saída do sistema anterior para cada sinal de entrada, $x_1(t)$ e $x_2(t)$, isoladamente. Verifique a defasagem e atenuação sofridas em cada caso e confira os resultados usando o diagrama de Bode do sistema realimentado.

8. Simulink

O Simulink é um programa que funciona de forma integrada ao MATLAB, usado para modelagem e simulação de sistemas dinâmicos lineares ou não-lineares, em tempo contínuo, tempo discreto ou uma combinação dos dois modos. Os resultados das simulações podem ser visualizados, gravados em variáveis do MATLAB ou em arquivos de dados.

8.1 Exemplo de uso

A criação de modelos no Simulink é feita de forma gráfica pelo posicionamento, interligação e configuração de blocos funcionais. Para ilustrar o uso do Simulink será mostrado como obter a simulação da resposta ao degrau do sistema representado na figura 8.1.

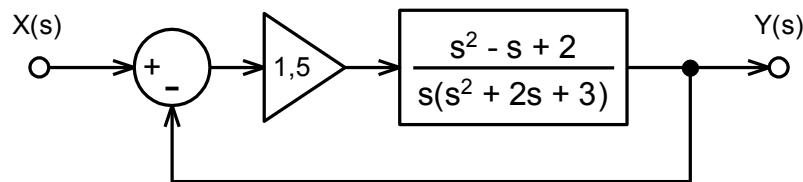


Figura 8.1: Sistema de controle em malha fechada

Inicie o Simulink a partir da linha de comando (digitando **simulink**) ou clicando no ícone do programa na barra de comandos do MATLAB. A janela principal do Simulink será exibida com as bibliotecas de blocos disponíveis para uso.

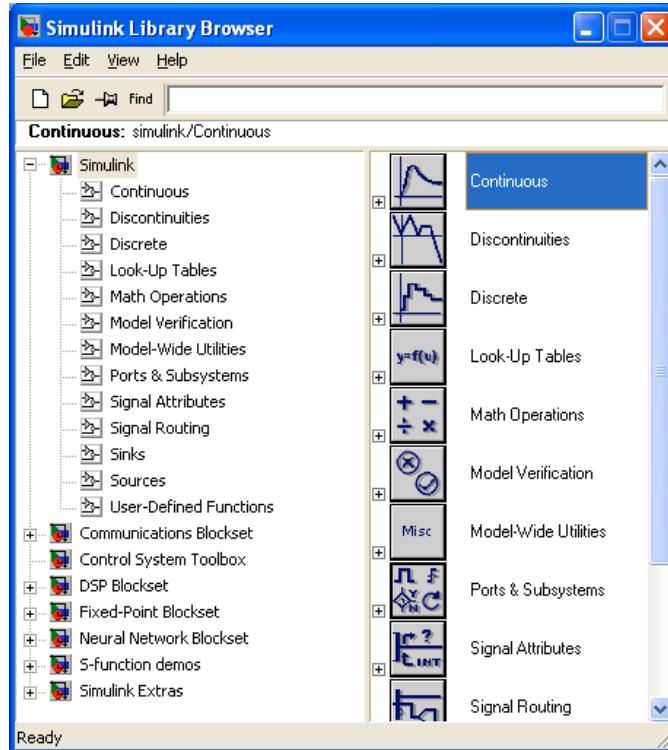


Figura 8.2: Bibliotecas de blocos do Simulink

Uma janela para edição de um novo modelo (*untitled*) deve ser aberta automaticamente. Se isso não acontecer, clique em *File > New > Model* na janela do Simulink.

Os blocos necessários para a simulação do exemplo estão nas bibliotecas **Sources**, **Sinks**, **Continuous** e **Math Operations**. Por exemplo, a figura 8.3 mostra alguns blocos da biblioteca de geradores de sinais.

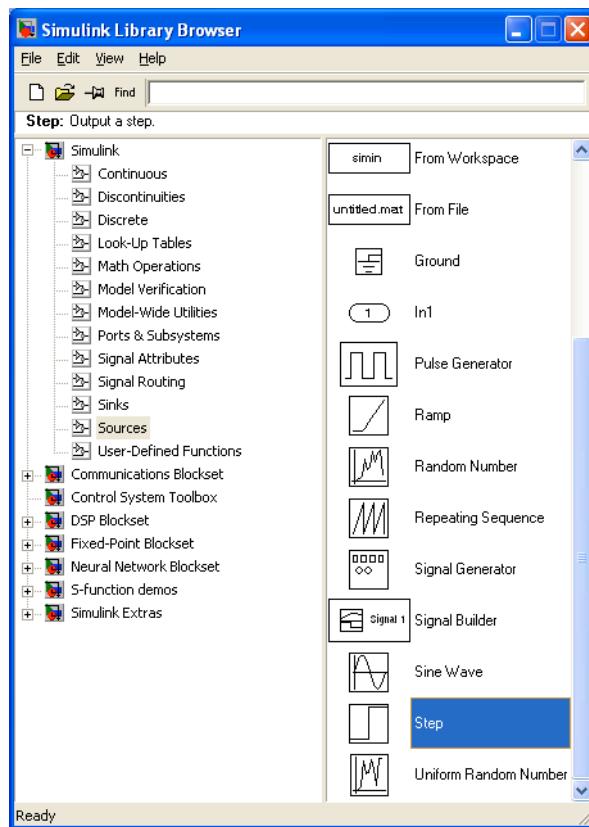


Figura 8.3: Biblioteca de blocos geradores de sinais

Para inserir um gerador de sinal tipo degrau no sistema basta clicar no bloco **Step** e arrastá-lo para a janela do modelo (ver figura 8.4).

