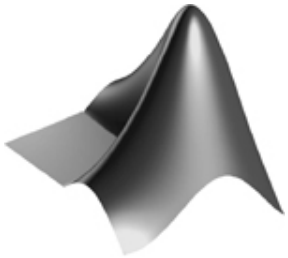
Universidade Federal Fluminense
Escola de Engenharia
Curso de Engenharia de Telecomunicações
Programa de Educação Tutorial
Grupo PET-Tele

Apostila de Introdução ao Octave/MATLAB®

(Versão: A2011M11D01)



Niterói - RJ
Novembro / 2011

Prefácio

Tendo em vista as diretrizes do MEC em pesquisa, ensino e extensão, o Programa de Educação Tutorial (PET) do curso de Engenharia de Telecomunicações da Universidade Federal Fluminense (UFF) desenvolveu um projeto de elaboração de apostilas e cursos voltados para a graduação. O intuito desse trabalho é auxiliar alunos no aprendizado de temas importantes para sua formação, que não estão presentes em nenhuma das disciplinas, e, além disso, servir de material didático para cursos de capacitação ministrados pelos bolsistas do Programa aos corpos discente e docente da graduação.

Abaixo segue a lista de apostilas preparadas nesse projeto.

HTML - linguagem de programação para hipertextos, empregada, principalmente, na construção de páginas da Internet (*webpages*).

LaTeX - sistema de edição de texto largamente utilizado em meios acadêmicos e científicos, bem como por editoras nacionais e internacionais.

Linux - introdução ao sistema operacional LINUX.

Linguagem C - linguagem de programação amplamente utilizada em problemas de engenharia e computação.

MATLAB - ambiente de simulação matemática, utilizado em diversas áreas profissionais.

Spice - ambiente de simulação de circuitos elétricos (analógicos e digitais), utilizado em projeto de circuitos discretos e integrados.

Este documento destina-se a introduzir o usuário ao ambiente do MATLAB®.

Sabendo-se da imensa quantidade de funções e da existência de um acervo de documentação próprio incluído no programa, esta apostila não pretende esgotar o tema como um manual. Ela destina-se a introduzir, de forma sucinta, o usuário ao ambiente do programa, mostrando e exemplificando as ferramentas básicas de utilização do mesmo. Para maiores informações e um estudo mais aprofundado, consulte as referências bibliográficas no fim do documento.

É muito importante, para um melhor aprendizado, que o leitor esteja em frente a um computador com o *software* instalado. À medida que se lê a apostila, comandos e operações devem ser testados no programa.

Interessa ainda mencionar que serão destacados, ao longo das páginas, comandos e funções úteis às disciplinas do curso de Engenharia de Telecomunicações, organizadas em seções relativas a suas aplicações.

Versão atual: Carina Ribeiro Barbio Corrêa

Últimas atualizações: Mariana da Costa Santos
Isabella França e França
Beatriz Costa Ribeiro
Vitor de Souza Lima
Alexandre Santos de la Vega

Sumário

Capítulo 1

Introdução

1.1 Informações iniciais

O MATLAB® (abreviatura de *MATrix LABoratory* - Laboratório de Matrizes) é um *software* de simulação matemática que realiza operações matriciais, constrói gráficos em duas ou três dimensões, auxilia no processamento de sinais, além de manipular outras funções especializadas.

Ele trabalha com uma linguagem de programação de alto nível, em um ambiente interativo, para o desenvolvimento de algoritmos, análise e visualização de dados e computação numérica. Próprio para as áreas técnica e científica, o *software* tem funções de tratamento numérico de alto desempenho, capazes de resolver problemas computacionais técnicos de forma mais eficiente do que as tradicionais linguagens de programação.

Além do ambiente interativo, outra facilidade do MATLAB® é a possibilidade de execução de arquivos texto, contendo uma sequência de instruções definidas pelo usuário. Esses arquivos texto, que têm extensão '.m', podem ser criados e editados dentro ou fora do seu ambiente.

1.2 Janelas

Através de objetos gráficos denominados janelas, o usuário opera as funcionalidades do programa de forma interativa.

A janela principal do MATLAB® chama-se *Command Window* (Janela de Comando), onde os dados e instruções são digitados no *prompt* '>>' pelo usuário e, após a tecla **Enter** ser pressionada, o programa os processa imediatamente e expõe na tela o resultado. Os comandos digitados são armazenados em um *buffer* de comandos, no qual pode-se navegar usando as teclas *seta-para-cima* '↑' e *seta-para-baixo* '↓'. Além disso, teclando-se o texto 'str', por exemplo, e usando-se as setas '↑' e '↓', navega-se por todos os comandos iniciados com o texto 'str'. A tecla **Esc** limpa o que estiver escrito na linha do comando.

Comandos terminados com ponto-e-vírgula não exibem as variáveis de resposta na tela. O uso do ponto-e-vírgula é útil quando a impressão do resultado na tela não interessa, ou quando a impressão é muito extensa como, por exemplo, para uma matriz 1000×1000. Deve-se ressaltar que, apesar da impressão ser suspensa, o comando é executado pelo programa.

Vários comandos podem ser digitados na mesma linha, desde que estejam separados

por vírgula ou ponto-e-vírgula. Comandos muito longos para uma linha podem ser interrompidos por três pontos '...' e continuados na linha seguinte. O exemplo a seguir é ilustrativo quanto a esses detalhes.

```
>> a=1, b=...
2; c=a+b

a =
     1
c =
     3
```

É importante apontar que a Janela de Comando normalmente é usada para testes de comandos e funções ou simples operações. Quando se deseja implementar algum programa, projeto ou trabalho, utiliza-se o *M-File Editor*. Neste editor, cria-se um arquivo texto '.m' com os comandos desejados. Para abrir um novo arquivo-M, clique em *File* > *New* > *M-file* ou simplesmente digite o comando **edit**. Após escrever o programa, pode-se executá-lo pela tecla de atalho *F5*.

Além da Janela de Comando e do Editor de Arquivo-M, há ainda as janelas *Help*, *Command History*, *Current Directory* e *Workspace*, que estão respectivamente relacionadas com ajuda, histórico dos últimos comandos digitados, diretório corrente do programa e o espaço de trabalho onde se visualizam dados e variáveis. Essas janelas podem ser mantidas fechadas ou abertas, dependendo da necessidade ou gosto do usuário. Essas outras janelas não são detalhadas nesse documento.

1.3 Ajuda

Através do comando **help**, o usuário pode consultar a ajuda do MATLAB®. Escrevendo-se **help** e o nome da função, é mostrado um pequeno resumo da função (normalmente de uma linha) seguido de uma descrição mais detalhada da mesma.

```
>> help ones
ONES   Ones array.
    ONES(N) is an N-by-N matrix of ones.
    ONES(M,N) or ONES([M,N]) is an M-by-N matrix of ones.
    ONES(M,N,P,...) is an M-by-N-by-P-by-... array of ones.
    ONES(SIZE(A)) is the same size as A and all ones.
    See also ZEROS.
```

É possível, ainda, procurar por funções cujos resumos contenham determinada palavra-chave. Para isto, basta digitar o comando **lookfor** seguido da palavra desejada. Caso se queira interromper a busca, deve-se teclar *CTRL+C*.

```
>> lookfor ones
ONES   Ones array. SPONES Replace nonzero sparse matrix elements
with ones. SLUPDATE Replace blocks from a previous release with
newer ones. FPUPDATE Replace blocks from a previous release with
newer ones. FPUPDT Replace older 1.x blocks with newer 1.1 ones.
BUFFERM buffer zones for vector data. ONEM  Constructs a regular
matrix map of all ones. UPDATEXPCENV Updates the new Environment
settings to become the current ones. DRAMADAH Matrix of zeros and
ones with large determinant or inverse.
```

1.4 Bibliotecas do MATLAB®

O MATLAB® apresenta uma série de comandos, operadores e funções primitivas, organizadas por categorias, assim como rotinas específicas de diversas áreas da engenharia, organizadas em bibliotecas denominadas *Toolboxes*. Estas categorias e bibliotecas são designadas por tópicos primários. Para a visualização de todos estes tópicos, basta digitar o comando **help**.

Alguns desses tópicos primários são:

- *general* - comandos gerais
- *ops* - operadores e caracteres especiais
- *control* - biblioteca de sistemas de controles
- *signal* - biblioteca de processamento de sinais
- *optim* - biblioteca de otimização

Outras bibliotecas abrangem campos variados como aquisição e processamento de imagem, bio-informática, telecomunicações, lógica difusa e realidade virtual.

Para se aprofundar mais em qualquer biblioteca, basta digitar o comando **help tópico**.

Capítulo 2

Variáveis

2.1 Declaração

Dado que o programa oferece um ambiente de execução de uma linguagem interpretada, uma característica conveniente do MATLAB® é que as variáveis não precisam ser dimensionadas antes do uso, pois são geradas automaticamente ao serem utilizadas. Para criar e/ou armazenar informações em variáveis definidas pelo usuário, basta digitar o nome da variável seguido do sinal de igual '=' e da expressão desejada. Na escolha dos nomes das variáveis, devem ser obedecidos os seguintes critérios:

- os caracteres podem ser alfanuméricos (letras e números), desde que iniciados por letras;
- letras maiúsculas e minúsculas definem nomes diferentes (linguagem *case sensitive*);
- o caracter '_' (*underscore*,*underline* ou sublinhado) pode ser usado no meio do nome;
- são permitidos nomes com, no máximo, 32 caracteres.

Caso seja executada uma expressão que gere um valor como resultado e, nessa expressão não haja uma atribuição do resultado para alguma variável definida pelo usuário, o resultado será armazenado na variável **ans**, pré-definida pelo ambiente.

2.2 Manipulação

Existem comandos próprios para manipulação de variáveis. Abaixo estão listados os mais utilizados.

- **who**: Lista os nomes das variáveis.
- **whos**: Lista o nome e o tipo das variáveis.
- **clear**: Elimina todas as variáveis da área de trabalho. Para se apagar uma ou mais variáveis utiliza-se o comando **clear** seguido dos nomes das variáveis separadas por espaço.

- **save**: Salva as variáveis em arquivo, podendo utilizá-las novamente na próxima vez que o programa for inicializado ou mesmo quando executado o comando **clear**.
- **load**: Recupera as variáveis previamente salvas em arquivo pelo comando **save**.
- **clc**: Limpa a janela de comandos.

2.3 Variáveis pré-definidas

No MATLAB®, existem algumas variáveis pré-definidas que podem ser úteis ao usuário, as quais são listadas na Tabela 2.1.

Variável	**Representação**
ans	Variável padrão para armazenar resultados.
pi	Razão entre o perímetro da circunferência e o seu diâmetro.
eps	Precisão de ponto flutuante, ou distância entre o 1 e o próximo número real.
inf	Infinito (por exemplo, resultado da divisão 1/0).
NaN ou *nan*	*Not-a-Number* ou valor não-numérico (exemplo: resultado de 0/0).
i ou *j*	Unidade numérica imaginária, igual a $\sqrt{-1}$.
nargin	Número de argumentos de entrada de uma função.
nargout	Número de argumentos de saída de uma função.
realmin	Menor número real positivo utilizável.
realmax	Maior número real positivo utilizável.
bitmax	Maior número inteiro positivo utilizável.

Tabela 2.1: Variáveis pré-definidas.

Caso o usuário defina e utilize uma variável com o mesmo nome de alguma das variáveis ou das funções pré-definidas, ela funcionará como uma variável comum, perdendo seu significado original. Quando apagada, ela retornará ao seu comportamento pré-definido.

```
>> i=999
i =
   999
>> clear i
>> i
ans =
        0 + 1.0000i
```

Capítulo 3

Números e matrizes

3.1 Representação numérica

Todas as informações numéricas do MATLAB® são armazenadas sob a forma de matrizes. Se, por exemplo, for digitado o número 5 no ambiente do programa, este valor será interpretado como uma matriz de dimensão 1×1.

Números negativos, com casas decimais, complexos e sob a forma de notação científica podem ser representados no MATLAB® usando-se a seguinte simbologia:

- números negativos: '$-$' ;

  Ex.: -5

- números com casas decimais: '.' ;

  Ex.: 3.2

- números complexos: 'i' ou 'j' (como sufixos, à direita da parte imaginária, ou como funções, equivalentes à raiz quadrada de -1);

  Ex.: $3 + 5i$, $3 + 5j$, $3 + 5 * i$, $3 + 5 * j$, $3 + i * 5$, $3 + j * 5$, $3 + 5 * sqrt(-1)$ e $3 + sqrt(-1) * 5$.

- notação científica: 'e' ou 'E'.

  Ex.: $5e3$ (equivalente a 5×10^3) e $-2e-4$ (equivalente a -2×10^{-4})

3.2 Formatos de visualização de números

Todos os cálculos são executados no MATLAB® com aritmética de dupla precisão. A visualização dos números nas janelas *Command Window* e *Workspace*, entretanto, pode ser feita em diversos formatos. Por definição, o MATLAB® exibe os resultados em três formatos diferentes: inteiro, real com quatro casas decimais ou em notação científica, adotando aquela que melhor convier.

Esse comportamento padrão pode ser alterado clicando-se em *File* > *Preferences*, ou simplesmente digitando na Janela de Comando a instrução **format** seguida do formato específico. A Tabela 3.1 ilustra cada um deles, exibindo o número $\sqrt{2}$ como exemplo.

Comando	$\sqrt{2}$	**Comentário**
format short	1.4142	5 dígitos (ponto fixo).
format long	1.41421356237310	15 dígitos (ponto fixo).
format short e	1.4142e+000	5 dígitos e expoente (ponto flutuante).
format long e	1.414213562373095e+000	15 dígitos e expoente (ponto flutuante).
format short g	1.4142	O melhor entre "short" e "short e".
format long g	1.4142135623731	O melhor entre "long" e "long e".
format hex	3ff6a09e667f3bcd	Hexadecimal em ponto flutuante.
format +	+	Positivo '+', negativo '−' ou zero ' '.
format bank	1.41	2 casas decimais representando moeda.
format rat	1393/985	Aproximação racional.
format debug	Structure address = 12ac208 m = 1 n = 1 pr = 12600c0 pi = 0 1.4142	"short g" e informações sobre armazenagem interna.

Tabela 3.1: Formatação da visualização de resultado numérico.

3.3 Definição de matrizes

Existem diversas formas de se montar uma matriz. A mais simples delas utiliza elementos denominados **aglutinadores**, representados simbolicamente por colchetes, '[' e ']' . A função dos aglutinadores consiste em concatenar (encadear), horizontalmente ou verticalmente, dados de um mesmo tipo, podendo formar matrizes numéricas ou vetores de caracteres (*strings*).

O exemplo abaixo mostra a concatenação de duas palavras com um espaço no meio, todos definidos entre aspas simples (tipo caracter).

```
>> a='Duas';
>> b=' ';
>> c='palavras.';
>> [a b c]
ans =
Duas palavras.
```

Entendido o conceito de aglutinação, que será muito utilizado daqui em diante, torna-se fácil a tarefa de definir uma matriz. Uma matriz é montada linha após linha, onde espaço ou vírgula indicam transição de coluna e ponto-e-vírgula indica transição de linha.

```
>> [1,2;3,4]
ans =
    1     2
    3     4

>> [ans [5;6]]
ans =
     1     2     5
     3     4     6
```

Outra maneira de definir matrizes consiste em criar vetores-linha com elementos em progressão aritmética, através da simples sintaxe:

valor_inicial : incremento : valor_final

onde os valores fornecidos não precisam ser necessariamente inteiros.

Basicamente, o primeiro elemento do vetor criado corresponde ao valor inicial e os elementos seguintes são acrescidos do passo (ou incremento), de tal maneira que nunca ultrapasse o valor final. Caso o incremento seja omitido, ele será entendido como igual a 1.

```
>> 1:5:20
ans =
    1     6    11    16

>> 2.71 : 5.71
ans =
    2.7100    3.7100    4.7100    5.7100
```

Uma terceira forma de criar vetores utiliza a função **linspace**, que espaça linearmente um determinado número de elementos entre um valor final e um inicial. Sua sintaxe é:

linspace(valor_inicial, valor_final, número_de_elementos)

```
>> linspace(0,10,5)
ans =
         0    2.5000    5.0000    7.5000   10.0000
```

A função **logspace** é similar, porém espaça os elementos logaritmicamente na base 10. Uma diferença muito importante em sua sintaxe é que os valores iniciais e finais são potências de 10, devendo ser escritos apenas seus **expoentes**.

```
>> logspace(0,1,5)
ans =
    1.0000    1.7783    3.1623    5.6234   10.0000
```

Existem muitas outras formas de se criar uma matriz. Apenas a título de curiosidade, a função **xlsread** lê uma matriz a partir de um arquivo XLS gerado pelo programa Microsoft Excel® .

3.4 Indexação

O padrão de indexação matricial no MATLAB® é a forma tradicional intuitiva (r, c), onde r representa o número da linha e c representa o número da coluna.

```
>> m=[.1 .2 .3 .4 .5 ; .6 .7 .8 .9 1 ; 1.1 1.2 1.3 1.4 1.5]
m =
    0.1000    0.2000    0.3000    0.4000    0.5000
    0.6000    0.7000    0.8000    0.9000    1.0000
    1.1000    1.2000    1.3000    1.4000    1.5000
>> m(2,5)
ans =
     1
```

Os elementos da matriz também podem ser indexados seqüencialmente, conforme ilustrado na Tabela 3.2.