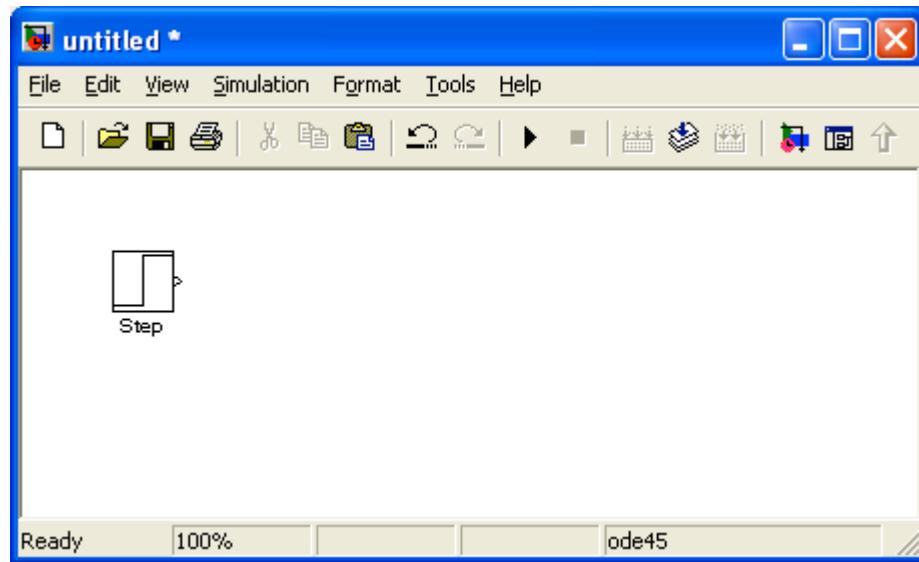


Figura 8.4: Janela do modelo em edição

Em seguida, insira um somador no modelo a partir da biblioteca **Math Operations** e dê um *duplo-clique* no somador para editar suas propriedades. Altere a lista de sinais para '+ -' (sem as aspas) e clique em **OK** (ver figura 8.5).

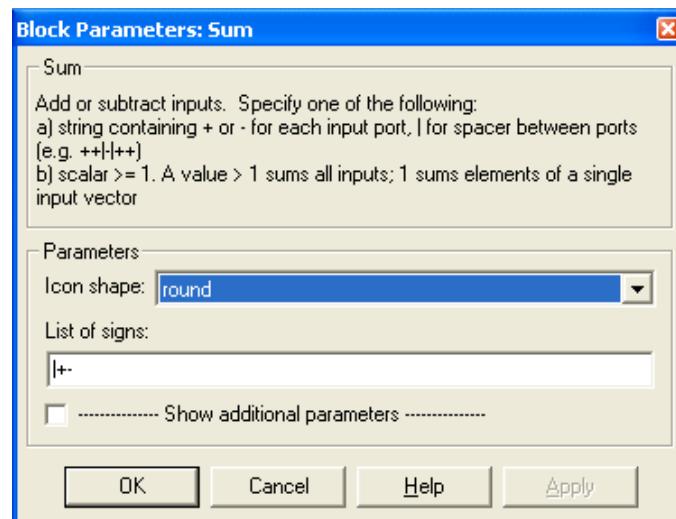


Figura 8.5: Caixa de diálogo com as propriedades do bloco somador

Crie uma ligação entre o gerador e a entrada '+' do somador. Para isso, posicione o *mouse* sobre a saída do gerador até que o cursor mude para a forma de uma cruz. Em seguida, clique e arraste o cursor até a entrada do somador – o cursor deve mudar para uma cruz dupla – para completar a ligação (ver figura 8.6).

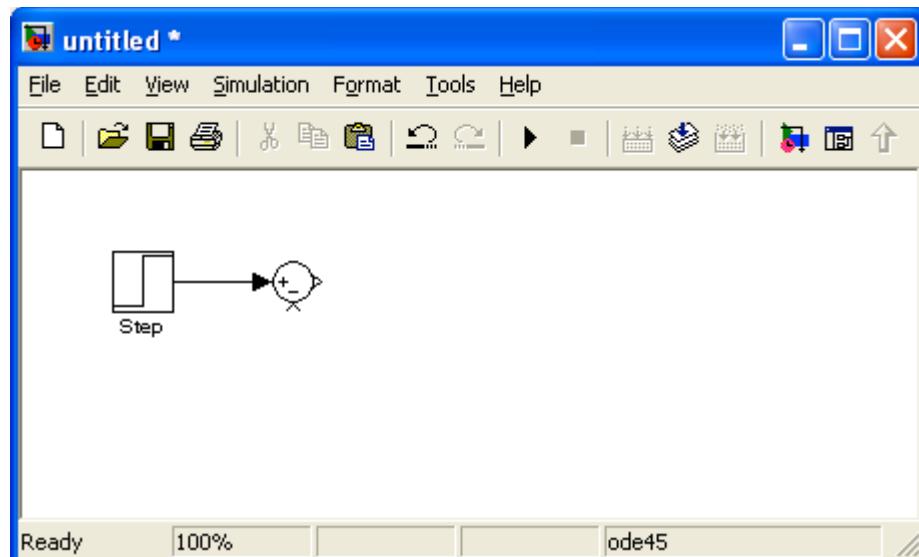


Figura 8.6: Ligação entre dois blocos

Insira os outros blocos no sistema (navegue pelas bibliotecas **Continuous** e **Sinks**) e complete as ligações até obter um modelo semelhante ao da figura 8.7. As técnicas válidas na maioria dos programas para ambiente Windows, como seleção múltipla de objetos, redimensionamento e movimentação também funcionam no Simulink. Operações de rotação e espelhamento estão disponíveis em menus acionados pelo clique do botão direito do *mouse* sobre os blocos.

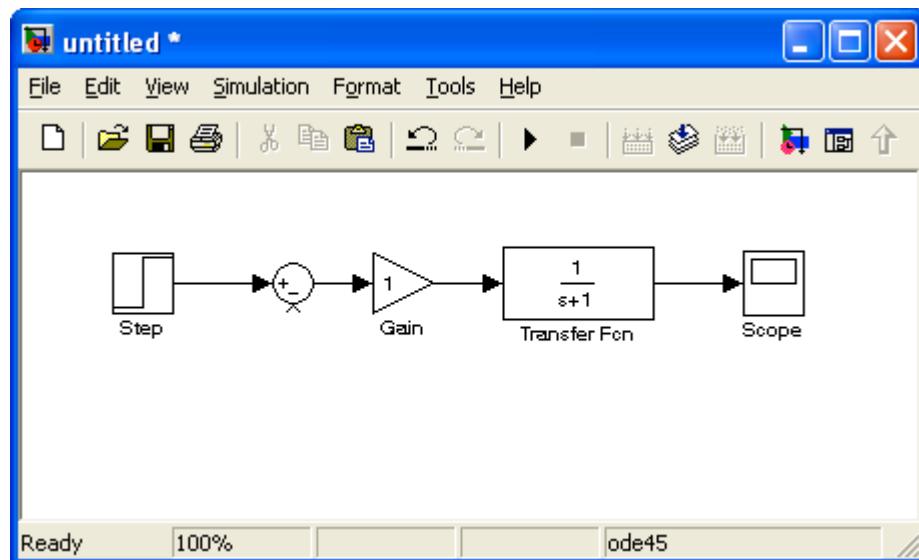


Figura 8.7: Modelo de simulação parcialmente construído

Crie a ligação de realimentação ligando a entrada do somador com a linha de ligação entre o bloco da função de transferência e o osciloscópio (*scope*) – podem ser necessárias algumas tentativas. Outra maneira de se fazer esta ligação é posicionar o cursor sobre a linha de ligação, pressionar a tecla **Ctrl** e arrastar o mouse até a entrada do somador. O processo de clicar e arrastar pode ser feito em etapas para que as linhas de ligação sejam posicionadas de forma conveniente.

Para completar o modelo, edite o valor do ganho do bloco amplificador e altere a função de transferência, digitando os polinômios do numerador ([1 -1 2]) e do denominador ([1 2 3 0]). Se desejar, edite os nomes dos blocos para descrever a função de cada um. Finalmente, grave o modelo como **sistema_01.mdl** antes de gerar a simulação. O resultado final deve ser semelhante ao mostrado na figura 8.8.