	1	2	3	4	5
1	(1) 0.1	(4) 0.2	(7) 0.3	(10) 0.4	(13) 0.5
2	(2) 0.6	(5) 0.7	(8) 0.8	(11) 0.9	(14) 1.0
3	(3) 1.1	(6) 1.2	(9) 1.3	(12) 1.4	(15) 1.5

Tabela 3.2: Indexação de elementos de uma matriz.

```
>> m(5)
ans =
    0.7000
```

É possível selecionar mais de uma linha ou coluna, usando-se dois-pontos ':' entre os índices inicial e final para indicar o intervalo. Quando os índices inicial e final são omitidos, indica-se o intervalo inteiro. A palavra reservada *'end'* indica o fim da linha ou da coluna.

```
>> m(3,1:4)
ans =
    1.1000    1.2000    1.3000    1.4000

>> m(:,5)
ans =
    0.5000
    1.0000
    1.5000

>> m(1,3:end)
ans =
    0.3000    0.4000    0.5000
```

O conceito de indexação pode ser ampliado quando no lugar dos índices colocam-se vetores ou matrizes de índices.

```
>> m([1 3],[2 4])
ans =
    0.2000    0.4000
    1.2000    1.4000

>> m([1 5 9; 4 8 12])
ans =
    0.1000    0.7000    1.3000
    0.2000    0.8000    1.4000
```

Caso se coloque um índice que exceda as dimensões da matriz, o resultado será uma mensagem de erro. Por outro lado, se for definido um novo elemento que exceda essas mesmas dimensões, então a matriz será redimensionada de forma a incluir esse novo elemento, sendo as novas posições preenchidas com 0.

```
>>m(4,6)=2.1
ans =
    0.1000    0.2000    0.3000    0.4000    0.5000         0
    0.6000    0.7000    0.8000    0.9000    1.0000         0
    1.1000    1.2000    1.3000    1.4000    1.5000         0
         0         0         0         0         0    2.1000
```

Capítulo 4

Operações com matrizes

4.1 Operações aritméticas

As operações aritméticas no MATLAB® podem ser de dois tipos: matricial ou escalar. As operações do tipo matricial referem-se às operações matemáticas sobre matrizes. As operações escalares são também denominadas de operações sobre conjuntos. Essas últimas são realizadas elemento a elemento de cada matriz, aplicando-se o operador em questão apenas entre elementos de mesma posição matricial.

Definindo-se $x = [23; 57]$ e $y = [16; 24]$, pode-se associá-los pelos seguintes operadores:

- '+' : adição (matricial e escalar).

  Ex.: $x + y = [3\ \ 9; 7\ \ 11]$

- '−' : subtração (matricial e escalar).

  Ex.: $x - y = [1\ \ -3; 3\ \ 3]$

- '∗' : multiplicação matricial.

  Ex.: $x * y = [8\ \ 24; 19\ \ 58]$

- '/' : divisão matricial à direita.

  Ex.: $x/y = (x * y^{-1}) = x * inv(y) = [-0.2500\ \ 1.1250; -0.7500\ \ 2.8750]$, onde: **inv(m)** é uma função que será explicada na Seção 5.2.

- '\' : divisão matricial à esquerda.

  Ex.: $x\backslash y = (x^{-1} * y) = inv(x) * y = [-1.0000\ \ -30.0000; 1.0000\ \ 22.0000]$

- '^' : potenciação matricial.

  Ex.: $x\hat{}2 = x^2 = (x * x) = [19\ \ 27; 45\ \ 64]$

- ''' : transposição matricial.

  Ex.: $x' = x^T = [2\ \ 5; 3\ \ 7]$

- '.∗' : multiplicação escalar.

  Ex.: $x. * y = [2\ \ 18; 10\ \ 28]$

- './' : divisão escalar à direita.

  Ex.: $x./y = (x. * y.^{-1}) = [2.0000 \;\; 0.5000; 2.5000 \;\; 1.7500]$

- '.\' : divisão escalar à esquerda.

  Ex.: $x.\backslash y = (x.^{-1}. * y) = [0.5000 \;\; 2.0000; 0.4000 \;\; 0.5714]$

- '.^' : potenciação escalar.

  Ex.: $x.\hat{}\,2 = [4 \;\; 9; 25 \;\; 49]$

Observações:

1. Para operações entre matriz e número escalar, o programa faz uma expansão escalar do número, executando a operação entre o número e cada elemento da matriz.

   Ex.: $x + 3 = x + [3 \;\; 3; 3 \;\; 3] = [5 \;\; 6; 8 \;\; 10]$

2. A precedência de operações pode ser controlada utilizando-se parênteses.

   Ex.: $((x + 3) * x')' = [28 \;\; 46; 67 \;\; 110]$

4.2 Operações lógicas e relacionais

Os operadores relacionais e lógicos são usados em expressões lógicas *booleanas* para implementar testes de tomadas de decisão, fornecendo respostas do tipo verdadeiro/falso a perguntas. Para o MATLAB®, qualquer número diferente de zero representa o valor lógico verdadeiro, e zero representa o valor lógico falso.

Os operadores relacionais e lógicos são apresentados na Tabela 4.1.

Operador	**Descrição**
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
==	Igual a (não confundir com =)
~=	Diferente de
&	E
|	OU
~	NÃO

Tabela 4.1: Operadores relacionais e lógicos.

É interessante ressaltar que no MATLAB® os operadores podem ser usados não somente com escalares, mas também com vetores e matrizes, sendo a operação realizada elemento a elemento.

```
>> a=-3; b=7;
>> a<b
ans =
     1

>> a<b & a==b
ans =
     0

>> x=1:10
x =
     1     2     3     4     5     6     7     8     9    10

>> y=x>=5
y =
     0     0     0     0     1     1     1     1     1     1
```

Capítulo 5

Funções matriciais

Qualquer função no MATLAB® tem como sintaxe geral o seguinte padrão:

[*Saída1, ... , SaídaN*] = *Nome (Entrada1, ... , EntradaN)*

Os parâmetros de entrada devem ser postos na ordem adequada e representam os dados fornecidos pelo usuário. Eles são processados pela função, que gera os parâmetros de saída, os quais são armazenados nas variáveis definidas pelo usuário no comando de chamada da função. Deve-se ressaltar que os parâmetros de entrada são escritos dentro de parênteses e os de saída dentro de colchetes, segundo a sintaxe geral de funções e matrizes, respectivamente.

Os parâmetros de entrada podem ser fornecidos de duas formas: escrevendo-se diretamente o número ou matriz, ou escrevendo-se o nome da variável correspondente.

Caso as variáveis de saída não sejam fornecidas pelo usuário, o programa retorna apenas a primeira saída (*Saída1*) na variável padrão **ans**.

Abaixo, são apresentadas algumas das funções mais utilizadas para criação e manipulação de matrizes. Matrizes definidas em alguns dos exemplos poderão ser utilizadas em exemplos posteriores desse mesmo capítulo.

A maioria das funções no MATLAB® apresenta inúmeras opções de funcionamento. Para cada função, será abordado apenas um exemplo simples. Detalhamento das funções pode ser obtido através do comando de ajuda **help nome_da_função**.

5.1 Matrizes elementares

- **ones**: cria uma matriz cujos elementos são todos iguais a 1, dados o número de linhas e colunas da matriz, respectivamente.

```
>> ones(2,3)
ans =
    1     1     1
    1     1     1
```

Para se criar uma matriz com todos os elementos iguais a um número qualquer N, basta multiplicar a matriz gerada por N ou multiplicar diretamente o comando por N (N*__ones__).

- **zeros**: cria uma matriz cujos elementos são todos iguais a 0, dados os números de linhas e colunas.

```
>> zeros(3,2)
ans =
    0     0
    0     0
    0     0
```

- **eye**: cria uma matriz identidade, dada sua ordem.

```
>> eye(3)
ans =
    1     0     0
    0     1     0
    0     0     1
```

- **rand**: cria uma matriz de elementos pseudo-aleatórios com distribuição uniforme entre 0 e 1, dados os números de linhas e colunas. O comando **rand('state',n)** reinicia o gerador de números pseudo-aleatórios para o n-ésimo estado.
Para reiniciar em um diferente estado a cada horário, pode-se usar o comando **rand('state',sum(100*clock))**. Um comando similar, o **randn**, funciona de forma similar, mas com distribuição normal.

```
>> rand(2,7)
ans =
    0.9501    0.6068    0.8913    0.4565    0.8214    0.6154    0.9218
    0.2311    0.4860    0.7621    0.0185    0.4447    0.7919    0.7382
```

Esta função é muito útil para, por exemplo, se fazer testes em programas que vão trabalhar com dados imprevisíveis.

5.2 Álgebra linear

- **det**: retorna o determinante da matriz de entrada.

```
>> a=[1 2;1 3]
a =
     1     2
     1     3

>> det(a)
ans =
    1
```

- **inv**: retorna a inversa da matriz de entrada.

```
>> inv(a)
ans =
     3    -2
    -1     1
```

- **eig**: retorna um vetor de autovalores da matriz de entrada. Para duas variáveis de saída, retorna a matriz de autovetores e a matriz diagonal de autovalores.

```
>> v=eig(a)
v =
    0.2679
    3.7321

>> [V D]=eig(a)
V =
   -0.9391   -0.5907
    0.3437   -0.8069

D =
    0.2679         0
         0    3.7321
```

5.3 Informações matriciais básicas

- **size**: retorna um vetor com o número de linhas e colunas da matriz de entrada.

```
>> size( [11 12 13 ; 21 22 23] )
ans =
    2     3
```

Este último é um exemplo em que se escreve diretamente uma matriz ao invés do nome da variável.