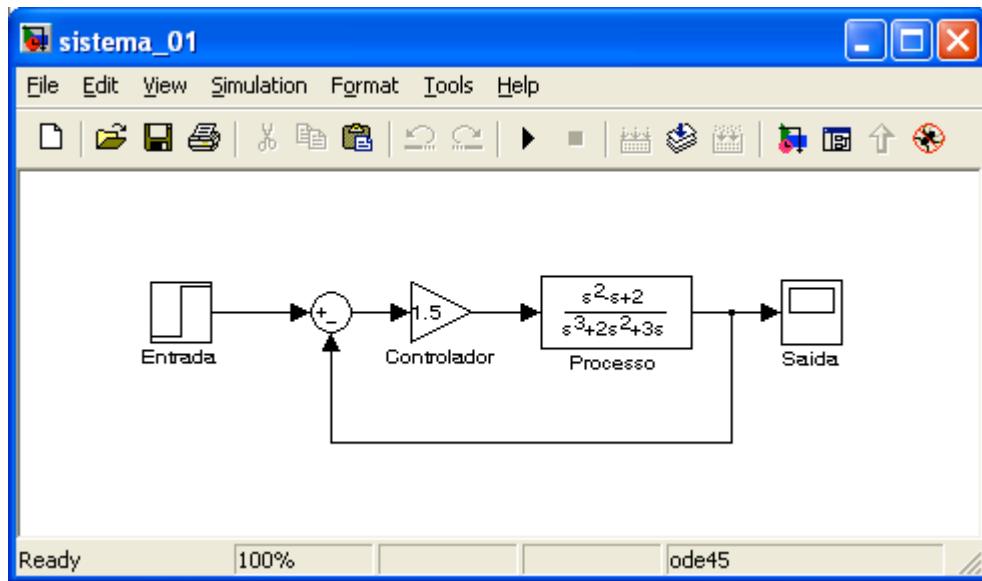


Figura 8.8: Modelo de simulação completo

8.1.1 Configurando os parâmetros de simulação

Clique em *Simulation > Parameters* para acessar as opções de simulação (ver figura 8.9).

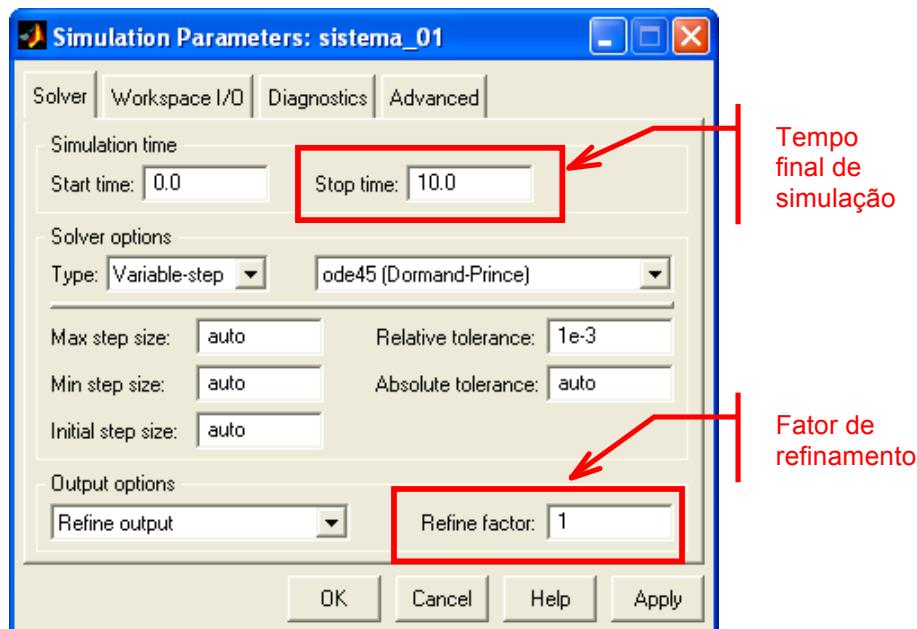


Figura 8.9: Parâmetros de simulação

Os botões da caixa de diálogo têm as seguintes funções:

- **Apply** – aplica os parâmetros de simulação atuais ao sistema, mesmo que exista uma simulação em curso, mantendo a caixa de diálogo aberta;
- **Cancel** – retorna os parâmetros da simulação aos últimos valores ajustados;
- **Help** – abre uma janela de ajuda para os comandos da caixa de diálogo;
- **OK** – aplica os parâmetros de simulação ao sistema e fecha a caixa de diálogo.

Algumas opções de interesse da aba **Solver**:

- Simulation time** – Intervalo de tempo em que a simulação é feita. O tempo que a simulação leva para ser completada depende de fatores como a complexidade do modelo e capacidade de processamento do computador.
- Solvers** – A simulação de um sistema dinâmico envolve a integração numérica de sistemas de equações diferenciais ordinárias. Para isso, o Simulink oferece vários métodos de resolução com passos de integração fixos ou variáveis. Normalmente, o algoritmo de passo variável **ode45**, fundamentado nos métodos de Runge-Kutta, fornece bons resultados.
- Step sizes** – É possível controlar os valores dos passos de integração dos algoritmos de passo variável, como **ode45**. Como regra geral, pode-se deixar o controle desses valores a cargo do Simulink em uma primeira simulação e alterá-los caso os resultados obtidos não sejam adequados. De preferência, deve-se manter valores iguais para os valores de passo máximo e inicial. Esta regra prática funciona de forma conveniente para a maioria dos problemas de simulação embora não seja a única nem a mais adequada para todos os casos. Em muitas situações é possível melhorar os resultados de uma simulação ajustando-se o fator de refinamento da simulação, como será discutido adiante.
- Error tolerances** – Os algoritmos de resolução usam técnicas de controle de erro a cada passo de simulação. Os valores estimados dos erros são comparados com um erro aceitável, definido pelos valores de tolerância relativa e absoluta, indicados na caixa de diálogo. Os algoritmos de passo variável reduzem o passo de integração automaticamente se o erro for maior que o aceitável. Em geral não é preciso alterar estes parâmetros.
- Output options** – Permite o controle dos instantes de tempo em que serão gerados os resultados da simulação. A opção mais útil é a do controle do fator de refinamento (*Refine factor*) que permite obter um número adicional de pontos de simulação entre aqueles que o algoritmo usaria normalmente. Por exemplo, se o fator de refinamento for definido como 5, cada passo de integração (de tamanho variável) será dividido em 5

subintervalos. Na prática, é mais simples e eficiente (do ponto de vista computacional) melhorar os resultados de uma simulação aumentando o fator de refinamento do que reduzindo o tamanho do passo de integração⁹.