- **length**: retorna o comprimento do vetor ou a maior dimensão de uma matriz de entrada.

```
>> length( [11 12 13 ; 21 22 23] )
ans =
     3
```

5.4 Manipulação de matrizes

- **reshape**: dá um novo formato, quando possível, à matriz de entrada, dados a matriz e seus novos números de linhas e colunas. Os elementos da matriz são percorridos sequencialmente, pelo modo de indexação $matriz(i)$, apresentado na Tabela 3.2.

```
>> reshape(1:10,2,5)
ans =
     1     3     5     7     9
     2     4     6     8    10
```

- **rot90**: rotaciona a matriz de entrada em 90 graus, no sentido anti-horário.

```
>> b=[1 2 3 ; 4 5 6 ; 7 8 9]
b =
     1     2     3
     4     5     6
     7     8     9

>> rot90(b)
ans =
     3     6     9
     2     5     8
     1     4     7
```

- **fliplr**: troca simetricamente de posição as colunas da esquerda com as da direita, de uma matriz de entrada.

```
>> fliplr(b)
ans =
     3     2     1
     6     5     4
     9     8     7
```

- **flipud**: troca simetricamente de posição as linhas de cima com as de baixo, de uma matriz de entrada.

```
>> flipud(b)
ans =
     7     8     9
     4     5     6
     1     2     3
```

- **diag**: cria um vetor a partir da diagonal principal de uma matriz de entrada ou cria uma matriz diagonal a partir de um vetor de entrada.

```
>> diag(b)
ans =
     1
     5
     9

>> diag(ans)
ans =
     1     0     0
     0     5     0
     0     0     9
```

- **triu**: retorna uma matriz triangular superior, a partir da matriz de entrada.

```
>> triu(b)
ans =
     1     2     3
     0     5     6
     0     0     9
```

- **tril**: retorna uma matriz triangular inferior, a partir da matriz de entrada.

```
>> tril(b)
ans =
     1     0     0
     4     5     0
     7     8     9
```

5.5 Análise de dados

- **sum**: retorna a soma dos elementos de cada coluna de uma matriz de entrada ou a soma dos elementos de um vetor de entrada.

```
>> sum(b)
ans =
    12    15    18

>> sum(ans)
ans =
    45
```

- **prod**: retorna o produto dos elementos de cada coluna de uma matriz de entrada ou o produto dos elementos de um vetor de entrada.

```
 > prod([2 3 1; 2 4 5])
ans =
     4    12     5

>> prod(ans)
ans =
   240
```

- **mean**: retorna a média aritmética dos elementos de cada coluna de uma matriz de entrada ou a média aritmética dos elementos de um vetor de entrada.

```
>> mean( [1 -2 3 -4; 5 -6 7 -8] )
ans =
     3    -4     5    -6
```

- **min** e **max**: retornam, respectivamente, o valor mínimo e o valor máximo dos elementos de cada coluna de uma matriz de entrada ou o valor mínimo e o valor máximo dos elementos de um vetor de entrada.

```
>> max( [1 -2 3 -4; 5 -6 7 -8] )
ans =
     5    -2     7    -4

>> min(ans)
ans =
    -4
```

- **sort**: retorna a matriz de entrada, com cada coluna ordenada em ordem crescente, ou o vetor de entrada, com seus elementos ordenados em ordem crescente.

```
>> x=sort( [1 -2 3 -4 5 -6 7 -8] )
x =
    -8    -6    -4    -2     1     3     5     7

>> fliplr(x)
ans =
     7     5     3     1    -2    -4    -6    -8
```

Capítulo 6

Funções matemáticas elementares

O MATLAB® possui diversas funções matemáticas elementares, as quais podem ser listadas pelo comando **help elfun**. Abaixo, estão listadas as mais comumente utilizadas.

6.1 Funções trigonométricas

As funções trigonométricas do MATLAB® trabalham com valores de ângulos expressos em radianos.

sin	seno
sinh	seno hiperbólico
asin	arco seno
cos	cosseno
cosh	cosseno hiperbólico
acos	arco cosseno
tan	tangente
tanh	tangente hiperbólica
atan	arco tangente

sec	secante
sech	secante hiperbólica
asec	arco secante
csc	cossecante
csch	cossecante hiperbólica
acsc	arco cossecante
cot	cotangente
coth	cotangente hiperbólica
acot	arco cotangente

```
>> cos(pi)
ans =
    -1

>> acos(ans)
ans =
    3.1416
```

6.2 Funções exponenciais

exp	exponencial (e^x)
log	logaritmo neperiano ou natural ($\ln x$)
log10	logaritmo na base 10 ($\log_{10} x$)
log2	logaritmo na base 2 ($\log_2 x$)
sqrt	raiz quadrada ($\sqrt{x}$)

```
>> log(exp(10))
ans =
    10

>> sqrt(log10(ans))
ans =
     1
```

6.3 Funções complexas

abs	valor absoluto do número (módulo)
angle	ângulo de fase do número, em radianos
conj	conjugado do número
real	parte real do número
imag	parte imaginária do número

```
>> z=4-3*i
z =
   4.0000 - 3.0000i

>> abs(z)
ans =
     5

>> real(z)+imag(z)*i
ans =
   4.0000 - 3.0000i

>> conj(z)
ans =
   4.0000 + 3.0000i
```

6.4 Funções de arredondamento e resto

- **fix**: Aproxima para o inteiro de menor valor absoluto, ignorando as casas decimais.
- **floor**: Aproxima para o inteiro antecessor.
- **ceil**: Aproxima para o inteiro sucessor.
- **round**: Arredonda para o inteiro mais próximo.
- **rem** (remainder after division): Resto de divisão inteira, dados respectivamente o dividendo e o divisor. Para dividendos negativos, sendo `x` e `y` inteiros positivos, `rem(-x,y)=-rem(x,y)`. Mais detalhes em **help rem**.

- **mod** (modulus after division): Resto de divisão inteira, dados respectivamente dividendo e o divisor. Para dividendos negativos, sendo `x` e `y` inteiros positivos, `mod(-x,y)=y-mod(x,y)` se `mod(x,y)`$\neq$`0`, ou `mod(-x,y)=0` caso contrário. Mais detalhes em **help mod**.
- **sign**: Retorna $+1$ para números positivos, -1 para números negativos e 0 para números iguais a zero.

```
>> x=[-4:4]
x =
    -4    -3    -2    -1     0     1     2     3     4

>> rem(x,3)
ans =
    -1     0    -2    -1     0     1     2     0     1

>> mod(x,3)
ans =
     2     0     1     2     0     1     2     0     1

>> sign(x)
ans =
    -1    -1    -1    -1     0     1     1     1     1
```

Observações:

1. REM(x,y) tem o mesmo sinal de x, enquanto MOD(x,y) tem o mesmo sinal de y.
2. REM(x,y) e MOD(x,y) são iguais se x e y possuem o mesmo sinal, mas diferem de y se x e y possuem sinais opostos.

Capítulo 7

Funções polinomiais

No MATLAB®, um vetor pode ser interpretado como um polinômio quando cada um de seus elementos é associado a cada um dos coeficientes do polinômio, começando pelos coeficientes dos termos de maior grau. Assim, por exemplo, `[2 -7 1]` será interpretado como $2x^2 - 7x + 1$.

Tendo isso em conta, listam-se abaixo algumas das funções que trabalham com polinômios. Vetores definidos em alguns exemplos poderão ser utilizados em exemplos posteriores.

- **roots**: Retorna um vetor com as raízes de um polinômio de entrada.

```
>> a=[1,5,6]
a =
    1     5     6

>> roots(a)
ans =
    -3
    -2
```

- **polyval**: Retorna o valor ou a imagem de um polinômio, dados respectivamente o polinômio e o valor de sua variável independente.

```
>> polyval(a,2)
ans =
    20
```

  Isto é o mesmo que $1(2)^2 + 5(2) + 6 = 20$.

- **poly**: Cria um polinômio a partir de um vetor de entrada contendo suas raízes.

```
>> poly([-3,-2])
ans =
    1     5     6
```

- **conv**: Convolui dois polinômios. Na prática, isso é o mesmo que multiplicar de forma distributiva dois polinômios (mas não dois vetores).

```
>> b=[2 4];
>> conv(a,b)
ans =
        2    14    32    24
```

Neste exemplo, mostra-se que $(x^2 + 5x + 6)(2x + 4) = 2x^3 + 14x^2 + 32x + 24$.

Capítulo 8

Gráficos

8.1 Gráficos de duas dimensões

O MATLAB® é um *software* muito eficiente na criação e manipulação de gráficos, apresentando diversas funções que auxiliam essas operações.

Basicamente, os gráficos são construídos conforme os passos abaixo:

1. Cria-se um vetor `X` com as coordenadas do eixo das abscissas;
2. Escreve-se a função desejada, a partir do vetor `X`, a qual criará um novo vetor `Y` das ordenadas;
3. Desenha-se o gráfico, o que será ensinado mais adiante.

Os pares ordenados `(x,y)` assim criados são marcados no gráfico e ligados por segmentos retas (interpolação linear). Observa-se, portanto, que, quanto menor for o incremento do vetor `X`, mais pares ordenados serão criados e, conseqüentemente, melhor será a precisão do gráfico. Em compensação, uma maior quantidade de memória é utilizada.

Entre as inúmeras funções existentes, estão listadas abaixo apenas as mais usadas.

- **plot**: Desenha gráficos de duas dimensões, dados o vetor de abscissas e o vetor de ordenadas.

```
>> X=[0:0.2:10];
>> Y=X.^2;
>> plot(X,Y)
```

A função **plot** pode ser também usada no desenho de mais de um gráfico em uma mesma figura. Para isso, usa-se a seguinte notação:

$$plot(X1,\ Y1,\ X2,\ Y2,\ \ldots\ ,\ Xn,\ Yn)\ ,$$

onde *X1* e *Y1* correspondem às coordenadas do gráfico 1, *X2* e *Y2* correspondem às coordenadas do gráfico 2, e assim sucessivamente.

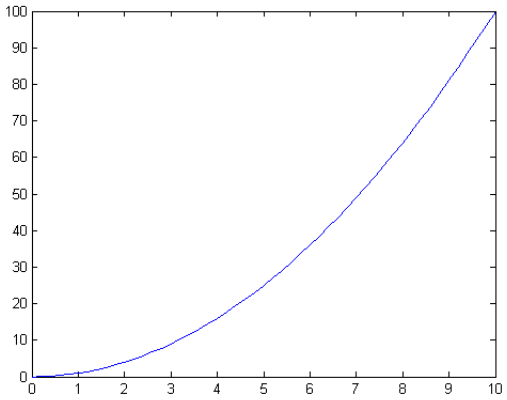


Figura 8.1: Gráfico do comando *plot(X,Y)*.

Cor		Marcador		Tipo de linha	
y	amarela	.	ponto	-	sólida
m	magenta	o	círculo	:	pontilhada
c	azul-claro	x	xis	-.	traço-ponto
r	vermelha	+	cruz	--	tracejada
g	verde	*	estrela		
b	azul	s	quadrado		
w	branca	d	losango		
k	preta	v	triângulo p/ baixo		
		^	triângulo p/ cima		
		<	triângulo p/ esquerda		
		>	triângulo p/ direita		
		p	pentagrama		
		h	hexagrama		

Tabela 8.1: Formatação, em gráficos, de cor, marcador e tipo de linha.

A formatação de cor, marcador e tipo de linha dos gráficos também pode ser configurada pelo usuário. Para isso usa-se a seguinte notação:

plot(X1, Y1, 'Conf1', X2, Y2, 'Conf2', ... , Xn, Yn, 'Confn') ,

onde, no lugar de *'Conf1', 'Conf2', ... , 'Confn'*, usam-se os caracteres especificados pela Tabela 8.1.

```
>> plot(X,Y,'ko--',X,10*X,'*')
```

Para criar gráficos em janelas diferentes, usa-se o comando **figure(n)** antes do comando **plot**, onde **n** é o número da janela de figuras. Caso contrário, cada novo gráfico criado será plotado na última janela aberta (gráfico corrente). Para apagar o gráfico da janela corrente sem fechá-la, escreve-se o comando **clf**.

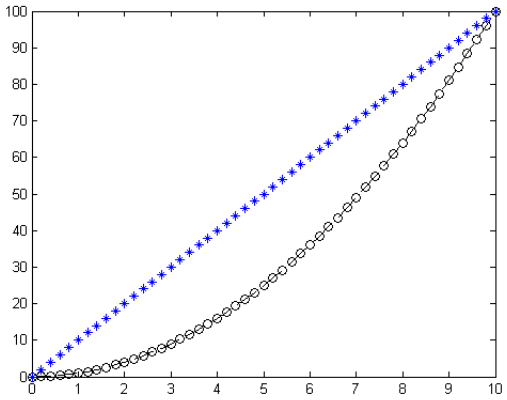


Figura 8.2: Gráfico de *plot(X,Y,'ko–',X,10*X,'*')*.

- **subplot**: Divide a janela de figuras em uma matriz `m x n` de sub-janelas, selecionando uma das sub-janelas pelo seu número correspondente. Esses números estão ordenados sucessivamente ao longo das linhas. As entradas dessa função são respectivamente o número de linhas e colunas da janela e a posição do par de eixos corrente.

```
>> subplot(2,2,1)
>> plot(X,X.^2)
>> subplot(2,2,2)
>> plot(X,10*X)
>> subplot(2,2,3)
>> plot(X,log10(X+1))
```

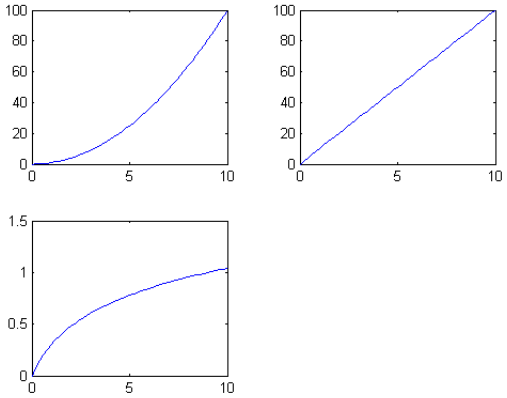


Figura 8.3: Par de eixos 1, 2 e 3 da janela de figuras.

- **semilogx**: Plota gráficos com o eixo x em escala logarítmica na base 10.
- **semilogy**: Plota gráficos com o eixo y em escala logarítmica na base 10.
- **loglog**: Plota gráficos com ambos os eixos em escala logarítmica na base 10.

```
>> clf
>> semilogx(X,10*X)
```

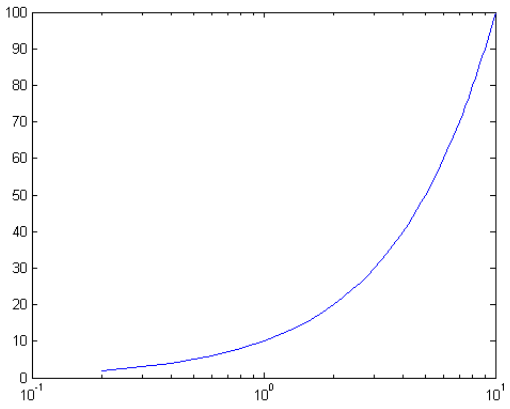


Figura 8.4: Reta representada com eixo x em escala logarítmica.

- **plotyy**: Plota duas curvas no mesmo gráfico com diferentes escalas para o eixo y. A primeira delas tem escala à esquerda, e a segunda tem escala à direita.

```
>> plotyy(X,Y,X,cos(X))
```

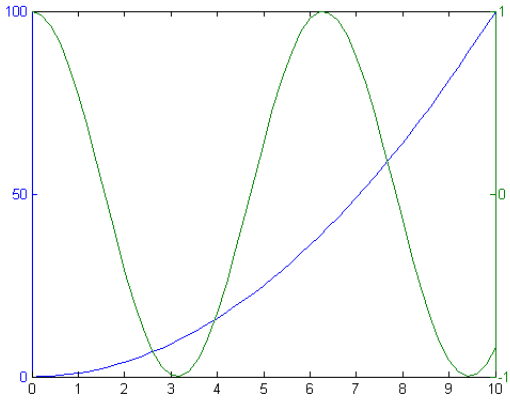


Figura 8.5: Parábola e cosseno com diferentes escalas para o eixo y.

- **polar**: Plota gráficos em coordenadas polares segundo a seguinte sintaxe:

  polar(theta, raio, 'Conf') ,

  onde *theta* é o vetor de valores angulares em radianos, *raio* é o vetor de mesma dimensão que *theta*, contendo os valores radiais correspondentes e $'Conf'$ segue as mesmas configurações da função **plot**.

```
>> ang=[0:.1:2*pi];
>> raio=10*ang;
>> polar(ang,raio);
```

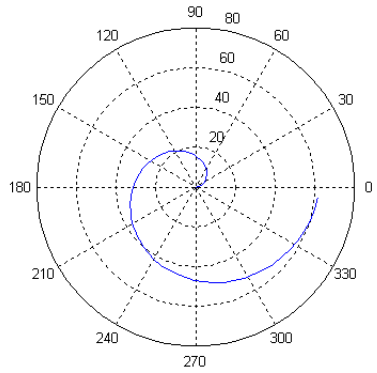


Figura 8.6: Gráfico da função polar.

8.2 Gráficos de três dimensões

O MATLAB® apresenta diversos recursos para apresentação de gráficos em 3D. As principais funções são listadas abaixo.

- **plot3**: Plota pontos e linhas em 3D, a partir de três vetores de coordenadas de mesmo tamanho.

```
>> t = 0:0.1:10*pi;
>> plot3(t.*sin(t),t.*cos(t),t);
```

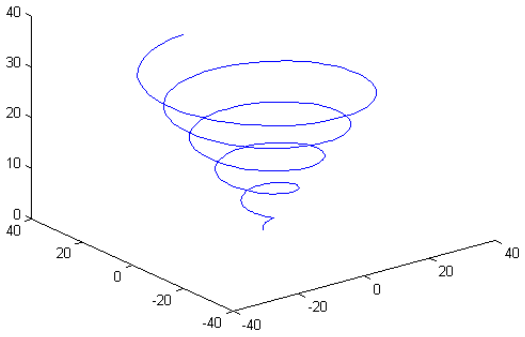


Figura 8.7: Gráfico do comando *plot3*.

- **meshgrid**: Cria duas matrizes X e Y, a partir de dois vetores x e y, onde as linhas da matriz X são cópias do vetor x, e as colunas da matriz Y são cópias do vetor y, sendo feitas tantas cópias quanto forem necessárias para que ambas as matrizes tenham as mesmas dimensões. Essa função é importante para a criação de malhas e superfícies em três dimensões.

```
>> x=[0 1]
x =
        0     1

>> y=[2 3 4]
y =
        2     3     4

>> [X,Y]=meshgrid(x,y)
X =
        0     1
        0     1
        0     1

Y =
        2     2
        3     3
        4     4
```

- **mesh**: Cria uma malha em 3D, a partir de três matrizes.