8.1.2 Simulando

Ajuste o tempo de simulação para 80 segundos, alterando o valor de *Stop Time* (80). Simule a resposta do modelo, clicando em *Simulation > Start* ou no ícone em forma de seta na barra de ferramentas do Simulink. O programa avisa que a simulação terminou emitindo um *beep* e exibindo a palavra "*Ready*" na parte inferior da janela do modelo. Dê um duplo clique no bloco do osciloscópio (*Scope*) para ver a simulação do sinal de saída. O resultado deve ser como mostrado na figura 8.10.

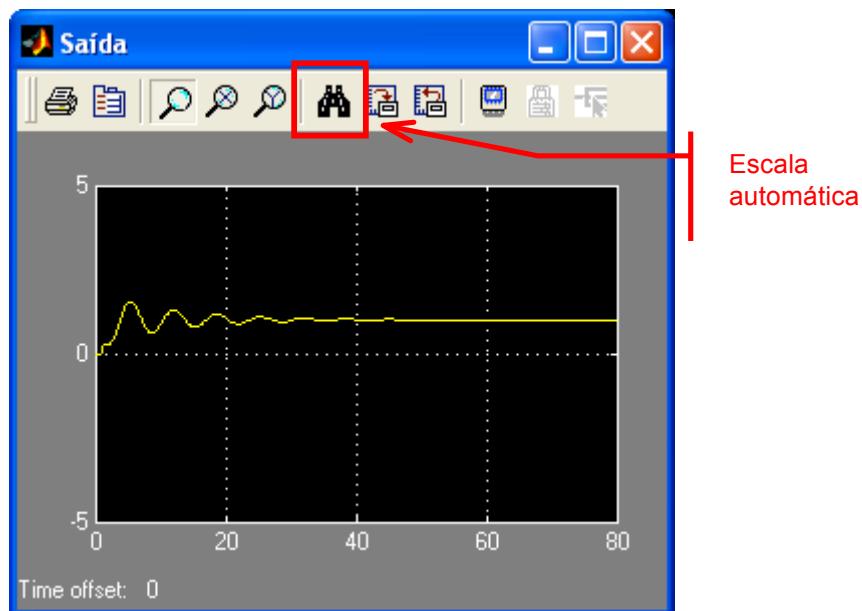


Figura 8.10: Resultado da simulação

É possível alterar as escalas dos eixos a partir das opções de configuração do bloco (clicando com o botão direito do *mouse* em algum ponto do gráfico) mas, geralmente, basta clicar no botão de escala automática (ver figura 8.10), indicado por um binóculo. O resultado final (ajuste de escala automática) é mostrado na figura 8.11. Se o resultado parecer pouco preciso, aumente o fator de refinamento (3 ou 5 costumam ser valores adequados) e simule novamente. Como padrão, o Simulink armazena o vetor de tempo usado na simulação em variável do *workspace* chamada **tout**. A criação desta variável, incluindo seu nome, podem ser ajustados na aba **Workspace I/O** da caixa de diálogo dos parâmetros de simulação.

⁹ Por questões de estabilidade numérica, o fator de refinamento não deve ser aumentado indefinidamente.

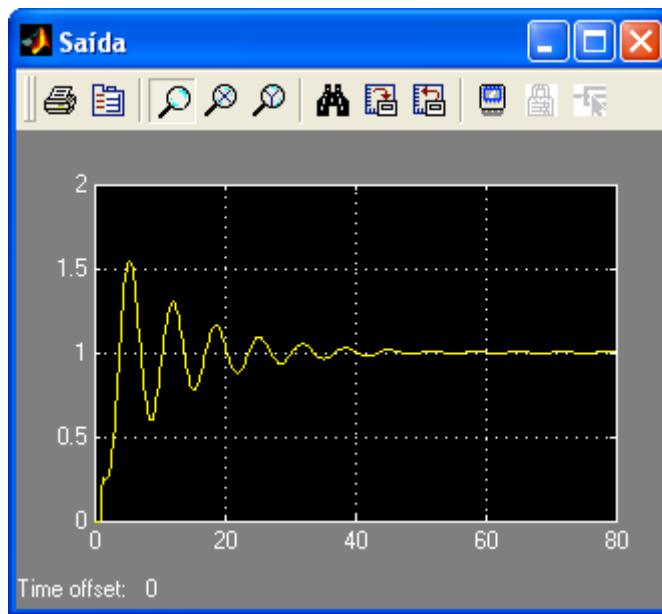


Figura 8.11: Resultado da simulação apóis ajuste de escala

8.2 Atividades

Atenção: Nas simulações pedidas a seguir use fator de refinamento = 5 e ajuste o tempo de simulação para valores convenientes.