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = X .* exp(-X.^2 - Y.^2);
>> mesh(X,Y,Z);
```

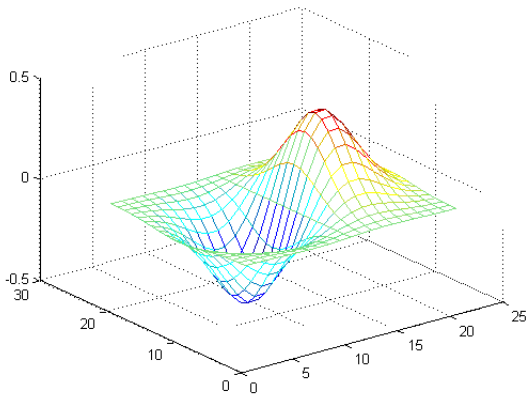


Figura 8.8: Gráfico do comando *mesh*.

- **surf**: Cria uma superfície em 3D, a partir de três matrizes.

```
>> surf(X,Y,Z);
```

As cores, tanto da superfície quanto da malha em 3D, estão codificadas de acordo com a altura relativa dos pontos no eixo z. Quanto maior for z, mais avermelhada será a cor, e quanto menor for z, mais azulada ela será.

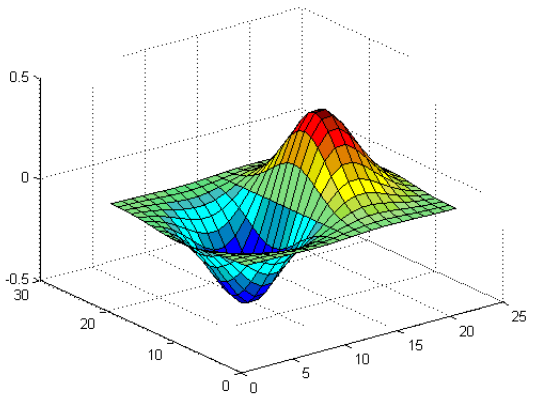


Figura 8.9: Gráfico do comando *surf*.

8.3 Funções auxiliares

Existem diversas funções que auxiliam a visualização e a formatação dos gráficos. As mais importantes são apresentadas abaixo.

- **title**: Adiciona um título ao gráfico corrente. Esse título é um texto que deve ser escrito entre aspas simples, conforme a sintaxe:

  title('Título do Gráfico')

- **xlabel**: Adiciona um texto ao eixo das abscissas do gráfico corrente.
- **ylabel**: Adiciona um texto ao eixo das ordenadas do gráfico corrente.
- **text**: Cria um texto posicionado na coordenada fornecida, conforme a sintaxe:

  text(X, Y, 'Texto')

- **grid**: Mostra linhas de grade no gráfico corrente.
- **axis**: Delimita os intervalos dos eixos de acordo com a sintaxe:

  axis([Xmin Xmax Ymin Ymax])

  Essa função ajusta o gráfico para os intervalos delimitados pelo usuário. Outras aplicações podem ser vistas em *help axis*.

- **hold on**: Permite que novos gráficos sejam criados na janela corrente sem que os gráficos anteriores sejam apagados.
- **hold off**: Desabilita o comando anterior.

- **whitebg**: Configura a cor de fundo da janela de figura, de acordo com a sintaxe:

  whitebg(n,[*red green blue*]*)* ,

  onde *n* é o número da janela desejada e [*red green blue*] é o vetor com as componentes primárias da cor. O vetor [0 0 0] corresponde ao preto e o vetor [1 1 1] ao branco.

```
>> x=[-5*pi:0.1:5*pi];
>> y=sin(x)./x;
>> plot(x,y);
>> title('\it{Sample}')
>> xlabel('x');
>> ylabel('sin(x)/x');
>> axis([-15 15 -0.3 1.1]);
>> text(-2*pi, 0, '-2\pi');
>> text(2*pi, 0, '2\pi');
>> whitebg([0.4057 0.9355 0.9169]);
```

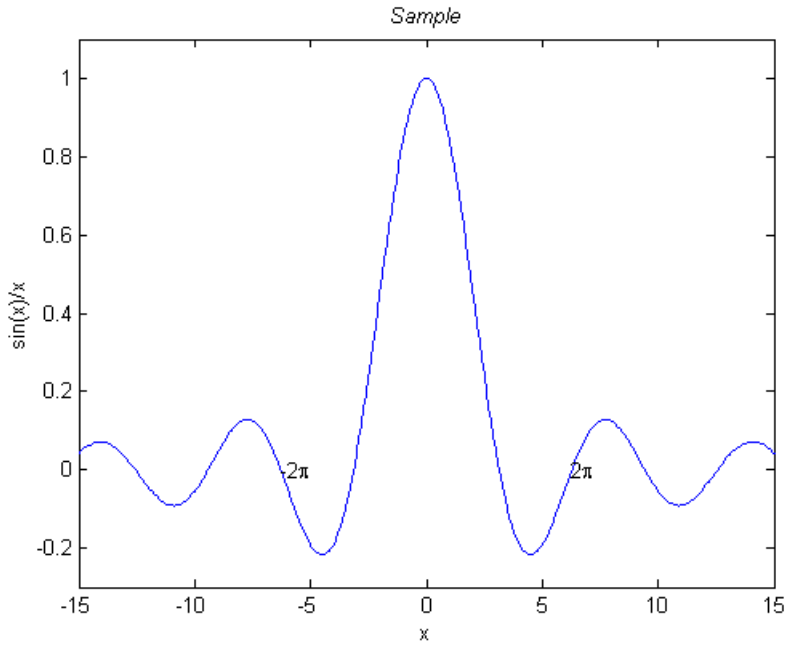


Figura 8.10: Exemplo de funções auxiliares de plotagem.

A cor do plano de fundo volta a ser branca quando a figura é exportada do MATLAB® . Nos textos inseridos nela, é possível mudar o tipo e o tamanho da fonte, escrever em negrito, itálico, subscrito, sobrescrito e ainda representar letras gregas. Isso pode ser compreendido pela Tabela 8.2.

- **clf**: Apaga todos os gráficos da janela corrente.
- **close**: Fecha a janela corrente. Para fechar a n-ésima janela, utiliza-se o comando **close(n)**. O comando **close all** fecha todas as janelas.

Propriedade	**Sintaxe**	**Exemplo**
Tipo de fonte	`\fontname{}`	`\fontname{'Arial'}`
Tamanho de fonte	`\fontsize{}`	`\fontsize{14}`
Negrito	`\bf{}`	`\bf{Negrito}`
Itálico	`\it{}`	`\it{Itálico}`
Subscrito	`_{}`	`R_{1}`
Sobrescrito	`^{}`	`x^{2}`
Letra grega	`\letra`	`\pi`

Tabela 8.2: Formatação de fonte de texto inserido em gráfico.

Capítulo 9

Funções polinomiais racionais complexas

Uma função polinomial racional complexa é definida como uma razão de polinômios de variáveis complexas, conforme a equação

$$H(s) = \frac{N_H(s)}{D_H(s)} = \frac{b_m s^m + ... + b_1 s + b_0}{a_n s^n + ... + a_1 s + a_0} = (k)\frac{(s+z_1)(s+z_2)...(s+z_m)}{(s+p_1)(s+p_2)...(s+p_n)} \ ,$$

onde os coeficientes $a_1, a_2, \cdots, a_n$ e $b_1, b_2, \cdots, b_m$ serão considerados reais e constantes, e $k = \frac{b_m}{a_n}$.

As raízes do numerador e do denominador são chamadas, respectivamente, de zeros e polos da função $H(s)$. A constante k é denominada ganho.

- **tf2zp**: Encontra zeros, polos e ganho, dados o numerador e denominador da função.

```
>> num=[1,1]
num =
        1     1

>> den=[1,5,6]
den =
        1     5     6

>> [z,p,g]=tf2zp(num,den)
z =
    -1

p =
    -3
    -2

g =
    1
```

- **zp2tf**: Encontra o numerador e denominador da função, dados zeros, polos e ganho.

```
>> [N,D]=zp2tf(z,p,g)
N =
     0     1     1

D =
     1     5     6
```

- **tf**: Cria uma função de transferência e a armazena em um objeto apropriado, dados o numerador e denominador.

```
>> H=tf(N,D)

Transfer function:
     s + 1
-------------
s^2 + 5 s + 6
```

- **pzmap**: Apresenta o diagrama de polos e zeros, dado o objeto da função de transferência. Cada raiz do numerador (zero) é representada no plano complexo por um círculo (∘), enquanto cada raiz do denominador (polo) é representada por um xis (×).