- a) Simule o comportamento do sistema da figura 8.12 para uma entrada senoidal de 5 rad/s.

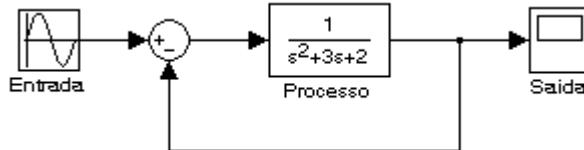


Figura 8.12: Sistema da atividade 8.2 a)

Determine o valor aproximado do ganho do sistema, analisando sua saída em regime permanente. Em seguida, calcule o valor exato do ganho fazendo $s = j5$ na função de transferência de *malha fechada* e compare os resultados.

- b) Determine o valor aproximado do módulo do ganho do sistema da figura 8.13 para sinais senoidais de 3 rad/s. Repita para sinais de 5 e 10 rad/s. O MATLAB pode emitir alguns avisos de "loop algébrico" porque a função de transferência do sistema realimentado possui numerador e denominador com mesmo grau.

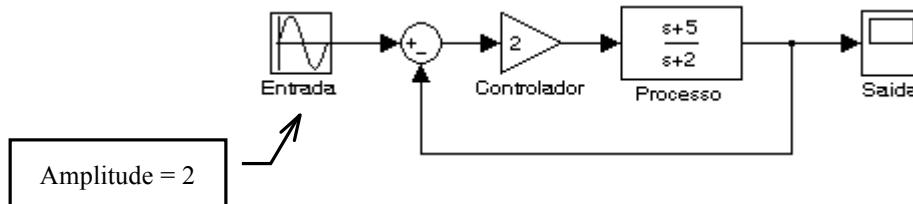


Figura 8.13: Sistema da atividade 8.2 b)

- c) Obtenha o valor aproximado do ganho do sistema da figura 8.14 para um sinal de entrada senoidal de 5 rad/s.

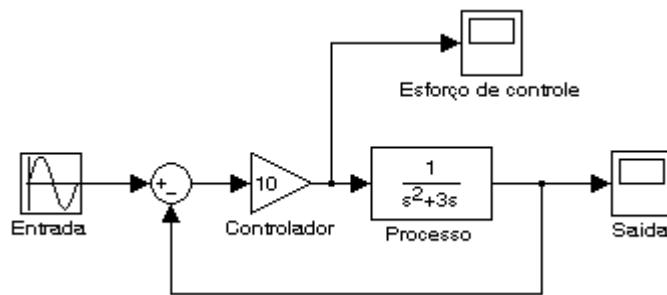


Figura 8.14: Sistema da atividade 8.2 c)

Em seguida, considere que a saída do controlador sature em **±8,5**. Simule o sistema nesta nova condição e verifique as alterações ocorridas em sua saída. Represente a saturação do controlador usando o bloco *Saturation* da biblioteca **Discontinuities** (ver figura 8.15).

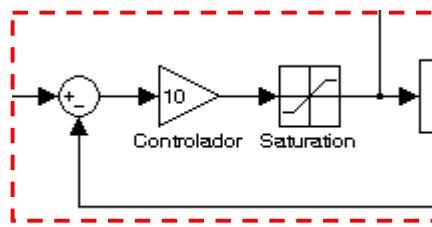


Figura 8.15: Ponto de inserção do bloco de saturação

- d) Investigue a atuação dos demais blocos da biblioteca **Discontinuities**.
- e) Simule o comportamento do sistema mostrado na figura 8.16, analisando a saída do sistema em relação ao sinal de entrada, após o misturador. Descreva a atuação do sistema. Sugestão para o tempo total de simulação: 5 s.

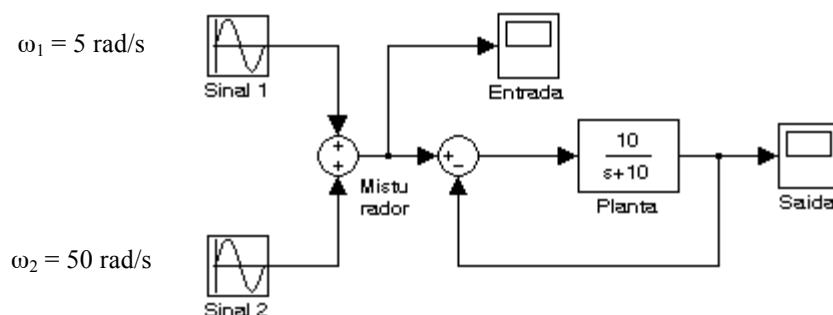


Figura 8.16: Sistema da atividade 8.2 e)

- f) Repita a simulação anterior, substituindo a planta original por:

$$i) \quad G_1(s) = \frac{100}{s^2 + 14s + 100}$$

$$ii) \quad G_2(s) = \frac{1.000}{s^3 + 20s^2 + 200s + 1.000}$$

Verifique as mudanças ocorridas no desempenho do sistema de acordo com a ordem do sistema que representa a planta. Descreva a atuação do sistema.