```
>> pzmap(H)
```

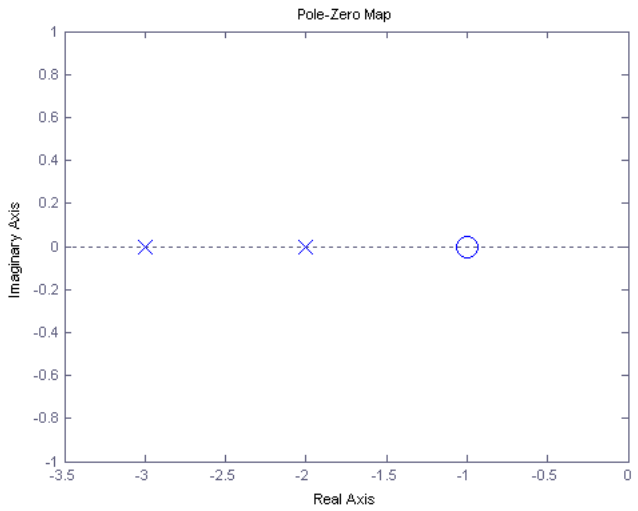


Figura 9.1: Diagrama de polos e zeros.

- **bode**: Desenha os gráficos de módulo $|H(j\omega)|_{dB}$ e ângulo de fase $\angle H(j\omega)$ da resposta em frequência $H(j\omega) = H(s)|_{s=j\omega}$, dado o objeto relativo à função de transferência $H(s)$.

```
>> bode(H)
```

O comando **bode(H,ω)** desenha o gráfico para um dado vetor de frequências ω, enquanto o comando **bode(H,{ω_{min},ω_{max}})** desenha-o para a faixa de frequências $[\omega_{min}; \omega_{max}]$.

```
>> bode(H,{1e-1,1e2})
```

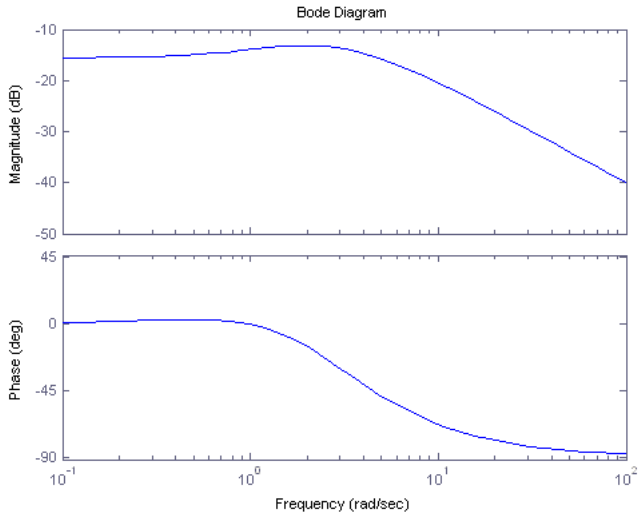


Figura 9.2: Gráficos de módulo e ângulo de fase da resposta em freqüência.

- **freqs**: Retorna um vetor complexo com a resposta em frequência $H(j\omega) = H(s)|_{s=j\omega}$, dados o seu numerador, o seu denominador e um vetor de frequências ω. Caso seja omitida a variável de retorno, esta função simplesmente exibe os gráficos de módulo e de ângulo de fase.

```
>> clf;
>> w=logspace(-1,2,101);
>> Hjw=freqs(num,den,w);
>> moddb=20*log10(abs(Hjw));
>> fase=180*angle(Hjw)/pi;
>> subplot(2,1,1), semilogx(w,moddb);
>> title('Resposta em freqüência'), ylabel('|H(j\omega)|_{dB}');
>> subplot(2,1,2), semilogx(w,fase);
>> ylabel('\angle H(j\omega) (graus)'), xlabel('\omega (rad/s)');
```

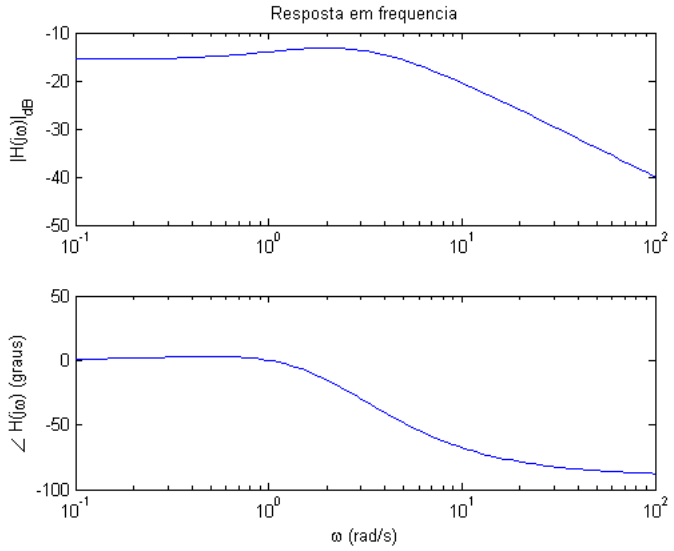


Figura 9.3: Gráfico de módulo e fase.

Capítulo 10

Programação

10.1 Funções e *scripts*

Funções e *scripts* são sequências de comandos do MATLAB® armazenadas em arquivos do tipo texto com extensão '.m' (*m-files*), que têm por finalidade automatizar processos repetitivos e implementar cálculos e comandos lógicos. A diferença básica entre ambos é que as funções aceitam parâmetros de entrada e retornam parâmetros de saída, enquanto os *scripts* unicamente executam a sequência de comandos.

As funções e os *scripts* podem ser criados escrevendo **edit** na linha de comando ou clicando na barra de ferramentas em *File* > *New* > *M-file*. Por ficarem armazenados em arquivos texto, eles podem ser editados em qualquer editor de textos.

Os mesmos comandos que o usuário escreveria no *prompt* da Janela de Comando são usados para a implementação do *script*. Para que o *m-file* seja interpretado como função, deve ser escrita a seguinte expressão no cabeçalho do arquivo:

function [*Saída1, ... , SaídaN*] = *Nome(Entrada1, ... ,EntradaM)*

onde *Saída1, ... , SaídaN* são os parâmetros de saída, *Entrada1, ... , EntradaM* são os parâmetros de entrada e *Nome* é o próprio nome da função.

Depois dessa expressão, seguem-se os comandos usados para estruturar a função. Tanto na função quanto no *script*, os comentários são escritos nas linhas iniciais. No caso da função, a primeira sequência de comentários aparecerá quando for digitado **help nome_da_função** na linha de comandos.

Abaixo segue um exemplo de função.

```
function[a,b,c,d,e] = complexo(x)

% COMPLEXO
%   [r,i,m,a,c] = COMPLEXO(x) retorna em cinco
%   variáveis as partes real e imaginária,
%   o módulo, o argumento e o conjugado de
%   um número complexo.

a=real(x); b=imag(x);
c=abs(x); d=angle(x);
e=conj(x);
```

Após escrito o código, o arquivo deverá ser salvo com o mesmo nome da função e com a extensão '.m' (no exemplo, 'complexo.m').

Uma condição necessária para a execução de arquivos M é que eles estejam no diretório corrente do MATLAB®. Para execução de um *script*, feita na barra de ferramentas da janela de edição pela opção *Debug>Run* ou *File>Run Script* (dependendo da versão do MATLAB®), a mudança de diretório corrente é automática, sendo desnecessária a intervenção do usuário.

A mudança de diretório corrente pode ser feita clicando-se no botão da janela principal do programa contendo '...', ao lado de *Current Directory*, ou por um dos seguintes comandos.

- dir - lista diretórios e arquivos do diretório corrente;
- cd *nome_diretório* - muda diretório corrente para *nome_diretório*;
- cd .. - retorna ao diretório pai, um nível acima;
- edit *nome* - edita uma função ou um *script* no diretório corrente;
- *nome_script* - executa um script, caso esteja no diretório corrente;
- [s1,s2]=*nome_função*(e1,e2) - chama uma função, caso esteja no diretório corrente ou na biblioteca do MATLAB®.
- cd - muda para o diretório raiz (*root*)

10.2 Controle de fluxo

As linguagens imperativas de programação de computadores caracterizam-se pela execução sequencial de comandos. Elas possuem ainda recursos que permitem o controle de fluxo de execução de comandos com base em estruturas de tomada de decisão. Nesta seção, serão vistas as estruturas mais comuns.

10.2.1 Estruturas condicionais

Uma estrutura condicional permite a seleção de um conjunto de comandos a serem executados quando uma dada condição for ou não satisfeita, podendo assim modificar o fluxo natural de comandos. As três estruturas condicionais básicas do MATLAB® estão listadas a seguir.

Estrutura *if* 1	Estrutura *if* 2	Estrutura *if* 3

if *condição*
  comandos
end
	if *condição1*
	  comandos1
	else
	  comandos2
	end
		if *condição1*
		  comandos1
		elseif *condição2*
		  comandos2
		elseif *condição3*
		  comandos3
		...
		else
		  comandosN
		end

As estruturas 1 e 2 são casos particulares da estrutura 3. As condições são formadas pelos operadores lógicos e relacionais. Abaixo é mostrado um exemplo de um *script* com uma estrutura *if*.

```
A=2; B=3;
if A==B
  display('A e B iguais.') % Exibe texto na Janela de Comando.
else
  display('A e B diferentes.')
end
```

Quando se faz uso repetido de testes de igualdade com apenas um argumento comum, costuma-se usar a estrutura *switch-case*. Essa estrutura tem a seguinte forma:

switch *var_teste*
case *expressão_teste1*
  comandos1
case {*expressão_teste2*, *expressão_teste3*, *expressão_teste4*}
  comandos2
otherwise
  comandos3
end

Se os testes de igualdade resultarem em falso para todos os casos, os comandos de *otherwise* (que é opcional) serão executados. O exemplo abaixo demonstra uma aplicação da estrutura *switch-case*.