- g) É possível enviar os resultados de uma simulação para o *workspace* com o bloco **To Workspace** da biblioteca **Sinks**: o nome da variável criada no *workspace* é definido pelas propriedades do bloco. Simule o comportamento do sistema da figura 8.17 durante 10 segundos, ajustando o gerador de pulsos para um

período de 2 segundos. Na janela de comandos do MATLAB, trace o gráfico do vetor **saida** em função de **tout**.

Importante: configure o bloco **To Workspace** para gerar valores de saída no formato *array* (o formato padrão é *Structure*).

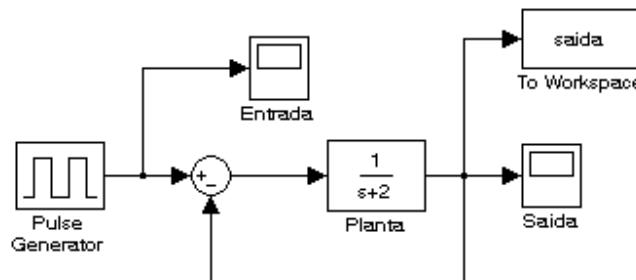


Figura 8.17: Sistema da atividade 8.2 g)

h) É possível usar variáveis do *workspace* como entradas para sistemas do Simulink, usando o bloco **From Workspace** da biblioteca **Sources**. Como exemplo, simule por 10 segundos o comportamento do sistema da figura 8.18 para a entrada $u(t) = 0,75 \sin(5t)$. Para isso, crie uma matriz de estímulos, **VU**, com os tempos de simulação na primeira coluna e os valores da função $u(t)$ na segunda.

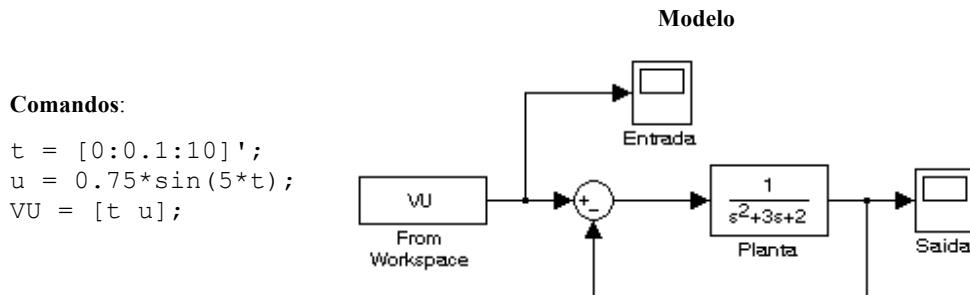


Figura 8.18: Sistema da atividade 8.2 h)

Observação: o bloco **From Workspace** deve estar associado à variável correta (**VU**).

8.3 Subsistemas

Modelos complexos podem ser simplificados pela criação de subsistemas, que são conjuntos de blocos relacionados logicamente. A maneira mais simples de se criar um subsistema é pelo agrupamento de blocos já existentes em um modelo. Também é possível adicionar um bloco **subsystem** vazio a um modelo e editá-lo.

8.3.1 Exemplo – controlador PI

Crie um modelo no Simulink, de acordo com a figura 8.19.

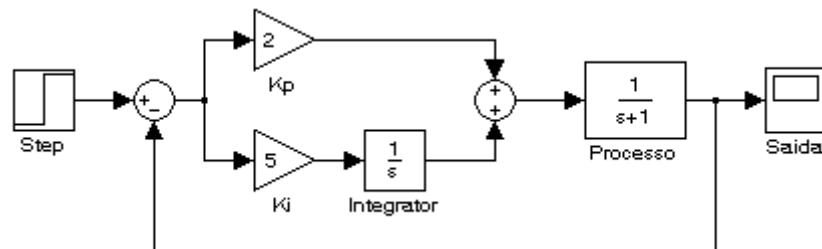


Figura 8.19: Sistema de controle com controlador PI

Usando o *mouse*, selecione os blocos do controlador PI (ver figura 8.20).

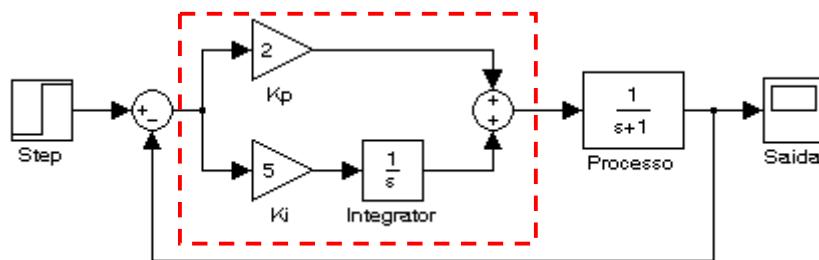


Figura 8.20: Seleção dos blocos do controlador PI

Clique em *Edit > Create Subsystem*. Os blocos selecionados serão substituídos por um bloco chamado *Subsystem*, com uma entrada (**In1**) e uma saída (**Out1**). Altere o nome do bloco para "Controlador PI", conforme a figura 8.21.