```
clear all
num=input('Escreva um inteiro: '); % Exibe texto e lê valor para a variável.
switch num
    case 1
        y=1
    case {2, 2.5}
        y=2
    case 'tres'
        y=3
    case {'quatro', 'Quatro', 'QUATRO' }
        y=4
    otherwise
        y=0
end
```

10.2.2 Estruturas de repetição

Uma estrutura de repetição faz com que uma sequência de comandos seja executada repetidamente enquanto se satisfaça uma dada condição. O MATLAB® possui duas estruturas de repetição: *for* e *while*.

Estrutura *for*	Estrutura *while*
for *variável=inic:incr:fim* *comandos* end	while *condição* *comandos* end

A condição da estrutura *for* é que *variável* seja menor ou igual a *fim*. Enquanto satisfeita essa condição, ao término de cada *loop* de comandos, *variável* é incrementada de *incr*. O incremento *incr* é considerado unitário caso seja omitido.

Abaixo, segue um exemplo de cálculo de média, usando-se, de forma comparativa, as duas estruturas.

```
clear all
x=input('Vetor de amostras: ');
compr=length(x); % retorna o comprimento do vetor
soma_f=0;
for k=1:compr
    soma_f=soma_f+x(k);
end
media_f=soma_f./compr;

soma_w=0;
k=1;
while k<=compr
    soma_w=soma_w+x(k);
    k=k+1;
end
media_w=soma_w./compr;

disp('Media do for: '); % Exibe string
disp(media_f); % Exibe valor da variável
disp('Media do while: ');
disp(media_w);
```

Apêndice A

Funções relativas a aproximações para filtros seletores em frequência

Este capítulo contém os principais comandos do MATLAB®, para o projeto de filtros analógicos e digitais. Existem várias técnicas de aproximação para a construção de filtros. Cada técnica é usada de acordo com as especificações do sistema a ser projetado.

A seguir, são apresentados os comandos com as opções utilizadas no projeto de filtros analógicos. Para os filtros digitais, consulte o manual, através do comando `help` *comando*.

A.1 Butterworth

BUTTORD calcula a ordem de um filtro *Butterworth*.

`[N, Wn] = buttord(Wp, Ws, Rp, Rs,'s')` – Este comando retorna a ordem do filtro *Butterworth* que tenha, no máximo, Rp dB de atenuação na banda de passagem e, no mínimo, Rs dB na banda de rejeição. Wp and Ws são as frequências da banda de passagem e rejeição, respectivamente. W deve ser dado em rad/sec.

Exemplo:

```
passa-baixa:          Wp = x         Ws = y
passa-alta:           Wp = y         Ws = x
banda de passagem:    Wp = [x1 x2]   Ws = [y1 y2]
banda de rejeição:    Wp = [y1 y2]   Ws = [x1 x2]
```

A frequência retornada W_n é a frequência onde ocorre uma atenuação de 3 dB.

BUTTER calcula os coeficientes de um filtro *Butterworth*.

`[num,dem] = butter(N,Wn,'s')` – Este comando retorna os coeficientes do numerador (num) e denominador (dem) da função de transferência do filtro *Butterworth* passa-baixa. N é a ordem do filtro e W_n a frequência de corte.

Se W_n for um vetor contendo duas frequências $Wn = [W_1\ W_2]$ este comando retorna a função de transferência do filtro passa-banda *Butterworth* de ordem 2N tendo Banda de passagem com $W1 < W < W2$.

`[num,dem] = butter(N,Wn,'high','s')` – Calcula um filtro passa-alta.
`[num,dem] = butter(N,Wn,'stop','s')` – Calcula um filtro rejeita-banda, se $Wn = [W1\, W2]$.

`[Z,P,K] = butter(...)` – Retorna zeros, polos e ganho escalar K.

A.2 Chebyshev

CHEB1ORD calcula a ordem de um filtro *Chebyshev* Tipo I .

`[N, Wn] = cheb1ord(Wp, Ws, Rp, Rs,'s')` – Este comando retorna a ordem do filtro *Chebyshev* que tenha, no máximo, Rp dB de atenuação na banda de passagem e, no mínimo, Rs dB na banda de rejeição. Wp and Ws são as frequências da banda de passagem e rejeição, respectivamente. W deve ser dado em rad/sec.

Exemplo:

```
passa-baixa:          Wp = x         Ws = y
passa-alta:           Wp = y         Ws = x
banda de passagem:    Wp = [x1 x2]   Ws = [y1 y2]
banda de rejeição:    Wp = [y1 y2]   Ws = [x1 x2]
```

A frequência retornada W_n é a frequência onde ocorre uma atenuação de 3 dB.

CHEBY1 calcula os coeficientes de um filtro *Chebyshev* Tipo I.

`[num,dem] = cheby1(N,R,'s')` – Este comando retorna os coeficientes do numerador (num) e denominador (dem) da função de transferência do filtro *chebyshev* passa-baixa. N é a ordem do filtro , R a atenuação máxima na banda de passagem e W_n a frequência de corte.

Se W_n for um vetor contendo duas frequências $Wn = [W_1\ W_2]$ este comando retorna a função de transferência do filtro passa-banda *Chebyshev* de ordem 2N tendo banda de passagem com $W1 < W < W2$.
`[num,dem] = cheby1(N,R,Wn,'high','s')` – Calcula um filtro passa-alta.
`[num,dem] = cheby1(N,R,Wn,'stop','s')` – Calcula um filtro rejeita-banda, se $Wn = [W1\, W2]$.

`[Z,P,K] = cheby1(...)` – Retorna zeros, polos e ganho escalar K.

CHEB2ORD calcula a ordem de um filtro *Chebyshev* Tipo II .

`[N, Wn] = cheb2ord(Wp, Ws, Rp, Rs,'s')` – Este comando retorna a ordem do filtro *Chebyshev* que tenha, no máximo, Rp dB de atenuação na banda de passagem e, no mínimo, Rs dB na banda de rejeição. Wp and Ws são as frequências da banda de passagem e rejeição respectivamente. W deve ser dado em rad/sec.

Exemplo:

```
passa-baixa:          Wp = x         Ws = y
```

```
passa-alta:          Wp = y        Ws = x
banda de passagem:   Wp = [x1 x2]  Ws = [y1 y2]
banda de rejeição:   Wp = [y1 y2]  Ws = [x1 x2]
```

A frequência retornada W_n é a frequência onde ocorre uma atenuação de 3 dB.

CHEBY2 calcula os coeficientes de um filtro *Chebyshev* Tipo II.

`[num,dem] = cheby2(N,R,'s')` – Este comando retorna os coeficientes do numerador (*num*) e denominador (*dem*) da função de transferência do filtro *Chebyshev* passa-baixa. N é a ordem do filtro , R a atenuação mínima na banda de rejeição e W_n a freqüência de corte da banda de rejeição.

Se W_n for um vetor contendo duas frequências $Wn = [W_1\ W_2]$ este comando retorna a função de transferência do filtro passa-banda *Chebyshev* de ordem 2N tendo banda de passagem com $W1 < W < W2$.
`[num,dem] = cheby2(N,R,Wn,'high','s')` – calcula um filtro passa-alta.
`[num,dem] = cheby2(N,R,Wn,'stop','s')` – calcula um filtro rejeita-banda, se $Wn = [W1\,W2]$.

`[Z,P,K] = cheby2(...)` – Retorna zeros, polos e ganho escalar K.

A.3 Elíptico (Cauer)

ELLIPORD calcula a ordem de um filtro *Elíptico* .

`[N, Wn] = buttord(Wp, Ws, Rp, Rs,'s')` – Este comando retorna a ordem do filtro *Elliptic* que tenha, no máximo, *Rp* dB de atenuação na banda de passagem e, no mínimo, *Rs* dB na banda de rejeição. Wp and Ws são as frequências da banda de passagem e rejeição respectivamente. W deve ser dado em rad/sec.

Exemplo:

```
passa-baixa:         Wp = x        Ws = y
passa-alta:          Wp = y        Ws = x
banda de passagem:   Wp = [x1 x2]  Ws = [y1 y2]
banda de rejeição:   Wp = [y1 y2]  Ws = [x1 x2]
```

A frequência retornada W_n é a frequência onde ocorre uma atenuação de 3 dB.

ELLIP calcula os coeficientes de um filtro *Elíptico*.

`[num,dem] = ellip(N,Rp,Rs,Wn,'s')` – Este comando retorna os coeficientes do numerador (*num*) e denominador (*dem*) da função de transferência do filtro *Elliptic* passa-baixa. N é a ordem do filtro, W_n a frequência de corte, *Rp* dB de atenuação na banda de passagem e *Rs* dB na banda de rejeição.

Se W_n for um vetor contendo duas frequências $Wn = [W_1\ W_2]$ este comando retorna a função de transferência do filtro passa-banda *Elliptic* de ordem 2N tendo Banda de passagem com $W1 < W < W2$.

`[num,dem] = ellip(N,Rp,Rs,Wn,'high','s')` – Calcula um filtro passa-alta.
`[num,dem] = ellip(N,Rp,Rs,Wn,'stop','s')` – Calcula um filtro rejeita-banda, se $Wn = [W1\, W2]$.

`[Z,P,K] = ellip(...)` – Retorna zeros, polos e ganho escalar K.