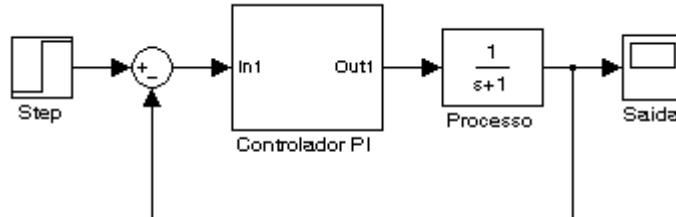


Figura 8.21: Modelo de simulação com subsistema

Dê um *duplo-clique* no subsistema para editá-lo, modificar os nomes das entradas e saídas ou mesmo para gravá-lo como um modelo independente do Simulink (ver figura 8.22).

A entrada e a saída do subsistema são representadas por blocos da biblioteca **Ports & Subsystems** e podem ter seus nomes ocultados individualmente pela opção **Format - Hide Name**, do menu de contexto acionado pelo botão direito do *mouse*. Se esta operação for feita no subsistema, fora de sua janela de edição, todos os nomes serão escondidos e o subsistema será representado por um ícone padrão.

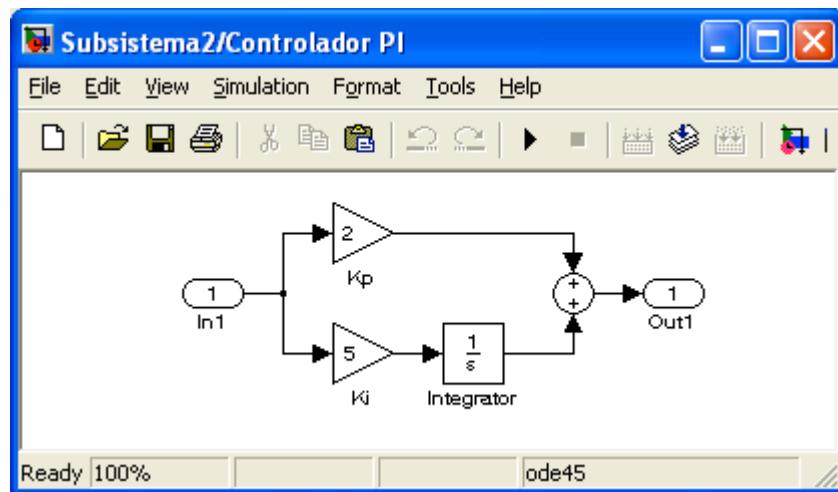


Figura 8.22: Janela de edição do subsistema

Existem ainda ferramentas avançadas, como as *máscaras*, que permitem a personalização da aparência dos subsistemas e a criação de caixas de diálogo (ver exemplo na figura 8.23).

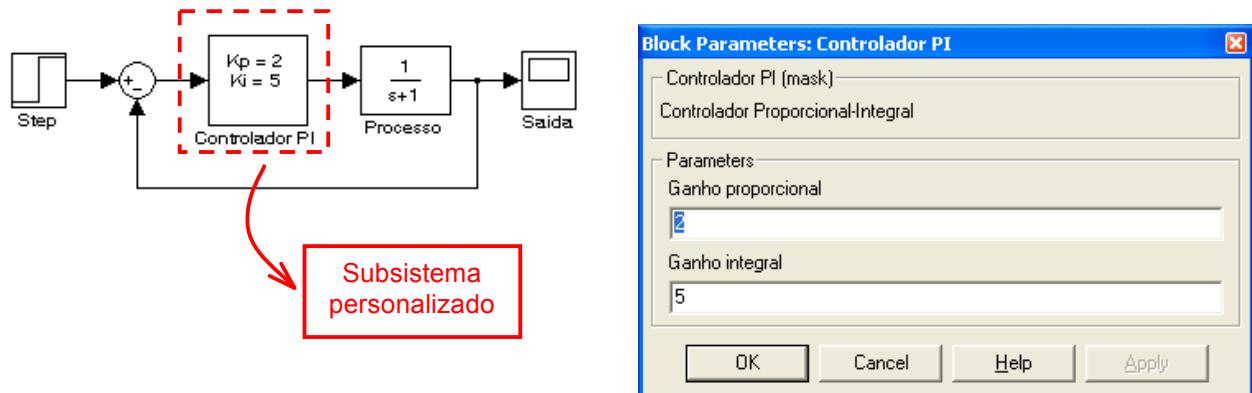


Figura 8.23: Subsistema personalizado e caixa de diálogo de propriedades

Se desejar, consulte a ajuda do MATLAB para aprender a trabalhar com estes recursos.
